Game Challenge: A Factorial Analysis Approach

(*Thesis Format*: Monograph)

by

Ian James Fraser

Graduate Program in Computer Science

Submitted in partial fulfillment
of the requirements for the degree of

Master of Computer Science

School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

June 5, 2012

© Ian James Fraser 2011

THE UNIVERSITY OF WESTERN ONTARIO

School of Graduate and PostDoctoral Studies

CERTIFICATE OF EXAMINATION

Supervisor	Examiners
Dr. Michael Katchabaw	Dr. Michael Bauer
Co-Supervisor	Dr. Jacquelyn Burkell
Dr. Robert Mercer	Dr. Kamran Sedig

The thesis by

Ian James Fraser

entitled

Game Challenge: A Factorial Analysis Approach

is accepted in partial fulfillment of the

requirements for the degree of

Master of Computer Science

June 5, 2012	Dr. Sylvia Osborn
Date	Chair of Thesis Examination Board

Abstract

Video games that customize to a player's experience level and abilities have the potential to allow a broader range of players to become engaged and maintain interest as they progress in experience level. A game that uniquely customizes the player's experience could attract additional demographics to gaming, which will result in a distinct edge in marketability and potential revenue. This thesis examines a subsection of adaptive gaming systems from the perspective of identifying game factors that alter the level of difficulty. Our focus is to provide a solution useful to both research and commercial gaming communities by developing a system that simulates results offline yet can be integrated into online play. While online performance is the main goal of an adaptive system, the offline simulation provides several benefits. Offline simulation allows the elimination of insignificant factors from inclusion in the training and evolution phase of machine learning algorithms. In addition it provides commercial games with a useful tool or method for performing game balancing and level tuning. To test our approach we designed a test-bed version of the game Pac-Man. The experimental testbed alters environment variables to evaluate their effect on a set of selected response variables. Observing the results of several response variables provides the potential to represent multiple player states, though our focus is on controlling the difficulty for a player. The testbed will simulate the actions of both Pac-Man and the ghosts over a variety of different settings and strategies. The evaluation of a factor's significance and its effect size are calculated using a factorial analysis approach. This method allows the identification of factors relevant to both individual strategies, and the set of all player strategies. Finally, as a proof of concept for both the online and adaptation prospects of this method, we developed a prototype adaptive system. Utilizing the relevant factor effects calculated in the factorial analysis, the prototype adapts to control the progress of the game towards targeted response variable intervals.

Keywords: Dynamic Difficulty, Auto-Dynamic Difficulty, Game Balancing, Level of Challenge, Level of Difficulty, Game Metrics, Adaptive Game System, Gameflow

Acknowledgements

Foremost, I would like to express my gratitude to my supervisors, Dr. Michael Katchabaw and Dr. Bob Mercer, for their patience, guidance and wisdom throughout my studies. From the beginning of my studies, their knowledge, expertise and foresight has guided my research into exciting and innovative areas that will contribute to the future of gaming. I am thankful for the assistance, feedback, critical analysis and the numerous other contributions my supervisors provided me. Most of all, I'm thankful to my supervisors for helping me find the perfect area of study. Performing research in an area that naturally blends my interest and abilites has helped me stay motivated and truly enjoy this experience. This experience has helped me develop into a stronger individual and inspired me to continue as a researcher.

I would like to express a special thanks to Dr. Charlie Goldsmith, without whom this research would not have been possible. Dr. Goldsmith helped our research progress past the commercial software size limitations and aided in validating and reinforcing the strength of our statistical analysis. Dr. Goldsmith's contributions and statistical expertise were invaluable and his knowledge, creativity, instruction and dedication are inspiring.

Finally, I am thankful to my family for their limitless support.

Contents

Certificate of Examination ii

Abstract iv

Acknowledgements v

1 Introduction 1

1.1 Overview 1

1.2 Background 3

1.3 Thesis Focus 6

1.4 Proposed Solution 8

1.5 Outline 11

2 Background 12

2.1 Player Enjoyment 12

2.1.1 Immersion 12

2.1.2 Flow 13

2.1.3 Telepresence 15

2.1.4 Feedback 15

2.1.5 GameFlow 17

2.1.6 Problems in Game Play 19

2.1.7 Challenge and Difficulty 21

2.2 Player Types 23

2.2.1 Casual to Core Players 24

2.2.2 Myers-Briggs Typology 24

2.2.3 Bartle's Player Types 27
2.2.4 Emotional Motivation 29
2.2.5 Temperament Theory 30
2.2.6 Demographics Game Design 1 (DGD1) 31
2.2.7 Summary of Player Types 32
2.3 Adversarial Algorithms 34
2.3.1 Flocking Algorithm 34
2.3.2 Minimax Algorithm 36
2.3.3 SSS-AB* 38
2.4 Machine Learning 40
2.4.1 Artificial Neural Networks (ANN) 41
2.4.2 Genetic Algorithms 43
2.4.3 Neuroevolution 44
2.4.4 Reinforcement Learning 44

3 Related Work **46**

3.1 User-System Experience (USE) Model 46
3.2 Adaptive Game System (AGS) 48
3.2.1 Auto-Dynamic Difficulty (ADD) 50
3.3 Measuring Player State 51
3.3.1 Player Skill Level 51
3.3.2 Emotional State (Affective Gaming) 53
3.4 Game Adjustments 54
3.4.1 Types of Adjustments 54
3.4.2 Player Characteristics 54
3.4.3 Game and Level Design 55
3.4.3.1 Inventory Control 56
3.4.4 Non-Playable Characters (NPC) 57

3.4.4.1 Reliable Adaptive Game Intelligence (RAGI) Requirements . . . 58
3.4.4.2 Genetic Algorithm Approaches . . . 59
3.4.4.3 Neural Network Approaches . . . 60
3.4.4.4 Neuroevolution Approaches . . . 61
3.4.4.5 Reinforcement Learning Approaches . . . 63
3.4.4.6 Dynamic Scripting . . . 64

4 Experimental Testbed Design 68

4.1 Adaptive Gaming Architecture Design Overview . . . 68
4.2 Overview of Testbed Game Design . . . 70
4.2.1 Original Pac-Man Game Description . . . 70
4.2.2 Research Testbed . . . 71
4.2.3 Object Loading . . . 73
4.3 Game Factors . . . 74
4.3.1 Agent Factors . . . 74
4.3.2 Bonus Factors . . . 76
4.3.3 Algorithm Factors . . . 78
4.4 Player Algorithms . . . 79
4.4.1 SSS-AB* . . . 79
4.4.2 Pac-Man Weighted (PW) Heuristics . . . 82
4.5 Ghost Algorithms . . . 84
4.5.1 The Flocking Algorithm . . . 85
4.5.2 Ghost Weighted (GW) Heuristics . . . 87
4.6 Performance Measures . . . 88
4.6.1 Proactive Measures . . . 89
4.6.2 Reactive Measures . . . 92
4.7 The Adaptive System . . . 94

4.7.1 Overview . . . 94
4.7.2 Adaptive Pac-Man . . . 94
4.7.3 Heuristics . . . 96
4.7.4 Limitations . . . 97

5 Experiments **99**

5.1 Factorial Design . . . 99
5.1.1 Calculating Variation . . . 102
5.1.2 $2^k r$ Factorial Design with Replications . . . 102
5.1.3 Calculating Lack of Fit . . . 103
5.1.4 Factorial Analysis and Experiment Design . . . 104
5.2 Experimental Environment . . . 107
5.2.1 Response Variables . . . 107
5.2.2 Factorial Analysis Customizations . . . 108
5.2.3 Model Evaluation . . . 110
5.2.4 Statistical Significance . . . 110
5.2.5 Game Significance . . . 111
5.2.6 Comparative Evaluation . . . 111
5.2.7 Adaptive System . . . 112
5.3 Results . . . 112
5.3.1 Model Evaluation . . . 114
5.3.1.1 SSS_FLOCK Model . . . 114
5.3.1.2 SSS_GW Model . . . 115
5.3.1.3 PW_FLOCK Model . . . 116
5.3.1.4 PW_GW Model . . . 117
5.3.2 Statistical Significance . . . 118
5.3.2.1 SSS_FLOCK Terms . . . 119
5.3.2.1.1 SSS_FLOCK Close Call Terms . . . 120

5.3.2.1.2 SSS_FLOCK Fruits Collected Terms 122
5.3.2.1.3 SSS_FLOCK Ghosts Eaten Terms 124
5.3.2.1.4 SSS_FLOCK Levels Completed Terms . . . 126
5.3.2.1.5 SSS_FLOCK Power-Pellets Collected Terms 128
5.3.2.1.6 SSS_FLOCK Repeated Squares Terms . . . 130
5.3.2.1.7 SSS_FLOCK Score Terms 132
5.3.2.1.8 SSS_FLOCK Steps Terms 134
5.3.2.1.9 SSS_FLOCK Tokens Collected Terms . . . 136
5.3.2.1.10 SSS_FLOCK Summary 138
5.3.2.2 SSS_GW Terms 138
5.3.2.2.1 SSS_GW Close Calls Terms 139
5.3.2.2.2 SSS_GW Fruit Collected Terms 142
5.3.2.2.3 SSS_GW Ghosts Eaten Terms 144
5.3.2.2.4 SSS_GW Levels Completed Terms 146
5.3.2.2.5 SSS_GW Power-Pellets Collected Terms . . 148
5.3.2.2.6 SSS_GW Repeated Squares Terms 150
5.3.2.2.7 SSS_GW Score Terms 152
5.3.2.2.8 SSS_GW Steps Terms 154
5.3.2.2.9 SSS_GW Tokens Collected Terms 156
5.3.2.2.10 SSS_GW Summary 158
5.3.2.3 PW_FLOCK Terms 159
5.3.2.3.1 PW_FLOCK Close Calls Terms 159
5.3.2.3.2 PW_FLOCK Fruits Collected Terms 160
5.3.2.3.3 PW_FLOCK Ghosts Eaten Terms 161
5.3.2.3.4 PW_FLOCK Levels Completed Terms . . . 162
5.3.2.3.5 PW_FLOCK Power-Pellets Collected Terms 163
5.3.2.3.6 PW_FLOCK Repeated Squares Terms . . . 164

5.3.2.3.7 PW_FLOCK Score Terms 166
5.3.2.3.8 PW_FLOCK Steps Terms 167
5.3.2.3.9 PW_FLOCK Tokens Collected Terms . . . 167
5.3.2.3.10 PW_FLOCK Summary 167
5.3.2.4 PW_GW Terms 168
5.3.2.4.1 PW_GW Close Calls Terms 170
5.3.2.4.2 PW_GW Fruits Collected Terms 171
5.3.2.4.3 PW_GW Ghosts Eaten Terms 171
5.3.2.4.4 PW_GW Levels Completed Terms 173
5.3.2.4.5 PW_GW Power-Pellets Collected Terms . . 173
5.3.2.4.6 PW_GW Repeated Squares Terms 174
5.3.2.4.7 PW_GW Score Terms 176
5.3.2.4.8 PW_GW Steps Terms 177
5.3.2.4.9 PW_GW Tokens Collected Terms 178
5.3.2.4.10 PW_GW Summary 179
5.3.3 Game Significance 181
5.3.3.1 Game Significance of Response Close Calls 183
5.3.3.2 Game Significance of Response Fruit Collected 184
5.3.3.3 Game Significance of Response Ghosts Eaten 185
5.3.3.4 Game Significance of Response Levels Completed . . 186
5.3.3.5 Game Significance of Response Power-Pellets 187
5.3.3.6 Game Significance of Response Repeated Squares . . 188
5.3.3.7 Game Significance for Response Score 189
5.3.3.8 Game Significance of Response Steps 189
5.3.3.9 Game Significance of Response Tokens 190
5.3.4 Separated Terms Comparison 191
5.3.4.1 Comparison of SSS_FLOCK Separated Terms . . . 191

5.3.4.2 Comparison of SSS_GW Separated Terms 193
5.3.4.3 Comparison of PW_FLOCK Separated Terms . . . 194
5.3.4.4 Comparison of PW_GW Separated Terms 201
5.3.5 Comparison of Algorithm Performance 211
5.3.5.1 Comparison of SSS-AB* and Ghost Algorithms . . . 211
5.3.5.2 Comparison of Flocking and Player Algorithms . . . 213
5.3.5.3 Comparison of Pac-Man Weighted (PW) and Ghost Algorithms 218
5.3.5.4 Comparisons to Ghost Weighted (GW) and Player Algorithms 225
5.4 Experimental Environment Summary 227
5.4.1 Model Evaluation Summary 227
5.4.2 Factorial Analysis Summary 228

6 The Adaptive System 231

6.1 Term Loading for Adaptive Pac-Man 231
6.2 Response Variable Selection 233
6.3 Heuristics 236
6.4 Results 239
6.5 Adaptive System Summary 240

7 Conclusion and Future Work 243

7.1 Contributions 243
7.2 Future Work 245
7.2.1 Factorial Analysis 245
7.2.2 Online User Observation 246
7.2.3 Adaptive Game System 248
7.2.4 Commercial Implementation 249

7.3 Conclusions 250

A Factor Effects **253**

A.1 SSS_FLOCK Factor Effects 253

A.1.1 SSS_FLOCK Close Calls Model Terms 254

A.1.2 SSS_FLOCK Fruits Eaten Model Terms 255

A.1.3 SSS_FLOCK Ghosts Eaten Model Terms 256

A.1.4 SSS_FLOCK Levels Completed Model Terms 257

A.1.5 SSS_FLOCK Power-Pellets Collected Model Terms 258

A.1.6 SSS_FLOCK Repeated Squares Model Terms 259

A.1.7 SSS_FLOCK Score Model Terms 260

A.1.8 SSS_FLOCK Steps Model Terms 261

A.1.9 SSS_FLOCK Tokens Collected Model Terms 262

A.2 SSS_GW Factor Effects 263

A.2.1 SSS_GW Close Calls Model Terms 264

A.2.2 SSS_GW Fruits Eaten Model Terms 265

A.2.3 SSS_GW Ghosts Eaten Model Terms 266

A.2.4 SSS_GW Levels Completed Model Terms 268

A.2.5 SSS_GW Power-Pellets Collected Model Terms 269

A.2.6 SSS_GW Repeated Squares Model Terms 270

A.2.7 SSS_GW Score Model Terms 271

A.2.8 SSS_GW Steps Model Terms 272

A.2.9 SSS_GW Tokens Collected Model Terms 273

A.3 PW_FLOCK Factor Effects 274

A.4 PW_GW Factor Effects 274

B Model Error **275**

B.1 SSS_FLOCK Model Error 275

B.2 SSS_GW Model Error 277
B.3 PW_FLOCK Model Error 282
B.4 PW_GW Model Error 282

C Proof of Concept Statistics **284**
C.1 SSS_FLOCK Mean Performance Per Life 284
C.2 SSS_GW Mean Performance Per Life 285
C.3 PW_FLOCK Mean Performance Per Life 285
C.4 PW_GW Mean Performance Per Life 286

D Regression Equations (Prototype) **287**
D.1 PW_FLOCK Regression Equation 287
D.2 PW_GW Regression Equation 287

E Response Variables Descriptive Statistics **289**
E.1 SSS_FLOCK Descriptive Statistics 289
E.2 SSS_GW Descriptive Statistics 290
E.3 PW_FLOCK Descriptive Statistics 290
E.4 PW_GW Descriptive Statistics 291

F Effects Descriptive Statistics **292**
F.1 Close Calls Factor Effects Descriptive Statistics 292
F.2 Fruit Eaten Factor Effects Descriptive Statistics 292
F.3 Ghosts Eaten Factor Effects Descriptive Statistics 293
F.4 Levels Completed Factor Effects Descriptive Statistics 293
F.5 Power-Pellets Collected Factor Effects Descriptive Statistics 293
F.6 Repeated Squares Factor Effects Descriptive Statistics 294
F.7 Score Factor Effects Descriptive Statistics 294
F.8 Steps Factor Effects Descriptive Statistics 295

F.9 Tokens Collected Factor Effects Descriptive Statistics 295

Bibliography **296**

Vita **303**

Chapter 1

Introduction

1.1 Overview

Over the last several years, the video game industry has steadily increased its financial contributions to the media sector and continues to be one of the largest areas for potential growth [58]. One of the keys to the success of the game industry has been the ability to find new demographics of game players outside the normal user groups [11]. A large portion of the recent success can be attributed to a developmental shift in the way players interact with their gaming system. Systems such as Nintendo Wii, PlayStation Move and Microsoft Kinect utilize a non-standard control system, which endorses a more intuitive set of motions to partially replace the use of button presses to perform actions. Additionally, as a large number of games developed for these systems are focused on simulating outdoor activities or being active these controller systems have helped spark interest among new active users groups. As an example of this success, Wii Sports for Nintendo Wii, which allows players to simulate playing tennis, baseball and other sports recently became the all time best selling game [40].

As the demographics of game players expands, so too will the range of players abilities and needs. Players will have varying skill levels in terms of characteristics such as: reaction times, hand eye coordination, and tolerance for failure. The wider range in players' abilities will increase the difficulty for game designers to sort players into the usual static and preset difficulty settings of easy, medium and hard. When the game's difficulty is not correctly matched to the player's ability it may cause the player to become bored or frustrated with the game, and may result in reduced play time or complete abandonment of the game. The difficulty setting of commercial games are generally the results of an enormous number of hours spent balancing

and tuning each level. Difficulty levels are determined to be appropriate based on testing by game testers and may not represent a diverse or introductory selection of players. The significant amount of time spent by game testers, level designers, and programmers can be quite costly for companies, and the end result may still not provide an adequate learning curve for new players. Allowing the player to manually switch between difficulty levels allows them to ease their own progression in the game. However, it can result in a number of problems from disrupting the immersion of the game by having to go switch the difficulty rating, to having the player trying to accurately judge their own level of play as well perform an accurate assessment of unseen tasks and opponents. An alternative method used in commercial games such as `God of War` is to ask the player if they wish to have the difficulty level lowered after a set number of deaths. This option does not have the ability to increase the difficulty and can frustrate players by suggesting they need to decrease the difficulty when they simply wish to try again.

An additional issue that arises with the development of games is developers over focus on creating games which appeal to themselves or to other well known user groups. This process is known as implicit audience modeling. Implicit audience modeling can result when developers believe they know what their audience wants out of the game instead of asking them. The traditional game development process focuses on developing games for these well known user demographics. This results in a large number of games being simple extensions of other successful games in the market. An important issue arises in attempting to develop for new player demographics: do developers actually know what these new players want out of their experience of playing the game? Since it is only recently that the gaming market has been able to lure these demographics into playing, we know very little about their playing styles and preferences. Developing games for these atypical gamer demographics starts with understanding what attracts their attention. However, if this information was readily available, developers would have already used this knowledge to attract these demographics. Thus, if we are interested in reaching and maintaining these and other atypical gamer demographics, we should be developing technology that identifies

and adapts to a players' needs. Adaptable game systems have been proposed as a possible solution to accommodate the growing variation of player's needs, preferences and abilities. The traditional game process rarely allows for adaptability within their games as a means of retaining or attracting a wider spectrum of users. The lack of adaptation is partially due to the additional cost, tight time-lines of projects and the believed additional required testing. However, as the process of customizing the game to the player matures, it should be capable of easing new players into the mechanics of gameplay, while providing challenging levels for experienced users. The wider the spectrum of players introduced to a game along with the positive adaptation of the player's experience should provide enough motivation to outweigh the initial costs of the system. Since it has been shown that games are often purchased on other players' recommendations [11], a game that has the ability to adapt to human players' strategies and abilities will provide a customized experience while keeping players engaged in game play. Ideally, adaptive games will attract additional demographics to their games, which will result in a distinct edge in marketability and potential revenue.

1.2 Background

The main goal of adaptive game systems is to customize the game to produce an immersive and compelling player experience. The goal of the adaptation process is to produce a desired experience within the player, the overall goal of an adaptive game is to produce an optimal immersive experience within the player, known as Flow. Flow, originally proposed by Csikszentmihalyi [20], is described as a state between boredom and frustration. "It is an experience so gratifying that people are willing to do it for its own sake with little concern for what they will get out of it, even if it is difficult or dangerous" [20]. This is the experience we want to induce in players of the game, where the motivation to continue playing is internalized and the player is highly engaged in the game. The work of Sweetser and Wyeth [47] mapped characteristics for producing Flow directly to video games to model player enjoyment,

aptly terming their extension GameFlow. In terms of game development the most significant components of GameFlow are: challenge, player skills, control, clear goals and feedback. These are the components game designers have the most control over, and thus provide the best opportunities in the game for customization to produce GameFlow.

The User-System Experience (USE) model further discussed in Section 3.1, is a model specifically for video games which emphasizes the relationship between a user and a system with the focus of producing Flow or other immersive states. This model forms the basis for our design to produce the desired level of interaction, but it lacks the required level of detail in terms of describing adaptive game systems. Adaptive game systems can adapt to two broad categories of problems: usability problems and playability problems. Section 2.1.6 briefly introduces the types of usability issues, although usability issues are important to adaptive game systems, they are beyond the scope of this thesis and our focus will remain on playability issues. The majority of commercial and academic research has focused on adaptation to playability issues from the perspective of altering the level of challenge of the game. The level of challenge is one of the main components in several psychological models that describe immersion. Adequately challenging the player in an engaging fashion can contribute to the player being within an immersive state during gameplay. In addition, alterations to the level of challenge are expected by players as part of the natural progression of the game and can be accomplished in a natural and seamless fashion. In order to be successful at producing an immersive experience in a player, the adaptive system must monitor the player's performance, make adjustments throughout the game and monitor the effect of those adjustments. The process of an adaptive game system altering the level of challenge is a portion of a concept known as auto-dynamic difficulty.

The concept of auto-dynamic difficulty deals with the process of allowing the challenge of the game to adjust to the abilities of the players. Ideally, this process will attract both regular gamers and non-gamers, by allowing non-gamers to become engaged by lowering the initial learning curve and altering the rate of progression

to avoid frustration that can cause newer users to give up on games. In addition, it may enhance the challenge for regular players, who are experienced with game genres, and have higher skill levels and expectations. Research in the area of auto-dynamic difficulty has generally focused on the challenge component of GameFlow and has approached the topic from two main perspectives: adaptation to the game environment or adaptive artificial intelligence (AI). An example method of game environment adaptation is an object quantity control approach [25], which alters the frequency of items or opponents the player will encounter in an area of the game. Using this method a game may alter the probability of finding extra health if the player's health is constantly low. In adaptive artificial intelligence(AI), the game agents alter the quality of their decisions based on the player's current level of difficulty. The term "agents" refers to characters in the game, which can include the player or other characters known as non-playable characters. Non-playable characters (NPC's) are characters in the game which the player has no control over, they can be opponents, allies or neutral to your character. In adaptive AI if the player is experiencing boredom, the game agents should choose moves that produce higher levels of challenge, thus increasing the difficulty of the game, likewise they should select less challenging moves when the player is at a level of difficulty beyond the player's ability [5].

An important discrepancy occurs in the game development community between commercial and research artificial intelligence. In commercial games the term AI usually describes the behavior and interactions of NPCs. Due to online requirements and lack of supervision for results, commercial AI is often hard-coded or scripted to perform the same actions with slight variations. Commercial AI rarely uses a training or learning phase, although it could be said that their training phase occurs from game testers and adaptation from their feedback. Throughout the remainder of this research, we use the term AI to represent the process by which an agent makes a decision, whether scripted or learned. Ultimately, commercial AI must begin to utilize techniques from the research area to produce truly adaptable game systems.

1.3 Thesis Focus

Our research will focus on adaptable game systems, in which user interaction with the game is described via the USE model. The USE model focuses on the interaction between user and game with the goal of optimizing the player's experience. However, the USE model is a general model of interaction which does not directly focus on adaptable games. To refine our research goals, we focus on a higher resolution model known as the adaptable game system model (AGS) discussed in Section 3.2. The AGS model easily integrates into the USE model to demonstrate how adaptation can occur with the goal of promoting an engaged and immersed player. The AGS model, proposed by Charles and Black [13], indicates that an adaptive game system needs to accomplish the following online tasks:

- Player modeling which attempts to discover and classify a player's type and needs.
- Online adaptation of the game environment in response to a player's needs.
- A monitoring system that evaluates the effectiveness of the adaptation.
- Dynamic player remodeling.

Our research has chosen to further investigate portions of two sections of the AGS model relating to the second and third tasks of Charles and Black's adaptive game system model [13]. We will investigate adaptation of the game environment in response to a player's needs, as well as monitor the system and evaluate the effectiveness of adaptation. Our research will investigate these two tasks from the perspective of performing auto-dynamic difficulty. The player's needs form a dynamic entity and can transition quickly in the same game session or over a longer period of time as they continue to the play the game. The study of transitional states of an individual is known as between-subject design. It is an important issue within adaptive gaming as it addresses the player's needs within a particular game session. It can also be used to predict the player's general behaviour as they progress over a longer term

[9]. However, our research will focus on within-subject design, which attempts to understand the player's current state. Our research will consider each experiment independent of transition those we will only be considering the player's current state. As previously mentioned, auto-dynamic difficulty is described as the task of altering the difficulty of the game online in order to match the player's skill level. A game with auto-dynamic difficulty will adjust the game to an easier setting if the player is continuously having difficulty completing a task, and increase the difficulty as the player masters different game skills [30]. Choosing to adapt the level of difficulty in our game focuses the player's needs solely in relation to requiring a higher or lower level of challenge. Thus our research can focus on factors which affect the level of challenge in either a positive or negative fashion. The majority of current research has focused on the level of challenge as the primary target for adaptive games. Nearly all current research defines positive or negative modification to the level of challenge in terms of a single response variable such as score or health. Our research will include a larger number of response variables from which the level of challenge could be defined, and positive or negative modification could be made. Throughout our research, the term "the level of challenge" refers to our selected set of response variables or a subset of which could be used to estimate the level of challenge provided by the game. Using multiple response variables allows adaptation to occur not sole based on a single observable metric such as score, but could adapt based on a combination of metrics. Having multiple response variables provides a more complete view of how the adaptive process is effecting the player's experience. In addition the use of multiple response variables allows a more diverse set of potential actions when adaptation is required and allows the adaptive process to balance and control other player preferences. Recent researchers have had relative success in using combinations of response variables from a session to predict frustration, challenge and fun [56]. Thus an aspiration of this project is that future work could progress towards combinations of response variables for producing improved emotional states of players.

Dynamically adapting the game requires specific knowledge about the player and the game. It requires creating a player model, which involves accurately predicting

how a player will react in specific situations, their tolerance for frustration, history of previous challenges, successes, and failures, as well as other preferences. Creating a player model not only requires the prediction long-term behaviour or strategies but also requires having a model of short-term behaviour which might simply be temporary adjustments based on strategy or temperament. However, having an adequate player model for all of this information still requires altering the difficulty of a game to challenge and accommodate these player preferences. As the effectiveness of adaptation can be measured without the use of accurate player models, in that we can adapt the game and then observe the reaction without knowledge of a player's preferences, we decided to focus on measuring adaptation, and leave the larger problem of online dynamic player modeling for future research. This thesis will examine the effectiveness of alterations to the non-playable characters (NPC's) and the game environment against a set of varied player strategies. To accomplish the larger goal of an adaptive game system, it is first essential to understand how alterations to variables of the game will affect the player's performance.

Throughout our research we will focus on several goals, the main focus will be to produce a method capable of contributing to current research techniques in adaptive games, as well being useful to commercial games for level balancing. The methodology used will identify game factors which provide significant impact to a set of response variables relating to the level of challenge. In order to successfully adapt to the player's level of challenge, we must understand which game factors impact the response variables and the direction and degree of their impact.

1.4 Proposed Solution

One intention of our research is to provide a methodology which is capable of being utilized by both the research and commercial game communities. To accomplish this goal we must consider that few commercial games are utilizing online machine learning techniques, although this trend is beginning to change, commercial games rarely include online learning for released games. Thus our selected approach to identify

game factors which affect the level of challenge is an offline approach, but an important role for our offline approach is that it should be capable of being integrated or used in combination with an online method with minimal effect on the online performance requirements. A critical issue with designing an adaptive gaming system is user-testing and collecting information about the effects of the game on each potential player. Our research will use a within-subject design which attempts to understand the player's performance under a variety of settings, but this is not accomplished with online users. Instead, we will attempt to simulate a wide range of different player strategies through static behaviours and attempt to identify factors which impact those static strategies. The ability to identify factors which impact the level of challenge is useful to current research in game communities as a preprocessing step, as game factors which are not important to the level of challenge can be omitted before training and online adaptation of NPC's strategies.

To experiment with our proposed research goals we require a testbed environment; we have chosen Pac-Man as our testbed game. Pac-Man is a well known game with a simple graphical interface, which is easy to learn and to become involved in, yet is a difficult game to master. Pac-Man is commonly used in adaptive game research due to the simplicity of its interface and game rules, while providing the ability to integrate complex strategies and team behaviors. Due to Pac-Man's wide use in the research community, adaptive game design results and methods could be compared to and reused in other research. Our Pac-Man testbed must be capable of adapting game factors in both an offline and online setting. To accomplish this, the testbed was altered to allow game factors and allowable levels for those factors to be loaded via files. In addition the testbed required a system to keep track of important events and information. We developed a logging system for the experiment to log statistical and event information for multiple game systems in an XML format, eventually to be used in calculating response variable information. Finally, we altered the testbed to simulate game play utilizing the selected game factors, agent movements and actions and logged the results.

Evaluation of our experiment will occur in three main stages; the first stage will

determine the effect size of all game factors and their interaction which will utilize a 2^k factorial analysis which is further described in Section 5.1. The factorial design allows us to observe factor interactions, as well as individually factor values. Observing the results of factor interactions allows us to discover any emergent effects that may unexpectedly occur. The first stage will also calculate a statistical model which indicates the amount of variance described in Pac-Man's performance.

The second stage after calculating the size of effects for factors of the experiment, utilizes the results in two methods: the first is an offline comparative method in which we evaluate factors' effects on the set of response variables. The performance of Pac-Man will be compared to other simulations of the experiment with the same algorithm but with different factor levels. Comparing the performance of the same algorithm with different factor levels allows us to examine the difficulty experienced by an individual algorithm. We hope to identify factors which increase or decrease the difficulty for certain algorithms but not necessarily all algorithms, as this will potentially show an ability to identify difficulty in different player types. The comparative method will also compare the performance of a factor across all algorithms to identify the global effect on the difficulty of the game. This will provide evidence of a factor that could provide alterations to the level of difficulty for all player types.

The final stage of our proposed solution is to build a proof of concept adaptive game system, which will utilize the information of the effects of factors and their interactions to dynamically adapt the progression of the game. While the comparative method identified trends and results which could be utilized to perform level balancing or model building, this section progresses toward the automatic adaptation similar to other current research. The prototype will demonstrate the practicality of an intermediate step between the current method of static difficulty settings and the potential of adaptive game systems. The prototype begins with two separate static difficulty settings, as the game progresses it adapts game factors values from one of difficulty settings to the other, thus allowing the game to smoothly transition between the two difficulty levels. In the prototype adaptation will occur to control for one or more response variables based on ranges selected prior to running the prototype.

We will specify desired target ranges for a response variable which could represent possible player preferences in future research. We will design a set of heuristics to estimate the player's progression during gameplay and the level of change required to meet the target range. Using the factor effects information from the experiment phase of our research, our system will select factors settings adequate to the level of requested change for the player.

1.5 Outline

This thesis is organized as follows: Chapter 2 introduces necessary background information to comprehend the current direction of research on auto-dynamic difficulty. The background section will introduce base psychological models for player enjoyment and player types. We will introduce methods of measuring player states, and modeling opponents utilizing static and learning algorithms. Chapter 3 reviews current research relating to adaptive game systems. Chapter 4 discusses in detail the testbed game designed for the proposed solution. Chapter 5 discusses the experimental environment and evaluation of the proposed solution. Using the results of the experimental phase of our solution, we build and review the results of the adaptive game system prototype in Chapter 6. Finally, Chapter 7 concluded our results from the experimental phase and prototype, and discusses the contributions of the proposed method, as well as highlights possible areas for future work.

Chapter 2

Background

This chapter focuses on introducing the necessary multidisciplinary concepts in order to study adaptive games. Section 2.1 introduces models explaining how players experience the game. Section 2.2 reviews reasons players play games and the classification of their gaming needs. Section 2.3 introduces algorithms for performing adversarial searches and group behavior. Finally, Section 2.4 introduces machine learning techniques utilized in current research.

2.1 Player Enjoyment

This section introduces the idea of immersion and the psychological model for optimal immersion, known as Flow. This section introduces key traits and terms from Flow and their application in game design. Finally, we summarize the research of this model applied to rating player enjoyment in video games.

2.1.1 Immersion

Bartle, co-creator of multi-user dungeons (MUD), describes immersion in terms of games as "Immersion is the quality of being your virtual self" [9]. The progression of becoming your virtual self, begins with two separate systems: the player and the avatar. The avatar is the player's character represented in the game world. Eventually, the player begins to identify with the avatar and recognizes himself as a character in the game. Finally, the game character becomes a persona which is the representation of his virtual self [9].

Ermi and Mayra [21] proposed that immersion was based on three fundamental

components; sensory, challenge-based and imaginative immersion. Their model, the SCI (sensory, challenge, imaginative) model, was formed based on questionnaires of young players, and stated that players could have different levels of participation in each of the three components. Participation and engagement in at least one of the three components was the requirement for producing a state of immersion for younger players. As an example, a game like Pac-Man could be immersive solely on challenge but possibly also imaginative.

Immersion is a difficult state to measure, as it must be implicitly studied while the player is in the state. Self-reports often cause the player to leave a state of immersion and direct focus away from the task an individual is immersed in [17]. This has led some researchers to attempt to purposely break the player's state of immersion in attempts to better understand it. Cheng and Cairns [17] provided an experiment in which once players were suspected of being immersed the realism of the game would be completely altered. Their experiment included changes to the environment graphics and behavior of game physics. Cheng and Cairns provided surprising results, in that large alterations to the game did not affect the player's experience. In fact, several of the participants did not recall the changes being made until prompted in post game reviews [17].

2.1.2 Flow

Flow, originally proposed by Csikszentmihalyi [20], is a state of immersion, where the individual is fully engaged, energized by and focused on a task. Flow is often described as a "pleasurable balance between anxiety and boredom"[20]. If a task is too simple, an individual will reach a state of boredom and lose interest in the task. Likewise, if the task is too difficult an individual will lose interest because theyfeel frustrated by an inability to complete the task. Thus, Flow occurs when an individual is engaged in a task, where their skill level is appropriate for the level of challenge provided by the task. Csikszentmihalyi originally divided Flow into several main elements:

1. A task to be completed.
2. The ability to concentrate on the task at hand.
3. Concentration is possible because of clear goals.
4. Task provides immediate feedback.
5. A sense of control over actions.
6. A deep but effortless involvement that removes awareness of the frustration of everyday life.
7. Concern for self disappears, but reemerges stronger afterwords.
8. Sense of time is altered.

Every element listed above does not need to be present to produce a state of Flow, a subset of several of these elements may be enough to induce Flow in an individual.

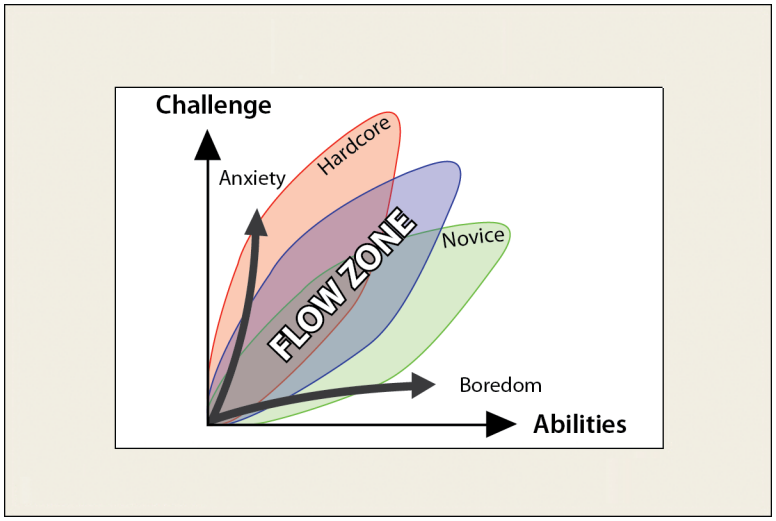


Figure 2.1: An illustration of the Flow Channel, which is the ideal balance for an individual between anxiety which occurs when the level challenge is above their ability and boredom which occurs when the level of challenge provided by the task is well below the ability of they individual. Adapted from[16].

2.1.3 Telepresence

Telepresence is an immersion model of experience like Flow, but is based on a more passive experience. It can be used in situations where challenge, and player skills are less applicable. Telepresence is described as an "experience of presence in an environment by means of a communication"[11], and individuals are usually responsive and sensually involved in the system. Telepresence is thought to account for immersion when an individual is not in a state of Flow [11]. An example of a situation in which people are likely to experience telepresence is when they are watching a movie.

2.1.4 Feedback

In terms of games, the player should constantly and consistently be receiving feedback whether it is from the player's avatar responding to game-pad input or feedback relating to the player's progression through the game. In Section 2.2 we will discuss how different player types require different amounts of feedback about their game progression. Detailed feedback can be difficult to provide when the player strays from the normal game path; unfortunately these are the players that likely require additional feedback. Another issue that arises from a player deviating from the normal game path is whether they are enjoying their exploration or whether they are simply unsure of what to do next. Players who are exploring will find feedback indicating they are going the wrong direction to be an unnecessary intrusion into their game playing.

Positive feedback is when the game becomes easier to complete as a result of the player completing some task or game objective. This can occur by players acquiring new weapons, or by a slight increase in the avatar's attributes in sports games while a team has momentum of the game. Using positive feedback helps the game progress and avoids states of stalemates in which neither player can win. The progression of a stalemate game shown in Figure 2.2, produces a back and forth match-up, in which neither player is capable of gaining a substantial lead or moving to a winning state. When the game progresses in a pattern similar to the stalemate situation it

may result in the player feeling as though their opponent will always catch up, which can result in the player externalizing match up results to luck or unfair play.

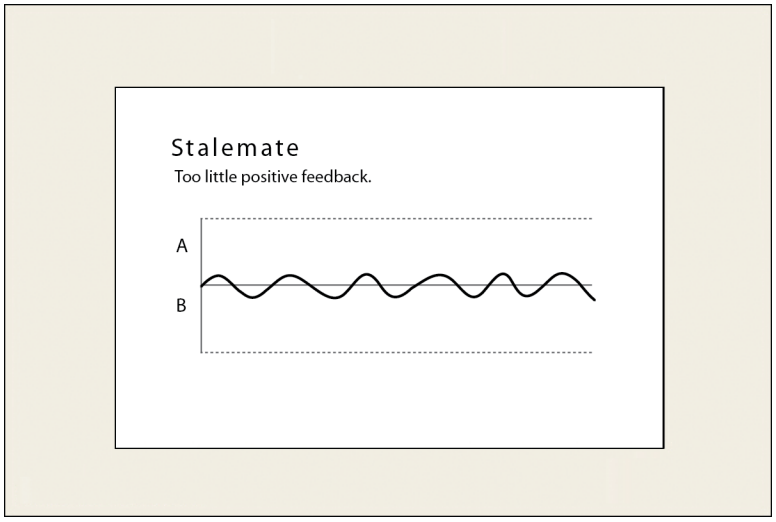


Figure 2.2: Represents a version of a stalemate game where players produces small leads but fail to progress to a winning state. Adapted from [1].

However, using too much positive feedback may result in an unbalanced game. Consider a situation in which a player is always given direction and the means to accomplish their goals; in this scenario the game can become too easy or limit opportunity to explore. Negative feedback is the opposite of positive feedback: a player's achievements result in the game difficulty increasing. Negative feedback naturally occurs as a player progresses through a game; the next level of a game is typically more difficult than the last. Negative feedback controls positive feedback and vice-versa. However, negative feedback can cause stalemates and large swings in the challenge of the game if performed improperly.

Figure 2.3 shows the ideal game progression that is balanced between the two players A and B with an appropriate level of feedback. Player A completes the first task and begins to progress toward winning the game, when negative feedback occurs and player B begins to catch up then surpass player A in the likelihood of winning. Once player B progresses toward a winning state negative feedback for player B ramps up, and slowly player A begins to catch up. This trend continues slowly progressing towards the advantage of player A. Each time player B falls a little further behind,

still able to almost draw even but taking the lead on fewer occasions [1]. This derives a back and forth type of game, which provides the illusion of winning to both players, where both players feel they are capable of winning if they just make a few minor adjustments. This helps players develop emotional attachment to the game, and promotes tweaking strategies to achieve victory.

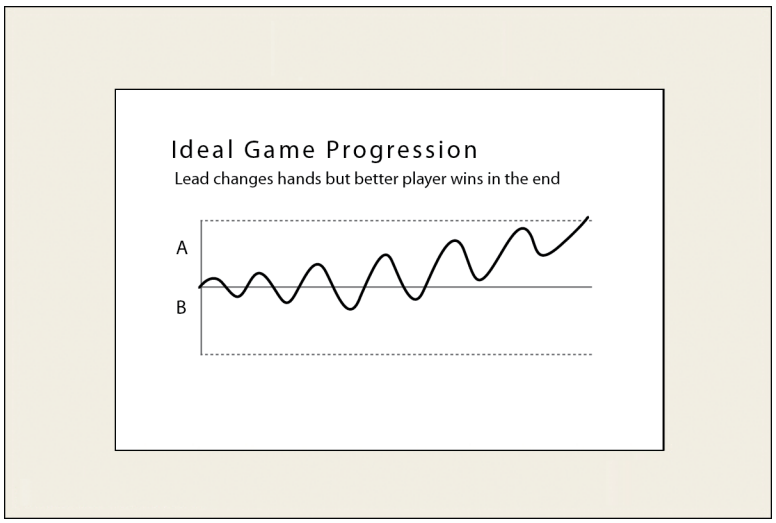


Figure 2.3: An ideal game progression where each player has the opportunity to win until eventually one player takes a controlling lead. Adapted from [1].

2.1.5 GameFlow

Flow was later directly applied to modeling player enjoyment in games by Sweetser and Wyeth. In their paper, Sweetser and Wyeth took the eight original components of Flow and defined each component in greater detail in terms of player requirements and game play [47].

1. The task to be completed is the game.

2. The game should immediately grab users' attention.

3. Perceived skills should match the task challenges, and both must exceed a threshold of boredom.

4. A player has control over their actions in the game, as well some sense of control over the direction of the game.

5. Goals throughout the game should be clear.

6. Immediate feedback should be given to the player.

7. Immersion in the game should be deep but effortless, and reduce the sense of self and time.

8. Social interactions should be supported through cooperative and competitive interactions among players.

Although game designers can promote methods to improve concentration, immersion and social interactions components, they ultimately have the least amount of control over these areas. While concentration and social interaction components are areas of GameFlow which are related to the player's habitual environment, immersion is part of the resulting experience. Games either include a social aspect such as multi-player modes or social interaction can occur via controller passing. Game designers have minimal opportunity to modify this aspect of the game post-production. The components challenge, player skill, clear goals, control and feedback are components of higher interest in developing a system with auto-dynamic difficulty as they can be modified in real time to help produce flow. The components of GameFlow can be categorized into three sections: game interface, game mechanics and game play. The game interface is highly correlated to the level of feedback, clear goals and control components. Game heuristics for the interface include: providing essential information in a clear method and allowing customization for controls. Game mechanics heuristics deal with clear goals and feedback, grabbing the user's attention immediately and reacting to theiractions. Game play heuristics deal with the level of challenge, clear goals, control and feedback. Players should always have both short and long term goals, and pressure should be applied to lead them towards these goals without frustrating them if they choose not to perform a particular goal. The level of game play should be balanced, and provide variable levels of difficulty [22]. The GameFlow

model has been applied to predicting player enjoyment in real time strategy (RTS) games. The results were similar in comparison to those of expert game reviews [47].

2.1.6 Problems in Game Play

In terms of GameFlow, designers have control over the level of challenge, control, producing clear goals and feedback. Designers will face two types of issues in controlling these components, the first being usability and the second being playability. We will briefly introduce some issues relating to usability here, which are included for completeness but are not discussed further. Usability problems are issues where the game is fully functional, yet lacks appropriate communication to the user of how to easily accomplish desired tasks [8]. A common usability problem that occurs in games is when a player is unsure of where to go next within the game world, often due to a lack of clear goals and feedback. According to Zap and colleagues [59], usability problems can be subdivided into four groups: knowledge base, intellectual, flexible and sensorimotor. In terms of games knowledge, problems occur when the player is given an inadequate amount of information to complete a task or progress in the game. Within the GameFlow model, knowledge usability problems occur in games due to a lack of clear goals or appropriate feedback of controls to achieve goals [48].

Intellectual usability problems can be further divided into three groups: thought, memory and judgment problems. Thought problems occur in games as a result of poor feedback; the player knows the goal they wish to accomplish, yet are struggling to achieve this goal as a result of uncertainty of their progress. As an example, if the player needs to complete an ordered set of tasks to complete a goal, if they don't receive an acknowledgment when they have completed a portion of the tasks, they may continue attempting the first task instead of progressing to the next task to accomplish the goal. Memory usability problems occur when the player has correctly identified the next goal and yet struggles to progress as a result of forgetting a portion of the process required to achieve this goal. Memory problems such as this can be caused by an inappropriate amount of feedback or by defining goals which take too long to complete. Finally, judgment usability problems occur when feedback is unclear

to the player and it is unclear whether the task has been successfully completed.

Flexible usability problems can be further divided into three groups: habit, omission and recognition problems. Habit problems occur when a player performs an action in the wrong context. An example is when a game does not utilize a standard genre control scheme and the player presses the standard scheme button instead of the game specific button for an action. Habit problems are caused by controls but can also be caused by lack of feedback indicating to the player that the game does not follow standard implementation. Omission usability problems occur when a player omits a routine portion of a scheme or task. Omission problems tend to occur when the player's skill level is above the level of challenge of a simple task or when feedback fails to draw player attention to a task. Finally, recognition problems occur when feedback is difficult to distinguish between other symbols or other goals. An example might be when the player is asked to collect an item, and he is shown a small blurry image which may be confused with a number of other items in the game

Lastly sensorimotor issues relate to the player being capable of successfully completing the required motor-skill. Sensorimotor issues can arise due to a lack of player skill, control issues or as a result of poor game interface design. However, in terms of usability sensorimotor issues usually relate to difficulty experienced by many individuals, and thus is less related to a player's level of skill.

Playability problems occur when portions of the game demotivate the player from progressing to the next task in the game. Playability issues can be segregated into four types of problems: control, fantasy, curiosity and challenge problems. Control problems relate not to controller issues but to who has control of the player's agent. During long introductory movie sequences or tutorials the player may not have full control of their agent, which causes the player to become impatient with their level of involvement. Fantasy problems relate to the rejection of a portion of the fantasy world that the game is attempting to entertain the user with. Fantasy problems can be catastrophic to a player's experience, as players are less likely to delve into a game in which they reject a portion of the basic premise. Fantasy problems are a major issue; however, the adaptability of a game can only extend so far to customize to the

player's experience, so players facing this issue should investigate alternative games. Curiosity playability problems occurs when a player has been involved with a task for an extended period of time and begins to feel bored due to lack of new tasks or actions. Essentially, it occurs when the player feels they have learned a task to their level of satisfaction, and yet are unable to progress in the game. Challenge problems occur when the level of challenge is either too high or too low for the player's level of skill.

The majority of usability problems that affect GameFlow are related to controls, clear goals, and adequate feedback. Having an intuitive and flexible game does not make the game fun. It helps the player get involved in the game and avoid unintentional frustration or demotivating the player. Ultimately, playability is the most important issue for game designers. In terms of playability, curiosity and challenge problems are of greater importance to the level of difficulty of the game. Although control problems can disrupt a player's sense of flow, it is good game design practice to allow the player to skip situations in which they do not have control such as tutorials or story sections. Fantasy problems are likely the result of a player not enjoying a genre of games or the story of a particular game. In either case this is an extremely difficult problem to solve. Challenge problems are the main focus of auto-dynamic-difficulty research to date. The key to challenge problems is to keep the player's level of difficulty in a range in which they are excited yet do not feel overwhelmed by the game. Finding a player's accepted range of difficulty means understanding the player's preferences and goals. To understand these, we will investigate player types. Understanding player types will also help us solve the curiosity problem. If we are capable of identifying the player's level of tolerance for repetition and frustration, we can identify when a player is likely becoming bored or frustrated with the given task and provide aid to allow for progression.

2.1.7 Challenge and Difficulty

The level of challenge of a game is directly related to the Flow experience of an individual. Challenge is also one of the three general components of the SCI model

for producing immersion, and is one of the major component of Flow in games [21]. The field of adaptive games defines the level of challenge as the amount of information an individual has to process, proportional to the rate of new information. Within this definition new information may simply refer to objects which are moving or are dynamic in the game. Challenge is often measured on two dimensions: the speed required to perform a task, and the level of cognitive involvement [21]. From the definition of challenge, we can observe that as a player gains experience with a game there is less new information to absorb, thus the level of challenge decreases.

The level of difficulty of the game refers to the level of the challenge relative to the player's level of skill. This definition highlights that a game may have a steady level of challenge between a set of players, and provide different levels of difficulty to each player. Thus in order to alter the level of difficulty we must perform adjustments to the level of challenge in relation to the player's skills.

Throughout the game a player will master new skills, acquire more powerful items and will optimize their strategies, thus in order to continually challenge the player the overall level of difficulty of the game must consistently increase in value. A natural progression in games is for the level of difficulty to remain constant while the player learns a new skill or progresses their player to a satisfactory level above the current level of challenge provided by the game. During this moment the player has the opportunity to experiment and understand their progression whether with a new weapon or strategy. Eventually the continuously successful methods lowers the level of challenge, and is not offering the opportunity for new information or learning. Before the level of challenge decreases such that boredom plagues the player, the level of difficulty must be raised so that learning and engagement can progress. The increase in the level of difficulty is an increase in negative feedback similar to the ideal game progression, shown in Figure 2.3. Figure 2.4 illustrates the natural progression of difficulty in the game in order to produce a learning and challenging environment [6].

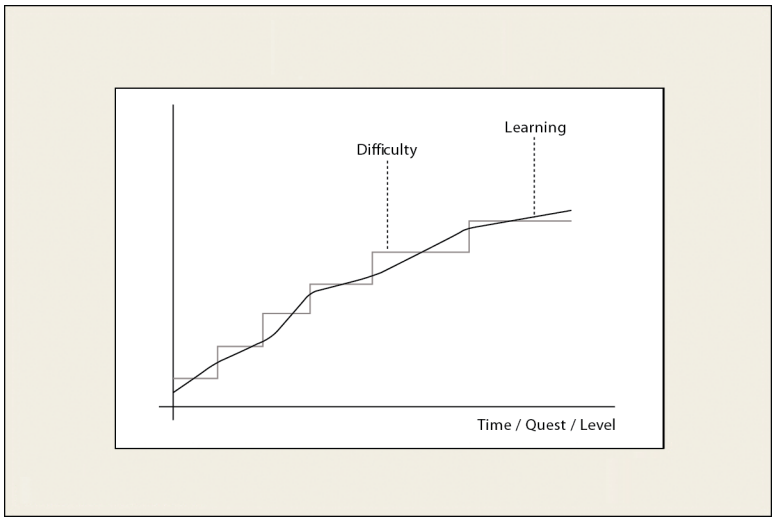


Figure 2.4: When a player begins a new game they must learn new skills to participate in the game. The challenge level remains steady as the player masters these skill, as a result the difficult of the game decreases. Before the player becomes bored from repetitive tasks, new obstacles must be introduced such that additional learning is required, which promotes interest. This process continues throughout the game and is known as the desired progression of difficulty. Adapted from [6].

2.2 Player Types

This section introduces classifiers for personality traits. The first classifier is the casual vs core spectrum which separates gamers based on their dedication levels. Next we introduce the Myers-Briggs typology as a more general classifier which provides additional information beyond the scope of the game and into the player's overall personality. Following the section on Myers-Briggs typology we introduce several ways of classifying game players based on their overall goals and motivations while playing the game. The first player classifier introduced in Section 2.2.3 is known as Bartle's player types and is based on Bartle's experience observing multi-level dungeon games. Section 2.2.4 reviews a classifier based on the emotional motivation of a player. Section 2.2.5 introduces a section on understanding patterns in behavior. Section 2.2.6 introduces research based on Myers-Briggs from a commercial company

to identify player types.

2.2.1 Casual to Core Players

The ability to dedicate a high level of concentration to a particular task plays an important role to the development Flow; with this in mind a key consideration for game designers is the amount of time a player is willing to invest in playing the game at any one period of time. Ip and Adams [26] proposed fifteen weighted factors that attempt to separate users into the casual-to-core player spectrum. These factors include: personal investment, tolerance to frustration, engagement with the game and other players, and goal oriented personalities. This creates a scale known as the casual to core gamer dedication. Ip and Adams [26] hypothesize the existence of five categories along this scale. These categories are presented in largest to smallest gamer population size respectively: ultra casual/non gamer, casual, transitional/moderate, hardcore and ultra-hardcore. Although research is still lacking in the probability of transitioning between categories, the scale does illustrate a largely untapped market of potential gamers in the middle of the casual and hardcore categories.

2.2.2 Myers-Briggs Typology

The Myers-Briggs typology is a psychological model for classifying individuals based on personality traits [11]. The model consists of four pairs of traits:

1. Introversion and Extroversion

   Extroverts tend to be motivated by social interaction, and tend to have an act first think later type attitude. Introverts tend to be motivated internally, and enjoy one-to-one communication.

2. Sensing and Intuition

   Sensors tend to use common sense, and prefer situations with clear information. Intuition individuals tend to be creative, and enjoy implied and theoretical information.

3. Thinking and Feeling

   Thinkers are task oriented, and enjoy logic, facts and are comfortable with conflict. Feelers focus on the consequences to other people, and are unsettled by conflict.

4. Judging and Perceiving

   Judgers are planners, who focus on one task at a time. Perceivers enjoy multi-tasking, freedom, variety and flexibility.

An individual's personality is composed of a percentage of each of the eight traits, however the dominant trait of each pair is used to fit them into one of sixteen groups. As an example, a person may be extroverted, sensing, judging and perceiving also known as ESJP. From the sixteen groups, it is hypothesized that the hardcore player type is characterized by introverted, thinking and judging an I_TJ trait set, and that sensing or intuition have less effect on the hardcore to casual player relation [11].

From the casual to core spectrum, the hardcore cluster is the smallest in population size, yet is thought to have the largest impact on the gaming industry. This is believed to be true because hardcore players are more likely to review games, play and purchase new games and thus propagate more information about potential games to the rest of the market [11, 26].

The Myers-Briggs typology provides information about the personality types of current gamers and their preferred game genres. It also illustrates personality types currently being neglected by or disinterested in the video game market. The current game market is heavily dominated by the introverted character trait. Therefore, a game interested in capturing the extroverted personality type may need to alter the structure of the game to better suit their needs. Typically games such as Dance Dance Revolution, that are turn based, after a short period of time are accepted among extroverts. One type of modification to the game that could promote higher levels of interest among extroverted players is providing additional feedback on current goals, if these players play infrequently they are likely to forget their previous position or tasks in the game [11].

The Myers-Briggs typology also illustrates how players prefer information displayed to them. The sense and intuition dichotomy illustrates how different people enjoy learning and solving problems. This is important for developing tutorials, puzzles, game challenges as well as setting the pace of the game. Players with a dominant sensing trait make up 70% of the population and are often more patient with repeated material, are less comfortable with abstract material and rely on their common sense to solve problems [11]. This indicates that sensing people are likely to prefer straight forward tutorials, as to comfort their ease into new situations. Likewise, the puzzles and challenges of the game should be directed towards knowledge and skills acquired during tutorials or game play. The intuition character trait, represents people who are more comfortable with abstract ideas and enjoy drawing their own conclusions. People with a dominant intuition trait would find the straight forward tutorial monotonous and restrictive to their overall enjoyment. Intuition individuals are more comfortable with unseen challenges and situations, they require less feedback than do sensing individuals [11].

The thinking versus feeling dichotomy provides insight into the motivations of the player as they play the game. Knowing the motivation of a player, allows the game to provide appropriate feedback to encourage the player progress. Altering the frequency of player feedback is not enough to ensure enjoyment; the feedback must be more specific to a user. Where thinking individuals respond well to clear goals, a feeling type may not respond well to a clear goal if they feel they are unable to accomplish it. Thus a feeling player may require additional progression information, and overall more feedback than their counter type. The difficulty of catering to the feeling type of gamer is that although they require more frequent feedback on their development in the game, they are also more sensitive to critical analysis. An example of the stark difference between how the two type interpret feedback is observed during the "Game Over" screen [23]. Feeling type players see this as an extra criticism with no positive feedback or options whereas the thinking type see this as an opportunity to reflect on their own play, only becoming frustrated once they are unable to develop alternative strategies to avoid the game-over screen. An additional benefit of knowing

the motivations of a player is that it allows the game to adjust rewards for completing objectives and goals. As an example, if the player is a feeling type, they could be rewarded with a feedback tool such as a beacon for a map, whereas a thinking player may be rewarded with a different weapon [11].

The judging and perceiving dichotomy provides insight into the motivation of the player from a goal-orientation perspective. The perceiving player generally plays to improve his own skills at a particular task, and thus are more comfortable within a less structured game model. On the other hand, judging players tend to prefer a straight forward path focusing on the task at hand, and are motivated by the overall completion of the game [11].

Finally, the combination of thinking-judging player is more focused and enjoys conflict and challenge. Its counterpart the feeling-perceiving player sees having easy fun as the key motivational factor, and is less tolerant to frustration. The thinking-judging player type is often associated with the hardcore player type from the Ip and Adams study [26].

2.2.3 Bartle's Player Types

Bartle's player types [9] help describe the general motivation of a player. It shows which types of actions the player currently finds "fun" within the context of the virtual world. A player's decisions are divided between their relation to other players or the world, and between actions or interactions. Thus, the original Bartle model consisted of four types of players: killers, achievers, explorers and socializers.

Although Bartle's model was widely accepted, it lacked an explanation for three major situations: distinct subgroups within a player type, transitions made between player types and it provided no explanation for immersion. Thus, Bartle proposed a slightly modified model that contained a new dimension implicit and explicit. Implicit actions are done without thought of consequence. Whereas explicit actions are performed in terms of an overall plan. The inclusion of a third dimension lead to solutions for all three states problems with the model.

As a result of Bartle's model changes, it now accounts for how players change

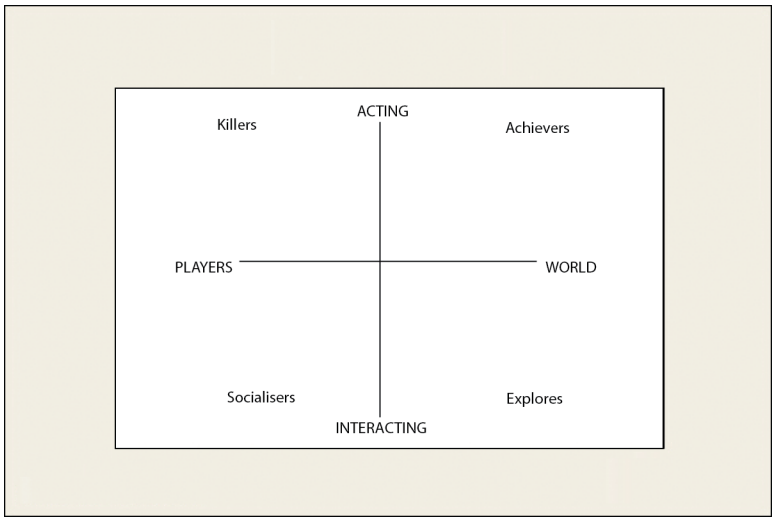


Figure 2.5: Bartle's original model describing player multiple player types. Adapted from [10].

type over time, new models for player progression were formed between types. A total of four sequences were produced that describe a player transition from being a new player to being a consistent player. The four sequences are:

1. Main Sequence = Griefer -> Scientist -> Planner -> Friend
2. Socializer Sequence = Griefer -> Politician -> Networker -> Friend
3. Explorer Sequence = Opportunist -> Scientist -> Planner -> Hacker
4. Minor Sequence = Opportunist -> Networker -> Planner -> Friend

The new dimension of Bartle's model adds a motivation to the original player types described in the previous generation of the model. The killer player type which enjoys defeating other players, is now a combination of politicians and griefers. Griefers are players who enjoy harming other players while politicians enjoy interacting and resolving issues. Explorers are players that enjoy exploring and understanding the rules of the environment. They are now split as scientists and hackers. Scientists are inquisitive players and hackers enjoy testing the bounds of discovered rules. Socializers are people who play to interact with other people. Socializers are sub-categorized into networkers and friends, to define the people whom they are attempting to interact

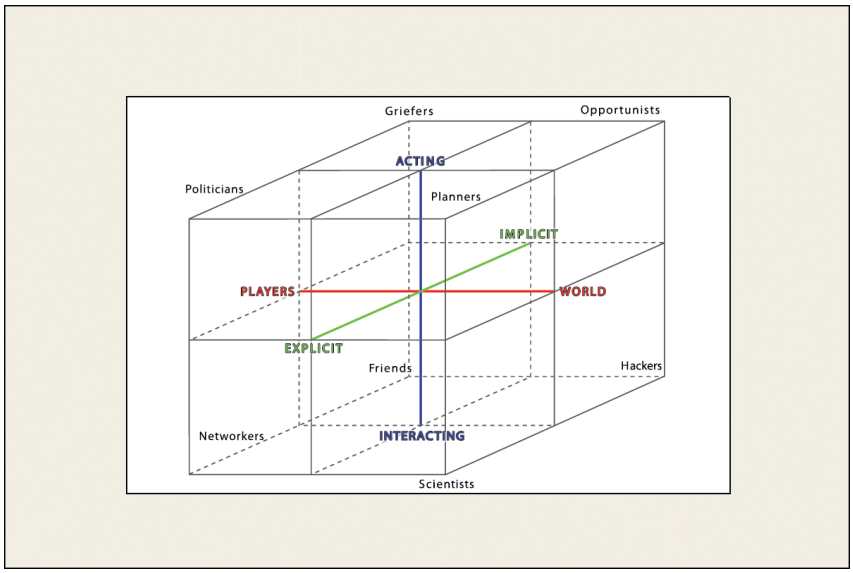


Figure 2.6: The adapted version of Bartle's player types model which includes a new dimension to explain player progression. Adapted from [10].

with. Finally, achievers are players who enjoy accomplishing goals provided within the game, they are divided into opportunists and planners. Planners enjoy being told the next goal to accomplish, whereas opportunists enjoy finding their own way to the goal and are more comfortable with improvisation needed for their plan [10].

2.2.4 Emotional Motivation

Lazzaro [29], demonstrated that players choose games not solely on the quality of the game, but based on an expected experience provided by the game. According to Lazzaro there are four main keys to promoting emotion: Hard Fun (emotions from challenge), Easy Fun (emotions from curiosity), Altered States (emotion through perception, and thought) and The People Factor (opportunities to socialize and compete). These four keys are closely related to Bartle's original model of player types. However, one of the key differences is that Lazzaro suggests that transitions between the four states may be the result of emotional needs or desire for a specific experience [29], whereas Bartle has explained transitions between player types in terms of planning actions. Emotional needs may implicitly be explained via the implicit and explicit dimension of Bartle's reconstructed model [9]. A key final note about Laz-

zaro's study is that it highlights the fact that player types can change dynamically and more frequently than is proposed in the original Bartle model; this illustrates a constant need to remodel the player model.

2.2.5 Temperament Theory

Temperament theory deals with understanding the patterns of behavior for an individual. It attempts to classify people not as a consistent personality type or as a pattern of perception, but as a summation of different dispositions to situations. Whereas Myers-Briggs looks at how an individual thinks, temperament theory investigates how they behave. For our purposes, temperament theory breaks down Myers-Briggs to four types of temperaments: rational, idealist, artisan and guardian, each with an associated skill set and tendencies towards problem solving [11].

The rational temperament comes from the Myers-Briggs intuition and thinking attributes. Rationalists utilize a strategic skill set, this involves planning and executing plans to meet defined goals [11].

The idealist temperament is based on the intuition and feeling attributes of Myers-Briggs. Idealists utilize diplomatic skill sets and are empathetic towards other players and try to resolve conflicts in the game. These players become immersed in character and story development [11].

The artisan temperament is based on the sensing and perceiving attributes of Myers-Briggs. This group tends to enjoy managing a situation and executing plans. They tend to enjoy a faster paced game in which they can perform immediate action [11].

The guardian temperament is based on the sensing and judging attributes of Myers-Briggs. The guardian skill set is referred to as a logistical skill set. It often involves protecting and ensuring the needs of other characters. This group enjoys organizing groups and tasks while trying to improve efficiency with the organization [11].

Using the skill sets from temperament theory and DGD1 player types, which are described in the following section, we can develop types of events and feedback that

are geared towards promoting Flow in the corresponding player types.

2.2.6 Demographics Game Design 1 (DGD1)

The Demographics Game Design 1 (DGD1) is research done by International Hobo Ltd to identify demographic preferences and player types [1]. Building on the framework of the Myers-Briggs personality types, the DGD1 identifies four types of player styles: conqueror, manager, wanderer and participants.

The Conqueror is a combination of the rational and guardian temperament or the strategic and logistical skill sets, corresponding to Bartle's Killer type and thinking and judging in Myers-Briggs. Conquerors are players who enjoy defeating the game and other players. This group enjoys hard fun and thrives on progression and success over challenge. This player type enjoys strategizing towards clear goals, but also enjoys repetitive or similar tasks so long as they are progressing towards their goal. There is a large shift between hardcore players and casual player types, hardcore players are more interested in producing alternative strategies while the casual players are interested in optimizing a strategy, not thinking of new strategies. The intuition and thinking also produce the hardcore group of conqueror player type from the DGD1 model. Overall, this group is very tolerant to challenge, and thus failure [11].

The Manager is a combination of the strategic and tactical skill sets, corresponding to Bartle's Achiever type and thinking and perceiving in Myers-Briggs. Managers are players who enjoy strategy and mastering skills. They enjoy hard fun problems, and prefer a high correlation between their actions and the game result. This player type is comfortable in planning and executing tasks. Managers are less interested in mastering games, as solving them with relatively high performance. Managers perform well in environments with immediate and long term goals which they can plan for and react to, such as racing games. The hardcore to casual are divided by the amount of time an individual is willing to spend planning for a task to execute. Casual players are less comfortable with time related puzzles. Clear short term goals help both the casual and hardcore groups promote immersion in the game [11]. The intuition and thinking also produce the hardcore group of manager player type from

the DGD1 model.

The Wanderer is a combination of the diplomatic and tactical skill set, corresponding to Bartle's Explorer and feeling and perceiving in Myers-Briggs. Wanderers are players who enjoy a large amount of freedom and prefer a unique experience. They are less focused on winning or defeating a game, and more on just having easy fun. This group is more opposed to games with perfect world information, and tend to enjoy games of mimicry. Wanderers enjoy solving issues, without an overall plan but more as the problems present themselves. Due to an interest in a unique experience they are more opposed to repetitive tasks. In this group, Flow is produced when players can choose their own pace, and perform simple tasks in an assortment of situations. There is little known about differences in play preferences between the hardcore to casual relationship of wanderers [11]. The idealist temperament is heavily found in the wanderer player type, but is also found in the participant group.

The Participant is a combination of diplomatic and logistical skill set, corresponding to Bartle's Socializer and feeling and judging in Myers-Briggs. Participants are the largest group in the general population, yet are the least represented in the DGD1 study. They enjoy games for the social factor, to participate and be social. Little information is available about this player type, except that they mostly play for social reasons. This particular group is thought to be interested in heavily character or story-based games. This group is easily frustrated with difficult challenges, as they enjoy problems that can be adequately solved in the moment [11].

International Hobo Ltd is currently in the early stages of a second generation of the demographic design known as DGD2. This research is based on temperament theory described in the previous section, as opposed to the Myers-Briggs.

2.2.7 Summary of Player Types

This section has introduced information relating to players personality types, emotional needs and disposition to problem solving. The research presented on player types in previous sections contains overlapping findings that reinforce four general types of players. Figure 2.7 reiterates related player information in a condensed form.

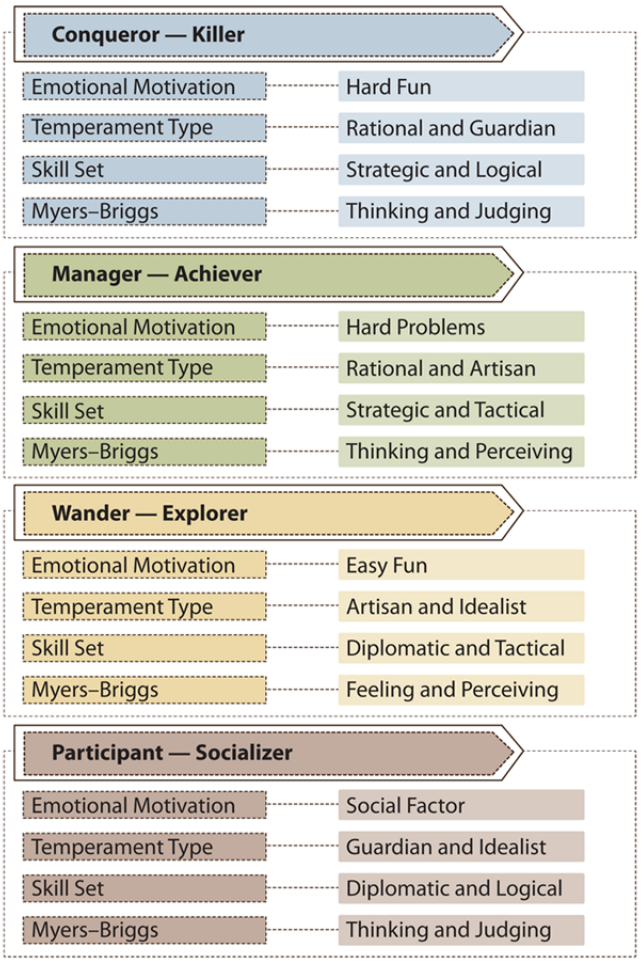


Figure 2.7: Summary of player types, organized into sets based on similar traits from personality research.

Although we introduced several ways of classifying players, there is less information as to the causes of player's drift between roles as the game progressed. We discussed progression of a player's role from Bartle's model which is that players' roles are altered via planning of actions. Lazzaro discussed player progression as a search for an emotional experience. Although Bartle's model may describe emotional motivation using the implicit and explicit dimension, it does not account for frequent or quick changes in behavior. Thus, it is important for game designers not only to be aware of Bartle's player types, but also to identify the player's current emotional

motivation in order to highlight situations where the player has changed game session goals. For instance, a player who is normally a killer type could enter into the game solely for the purpose of exploration. During this instance, the player's goal would not be for hard fun but easy fun, thus the player model would not be correct, given his emotional motivation.

Similarly, both Myers-Briggs and temperament theory likely play a role in mapping player behavior to their player type preferences. As Myers-Briggs investigates a player's personality and how they think, it should provide insight into a concrete personality base. While temperament theory views how an individual behaves, it may provide insight into instances of behavior that are irregular from a normal player. As previously mentioned DGD2 is in the early stages of research for mapping temperament theory to player types. The results of the DGD2 are highly anticipated because player behavior is easier to monitor than thought, which will improve the ability to classify players based on similar behavior. Whether patterns that emerge from monitoring player behavior are capable of matching or indicating game preferences remains to be seen.

2.3 Adversarial Algorithms

This section introduces non-learning algorithms for describing behavior in game agents. The flocking algorithm, discussed in Section 2.3.1, is only utilized for representing movement of groups of non-playable characters. On the other hand the minimax and SSS-AB* algorithms can be used to represent the optimal performance of the player or non-playable characters.

2.3.1 Flocking Algorithm

Flocking algorithms were originally proposed by Craig W. Reynolds [37], as an algorithm for describing the behavior of flocks of birds. A boid refers to a single agent from a flock or group. Boids act in relation to other visible boids and under the influence of several steering forces. The first three forces are from Reynold's original

paper, the last two forces are recent additions to the flocking algorithm [12].

- Cohesion: A force to bring individual boids closer together.
- Alignment: A force to direct boids in the same direction.
- Separation: A force causing boids to steer away from one another.
- Hunger: A force to direct a boid towards a specific goal.
- Obstacle avoidance: A force to direct a boid from a solid object.

These forces acting in unison dictate the movements and behaviors of a flock. Flocking algorithms are of interest to game designers because they provide an aesthetically "natural" method of movement for large groups of objects. They also provide a method of creating multi-agent predator tactics. Including the last two forces, hunger and obstacle avoidance, creates a flock with an intended goal such as tracking an opponent and avoiding obstacles [12].

Let us consider the flocking algorithm process for a single boid, which we will refer to as the active boid. The first step in a flocking algorithm is to obtain a list of other boids visible to the active boid; these boids will influence all calculations and decisions for the active boid. A boid's vision can be calculated via a user defined function which could allow the boid to view all other boids or can attempt to imitate human vision by allowing the viewing range to be in front of the boid with some peripheral vision. The first force to be applied will be cohesion which attempts to keep the flock closer together. Cohesion is calculated by finding the mean position of all visible boids and awarding movement closer to their mean position a higher score. Alignment calculates the position visible boids appear to be heading to based on their current position, velocity and direction. A mean alignment position is calculated using the anticipated positions of other boids. Then moves which minimize the distance between the mean alignment position and the active boid's position are awarded higher scores. The two forces discussed thus far move the flock closer together, the separation rules stop the flock from colliding. The separation rule, depending on implementation, can be used

as a short range separation or long range separation. As a short range separation visible boids are given a maximum value in which separation is still rewarded. In this case the reward becomes bell curved in that positions which are too close to the mean position of the flock and positions too far from the flock are awarded low scores, while moves within this interval are awarded higher values. For long range separation the active boid simply receives a higher score for moving further from the mean position. The hunger force can be used with a defined goal. Higher scores are awarded the closer the active boid moves towards this hunger goal. Similarly, this force could be used to avoid being eaten by another prey object by awarding low scores for moving closer to objects we would like to avoid. While hunger forces deal with obtaining or avoiding goals, obstacle avoidance simply deals with avoiding collision with obstacles that are visible. If a boid's alignment is towards an obstacle then that movement will receive a lower score, whereas movement which is towards open areas will be rewarded higher scores. The score from each force is then normalized, and the move which has the highest score after the summation of all normalized forces is selected. This process is performed for each boid in the flock during a turn.

2.3.2 Minimax Algorithm

The minimax algorithm is an adversarial search, which means that it considers both the moves of the player and other agents during the search. The algorithm attempts to maximize the score for the player while minimizing the score for the opponents. In its basic form minimax is an exhaustive search, meaning it simply enumerates all the possible moves for each agent in the game, thus choosing the best scenario for the player to move to because it has looked at all scenarios [38]. During the minimax algorithm, a decision tree is built from all the player's possible moves known as max states followed by all the opponent's possible moves known as min states. One of the key assumptions of minimax is that it assumes the opponents will play optimally. This means that the minimax returns the best score a player can achieve in the worst case. A player may be able to achieve a higher score, however it will involve an increased risk of getting a lower score.

The number of states in the minimax algorithm grows exponentially, and can quickly become too large a search space to examine exhaustively. An easy way to reduce the search space size is by setting a maximum depth for the search. Once we arrive at the maximum depth, we utilize an evaluation function to evaluate the current state of the game for the player. Alpha-beta pruning is another technique for reducing the number of states to check without effecting the overall result of the game tree. Alpha-beta allows the branching factor, which is the number of branches to be searched to be reduced from b to $\sqrt{b}$ [38]. Alpha-beta pruning works by storing a max (alpha) and a min(beta) value at each level of the tree. If the player is attempting to maximize score at this level of the tree and a value lower than alpha is returned, the rest of that branch can be safely pruned without affecting the result. Likewise, when the player is predicting the opponents move, and are attempting to minimize score any value returned above the beta value for a min level will be pruned.

As an example usage of a minimax algorithm using alpha beta consider Figure 2.8. In this example triangles pointing up are max states and triangles facing down are min states. In part 1) of the figure we've expanded a path as far as possible down the left child branches. This shows that we are examining the first possible move of the player (max states) followed by the first move of the opponent which produces a path of A-B-4 and a value of 4. Part 2) examines any other possible moves for the opponent given the player's initial move. The full expansion of the opponents moves is explored and returns the lowest score from the leaf nodes 4-5-3, which is 3. Part 3) explores the player's second possible move and the opponent reaction move which receive a value of 2. This is our first opportunity to utilize alpha-beta pruning, where the original min-max algorithm would continue expanding the opponents remaining possible moves at leaf nodes 1-4, alpha-beta returns the values 2 at this point because the opponent will only be looking to minimize that value. However from the first branch in part 2) we know the player already is capable of scoring 3, thus anything below 3 can safely be pruned. Part 4) expands the player's last possible move when we expand the opponents first move we see a score of 4 which is above the player's current max value, thus the opponent must try to further reduce

that score, so the next branch is expanded which results in a score of 2. Similarly to part 3), the rest of the opponent moves can be safely trimmed as we know the player is going to choose his first possible move at A. Utilizing minimax without alpha-beta pruning would have resulted in every node being examine in order from left to right. With alpha-beta pruning we managed to avoid examining three nodes which is quite a considerable improvement for this small example.

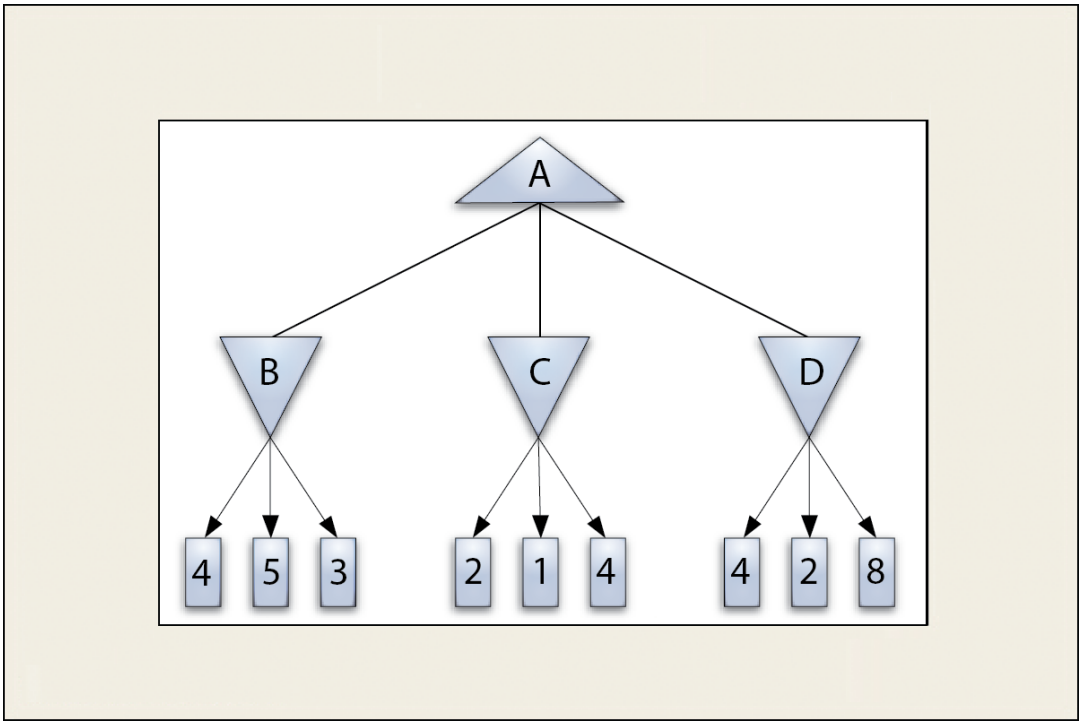


Figure 2.8: A tree described in the minimax and SSS-AB* examples.

2.3.3 SSS-AB*

The algorithm SSS-AB* is a derivation of the original SSS* algorithm. SSS* is a minimax algorithm which dominates Alpha-Beta search in the number of leaf nodes expanded. This means that on average SSS* expands fewer leaf nodes. In the case of a perfectly ordered game tree both algorithms evaluate the same number of leaf nodes. SSS* has failed to receive important practical use or the popularity of Alpha-Beta for three reasons. Firstly, there has been confusion as to how SSS* relates to other minimax algorithms in the academic community. Secondly, the algorithm requires

an open sorted list of states. The overhead of sorting negates some of the value of visiting fewer states. Lastly, the algorithm has larger memory requirements due to sorting, and storing additional states [33].

These three problems where solved by the algorithm SSS-AB*. The SSS-AB* algorithm works in two parts. The first part is the SSS* which continues looping until the current maximum value matches the maximum value of the previous upper bound. Each call produces an upper bound for the visited nodes in the game tree, where only nodes below the previous maximum are expanded. Once consecutive calls to SSS-AB* have matching upper bounds, the value of the game tree has been found. The SSS*-AB works as a series of calls to a special case of Alpha-Beta. The special case of Alpha-Beta occurs when alpha is initially set to infinity-1 and beta is set to infinity. This results in all of the successor branches of a max state being expanded while only the first min state is expanded. The resulting upper limit found by Alpha-Beta is used as a new value of Alpha (subtracting 1), thus only values equal to or above alpha will be expanded. This results in branches that produced the upper limit being expanded first, while values that are below the upper bound are pruned immediately. Since each call to Alpha-Beta searches the critical path (the path to produce the latest upper bound), it expands additional min states until it finds a lower state or has exhausted all successor states. This expansion of min states continues at higher levels of the tree until it is shown that no min state will produce a lower result [33].

To provide an example of how SSS-AB* traverses a tree we review the same tree used for the Minimax algorithm in Figure 2.8. The algorithm starts in part 1) by coming to a max state in this state the algorithm will expand every possible move for the player. On the first pass of the algorithm it will expand only the first possible move on min states, which is caused by each min states value being lower than the upper bound which is initially set to a maximum integer value, thus the first path is expanded resulting in a score of 4. The algorithm then backtracks and expands the first min move of state C which results in a score of 2. Finally the first move of min state D is expanded resulting in a score of 4. The algorithm now returns 4 has the

highest achievable bound which is lower than the previous upper bound of infinity. In part 2) we set the upper bound to 4 and run the algorithm again, which means that min states will now be evaluated until they are below this upper limit or until all leaf nodes have been expanded. SSS-AB* travels down the first move of the player and opponent again utilizing a save table to avoid recalculating this previously visited state of which resulted the path A-B-4. It then expands the second move from min state A which results in a score of 5, this is above the upper bound score thus the next node must be expanded which results in a score of 3 which is below the upper bound, so we end our search on that branch. In part 3) state C and D are expanded to check whether a score of 4 is possible. When it checks state C we immediately retrieve a value of 2 which is lower than the upper bound, so we do not expand further. Expanding the second move from state D it receives a score of 2 which is also below the score 4. The SSS-AB* recurses with the maximum value below the limit which is a value of 3, which the SSS* part of the algorithm checks against the previous upper bound of 4. Since the previous and new upper bound do not match the alpha-beta returns and checks that the critical path and find that no more nodes are available to be expanded from State A and thus 3 is capable of being achieved. This example shows how SSS-AB* traverses the tree, although the true benefit of this algorithm is not displayed in this example it does illustrate that SSS-AB* has expanded the same number of nodes as alpha-beta pruning. However SSS-AB* can result in additional costs via the need to recreate positional information of nodes. An example of this occurs in part 2) where we must recreate the critical path to check the second path from state B, when we had already created the majority of the environment to check the critical path in part 1).

2.4 Machine Learning

This section introduces techniques from the field of artificial intelligence which are applicable to video game design and current research in auto-dynamic difficulty. The first section introduces artificial neural networks which is a machine learning technique

for learning complex behaviors and relationships. Section 2.4.2 introduces genetic algorithms which mimic the evolutionary process at an accelerated rate to produce a variety of solutions. Section 2.4.3 introduces an approach called neuroevolution which utilizes a combination of artificial neural networks and genetic algorithms. The final section will introduce reinforcement learning which saves results from a trial and error approach to find rewarding trails.

2.4.1 Artificial Neural Networks (ANN)

Neural nets are a method of mathematically modeling a function. Based on the biological neural net each artificial neural net is composed of a set of layers, each layer contains multiple neurons or nodes. A neuron has an input, an output and a weight associated with it. When a neuron is given an input value, it combines it with that node's weight and may use an evaluation function to produce an output [39]. For simplicity Figure 6 illustrates a basic feed-forward neural network where only one hidden layer is shown, although multiple hidden layers may be used. Other types of neural networks allow data to travel in bidirectional fashion or using loops, however our discussion will focus only on feed-forward neural networks. In Figure 2.9 a circle represents a neuron or node and an arrow represents an output that is used as input for another neuron. This neural net is composed of three layers, an input layer, a hidden layer and an output layer.

Initially, information is passed to the input layer of the neural network. Each neuron in the input layer is evaluated and its output is passed to the hidden layer as input which is used in conjunction with the connection weight to compute the value of evaluation function. The corresponding hidden layer output is then multiplied by a connection weight and passed to the output layer, as a result for this set of input.

Initially each connection of nodes contains a weight which can be set to predetermined or random values. To produce adequate results ANNs must go through a training phase in which the weights of connections are adjusted to produce results similar to the correct output to expect given a particular input. ANNs must go through a learning a phase, there are two types of learning for ANNs; supervised and

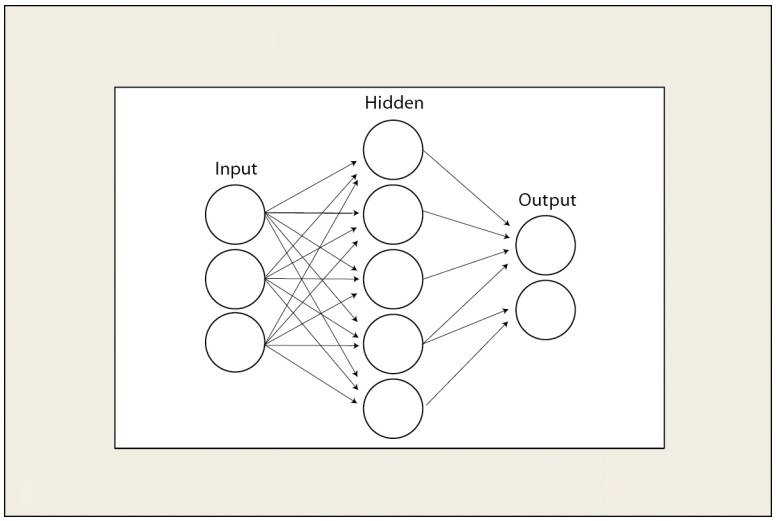


Figure 2.9: Example layout of ANN, adapted from [39].

unsupervised learning. In supervised learning an input is given, and the desired output is known. The connection weights are adjusted to minimize the error between the input and output. In unsupervised learning we are given input but no corresponding output, instead the goal is to minimize a cost function dependent on the task.

One of the main reasons that ANN's have become important to game designers is that they provide the ability to learn complex functions and relationships, which would be difficult for designers to model. The most frequent use of ANN's in game development is in terms of game artificial intelligence. Game AI refers to the planning, actions and decisions of non-playable characters (NPC's) in a game. However the majority of AI in the game commercial community utilize techniques which are not recognized as AI within the research community. Commercial game AI is often hard-coded or scripted to perform an action from a predefined static set of actions. The ultimate goal of using Ann's for game AI is that it can continue to learn and alter behavior in an unsupervised fashion during online play. However this goal has yet to be realized with commercial games due to unpredictable and the difficulty of unsupervised learning.

2.4.2 Genetic Algorithms

Genetic algorithms are a machine learning technique used to explore a range of possible solutions in a domain [39]. A genome is a list of all the attributes that may affect learning for a particular agent. A population is a group of agents each containing their own genome. Each population utilizes a subset of the current population to evolve the next generation. Members of a population are evaluated based on a fitness function. The fitness function defines what behaviors will be rewarded and thus choose behaviors that are improving in terms of your defined goals. Members of the population are then selected to breed based on a selection method. A variety of selection methods exist; examples are an elitist approach where the highest fitness function scores are chosen first, or a roulette wheel where an individuals chances of being selected are proportional to their fitness score [39].

However, strictly cloning members of a population from one generation to the next does not provide a range of results. Thus breeding strategies and mutation may occur from one generation to the next. An example of a breeding strategy is called cross over, where a portion of an offspring's genome set is received from one parent and another portion of their genome is from another parent. During the breeding phase, mutations can occur which alter attributes of the genome set to values that are not found in any of their parents genome set [39]. Mutation causes a further deviation from the parents genome and the population set, which allows for new genomes to be explored.

Genetic algorithms are a useful tool to game designers because they allow for the breeding of a wide range of approaches to a problem. Genetic algorithms would be effective in creating a population of NPCs. These NPCs could have large or small variation in the style of play, which would promote higher levels of interest in the player.

2.4.3 Neuroevolution

Neuroevolution is a combination of neural networks and genetic algorithms. Neuroevolution is an ideal way of training both neural net weights and topology (TWEANNS) at the same time without manual adjustment [45]. In neuroevolution, the weights and topology of the neural net are part of the genome set where they are reproduced and mutated as any other information in a genetic algorithm would be. Neuroevolution combines many of the advantages and disadvantages of both machine learning algorithms. The combination of the two algorithms is especially useful in cases where paired input output test data are not available for supervised learning [2]. A few recent studies have even shown neuroevolution to be more powerful than reinforcement learning in domains that are continuous or contain hidden state information [2].

2.4.4 Reinforcement Learning

Reinforcement learning is described as learning how to map situations to actions so as to maximize a numerical reward [46]. An agent has a set of states and set of possible actions within each state. The agent then attempts to choose the set of actions that will provide the largest reward. The algorithm uses a trial and error approach to navigate through the environment. The agent must decide between traveling old paths (exploitation) to obtain a reward and exploring new paths that may lead to different rewards. While navigating through an environment a reinforcement agent must attempt to develop a policy that maximizes their reward [31]. Thus reinforcement learning provides a method of comparing the trade-off of long-term and short-term goals, while maintaining online performance.

Reinforcement learning is only provided input via the actions of an agent at a particular time, the response to this input is a cumulative reaction from current situational settings. The response to an action is never clarified as being correct or incorrect, thus the player will expect that their action will produce the same response given that situational setting are the same until they have learned a possible new response.

Reinforcement learning is particularly useful to game designs because the training phase is done by matching player actions to the rules within the game world, which can be simulated to produce approximations to the actual response. Reinforcement learning does not require the correct input to learn which is an advantage over supervised learning. As well, it is difficult to produce an adequate cost function to perform unsupervised learning.

Chapter 3

Related Work

This section reviews current research on improving user experience within video games. Our review begins with a general model of user interaction within virtual entertainment known as the User-System Experience (USE) model. Section 3.2 delves into a refined area compatible with the USE model, which is a framework for adaptive game systems (AGS). The AGS framework focuses specifically on challenge and curiosity playability problems that were previously discussed in Section 2.1.6. Section 3.2.1 provides an introduction to the discussion and goals for auto-dynamic difficulty. Section 3.4 introduces methods of adapting the game system from a perspective of playability. The final section provides a detailed review of the three types of game adjustments: player characteristics, level design and non-playable characters.

3.1 User-System Experience (USE) Model

To address the issues of adaptive game systems, we require a model which explains user interaction with the system, with the goal of producing an enjoyable experience. The Person-Artefact-Task (PAT) model provides a framework which focuses on user-interaction with a system from the perspective of optimizing production from a person's work with an artefact. Since the major focus of the PAT model was to optimize production as opposed to experience, Cowley and Black [19] felt it inadequately described a gaming system which focuses more on playability rather than usability. Cowley and Black felt the presentation of Flow within the PAT model was inaccurate and could not be directly applied to games, in that it could not completely describe all experiences with a system. Thus, Cowley and Black adapted the PAT model to create the User-System Experience model

The USE model provides an overview of how the user interacts with the system and provides insight into opportunities for adaptation from a usability and playability perspective. The USE model is capable of describing multiple types of usage experiences such as when the participation is low yet the individual is still interested in the game they may experience a state of telepresence. Unlike other models, the USE models accounts for disinterest, participation, telepresence and variation in the level of Flow. The USE model seen in Figure 3.1 is composed of three main components: the internal state of the user, elements of gameplay system and the usage experience.

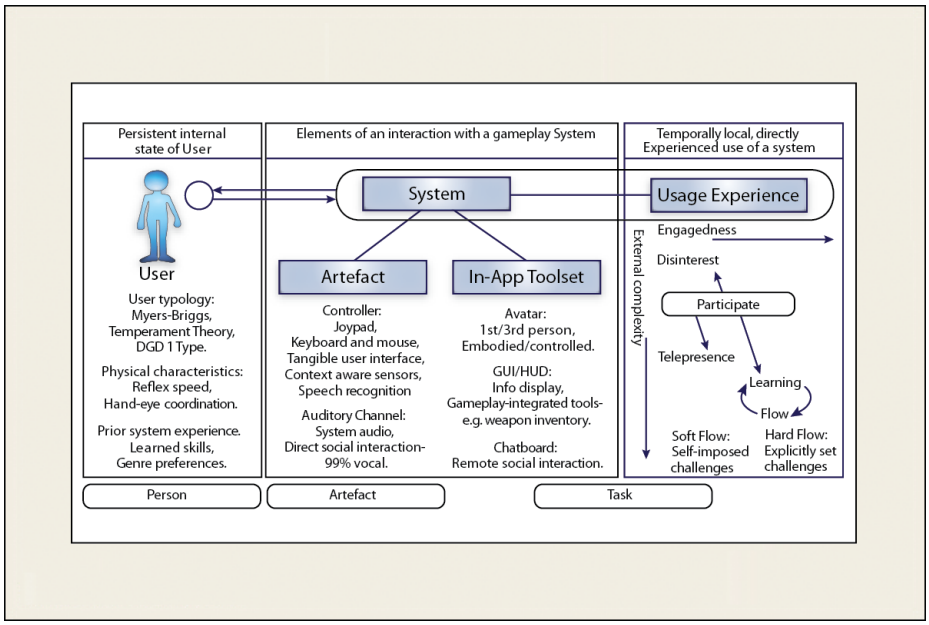


Figure 3.1: The USE model separates computer and game system interaction into three sections: the player, interaction with the game system and the experience produced. Adapted from [14].

The internal state of the user can be dissected into three types of personal information; the first type of information known as user typology deals with the personality and player type of the user. Information on user typologies such as Myers-Briggs and the DGD1, are found in Section 2.2, refers to the user's personality which includes their preferences for which to optimize experience. The second type of player information is physical characteristics, which are unique to each individual and must be measured during game play. Physical characteristics would ideally be initialized to values relating to population means and than adjusted accordingly. Finally the last

type of player information deals with prior system experience. This means the player has gained experience, knowledge or skills through playing other games.

The second component of the USE model is the game play system which is composed of two portions: Artefacts and In-App tools. Artefacts are external methods of communication between the player and the game, such as the game-pad or speakers. The In-App toolset is essentially the game, it contains the methods in which the player interacts and views the game world.

The third component of the USE model deals with user experience. The user experience is defined on two axes: the level of engagement from the player and the level of complexity of the task. At the lowest level of engagement the player is disinterested in playing the game, as the player's participation level increases they can experience telepresence, with increased participation they can experience a state of Flow. The Flow experience can take on two forms; soft Flow or hard Flow. Soft Flow occurs when the player has already mastered portions of the game, they're still engaged in game play but their experience is enhanced mostly via creating internalized challenges. Hard Flow occurs while the player is still highly involved in the learning process and challenges are still explicit and require a high level of player's skill.

The USE model illustrates user system interaction with the goal of optimizing experience, however it does not provide a detailed description of performing adaptation within the system. Thus the next section will introduce the required higher resolution model to illustrate the process of adaptation of the system while optimizing the player's experience.

3.2 Adaptive Game System (AGS)

A framework for adaptive game systems was produced by the same research group that produced the USE model. The AGS framework includes additional detail in the process of adapting the game system to promote an enjoyable player experience. The AGS model Figure in 3.2 is composed of two main sections: an offline stage outside portion and an online stage contained within the dotted rectangle. The offline stage

stores information relating to player types discussed in Section 2.2 and can be used to identify and adapt to perceived preferences of the user's typology. The online stage would also include information pertaining to general gamer preferences to be utilized with their user typology. Knowledge of general gamer preferences would provide hints as how to perform adaptation based on a user's personality and player type.

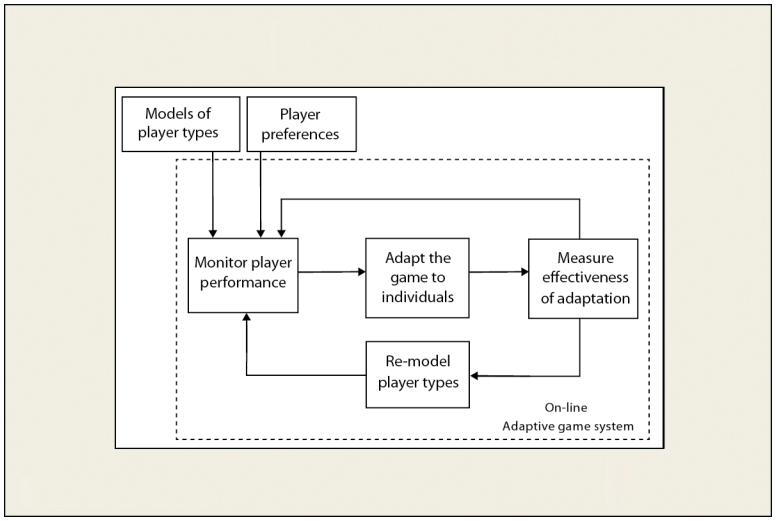


Figure 3.2: Adaptive Game System model illustrates the feedback loop required for producing adaptive video games, which monitors then modifies the game in relation to expected player needs. [15]

The online portion of AGS is composed of a four stage feedback loop. In the initial stage player performance is monitored. This information will be utilized to determine whether the game needs to make adjustments. Monitoring player performance will be discussed further in Section 3.3. The second stage of the online AGS performs adjustments to match the system's perceived level of player desired complexity. Performing adaptation to the game world will be further discussed in Section 3.4. Thus the adaptive system alters the game in a way which it believes the player is capable of handling without demotivating the player. The third stage of the online AGS measures how effective the adjustments were for meeting the proposed goal set in the second stage of the adaptive system. The last stage of the online AGS is remodeling information relating to the player's user typology. The adaptation occurred in response to

a perceived discrepancy between the player's needs and the game. The adjustments made in the second phase are based on the current user profile and suspected user preferences. If the adjustments do not produce the desired response, it may indicate that we have not correctly identified a portion of the player's user typology or the modified factors do not have the expected effect. If the adjustments are successful it reinforces our believe in the current player information, and provides an example for future use. Additionally player information may change, as player types change throughout the course of the game, as seen in Section 2.2.3 on Bartle's player types. Thus the player model constantly needs to be remodeled, as it is also susceptible to concept drift, which occurs as part of a natural progression in strategy or behavior.

3.2.1 Auto-Dynamic Difficulty (ADD)

As previously mentioned Auto-Dynamic Difficulty (ADD) is the adaptation of the challenge of the game to match the abilities of the player. If the player is struggling to progress during game play, the level of challenge may be reduced; likewise if the player is progressing too easily, it may adapt to provide more challenge. Matching the challenge of the game to the abilities of the player will increase their interest in playing the game.

Although matching challenge to the player's abilities is the main objective of ADD, it may also provide other benefits. ADD may mean providing more specific and helpful feedback when the player is lost or uncertain of the next step. It would recognize repetitive player locations and lack of progress in the game, and then provide additional hints. ADD could also be used to detect imbalances in game dynamics. If the player is consistently using the same strategy it may be the result of an unforeseen exploit, which may result in successful play of the game, which depending on player type this could ultimately cause dissatisfaction due to the repetitiveness of the game. An ADD system would be able to recognize this overuse and could alter the game so that a particular strategy is less successful [13].

A system that performs ADD should consider two main points [30]. The first point is that the player should not be aware that ADD is adapting the game. If the

player knows that the system is adapting to their playing style, they are likely to react negatively towards it, blaming the ADD system for their failures. Alternatively they may attempt to exploit the system by initially playing poorly until the challenge of the game decreases. Knowledge that the system is adapting hinders a player's immersion and sense of achievement, as it directs the players focus away from the game and towards the ADD system. Thus it is beneficial to altering variables of the game that are not viewable by the player. Secondly, the ADD system should not make large incremental changes. This helps hide the modification process from the user, as well as helps avoid drastic changes in difficulty that may cause the player to become frustrated or bored quickly. Using small incremental changes also helps avoid the problem of players exploiting the system by performing poorly at the beginning of the game to lower the difficulty.

3.3 Measuring Player State

This section introduces concepts involved in monitoring player performance within the AGS framework. Measuring the player's state deals with understanding multiple states of the player's experience such as enjoyment and frustration.

3.3.1 Player Skill Level

Creating an accurate measure of the player's skill requires defining a task to be accomplished, then rating the player's performance at that type of task. Although tasks are somewhat genre and game specific, a few examples are:

- The amount of time to complete a task.
- The player's average health level.
- The number of additional items collected by the player.
- The number of opponents defeated in a level.

However, it is easy to create a measure which appears effective yet is easily biased, as some measures of player skill that intuitively appear to be adequate can be somewhat misleading. As an example the number of player deaths and game loads [7, 30]. The number of times a player dies might be misleading because a player may not initially try again. Players like participants and wanderers have a lower tolerance for failure, and play only for enjoyment; failure or defeat could diminish that enjoyment. These types of players will have higher frustration levels at lower death totals than types such as the conqueror who is motivated by challenge. As well, a player could quit the game, before the system has had an opportunity to update and save measurements. Similarly, the number of game saves/loads could be related to their playing environment as opposed to skill level. Players may be continually disrupted, or have little time to play and thus save more often despite being skilled players.

As a result of experience and practice, the player's skill level increases as they progress through the game. This creates a need for constant increases in the level of challenge provided at intervals in the game. The pace of the game refers to the rate of change of challenge, this must match the player's level of skill. The pace of the game should occur at a similar rate to the ideal game progression. Initially, the player finds the game challenging but doable. Once the player masters the set of skills for the current level of challenge, the challenge level decreases until the player feels unchallenged by the game. At this point, additional challenges should be introduced to stop the player from becoming bored. When challenges and items are introduced too frequently, the player can become overwhelmed and forget the uses of each item. Pacing the game properly has the same advantages as "just in time" information, which gives the player information just before they need it. This results in less information for the player to remember and positive feedback that they are progressing correctly [7, 30].

Cowley and Charles [19] researched the predictability of measuring the player performance in relation to the optimal performance. Ultimately, the results of their research indicated an inability to predict performance 50% of the time. The inability to predict player performance may have been related to using only the next square

in the evaluation of optimal performance, as opposed to a multiple step look ahead. This measurement of player skill can be used with the minimax algorithm, and may be useful in games with smaller state spaces.

3.3.2 Emotional State (Affective Gaming)

Affective gaming measures the emotional state of an individual, while they are playing a game. The main goal of affective gaming is to be able to identify and communicate the affective state of the game player [49]. Using the player's affective state information, different types of content will be delivered to the player. It has been shown that it is possible to detect the affective state's anxiety, arousal, engagement, boredom and frustration through the use of physiological sensing [36, 49]. The use of physiological feedback tools has varied in studies from the use of current game player technology such as the game-pad, to more intrusive methods such as measuring pupil dilation.

Sykes and Brown [49], investigated whether the difficulty of a game resulted in a player's pressing the game-pad with additional force. Their study showed an increase in button pressure as the challenge of the game increased. However, there was no measure of whether this extra force was a positive or negative result on the player's experience.

Laufer and Bottyan [28], measured the galvanic skin response (GSR) of players during a game session. Neural networks analyzing alterations in the GSR were capable of predicting the jumping behavior of players two seconds before the jump actually occurred. This type of information could be used to aid a player with a slower reaction time, or to frustrate the player by lessening the amount of time they have to jump.

Rani, Sarkar and Liu [36] investigated altering the difficulty of the game Pong directly based on performance and affective state feedback. Initially, using physiological self-reports and task-related data, they build models of participants' affective states. Task difficulty was measured to produce states of anxiety, engagement, frustration and boredom. In the second stage of the experiment, the level of difficulty would be lowered if their performance was poor or they showed high levels of anxiety. The level of difficulty would be increased if they showed low, medium anxiety and

excellent levels of performance. The level of difficulty was static and predetermined by the experimenters. The result of the experiment was that players reported lower levels of anxiety, and higher levels of performance and challenge while the game was adjusting to their affective state.

3.4 Game Adjustments

This section investigates the second stage of the adaptive game system framework, in which adjustments to the game are required to enhance the player's experience. Subsection 3.4.1 provides an overview of the types of game adjustments. Subsequent sections provide additional detail on each type of game element which can be adjusted.

3.4.1 Types of Adjustments

Dynamic adjustments can be either made proactively or retroactively [7]. Proactive adjustments occur based on previous experience with a similar type of task. Proactive adjustments are made to a task before a player has reached the task. Retroactive adjustments occur after a player has attempted a type of task a couple of times, and adjustments need to be made to ramp up or decrease difficulty of these types of tasks.

3.4.2 Player Characteristics

Alterations to the player's character is one method of adapting a game. It is hypothesized that if the player experiences a feedback loop of action-consequence-action, the player will develop a greater sense of embodiment with the character and this will increase immersion in the game [15]. Player's expect their character to remain at a consistent level or become stronger throughout the game. As a player is more aware of changes to their character than other portions of the game, they are less tolerant of changes that cause frustration with their control of the game. Thus, adjustments to the player's character should be on variables hidden from the player or must be consistent within the logical progression of the story. An example of altering the

avatar occurs in the game *Resident Evil*, where as the player sustains damage, the character's walk becomes closer to a limp and their speed is slowed to match the animation.

3.4.3 Game and Level Design

The game environment can be modified in a number of ways. The design of levels can be modified to increase or decrease the difficulty of the game. As an example in a first-person shooter (FPS), if a player is struggling, the level could be designed to include more walls in open areas to protect the player. Redesigning a level at run-time is computationally expensive in terms of game-AI pathfinding and planning, as well as for graphics in areas such as lighting and shadows. Thus, it is more plausible that levels could be separated into multiple sections that are predefined and then recombined before the start of the level based on the player's previous difficulty with other levels. Another way that the game environment could adapt would be by altering the variable rate of a particular item to be given to the player. An example of this type of inventory control is when the the player is consistently low on health, the game could slightly increase the probability of finding health, as opposed to other items.

Depending on game genre, different types of tasks and goals could be altered to increase enjoyment and challenge. As an example, if a player is struggling with a certain type of puzzle, the puzzle could be preemptively rearranged to a position closer to the solution. Similarly, if the player is consistently seeking out a specific type of objective in a game, we could increase this type of objective in the game assuming they would enjoy new challenges or we could offer fewer of these objectives to promote other challenges in the game.

Along similar lines to altering the types of tasks, some researchers are investigating personalized story content for players, where minor events within the story are altered based on perceived player preferences [60]. This type of alteration would be useful in providing matching side quests for players within role playing games. The storyline progresses via a number decisions in the game that provide the player the opportunity

to explore new development in the story. The storyline is controlled by having several key points that all players progress toward no matter their previous deviations in the story. This method of storyline progression provides a positive variation to gameplay as players have unique experiences in the game, yet it allows developers to control and limit the content needed to unfold the story.

3.4.3.1 Inventory Control

Inventory control is a method of performing dynamic difficulty adjustment via controlling the frequency of items the player is capable of collecting [25]. A typical example is if a player is consistently low on health, the adaptive system would slightly increase the probability of finding health, as opposed to other items such as currency or extra ammunition. Inventory control is a method of adapting the game world, as it can include altering the locations at which items are distributed. A simple method of adaptation would be to have static locations for all items, and then to alter the number of available items of that type in the area. A second method involves dynamically selecting the location of an item, and the number available to the player. An alternative approach is to alter the value of each pickup, as an example the amount of health given for each item received. A flaw in the first method with static locations, is it does not provide adaptation to locations where a player may need it. Thus it is dependent on the predefined locations for items set by the designer. A possible downside to the second approach where the item location is dynamically altered, is that the player's strategy may be based on the expectation that an item will consistently be in a certain location, resulting in the player feeling uncertain of the reward for their actions and producing a possibly frustrating unpredictability.

A couple of key components of this system are: assessing when adjustments are necessary, determining what should be adjusted and the level of change required, and executing those changes without disrupting immersion of the player [25]. Simply adjusting the system every time the player is in need will provide to much positive feedback helping the player progress but destroying the challenge of the game. Thus,

changes should only be made when a player is predicted as being in a flailing state, and predicted to enjoy challenge. A flailing state is "repeated movement towards a state where their current means, can no longer accomplish necessary and immediate ends" [25]. Determining which item's frequency should be increased is based on immediate player need, though this might not always be obvious as combinations of items may be needed. Determining the level of change required is also difficult and is often based on heuristic functions.

Controlling the frequency of items properly allows for the game to deliver a "just in time" item that a player may require. This technique integrates easily with a balanced momentum path. After a certain measure of time, when a player is found to be moving towards a flailing state, or moving towards a losing position, it can provide items to help the player turn the tide and regain a possible winning position in the game. These swings in momentum should increase player interest, whether the player is in a winning or losing position provided the swings are not to frequent or overwhelming.

3.4.4 Non-Playable Characters (NPC)

Non-Playable characters are neutral, allied or opponent characters that the player is unable to control during the game. NPCs provide one of the largest areas for adaptation of a game. Allied players' movements can be altered to force the player to attempt new strategies or to provide direction if the player is lost. The overall strength of an allied player could be monitored to protect a player if they are struggling or be less involved if a player is performing well. Opponent characters could be adapted in their overall numbers, their abilities and even their basic decisions could be altered to match the player's abilities. Since opponent and allied attributes may not visible to the players, altering these values is easier than making modifications to the player's character. The remainder of this section will review research utilizing machine learning techniques to alter NPC's strategies and decisions and will discuss the benefits and disadvantages of each machine learning technique.

3.4.4.1 Reliable Adaptive Game Intelligence (RAGI) Requirements

Spronck [41] defined reliable adaptive game AI as game AI that meets eight requirements for online learning in a game. From this definition the term reliable is based heavily on the needs of commercial game designers. Spronck divided the eight requirements into two groups: computational and functional. The four computational requirements are:

- Speed refers to computational speed, since learning occurs online.
- Effectiveness refers to the ability to produce and differentiate between superior and inferior alterations to the strategy.
- Robustness to randomness or natural variation that occurs throughout gameplay.
- Efficiency refers to the minimum number of trials required to achieve the desired level of result.

The four functional requirements are:

- Clarity refers to the ability to understand how the game AI is behaving.
- Variety refers to differences in behavior, at an appropriate level of play.
- Consistency refers to a low variance between the trials needed to produce strategies.
- Scalability refers to the ability to adapt to the players ability and performance.

Meeting these requirements will aid in the commercial acceptance of ADD technology. However, there is a divide between game AI for commercial use and for academic purposes. Commercial games are still reluctant to utilize technology that is not strongly based on the clarity requirement, at least in terms of released product. Offline learning is becoming a powerful tool for game developers, in terms of decreasing

the amount of time required to test games, as well as developing alternative player strategies. Unfortunately, with clarity and efficiency being two main requirements, fewer online learning techniques can be utilized in commercial games.

3.4.4.2 Genetic Algorithm Approaches

One of the main tasks of auto-dynamic difficulty is to develop a variety of different solutions to a problem. The variety of these solutions can be used to match a player's style and skill level. If you keep a collection of the population from each evolved generation and their fitness function results, portions of the stored genomes will have similarities and yet have slight differences in their attributes that may provide a different challenge to the player [53]. We could compare the player's previous games offline to the database to narrow the range to obtain a comparable level of game AI opponents [52]. A relevant example would be in race car games where the majority of cars have slight performance differences and should be driven differently, whereas scripting all these small difference would be tedious and a misuse of resources [39].

In the work of Togelius et al. [51], racing car tracks were evolved with the goal of being more fun to the player. They created three fitness functions based on total progress, speed at way points and orthogonal deviation, generations were chosen based on a cascading elitist approach. Each of the three fitness functions contributed to the final score used for breeding purposes. Their results showed the ability to evolve tracks which were difficult to drive and had a greater number of sharp turns for experienced players who progressed easily through the tracks in the learning phase, while also being capable of evolving tracks with a greater number of straightaways and fewer sharp turns for players who averaged lower speeds and made less progress.

The downside to genetic algorithms is that the evolution process may take hundreds of generations to produce acceptable results. The length of time required to train genetic algorithms is one of the reasons that commercial games have not used genetic algorithms in their product. In addition there is no guarantee that training will evolve towards an optimal solution. Another shortcoming of genetic algorithms

is that once it has evolved to a specific set of values, it can be difficult to adjust or include attributes or functionality while maintaining consistent results.

3.4.4.3 Neural Network Approaches

Neural networks provide a strong problem solving tool in problems such as classification, pattern recognition and function approximation. Solutions to these types of problems need to be addressed to create an auto-dynamic difficulty (ADD) system [15]. The classification of a player model would help make efficient estimations about where other players with similar abilities had problems. Although players with the same level of skill may have different levels of difficulty with the same problem, it is an adequate place to begin searching. Neural nets have been successful in finding "normal" behaviors of usage areas such as intrusion detection [39]. As the problem of ADD is an attempt to match the human player's ability to the challenge level of the game, being able to recognize patterns in style of play would be useful. Once we have recognized patterns in a player's style of play we can adapt the environment to challenge them to develop new strategies of play. Without pattern recognition we would be unable to discover what types of problems the player is finding difficult or too easy, resulting in an inability to effectively customize the game to help them. Lastly, function approximation can be utilized in terms of a GameFlow function and other heuristics, it is difficult to develop accurate mathematical models to express certain components of GameFlow [47] such as the enjoyment or challenge that a player is having during game play or to predict the players expected competence during gameplay.

The work of Wong et al. [52] utilized the idea of dynamic difficulty levels for the use in edutainment games, which are educational games. Their work identified edutainment games as an important area for dynamic difficulty as promoting learning at the pace of the student. The research game used student profiles to maintain a record of performance and used back propagation neural networks to learn behaviors for the game given the player's rate of accuracy in answering questions. They allowed

the player to chose a level of difficulty in the range easy, medium or hard but allowed adaptation to occur within a range for each difficulty. Thus if at the hardest level the player was required to accurately answer 95% of the questions, the game could adapt to allow for accuracy 85% to still allow the player to progress if they were struggling. Their research focused on whether dynamic difficulty was applicable to edutainment, in which they concluded that that the real-time reaction to player's level of challenge is the largest obstacle. However, they proposed a pre-trained database of results to improve online performance.

The main weakness of neural nets is that creating a cost function for unsupervised learning can be difficult, and using supervised learning would require us to have a set of test data to train and test on. This type of data is not only difficult to develop but would also be an enormous amount of data to process and thus training is too slow for online requirements [39]. The results of training an ANN are highly dependent on the initial test data. Incorrectly choosing an initial data set could lead to problems with the ANN such as finding correlation between input values that should not be there. Other issues such as over-fitting and catastrophic unlearning can also occur. Over-fitting is when an ANN learns a specific situation in the training data at the expense of generalization. Catastrophic unlearning is when over the course of several inputs, everything that was learnt, has been unlearned.

3.4.4.4 Neuroevolution Approaches

Since neuroevolution is the combination of two previously mentioned algorithms, it would be possible to use it in the same situations as neural nets and genetic algorithms. Neuroevolution would be useful in creating character profiling and pattern recognition in a player's style of play. It also had the advantage of being easier to train because it does not require correct input and output pairs, while still being capable of producing adequate solutions and creating a large population of interesting opponents [53].

The work of Yannakakis and Hallam [54] recreated a simplified version of Pac-Man. In their experiment, ghost behaviors were neuroevolved to maximize the fitness

function of player interest. Their formula included scoring heuristics for the number of steps in the game, standard deviation in the time to eat Pac-Man and finally the level of entropy in terms of visited squares. Their ghosts utilized four inputs the difference in x and y coordinates between themselves and Pac-Man, and themselves and the closest ghost. Their genome would contain those four values as well as connection weights. Ghosts where trained offline followed by tweaking in the online stages. Yannakakis and Hallam [54] were capable of showing higher levels of interest after online training, which was independent of level typology.

Recently a group of researchers [57] created a highly successful set of experiments of an adaptive game system, based on the game Super Mario Brothers. Players involved in the experiment played in an online setting, upon completion of their game they were required to complete a survey. To provide adaptation, level design and creation were parametrized based on four parameters; the number of gaps, the average width of gaps, the entropy between gaps and the number of direction switches. The survey was comprised of a set of questions where players rated their experience between two unique sets of level parameters in relation to one of these states: fun, challenging, frustration, predictable, anxiety and boredom. Using the level parameters and the results of gameplay response variables as input for a perceptron, they were capable of predict player experiences based on surveys, which predicted fun with 69.18%, challenge with 77.77% and frustration to 88.66%. In a follow-up study they included additional states, and included multi-layer perceptrons attempting to increase the prediction rate of their models. In their most successful model they produced prediction rates of: fun (74.21%), challenge (79.37%), frustration (91.33%), predictability (76.28%), anxiety (77.28%) and boredom (73.19%).

An important relationship occurs between our research and the work on Super Mario Brothers in that their and our research are the first set of experiments to include multiple gameplay response variables. The importance of extracting multiple gameplay response variables is that it allows for emergent states whether emotional or skill based to be described based on the contribution of sets of responses. Their research includes an initiative to link gameplay response variables based on their

importance to the emotional states listed above.

3.4.4.5 Reinforcement Learning Approaches

One of the main advantages of using reinforcement learning for the problem of auto-dynamic difficulty is that it is adaptable to online performance [5]. By diminishing the emphasis of exploration and creating additional emphasis on exploitation, learning is minimized and thus the algorithm is better suited for online time constraints. Another advantage of reinforcement learning is that much like neuroevolution it does not require paired input or output for supervised learning [46]. Q-learning, a type of reinforcement learning, has been combined with challenge functions for the problem of ADD. A challenge function is a measure of how difficult the player is finding the game. A challenge function indicates to the game controlled agents whether to choose moves with higher or lower difficulty for the human players[5].

In the research of [5] they implemented a game called *Knock'em,* a real time fighting game in the style of *Mortal Kombat* or *Street Fighter.* In *Knock'em* two combatants attempt to defeat one another, by draining the opponent's hit points to zero. The researchers utilized a challenge function based on health. Thus, if the range between one player's health was above 10 then the game would adapt for the player who was losing. Rewards where based on actions which caused more health damage than what was received. For competitions and training they utilized four algorithms: random, a state machine or fixed strategy, an optimal reinforcement agent, and an adaptive reinforcement agent. As could be expected the random algorithm was the easiest and after training the optimal reinforcement agent had exploited flaws in the state machine strategies. Their experimental results showed that the adaptive reinforcement agent was capable of learning to play to the level of each opponent with the largest variance in health levels occurring from the random opponent. A potential concern with adaptive game systems is providing the ability to quickly adapt between strong and weak players, their use of reinforcement learning demonstrated decreased training time compared to other learning algorithms. A concern with the methodology

used in this research is whether it maintains a consistent level of believability in the decision process. This implementation of a challenge function is susceptible to large shifts in the quality of decisions and thus the challenge of the game. Future research using a challenge function needs to address the issue of smoother transitions to improve believability. An example scenario where the challenge function provides unfulfilling gameplay is if the adaptive agent stood still if it were winning by a large amount allowing the opponent to catch up.

The main criticism of reinforced learning for game development is that dynamic programming is not as effective if the environment changes quickly as little learning can be done if rewards may have disappeared by the time the agent arrives to that state. However for many games this is not that large of a problem as often player movement is the only dynamic component in the environment [31].

3.4.4.6 Dynamic Scripting

Scripts are used in games to describe the actions, behaviors and attributes of an object. The advantage of using scripts is that they provide a clear explanation of object behavior, and are highly adaptable. However, scripts can become quite large, resulting in decreased speed of execution, and increased difficulty debugging [39]. Overusing scripting can result in a less compelling game world, if there is not enough variety of events or actions. A common usage of scripts occurs in role-playing games (RPG's) where a neutral non-playable character continually performs the same actions and repeats the same conversation. Dynamic scripting allows changes to opponents and other non-playable characters (NPC) behaviors during runtime.

Dynamic scripting is based on reinforcement learning, where behaviors that are unsuccessful are punished and successful behaviors are rewarded. Dynamic scripting can be applied to games with the following three requirements: the game AI can be scripted, the domain knowledge on the characteristics of a successful script can be recorded and an evaluation function can be created to assess the success of scripts [42, 43]. The domain knowledge database from which rules are designed is usually

hand crafted by game designers. Rules dictating the behavior of a character are chosen online based on the probability of a weight associated with that rule. Recent research has attempted to automatically generate scripting rules for the database, but for the most part rules are manually edited [34, 35]. A given rule's weight is altered based on the success and failure of that rule in previous situations. If the rule succeeds the probability of selecting that rule is increased, and if the rule does not achieve a positive result, its value is lowered. This process ensures that the most challenging and successful rules have a higher probability of being chosen.

Dynamic scripting forces the player to develop different sets of strategies, because it specifically chooses rules that the player has struggled with. The process of increasing the probability of selecting successful rules is not a full auto-dynamic difficulty system, as the challenge continually increases. A method of allowing the level of challenge to decrease using dynamic scripting is known as difficulty scaling[44]. Difficulty scaling was integrated in the dynamic scripting framework by restricting rules that have continually proved to be successful against a player. Including difficulty scaling results in database rules being pruned because of two reasons: they rarely achieve success or they consistently achieve success. Thus the database allows a subset of rules to be selected, these rules are within the player's capability, and should provide a balanced result.

Game difficulty can be scaled based on three methods: high fitness penalizing, weight clipping and top culling. Difficulty scaling proved to be effective at balancing the game against a variety of static strategy opponents [44]. High fitness penalizing awards the highest score to the technique that provided the closest competition. Thus, strategies with the lowest variance between the players are evolved as the ideal strategy. Weight clipping provides a window of strategies that are available; the size of the window is based on a weight. If the weight is low, indicating low performance, the window is made larger providing a larger variety of strategies to be chosen. Inversely, if the weight is high, the window provides a smaller number of strategies to be chosen. Top-culling is a window based approach where extremely successful techniques are culled. Once a player defeats medium level strategies, higher level strategies become

reactivated. Top-culling resulted in the lowest variance in the number of wins between the three strategies. It was also the most successful at balancing the number of wins against weaker opponents.

An outcome of dynamic scripting is that it adapts to continuous exploitation of a particular strategy automatically. Without dynamic scripting, when a player uses a certain strategy that always works, they will use that strategy to "cheat" the system. The ability to exploit the same strategy continuously affects the enjoyment of player types in different ways. Conquerors and managers may experience this deviant gamer behavior as an immediate progress method, but overall it decreases the enjoyment of the experience, as it eliminates the need to discover new strategies and solve problems, thus reducing the sense of accomplishment for beating a game. On the other hand, wanderers and participants are often more interested in the development of characters and the story. These players will gain more from progressing the story than from the accomplishment of defeating the game in a meaningful way.

One of the main criticisms of dynamic scripting is that it requires a reduced state and action space to meet the efficiency requirement from the RAGI requirements [34]. This limits the use of the dynamic scripting from certain game genres with larger state spaces. A highly beneficial addition to dynamic scripting has been automatically generated game tactics. Typically, scripts are manually edited by programmers and game developers; this is expensive in the number of hours required for development. Since dynamic scripting weights different scripts based on their performance, it is limited by the number of initial strategies implemented in the rule base. This means that if the player's abilities are outside the range of initial script strategies, dynamic scripting will not be able to adapt to the player. Previous research by Molineaux and Ponsen [3] built a case-based reasoning system for evaluating the effectiveness of techniques for a given state and opponent. Ponsen et al. [34], investigated automatically generating game tactics through the use of evolutionary learning, where each game state contained a corresponding available action that was related to research, economy, combat or building. The resulting evolved tactics were compared to a set of scripts not in the training set, in a new game world to test whether evolved tactics could

outperform a variety of strategies. Results showed that dynamic scripting was capable of consistently defeating all but the strongest scripts, which provided an even level of challenge for dynamic scripting.

Chapter 4

Experimental Testbed Design

This chapter provides extensive insight into our dissection of the problem of auto-dynamic difficulty for our research purposes. Section 4.1 provides an overview of decisions made during the design process to reduce the problem to a manageable size. Section 4.2 provides an introduction to Pac-Man, the testbed game selected for our research. The remaining sections in this chapter will provide a detailed view of the design of the game Pac-Man as utilized in the experimentation phase. Section 4.3 introduces the initial phase of the game design in which we developed and prioritized a detailed list of game factors that our experimentation stage would investigate. Section 4.4 introduces strategies used to represent player behavior, followed by Section 4.5 which introduces algorithms used to dictate opponent behavior. Section 4.6 introduces and discusses performance measures which will be tracked and used to understand the performance as well as predicting the form of heuristics. Finally, Section 4.7 introduces the adaptive system and its method of predicting performance and performing adaptation. The selection of response variables to be controlled via the adaptive system will be left until the analysis phase presented in Chapter 5 as the selection process is dependent on the performance results.

4.1 Adaptive Gaming Architecture Design Overview

The design phase of our research is conducted from the perspective of GameFlow and the USE model with the long term goal of studying auto-dynamic difficulty. The USE model, reviewed in Section 3.1, provides a high-level view of user interactions with the system. The USE model was designed specifically for explaining relationships between users' experience and video games. A key feature of the USE model is the

separation of usability and playability while being capable of describing multiple types of usage experiences.

Within the USE model we wish to address the issue of an adaptable system; thus, we narrowed the design focus to the adaptive game system (AGS) framework. As discussed in Section 3.2, the AGS model is composed of four main components: monitoring a player's performance, adapting in response to a player's needs, monitoring the effectiveness of adaptation, and dynamic player modeling. Monitoring the player's performance can be task specific, and unique to each game. Often, heuristics are devised to approximate the level of difficulty the player is experiencing in performing a task or playing the game. As explained in Section 3.3, although some heuristics may appear as obvious factors to the level of difficulty, they can be misleading [7]. The fourth component of the AGS model deals with the dynamic remodeling of the player types, which ideally requires being able to identify a player's player type solely by observing game play, but could be aided by utilizing other intrusive research techniques such as pre-game questionnaires. Our research will focus our experimental design on two components of the AGS framework: adaptation in response to a player's needs and monitoring the effectiveness of adaptation.

The main task of performing auto-dynamic difficulty is maintaining an adequate level of challenge for the player's level of skill. Thus our focus will be on adaptation in relation to the player's level of challenge, as well as the perceived level of challenge. Our research will focus on the two main types of playability issues: challenge and curiosity. Focusing on the level of challenge for adaptation purposely filters out some of the player's needs, such as when to provide feedback and what type of feedback. Our focus on challenge investigates the player's current scenario and provides within-subject information relating all of the game possible factor settings.

Our design will focus on identifying important factors for adaptation in the game environment which could potentially be used in response to a player's known needs. Our design focuses on being applicable to both the research and commercial communities. In the research community, NPCs utilize machine learning techniques that require a large number of training iterations in which the game usually trains against

itself to produce adequate behaviors. However few commercial games are distributed with the ability to adapt outside predefined tested intervals. One major issue in current adaptable game system research is the ability to quickly adapt from a novice skill level to an expert level. Our approach hopes to provide additional information to help understand which factors have the greatest influence on the difficulty of the game, such that the NPC's learning process could more effectively bridge larger gaps in skill level. The same information could be utilized in the commercial community by game experts specifically to tune factors for level design issues.

We utilized a closed experiment where adaptation occurs offline between simulations. An offline setting creates a controlled environment, allowing inspection of each game factor to occur individually from the beginning of a new game. Throughout game play, all major game event information will be logged for evaluation. Logged information will be utilized during a post-game analysis phase in which we will determine each factor's statistical significance to the level of difficulty.

4.2 Overview of Testbed Game Design

For our research purposes we have chosen to focus on the game Pac-Man. This decision was made because Pac-Man is a well known game with simple rules, interface and goals, yet has complex interactions in terms of group behavior. Additionally Pac-Man is frequently utilized by other researchers in terms of generating interesting opponent behaviors and for generating dynamic behaviors.

4.2.1 Original Pac-Man Game Description

Pac-Man is a 2D game, where the object of the game is to navigate your way through a fully visible maze and collect all of the tokens in the level. While in the maze, four ghosts attempt to stop Pac-Man by occupying the same square. Pac-Man and the ghosts move one square at a time. Ghosts follow a path until an intersection is reached at which point they will choose a new path. Pac-Man is capable of changing direction at any point on a path. A ghost kills Pac-Man when it occupies the same

square and the ghost is in a predator mode. Ghosts are always in predator mode, except for a short period of time after Pac-Man has collected a power-pellet. A ghost can be visually identified as being in a predator mode when it has a vibrant color. Ghosts will be dark blue when they are prey. While the ghosts are in prey mode, Pac-Man can obtain additional points by eating them. Tokens come in two forms: regular tokens and power-pellets. Power-pellets temporarily shift the balance of power. When Pac-Man eats a power-pellet, he becomes the predator and is capable of eating the ghosts. Pac-Man is awarded points during game play for eating tokens, power-pellets, fruit, ghosts and completing levels. When a ghost has been killed it slowly returns to the spawn position, and regenerates after a set number of seconds. To progress past a level, Pac-Man must collect all the tokens and power-pellets in the level. At irregular intervals during game play, bonus items in the form of fruit appear on screen. These bonus items will only be available for a short time for Pac-Man to consume to obtain extra points.

4.2.2 Research Testbed

Our version of Pac-Man works similar to the original with a few minor differences. Our experiment will simulate the player's interactive role automatically via an algorithm described in Section 4.4. Automating the player's portion of the game allows us to simulate and perform analysis on a large number of games quickly. Similarly, ghost behavior will be simulated using an algorithm from the ghost algorithm described in Section 4.5. All ghosts will use the same algorithm during a session. Each agent of the Pac-Man game (an agent is either Pac-Man or a ghost) has four states describing their actions during gameplay. The states are: fleeing, chasing, wandering, and dead or inactive. The original version of Pac-Man allows Pac-Man to make a directional decision at each square of the map. Our version only allows decisions to be made at each intersection point. The reason for this adaptation is to decrease the size of the search space, as well as to place additional emphasis on the decisions made by the algorithms. The modification of only allowing Pac-Man to make decisions at intersections, forces the player's play to be similar to that of the ghosts.

In Section 4.3, we outline in detail our research parameters for the Pac-Man game. These parameters will be altered between simulations. We believe these modifications will provide an impact on the level of challenge of our game. Each simulation will run the Pac-Man game with a unique set of parameters values specified in a parameter file. Parameter files contain a list of possible parameter values and whether they are adaptable. In the experimental phase the parameter files are only used to load the correct parameters before the start of each simulation. The actions performed during each game will be logged to a game file which will be retrospectively parsed and examined in the analysis phase of the experimentation.

Our testbed version contains limitations specifically chosen for the purposes of controlling experimental results to produce comparable results. The first limitation is that Pac-Man is unable to acquire additional lives within our game. This forces situations in which each player must accomplish tasks with the same number of lives and aids in producing comparable results. Pac-Man is capable of finishing levels which adds to the player's score. Upon completion of a level, the board will be reload and agents will be returned to their initial positions to replay the same level with no additional increases in difficulty. Our research testbed does allow for different levels to be added and played out; however, we decided to focus on playing only one board as level design is not one of the factors that we are investigating. The board layout for our test level is a replica of the first level of the original game with the exception that the level is fully enclosed and thus Pac-Man can not transport from one side of the screen to the other. We have limited the number of steps allowable per life to 350. This number of steps allows a player to complete up to two levels. The main reason for using an upper bound on the number of steps is to limit a strong player from continually defeating the same level over and over.

Bonus items will be generated at regular intervals during game play. Bonus items will begin being generated after an initial set of 50 moves into the simulation, and continue until the user has reached the 300th step of the simulation. In the final 50 steps of the simulation no bonus items are generated to ensure that Pac-Man has a fair chance of obtaining any remaining items before the simulation is complete due

to the maximum number of steps.

4.2.3 Object Loading

An important issue in creating an adaptive system, is creating a versatile and efficient method of altering and loading factors and objects at run-time. Initially, we created a system in which an algorithm could be selected and a set of factors and their values could be specified by researchers wishing to observe simulated results. This system would then create attribute files for all combinations of these factors which could be loaded at any time during game play. While this system allowed us to specify as many factors and factor values as we wished, the process of creating all combinations of attribute files could be quite slow. Thus, we revamped the system to optimize it specifically for the analysis phase of 2^k factorial design, where k is the number of dynamic factors and 2 is the number of levels per factor. The first optimization allowed each factor and its current value to be available via a look-up table, this eased not only the readability of these files, but also the dynamic modification process. The next important optimization was to allow a factor to have exactly 2 levels of values. Although this was always our intention, making it explicit allowed the loading process to utilize a bitwise system. With these optimizations the loading process changed such that we were no longer required to create attribute files in the same way. The new system allowed us to create one master file which listed all objects and factors by a key name followed by 2 values, their high and low levels. The master file listed all loadable parameters, whether they were to be dynamically altered in the game or not. A second file was used to specify the dynamic objects and factors we wished to observe during the simulation. The dynamic object file would contain keys which matched the master file. Following these improvements to the system, we simply needed to pass a number between 0 and 2^k-1. This number would indicate whether the factor was in a low(0) or high(1) state. The object modification and loading portion of the system is quite simple in the new system. Each object simply has to call the Object Loader with the key it wishes to load, and it will return the current value of the system. Modifications work in a similar fashion but provide an additional

term to replace the current value.

4.3 Game Factors

The Pac-Man testbed is designed to experiment with the level of challenge of the game by altering game parameters and monitoring the performance results on the set of response variables. The game environment is set up to allow for as many parameters to be loaded as possible and each simulation run is performed with a different set of game parameter or factors values. Examples of game parameters are the agents' speed and time on screen for bonus fruit.

In the initial stage of the experiment, we decided to investigate game parameters which we believed would provide significant results in terms of alteration to the level of difficulty. Initially, we selected as many parameters as we could, we began narrowing your focus of which parameters to include through discussion, algorithm selection and small simulations. Once the parameters to investigate were selected, we selected values for each parameter, consisting of a high and low value to make the required two levels of 2^k. The factors that we chose to investigate can be categorized into three groups relating to agents, bonus items and algorithm factors.

4.3.1 Agent Factors

Agent factors directly relate to the attributes of Pac-Man or the ghosts' behavior. As previously mentioned, each agent has 3 active states; fleeing when the agent is escaping predators, chasing when the agent is a predator and can see the prey, and wandering when an agent is unable to view any predators or prey. Ghosts can be in a fourth state of inactivity, which occurs when they have been killed and are returning to their spawn position on the board. The factors we selected to further investigate control the time spent in each state and the vision range of the agent.

The factors relating to state time will determine the maximum amount of time an agent will spend in each state, as certain actions within the game may force a change of state. The state factors apply to all agents so the length of time spent in

each state is consistent among all agents. Pac-Man is always in a flee state unless a power-pellet was collected, in which case Pac-Man will be in a chase state. In the development of our testbed, we initially allowed game agents to use different algorithms for any of the three active states. However, during the factor pruning process, we limited each agent to use only one algorithm for all states. This decision contributed to the simplification and organization of the algorithms for the analysis section. The simplification of the algorithm occurred by limiting each agent to use one algorithm; it provided the opportunity to observe in isolation the performance of each of the algorithms. Ultimately, we decided to examine different factor levels for flee and death states, because with one algorithm, the ghost's states wander and chase perform similar actions differing only on the awareness of Pac-Man's position. The state factor flee time (FLEE_TIME) alters the length of time Pac-Man remains in a predatory state, potentially allowing Pac-Man to relieve stress from opponent pressure or to be used in aggressive tactics to gain points for eating ghosts. The state factor death time (DEATH_TIME) alters the time the ghosts remain out of the game once they have been eaten, thus altering the benefits for Pac-Man's counter-attacking the ghosts after eating a power-pellet.

The next set of agent factors relate to the vision range or the amount of the board that agents are capable of viewing at any particular point. Agents initially had three types of vision: complete vision of the board, direct line of sight and surrounding area. Direct line of sight allows the agent to see as far as possible in any of the four directions. With this vision type agents cannot see through walls. The surrounding area vision type allows agents to view a boxed area around them, agents with this type of vision may see past walls, but not outside their area. The vision parameters can set a limiting vision range, which forces the size of the surrounding box or line of sight. Eventually, our research focused on the surrounding area vision type and chose to control the range of vision as the main parameter. The surrounding vision type encompasses the ability to perform complete vision and provides a more accurate depiction of how players would view the board than line of sight. Also, the performance of the direct line of sight in initial tests of the system proved to be

extremely poor.

Vision parameters play an important role in representing different player abilities and skill level. A person with no prior experience with a game will feel that there is a lot of information to observe, and thus will only be capable of truly focusing attention on a small portion of the board before feeling overwhelmed. As a player gains experience and becomes accustomed to the rules of game, they will become better at filtering unnecessary information which will result in the ability to expand the range of view the player can pay attention to before feeling overwhelmed. As a player's view encompasses a larger area, they will have the ability to plan further ahead, which is why we feel the inclusion of a vision range factor can represent a varied level players' experiences and abilities.

A brief summary of the selected agent factors and their low and high level values is available in Table 4.1.

Factor ID	Factor Description	Low Level	High Level
FLEE_TIME	The length of the state flee time.	10	20
DEATH_TIME	The length of the state death time.	10	20
PAC_VIS_LEN	The range of Pac-Man's vision.	5	10
GH_VIS_LEN	The range of the ghost's vision.	5	10

Table 4.1: Summary of the agent factors selected for the experiment. Included is Factor ID, which is a short form name, a description and the low and high level values of each factor will be assigned during the simulation.

4.3.2 Bonus Factors

Bonus factors relate to the bonus fruit which in our version of the game will be generated at regular intervals in a random empty square. The three selected bonus fruit factors are: the length of time the item is available, the perceived value of the bonus item and the frequency at which the item will be generated. The bonus factors can be used as a method of investigating the perceived level of difficulty. Pac-Man is not required to collect the bonus fruit to accomplish any of the level requirements, thus when Pac-Man plans to go for a fruit it is based solely on the decision that the scenario provides an accepted level of risk to the level of reward. The bonus factors

provided possible insight into the perceived level of difficulty by the player, since obtaining the bonus item and completing level tasks could indicate that the player is capable of moving to a higher level of challenge. The amount of time the bonus factor is available indicates to the player or other agents whether they need to immediately alter their plans to obtain the bonus item or whether it is out of range and should be ignored. The perceived value of the bonus item and the frequency could be a factor in the decision to alter the current plan to go for these items. If bonus items occur frequently or have too low a perceived value, it may be better for Pac-Man to focus on completing level tasks and only attempt to retrieve bonus items when little or no change to the strategy is required. Using the perceived value of the fruit instead of modifying the actual value of the bonus item produces comparable score results. This resulted in the point value for eating the fruit always remaining consistent, while the motivation of the player to obtain bonus items is being altered. The frequency of the bonus item will alter the time interval at which the fruit is generated, either creating a larger number of bonus items by shortening the interval or creating fewer bonus fruits by increasing the interval. By increasing or decreasing the frequency of the fruit, we alter the highest possible score that a player can achieve during a single session. Thus, we controlled for this variable by splitting players into two different categories based on high and low frequency of fruit creation. This modification allows the scores to be comparable only to other players with the same fruit frequency level. The issue of separating factors for controlling and comparing results is discussed further in the experimental environment Section 5.2.

The perceived level of challenge is difficult to measure and to introduce but is of interest because it influences the player's decision process. If the perceived level of challenge is too low the player may expand the number of tasks he takes on, whereas if the perceived level of challenge is high he may attempt to accomplish fewer bonus tasks. As we briefly mentioned in the previous section, the level of difficulty refers to the goals accomplished by the player; it has a straight forward measure. However, to fully understand perceived difficulty, we need to understand what information plays a role in the player's decisions, as well as when the player's goals change to

focus on bonus tasks. The game Pac-Man has two types of dynamic information to influence the perceived level of challenge: fruit and ghosts. In the Pac-Man game, we know that the bonus fruit is only available for a set amount of time; thus the player can only focus on the fruit during these times. If the player is interested in this fruit, the perceived challenge will increase. If the player does not focus on the fruit, the perceived level of challenge remains the same. The number of fruits collected in comparison to the number of fruits created will indicate the proportion of bonus tasks the player took part in and completed. The other type of information which plays a role in the perceived level of challenge is the ghosts. The level of perceived challenge that the player is experiencing is based on two ghost factors: their proximity and whether they are a predator. Pac-Man is less worried about ghosts if they are far away, or if Pac-Man is the predator. If the ghosts are close it creates an increased amount of information to process and a higher level of difficulty to make the correct choice, and both processes require additional time to effectively calculate a decision. As Pac-Man game is a fast paced game, it means that decisions must be made quickly to keep up with gameplay.

A brief summary of the selected bonus item factors and their low and high level values is available in Tables 4.2 and 4.3. Technically, the perceived value of the fruit is used specifically for the SSS-AB* algorithm simulations. However, it is included here to reenforce that the bonus item value is alter only via its perceived value.

Factor ID	Factor Description
PER_FRT	Perceived value of the bonus fruit.
FRT_FREQ	Frequency of bonus fruit occurring in steps.
FRUIT_TIME	Time the bonus item is available on screen.

Table 4.2: Summary of the bonus item factors selected for the experiment. Included is the short form name and a description of each factor.

4.3.3 Algorithm Factors

The algorithm factors are unique to each algorithm but generally provide a weight indicating the importance of a particular heuristic or rule in the decision process.

Factor ID	Low Level	High Level
PER_FRT	75	200
FRT_FREQ	50	100
FRUIT_TIME	10	20

Table 4.3: Summary of the bonus item factors selected for the experiment. Included is the short form name, a description and the low and high level values of each factor that will be assigned during the simulation.

These factors will be discussed in detail in Sections 4.4 and 4.5.

4.4 Player Algorithms

The player algorithms are composed of two types that provide a base level of interaction from which different strategies can emerge based on values of the parameters. This section introduces Pac-Man's two main algorithms: SSS-AB* and weighted heuristics.

4.4.1 SSS-AB*

As previously mentioned in Section 2.3.3, SSS-AB* is a special case of the minimax algorithm. This algorithm was selected for our experiment to investigate a higher level, possibly near-optimal player which is capable of surviving and obtaining a high score. The SSS-AB* algorithm minimizes the number of states visited by the minimax algorithm, which becomes important even in a small game such as Pac-Man. Our SSS-AB* algorithm utilizes a search depth parameter which indicates the number of intersection points to simulate, i.e. the number of decisions that Pac-Man has to plan ahead. If the search depth is one, Pac-Man will only simulate paths associated with the current intersection point to the next intersection point. However, if the search depth parameter is two, Pac-Man would examine successor paths from the current position, as well as connected paths from those paths. The depth of search alters the number of intersection points to search and plan in advance. Initially, we allowed a search depth of two; however, we eventually focused on a search depth of one after comparing the minimal improvement in performance to the significant increase in the

run-time move selection and simulation time. The increase in the length of run-time for a search depth of two was deemed unacceptable for our time requirements, and certainly beyond any real-time requirements.

Pac-Man's vision range may not allow the SSS-AB* algorithm to fully simulate all the results over the set of possible successor paths in the set search depth. In this scenario, the algorithm will return a score that Pac-Man can achieve given a limited view. In a similar situation Pac-Man may be unable to view all ghosts. Ghosts outside the view range are pruned from the search space to optimize performance which will result in a vision-based optimal solution.

A number of pruning techniques are utilized to diminish the size of the search space. As discussed above, any ghosts not visible from Pac-Man's current position are excluded. Ghosts occupying the same board position are counted only as a single ghost. If ghosts are on the same path and heading in the same direction, the trailing ghosts are removed from the search space. Finally, any ghosts whose positions make it impossible to reach the set of possible Pac-Man paths are also excluded from the search. This scenario eliminates ghosts that are initially visible but move outside Pac-Man's vision range.

SSS-AB* begins with Pac-Man using a heuristic to sort the available successor moves based on their point values, each successor move will be examined in order from highest to lowest point value. Once Pac-Man has chosen a move, the moves of all agents are simulated until a decision is required, i.e. until an agent of the game reaches an intersection point. During the simulation steps the pruning methods discussed above pruned any ghosts determined to be uninfluential for the next set of available moves. The simulation step may end as a result of specific actions in the game such as the player dying or completing the level or reaching the maximum number of simulations steps or the maximum depth of search has been achieved. When the next decision is required, the acting agent may be either Pac-Man or one of the ghosts. If the agent is a ghost, the set of available moves is considered in sorted order based on a heuristic of proximity to Pac-Man. Ghost vision parameters are independent of Pac-Man's vision parameters. These parameter values are unknown

to Pac-Man. As such, Pac-Man must assume the ghosts have full vision of the board and will play optimally given their vision of the board. Assuming all ghosts are alive and visible, the general progression of the SSS-AB* algorithm is to examine Pac-Man's first move then each of the ghost's first response candidate moves.

The search space is further reduced by allowing Pac-Man to investigate a limited number of paths. There is no upper bound the number of paths the ghost may search. This restriction limits Pac-Man from planning too far ahead and utilizing too much computational time. This results in Pac-Man's planning space being variable length, the length can be a value in the interval of 2 to 10 squares ahead. Minimizing the search space via depth reduction has two implications. First, the player is no longer optimal and second, the value of potential moves must be estimated utilizing a heuristic. The heuristic we developed uses the score the player was capable of achieving during the current set of simulated moves, plus a score based on the distance to the closest visible item. This heuristic focuses on the results from the simulation with a small increment for positioning towards other edible items. The increment is a normalized value between [0,1] to aid in tie-breaking scenarios. The heuristic may return scores for other game results. If the player died during the turn a high negative value would be added to the heuristic score. Similarly, a bonus value would be added to the heuristic score for completing the level.

As previously mentioned the testbed game Pac-Man was in part selected due its use in related research. Some of the most influential research are [18, 50, 55, 53]. The algorithms selected for the player's strategy are influenced from their research. The SSS-AB* algorithm was selected to represent a high level player who makes optimal or near optimal decisions. In comparison decision theory has been used to maximize a utility function [18]. Pac-Man was capable of planning up to five moves in advance with full vision of the board. We decided to use an adversarial search, as we were not focusing on optimizing Pac-Man's behavior but simply wanted a high level of game play. The algorithm introduced in the next section is a combination of new rules and rules that have proved successful in [50, 53].

A brief summary of the factor selected for the SSS-AB* algorithm is available in

Table 4.3.2.

Factor ID	Factor Description	Low Level	High Level
PER_FRT	The perceived value of the bonus fruit.	75	200

Table 4.4: Summary of the SSS-AB* algorithm factor selected for the experiment. Included is the short form name, a description and the low and high level values for each factor that will be assigned during the simulation.

4.4.2 Pac-Man Weighted (PW) Heuristics

The Pac-Man algorithm introduced in this section utilizes a summation of weighted heuristics to determine the next move of Pac-Man. Each heuristic calculates information about the world that the agent is capable of viewing and produces a score for that observation. Based on a sum of these heuristics a resulting score will be produced. A cumulative score will be produced from the heuristic scores for each of the four possible directions the agent could move, the position with the highest cumulative score is selected as the next move. Each heuristic has an associated weight this number represents its priority or influence for a particular heuristic. Each heuristic has its own weight. In the case of a tie, a random direction is chosen from the highest tied results. Throughout the course of this algorithm we will discuss the use of distance. In our testbed distance was measured utilizing the Manhattan distance function [38]. The Manhattan distance indicates the minimal number of squares used to traverse between two positions. The Manhattan distance improves the estimates of movement distance over Euclidean distance as Pac-Man and the ghosts can not move diagonally. Pac-Man's heuristics are can be organized into to three categories; edible goals, avoiding ghosts and global positioning.

Pac-Man's edible goals are based on the distance to the following set of game objects: tokens, power-pellets, fruits and any edible ghosts. Each distance function will be weighted to specify the goal's importance to the game strategy. As tokens and power-pellets must be collected to progress past a level they represent an essential role in the game. As such we expect tokens and power-pellets to be important whether prioritized or not. Our interest in prioritizing these objects was to observe possible

scenarios where a player would attempt to completely clear an area before moving to the next area or would prioritize power-pellets and move quickly between them leaving a large number of tokens behind. Pac-Man's accomplishments of eating fruits or ghosts are rewarded in the form of extra points. In addition, Pac-Man eating ghosts provides a strategic advantage as well, as the ghosts are temporarily removed from the game board. The bonus fruit can be placed anywhere on the game board causing the player additional difficulty to pursue the extra points. Weighting heuristics for each of these four game elements potentially represents different player strategies. When the weight for tokens and power-pellets is high the player is focused on completing the level. A high fruit weight represents a player who is seeking a higher level of challenge from the game. If a weight for power-pellets is high, yet the token and edible ghost weights are low, this scenario may highlight a player who is struggling with the game and attempting to only use the predator mode to collect tokens.

Pac-Man must avoid ghosts while playing the game, thus a distance and direction function is used for each ghost. The weights for these functions represent the player's comfort level for approaching ghosts. The highest level weights represent a player unwilling to head in the direction of ghosts unless absolutely necessary. A low level potentially represents a player less concerned with the close proximity of the ghosts positioning.

The final section introduces heuristic for Pac-Man's global positioning or at least positioning based on the visible board. There are two global positioning heuristics, one which keeps Pac-Man away from the centroid of the ghosts, and another which moves Pac-Man towards the centroid of the remaining items. For each of these heuristics, centroids are calculated based on visibility.

The weighted heuristics algorithm was inspired from the work of Yannakakis and Hallam [53] and later modified due to research results of Szita and Lorincz [50]. Yannakakis and Hallam utilized a greedy algorithm for Pac-Man's strategy. In their version of Pac-Man, each square was given a value, such as 0 for squares occupied by tokens, 10 for empty squares and 100 for squares occupied by ghosts. Pac-Man would choose squares which minimized the value of his next move. After initial trials

to produce a base level of performance, they included two additional rules. The first improved global token consumption by moving towards the closest token if all neighboring squares were empty. As well, they included an additional ghost avoidance rule to help avoid traveling in the direction of visible ghosts. There were a couple of issues not addressed within the Yannakakis and Hallam's version of Pac-Man which needed to be addressed in our version of the game. Their version of Pac-Man did not include power-pellets or the bonus fruits as they deemed those items to be less important to the level of player interest. Thus our version of the game includes additional heuristics to account for different values for eating tokens, power-pellets and fruits. Also our version allows for Pac-Man to become a predator which means that moving towards ghosts can be beneficial to both strategy and score.

Szita and Lorincz [50] created a list of action modules for the creation of rules within their version of Pac-Man. Action modules are actions which will become prioritized as a result of observations made by the player during play. For our version we are not attempting to learn while simulating, but simply need the actions and an approximation function for evaluating the effectiveness of performing each action at a specific point in the game. From their results we utilized the actions from the two most successful learned policies. The actions included are: moving towards and away from power-pellets depending on whether Pac-Man is a predator, moving towards edible ghosts, moving towards the center of items and moving away from the center of predator ghosts.

A brief summary for the factors selected for the Pac-Man weighted (PW) algorithm are available in Tables 4.5 and 4.6.

4.5 Ghost Algorithms

This section will introduce two main algorithms flocking and weighted heuristics. The ghost's algorithms provide a template for the decision process and each algorithm factor provides the opportunity to customize the behavior. The weighted heuristics algorithm performs in similar fashion to the algorithm described in the player section

Factor ID	Factor Description
PW_FRUIT_FOR	The influence of the fruits position.
PW_TOK_FOR	The influence of the closest tokens position.
PW_PP_FOR	The influence of the closest power-pellet.
PW_EDIBLE_GH	Influence of prey ghosts.
PW_BADGH	Influence of the predator ghosts.
PW_BADGH_CTR	Influence of predator ghosts centroid position.
PW_ITEM_CTR	Influence of the centroid of all items.

Table 4.5: Summary of the PW algorithm factor selected for the experiment. Included is the short form name and a description of each factor.

Factor ID	Low Level	High Level
PW_FRUIT_FOR	0.5	1.0
PW_TOK_FOR	0.5	1.0
PW_PP_FOR	0.5	1.0
PW_EDIBLE_GH	0.5	1.0
PW_BADGH	0.5	1.0
PW_BADGH_CTR	0.5	1.0
PW_ITEM_CTR	0.5	1.0

Table 4.6: Summary of the PW algorithm factor selected for the experiment. Included is the short form name, the low and high level values of each factor that will be assigned during the simulation.

with different heuristics.

4.5.1 The Flocking Algorithm

As discussed in Section 2.3.1, flocking is an emergent behavior algorithm. The flocking algorithm simulates the natural group movement of flocks of birds. In the flocking algorithm each agent follows guidelines of the flock and then acts in relation to the other visible agents. The original proposal for a flocking algorithm utilized three rules that an agent would follow: cohesion, separation and alignment. These three rules are covered in depth in the background section. To reiterate the important points, cohesion is a rule that governs the group staying within a specific proximity. Separation performs the opposite function where agents focus on moving away from the group. Finally, alignment is captured by a rule which controls travel based on the group's direction. In our testbed we have utilized another common rule which

is hunger. A hunger rule is often utilized in predator-prey situations where a goal is presented for the group. The goal of the ghosts hunger rule will be to eat Pac-Man. Thus positions closer to Pac-Man's position will be given a higher score.

A flocking algorithm works by summing the normalized scores for each of the governing rules and combining the rule scores to form a final score for each of the possible moves. Each governing rule will be affected by a weighting parameter, as well as by the vision parameters. The weighting parameter will place additional emphasis on the scoring power of a rule. The vision parameters should have an important impact on the flocking algorithm as behavior is defined via visible neighbor agents and visible goals.

The flocking algorithm was selected because it simulates natural group movement especially in predator-prey relationships and is easily parametrized for adaptation. Another factor in our selection of this algorithm was that according to Yannakakis and Hallam [53] formula for interesting behavior, it should be capable of producing interesting opponent behavior. This behavior arises because our flocking algorithm focuses on producing fluid group movement as well as accomplishing the goal of eating Pac-Man, within the vision limitations of each agent. We feel that the adjustments to the vision parameters can produce group behavior which is not too competitive for the player in producing a direct attack but which enables surrounding tactics. The strength of the attack on Pac-Man will vary depending on the group's ability to move together, producing varied results in terms of length of life as well as high levels of variation in terms of ghost movement across the board. The flocking algorithm is of interest because it provides a platform for emergent behavior. One example of potentially interesting behavior occurs when only one of the ghosts is able to see Pac-Man. In this situation we will observe whether the weight levels of a single agent will be enable it to become the leader of the flock as a result of the additional goal to eat Pac-Man.

A brief summary for the factors selected for the flocking algorithm are available in Table 4.7.

Factor ID	Factor Description	Low Level	High Level
FLOCK_SEP	The separation value of the flock.	0.5	1.0
FLOCK_ALI	The alignment value of the flock.	0.5	1.0
FLOCK_COH	The cohesion value of the flock.	0.5	1.0
FLOCK_HUNGER	The flock's hunger value.	0.5	1.0

Table 4.7: Summary of the flocking algorithm factors selected for the experiment. Included is the short form name, a description and the low and high level values for each factor that will be assigned during the simulation.

4.5.2 Ghost Weighted (GW) Heuristics

The final algorithm implements a strategy of weighted heuristics which is similar in structure to the strategy proposed for Pac-Man in Section 4.4.2, but for ghosts. Similar to the choices of heuristics for Pac-Man, we utilize other research to support our choices of heuristics for the ghosts. Yannakakis and Hallam utilized three fixed ghost strategies in their experiment; random, followers and near optimal [53]. Our experiment will utilize portions of their followers and near-optimal static ghost strategies. Followers are simply ghosts which continuously chase Pac-Man, attempting to minimize the distance between themselves and Pac-Man. We include the distance to Pac-Man as an obvious contributor, but we also include a second calculation based on Pac-Man's current direction. Their near-optimal strategy includes an additional force in which ghosts are repulsed by other ghosts. This repulsion forces ghosts to spread out unless extremely close to Pac-Man. The inclusion of a repulsion calculation based on distance to other ghosts has been selected not only because of its success in their research but due also to the similarity of the separation rule in the flocking algorithm.

Similar to the player strategy heuristics, the ghost weighted heuristics include some heuristics from the research of Szita and Lorincz [50]. The Szita and Lorincz research focuses on optimizing rules for Pac-Man's play not ghost strategies, so we have adapted their rules to be used for ghost behavior. From their research we have decided to use a heuristic to award points based on increasing the distance from the centroid of visible ghosts. We have also selected a heuristic to award points based on moving towards the centroid of the remaining edible items. In addition, we have included heuristics similar to those for Pac-Man. They include distance functions

and weights for each edible item in the game: token, power-pellet, and fruit. Our belief is that including weights for each item for each ghost, allows ghosts to prioritize protection of each item. Thus, we could produce ghosts which had a higher priority to protect power-pellets over tokens. Likewise, if the player always goes for the fruit, we could include ghosts which chose to protect bonus items over the regular items for level progression. Finally, we included heuristic for moving towards the end of Pac-Man's current path given Pac-Man's current direction.

Each of these heuristics is greatly influenced by the vision of the ghost. If a ghost is unable to see Pac-Man, it will default to a protection setting, moving towards the closest game object with the highest priority without moving close to other visible ghosts. If a ghost is unable to see either Pac-Man or any items, its play will be dependent on avoiding the same area as other ghosts. If a ghost is also unable to view others ghosts, its strategy will degenerate to a near discovery state, in which the ghost explores any path except the last path traveled in attempts to discover new information.

A brief summary for the factors selected for the ghost weighted (GW) algorithm is available in Table 4.8.

Factor ID	Factor Description
GW_TOKEN	Influence toward protecting tokens.
GW_PP	Influence toward protecting power-pellets.
GW_FT	Influence toward protecting the fruit.
GW_PAC	Influence towards Pac-Man's position.
GW_PAC_DIR	Influence towards Pac-Man's next position.
GW_AWAY_GH	Influence away from other ghosts.
GW_TO_GH	Influence toward other ghosts.

Table 4.8: Summary of the GW algorithm factor selected for the experiment. Included is the short form name and a description of each factor.

4.6 Performance Measures

Pac-Man's performance is measured to evaluate the level of difficulty each strategy is having with the adaptations to game factors. All analysis will be performed after the

Factor ID	Low Level	High Level
GW_TOKEN	0.5	1.0
GW_PP	0.5	1.0
GW_FT	0.5	1.0
GW_PAC	0.5	1.0
GW_PAC_DIR	0.5	1.0
GW_AWAY_GH	0.5	1.0
GW_TO_GH	0.5	1.0

Table 4.9: Summary of the GW algorithm factor selected for the experiment. Included is the short form name, the low and high level values of each factor that will be assigned during the simulation.

games have been simulated. We have devised several measures which are intended to identify Pac-Man's level of task difficulty. These measures are play a role in the a number of the heuristics created for the algorithms, discussed in detail in Section 6.3.

4.6.1 Proactive Measures

Pac-Man's performance will be measured to evaluate the level of difficulty each Pac-Man strategy is having with the adaptations to game factors. Proactive heuristics identify situations where Pac-Man is struggling to progress in the game prior to failure. In the first part of our experiment all analysis of the proactive heuristics will be performed retroactively, after the games have been simulated to calculate effects of factors. During our experiment and analysis phases we will use the statistical analysis terminology "response variables" when referring to the collected results of either proactive or reactive measures. During the development of the adaptive game in the second part of the experiment we will reuse a selection of these retroactive measurements to proactively estimate the future performance of the player. In this section we preview several measures which could contribute to the modeling of the level of difficulty for the player. Our focus in this section is to highlight factors which potentially affect score. We feel score is one of the most important response variables in the experiment and is the easiest to discuss in relation to actions and consequence. This experiment will investigate the following proactive measures:

- Close calls occur when a ghost has come within 2 squares of Pac-Man.

- The repetition of squares by Pac-Man that contain no point value.

- The number of steps since Pac-Man has collected an object, such as a token, a power pellet, a fruit or a ghost.

- The number of power pellets remaining on the board.

The number of close calls measure identifies situations in which ghosts are close to catching Pac-Man but have not been able to accomplish the task. This variable could provide insight on how to adapt gameplay when the player has produced a high score. A high score indicates a situation in which the level of challenge should be altered. If the number of close calls is low, the player is likely feeling unchallenged and a larger alteration could be made, whereas a high close calls value, indicates the level of challenge may not need to be changed or only slightly increased due to the potential contribution of luck. The number of close calls will be normalized via the number of steps per life to avoid situations where one player would have played longer than another. A summary of the expected effects for the close calls measure is listed in Table 4.10. When combined with player score, it should provide a reasonable base estimation for the perceived level of challenge and player's experience.

Tables presented in this section demonstrate the possible extreme values of the proactive measures, and the player's score. Although the player's excitement would be distinct for each player, and cannot be accurately assessed given only these measures, we have included the level of excitement as a likely base emotional state towards which the player's experience is being directed by the current level of difficulty. An important note in terms of situations for adaptation is that providing a medium challenge or matching challenge and excitement states may require additional observations before adapting the game to account for the possibility of luck within the game.

		Close Calls: Low	Close Calls: High
Score	Low	High Challenge High Frustration	High Challenge Possible Frustration
	High	Low Challenge Medium Excitement	High Challenge High Excitement

Table 4.10: Potential player experiences relating to the response variable close calls and score

The repetition of squares which contain no point value and the number of steps since Pac-Man collected an object indicate situations where Pac-Man is failing to progress in the game. The repetition of squares which contain no point value indicates that the player either does not understand how to accomplish the goals in the game or is struggling to get to the desired location as a result of being chased by ghosts.

		Repetition of Squares: Low	Repetition of Squares: High
Score	Low	High Challenge Possible Frustration	High Challenge Possible Frustration
	High	Low Challenge Medium Excitement	High Challenge High Excitement

Table 4.11: Potential player experience comparing the response variables repeated number of squares and score

Likewise the number of steps since Pac-Man has collected an object indicates a failure to obtain points during the limited number of moves. This can caused by Pac-Man being redirected from goals via ghosts' actions or failed attempts to obtain bonus items. As the level progresses, a greater number of squares which have no point value will become available, as Pac-Man has already obtained the points from those squares. Thus Pac-Man will have to traverse a larger number of squares to obtain the remaining tokens. This will increase both the repetition of squares and the number

of steps since the last completed task. The two measures, repetition of squares and moves without eating, were selected because they both directly affect the player's overall score and the number of tokens collected.

		Time since goal accomplished	
		Low	High
Score	Low	Medium-High Challenge	High Challenge Possible Frustration
	High	Low Challenge Medium Excitement	High Challenge High Excitement

Table 4.12: Potential player experience reviewing time since goal accomplished and score.

The number of remaining power-pellets could be an important measure due to the fact that eating a power-pellet alters the ghost strategies. The presence of power-pellets allows Pac-Man to turn the ghosts into prey which allows the player to obtain more points for eating the ghosts. In addition, it allows Pac-Man to protected for a short period of time and get out of dangerous positions. The presence of power-pellets alters the optimal strategy of the ghosts, in that the ghosts should not converge when Pac-Man is closer to a power-pellet. Likewise they should disperse as quickly as possible as prey when in close proximity to Pac-Man to avoid being collectively eaten. The presence of power-pellets allows Pac-Man to use a wider set of skills when no power-pellets are present. If no power-pellets remain on the board Pac-Man must avoid ghosts purely on skill, but when present, power-pellets offer the opportunity to be more strategic by setting traps or attacking dangerous areas of the board.

4.6.2 Reactive Measures

Reactive heuristics indicate the level of success the player had in collecting items and staying alive. The following reactive measures were used to evaluate the player's performance and to gauge the level of difficulty had during game play. It is important to note that in the context of adaptive gaming we will use the term reactive measures.

However, during our experiment and analysis phases we will use the statistical analysis terminology "response variables" when referring to the collected results of either proactive or reactive measures. The reactive measures are:

- the number of steps
- the number of tokens, power-pellets, and fruit collected per life
- the number of ghosts eaten.
- the number of levels completed
- the overall score of the player

The number of steps can be utilized as a measure in two ways. If Pac-Man does not complete the tasks with a low number of steps, the difficultly may be too high, whereas if Pac-Man completes the tasks with a low number of steps, the task may be too easy. Likewise, a high number of steps with a high score may indicate that the player's performance is balanced with the current level of challenge, which may require small tweaks or no change at all. In this case the player accomplished a high number of tasks, plus possibly bonus tasks or was heavily chased but still managed to avoid ghosts.

The number of tokens that Pac-Man collects during a turn indicates progression toward level completion. However, for each successive life, Pac-Man has fewer tokens to collect on the board, unless a level is completed. The mean number of tokens collected was used in Yannakakis' interest formula [53]. When the mean number of tokens collected varies, this variation indicates that the player is experiencing different actions in game play, which may mean the player is unable to learn a single strategy to complete a level. Thus, Pac-Man should be capable of obtaining a relatively equal portion of available tokens each game. Slight deviations are expected for each turn in the number of items collected, thus a general behavior must be observed as an indication of the level of challenge. Likewise, the number of fruits obtained indicates that Pac-Man is capable of achieving additional game goals. Fruits are valuable

items obtained during game play. However, obtaining this bonus item will require backtracking and delaying the goal of level completion. Finally, the overall score of the player indicates whether he was successful in progressing in the game. This heuristic is often used in other research [25, 53], especially those utilizing machine learning to model a NPC's behavior.

4.7 The Adaptive System

4.7.1 Overview

The final phase of our research will use the information gained in the factorial analysis stage of our experiment to develop an adaptive game. In Section 3.2, we noted that an adaptive system requires a feedback loop which performs the following: estimates the player's progress, identifies a required level of change, and adapts factors of the game to meet that requested level of change. Our adaptive system is composed of three main components: heuristics, player modeling and system management. The heuristics predict the player's performance for the values of the response variables based on recent gameplay and the player model. The player model stores the information acquired about a particular player during the simulation phase. Player model information includes: the effect of each factor setting to a particular response variable and the game statistics the player was capable of achieving per game and per life. The player model decides which factor settings will best suit the requested changes to the level of challenge. The system also manages the adaptive process, runs the heuristics, requests adjustments to the factor settings and implements the changes to the object loader and the game objects.

4.7.2 Adaptive Pac-Man

Our adaptive system is a feedback loop which functions by allowing opportunity for adjustment every five moves. Whether adjustments are necessary or not is determined by the heuristics. The system begins with the researcher specifying the desired target

interval for one or more response variables, this provides a target upper and lower bound for the adaptation progress to direct toward. We created a unique heuristic to estimate the potential performance in a response variable over each life. The key to an effective adaptive game system, lies in its ability to accurately predict the player's performance and progress. Any inaccurate information or estimates could cause factor selection which result in disastrous changes. During gameplay the adaptive system will attempt to maintain the selected response variable in the designated zone, or have the final result finish within that zone. The main loop of the system creates estimates based on the heuristics for each response variables the researcher has chosen to control and adapt. The player model uses the estimates and the size of effect information from the factorial analysis stage to develop solutions to all the factor settings. Factor settings are the combination of several values from our analysis. They have three pieces of information: a list of factor names, an ID number between 0 and $2^k - 1$ where k is the number of factors used by the object loader to set the necessary parameter information and lastly the total effect size which sums the combination of all active terms effect size in the current setting. The resulting factor settings will have an effect value equivalent or close to the difference between the current estimated result and the predefined interval.

The previous phase of the experiment calculated each response variable for all the factor settings and sorted the results based on the effect size, which allows quick retrieval for adaptation. Factor settings are first selected by minimizing the difference between the potential effect sizes and required change size. Once a set of suitable factor settings are selected, factor settings which reduce the number of modifications required to the current factor settings are given priority for selection. Selecting factor settings which minimize the number of alterations is important for two reasons. One, it minimizes the number of objects to modify, which results computational requires less work and two, it decreases the chance of a drastic change in gameplay which may greatly alter the difficulty and draw unwanted attention from the user. Finally, the system must modify the game objects with their new factor settings, which is greatly simplified by our initial factor loading system. This process continues throughout the

life of Pac-Man to form the feedback loop for the adaptive game.

Utilizing the statistical tools built for the factorial analysis stage of the experiment, the researcher can collect the results of the adaptive simulated game. The researcher will analyze the collected data to identify the situations in which the adaptive system has improved the number of results within the specified target intervals. Although we are manually setting the target intervals for response variables to be adapted, that information could easily be coming from another system which could be controlling the rate of adaptation or desired level of challenge to produce a true adaptive system. We believe controlling and adapting the game based on a very simple set of heuristics and adaptive system setup demonstrates that this methodology provides a positive step towards building a complete adaptive gaming system.

4.7.3 Heuristics

Pac-Man is frequently used as a testbed for game and artificial intelligence research due to its simple interface, yet complex and emergent interactions. Despite the simple interface, it can be a highly involved process to produce effective heuristics for Pac-Man that estimate the future performance or the current struggles of the player. The challenging part of creating an effective heuristic to estimate different response variables in Pac-Man boils down to two major issues. The first issue is that Pac-Man has only 1 health unit, as soon as a ghost occupies the same square, Pac-Man's life has ended. This requires identifying life threatening situations before they occur and while Pac-Man still maintains the opportunity to escape, which necessitates having a good player model as well as effectively evaluating the strategic position of the board. The second issue is that scoring progression is non-linear. The player's life can progress without scoring additional points and this situation may or may not reflect any additional difficulty for the player. This situation could be explained by attempting to collect a fruit, to backtrack for a missed area or attempting to avoid chasing ghosts.

4.7.4 Limitations

As a large amount of additional work is needed to develop a fully adaptive system, we limited our adaptation system in several important ways. In our proof of concept, we did not focus on the rate of alterations. Instead, we set a constant time interval at which the adaptive game could check whether the game required adjustment. Ideally, an adaptive game system would gradually increase or decrease the difficulty in subtle ways over a period of time. Subtlety is not a major issue for our users in our research, as we have no human players. However, the downside to the lack of subtlety is that since player progression estimates are calculated based on heuristics, the amount of adjustment requested can be quite different from one estimate to the next. This can lead to large swings in factor levels, that may cause inaccurate or irreversible results. We allowed the adaptive system to alter as many factors as it deemed necessary, although the adaptive system places an emphasis on choosing factors settings with minimal difference from the current settings, after a set of possible solutions is produced.

Additionally, we limited the factor settings to values we had previously experimented with in the other experimental stages. Our experiment allowed each factor to have 2 levels, a high and low value. These were the only acceptable settings during the adaptive system testing. This simplification allowed us to avoid interpolating the effects of factors and the response variables. We consider this simplification to be a similar methodology to what a commercial tester might use when a game cannot be shipped with potentially unseen behavior. As each experiment has 10 factors, this limits choices to 2^{10} or 1024 possible factor settings.

A goal of dynamic difficulty systems should be to create interesting behavior, in addition to appropriately matching the player's ability to the level of challenge of the game. Creating interesting behavior was not a focus of our prototype, as we simply attempted to control the result of the response variables. Adding functionality to create this behavior would be relatively simple although game specific. We could define a function that selects factor settings based on a function of interesting or

diverse behavior in combination with minimizing the distance between factor settings. This functionality could also be added by including interesting or diverse behavior as one of the response variables which are calculated in the analysis phase.

Chapter 5

Experiments

The experiment has been designed with two main goals: the initial goal is to investigate a methodology for identify significant factors relating to the level of challenge of the game. The secondary goal is to test and develop an intermediate step progressing toward the larger goal of a fully functional auto-dynamic difficulty system in video games. As previously stated, the main component of auto-dynamic difficulty is to automatically adjust the level of challenge to the player's level of skill. To determine the full effect of a game factor and its significance for the response variables, we must view the effects of that factor in isolation, as well as in relation to other factors to discover possible emergent effects. Different player strategies will be affected by the game factors in different ways, thus we must explore each game factor's effect in relation to a number of player strategies. Using a number of different player strategies increases the potential of representing different players types.

Section 5.1 provides a brief introduction to factorial analysis, which is our evaluation method for identifying factors with significant effects on the results of the response variables. Section 5.2 provides a detailed view of how the experiment was specifically designed for the Pac-Man game. In addition, this section introduces the methods of analysis utilized to calculate the results and provides an overview of the purpose of each method. Section 5.3 observes the results of the analysis sections, a summary of the experiment is provided in Section 5.4.

5.1 Factorial Design

The analysis for the results of the factorial experiment identifies factors and factor interactions which significantly affect the response variables or explain a portion of

the variation of a response variable. A response variable is a reactive measure or a collected result in response to an activity during the experiment. In our research our response variables are performance measures, listed in Section 4.6. For these simulations, we will utilize a full 2^k factorial design, to identify the variation created by each factor level. The factorial design will allow us to compare factors and to rank them based on their impact on the set of response variables. Ranking the factors provides a priority guideline for which game parameters could be altered based on the needed level of difficulty.

We have selected a 2^k factorial design, because it will determine the effect of k factors also known as main effects, each of which has two values in the experiment also known as levels. The levels of a factor are coded to values of -1 indicating the low level and 1 indicating the high level. The 2^k analysis is a simple theoretical analysis method that can provide human readable results, and allow factors or the interaction terms of multiple factors to be estimated. An additional property of the 2^k analysis is that factors can be added to the model of the analysis without the need to rerun previous games setting, only running required games with the new value. Determining the effect of each factor will allow us to order the factors based on their statistical significance. To evaluate an experiment with k factors at 2 levels, we must perform 2^k experiments. Interacting factors, are factors in which the resulting effect of one factor is dependent on the value of the other factor. The experiment will produce $\binom{k}{n}$ n-factor interactions, for example it will produce k main effects for each factor, and $\binom{k}{n}$ 2-factor interactions, etc.

To demonstrate some of the properties of 2^k factorial design, for simplicity we provide an example from [27] for a 2^2 factorial design, the ideas presented here and in Table 5.1 directly extend to the 2^k experiment. The 2^k design works by creating a signed table for all of the factors and all combinations of interactions. In the case of 2^2, the two main factors A and B will be considered. We have included the identity column (I), as well as a column for the interaction between A and B shown as column AB. Since there is only two possible values or levels for a factor we represent the low level as -1 and the high as 1. The AB value for each row is calculated by multiplying

the row values of A and B together. In an example with three factors the interaction ABC would be the combination of the column values A*B*C from that particular row. The Y column represents our response variable from the experiment. Each row represents a run of the experiment where A and B use coded values and Y is the result of a response from that run. In later sections we will refer to the ideas of a signed table and an interaction table. A signed table will refer to the portion of the table with main factors A and B. An interaction table will refer to the interactions of those factors such as the AB column.

I	A	B	AB	Y
1	-1	-1	1	15
1	1	-1	-1	45
1	-1	1	-1	25
1	1	1	1	75
160	80	40	20	Total
40	20	10	5	Total/4

Table 5.1: Example 2^2 Design Experiment Table [27]

The total for each column is calculated by summing the factor values of each row multiplied by their Y column, also known as the dot product of the two column vectors. For example column A's total is 80 which is calculated by $A \cdot Y$ or $(-1 * 15) + (1 * 45) + (-1 * 25) + (1 * 75) = 80$ [27]. The total values are divided by 2^k, in this case 4 to produce the coefficients for a regression equation. The column titled I produces the coefficient of the regression equation. In this example the regression equation would be $\widehat{y} = 40 + 20x_a + 10x_b + 5x_a x_b$, where x_a and x_b represent possible row values of the A and B column respectively. The symbol $\widehat{y}$ indicates that a model is fitted. the A regression equation is an equation that models the relationship between variables. Regression analysis is used to find dependent and independent relationships between variables. An important property is that the signed and interaction tables can be reused for analysis that have the same number of factors. Using this property we reused an interaction table with 10 factors in our experiment. Another important property is that coefficients of the regression equation can be calculated independently, using the columns of the interaction table.

5.1.1 Calculating Variation

To identify the importance of each factor, we must measure the variation of a factor in relation to the total variation. The total variation, also known as the Sum of Squares Total (SST) for the 2^k experiments, can be calculated in the example by the formula $SST = 2^k(q_a^2 + q_b^2 + q_{ab}^2)$, where q_x refers to the coefficients of the xth column from the regression equation retrieved from the design table [27]. If we would like to calculate the variation for a single factor we can do so via the following formula $X = \frac{SSX}{SST}$, where SSX is the variation for any factor, and is calculated by $2^k * q_x^2$ [27].

Variation indicates the ability and force of a factor to modify a response variable of a game. It is important to identify the variation as it indicates how important the change in the game could be. In addition, it provides the adaptive system with a range of values to focus on for selection in the adaptive process. The perceived level of required adaptation is an estimate of the player's needs, estimated by the adaptive system. If significant adaptation is required, the adaptation process will investigate factors with higher variation values first. As the perceived level of required adaptation shrinks, factors with smaller variation will be adapted to tune the game more precisely.

5.1.2 $2^k r$ Factorial Design with Replications

To stimulate player's interest the game should provide slight variation in gameplay. Thus the same actions should not occur in the same order every time the player plays the game. Gameplay will fluctuate as a result of differences in player tuning their strategy and the presence of luck. Since variation should occur naturally in gameplay, we need to account for this by repeating the experiment with the same factor level to produce means of observed effects. Performing replications also provides the additional benefit of being able to calculate an error term for the experiment as well as confidence intervals.

The $2^k r$ factorial design is comparable to 2^k factorial design with r replications, the addition of replication means the performance measures are now calculated as a mean of all games played with the same factor levels [27]. In our experimentation

r was set to a value of 3, that is each simulation ran 3 times with the same set of factor values. Each run included a single Pac-Man game containing three lives. The error term is calculated as the sum of squared errors (SSE), which can be used to estimate variance and compute confidence intervals. The measured response value is the observed results of the experimentation process or the resulting performance measure, represented by the value Y_i, where i represents the i th replication. The estimated response is the mean or average of the Y_i values. SSE is calculated by the formula $SSE = \sum_{i=1}^{2^k} \sum_{j=1}^{r} e_{ij}^2$ [27]. The error term e_{ij} is calculated as the measured response value minus the mean response. The formula for SST is similar to 2^k factorial design, but now includes theSSE term. The inclusion of the error term SST for $2^k r$ is given by $SST = 2^k r(q_a^2 + q_b^2 + q_{ab}^2 + ...) + SSE$ [27].

5.1.3 Calculating Lack of Fit

As the number of factors in the experiment increases, the number of interaction terms increases combinatorially. However, most high level interactions are unlikely to be significant factors. Additionally, our model will be limited to containing a maximum of 127 terms. Thus, our analysis requires an additional test to ensure the terms excluded from the model are not contributing significantly to the results of the experiment. This information is calculated using the lack-of-fit test and the pure error term. A full 3-factor model is given by $SST = 2^k r(q_a^2+q_b^2+q_c^2+q_{ab}^2+q_{ac}^2+q_{bc}^2+q_{abc}^2)$ [27]. To calculate the lack-of-fit, one or more terms must be excluded from the model; we can exclude any term except for the main effects. As an example we could remove all 2-factor interactions resulting in the model estimate equation $SST = 2^k r(q_a^2 + q_b^2 + q_c^2 + q_{abc}^2)$. After calculating the q_x coefficient values the new model equation estimates the value of the response value Y, given the experiment input values a, b, c; this estimate is known as the fit value. The residual is the difference between the actual experiment run value of Y and the estimate of $\widehat{Y}$. The residual error can be calculated by taking the sum of squares of the residuals ($SSRE$). The Lack of Fit value is calculated by $SSLOF = r \sum_{i=1}^{2^k} (\widehat{Y} - \bar{Y})$ [27], where $\widehat{Y}$ is the fitted value and $\bar{Y}$ is the mean of the experiment runs. The pure error can be calculated by $SSRE - SSLOF$, and we use

pure error sum of squares ($SSPE$) to calculate the significance of the lack-of-fit.

The statistical significance of the lack-of-fit value is calculated by first calculating the F-value and then looking up the P-Value or significance. The F-Value is calculated by $F = \frac{SSLOF/DnF}{SSPE/DdF}$, where DnF is degrees of freedom of the numerator and DdF is the degrees of freedom of the denominator. The DnF is the number of terms excluded from from the model; in our example DnF is 3. The DdF is the degrees of freedom of the denominator $(r-1)*2^k$.

5.1.4 Factorial Analysis and Experiment Design

Initially, our evaluation of the experiment intended to do a complete factorial analysis of both Pac-Man and ghost factors simultaneously. Ideally, having both sets of factors in the analysis would provide insight into how to improve the tactics of both the player and opponents. However, this led to high k values (maximum of 20) which could not be analyzed utilizing commercial statistical programs such as SPSS or Minitab due to memory limitations. The commercial software was capable of doing a full factorial analysis for a maximum of 8 factors, which was well below our desired interval. To compensate for this large amount of factors we began investigating other methods of analysis based on fractional factorial designs.

The first issue of fractional designs for our experiment was confounding, in which the value of some of the effects cannot be determined, only the combination of their influence can be. This was potentially a large issue for our experiment as we required the knowledge of how each factor effected the game. The intent for an adaptive system is to keep changes minimal and subtle, the factors can not be separted we risk drawing attention to the adaptive system because we must change a higher number of factors, it also increases the risk of altering an unintended aspect of the game. With confounding of factors we potentially diminish the ability of the adaptive system to perform more detailed adjustments. The second issue of using fractional designs for our experiment was that fractional designs are based on the assumption that higher order interactions have small effects. However, our algorithms were specifically selected with the intention of producing higher order interactions and emergent behav-

ior. Intuitively we suspected and then later observed in early runs of our experiment that the algorithms we selected based on producing emergent behavior would produce higher order interactions with large effect values. A final reason why fractional designs were not well suited for our study was that the advantage of that method is removing a non-significant factors during the early stage, although this method could be used to identify insignificant factors to be removed our intention was to retain all factors throughout the process in case they were significant to other factors settings or players.

Ultimately, the inability to use either large values with a full factorial or fractional factorial design led to the restructuring of the problem space to create a more manageable size. Thus we restructured the problem by dividing the players factors and the rest of the game factors into subsets, thus creating groups with manageable factor sizes. We separated each algorithm set into groups of 10 factors, which was still above the commercial term limit of 8. However, by exploiting the properties of full factorial design, we were capable of developing our own program that would initially allow us to investigate up to 15 factors. The separation of factorial design is one of the use properties we utilized to perform our analysis. For instance, if the design had 12 factors the separation of the design to 10 factors would create 2^2 or 4 cases. In these 4 cases, the values of the 2 separated factors are implicitly defined in the model. Table 5.2 illustrates the implied values of the 11th and 12th factor in each case. In this example, each case produces its own model, which results if 4 models and every term having 4 values. The limitation of separating the design is that we do not have access to the intersection values of the separated factors. Thus the main effects and any interactions between factors and factor 11 or 12 are unavailable, unless the design is reconstructed.

We selected to use groups of 10 factors for several reasons, the first being that our initial factorial analysis with higher factors values showed very low R-Sq results. This was partial due to a limitation of the commercial software which limits the model to a maximum of 127 terms. Secondly, grouping to 10 factors drastically decreased the computational time required to perform the analysis. The decreased

	Factor 11	Factor 12
Case 1	-1	-1
Case 2	1	-1
Case 3	-1	1
Case 4	1	1

Table 5.2: Example of separating the factorial design, the values of factor 11 and 12 are implicit applied to the model's coefficient and terms.

results of the R-Sq values was expected because as the number of terms increases the commercial limitation of 127 factors becomes a greater constraint, as additional significant effects may occur outside the top 127 terms. Our program utilized the fact that terms effects can be independently evaluated using the effects table and the response variable. Using this property we avoided the expensive computation memory problems at the expense of computational speed. The effects and sum of squares (SS) where independently calculated then recombined and sorted to define an ordered list of the terms with the greatest effect on a specified response variable. We loaded the ordered terms into our commercial software (Minitab) for this project to calculate the significance for the terms and R-Sq values for the model. At this stage, we encountered another limitation of the commercial software, Minitab could only included a maximum of 127 terms per model. Although our experience with SPSS allowed a model to load above 200 terms, even performing analysis on models with 127 terms in SPSS took significantly longer. This limitation occurs only for the commercial software as we are capable of including all terms into the adaptive game models. However, despite the limitation of 127 terms per model we still produced adequate R-Sq values for the algorithms. Depending on the results of the lack-of-fit tests the terms excluded from the model may not to contribute significantly to the experiment results, and thus 127 term could produce adequate models.

The restructuring of the player factors created a couple of advantages on top of allowing us to complete the analysis. This organization allows each session to be treated as an individual player, and eases the creation of models for the adaptive models of each algorithm. The factor separation allows for an effective method to investigate alterations to game objects or game design that are too large or noticeable

to be included in the adaptation process, such as level design changes or attributes that are viewable to the player. The separation also allows us to organize and observe player information in a similar fashion to how it would be received in an online setting, which is a potential logical progression of this research. In addition this organizational system is better suited for extracting player information, which will be demonstrated in the proof of concept adaptive system.

5.2 Experimental Environment

The experiment was designed to ease modification and loading of any of the parameters of the Pac-Man game. In addition the game loaded a number of experiment arguments when the game was first run, which included information such as the level to load, the maximum number of simulation steps, the number of ghosts to use and the ID of the game to run. The ID of the game indicated which dynamic and attribute files to load, such that each simulation could run independently and simultaneously. Given the ID of a game all the information was calculated or retrieved to simulate the game with all the correct factor information. Each experiment run would simulate Pac-Man games with three lives, and use a repetition value of three. Throughout the course of the experiment runs every action in the game would be logged in an XML format for post-game analysis.

5.2.1 Response Variables

The first step upon completion of the simulation portion of the experiment was to use a set of error checking tools to ensure the required number of games had finished running, the correct number of lives and the correct factor values were also checked. The next step included parsing, collecting and preparing all of the information needed from the XML files, the most important of which was to accumulate the values of the proactive and reactive performance measures. Throughout this chapter the term performance measures will be referred to by the statistical terminology response variables. This section uses the statistical terminology response variables, which specifies

the collected results of the performance measures discussed in Section 4.6. The response variables we selected for this experiment are: 1: score, 2: number of steps, 3: number of close calls, 4: number of repeated steps, 5: number of fruit created, 6: number of fruits collected, 7: number of tokens collected, 8: number of power-pellets collected, 9: number of ghosts eaten and 10: the number of levels completed. Additional information about the selection of response variables can be found in 4.6. The number of fruit created will be excluded from the final analysis as fruit are created at set intervals depending on the number of steps.

5.2.2 Factorial Analysis Customizations

Once all the information was collected we organized specific experimental runs into groups to be able to proceed in the evaluation process for the restructuring reasons discussed in Section 5.1.4. Once the data were separated and organized we began preparing for the factorial analysis stage, which was partially customized for the large number of factors in our experiment.

The first step was a one time preprocessing step which calculated the interaction table for all factor interactions. The process begins with creating a full signed table from 0 to $2^k - 1$ for the 10 main factors for each experiment, except that 0 is replaced by -1 for the purpose of multiplication. Next, an interaction table is created for every possible combination of those factors, each cell in the interaction table is calculated by multiplying values from the signed table row by the column values of the term. For example if the interacting term was A*B*D*J and row 0, we would multiply columns from the signed table A by B by D and then by J the results of which are placed in column ABDJ. In row 0 all factor columns contain -1 representing 0 thus the result of the multiplication would be 1. As we organized the factorial design into groups with 10 factors the interactions table and factor table are reusable for each experiment, thus process is only completed once for the entire experiment. Part of the customization that allows an analysis on a greater number of factors than the commercial software occurs in this stage, as each column of these interaction tables and the response variables represents all the calculations required to determine

the effect and SS value of a factor on a particular response variable. This enables terms to be calculated independently reducing a computationally expensive step into smaller pieces which can be reassembled upon completion because all model terms are orthogonal to each other.

Using the interactions tables and the response variables computed in the organization portion of this experiment, we calculate the effects and sum of squares (SS) for every possible combination of factors. The coefficients for a regression equation are calculated by taking the dot product of the factor's column vector from the interaction table and the response variable column followed by dividing by the number rows in the columns which will be determined by 2^k . The effect of a term on a response variable is calculated by multiplying the coefficient of the regression equation by 2. The sum of squares for each factor is calculated by $2^k * q^2$, where q is the factor's coefficient of the regression equation. Once all the effect sizes and the SS values for each term are calculated, we sorted all the terms based on the SS values, so that the top 127 terms could be selected for use in the commercial software portion of the analysis. The phrasing "selecting the top 127 terms" can be somewhat misleading in this case, as a requirement of the software was any main effects which contributed to a significant interacting term was also required to be included in the model. As an example if the term A*B*D*J had one of the highest SS values, all four ID's A, B, D and J would need to be included in the model whether they where potentially significant or not. Main effects that were required to be in the model, replaced the lowest scoring terms from the top 127 terms provided those terms were not main effects themselves. The main effects are required by the model as their inclusion allows larger interacting terms to be constructed via the orthogonal properties of the design. After identifying the top terms to be used in the model, we structured the information in preparation for the commercial software, which included setting the design for a factorial design given the terms, and integrating a signed table for the number of factors and the response variables from the experiment runs. The factorial design calculations computed all the information in the three sections: model evaluation, terms effects and significance and comparative evaluation we have separated

them into separate sections for organizational purposes.

5.2.3 Model Evaluation

The first method of evaluation for our experiment was used as a guideline to indicate the percent of variation that our model was capable of explaining with the 127 highest sum of square values. The limitation of 127 terms came from the commercial software, but does provide information as to whether the adaptive system requires the inclusion additional terms. Calculating the R-Squared values will indicate the percentage of variance our model explains and its accuracy in predicting other data points. The second portion of the model evaluation process investigates the consequences of the the term commercial term limitation. The model evaluation process involves calculating lack-of-fit values to review the significance of the terms not included on the model, i.e. the terms not included in the 127 model terms. The lack-of-fit testing will determine whether terms excluded from the model played a significant role in the results of the experiment and ultimately whether additional terms need be included in adaptive game system.

5.2.4 Statistical Significance

This section identifies the statistical significance of terms and terms with the largest effects. The process created simply too many terms to review and discuss, thus we selected only the statistically significant main effects and 2-factor interactions to present. We will discuss the largest effects and any surprising results. The tables for this information will be available via the appendices. In the results section, we will simply summarize any observation or patterns we were able to identify. The effect sizes will be listed for all statistically significant terms but discussion of their value will be limited.

5.2.5 Game Significance

Section 5.2.4 identified the effects and statistical significance of terms included in each model; the game significance section will review these results and interpret their impact on the players' performance. This section will qualitatively investigate the model terms and will differentiate the statistically significant terms from those that provide an important impact on game play.

5.2.6 Comparative Evaluation

The comparative evaluation phase will utilize the performance results for simulated Pac-Man games and compared them to other simulations of the experiment with either the same algorithm with different factor levels or different algorithms with the same factor levels. The comparative evaluation will occur only between factor settings which are not included in the term evaluation section. As an example we could compare Pac-Man's final score while playing against the flocking algorithm, when the vision range is 3 and when the vision range is 5. This should indicate whether a factor increases or decreases the difficulty for a particular player strategy. Similarly, the performance of Pac-Man will be compared utilizing the same factor values but with different player and NPC's strategies. As an example, comparing the flocking algorithm with vision range of 5 against the SSS-AB* and the weighted heuristics algorithms. Comparing the same factor values with different strategies will identify the performance of that factor across all algorithms to identify the global effect on the difficulty of the game. This will provide evidence that a factor could provide alterations to the level of difficulty for all player types. It is our hope that the results will indicate factors which alter the difficulty at different rates for the individual player strategies, as well as finding factors which effect the difficulty for all player strategies.

5.2.7 Adaptive System

The final portion of our experiment is to implement an adaptive prototype Pac-Man game which will be introduced separately in Chapter 6. The adaptive prototype will use the results calculated in this chapter to produce a database for possible modifications to the game and their result on the gameplay experience.

5.3 Results

This section presents the results of our experiment. The results will be presented in four major sections. The first section presents the results of measuring model variance. These results, described in Section 5.3.1 were obtained as a portion of Minitab analysis of identifying factor significance. Second, Section 5.3.2 highlights the statistically significant factors and their results on each response variable. Third, Section 5.3.3 calculates ranges for game significant factors and identifies prominent factors. Next, Section 5.3.4 presents the results of the separated factors, followed by a comparative analysis of the algorithms in Section 5.3.5. Throughout this section, we will utilize a number of short hand abbreviations for the algorithms and factors, which will follow the structure of being completely capitalized with separation occurring via underscores, as an example SSS_FLOCK or PW_GW would refer to algorithms SSS-AB* playing against the flocking algorithm and Pac-Man weight heuristics against the ghost weight heuristics respectively. Factors associated with algorithms will be preceded with by a string identifying the proper algorithm, as an example PW_BADGH_CTR would indicate algorithm is the Pac-Man weighted heuristics and the factor the weight for avoiding the centroid of the ghosts.

The results from the factorial analysis phase of our experiment contained a large amount of data. We organized and assembled the data based on pairs of algorithms the first from the Pac-Man player and the second from the ghosts. This resulted in four groups of algorithms, which will be identified by the keyword in brackets, SSS-AB* and flocking (SSS_FLOCK), SSS-AB* and ghost weighted heuristics (SSS_GW), Pac-Man weighted heuristics and flocking (PW_FLOCK), and finally

Pac-Man weight heuristics and ghost weight heuristics (PW_GW). In addition, this section will use short forms for the names of the response variables, they include the number of: close-calls (CC), fruit created (FC), fruit eaten (FE), ghosts eaten (GH), levels completed (LVL_COMP), power-pellets eaten (PP), repeated steps (RSQ), steps taken (St), tokens collected (TO) and the score (Sc).

It is important to remember the issues discussed in Section 5.1 that limited and ultimately altered how we performed our analysis. As mentioned, due to size restrictions, specific factors had to be separated and reviewed independently. For each factor we separated it created 2^k different game configurations, one for the high value and another for the low value, where k is the number of factors to separate. Our selection process for choosing factors to separate was based on the following general order; first factors which allowed the highest achievable score of the game to differ, followed by factors relating to player skill and when necessary relating to the fruit. It was important to control experiments which differed on highest achievable score, because the two groups produced a unfair comparison for the response variables. The reasoning for prioritizing selecting factors relating to player skill was that the game does not select these values, they must be estimated during gameplay. During a gameplay session we would not be provided accurate values of player attributes, thus this process would resemble a more realistic process. Ultimately, the true fulfillment of this process would include more generic player characteristics such as tolerance for frustration so that correlations between the difficulty of the game and specific player's characteristics could be determined. However, since we are simulating the player's performance and not using human players, this is left for future research.

The majority of results presented in this section use results from the simulation runs which were separated based on the large number of factors in an effort to reduce the factor size to 10. The results presented in nearly all of the following subsections use collected data from these separated cases. A few subsections were capable of recombining the separated cases, this produced a more general view of the algorithm and factors effects. However, this task was only accomplished for the smaller cases and thus a different type of analysis will be used for these cases. Sections with data

that was recombined will be identified their respective introductions.

5.3.1 Model Evaluation

This section presents the results of the R-Squared (R-Sq) values for each of the algorithm pairs. For each of the sections the R-Squared adjusted (Adj) are calculated for all of the response variables. This section is divided into four subsections one for each algorithm pair: Section 5.3.1.1 has the SSS-AB* against the flocking algorithm, Section 5.3.1.2 presents the results of the SSS-AB* against the ghost weight algorithm, Section 5.3.1.3 contains the results of Pac-Man weighted heuristics algorithm against the flocking algorithm and finally Section 5.3.1.4 presents the Pac-Man weighted heuristics algorithm against the ghost weighted heuristics algorithm.

5.3.1.1 SSS_FLOCK Model

The SSS_FLOCK model displayed the highest level of confidence in explaining the variance of all the models. As Table 5.3 demonstrates, the analysis produced a fairly high level of confidence in explaining the variation. Throughout all response variables the mean R-Sq(adj) ranged from 69.8 to 81.5%. Our interpretation of these high values is that SSS_FLOCK competition had a limited number of terms which had a statistically significant effect on the outcome. The SSS-AB* is a consistent and optimal algorithm, though limited in our experiment the SSS-AB* still proved to be the algorithm capable of consistently achieving highest scores and avoiding enemy attack.

The lowest explanation of variance in this model comes from the response variable for fruit collected (FE) at 69.8%, all other responses are close to 80%. Our intuitive expectations were that the number of fruit collected would be a complex response to explain. Our expectation was due to the fact that the decision to attempt to collect a fruit is based on several pieces of situational information such as: the distance to the fruit, the amount of risk in the current area and in the area of the fruit. Considering these R-Sq values were achieved after excluding a large number of terms from model,

SSS_FLOCK	R-Sq(Adj)%
CC	81.5
FC	78.0
FE	69.8
GH	79.5
LVL_COMP	75.7
PP	77.0
RSQ	77.3
Sc	76.8
St	77.5
TO	77.6

Table 5.3: The calculated R-Sq values from the SSS_FLOCK algorithm for each of the response variables.

a lack-of-fit test must be performed to validate the model.

To test whether the exclusion of the 897 other terms played a significant role in the experiment, we calculated whether the lack-of-fit was significant. The full table for the lack-of-fits calculations can be reviewed in Appendix B. The F-values where calculate for each response variable and all models, the SSS_FLOCK algorithm F-values ranged from 0.16 to 0.38, these results proved the lack-of-fit was not significant for any of the SSS_FLOCK experiment models.

5.3.1.2 SSS_GW Model

Table 5.4 illustrates that the ghost weighted heuristics algorithm produces quite a different experience than the flocking algorithm creating an environment with more terms contributing to the variance. The mean R-Sq values for this model are much lower than the SSS_FLOCK algorithm. Our results show that 56-69% of the variance is explained via our model. The low level of explanation may indicate future difficulty in the adaptation portion of the experiment. However, with more terms having an effect on variance the finer the potential granularity of the adaptation process. Similar to the last model of the SSS_FLOCK we observe that the fruit collected and level completion has the lowest explanation due the higher complexity of these responses. The close calls and ghosts killed have the highest explanations of variance in this model.

SSS_GW	R-Sq(Adj)%
CC	63.7
FC	62.3
FE	56.3
GH	68.7
LVL_COMP	55.7
PP	59.5
RSQ	61.1
Sc	57.9
St	62.2
TO	61.0

Table 5.4: The calculated R-Sq values from the SSS_GW algorithm for each of the response variables.

The significance of the lack-of-fit was tested for all response variables models for all 16 separated cases of the SSS_GW algorithm. The full table for the lack-of-fits calculations can be reviewed in Appendix B. The F-values calculated ranged from 0.30 to 0.54 and proved the lack-of-fit was not significant to the models.

5.3.1.3 PW_FLOCK Model

Table 5.5 illustrates slightly lower mean R-Sq (Adj) values where achieved in comparison to the SSS_FLOCK, but achieved higher R-Sq(adj) than the SSS_GW. Thus we were capable of explaining a large portion of the variance of the problem. As the weighted heuristics algorithms are highly parametrized, we initially expected lower R-Sq values due to a greater number of terms having an effect on the result. Producing R-Sq(adj) scores ranging from 67.7-79.2%, given the type of algorithm this hints that either a small number of terms produced large effect sizes or that a large number of factors contributed consistent scores. Similar to the other cases in this section the number of fruit eaten and levels completed had the lowest value of explained variance.

The significance of the lack-of-fit was tested for all response variable models for all 128 separated cases for the PW_FLOCK algorithm. Due to size constraints the full table in not available in the Appendix B, instead a condensed table is available which demonstrates the maximum and minimum values and the range of F-values

PW_FLOCK	R-Sq(Adj)%
CC	74.0
FC	76.3
FE	67.7
GH	79.2
LVL_COMP	71.6
PP	73.8
RSQ	76.5
Sc	74.0
St	77.2
TO	74.5

Table 5.5: The calculated R-Sq values from the PW_FLOCK algorithm for each of the response variables.

calculated for each response variable. The F-values calculated ranged from 0.15 to 0.54 and proved the lack-of-fit was not significant to the models.

5.3.1.4 PW_GW Model

The final algorithm pair produced adequate results of 53.0-74.0% for R-Sq (adj) values which are presented in table 5.6. The results for both GW sections explained less of the variance than the flocking algorithm models. The R-Sq(adj) results in this section are slightly higher than the SSS_GW algorithm. This lends to the idea that the SSS-AB* algorithm is provides more variation in gameplay, possibly as a better matched opponent. One notable trend between the sets of flocking and GW cases is that while the number of fruit eaten and levels completely are consistently have the lowest variance explanations in all the models, the number of fruit is the lowest the flocking algorithms while the levels completed has the lowest explanation in the GW algorithms.

The significance of the lack-of-fit was tested for all response variable models for all 1024 separated cases for the PW_GW algorithm. Due to size constraints the full table in not available in the Appendix B, instead a condensed table is available which demonstrates the maximum and minimum values and the range of F-values calculated for each response variable. The F-values calculated ranged from 0.15 to 0.67 and proved the lack-of-fit was not significant to the models.

PW_GW	R-Sq(Adj)%
CC	64.1
FC	72.7
FE	62.6
GH	72.3
LVL_COMP	53.0
PP	63.8
RSQ	72.9
Sc	65.6
St	74.0
TO	66.9

Table 5.6: The calculated R-Sq values from the SSS_PW_GW algorithm for each of the response variables.

5.3.2 Statistical Significance

In this section we present the results of the statistically significant effects of specific model terms. Due to the large volume of results, the presentation of the terms was organized to present only the main effects and the 2-factors interactions that proved statistically significant. Reviewing the main effects provides the opportunity to identify factors to potentially eliminate from the analysis, these factors can not be eliminated solely on the basis of their main effect statistical significance as they may interact with other factors in a significant way.

It should be noted that the statistical significance of the terms could be altered by their inclusion in commercial analysis phase. Due to the limitation of the statistical program to include greater than 127 terms, our analysis phase cannot include all terms in the model. This limitation slightly alters the calculation for the sum of squares and the size of the effects. However, this effects the terms uniformly thus they remain comparable. As our adaptive system is independent of the commercial limitations and utilizes the original calculations of a term's effect sizes to make alterations to the game, it does not disrupt the results from other sections.

The results for this section have been recombined, this reduces the number of cases and improves the ability to present and discuss the results. The results for this section are presented using two different methods. The SSS_FLOCK and SSS_GW

present the statistically significant terms in a set of factorial plots, with the full model terms being available in the Appendix A. These are standard graphical plots used for the presentation of factorial analysis. Due to the volume of the PW_FLOCK and PW_GW experimental sets our statistical programs were unable to create factorial plots for these two cases. Thus, we provided tables for the statistically significant main effects and 2-factors interactions.

Additionally, in this section we have omitted the response variable fruits created. The fruit creation is entirely based on the number of steps the player has taken and the frequency of the fruit factor, the inclusion of these tables seemed superfluous.

5.3.2.1 SSS_FLOCK Terms

The first algorithm pair we will present results for is the SSS_FLOCK, this section was the simplest of all cases as the experimental design was separated based on only one factor. As previously mentioned the frequency of fruit creation or the rate of fruit creation will be a separation factor for all cases as it alters the highest achievable score. If a higher number of fruit are available more points can be gained over the same number of steps. The separation of the factor means that SSS_FLOCK pair has to consider two game sessions, one for the low value the frequency of creation set to 1/100 per steps and one set to high value at 1/50 per steps.

For determining the statistical significance we were able to recombine the two sub-cases separated based on the fruit frequency. This allowed us to calculate the effect sizes for all of the factors and their interactions, excluding those which interacted with the frequency of the fruit creation. This reduces the number of figures required, and eases the graphical demonstration of results. The following subsection displays the results of the models for each of response variables. A large amount of similarity occurs between the results of each response variable. This similarity is expected as factors which alter the length of time Pac-Man is alive ultimately alter the length of time to acquire tokens, power-pellets or other items.

5.3.2.1.1 SSS_FLOCK Close Call Terms

Figure 5.1 illustrates the statistically significant main effects of the SSS_FLOCK model for the response variable close calls. The FLOCK_HUNGER main effect produces the largest variation of the main effects on the number of close calls. As intuitively expected the number of close calls increases as the value of the flock's hunger increased. This occurs because the flock places a higher priority on chasing Pac-Man. The FLEE_TIME factor at its high level value decreased the number of close calls. This occurs due to ghosts avoiding Pac-Man when in a fleeing state and thus the longer they remain in a fleeing state the longer they stay away for Pac-Man. The ghost's vision range at its high level decreased the number of close calls, possibly due to greater success in capturing Pac-Man.

Figure 5.2 illustrates the 2-factor interactions of the SSS_FLOCK model for the response variable close calls. The figure includes only factors which contributed significantly to the interaction of at least one other factor. If the lines intersect in a cell it indicates that an interaction occurred between the factors. If parallel lines occur in a cell it indicates that the factors do not interact. Figure 5.2 indicates that FLOCK_HUNGER interacts with several other factors. In addition DEATH_TIME which was not a significant main effect interacts with most of the listed factors. We observe interaction between all of the flocking algorithm factors and usually with the ghosts vision length.

A full list of the terms included in the SSS_FLOCK close calls model can be found in Appendix A.1. The full list of terms indicates that the largest factor effect was for the FLOCK_HUNGER followed by the interaction of FLOCK_HUNGER and FLOCK_SEP. This result justifies an apriori expectation that prioritizing chasing and spreading out across the border would result in an increased number of close calls. The lack-of-fit value for this value was $p = 1.00$, which indicates that terms omitted by this model did not play a significant role in the outcome of the experiment.

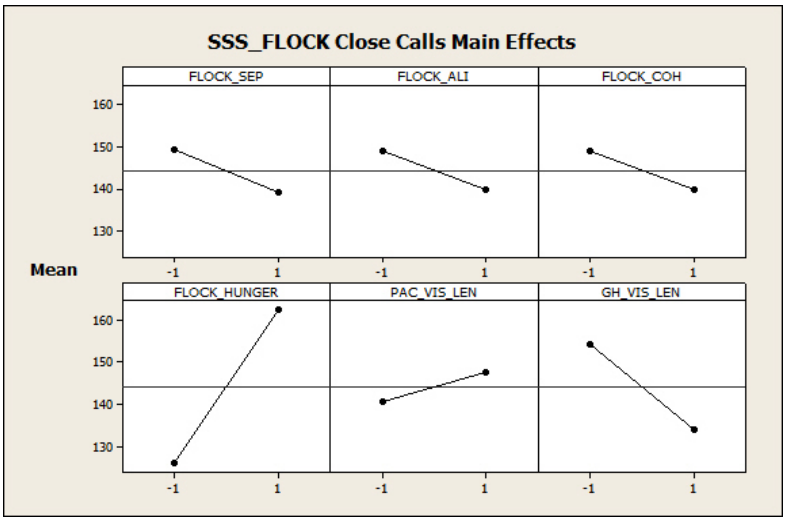


Figure 5.1: Demonstrating the SSS_FLOCK significant main effects for the response variable Close Calls. The coded values -1 and 1 represent the low and high factor levels, respectively. The steeper the slope of the line the larger the difference effect sizes.

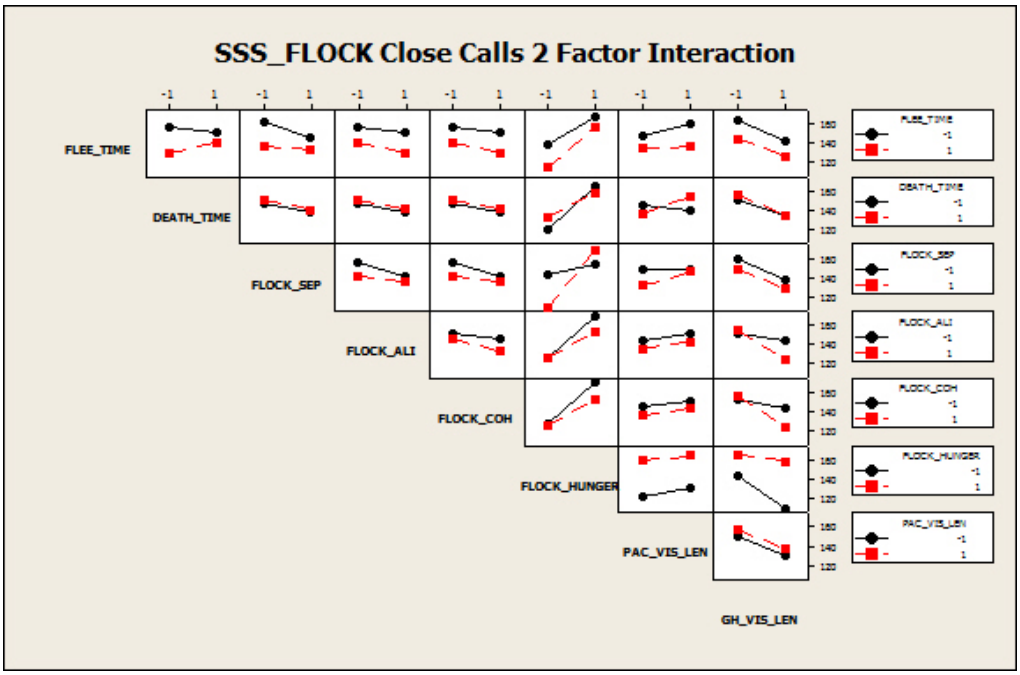


Figure 5.2: Demonstrating the SSS_FLOCK significant 2-factor interactions for the response variable Close Calls. Parallel lines indicate that no interaction between the two factors. Intersecting lines indicate that an interaction did occur.

5.3.2.1.2 SSS_FLOCK Fruits Collected Terms

Figure 5.3 illustrates the statistically significant main effects of SSS_FLOCK model for the response variable fruits eaten. The largest positive main effect occurred from the factor FRUIT_TIME, and the largest negative main effect from GH_VIS_LEN. These results have an intuitive explanation that the number of fruits eaten should increased the longer they are available on screen. As the performance of the ghosts improves with increased vision range the number of fruits eaten decreases. This is one of the few cases where FRUIT_TIME and PER_FRT values are statistically significant as main effects or 2-factor interactions.

Figure 5.4 illustrates the 2-factor interactions of the SSS_FLOCK model for the response variable fruits eaten. The figure includes only factors which contributed significantly to the interaction of at least one other factor. If the lines intersect in a cell it indicates that an interaction occurred between the factors. If parallel lines occur in a cell it indicates that the factors do not interact. The ghost's vision range (GH_VIS_LEN) interacted with all other factors. In addition, we observed FLEE_TIME interacted with most factors causing slight improvements in fruit eaten. The factors FLEE_TIME, FLOCK_ALI, FLOCK_COH and FLOCK_HUNGER were not significant main effects.

A full list of the terms included in the SSS_FLOCK fruits eaten model can be found in Appendix A.1. The full list of terms indicates that the largest factor effect was for the FRUIT_TIME followed by GH_VIS_LEN. The lack-of-fit value for this value was $p = 1.00$, which indicates that terms omitted by this model did not play a significant role in the outcome of the experiment.

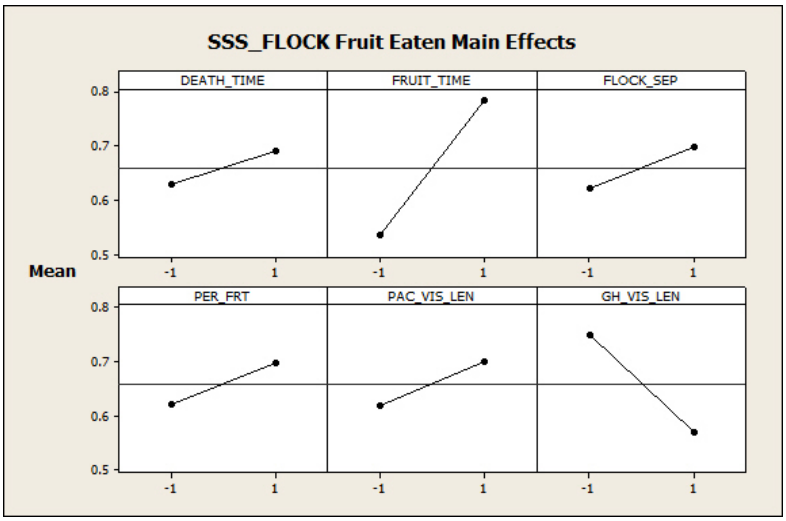


Figure 5.3: Demonstrating the SSS_FLOCK significant main effects for the response variable fruit eaten. The coded values -1 and 1 represent the low and high factor levels, respectively. The steeper the slope of the line the larger the difference effect sizes.

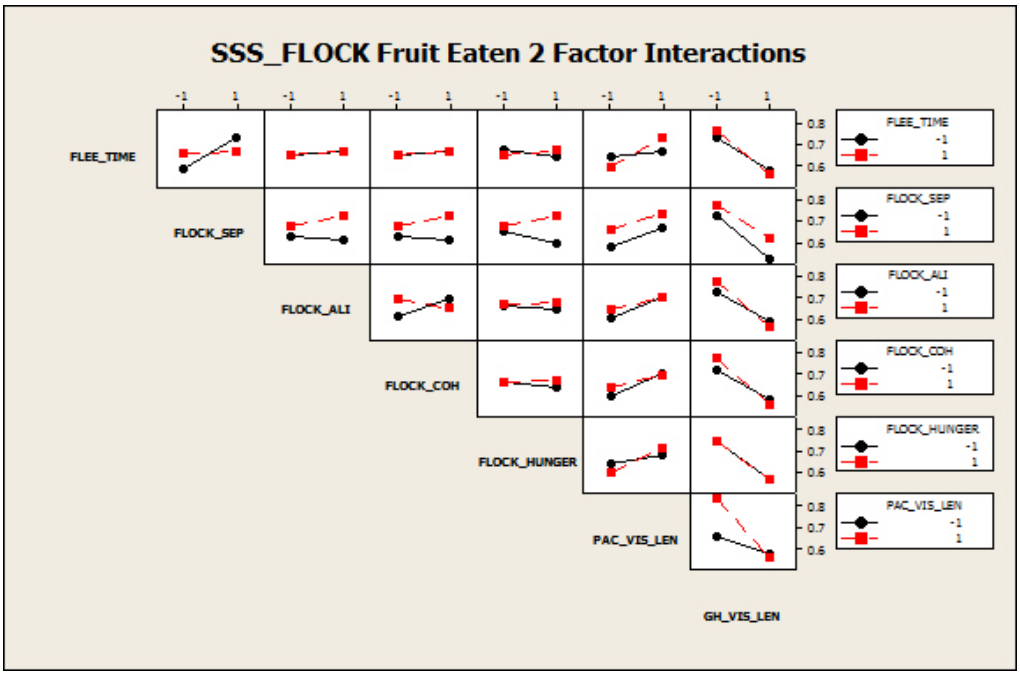


Figure 5.4: Demonstrating the SSS_FLOCK significant 2-factor interactions for the response variable fruit eaten. Parallel lines indicate that no interaction between the two factors. Intersecting lines indicate that an interaction did occur.

5.3.2.1.3 SSS_FLOCK Ghosts Eaten Terms

Figure 5.5 illustrates the statistically significant main effects of SSS_FLOCK model for the response variable ghosts eaten. The largest positive main effect occurred from the factor FLEE_TIME, and the largest negative main effect was the GH_VIS_LEN. These results have an intuitive explanation that longer the ghosts remain in a fleeing state the longer they can be eaten. Increasing the ghost's vision range increases the ghosts ability to view and avoid Pac-Man. Interestingly, FLOCK_HUNGER was not a significant main effect indicating that although the ghosts prioritized chasing Pac-Man, Pac-Man was unable to improve the number of ghosts eaten by counter-attacking.

Figure 5.6 illustrates the 2-factor interactions of the SSS_FLOCK model for the response variable ghosts eaten. The figure includes only factors which contributed significantly to the interaction of at least one other factor. If the lines intersect in a cell it indicates that an interaction occurred between the factors. If parallel lines occur in a cell it indicates that the factors do not interact. Similar to the previous cases we observe that the ghosts and Pac-Man vision range interact with a large number of other factors. Although the DEATH_TIME factor was not a significant factor, it interacts with every factor listed in Figure 5.6. This is a reoccurring trend for DEATH_TIME factor throughout the first few cases. The perceived value of the fruit (PER_FRT) interacts with FLOCK_HUNGER, FRUIT_TIME and DEATH_TIME indicating a trade-off between collecting fruits and the number of ghosts eaten.

A full list of the terms included in the SSS_FLOCK ghosts eaten model can be found in Appendix A.1. The full list of terms indicates that the largest factor effect was for the FRUIT_TIME followed by GH_VIS_LEN. The lack-of-fit value for this value was $p = 1.00$, which indicates that terms omitted by this model did not play a significant role in the outcome of the experiment.

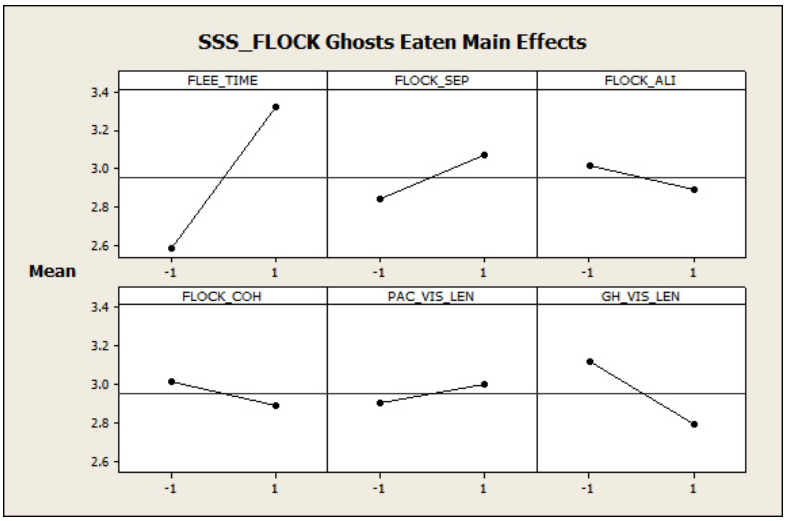


Figure 5.5: Demonstrating the SSS_FLOCK significant main effects for the response variable Close Calls. The coded values -1 and 1 represent the low and high factor levels, respectively. The steeper the slope of the line the larger the difference effect sizes.

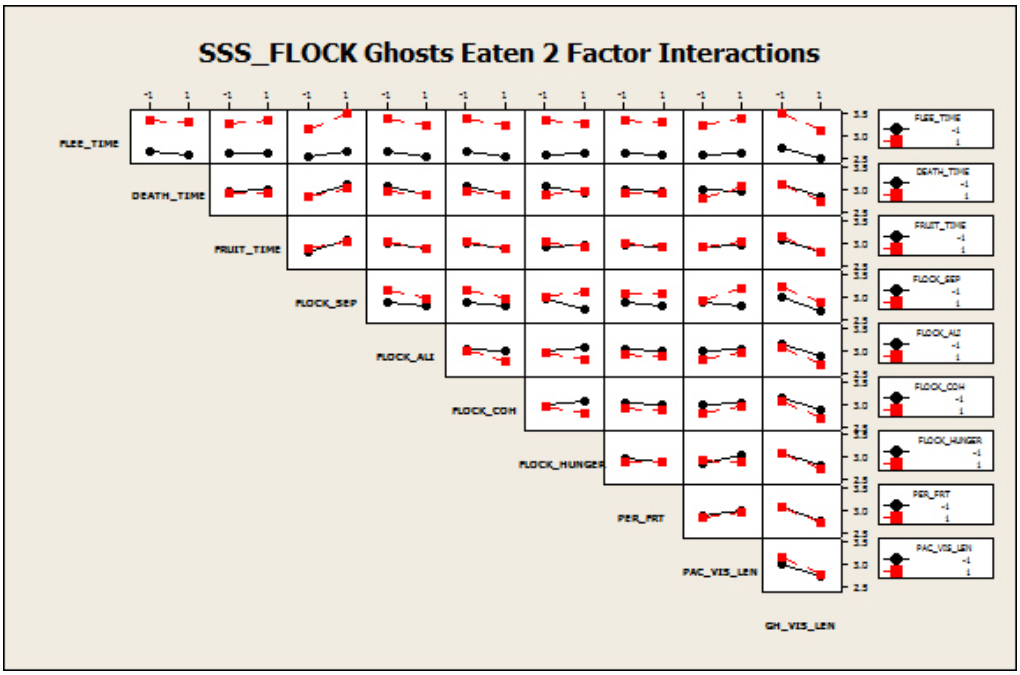


Figure 5.6: Demonstrating the SSS_FLOCK significant 2-factor interactions for the response variable ghosts eaten. Parallel lines indicate that no interaction between the two factors. Intersecting lines indicate that an interaction did occur.

5.3.2.1.4 SSS_FLOCK Levels Completed Terms

Figure 5.7 illustrates the statistically significant main effects of SSS_FLOCK model for the response variable levels completed. The largest positive main effect occurred from the factor PAC_VIS_LEN, and the largest negative main effect from GH_VIS_LEN. As the state factors FLEE_TIME and DEATH_TIME increased so too did the number of levels completed. Interesting, an increase in the value of the factor FLOCK_HUNGER increased the number of levels completed. One possible reason for this result is that Pac-Man was capable of efficiently avoiding the flock, while the ghosts constant chasing provided the opportunity to explore less protected areas of the board.

Figure 5.8 illustrates the 2-factor interactions of the SSS_FLOCK model for the response variable levels completed. The figure includes only factors which contributed significantly to the interaction of at least one other factor. If the lines intersect in a cell it indicates that an interaction occurred between the factors. If parallel lines occur in a cell it indicates that the factors do not interact. Similar to the previous cases we observe that the ghosts and Pac-Man vision range interact with a large number of other factors. This case that DEATH_TIME was a significant main effect, but contributes less overall interaction with other factors.

A full list of the terms included in the SSS_FLOCK levels completed model can be found in Appendix A.1. The full list of terms indicates that the largest factor effects were the vision parameters. In addition the flocking alignment and cohesion interacted significantly with a number of the largest terms. The lack-of-fit value for this value was $p = 1.00$, which indicates that terms omitted by this model did not play a significant role in the outcome of the experiment.

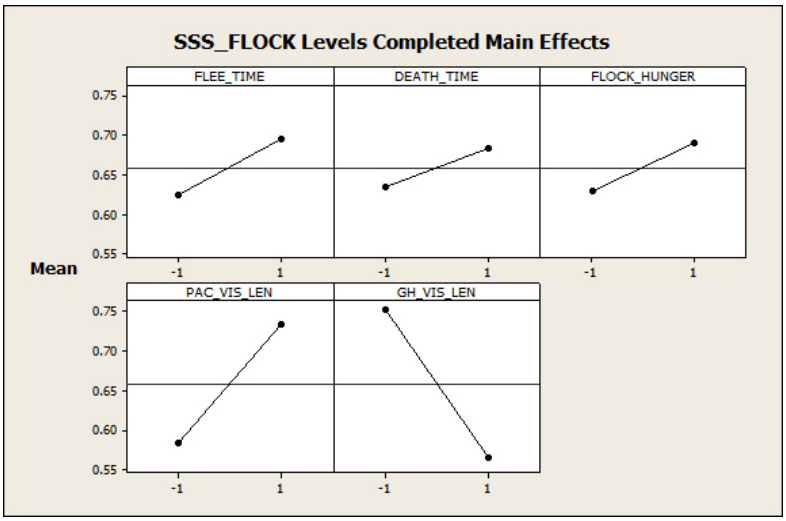


Figure 5.7: Demonstrating the SSS_FLOCK significant main effects for the response variable levels completed. The coded values -1 and 1 represent the low and high factor levels, respectively. The steeper the slope of the line the larger the difference effect sizes.

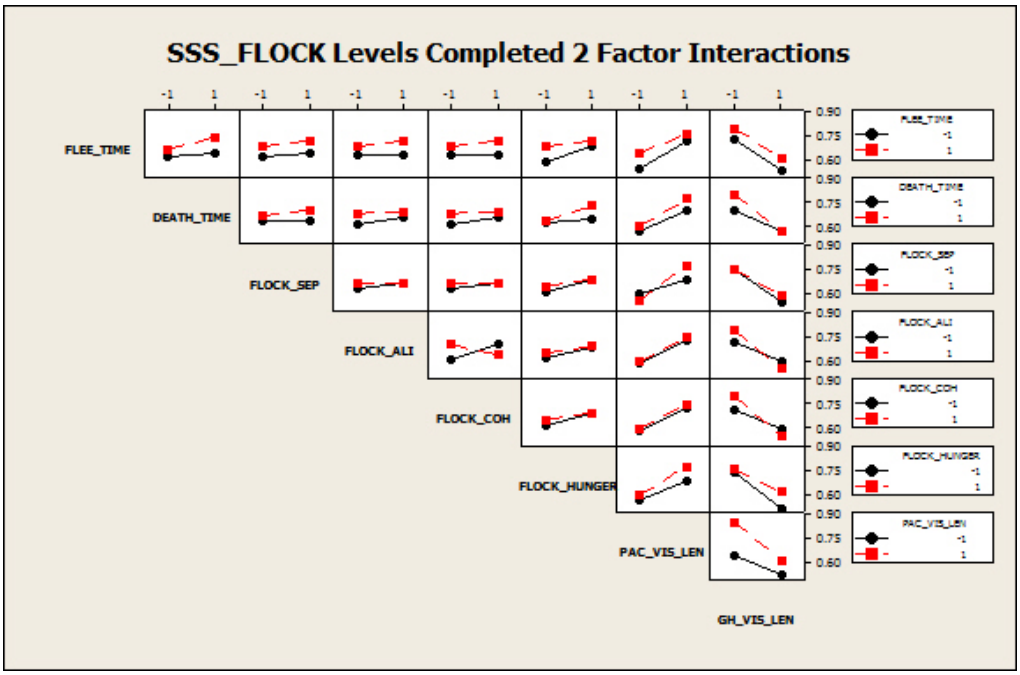


Figure 5.8: Demonstrating the SSS_FLOCK significant 2-factor interactions for the response variable levels completed. Parallel lines indicate that no interaction between the two factors. Intersecting lines indicate that an interaction did occur.

5.3.2.1.5 SSS_FLOCK Power-Pellets Collected Terms

Figure 5.9 illustrates the statistically significant main effects of SSS_FLOCK model for the response variable power-pellets collected. The largest positive main effect is PAC_VIS_LEN, and the largest negative main effect was the GH_VIS_LEN. As the state factors FLEE_TIME and DEATH_TIME increased so too did the number of power-pellets collected.

Figure 5.10 illustrates the 2-factor interactions of the SSS_FLOCK model for the response variable power-pellets eaten. The figure includes only factors which contributed significantly to the interaction of at least one other factor. If the lines intersect in a cell it indicates that an interaction occurred between the factors. If parallel lines occur in a cell it indicates that the factors do not interact. Similar to the previous cases we observe that the ghosts and Pac-Man vision range interact with a large number of other factors. The FLEE_TIME and DEATH_TIME factors interact with fewer factors than most of the other cases.

A full list of the terms included in the SSS_FLOCK power-pellets eaten model can be found in Appendix A.1. Excluding the previously mentioned main effects the terms with the largest effect on the power-pellets collected related to the interaction of flocking alignment, cohesion and Pac-Man's vision range. The lack-of-fit value for this value was $p = 1.00$, which indicates that terms omitted by this model did not play a significant role in the outcome of the experiment.

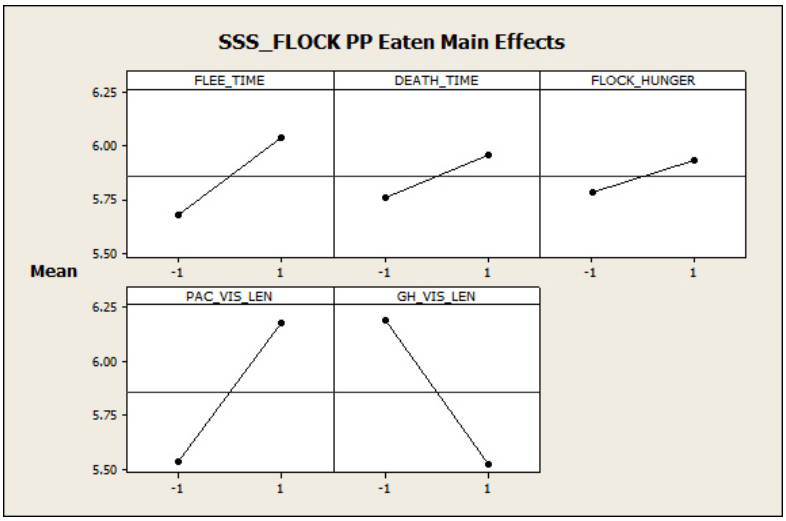


Figure 5.9: Demonstrating the SSS_FLOCK significant main effects for the response variable Power-Pellet eaten. The coded values -1 and 1 represent the low and high factor levels, respectively. The steeper the slope of the line the larger the difference effect sizes.

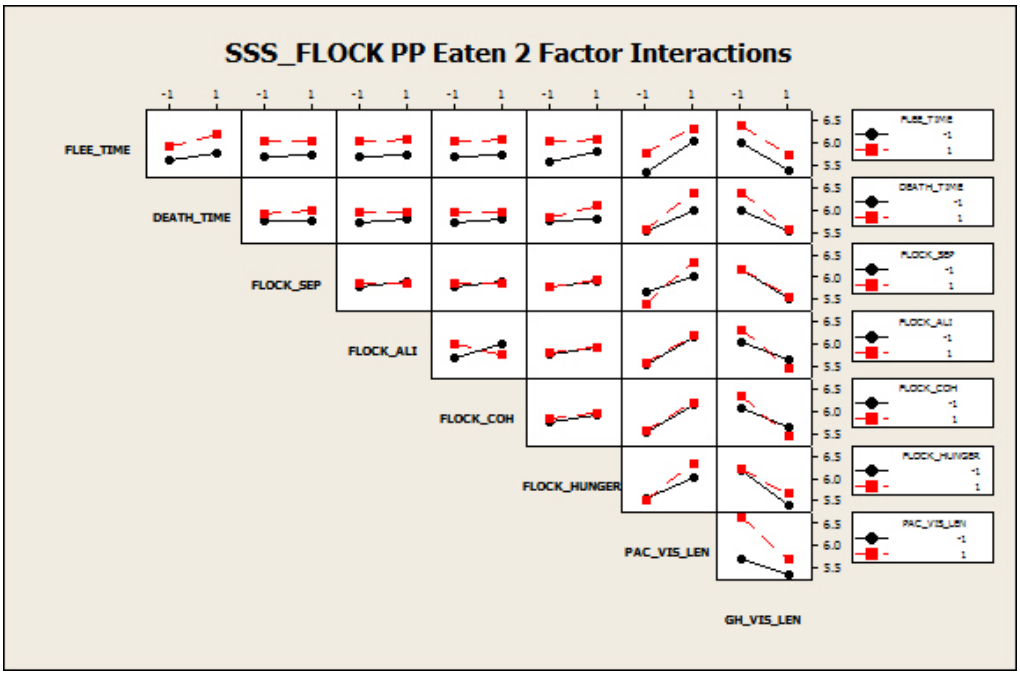


Figure 5.10: Demonstrating the SSS_FLOCK significant 2-factor interactions for the response variable Power-Pellets eaten. Parallel lines indicate that no interaction between the two factors. Intersecting lines indicate that an interaction did occur.

5.3.2.1.6 SSS_FLOCK Repeated Squares Terms

Figure 5.11 illustrates the statistically significant main effects of SSS_FLOCK model for the response variable repeated squares. The largest positive main effect is PAC_VIS_LEN, and the largest negative main effect was the GH_VIS_LEN. The factors FLEE_TIME has a larger effect than the DEATH_TIME factor. In addition, we observe that factors FLOCK_SEP and FLOCK_HUNGER are both causing a statistically significant increase in the number of repeated squares.

Figure 5.12 illustrates the 2-factor interactions of the SSS_FLOCK model for the response variable repeated squares. The figure includes only factors which contributed significantly to the interaction of at least one other factor. If the lines intersect in a cell it indicates that an interaction occurred between the factors. If parallel lines occur in a cell it indicates that the factors do not interact. Similar to the previous cases we observe that the ghosts and Pac-Man vision range interact with a large number of other factors.

A full list of the terms included in the SSS_FLOCK repeated squares model can be found in Appendix A.1. Excluding the previously mentioned main effects the terms with the largest effect on the repeated squares response factors interacting with the ghosts and Pac-Man's vision range. The lack-of-fit value for this value was $p = 1.00$, which indicates that terms omitted by this model did not play a significant role in the outcome of the experiment.

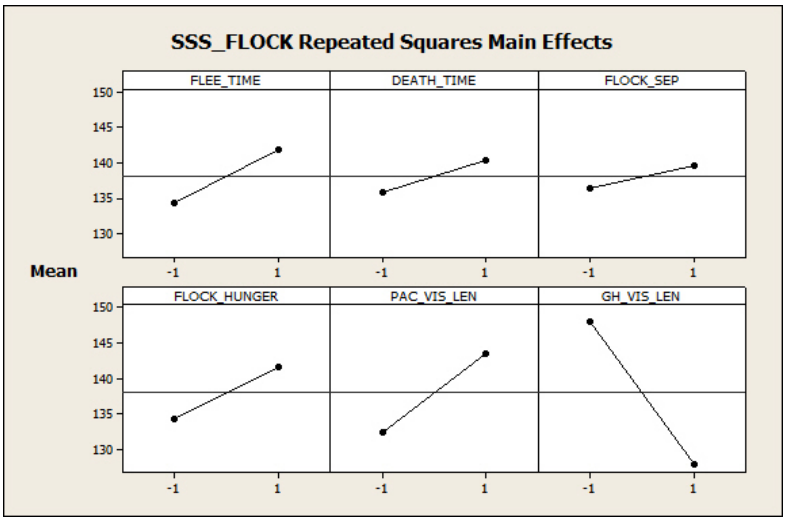


Figure 5.11: Demonstrating the SSS_FLOCK significant main effects for the response variable repeated squares. The coded values -1 and 1 represent the low and high factor levels, respectively. The steeper the slope of the line the larger the difference effect sizes.

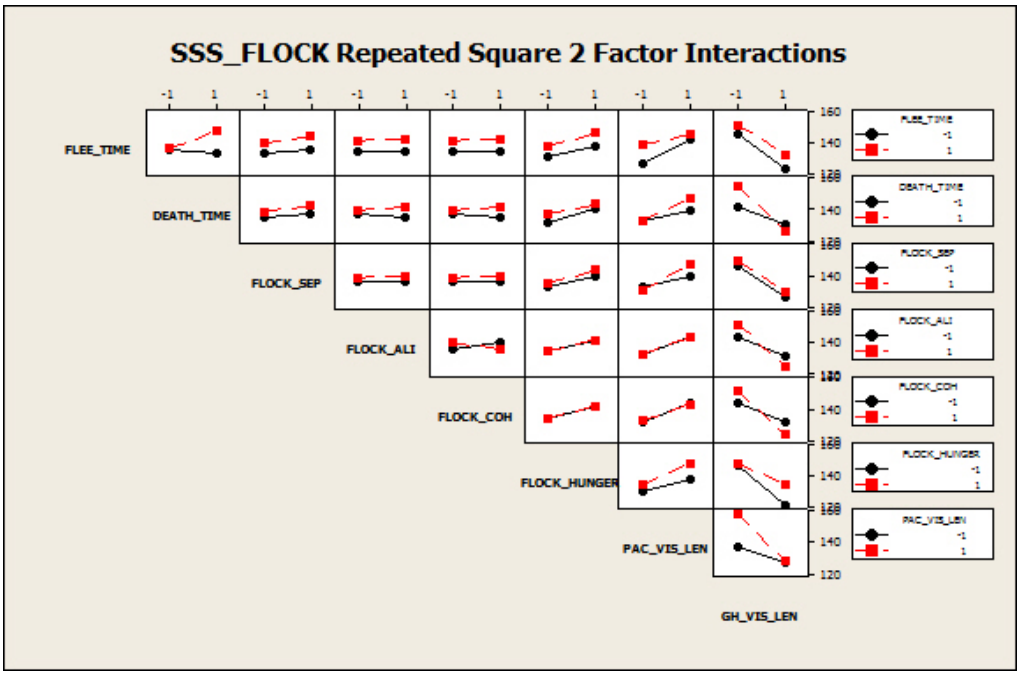


Figure 5.12: Demonstrating the SSS_FLOCK significant 2-factor interactions for the response variable repeated squares. Parallel lines indicate that no interaction between the two factors. Intersecting lines indicate that an interaction did occur.

5.3.2.1.7 SSS_FLOCK Score Terms

Figure 5.13 illustrates the statistically significant main effects of SSS_FLOCK model for the response variable score. The largest positive main effect occurred from the factor PAC_VIS_LEN, and the largest negative main effect from GH_VIS_LEN. The FLEE_TIME main effect contributes only slightly less than increasing Pac-Man's vision range.

Figure 5.14 illustrates the 2-factor interactions of the SSS_FLOCK model for the response variable score. The figure includes only factors which contributed significantly to the interaction of at least one other factor. If the lines intersect in a cell it indicates that an interaction occurred between the factors. If parallel lines occur in a cell it indicates that the factors do not interact. Similar to the previous cases we observe that the ghosts and Pac-Man vision range interact with a large number of other factors. The FLEE_TIME factor interacts in statistically significant way with only DEATH_TIME and FLOCK_HUNGER.

A full list of the terms included in the SSS_FLOCK score model can be found in Appendix A.1. Excluding the previously mentioned main effects, the terms with the largest effect on the score response are the results of the interactions of the flocking separation and hunger and the interaction of alignment and cohesion The lack-of-fit value for this value was $p = 1.00$, which indicates that terms omitted by this model did not play a significant role in the outcome of the experiment.

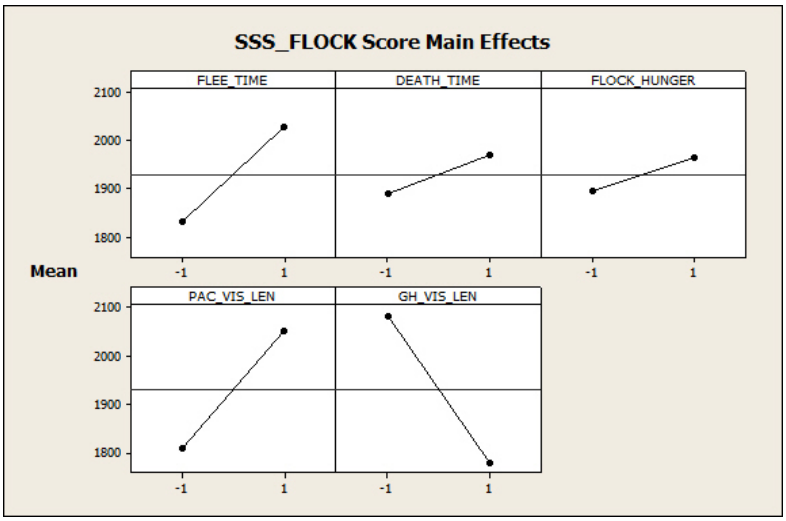


Figure 5.13: Demonstrating the SSS_FLOCK significant main effects for the response variable score. The coded values -1 and 1 represent the low and high factor levels, respectively. The steeper the slope of the line the larger the difference effect sizes.

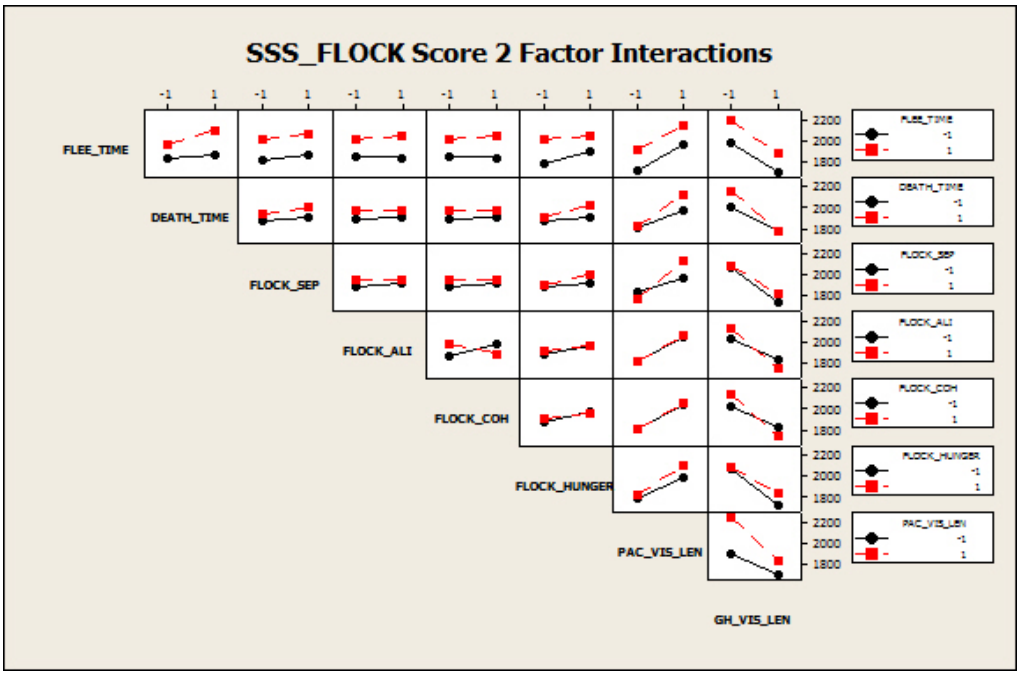


Figure 5.14: Demonstrating the SSS_FLOCK significant 2-factor interactions for the response variable score. Parallel lines indicate that no interaction between the two factors. Intersecting lines indicate that an interaction did occur.

5.3.2.1.8 SSS_FLOCK Steps Terms

Figure 5.15 illustrates the statistically significant main effects of SSS_FLOCK model for the response variable steps. The largest positive main effect occurred from the factor PAC_VIS_LEN, and the largest negative main effect from GH_VIS_LEN. The FLEE_TIME main effect contributes only slightly less than increasing Pac-Man's vision range.

Figure 5.16 illustrates the 2-factor interactions of the SSS_FLOCK model for the response variable steps. The figure includes only factors which contributed significantly to the interaction of at least one other factor. If the lines intersect in a cell it indicates that an interaction occurred between the factors. If parallel lines occur in a cell it indicates that the factors do not interact. Similar to the previous cases we observe that the ghosts and Pac-Man vision range interact with a large number of other factors. The FLEE_TIME factor interacts in statistically significant way with only DEATH_TIME and FLOCK_HUNGER.

A full list of the terms included in the SSS_FLOCK steps model can be found in Appendix A.1. Excluding the previously mentioned main effects, the terms with the largest effect on the score response are the results of the interactions of the flocking separation and hunger and the interaction of alignment and cohesion The lack-of-fit value for this value was $p = 1.00$, which indicates that terms omitted by this model did not play a significant role in the outcome of the experiment.

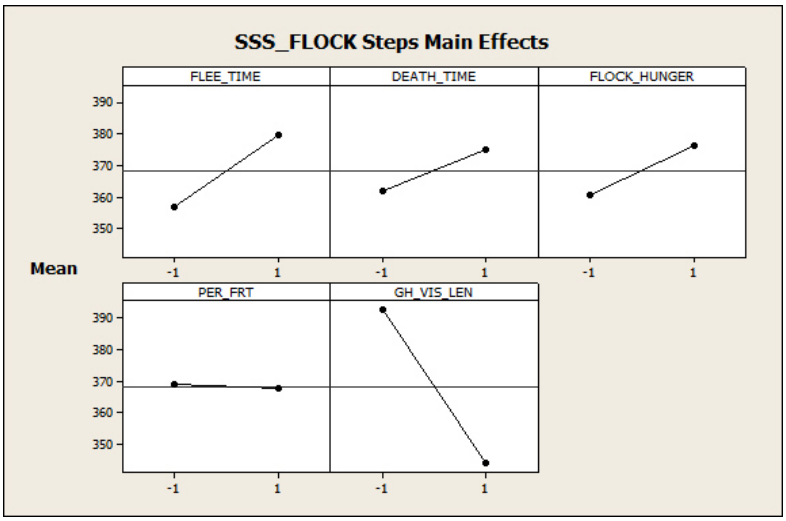


Figure 5.15: Demonstrating the SSS_FLOCK significant main effects for the response variable steps. The coded values -1 and 1 represent the low and high factor levels, respectively. The steeper the slope of the line the larger the difference effect sizes.

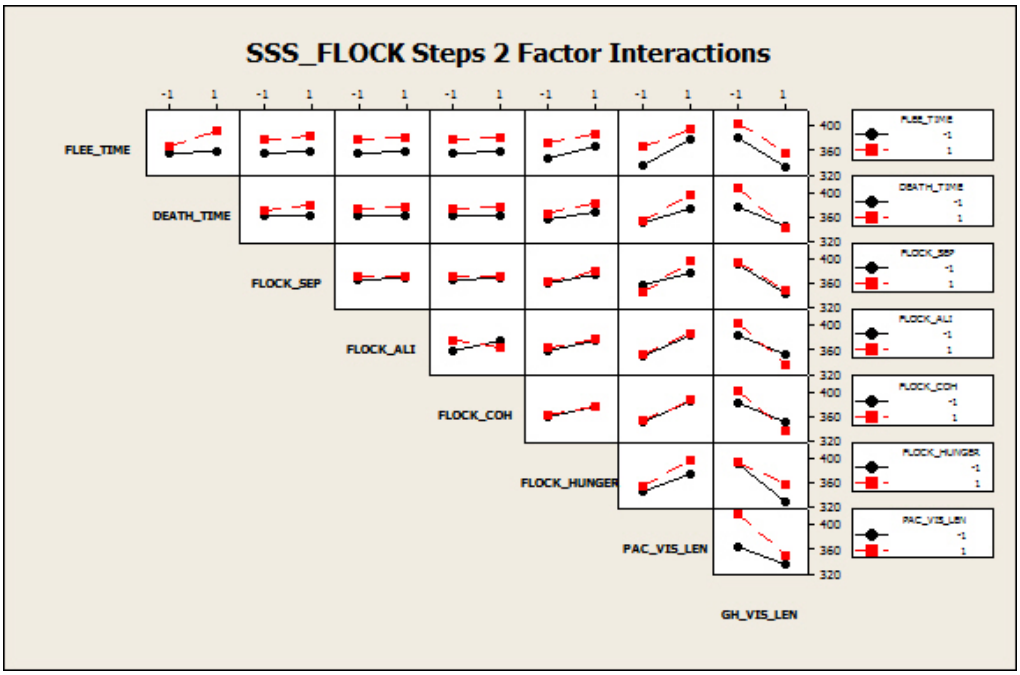


Figure 5.16: Demonstrating the SSS_FLOCK significant 2-factor interactions for the response variable steps. Parallel lines indicate that no interaction between the two factors. Intersecting lines indicate that an interaction did occur.

5.3.2.1.9 SSS_FLOCK Tokens Collected Terms

Figure 5.17 illustrates the statistically significant main effects of SSS_FLOCK model for the response variable tokens collected. The largest positive main effect is PAC_VIS_LEN, and the largest negative main effect was the GH_VIS_LEN. The FLEE_TIME main effect contributes only slightly less than increasing Pac-Man's vision range.

Figure 5.18 illustrates the 2-factor interactions of the SSS_FLOCK model for the response variable tokens collected. The figure includes only factors which contributed significantly to the interaction of at least one other factor. If the lines intersect in a cell it indicates that an interaction occurred between the factors. If parallel lines occur in a cell it indicates that the factors do not interact. Similar to the previous cases we observe that the ghosts and Pac-Man vision range interact with a large number of other factors.

A full list of the terms included in the SSS_FLOCK steps model can be found in Appendix A.1. Excluding the previously mentioned main effects, the terms with the largest effect on the score response are the results of the interactions of the flocking separation and hunger and the interaction of alignment and cohesion The lack-of-fit value for this value was $p = 1.00$, which indicates that terms omitted by this model did not play a significant role in the outcome of the experiment.

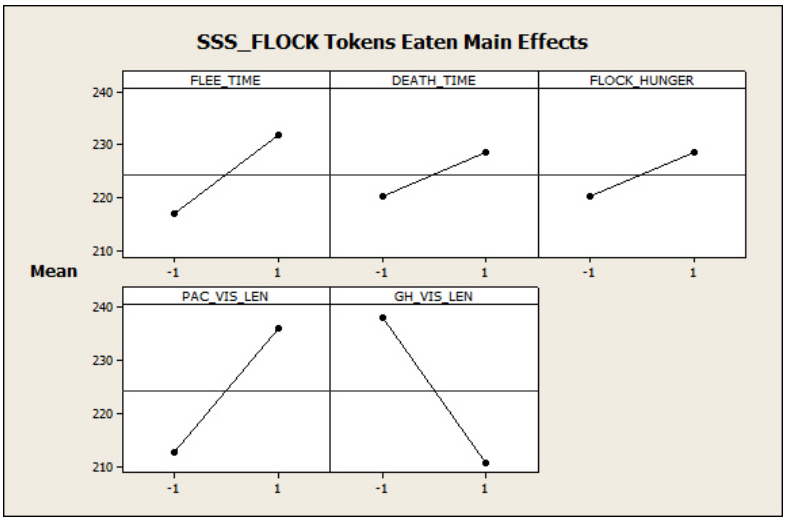


Figure 5.17: Demonstrating the SSS_FLOCK significant main effects for the response variable tokens. The coded values -1 and 1 represent the low and high factor levels, respectively. The steeper the slope of the line the larger the difference effect sizes.

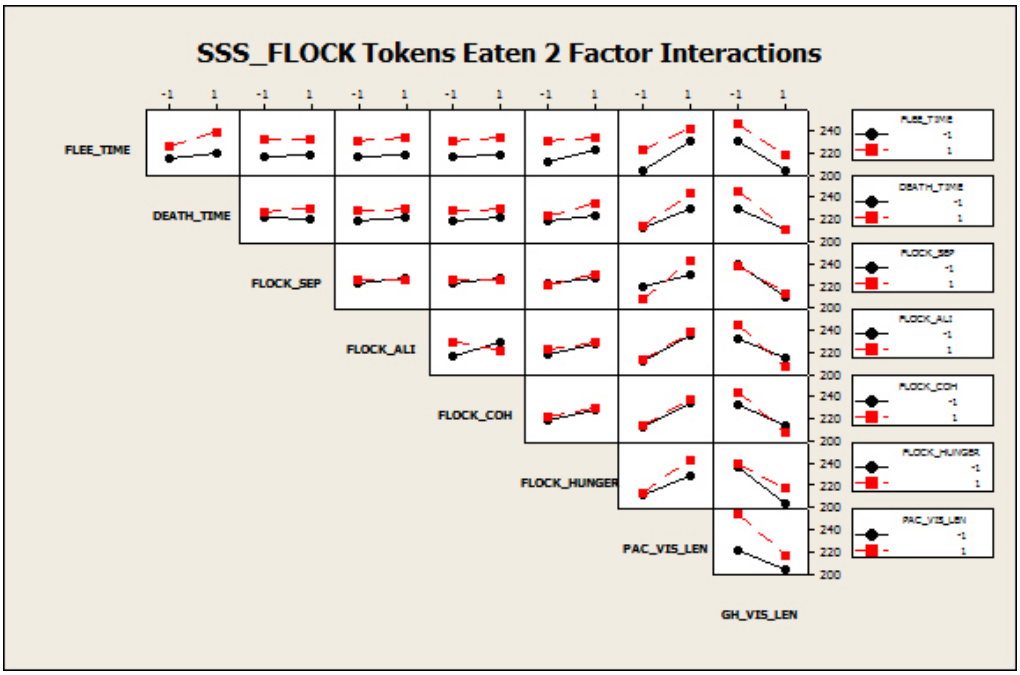


Figure 5.18: Demonstrating the SSS_FLOCK significant 2-factor interactions for the response variable tokens. Parallel lines indicate that no interaction between the two factors. Intersecting lines indicate that an interaction did occur.

5.3.2.1.10 SSS_FLOCK Summary

In summary, the results of the SSS-AB* and flocking algorithm illustrated a couple of reoccurring themes. Generally, as expected the vision range factors are consistently statistically significant main effects and are often the largest positive and negative effects. The factor FLEE_TIME produced consistently results as a statistically significant main effect. For the response variables: score, repeated steps, tokens and power-pellets collected the FLEE_TIME was similar in effect size to Pac-Man's vision range. The factor for the duration of the death state tended to be a statistically significant main effect but also showed to interact with a number of other factors, especially the flocking separation factor. The length of time the fruit was available played a minor role in most response variables, and played an expected role in the number of fruit acquired. The factors related to the flocking algorithm tended to be dominated by the separation and hunger factors. Often as main effects the flocking factors would be non-significant, this is explained by the fact that each factor contributes equally to the algorithm and thus may not sway the decision process with only minor adjustments. However, the interactions of the flocking algorithm were a consistent staple in the top effects list. The flocking hunger often interacted with the other flocking algorithm factors, but also occurred most often as a statistically significant main effect. The hunger contributed to increasing the number of close calls and diminishing Pac-Man's overall results in a number of response variables such as tokens collected and steps. As the results in Appendix A.1 will indicate the flocking algorithm factors interacted mostly in pairs. The factors flock hunger and separation appeared as number of the largest statistically significant terms. However, the other reoccurring interaction set was the flock factors alignment and cohesion usually appearing in unison in the set of larger interactions.

5.3.2.2 SSS_GW Terms

The second algorithm pair evaluated was the SSS_GW. We selected four factors to separate our game information, this resulted in 2^4 or 16 game combinations. The factors selected were the perceived value of the fruit (SSS_PERCEIVED_FRUIT), the

frequency of the fruit creation (FRUIT_FREQ), the time the fruit was available on screen (FRUIT_TIME) and finally the range of Pac-Man's vision (PAC_VIS_LEN).

Due to the large number of game combinations for this case creating factorial plots for the main effects and 2-factor interactions became an excessive task. The previous section on SSS_FLOCK terms recombined all of the models for both cases. This resulted in 10 different models, one to explain each of the response variables. This was possible due to the fact that restructuring the terms and selecting the top 127 terms including the necessary main effects, produced models that did not have significant lack-of-fit. This indicated that no significant terms were excluded from the model. However, lack-of-fit tests for recombining the SSS_GW terms indicating a significant lack-of-fit for each response variable. Thus, the SSS_GW model would be limited by the 127 maximum number of terms per model. Thus a recombined model would not be valid, and the results can not be used. The statistical significance of each term was calculated outside of this limitation. The factorial plots were calculated for a model containing only the main effects and 2-factor interactions.

5.3.2.2.1 SSS_GW Close Call Terms

Figure 5.19 illustrates the statistically significant main effects of SSS_GW model for the response variable close calls. The largest positive main effect is GH_VIS_LEN, and the largest negative main effect was the GW_AWAY_GH which moves ghosts away from each other. The GW_AWAY_GH factor performs a similar task as the FLOCK_SEP which provides an impulse to push ghosts away from each other. Similar to the results of the FLOCK_SEP, the GW_AWAY_GH decreases the number of close calls. This result occurs due to an increase in the effectiveness of the ghosts performance. The GW_PAC_DIR which prioritizes moving towards Pac-Man's next square based on the current path and direction, is one of the largest effects and has a positive effect on the number of close calls. The main effect size of the Pac-Man's direction is larger than the main effect size of GW_PAC, which prioritizes Pac-Man's current position. In addition, the FLEE_TIME and DEATH_TIME decreased the number of close calls as increased. Interestingly, this is one of the few cases where

the GW_FT main effect is statistically significant. The GW_FT indicates a priority to protect the fruit for the ghosts.

Figure 5.20 illustrates the 2-factor interactions of the SSS_GW model for the response variable close calls. The figure includes only factors which contributed significantly to the interaction of at least one other factor. If the lines intersect in a cell it indicates that an interaction occurred between the factors. If parallel lines occur in a cell it indicates that the factors do not interact. The largest interactions occurred as a a result of the vision parameters. The GW_PP factor which was not a statistically significant main effect, interacts in a statistically significant way with the the vision and state factors producing the next largest effect sizes.

A full list of the statistically significant main effects and 2-factor interactions for the SSS_GW close calls model can be found in Appendix A.1. The lack-of-fit value for this value was $p = 0.00$, which indicates that terms omitted by this model played a significant role in the outcome of the experiment. In this case the 127 maximum number of terms limited the analysis of the model, and the statistical significance of the terms was calculated independent of the term limitation.

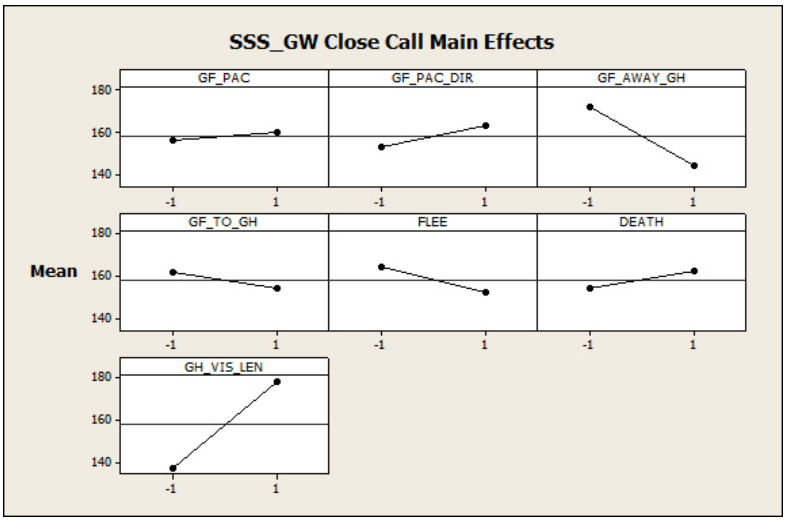


Figure 5.19: Demonstrating the SSS_GW significant main effects for the response variable Close Calls. The coded values -1 and 1 represent the low and high factor levels, respectively. The steeper the slope of the line the larger the difference effect sizes.

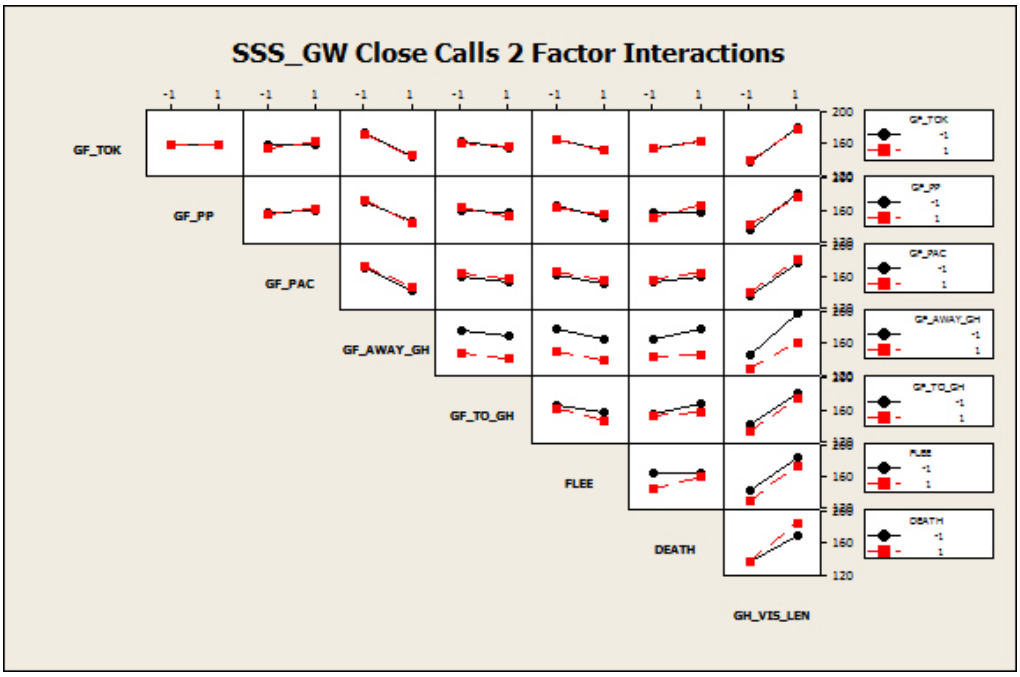


Figure 5.20: The SSS_GW significant 2-factor interactions for the response variable Close Calls. Parallel lines indicate no interaction occurred between terms, intersecting lines indicate interaction.

5.3.2.2.2 SSS_GW Fruits Collected Terms

Figure 5.21 illustrates the statistically significant main effects of SSS_GW model for the response variable fruit collected. The largest negative main effect occurred from factor GH_VIS_LEN, and the largest positive main effect occurred from the factor DEATH_TIME. The positive effect of the DEATH_TIME factor indicates that Pac-Man used a portion of this time to achieve bonus task. The FLEE_TIME and GW_AWAY_GH had some of the largest statistically significant effect sizes.

Figure 5.22 illustrates the 2-factor interactions of the SSS_GW model for the response variable fruit collected. The figure includes only factors which contributed significantly to the interaction of at least one other factor. If the lines intersect in a cell it indicates that an interaction occurred between the factors. If parallel lines occur in a cell it indicates that the factors do not interact. The largest interactions occurred as a result of GW_TO_GH and FLEE_TIME factor. This supports the idea that Pac-Man was effective in consuming ghosts with the additional flee time and then used the ghost free time to collected fruits. Additional support for this play style is provided by another of the largest interactions coming from the interaction of GW_TO_GH and GW_PP factors.

A full list of the statistically significant main effects and 2-factor interactions for the SSS_GW fruit collected model can be found in Appendix A.1. The lack-of-fit value for this value was $p = 0.00$, which indicates that terms omitted by this model played a significant role in the outcome of the experiment. In this case the 127 maximum number of terms limited the analysis of the model, and the statistical significance of each term was recalculated independent of the term limitation.

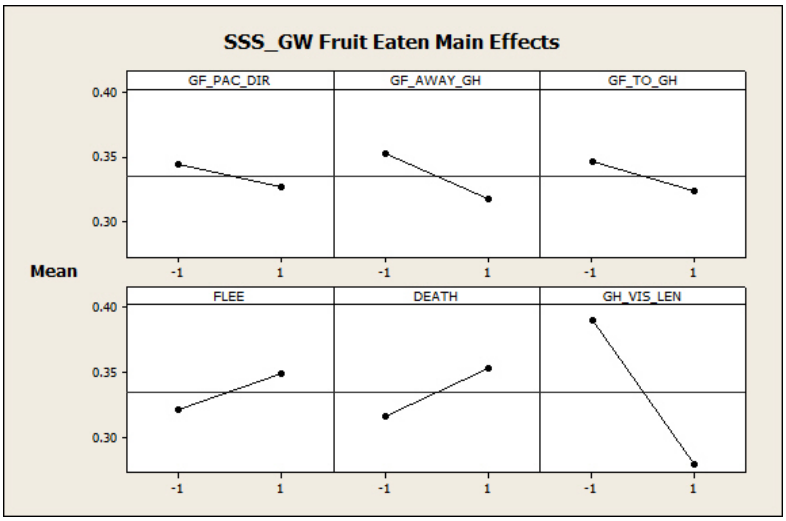


Figure 5.21: Demonstrating the SSS_GW significant main effects for the response variable fruits eaten. The coded values -1 and 1 represent the low and high factor levels, respectively. The steeper the slope of the line the larger the difference effect sizes.

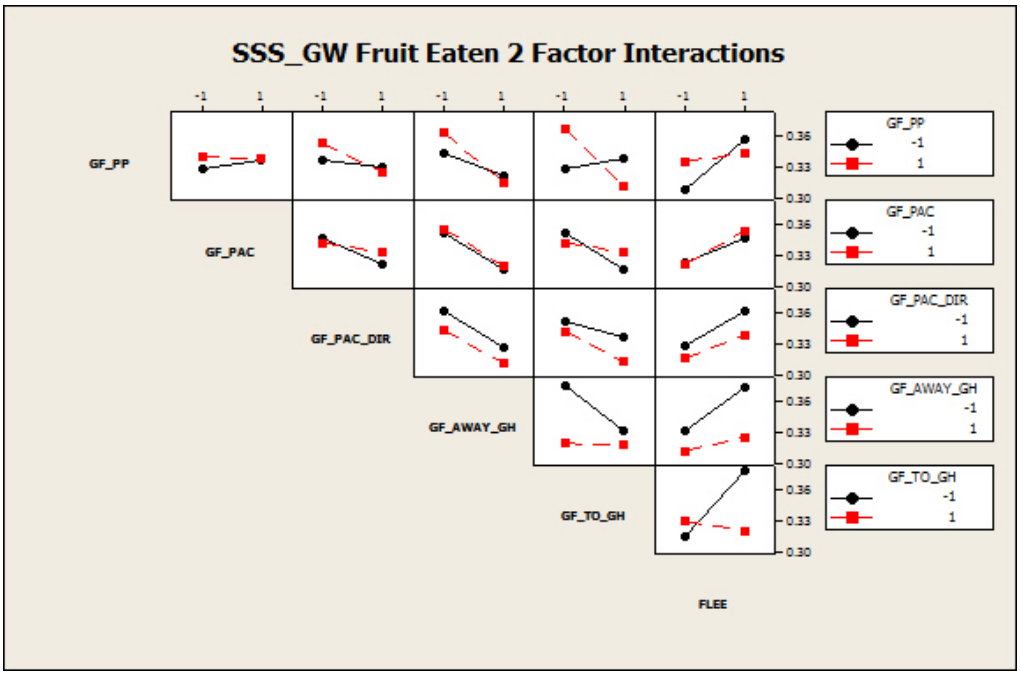


Figure 5.22: Demonstrating the SSS_GW significant 2-factor interactions for the response variable fruit eaten. Parallel lines indicate that no interaction between the two factors. Intersecting lines indicate that an interaction did occur.

5.3.2.2.3 SSS_GW Ghosts Eaten Terms

Figure 5.23 illustrates the statistically significant main effects of SSS_GW model for the response variable ghosts eaten. The largest positive main effect occurred from the GH_VIS_LEN factor, and the largest negative main effect occurred from factor GW_AWAY_GH. As expected, the GW_TO_GH played a statistically significant role for this response as moving fleeing ghosts towards each other would allow for a greater number to be eaten in a short span of time. This cases contained a large number of statistically significant main effects and 2-factor interactions, only the GW_FT factor was not significant.

Figure 5.24 illustrates the 2-factor interactions of the SSS_GW model for the response variable ghosts eaten. The figure includes only factors which contributed significantly to the interaction of at least one other factor. If the lines intersect in a cell it indicates that an interaction occurred between the factors. If parallel lines occur in a cell it indicates that the factors do not interact. The majority of the interactions had negative effects, the largest negative interaction occurred as a result of the FLEE_TIME and GH_VIS_LEN.

A full list of the statistically significant main effects and 2-factor interactions for the SSS_GW ghosts eaten model can be found in Appendix A.1. The lack-of-fit value for this value was $p = 0.00$, which indicates that terms omitted by this model played a significant role in the outcome of the experiment. In this case the 127 maximum number of terms limited the analysis of the model, and the statistical significance of each term was recalculated independent of the term limitation.

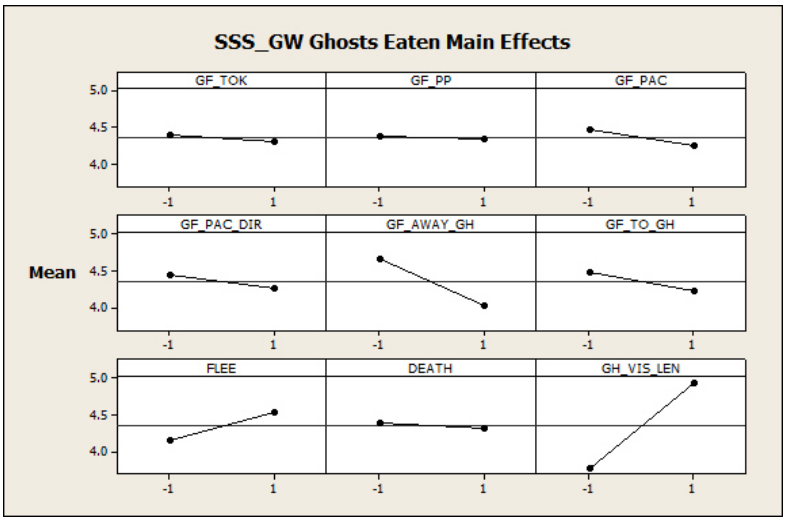


Figure 5.23: Demonstrating the SSS_GW significant main effects for the response variable ghosts eaten. The coded values -1 and 1 represent the low and high factor levels, respectively. The steeper the slope of the line the larger the difference effect sizes.

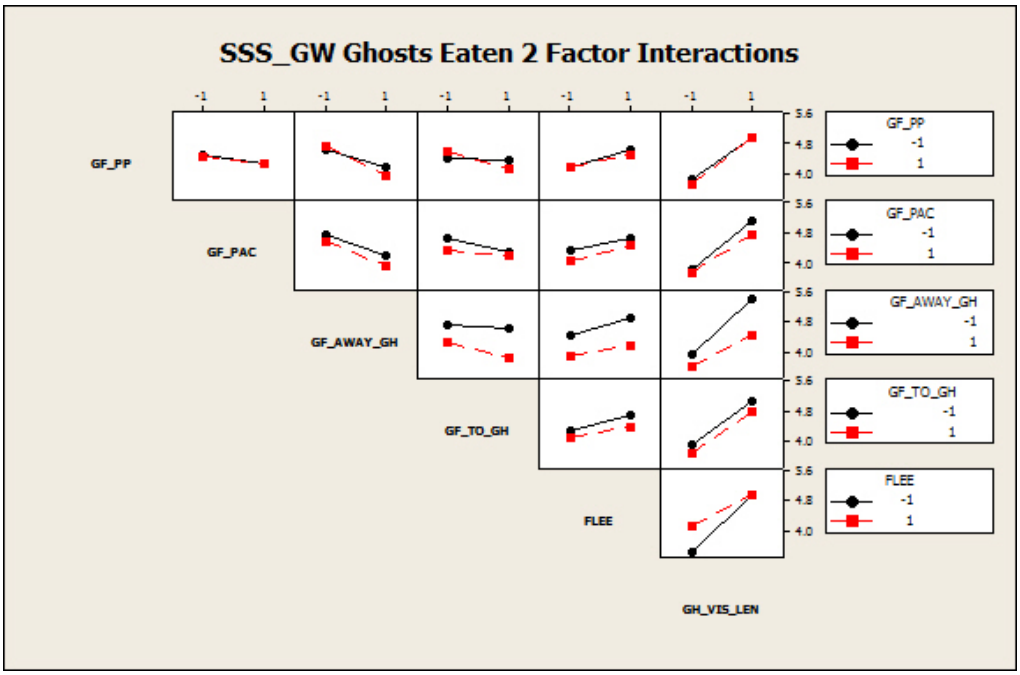


Figure 5.24: Demonstrating the SSS_GW significant 2-factor interactions for the response variable ghosts eaten. Parallel lines indicate that no interaction between the two factors. Intersecting lines indicate that an interaction did occur.

5.3.2.2.4 SSS_GW Levels Completed Terms

Figure 5.25 illustrates the statistically significant main effects of SSS_GW model for the response variable levels completed. The two largest main effect were both negative, they were the GW_AWAY_GH and GH_VIS_LEN. The largest positive main effects were the FLEE_TIME and DEATH_TIME indicating the additional predator time is being used to complete level tasks in addition to the bonus tasks shown in the previous section.

Figure 5.26 illustrates the 2-factor interactions of the SSS_GW model for the response variable levels completed. The figure includes only factors which contributed significantly to the interaction of at least one other factor. If the lines intersect in a cell it indicates that an interaction occurred between the factors. If parallel lines occur in a cell it indicates that the factors do not interact. The factors with the largest statistically significant effects are interactions include the GW_TO_GH and DEATH_TIME factors and the FLEE_TIME and DEATH_TIME. In addition, the interaction of the GW_PP and GH_VIS_LEN produces a negative effective on the number of levels completed.

A full list of the statistically significant main effects and 2-factor interactions for the SSS_GW levels completed model can be found in Appendix A.1. The lack-of-fit value for this value was $p = 0.00$, which indicates that terms omitted by this model played a significant role in the outcome of the experiment. In this case the 127 maximum number of terms limited the analysis of the model, and the statistical significance of each term was recalculated independent of the term limitation.

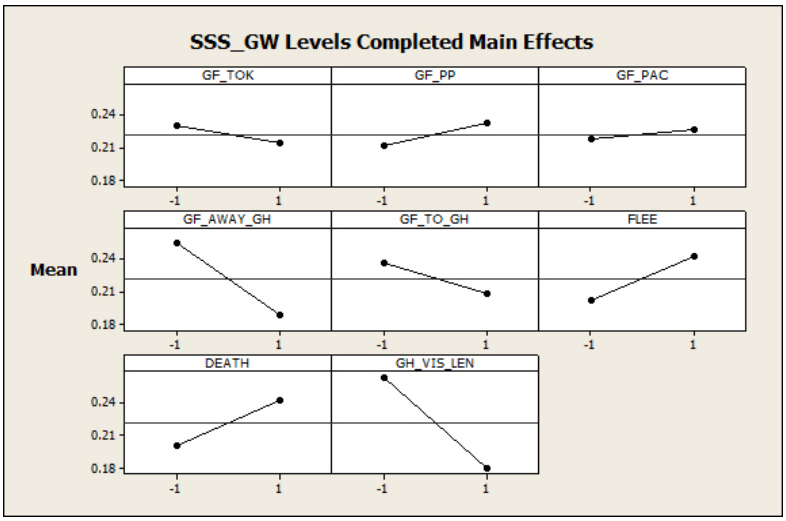


Figure 5.25: Demonstrating the SSS_GW significant main effects for the response variable levels completed. The coded values -1 and 1 represent the low and high factor levels, respectively. The steeper the slope of the line the larger the difference effect sizes.

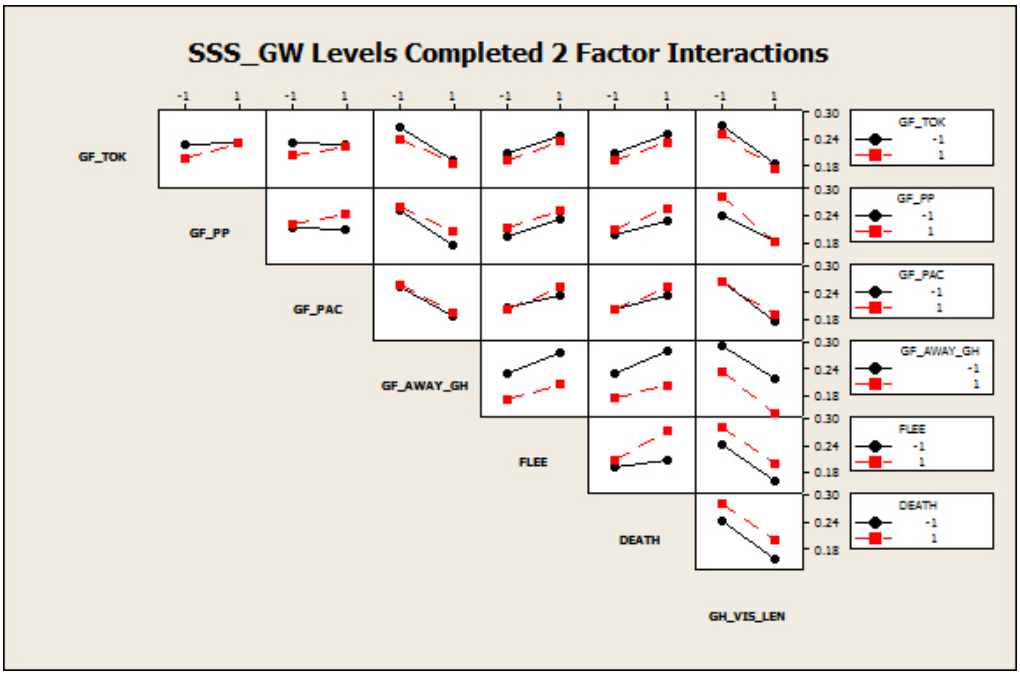


Figure 5.26: Demonstrating the SSS_GW significant 2-factor interactions for the response variable levels completed. Parallel lines indicate that no interaction between the two factors. Intersecting lines indicate that an interaction did occur.

5.3.2.2.5 SSS_GW Power-Pellets Collected Terms

Figure 5.27 illustrates the statistically significant main effects of SSS_GW model for the response variable power-pellets collected. The largest main effect was the GH_VIS_LEN. The largest positive main effect was the FLEE_TIME factor.

Figure 5.28 illustrates the 2-factor interactions of the SSS_GW model for the response variable power-pellets collected. The figure includes only factors which contributed significantly to the interaction of at least one other factor. If the lines intersect in a cell it indicates that an interaction occurred between the factors. If parallel lines occur in a cell it indicates that the factors do not interact. The largest 2-factor interaction occurred as a negative effect of the GW_PP and GH_VIS_LEN interaction. Unlike the previously discussed models the 2-factor interactions effect size where similar in size for quite a few interactions.

A full list of the statistically significant main effects and 2-factor interactions for the SSS_GW power-pellets collected model can be found in Appendix A.1. The lack-of-fit value for this value was $p = 0.00$, which indicates that terms omitted by this model played a significant role in the outcome of the experiment. In this case the 127 maximum number of terms limited the analysis of the model, and the statistical significance of each term was recalculated independent of the term limitation.

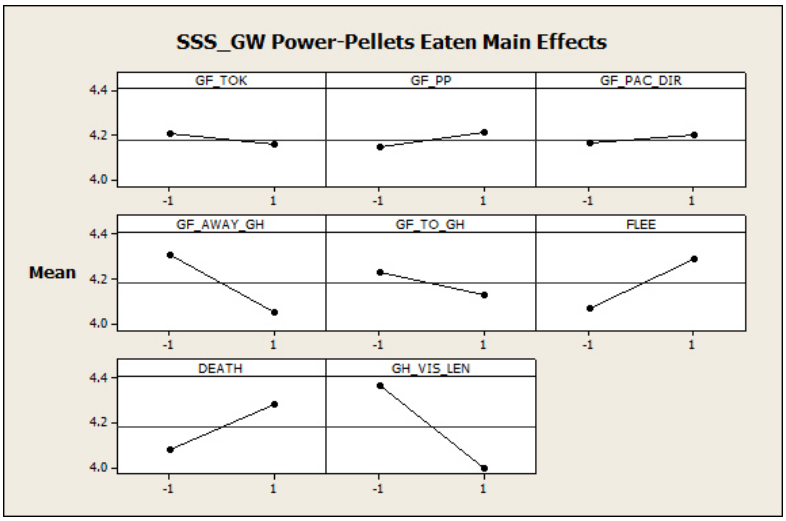


Figure 5.27: Demonstrating the SSS_GW significant main effects for the response variable Power-Pellets. The coded values -1 and 1 represent the low and high factor levels, respectively. The steeper the slope of the line the larger the difference effect sizes.

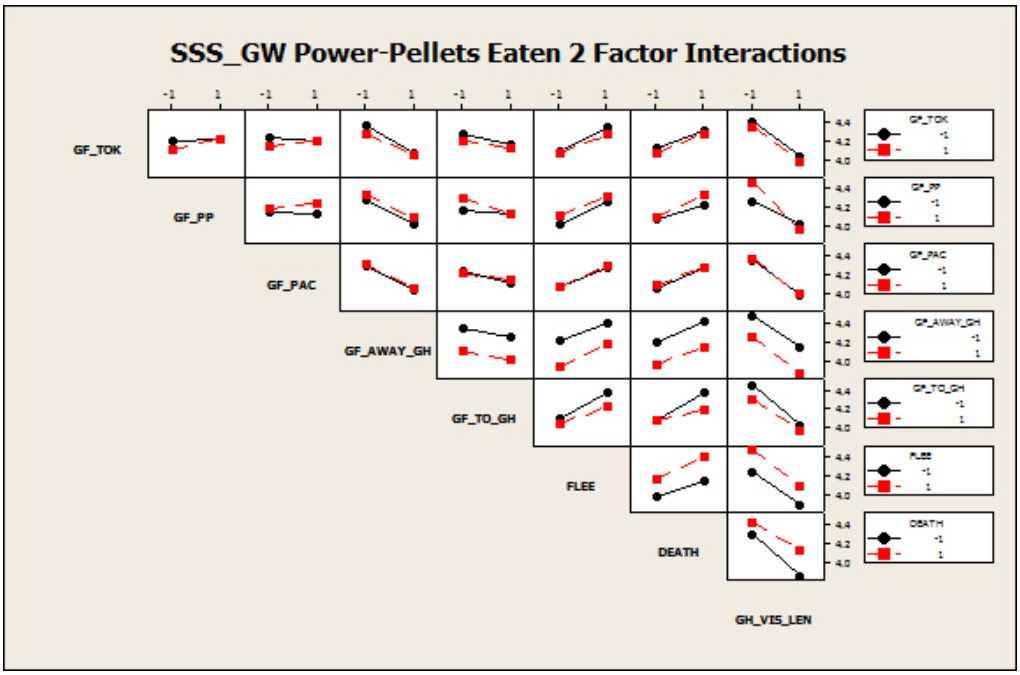


Figure 5.28: Demonstrating the SSS_GW significant 2-factor interactions for the response variable Power-Pellets. Parallel lines indicate that no interaction between the two factors. Intersecting lines indicate that an interaction did occur.

5.3.2.2.6 SSS_GW Repeated Squares Terms

Figure 5.29 illustrates the statistically significant main effects of SSS_GW model for the response variable repeated squares. The largest negative main effect was the GH_VIS_LEN. The largest positive main effect was the FLEE_TIME factor. Interestingly, this is one of the few cases where the GW_FT main effect is statistically significant. The GW_FT indicates a priority to protect the fruit for the ghosts.

Figure 5.30 illustrates the 2-factor interactions of the SSS_GW model for the response variable repeated squares. The figure includes only factors which contributed significantly to the interaction of at least one other factor. If the lines intersect in a cell it indicates that an interaction occurred between the factors. If parallel lines occur in a cell it indicates that the factors do not interact. The largest positive 2-factor interaction occurred from the FLEE_TIME and DEATH_TIME factors. The interaction of GW_PAC_DIR and GH_VIS_LEN produced one of the highest negative interactions. This indicates improved vision and focus on Pac-Man's direction can effectively decrease Pac-Man's life span and repeated squares.

A full list of the statistically significant main effects and 2-factor interactions for the SSS_GW repeated squares model can be found in Appendix A.1. The lack-of-fit value for this value was $p = 0.00$, which indicates that terms omitted by this model played a significant role in the outcome of the experiment. In this case the 127 maximum number of terms limited the analysis of the model, and the statistical significance of each term was recalculated independent of the term limitation.

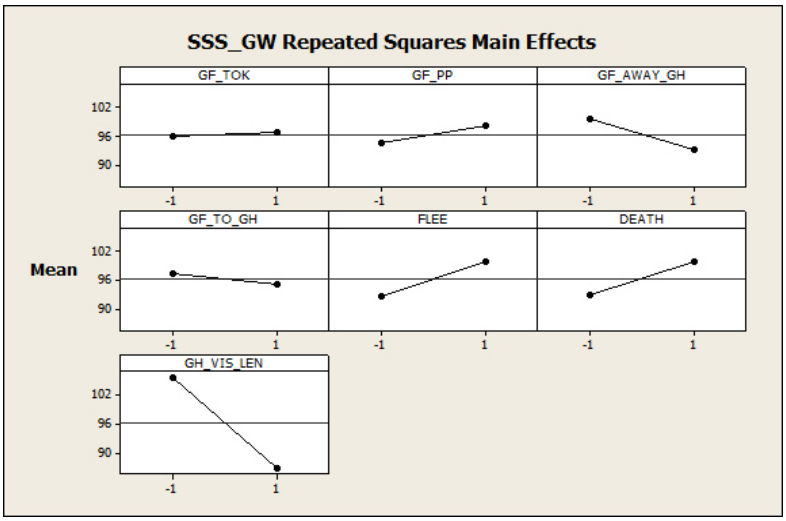


Figure 5.29: Demonstrating the SSS_GW significant main effects for the response variable repeated squares. The coded values -1 and 1 represent the low and high factor levels, respectively. The steeper the slope of the line the larger the difference effect sizes.

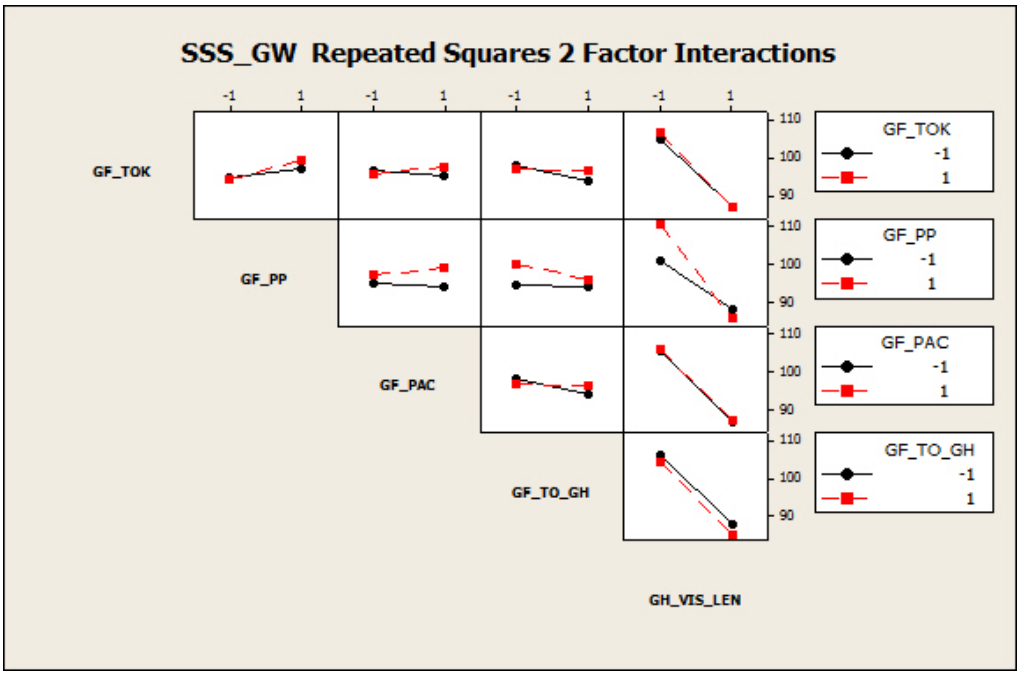


Figure 5.30: Demonstrating the SSS_GW significant 2-factor interactions for the response variable repeated squares. Parallel lines indicate that no interaction between the two factors. Intersecting lines indicate that an interaction did occur.

5.3.2.2.7 SSS_GW Score Terms

Figure 5.31 illustrates the statistically significant main effects of SSS_GW model for the response variable score. The largest negative main effect was occurred from the factor GW_AWAY_GH. The largest positive main effect was the FLEE_TIME factor. This was one of the first cases where the ghosts' vision range was not one of the largest effect sizes, although the ghosts' vision range is still statistically significant for this case.

Figure 5.32 illustrates the 2-factor interactions of the SSS_GW model for the response variable score. The figure includes only factors which contributed significantly to the interaction of at least one other factor. If the lines intersect in a cell it indicates that an interaction occurred between the factors. If parallel lines occur in a cell it indicates that the factors do not interact. The largest positive 2-factor interaction occurred from the FLEE_TIME and DEATH_TIME factors. The largest negative effect occurred as a result of GW_AWAY_GH and GH_VIS_LEN. The interaction of the GW_PP and GW_TO_GH produced a highly negative effect. This result is potentially caused by one of two scenarios, the first scenario is Pac-Man is unable to obtain the power-pellets because the ghosts are clustered tightly protecting the power-pellets. The second scenario involves Pac-Man successfully obtaining the power-pellet but being killed by ghost's turning from prey to predator.

A full list of the statistically significant main effects and 2-factor interactions for the SSS_GW score model can be found in Appendix A.1. The lack-of-fit value for this value was $p = 0.00$, which indicates that terms omitted by this model played a significant role in the outcome of the experiment. In this case the 127 maximum number of terms limited the analysis of the model, and the statistical significance of each term was recalculated independent of the term limitation.

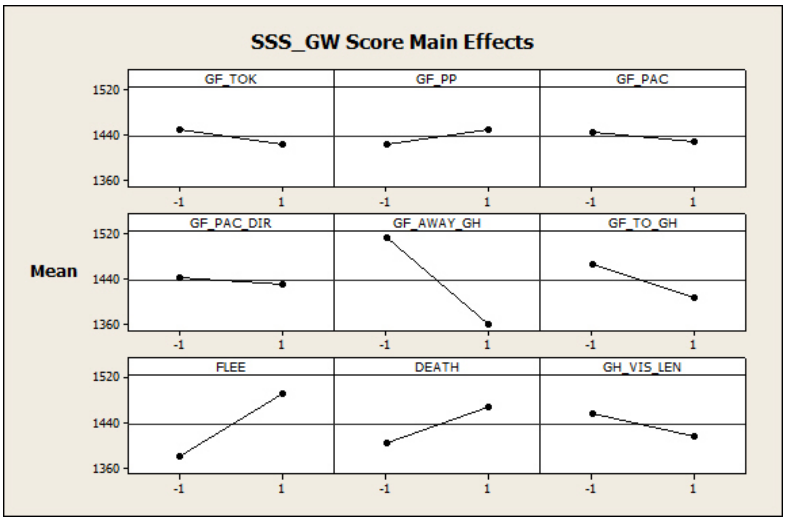


Figure 5.31: Demonstrating the SSS_GW significant main effects for the response variable score. The coded values -1 and 1 represent the low and high factor levels, respectively. The steeper the slope of the line the larger the difference effect sizes.

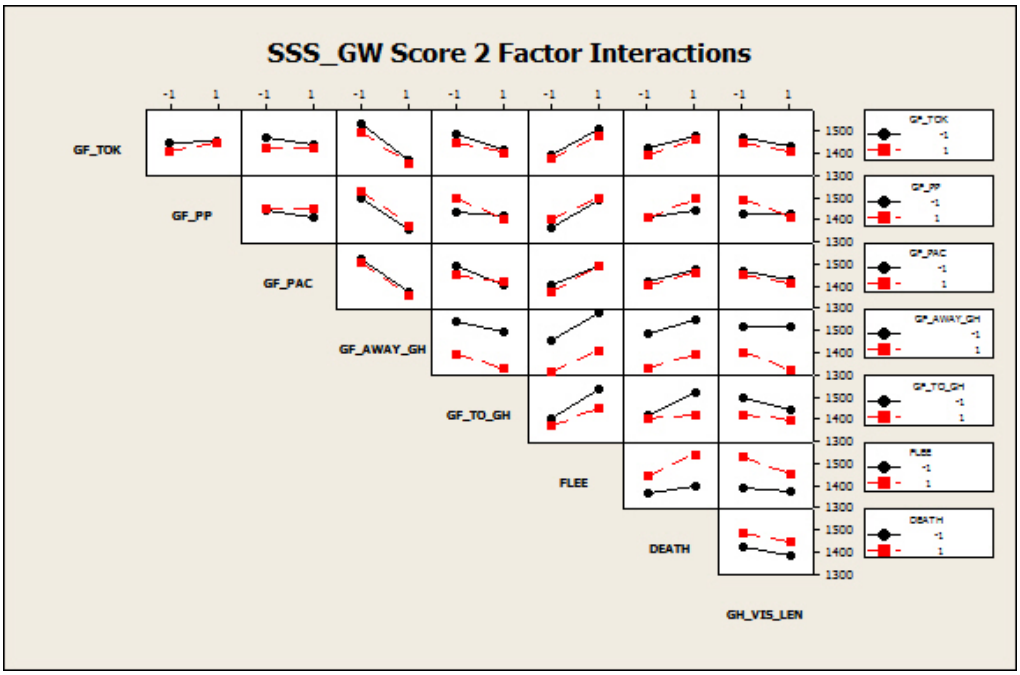


Figure 5.32: Demonstrating the SSS_GW significant 2-factor interactions for the response variable score. Parallel lines indicate that no interaction between the two factors. Intersecting lines indicate that an interaction did occur.

5.3.2.2.8 SSS_GW Steps Terms

Figure 5.33 illustrates the statistically significant main effects of SSS_GW model for the response variable steps. The largest statistically significant main effects where similar to other models. The largest negative main effects where the GH_VIS_LEN and GW_AWAY_GH. The largest positive main effects where the FLEE_TIME and DEATH_TIME. Interestingly, this is one of the few cases where the GW_FT main effect is statistically significant. The GW_FT indicates a priority to protect the fruit for the ghosts.

Figure 5.34 illustrates the 2-factor interactions of the SSS_GW model for the response variable steps. The figure includes only factors which contributed significantly to the interaction of at least one other factor. If the lines intersect in a cell it indicates that an interaction occurred between the factors. If parallel lines occur in a cell it indicates that the factors do not interact. Aside from the interactions of the four largest main effects, we observe that GW_TO_GH, GW_PP, GW_PAC and GW_PAC_DIR factors produce the next set of largest interactions.

A full list of the statistically significant main effects and 2-factor interactions for the SSS_GW steps model can be found in Appendix A.1. The lack-of-fit value for this value was $p = 0.00$, which indicates that terms omitted by this model played a significant role in the outcome of the experiment. In this case the 127 maximum number of terms limited the analysis of the model, and the statistical significance of each term was recalculated independent of the term limitation.

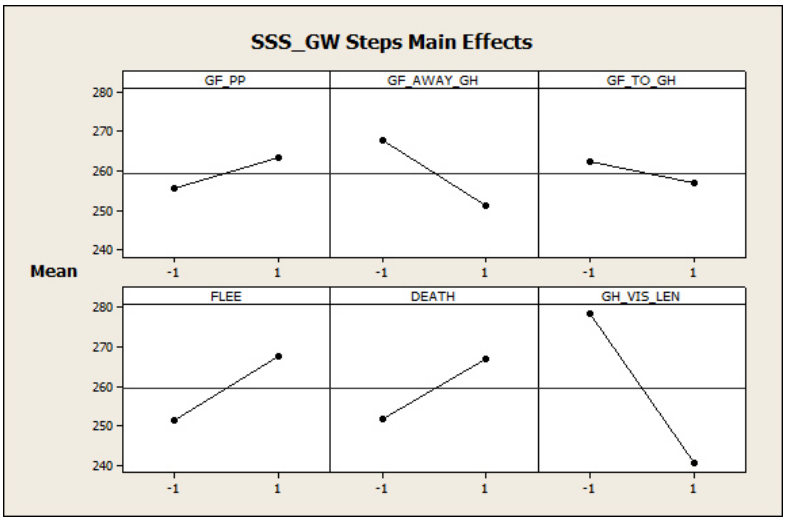


Figure 5.33: Demonstrating the SSS_GW significant main effects for the response variable steps. The coded values -1 and 1 represent the low and high factor levels, respectively. The steeper the slope of the line the larger the difference effect sizes.

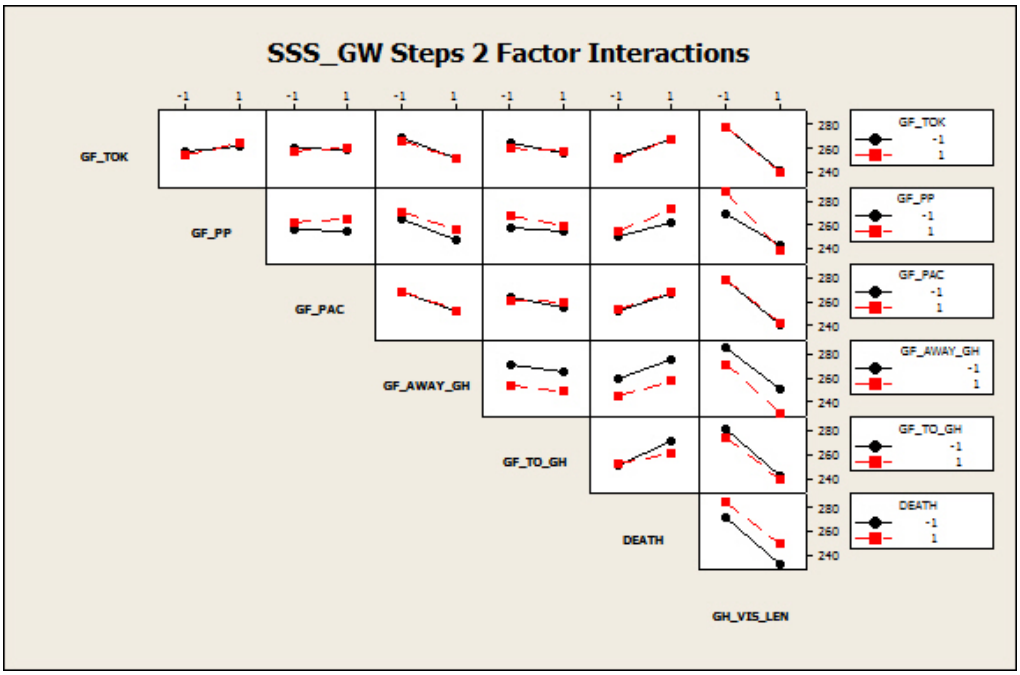


Figure 5.34: Demonstrating the SSS_GW significant 2-factor interactions for the response variable steps. Parallel lines indicate that no interaction between the two factors. Intersecting lines indicate that an interaction did occur.

5.3.2.2.9 SSS_GW Tokens Collected Terms

Figure 5.35 illustrates the statistically significant main effects of SSS_GW model for the response variable steps. The largest statistically significant main effects where similar to other models. The largest negative main effect was the GH_VIS_LEN. The largest positive main effect the FLEE_TIME. For this case only 7 of the main effects were statistically significant.

Figure 5.36 illustrates the 2-factor interactions of the SSS_GW model for the response variable steps. The figure includes only factors which contributed significantly to the interaction of at least one other factor. If the lines intersect in a cell it indicates that an interaction occurred between the factors. If parallel lines occur in a cell it indicates that the factors do not interact. Aside from the interactions of the four largest main effects, we observe that GW_TO_GH, GW_PP, GW_PAC and GW_PAC_DIR factors produce the next set of largest interactions.

A full list of the statistically significant main effects and 2-factor interactions for the SSS_GW steps model can be found in Appendix A.1. The lack-of-fit value for this value was $p = 0.00$, which indicates that terms omitted by this model played a significant role in the outcome of the experiment. In this case the 127 maximum number of terms limited the analysis of the model, and the statistical significance of each term was recalculated independent of the term limitation.

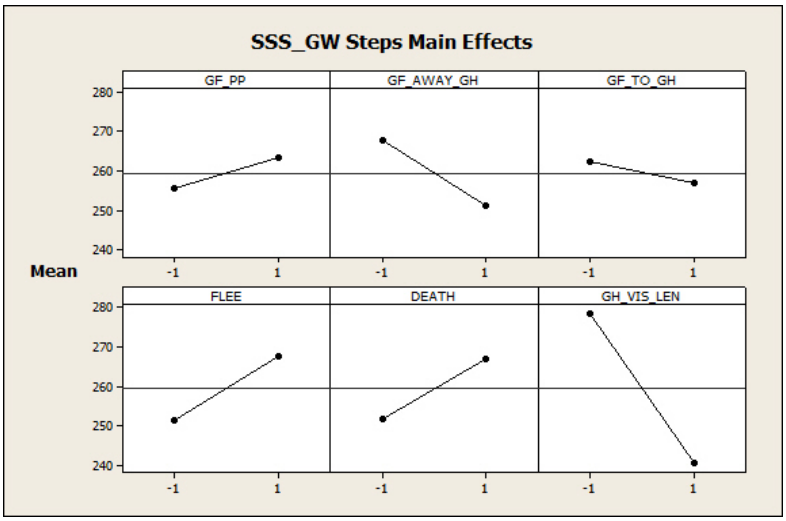


Figure 5.35: Demonstrating the SSS_GW significant main effects for the response variable steps. The coded values -1 and 1 represent the low and high factor levels, respectively. The steeper the slope of the line the larger the difference effect sizes.

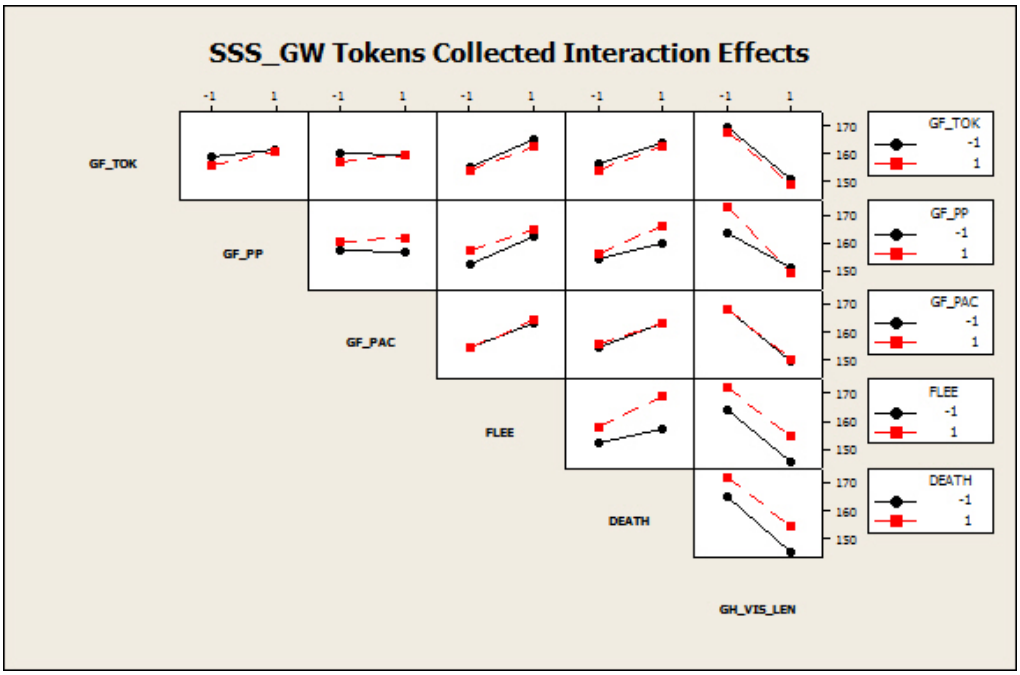


Figure 5.36: Demonstrating the SSS_GW significant 2-factor interactions for the response variable tokens. Parallel lines indicate that no interaction between the two factors. Intersecting lines indicate that an interaction did occur.

5.3.2.2.10 SSS_GW Summary

The results presented in this section showed a significant lack-of-fit for all models. This indicates statistically significant terms were excluded from the model. In addition, it indicates that our models have to the potential for higher explanation of variation.

The results from the SSS_GW response variables found in Appendix A.2 consistently indicate that the ghost vision range, the length of death time, the length of flee time and the factor to move away from other ghosts (GW_AWAY_GH) are the largest contributors in terms of effect sizes. The length of flee and death state time factors for ghosts are the largest contributors for the positive effects, while ghost vision range and the force to move away from other ghosts (GW_AWAY_GH) are the largest and most consistent among the negative effects. As main effects the factors GW_TOKEN, GW_TO_GH and GW_PAC tend to be the next most influential factors, which indicate that protecting tokens, supporting other ghosts and chasing Pac-Man are the most effective portions of the ghost strategies in this situation.

The factor GW_TOKEN plays a significant negative role in terms of the number of levels completed. We attribute this to an increase in difficulty as Pac-Man collects tokens. As this collection takes place, the number of areas to protect decreases and so the ghosts can focus on centralizing their defensive strategy to protect specific areas. The factor GW_TO_GH becomes a prominent factor for the number of ghosts eaten response variable due to the fact that ghosts which group closer together can be eaten by Pac-Man more quickly. As well, larger groups attract Pac-Man's attention over smaller separated groups. An interesting note for comparing factor GW_PAC and GW_PAC_DIR which represent Pac-Man's current position and Pac-Man's assumed next position given direction is that despite the close proximity of the two positions the effects size can greatly differ between response variables. The GW_PAC_DIR main effect is statistically significant for a greater number of response variables. In addition, the effect size for the the GW_PAC_DIR is much larger than the GW_PAC effect size for the response variable close calls. This indicates that guessing Pac-Man's next position serves to slightly increase the chasing and capturing capabilities of the

ghosts. The GW_PAC_DIR allows slightly decreases the number of ghosts eaten and the overall score of the player.

5.3.2.3 PW_FLOCK Terms

The next algorithm pair we evaluated was the PW_FLOCK, we separated the data based on seven factors the first was the frequency of fruit creation followed by six factors related to the Pac-Man weighted heuristics algorithm which were the priority of the: token, power-pellet, edible ghosts , moving away from predator ghosts, moving away from the center of the predator ghosts and toward the center of the items. Unlike the previous sections which utilized factorial plots graphics to further elaborate the experiment results, the results from this section and the following section contain too many experiment runs to be calculate the factorial plots graphics. Instead, this section will simply present tables of the main effects and the 2-factor interactions. The terms statistical significance was calculated based on 95% confidence or $\alpha = 0.05$. Section 5.3.4.3 contains further information pertaining to the confidence intervals for the PW_FLOCK algorithm.

5.3.2.3.1 PW_FLOCK Close Calls Terms

Table 5.7 illustrates the PW_FLOCK model statistically significant main effects and 2-factor interactions for the response variable close calls. A * indicates main effects which are not statistically significant to the experiment results. Main effects which are not statistically significant have been included in the table because one of their interactions is significant. The largest positive main effect occurred from the factor FLOCK_HUNGER. The largest negative main effect is GH_VIS_LEN. Unlike previous algorithms the DEATH_TIME factor is not a significant main effect, although it does contribute to other significant interactions. The largest interactions are the result of the FLOCK_SEP and FLOCK_HUNGER and FLOCK_HUNGER and GH_VIS_LEN, both of which are positive effects.

Main Effect	Effect Size	2-Factor Interaction	Effect Size
PW_FRT (A)	-0.33	A*I	-0.45
FLOCK_SEP (B)	-0.79	B*C	1.36
FLOCK_ALI(C)	-4.34	B*D	1.36
FLOCK_COH (D)	-4.34	B*E	9.84
FLOCK_HUNGER(E)	13.16	B*G	-0.54
FRT_TIME(F)	*	B*I	0.73
FLEE_TIME(G)	-4.08	B*J	1.28
DEATH_TIME (H)	*	C*D	-0.40
PAC_VIS_LEN (I)	4.76	C*E	-1.17
GH_VIS_LEN (J)	-6.97	C*I	-0.68
		C*J	-1.49
		D*E	-1.17
		D*I	-0.68
		D*J	-1.49
		E*G	-0.59
		E*I	1.21
		E*J	3.74
		G*H	1.22
		G*I	-2.11
		G*J	1.98
		H*I	-0.51
		H*J	0.55
		I*J	-0.76

Table 5.7: Statistically significant main effects and 2-factor interactions from the PW_FLOCK algorithm for the response variable Close Calls. * indicates terms whose main effect are not statistically significant but contribute to a significant 2-factor interaction.

5.3.2.3.2 PW_FLOCK Fruits Collected Terms

Table 5.8 illustrates the PW_FLOCK model statistically significant main effects and 2-factor interactions for the response variable fruits collected. A * indicates main effects which are not statistically significant to the experiment results. Main effects which are not statistically significant have been included in the table because one of their interactions is significant. The largest positive main effect is the FRUIT_TIME. The largest negative main effect is GH_VIS_LEN. Similar to the previous PW_FLOCK response variable the DEATH_TIME main effect was not significant, however in this case DEATH_TIME only interacts with FLEE_TIME. In addition, the PW_FRUIT_FOR factor that emphasis collecting the fruit does

produce one of the largest positive main effects. The significant 2-factor interactions of this case produced similar results for all interactions.

Main Effect	Effect Size	2-Factor Interaction	Effect Size
PW_FRT (A)	0.05	A*F	0.01
FLOCK_SEP (B)	*	A*G	0.01
FLOCK_ALI(C)	0.01	A*I	0.01
FLOCK_COH (D)	0.01	A*J	-0.01
FLOCK_HUNGER(E)	-0.01	B*E	0.01
FRT_TIME(F)	0.12	B*J	0.01
FLEE_TIME(G)	0.02	C*D	-0.01
DEATH_TIME (H)	*	C*F	0.01
PAC_VIS_LEN (I)	0.03	C*I	-0.01
GH_VIS_LEN (J)	-0.04	C*J	-0.01
		D*F	0.01
		D*I	-0.01
		D*J	-0.01
		E*G	-0.01
		E*I	0.01
		F*G	0.01
		F*I	0.01
		F*J	-0.01
		G*H	0.01
		G*I	-0.01
		G*J	0.01
		I*J	-0.01

Table 5.8: Statistically significant main effects and 2-factor interactions from the PW_FLOCK algorithm for the response variable fruit collected. * indicates terms whose main effect are not statistically significant but contribute to a significant 2-factor interaction.

5.3.2.3.3 PW_FLOCK Ghosts Eaten Terms

Table 5.9 illustrates the PW_FLOCK model statistically significant main effects and 2-factor interactions for the response variable ghosts eaten. A * indicates main effects which are not statistically significant to the experiment results. Main effects which are not statistically significant have been included in the table because one of their interactions is significant. The largest positive main effect is the FLEE_TIME. The largest negative main effect is FLOCK_HUNGER. Similar to the previous response variable, the DEATH_TIME main effect was not significant, however in this

case DEATH_TIME only interacts with the FLOCK_SEP and FLOCK_HUNGER.

Main Effect	Effect Size	2-Factor Interaction	Effect Size
PW_FRT (A)	*	A*B	0.01
FLOCK_SEP (B)	0.04	A*G	-0.01
FLOCK_ALI(C)	0.01	A*J	-0.01
FLOCK_COH (D)	0.01	B*C	-0.01
FLOCK_HUNGER(E)	-0.05	B*D	-0.01
FRT_TIME(F)	*	B*E	0.01
FLEE_TIME(G)	0.33	B*H	0.01
DEATH_TIME (H)	*	B*I	0.02
PAC_VIS_LEN (I)	0.06	B*J	0.04
GH_VIS_LEN (J)	-0.03	C*D	0.01
		C*I	-0.01
		C*J	-0.01
		D*I	-0.01
		D*J	-0.01
		E*G	-0.02
		E*H	-0.01
		E*J	-0.01
		G*I	-0.01
		G*J	-0.03
		I*J	0.01

Table 5.9: Statistically significant main effects and 2-factor interactions from the PW_FLOCK algorithm for the response variable ghosts eaten. * indicates terms whose main effect are not statistically significant but contribute to a significant 2-factor interaction.

5.3.2.3.4 PW_FLOCK Levels Completed Terms

Table 5.10 illustrates the PW_FLOCK model statistically significant main effects and 2-factor interactions for the response variable levels completed. A * indicates main effects which are not statistically significant to the experiment results. Main effects which are not statistically significant have been included in the table because one of their interactions is significant. The largest positive main effect is the PAC_VIS_LEN. The largest negative main effect is GH_VIS_LEN. The interactions for factors all showed to be similar in response.

Main Effect	Effect Size	2-Factor Interaction	Effect Size
PW_FRUIT_FOR (A)	*	B*C	-0.01
FLOCK_SEP (B)	*	B*D	-0.01
FLOCK_ALI (C)	0.01	B*E	0.01
FLOCK_COH (D)	0.01	B*G	-0.01
FLOCK_HUNGER (E)	*	B*I	0.01
FRUIT_TIME (F)	*	B*J	0.01
FLEE_TIME (G)	0.02	C*D	-0.01
DEATH_TIME (H)	0.01	C*J	-0.01
PAC_VIS_LEN (I)	0.04	D*J	-0.01
GH_VIS_LEN (J)	-0.04	E*H	-0.01
		E*J	0.01
		F*J	0.01
		G*H	0.01
		G*J	0.01
		H*I	-0.01
		I*J	-0.01

Table 5.10: Statistically significant main effects and 2-factor interactions from the PW_FLOCK algorithm for the response variable levels completed. * indicates terms whose main effect are not statistically significant but contribute to a significant 2-factor interaction.

5.3.2.3.5 PW_FLOCK Power-Pellets Collected Terms

Table 5.11 illustrates the PW_FLOCK model statistically significant main effects and 2-factor interactions for the response variable power-pellets collected. A * indicates main effects which are not statistically significant to the experiment results. Main effects which are not statistically significant have been included in the table because one of their interactions is significant. The largest positive main effect is the PAC_VIS_LEN. The largest negative main effect is GH_VIS_LEN. The FLEE_TIME main effect was one of the larger positive effects, indicating that the additional flee time was aiding Pac-Man get to the next power-pellet. The largest 2-factor interactions are positive and occur as a result of the flocking algorithms separation and hunger along with improving the ghosts vision range. A potential explanation of this results is that the flock maintained a tight formation which made some areas of the board to dangerous for Pac-Man. With improved vision and separation the ghosts would spread out a bit more to chase Pac-Man.

Main Effect	Effect Size	2-Factor Interaction	Effect Size
PW_FRUIT_FOR (A)	*	B*C	-0.01
FLOCK_SEP (B)	-0.01	B*D	-0.01
FLOCK_ALI (C)	0.01	B*E	0.04
FLOCK_COH (D)	0.01	B*F	0.01
FLOCK_HUNGER (E)	-0.02	B*G	-0.01
FRUIT_TIME (F)	*	B*I	0.01
FLEE_TIME (G)	0.09	B*J	0.04
DEATH_TIME (H)	0.01	C*E	-0.01
PAC_VIS_LEN (I)	0.2	C*I	-0.01
GH_VIS_LEN (J)	-0.19	C*J	-0.01
		D*E	-0.01
		D*I	-0.01
		D*J	-0.01
		E*H	-0.01
		E*I	0.01
		F*J	0.01
		G*H	0.01
		G*I	-0.01
		G*J	0.01
		H*I	-0.01
		I*J	-0.01

Table 5.11: Statistically significant main effects and 2-factor interactions from the PW_FLOCK algorithm for the response variable power-pellets collected. * indicates terms whose main effect are not statistically significant but contribute to a significant 2-factor interaction.

5.3.2.3.6 PW_FLOCK Repeated Squares Terms

Table 5.12 illustrates the PW_FLOCK model statistically significant main effects and 2-factor interactions for the response variable repeated squares. A * indicates main effects which are not statistically significant to the experiment results. Main effects which are not statistically significant have been included in the table because one of their interactions is significant. The largest positive main effect is the FLEE_TIME. The largest negative main effect is GH_VIS_LEN. Similar to the other cases, the flocking separation and hunger along with the ghosts' vision played an statistically significant role.

Main Effect	Effect Size	2-Factor Interaction	Effect Size
PW_FRUIT_FOR (A)	-0.37	A*I	-0.40
FLOCK_SEP (B)	-0.72	B*C	-1.04
FLOCK_ALI (C)	1.57	B*D	-1.04
FLOCK_COH (D)	1.57	B*E	3.96
FLOCK_HUNGER (E)	-1.13	B*G	-1.40
FRUIT_TIME (F)	*	B*H	0.51
FLEE_TIME (G)	9.35	B*I	0.89
DEATH_TIME (H)	1.24	B*J	3.80
PAC_VIS_LEN (I)	9.02	C*D	-0.41
GH_VIS_LEN (J)	-14.53	C*E	-0.39
		C*I	-1.04
		C*J	-1.15
		D*E	-0.39
		D*I	-1.04
		D*J	-1.15
		E*I	0.33
		E*J	-0.34
		G*H	1.56
		G*I	-1.93
		G*J	1.4
		H*I	-0.98
		I*J	-0.36

Table 5.12: Statistically significant main effects and 2-factor interactions from the PW_FLOCK algorithm for the response variable repeated squares. * indicates terms whose main effect are not statistically significant but contribute to a significant 2-factor interaction.

5.3.2.3.7 PW_FLOCK Score Terms

Table 5.13 illustrates the PW_FLOCK model statistically significant main effects and 2-factor interactions for the response variable score. A * indicates main effects which are not statistically significant to the experiment results. Main effects which are not statistically significant have been included in the table because one of their interactions is significant. The largest positive main effect is the PAC_VIS_LEN. The largest negative main effect is GH_VIS_LEN. The FLEE_TIME is another of the largest main effects. In addition, we observe that the FRUIT_TIME is a large positive main effect indicating that Pac-Man is making the fruit a priority in the game. Similar to the other cases, the flocking separation and hunger along with the ghosts' vision played an statistically significant role.

Main Effect	Effect Size	2-Factor Interaction	Effect Size
PW_FRUIT_FOR (A)	6.53	A*F	2.18
FLOCK_SEP (B)	4.51	B*C	-5.76
FLOCK_ALI (C)	7.17	B*D	-5.76
FLOCK_COH (D)	7.17	B*E	20.61
FLOCK_HUNGER (E)	-9.35	B*G	-7.34
FRUIT_TIME (F)	18.63	B*I	5.52
FLEE_TIME (G)	76.13	B*J	22.88
DEATH_TIME (H)	5.28	C*D	-4.58
PAC_VIS_LEN (I)	84.54	C*I	-2.51
GH_VIS_LEN (J)	-74.46	C*J	-9.71
		D*I	-2.51
		D*J	-9.71
		E*G	-2.26
		E*H	-3.25
		E*I	1.94
		F*G	2.50
		F*I	2.53
		G*H	7.72
		G*I	-5.63
		I*J	-7.72

Table 5.13: Statistically significant main effects and 2-factor interactions from the PW_FLOCK algorithm for the response variable score. * indicates terms whose main effect are not statistically significant but contribute to a significant 2-factor interaction.

5.3.2.3.8 PW_FLOCK Steps Terms

Table 5.14 illustrates the PW_FLOCK model statistically significant main effects and 2-factor interactions for the response variable steps. A * indicates main effects which are not statistically significant to the experiment results. Main effects which are not statistically significant have been included in the table because one of their interactions is significant. The largest positive main effect is the PAC_VIS_LEN. The largest negative main effect is GH_VIS_LEN. Similar to the other cases, the flocking separation and hunger along with the ghosts' vision played an statistically significant role. Also, we observe a fairly large interactions from the FLEE_TIME factor.

5.3.2.3.9 PW_FLOCK Tokens Collected Terms

Table 5.15 illustrates the PW_FLOCK model statistically significant main effects and 2-factor interactions for the response variable tokens. A * indicates main effects which are not statistically significant to the experiment results. Main effects which are not statistically significant have been included in the table because one of their interactions is significant. The largest positive main effect is the PAC_VIS_LEN. The largest negative main effect is GH_VIS_LEN. Similar to the other cases, the flocking separation and hunger along with the ghosts' vision played an statistically significant role.

5.3.2.3.10 PW_FLOCK Summary

The PW_FLOCK algorithm demonstrates a consistent set of factors with large effects, similar to other algorithm results the vision parameters are the reoccurring statistically significant factors. The length of the flee state time factor along with the Pac-Man's vision range continued to produce positive effects. The sole exception for the flee time factor is in the close calls response variable, in which flee time plays an important role in decreasing the number of close calls. As expected the length of time the fruit was available and the player's perceived value of the fruit created the largest effect sizes for the response variable the number of fruit collected.

Main Effect	Effect Size	2-Factor Interaction	Effect Size
PW_FRUIT_FOR (A)	-0.52	A*I	-0.49
FLOCK_SEP (B)	-0.73	B*C	-1.56
FLOCK_ALI (C)	2.33	B*D	-1.56
FLOCK_COH (D)	2.33	B*E	5.88
FLOCK_HUNGER (E)	-1.98	B*F	0.49
FRUIT_TIME (F)	*	B*G	-2.32
FLEE_TIME (G)	14.35	B*I	1.22
DEATH_TIME (H)	1.81	B*J	5.57
PAC_VIS_LEN (I)	17.14	C*D	-0.61
GH_VIS_LEN (J)	-21.50	C*E	-0.49
		C*I	-1.28
		C*J	-1.95
		D*E	-0.49
		D*I	-1.28
		D*J	-1.95
		E*H	-0.58
		E*I	0.60
		E*J	-0.54
		F*J	0.49
		G*H	2.33
		G*I	-2.60
		G*J	1.82
		H*I	-1.17
		I*J	-1.05

Table 5.14: Statistically significant main effects and 2-factor interactions from the PW_FLOCK algorithm for the response variable steps. * indicates terms whose main effect are not statistically significant but contribute to a significant 2-factor interaction.

The most reoccurring significant negative term was the ghost's vision range. The flocking algorithm factors were in most cases statistically significant main effects. The flocking hunger and separation were frequently the largest 2-factor interactions. Also, we observed the factors of the flocking algorithm interacting with the ghosts' vision range, showing improved performance with greater range.

5.3.2.4 PW_GW Terms

The final algorithm pair we evaluated was the PW_GW which contained the highest number of game parameters, as such the organization process required the largest

Main Effect	Effect Size	2-Factor Interaction	Effect Size
PW_FRUIT_FOR (A)	-0.15	B*C	-0.50
FLOCK_SEP (B)	*	B*D	-0.50
FLOCK_ALI (C)	0.73	B*E	1.87
FLOCK_COH (D)	0.73	B*F	0.22
FLOCK_HUNGER (E)	-0.83	B*G	-0.90
FRUIT_TIME (F)	*	B*I	0.31
FLEE_TIME (G)	4.90	B*J	1.73
DEATH_TIME (H)	0.55	C*D	-0.19
PAC_VIS_LEN (I)	7.91	C*I	-0.23
GH_VIS_LEN (J)	-6.76	C*J	-0.78
		D*I	-0.23
		D*J	-0.78
		E*H	-0.29
		E*I	0.25
		E*J	-0.19
		F*J	0.20
		G*H	0.75
		G*I	-0.65
		G*J	0.41
		H*I	-0.19
		I*J	-0.67

Table 5.15: Statistically significant main effects and 2-factor interactions from the PW_FLOCK algorithm for the response variable tokens collected. * indicates terms whose main effect are not statistically significant but contribute to a significant 2-factor interaction.

number of separation factors. This algorithm pair used ten factor separators resulting in 2^{10} or 1024 game combinations. The tables identifying term significance for the PW_GW algorithm appear in Appendix A. Unlike the earlier sections which utilized factorial plots graphics to further elaborate the experiment results, the results from this section and the following section contain too many experiment runs to calculate the factorial plots graphics. Instead, this section will simply present tables of the main effects and the 2-factor interactions. The terms statistical significance was calculated based on 95% confidence or $\alpha = 0.05$. Section A.4 contains further information pertaining to the confidence intervals for the PW_GW algorithm.

5.3.2.4.1 PW_GW Close Call Terms

Table 5.16 illustrates the statistically significant main effects and 2-factor interactions of PW_GW model for the response variable close calls. A * indicates main effects which are not statistically significant to the experiment results. Main effects which are not statistically significant have been included in the table because one of their interactions is significant. The largest positive main effect is the GH_VIS_LEN. The largest negative main effect is FLEE_TIME. The largest interactions occur from the interaction of the factors GW_AWAY_GH, FLEE_TIME, and GH_VIS_LEN.

Main Effect	Effect Size	2-Factor Interaction	Effect Size
GW_TOK (A)	-0.45	A*G	0.35
GW_PP (B)	0.67	A*J	0.58
GW_FT (C)	0.18	B*D	-0.34
GW_PAC (D)	2.68	B*G	-0.33
GW_PAC_DIR (E)	6.15	B*I	0.28
GW_AWAY_GH (F)	-11.29	B*J	0.38
GW_TO_GH (G)	-5.74	D*E	-0.44
FLEE_TIME (H)	-16.73	D*F	1.39
DEATH_TIME (I)	0.71	D*G	1.46
GH_VIS_LEN (J)	12.80	D*H	-1.36
		D*J	2.07
		E*F	-0.62
		E*H	-0.89
		E*I	0.38
		E*J	-0.69
		F*G	0.86
		F*H	3.63
		F*J	-2.46
		G*H	1.52
		G*I	-0.34
		G*J	1.57
		H*J	-4.53
		I*J	0.49

Table 5.16: Statistically significant main effects and 2-factor interactions from the PW_GW algorithm for the response variable Close Calls. * indicates terms whose main effect are not statistically significant but contribute to a significant 2-factor interaction.

5.3.2.4.2 PW_GW Fruits Collected Terms

Table 5.17 illustrates the statistically significant main effects and 2-factor interactions of PW_GW model for the response variable fruits collected. A * indicates main effects which are not statistically significant to the experiment results. Main effects which are not statistically significant have been included in the table because one of their interactions is significant. The largest negative main effect is FLEE_TIME.

Main Effect	Effect Size	2-Factor Interaction	Effect Size
GW_TOK (A)	0.01	A*J	-0.01
GW_PP (B)	0.01	B*C	0.01
GW_FT (C)	-0.01	B*E	0.01
GW_PAC (D)	0.01	B*J	-0.01
GW_PAC_DIR (E)	-0.01	C*G	-0.01
GW_AWAY_GH (F)	-0.01	C*J	0.01
GW_TO_GH (G)	-0.01	D*F	0.01
FLEE_TIME (H)	0.01	D*G	0.01
DEATH_TIME (I)	0.01	D*J	0.01
GH_VIS_LEN (J)	-0.15	E*H	0.01
		E*J	-0.01
		F*G	-0.01
		F*H	0.01
		F*I	-0.01
		F*J	0.01
		G*H	0.01
		G*I	-0.01
		G*J	0.01
		H*I	0.01
		H*J	-0.01
		I*J	-0.01

Table 5.17: Statistically significant main effects and 2-factor interactions from the PW_GW algorithm for the response variable fruit eaten. * indicates terms whose main effect are not statistically significant but contribute to a significant 2-factor interaction.

5.3.2.4.3 PW_GW Ghosts Eaten Terms

Table 5.18 illustrates the statistically significant main effects and 2-factor interactions of PW_GW model for the response variable ghosts eaten. A * indicates main effects which are not statistically significant to the experiment results. Main effects

which are not statistically significant have been included in the table because one of their interactions is significant. The largest positive main effect is the FLEE_TIME. The largest negative main effect is GW_PAC_DIR. The GW_PAC_DIR factor prioritizes moving towards an estimate of Pac-Man's next position. The negative of impact of the GW_PAC_DIR indicates that the ghosts are eating Pac-Man before he can eat them. The largest interactions occur from the interaction of the factors FLEE_TIME and GH_VIS_LEN and GW_PAC and GH_VIS_LEN.

Main Effect	Effect Size	2-Factor Interaction	Effect Size
GW_TOK (A)	*	A*B	0.01
GW_PP (B)	-0.01	A*G	0.01
GW_FT (C)	*	A*H	-0.01
GW_PAC (D)	-0.02	B*E	-0.01
GW_PAC_DIR (E)	-0.15	B*F	-0.01
GW_AWAY_GH (F)	-0.11	B*H	0.01
GW_TO_GH (G)	-0.04	C*J	-0.01
FLEE_TIME (H)	0.40	D*E	0.03
DEATH_TIME (I)	-0.01	D*F	-0.01
GH_VIS_LEN (J)	0.07	D*G	0.01
		D*H	-0.01
		D*I	-0.01
		D*J	0.03
		E*F	0.01
		E*G	0.01
		E*H	-0.03
		E*I	0.01
		E*J	0.02
		F*G	0.01
		F*H	-0.01
		F*I	-0.01
		F*J	-0.03
		G*I	-0.01
		H*J	-0.04

Table 5.18: Statistically significant main effects and 2-factor interactions from the PW_GW algorithm for the response variable ghosts eaten. * indicates terms whose main effect are not statistically significant but contribute to a significant 2-factor interaction.

5.3.2.4.4 PW_GW Levels Completed Terms

Table 5.19 illustrates the statistically significant main effects and 2-factor interactions of PW_GW model for the response variable levels completed. A * indicates main effects which are not statistically significant to the experiment results. Main effects which are not statistically significant have been included in the table because one of their interactions is significant. The largest positive main effect is the FLEE_TIME. The largest negative main effects are GW_AWAY_GH and GH_VIS_LEN. Similar to other cases, the largest interactions occur as a result of FLEE_TIME and GH_VIS_LEN.

Main Effect	Effect Size	2-Factor Interaction	Effect Size
GW_TOK (A)	-0.002	A*J	0.002
GW_PP (B)	0.001	B*F	0.000
GW_FT (C)	*	B*H	0.000
GW_PAC (D)	0.000	B*J	0.000
GW_PAC_DIR (E)	0.003	D*F	0.001
GW_AWAY_GH (F)	-0.013	D*G	0.002
GW_TO_GH (G)	-0.006	D*H	0.000
FLEE_TIME (H)	0.003	D*J	0.001
DEATH_TIME (I)	0.002	E*F	0.000
GH_VIS_LEN (J)	-0.013	E*I	0.000
		E*J	0.000
		F*G	0.000
		F*H	0.002
		F*I	0.000
		F*J	-0.001
		G*I	0.000
		G*J	0.001
		H*I	0.000
		H*J	-0.003

Table 5.19: Statistically significant main effects and 2-factor interactions from the PW_GW algorithm for the response variable levels completed. * indicates terms whose main effect are not statistically significant but contribute to a significant 2-factor interaction.

5.3.2.4.5 PW_GW Power-Pellets Collected Terms

Table 5.20 illustrates the statistically significant main effects and 2-factor interactions of PW_GW model for the response variable power-pellets collected. A *

indicates main effects which are not statistically significant to the experiment results. Main effects which are not statistically significant have been included in the table because one of their interactions is significant. The largest positive main effect is the FLEE_TIME. The largest negative main effect is GH_VIS_LEN. Similar to other cases, the largest interactions occur as a result of FLEE_TIME and GH_VIS_LEN.

Main Effect	Effect Size	2-Factor Interaction	Effect Size
GW_TOK (A)	-0.01	A*B	0.01
GW_PP (B)	0.01	A*H	-0.01
GW_FT (C)	*	A*J	0.01
GW_PAC (D)	0.01	B*E	-0.01
GW_PAC_DIR (E)	0.01	B*F	-0.01
GW_AWAY_GH (F)	-0.05	B*H	0.01
GW_TO_GH (G)	-0.02	B*J	-0.01
FLEE_TIME (H)	0.02	D*F	0.01
DEATH_TIME (I)	0.01	D*G	0.01
GH_VIS_LEN (J)	-0.15	D*H	-0.01
		D*I	-0.01
		D*J	0.01
		E*F	-0.01
		E*I	0.01
		E*J	-0.01
		F*H	0.01
		F*I	-0.01
		G*I	-0.01
		G*J	0.01
		H*I	0.01
		H*J	-0.05
		I*J	0.01

Table 5.20: Statistically significant main effects and 2-factor interactions from the PW_GW algorithm for the response variable power-pellets collected. * indicates terms whose main effect are not statistically significant but contribute to a significant 2-factor interaction.

5.3.2.4.6 PW_GW Repeated Squares Terms

Table 5.21 illustrates the statistically significant main effects and 2-factor interactions of PW_GW model for the response variable repeated squares. A * indicates main effects which are not statistically significant to the experiment results.

Main effects which are not statistically significant have been included in the table because one of their interactions is significant. The largest positive main effect is the FLEE_TIME. The largest negative main effect is GH_VIS_LEN. Similar to other cases, the largest interactions occur as a result of FLEE_TIME and GH_VIS_LEN.

Main Effect	Effect Size	2-Factor Interaction	Effect Size
GW_TOK (A)	1.62	A*B	0.36
GW_PP (B)	3.82	A*G	0.34
GW_FT (C)	-0.25	A*H	0.31
GW_PAC (D)	0.78	A*I	0.22
GW_PAC_DIR (E)	0.65	A*J	-1.36
GW_AWAY_GH (F)	-5.89	B*F	0.52
GW_TO_GH (G)	-3.14	B*H	0.56
FLEE_TIME (H)	8.67	B*I	-0.23
DEATH_TIME (I)	4.16	B*J	-3.20
GH_VIS_LEN (J)	-56.45	C*G	-0.27
		C*J	0.26
		D*E	-0.25
		D*F	1.12
		D*G	0.63
		D*H	-0.31
		D*J	1.03
		E*F	-0.31
		E*H	0.51
		E*J	-0.27
		F*H	1.09
		F*I	-0.54
		F*J	1.98
		G*H	0.42
		G*I	-0.30
		G*J	2.72
		H*I	1.00
		H*J	-5.78
		I*J	-1.54

Table 5.21: Statistically significant main effects and 2-factor interactions from the PW_GW algorithm for the response variable repeated squares. * indicates terms whose main effect are not statistically significant but contribute to a significant 2-factor interaction.

5.3.2.4.7 PW_GW Score Terms

Table 5.22 illustrates the statistically significant main effects and 2-factor interactions of PW_GW model for the response variable repeated squares. A * indicates main effects which are not statistically significant to the experiment results. Main effects which are not statistically significant have been included in the table because one of their interactions is significant. The largest positive main effect is the FLEE_TIME. The largest negative main effect is GH_VIS_LEN. Also, the GW_AWAY_GH main effect has a large negative effect. Similar to other cases, the largest interactions occur as a result of FLEE_TIME and GH_VIS_LEN.

Main Effect	Effect Size	2-Factor Interaction	Effect Size
GW_TOK (A)	*	A*B	1.03
GW_PP (B)	6.40	A*H	-0.99
GW_FT (C)	*	B*F	-2.04
GW_PAC (D)	1.52	B*H	1.89
GW_PAC_DIR (E)	-11.40	B*J	-5.33
GW_AWAY_GH (F)	-34.39	D*E	2.72
GW_TO_GH (G)	-14.89	D*F	3.28
FLEE_TIME (H)	59.26	D*G	3.86
DEATH_TIME (I)	7.90	D*H	-2.85
GH_VIS_LEN (J)	-90.44	D*I	-1.42
		D*J	7.96
		E*F	-1.01
		E*H	-2.99
		E*I	1.49
		F*G	2.78
		F*H	2.88
		F*I	-1.94
		F*J	-1.58
		G*H	1.36
		G*I	-1.63
		G*J	5.95
		H*I	2.25
		H*J	-18.88

Table 5.22: Statistically significant main effects and 2-factor interactions from the PW_GW algorithm for the response variable score. * indicates terms whose main effect are not statistically significant but contribute to a significant 2-factor interaction.

5.3.2.4.8 PW_GW Steps Terms

Table 5.22 illustrates the statistically significant main effects and 2-factor interactions of PW_GW model for the response variable steps. A * indicates main effects which are not statistically significant to the experiment results. Main effects which are not statistically significant have been included in the table because one of their interactions is significant. The largest positive main effect is the FLEE_TIME. The largest negative main effect is GH_VIS_LEN. Also, the GW_AWAY_GH main effect has a large negative effect. We observe a large positive effects from DEATH_TIME. Also, if the ghosts focus on protecting the power-pellets it has a positive effect on the

number of steps. Similar to other cases, the largest interactions occur as a result of FLEE_TIME and GH_VIS_LEN.

Main Effect	Effect Size	2-Factor Interaction	Effect Size
GW_TOK (A)	1.40	A*B	0.43
GW_PP (B)	4.56	A*G	0.37
GW_FT (C)	-0.21	A*J	-1.09
GW_PAC (D)	1.45	B*F	0.45
GW_PAC_DIR (E)	0.95	B*G	-0.25
GW_AWAY_GH (F)	-7.75	B*H	0.67
GW_TO_GH (G)	-3.86	B*J	-3.75
FLEE_TIME (H)	10.93	C*G	-0.30
DEATH_TIME (I)	5.10	D*E	-0.30
GH_VIS_LEN (J)	-65.43	D*F	1.52
		D*G	0.80
		D*H	-0.52
		D*I	-0.33
		D*J	1.73
		E*F	-0.35
		E*H	0.58
		E*J	-0.53
		F*H	1.40
		F*I	-0.70
		F*J	2.17
		G*H	0.56
		G*I	-0.39
		G*J	3.35
		H*I	1.24
		H*J	-7.22
		I*J	-1.33

Table 5.23: Statistically significant main effects and 2-factor interactions from the PW_GW algorithm for the response variable tokens collected. * indicates terms whose main effect are not statistically significant but contribute to a significant 2-factor interaction.

5.3.2.4.9 PW_GW Tokens Collected Terms

Table 5.24 illustrates the statistically significant main effects and 2-factor interactions of PW_GW model for the response variable tokens collected. A * indicates main effects which are not statistically significant to the experiment results. Main effects which are not statistically significant have been included in the table because one of

their interactions is significant. The largest positive main effect is the FLEE_TIME. The largest negative main effect is GH_VIS_LEN. Similar to other cases, the largest interactions occur as a result of FLEE_TIME and GH_VIS_LEN.

Main Effect	Effect Size	2-Factor Interaction	Effect Size
GW_TOK (A)	-0.21	A*H	-0.09
GW_PP (B)	0.73	A*J	0.25
GW_FT (C)	0.04	B*E	-0.08
GW_PAC (D)	0.65	B*H	0.10
GW_PAC_DIR (E)	0.28	B*J	-0.54
GW_AWAY_GH (F)	-1.80	D*F	0.38
GW_TO_GH (G)	-0.69	D*G	0.17
FLEE_TIME (H)	2.23	D*H	-0.20
DEATH_TIME (I)	0.92	D*I	-0.13
GH_VIS_LEN (J)	-8.82	D*J	0.68
		E*J	-0.25
		F*H	0.30
		F*I	-0.15
		F*J	0.18
		G*H	0.14
		G*I	-0.09
		G*J	0.62
		H*I	0.23
		H*J	-1.38
		I*J	0.20

Table 5.24: Statistically significant main effects and 2-factor interactions from the PW_GW algorithm for the response variable tokens collected. * indicates terms whose main effect are not statistically significant but contribute to a significant 2-factor interaction.

5.3.2.4.10 PW_GW Summary

The flee state time is the prominent positive factor in all response variables. For the majority of the response variables it is the leading positive factor. However, for the number of close calls and the number of power-pellets collected, flee time has a negative affect. As previously explained, as Pac-Man's predatorial time increases it causes longer periods where Pac-Man will not receive close calls, likewise as Pac-Man spends longer time periods in a predator mode the number of power-pellets collected can decrease because power-pellets can be saved for strategic play when Pac-Man has

to venture into a danger board area.

The factor GW_PP which emphasizes that ghosts should protect power-pellets when they are close by produces a positive effect in all response variables including the number of power-pellets collected and ghosts eaten. These results indicate a couple of possible scenarios: 1) where Pac-Man is drawn to locations with a power-pellet during the predatorial phase due to the large number of fleeing ghosts surrounding the area or 2) that Pac-Man is capable of avoiding the protecting ghosts which requires collecting the power-pellet. The final reoccurring positive factor is the state variable length of death time which creates positive results in the number of close calls and the number of levels completed.

As expected, the ghost's vision range factor plays a prominent negative roll in all response variables except the number of close calls and ghosts eaten which is attributed to the improved chasing ability of the ghosts. The GW algorithm contains two prominent negative factors other than the ghost's vision when competition against the PW algorithm, the two competing factors are the GW_AWAY_GH and GW_TO_GH. These two factors emphasis moving away from other ghosts and moving towards other ghosts, interestingly this could be considered comparable to the flocking algorithm cohesion and separation. These two factors play prominent roles in decreasing the number of close calls, number of levels completed and power-pellets collected.

The factors GW_PAC and GW_PAC_DIR produce similar results for about half of the response variables. Comparing the two factors, we observe increased similarity in comparison to the SSS_GW factor results. Although, the main effects are close in value for a number of the response variables, GW_PAC_DIR produces a much larger effect for the following response variables: close calls, ghosts eaten, levels completed and the score. These results indicate that if we focus on Pac-Man's current position, the ghosts arrive too late to challenge Pac-Man but early enough to be counter-attacked, while if the ghosts focus on Pac-Man's next possible position, they provide an increased challenge to Pac-Man forcing previously traveled squares to be revisited.

5.3.3 Game Significance

The previous section presented the statistically significant terms for all algorithms and response variables. The statistical significance provides information pertaining to the reliability and confidence in the term's value. Thus, the statistical significance of a term provides only a portion the necessary information. Our analysis must provide estimates for the practical significance of a term. For instance, consider an example with two statistically significant terms from the score response model. The first term could have an effect size of 5, the second term having an effect size of 60. Although both terms have proven to be statistically significant, an effect size of 5 for the score model is not a large increase over 3 lives, especially considering the analysis performed 3 repetitions. Thus, the second would be considered practically significant, while the first term might not be practically significant. Throughout this section we will review the practical significance of the terms for each algorithm set of response variables. Our analysis will refer to the practical significance as the game significance throughout this section.

Section 5.3.2 presented the statistically significant terms for all of the algorithms and their sets of response variables. These results had been amalgamated from the separated game sessions to provide an overall view of the factors' results on the performance of the algorithm. The analysis in this section does not use the amalgamated data; instead this section uses the separated game sessions data set. The first reason we will use the separated game sessions data is because the R-Sq values and lack-of-fit tests were all calculated using this data set. The lack-of-fit tests for the game sessions data showed no significant lack-of-fit. Secondly, the separated game sessions provide structured results similar to observing individual player sessions; thus portraying a methodology for performing the analysis on a set of individual player sessions or online game sessions. The adaptive game prototype was designed to experiment with individual player sessions; this provides game significant results in a format usable by the adaptive prototype.

The game significance of a term is more subjective than the statistical significance

for a number of reasons. One reason is that no comparable results from other research exist or are likely to exist for a single video game. To have comparable results, research would have to be conducted on standard implementations of a game, and no such standard version of games exists. Another reason is that the player's performance can be volatile; thus selecting boundaries for game significance can be difficult. The means, standard deviation and other descriptive statistics used in this section can be found in AppendixE. It is important to remember the response variable means calculated in our analysis are the result of 3 lives.

In considering a term's game significance, we can review, among other things, descriptive statistics such as means and standard deviation and the term effect sizes. The term's effect sizes provides important information about how altering a factor alters the results of a response variable. However, game significance should address a number of issues in combination with altering response variables if considering integration in an auto-dynamic difficulty game system. For instance, the number of modifications required to transition between two terms could play a role in game significance. If two terms have relatively equal effect sizes, selecting the term which minimizes the number of required modifications provides two advantages. First, it lowers the chance of alerting the player to the dynamic difficulty system, which as previously discussed may cause players to externalize results. Second, it reduces the chance of making a catastrophic adjustment which alters the difficulty of the game in such a drastic way that it overwhelms or bores the user. This is an example of dynamic game significance which would be calculated at run-time and ranks terms with similar effect sizes. Our analysis will focus on the effect size and descriptive stats to justify our selection of game significance.

Other areas of academic research separate practical significance into categories based on an estimate of significance. Generally, results will be categorized based on titles such as: trivial, beneficial and negative [24]. Terms with minimal difference between their effect sizes may be used to dynamically adjust between different player types or add variety. Terms with smaller effect sizes may be used to adjust the game on a very fine granularity to the difficulty of the player. Thus, although we will use

the terminology trivial, this applies only to the effect size on the response variable and not the impact on the game. Our categorical dissection will exclude the negative category, as results will be separated based on the absolute value of the effect size. Our categorical separation will use the terminology trivial, for terms with minimal effect sizes on the response variables. Beneficial terms will contribute higher absolute effect sizes than the trivial terms. Prominent terms will be the game significant terms with the largest absolute factor effects.

5.3.3.1 Game Significance of Response Close Calls

Identifying the game significance of the close calls is a difficult task due to the rules of the game. Pac-Man is a game in which the player is either alive or dead; there is no health. This makes dynamic adjustments more difficult, as one close call could potentially kill Pac-Man. The descriptive effect results presented in Appendix F indicates that the mean absolute value occurs between 2.34 and 3.49. The standard deviation for the close calls factor effects ranges from 4.14 to 5.20. Using this information and general knowledge about the game we can build intervals for each of the categories of statistical significance. The trivial category could be considered terms whose absolute effect sizes range from 0 to 2.34. Intuitively, we can consider effect sizes below 3 as trivial because on average they provide one additional close call per life. Using the standard deviation we define our beneficial range as 2.35 - 6.75. Thus prominent terms will be classified as having values of 6.75 or larger.

SSS_FLOCK	SSS_GW	PW_FLOCK	PW_GW
GH_VIS_LEN	GH_VIS_LEN	GH_VIS_LEN	GH_VIS_LEN
FLOCK_SEP	GW_AWAY_GH	FLOCK_ALI	GW_AWAY_GH
FLEE_TIME			FLEE_TIME

Table 5.25: Prominent Factors for Close Calls Response Variable.

A summary of the prominent factors for each set of competing algorithms occurs in Table 5.25. For the flocking algorithm and SSS-AB* algorithm the prominent game main effects become: the flee time, the flocks hunger and the ghosts vision range. For the SSS-AB* algorithm and the ghost-weighted algorithm the prominent game main

effects become: the ghost's vision range and prioritizing ghost's spreading out. The Pac-Man weighted algorithm against the flocking algorithm produced the prominent game main effects flocking alignment and ghost's vision range. Pac-Man Weighted algorithm against the ghost weighted algorithm produced the following prominent main effects: ghost separation, the range of ghosts' vision and the length of the state flee time.

5.3.3.2 Game Significance of Response Fruit Collected

In Pac-Man collecting fruit is a bonus task which contributes extra points to Pac-Man's score when collected. Identifying the game significance of terms for the fruits collected can be misleading because the game significance is dependent on the decision to participate in trying to retrieve the fruit. If Pac-Man chooses not to participate in collecting fruit the terms have no game significance. If the decision is made to attempt to retrieve a bonus fruit and the task is not accomplished, the response variable will not reflect this decision.

The descriptive effect results presented in Appendix F indicates that the mean absolute value occurs between 0.01 and 0.19. The standard deviation for fruits collected factor effects ranges from 0.02 to 0.03. Using this information and general knowledge about the game we can build intervals for each of the categories of statistical significance. The trivial category could be considered terms whose absolute effect sizes range from 0 to 0.01. Using the standard deviation we define our beneficial range as 0.01 to 0.035. Thus prominent terms will be classified as having values of 0.035 or larger.

SSS_FLOCK	SSS_GW	PW_FLOCK	PW_GW
FLOCK_SEP	GH_VIS_LEN	FLOCK_COH	FLEE_TIME
PER_FRT		DEATH_TIME	DEATH_TIME
PAC_VIS_LEN		GH_VIS_LEN	GH_VIS_LEN
GH_VIS_LEN			

Table 5.26: Prominent Factors for Fruit Collected Response Variable.

A summary of the prominent factors for each set of competing algorithms occurs

in Table 5.26. For the flocking algorithm and SSS-AB* algorithm the prominent game main effects become: the flock separation, perceived value of the fruit, Pac-Man and the ghost's vision range. For the SSS-AB* algorithm and the ghost-weighted algorithm the only prominent game main effect is the ghost's vision range. The Pac-Man weighted algorithm against the flocking algorithm produced the following prominent main effects: flocking cohesion, the length of the state death time and ghost's vision range. Pac-Man weighted algorithm against the ghost weighted algorithm produced the following prominent main effects: the range of ghost's vision, the length of the state flee time and the length of the state death time.

5.3.3.3 Game Significance of Response Ghosts Eaten

The descriptive effect results presented in Appendix F indicates that the mean absolute value occurs between 0.04 and 0.06. The standard deviation for number of ghosts eaten factor effects ranges from 0.05 to 0.09. Using this information and general knowledge about the game we can build intervals for each of the categories of statistical significance. The trivial category could be considered terms whose absolute effect sizes range from 0 to 0.05. Using the standard deviation we define our beneficial range as 0.05 to 0.12. Thus prominent terms will be classified as having values of 0.12 or larger.

SSS_FLOCK	SSS_GW	PW_FLOCK	PW_GW
FLEE_TIME	GW_TO_GH	FLOCK_HUNGER	GW_PAC_DIR
GH_VIS_LEN	GW_AWAY_GH		FLEE_TIME
	FLEE_TIME		

Table 5.27: Prominent Factors for Ghosts Eaten Response Variable.

A summary of the prominent factors for each set of competing algorithms occurs in Table 5.27. For the flocking algorithm and SSS-AB* algorithm the prominent game main effects become: the length of time of the flee state and the ghost's vision range. For the SSS-AB* algorithm and the ghost-weighted algorithm the prominent game main effects are: the ghost's separation and cohesion effect and the length of flee time. The Pac-Man weighted algorithm against the flocking algorithm produced one

prominent main effect, the flocking hunger. Pac-Man weighted algorithm against the ghost weighted algorithm produced the following prominent main effects: the ghost's moving towards an estimate of Pac-Man's next position and the length of the state flee time.

5.3.3.4 Game Significance of Response Levels Completed

The game significance of the number of levels completed is an one of the most important response variables because it indicates the player's progression. However, as we observed in Section 5.3.1.1 that calculated the variation and validated our models, the models for this response variable had the lowest values explanation of variation.

The descriptive effect results presented in Appendix F indicates that the mean absolute value occurs between 0.01 and 0.06. The standard deviation for levels completed factor effects ranges from 0.01 to 0.02. Using this information and general knowledge about the game we can build intervals for each of the categories of statistical significance. The trivial category could be considered terms whose absolute effect sizes range from 0 to 0.02. Using the standard deviation we define our beneficial range as 0.02 to 0.035. Thus prominent terms will be classified as having values of 0.035 or larger.

SSS_FLOCK	SSS_GW	PW_FLOCK	PW_GW
FLEE_TIME	GW_AWAY_GH	DEATH_TIME	
GH_VIS_LEN	FLEE_TIME	GH_VIS_LEN	
PAC_VIS_LEN			

Table 5.28: Prominent Factors for Levels Completed Response Variable.

A summary of the prominent factors for each set of competing algorithms occurs in Table 5.28. For the flocking algorithm and SSS-AB* algorithm the prominent game main effects become: the length of time of the flee state, Pac-Man's vision range and the ghost's vision range. For the SSS-AB* algorithm and the ghost-weighted algorithm the prominent game main effects are: the ghost's separation and the length of flee time. The Pac-Man weighted algorithm against the flocking algorithm produced the following prominent main effects: the length of the state death time and

the ghost's vision range. Pac-Man weighted algorithm against the ghost weighted algorithm produced the no prominent main effects.

5.3.3.5 Game Significance of Response Power-Pellets

The descriptive effect results presented in Appendix F indicates that the mean absolute value occurs between 0.02 and 0.05. The standard deviation for power-pellets collected factor effects ranges from 0.04 to 0.08. Using this information and general knowledge about the game we can build intervals for each of the categories of statistical significance. The trivial category could be considered terms whose absolute effect sizes range from 0 to 0.035. Using the standard deviation we define our beneficial range as 0.035 to 0.10. Thus prominent terms will be classified as having values of 0.1 or larger.

SSS_FLOCK	SSS_GW	PW_FLOCK	PW_GW
FLEE_TIME	GW_AWAY_GH	DEATH_TIME	GH_VIS_LEN
GH_VIS_LEN	FLEE_TIME	GH_VIS_LEN	
PAC_VIS_LEN	DEATH_TIME		
DEATH_TIME	GH_VIS_LEN		

Table 5.29: Prominent Factors for Power-Pellets Collected Response Variable.

A summary of the prominent factors for each set of competing algorithms occurs in Table 5.29. For the flocking algorithm and SSS-AB* algorithm the prominent game main effects become: the length of time of the flee and death state, Pac-Man's vision range and the ghost's vision range. For the SSS-AB* algorithm and the ghost-weighted algorithm the prominent game main effects are: the ghost's separation, the length of flee and death time and the ghost's vision range. The Pac-Man weighted algorithm against the flocking algorithm produced the following prominent main effects: the length of the state death time and the ghost's vision range. Pac-Man weighted algorithm against the ghost weighted algorithm produced one prominent main effect, ghost's vision range.

The game significance of the power-pellets collected response variable was such a prominent factor that it became one of the major contributors to the heuristics built

in the adaptive prototype.

5.3.3.6 Game Significance of Response Repeated Squares

The descriptive effect results presented in Appendix F indicates that the mean absolute value occurs between 1.55 to 3.52. The standard deviation for repeated squares factor effects ranges from 2.02 and 6.11. Using this information and general knowledge about the game we can build intervals for each of the categories of statistical significance. The trivial category could be considered terms whose absolute effect sizes range from 0 to 2.25. Using the standard deviation we define our beneficial range as 2.5 to 4.0. Thus prominent terms will be classified as having values of 4.0 or larger.

SSS_FLOCK	SSS_GW	PW_FLOCK	PW_GW
FLEE_TIME	GW_AWAY_GH	DEATH_TIME	GH_VIS_LEN
GH_VIS_LEN	FLEE_TIME	GH_VIS_LEN	
PAC_VIS_LEN	DEATH_TIME		
DEATH_TIME	GH_VIS_LEN		

Table 5.30: Prominent Factors for Repeated Squares Response Variable.

A summary of the prominent factors for each set of competing algorithms occurs in Table 5.30. For the flocking algorithm and SSS-AB* algorithm the prominent game main effects become: the length of time of the flee and death state, Pac-Man's vision range and the ghost's vision range. For the SSS-AB* algorithm and the ghost-weighted algorithm the prominent game main effects are: the ghost's separation, the length of flee and death time and the ghost's vision range. The Pac-Man weighted algorithm against the flocking algorithm produced the following prominent main effects: the length of the state death time and the ghost's vision range. Pac-Man weighted algorithm against the ghost weighted algorithm produced one prominent main effect, ghost's vision range.

5.3.3.7 Game Significance for Response Score

The descriptive effect results presented in Appendix F indicates that the mean absolute value occurs between 11.87 to 22.03. The standard deviation for score factor effects ranges from 18.03 and 36.72. Using this information and general knowledge about the game we can build intervals for each of the categories of statistical significance. The trivial category could be considered terms whose absolute effect sizes range from 0 to 15. Using the standard deviation we define our beneficial range as 15 to 50. Thus prominent terms will be classified as having values of 50 or larger.

SSS_FLOCK	SSS_GW	PW_FLOCK	PW_GW
FLEE_TIME	GW_AWAY_GH	DEATH_TIME	GH_VIS_LEN
GH_VIS_LEN	FLEE_TIME	GH_VIS_LEN	FLEE_TIME
PAC_VIS_LEN		FLOCK_HUNGER	

Table 5.31: Prominent Factors for Score Response Variable.

A summary of the prominent factors for each set of competing algorithms occurs in Table 5.31. For the flocking algorithm and SSS-AB* algorithm the prominent game main effects become: the length of time of the flee state, Pac-Man's vision range and the ghost's vision range. For the SSS-AB* algorithm and the ghost-weighted algorithm the prominent game main effects are: the ghost's separation, the length of flee and the ghost's vision range. The Pac-Man weighted algorithm against the flocking algorithm produced the following prominent main effects: the length of the state death time, the flocking hunger and the ghost's vision range. Pac-Man weighted algorithm against the ghost weighted algorithm produced the following prominent main effects: the length of the flee time and ghost's vision range.

5.3.3.8 Game Significance of Response Steps

The descriptive effect results presented in Appendix F indicates that the mean absolute value occurs between 1.42 to 4.04. The standard deviation for steps factor effects ranges from 2.24 and 7.03. Using this information and general knowledge about the game we can build intervals for each of the categories of statistical significance. The trivial category could be considered terms whose absolute effect sizes range from 0

to 2.70. Using the standard deviation we define our beneficial range as 2.70 to 6.75. Thus prominent terms will be classified as having values of 6.75 or larger.

SSS_FLOCK	SSS_GW	PW_FLOCK	PW_GW
FLEE_TIME	GW_AWAY_GH	DEATH_TIME	GH_VIS_LEN
GH_VIS_LEN	FLEE_TIME	GH_VIS_LEN	FLEE_TIME
PAC_VIS_LEN	DEATH_TIME	FLOCK_HUNGER	GW_AWAY_GH
FLOCK_HUNGER	GH_VIS_LEN		

Table 5.32: Prominent Factors for Steps Response Variable.

A summary of the prominent factors for each set of competing algorithms occurs in Table 5.32. For the flocking algorithm and SSS-AB* algorithm the prominent game main effects become: the length of time of the flee state, the flock's hunger, Pac-Man's vision range and the ghost's vision range. For the SSS-AB* algorithm and the ghost-weighted algorithm the prominent game main effects are: the ghost's separation, the length of flee and death times and the ghost's vision range. The Pac-Man weighted algorithm against the flocking algorithm produced the following prominent main effects: the length of the state death time, the flocking hunger and the ghost's vision range. Pac-Man weighted algorithm against the ghost weighted algorithm produced the following prominent main effects: the length of the flee time, the ghost's separation and ghost's vision range.

5.3.3.9 Game Significance of Response Tokens

The descriptive effect results presented in F indicates that the mean absolute value occurs between 0.94 to 2.00. The standard deviation for tokens collected factor effects ranges from 1.45 and 4.00. Using this information and general knowledge about the game we can build intervals for each of the categories of statistical significance. The trivial category could be considered terms whose absolute effect sizes range from 0 to 1.50. Using the standard deviation we define our beneficial range as 1.50 to 4.25. Thus prominent terms will be classified as having values of 4.25 or larger.

A summary of the prominent factors for each set of competing algorithms occurs in 5.33. For the flocking algorithm and SSS-AB* algorithm the prominent game main

SSS_FLOCK	SSS_GW	PW_FLOCK	PW_GW
FLEE_TIME	GW_AWAY_GH	DEATH_TIME	GH_VIS_LEN
GH_VIS_LEN	FLEE_TIME	GH_VIS_LEN	
PAC_VIS_LEN	GH_VIS_LEN	FLOCK_HUNGER	

Table 5.33: Prominent Factors for Tokens Response Variable.

effects become: the length of time of the flee state, Pac-Man's vision range and the ghost's vision range. For the SSS-AB* algorithm and the ghost-weighted algorithm the prominent game main effects are: the ghost's separation, the length of the flee state time and the ghost's vision range. The Pac-Man weighted algorithm against the flocking algorithm produced the following prominent main effects: the length of the state death time, the flocking hunger and the ghost's vision range. Pac-Man weighted algorithm against the ghost weighted algorithm produced the following prominent main effect: ghost's vision range.

5.3.4 Separated Terms Comparison

This subsection presents the results from the factorial analysis, in which we compare the results of the separated factors from the term significance analysis Section 5.2.4. The analysis performed in this section will be simplified, it does not calculate the effects of each factor but compares results graphically. In this section we present results using graphs that illustrate a percentage comparison for each response variable to highlight the difference in performance between factors. We will compare the base case which has all factors at their lowest levels against the high case of each separated factor.

5.3.4.1 Comparison of SSS_FLOCK Separated Terms

An important result for the analysis of this algorithm pair is that almost all response variables are statistically similar whether we using the high or low value of fruit frequency. If we review the results in Table 5.34 the high level value of fruit creation indicated by game settings 1, indicates that the average number of fruits collected nearly triples to 0.97, which increases at a rate slightly above the difference in the

rate of fruit creation.

Game Setting	ID_0	ID_1	Ratio Difference
CC	146.14	142.22	0.97
FC	2.29	5.83	2.55
FE	0.35	0.97	2.77
GH	3.038	2.87	0.95
LVL_COMP	0.68	0.64	0.94
PP	5.92	5.80	0.98
RSQ	138.90	137.17	0.98
Sc	1911.80	1947.33	1.02
St	371.78	364.71	0.98
To	226.96	221.74	0.98

Table 5.34: Comparing response variable results between two game sessions for SSS_FLOCK. Game Setting ID 0 is the low level of fruit frequency, while Game Setting ID 1 is the high value.

As expected with increasing the rate of fruit creation increases the score, however this increase is expected to be higher based on the values from the number of fruit collected. Intuitively if a fruit is worth 150 points if collected, our expectation is for the score difference to be higher than the roughly 36 point difference. Other response variables such as eating fewer ghosts indicate a possible reason for this increase in the number of close calls, which occur when the ghosts come close to killing Pac-Man. The increase in close calls indicates that the creation of additional fruits is creating additional opportunities or possible traps where the ghosts can improve their chances of eating Pac-Man. Although the flocking algorithm is unaware of the position of fruit, fruit are positioned at squares Pac-Man has already visited. Thus the more intuitive explanation for this result is Pac-Man revisiting dangerous areas, backtracking into guarded areas or chasing ghosts. The results here indicate that the increased creation of the fruit is lowering Pac-Man's task competition of other response variables.

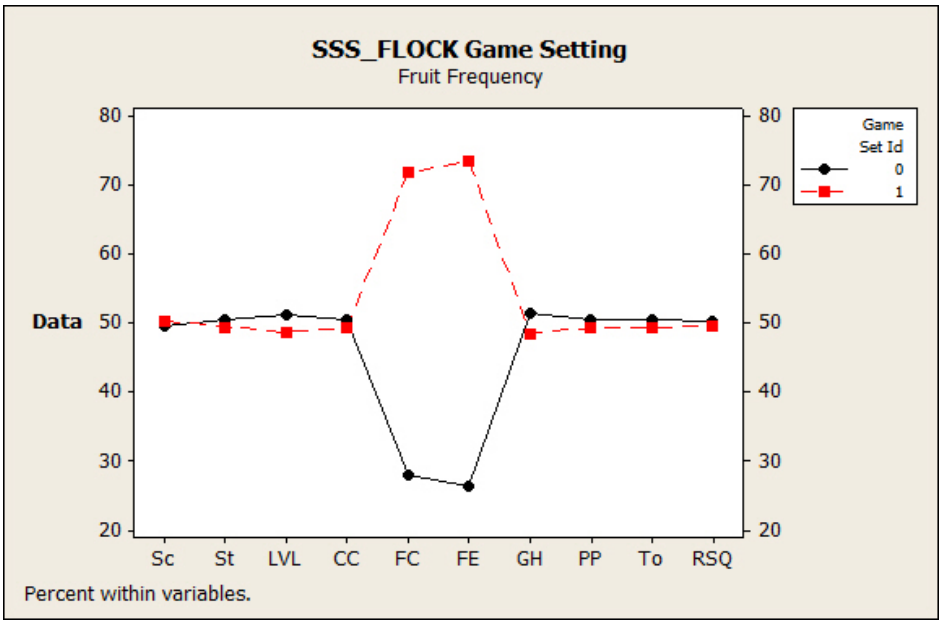


Figure 5.37: Comparison of the response variables using SSS_FLOCK and modifying the level of the Fruit Frequency. Game Setting ID 0 is the low level of fruit frequency, while Game Setting ID 1 is the high value.

5.3.4.2 Comparison of SSS_GW Separated Terms

Next we review the separated factors that were excluded from the statistical and game significant sections of the SSS_GW analysis. Figure 5.38 demonstrates the average results of each of these factors at their low level (GW_BASE), followed by the high level results indicated by their ID number. As expected in GW_SSS_FRT_FREQ as the frequency of fruit creation increases, so too does the number of fruit collected as well as the score. Unlike the SSS_FLOCK algorithm, the difference in both the number of fruit collected and the score is reflective of the difference in the ratio of fruit creation. Similarly, GW_SSS_FT produces an increase in the number of fruit collected, however that correlates to increasing the amount of time the fruit is available. An interesting result from comparing these factors is a noticeable increase in the completion of levels occurring in certain game settings. This pattern occurs when an increase in FRUIT_TIME or FRUIT_FREQ occurred, leading to a couple of possible explanations. One explanation is that ghosts are moving to protect fruit while leaving key areas of the board unguarded, another explanation is that as Pac-

Man attempts to collect the fruit it draws the ghost's attention away from unfinished areas of the board that they were protecting thus helping Pac-Man complete level tasks.

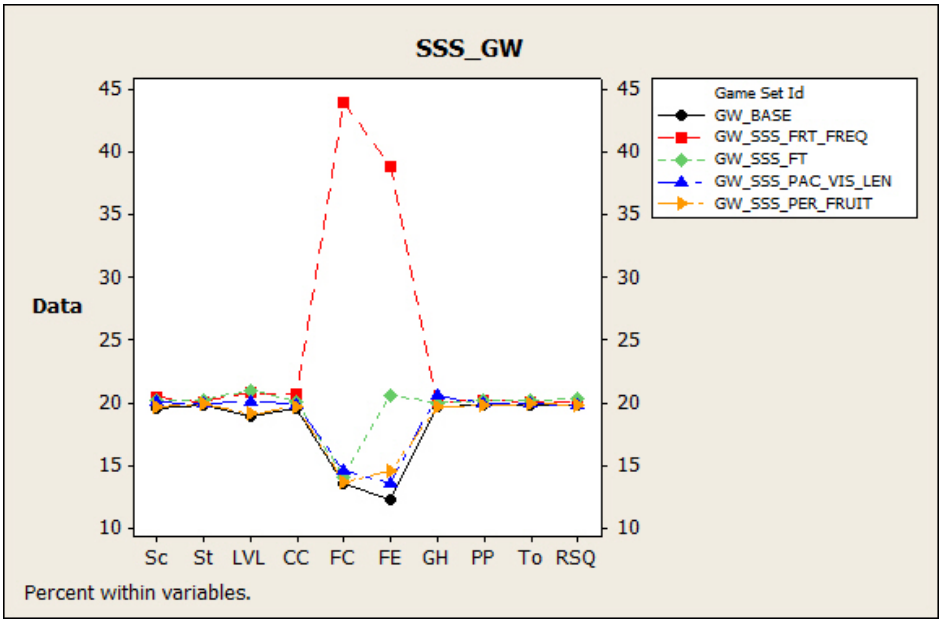


Figure 5.38: Game Sessions results for SSS_GW games. The ID for each game session represent the following factors at their high level SSS_PERCEIVED_FRUIT (GW_SSS_PER_FRUIT), FRUIT_TIME (GW_SSS_FT), FRUIT_FREQ (GW_SSS_FRT_FREQ), PAC_VIS_LEN (GW_SSS_PAC_VIS_LEN) and the base case (GW_BASE) all factors set to their lowest level value.

5.3.4.3 Comparison of PW_FLOCK Separated Terms

To compare the game session for the separated factors in the PW_FLOCK algorithm, we review seven sessions excluded from the previous analysis sections. The first six relate to parameters of the PW algorithm followed by a final case which highlights the results of the fruit frequency. The first factor we will investigate is PW_TOK_FOR as (ID_1), while the base case or factors at their low level is illustrated by (ID_0). Figure 5.39 illustrates a direct comparison between these cases. It appears that prioritizing tokens ahead of other rules produces a negative effect on the game results. An explanation for this result is the board is full of tokens and prioritizing tokens in close proximity may lead to prioritizing dangerous tokens over ones available in safer

zones. As results of the flocking algorithm have demonstrated, it is less competitive than the GW algorithm but if Pac-Man remains in the same area collecting all of the tokens, the flocking algorithm will have a greater chance of surrounding and capturing Pac-Man.

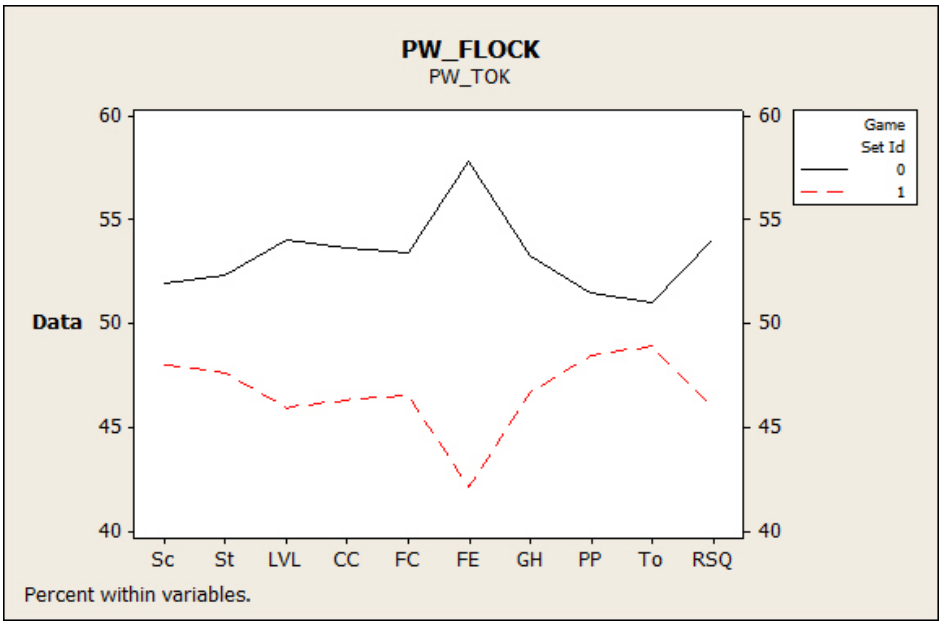


Figure 5.39: Review of the PW_FLOCK game session with low (ID_0) and high values (ID_1) of PW_TOK factor.

The next factor we will investigate is the PW_PP_FOR (ID_2) or priority for moving toward and collecting visible power-pellets. As expected, focusing on power-pellets results in Pac-Man eating more power-pellets, however an unexpected result occurs during the usage of Pac-Man's predator time. As results in Figure 5.40 illustrate, this factor causes increases in the number of close calls and fruit collected, while a minimal difference in the number of ghosts eaten is observed. Two possible causes for the increase in the number of close calls could be (1) while Pac-Man is chasing edible ghosts their flee state ends or (2) Pac-Man focusing on power-pellets which the ghosts are protecting. As Pac-Man was unable to eat additional ghosts with the increased predatorial time, it strengthens the possibility of being unable to catch the ghosts or that the predator time was used in a different way. The results demonstrate a large spike in the number of fruits collected indicating that perhaps

Pac-Man used the predator time to accomplish bonus tasks rather than level completing tasks. This possibility is reinforced by the fact that both tokens and level completion values where also reduced.

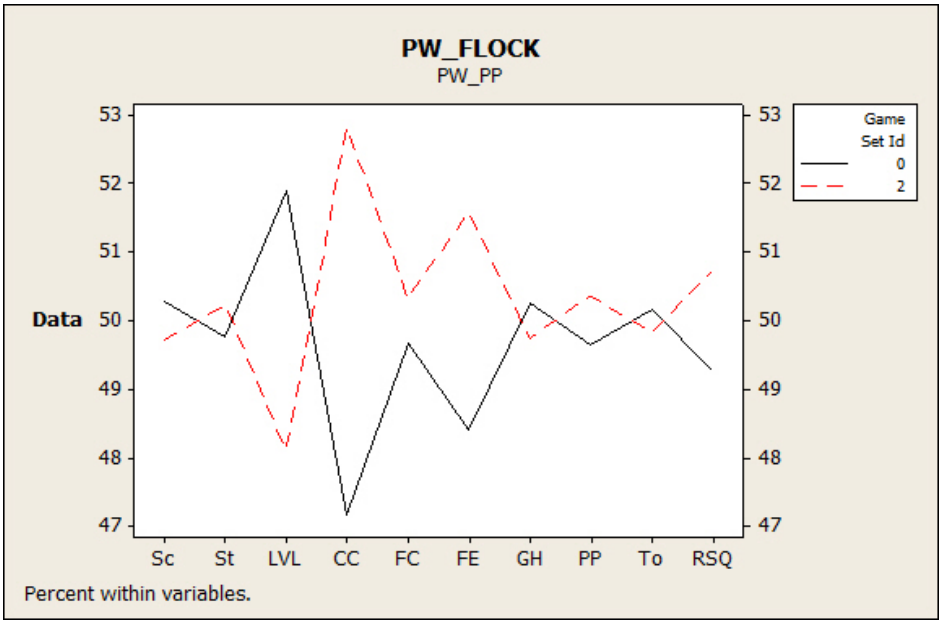


Figure 5.40: Review of the PW_FLOCK game session with low (ID_0) and high values (ID_2) of PW_PP factor.

The PW factor PW_EDIBLE_GH (ID_4) prioritizes eating ghosts during Pac-Man's predator time. The results indicate that focusing too heavily on eating edible ghosts diminished all other response variables except the number of ghosts eaten and the number of close calls. The increased number of close calls is indicative of the change from prey to predator for the ghosts while Pac-Man is chasing them.

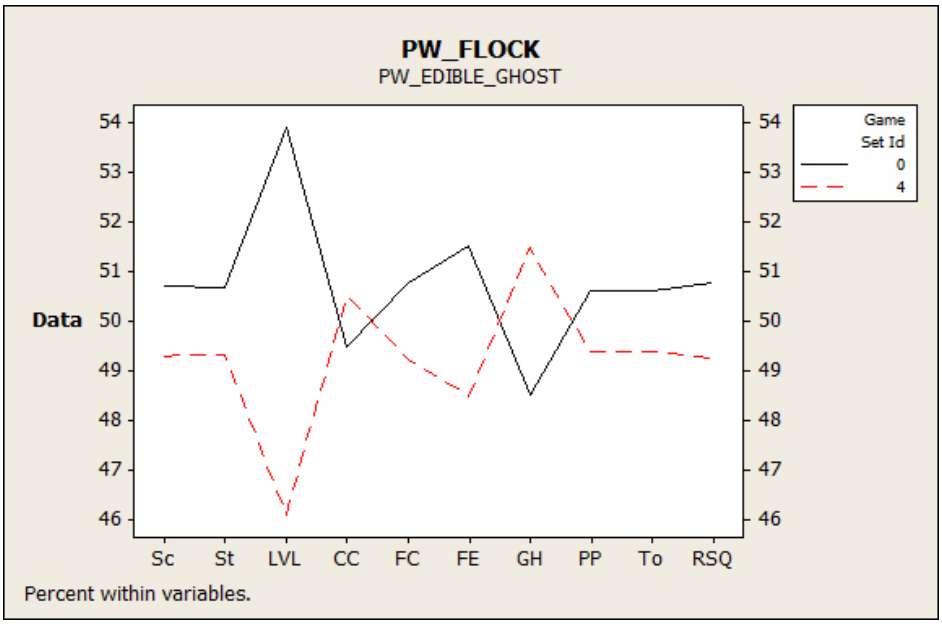


Figure 5.41: Review of the PW_FLOCK game session with low (ID_0) and high values (ID_4) of PW_EDGH factor.

The PW factor PW_BADGH(ID_8) emphasizes avoiding ghosts, preferring to avoid them as much as possible. This factor produces a vast improvement over the performance of the base level in every response variable. Considering the increase in the number of steps and power-pellets collected, the results indicate a quite small increase in the number of ghosts eaten. This result and the large increase in repeated squares indicates that Pac-Man is avoiding ghosts during his predator and prey time. This factor is further discussed in the section comparing the player algorithms against the flocking algorithm.

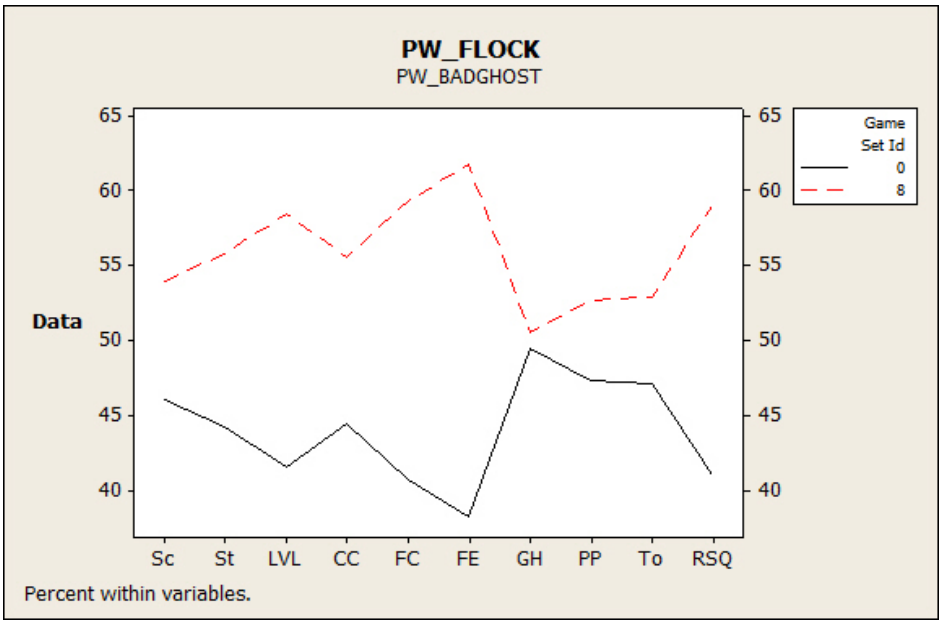


Figure 5.42: Review of the PW_FLOCK game session with low (ID_0) and high values (ID_8) of PW_EDGH

The PW factor PW_BADGH_CTR(ID_16) emphasizes Pac-Man avoiding the center of the ghost's positions in prey mode and moving towards this position in predator mode as a heuristic to avoid being surrounded by the ghosts. We observe that this heuristic increases the average life span of Pac-Man and the number of fruits obtained, however the extra life span is ineffective in acquiring additional points and increases the number of repeated squares. A possible explanation is that Pac-Man is failing to move toward high risk and high reward areas. The risk involved in moving toward the centroid of the ghosts is dynamic and the true risk may fluctuate based on the separation of the ghosts from this position. Thus Pac-Man could be missing opportunities of high reward and minimal risk that are incorrectly deemed high risk.

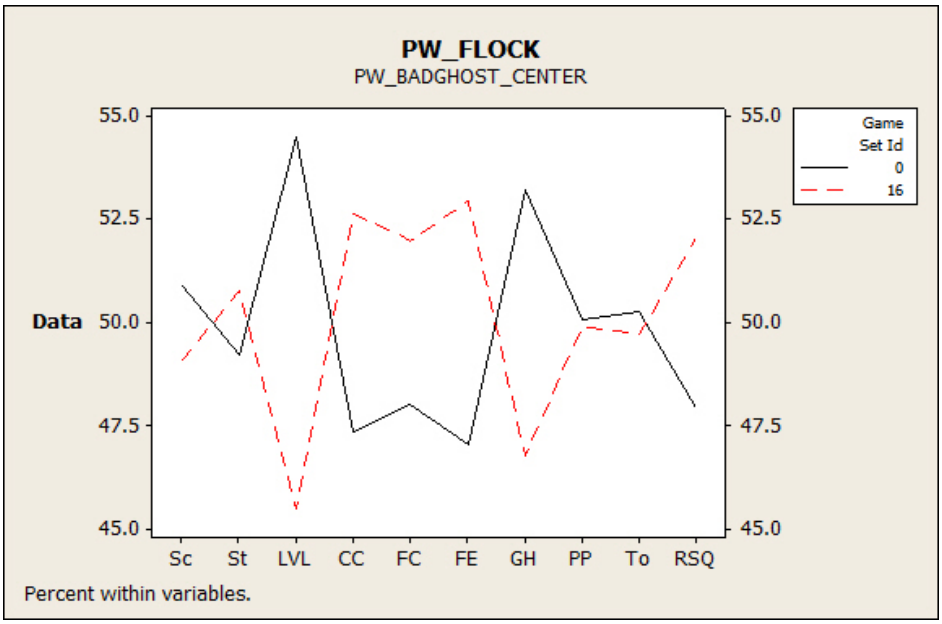


Figure 5.43: Review of the PW_FLOCK game session with low (ID_0) and high values (ID_16) of PW_BADGHOST_CENTER

The factor PW_ITEM_CTR (ID_32) prioritizes moving toward the center of the visible items on the board. The results indicate a negative effect on the majority of response variables. Notably, the number of the close calls increases indicating additional risk in these positions which is likely closer to the center of the board. As previous mentioned in the analysis of the previous factor PW_BADGHOST_CENTER, moving toward or away from a centroid can be misleading as the position of items can be at opposite ends of the board, but the centroid would be in the middle. Emphasizing this factor in isolation leads to prioritizing items in the center of the board which tends to be a more dangerous areas of the board, while potentially ignoring unprotected areas along the border of the board.

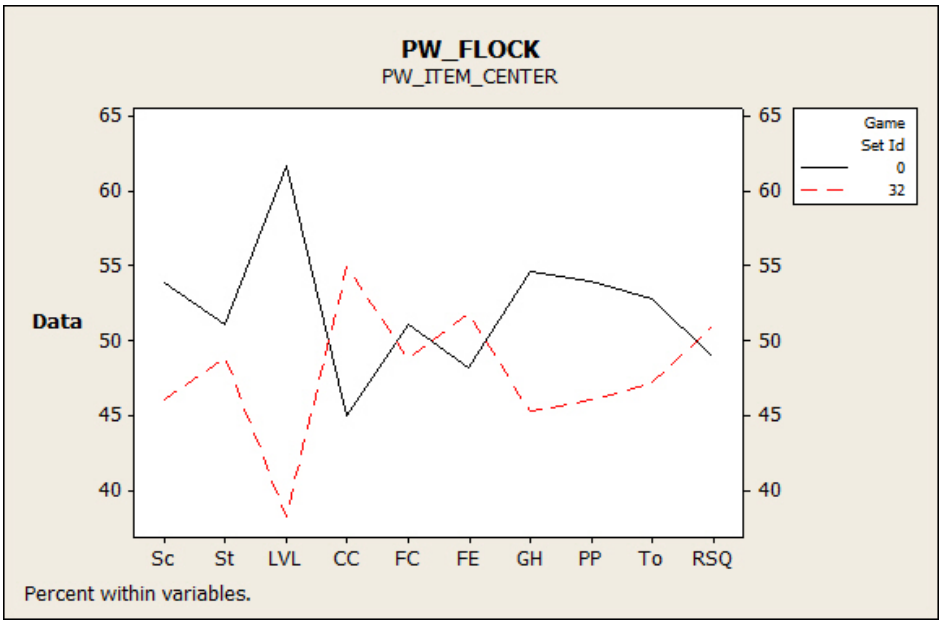


Figure 5.44: Review of the PW_FLOCK game session with low (ID_0) and high values (ID_32) of PW_ITEM_CTR

The factor FRUIT_FREQ (ID_64) produces results that are similar to the other previously discussed algorithms. Although the basic pattern for this factor remains the same, the separation of score, steps and repeated squares are more pronounced for this case. Altering the frequency of the fruit for the algorithm PW_FLOCK produced similar results especially in comparison to SSS_FLOCK algorithm.

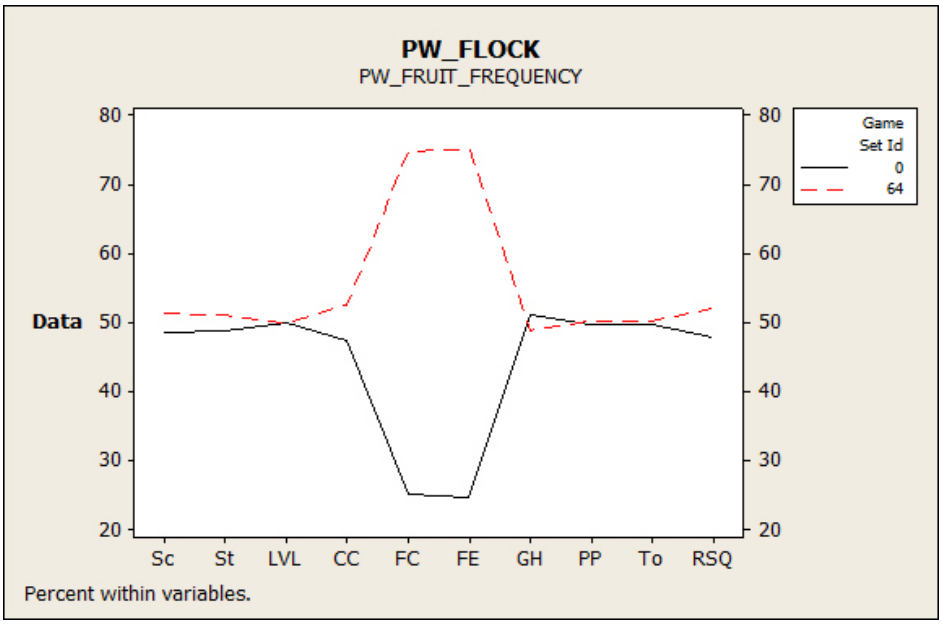


Figure 5.45: Review of the PW_FLOCK game session with low (ID_0) and high values (ID_64) of Fruit Frequency

5.3.4.4 Comparison of PW_GW Separated Terms

The final algorithm pair for which we will review sessions is the PW_GW. It is the largest case and as such contains the largest number of separated factors. The factors where divided into groups, the first relating to the PW algorithm which produced game sessions, the second group contains information for Pac-Man's vision and factors relating to the fruit.

The factor PW_FRUIT_FOR (ID_1) emphasizes that Pac-Man focuses on obtaining the bonus fruits, the results of which are displayed in Figure 5.46. As expected, we observe an increase in the number of fruits collected despite the decrease in length of life. Although arriving at similar scores, the results indicate that the increased fruit consumption occurred at the expense of the number of levels completed. This result indicates that Pac-Man was successful in consuming the bonus fruit often enough to cover the cost of going for a fruit, but was unable to achieve higher scores indicating that shortly after eating a bonus fruit Pac-Man lost a life.

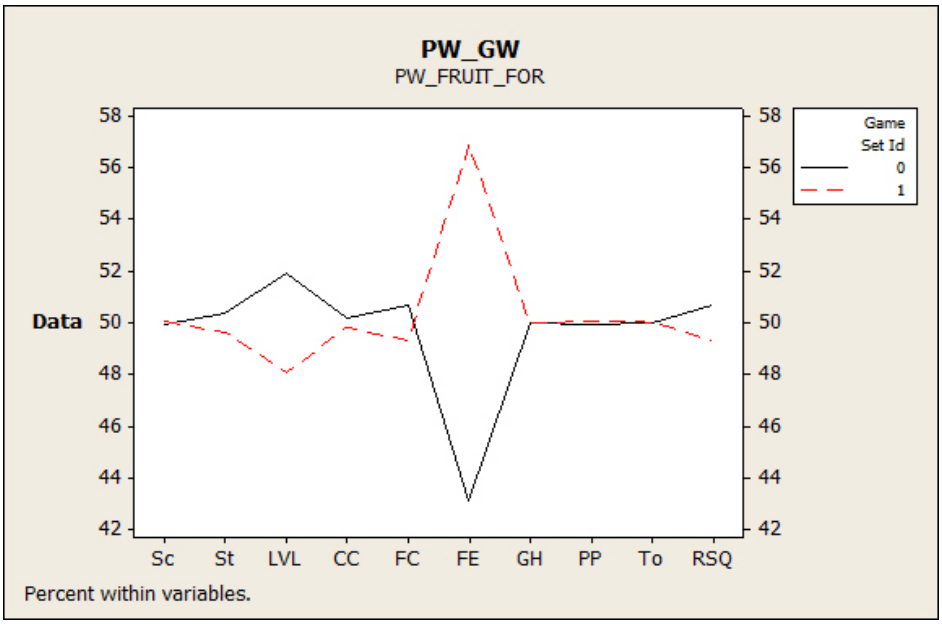


Figure 5.46: Review of the PW_GW game session with low (ID_0) and high values (ID_1) of PW_FRUIT

The factor PW_TOK_FOR(ID_2) prioritizes the collection of tokens above other game elements. Figure 5.47 indicates similar results to those of PW_TOK_FOR in the PW_FLOCK algorithm. Overall, we view a decline in all response variables, the minimal difference occurring between the number of tokens and power-pellets collected.

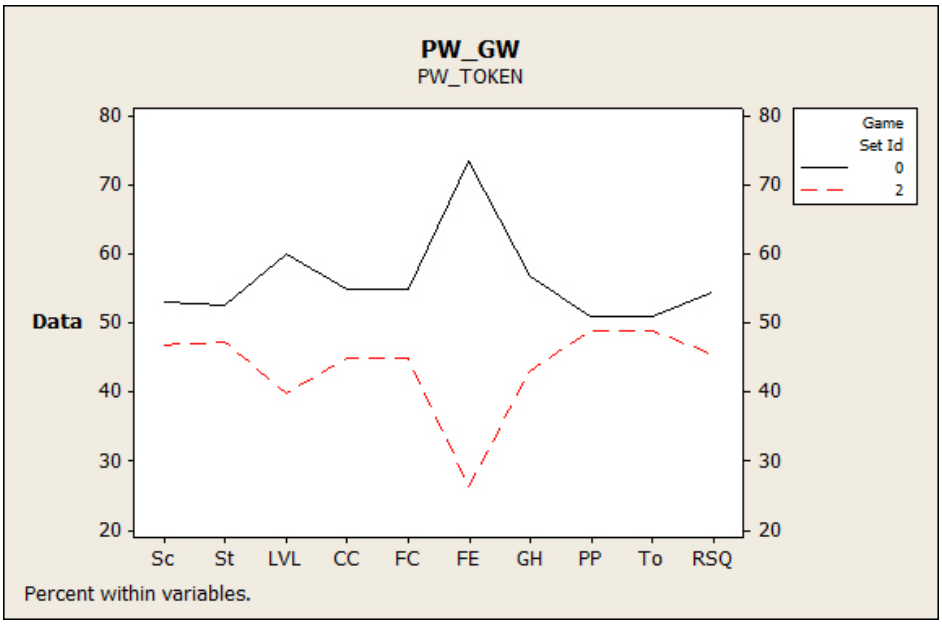


Figure 5.47: Review of the PW_GW game session with low (ID_0) and high values (ID_2) of PW_TOK

The factor PW_PP_FOR(ID_4) prioritizes the collection of power-pellets above other objectives in the game. The results displayed in Figure 5.48 illustrate a different scenario from the same factor in the PW_FLOCK algorithm. Our observation for the increased consumption of power-pellets is that it contributes to an increase in the number of levels completed, a reduced number of repeated squares and a slight increase in the score. This indicates that unlike the results of the flocking algorithm, the increase in the number of power-pellets collected is being utilized to perform level completion tasks as opposed to being utilized for bonus tasks which occurred in the flocking algorithm results. The results demonstrate a decrease in the number of fruits collected, but an increase in the total number of tokens collected. Similar to the flocking algorithm results, we observe an increase in the number of close calls despite similar levels of steps.

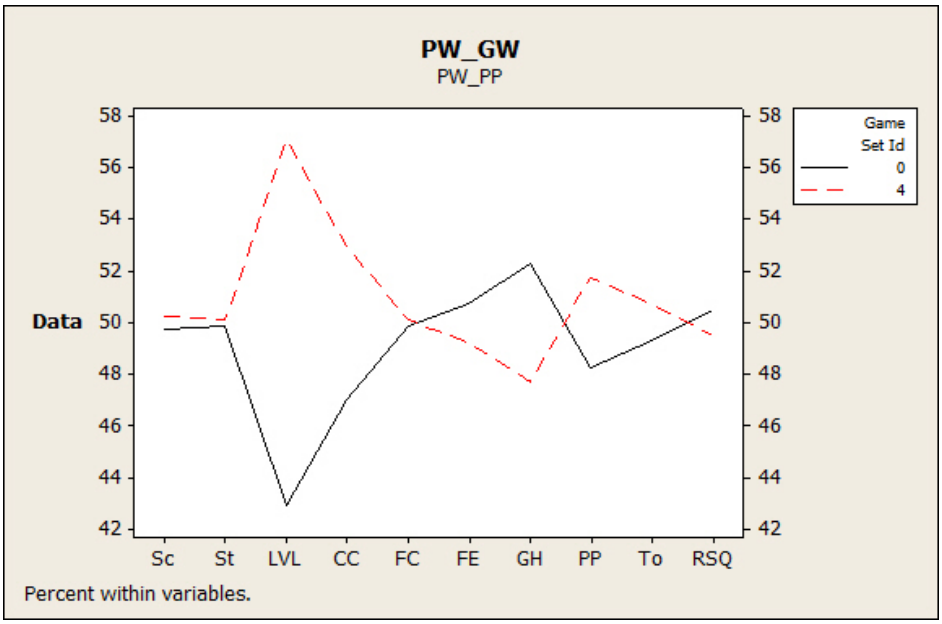


Figure 5.48: Review of the PW_GW game session with low (ID_0) and high values (ID_4) of PW_PP

The factor PW_EDIBLE_GH(ID_8) prioritizes collecting edible ghosts above the other rules in the game. The results against the base case are illustrated in Figure 5.49. As expected, the figure indicates an increase in the number of ghosts eaten. The figure illustrates a decrease in nearly all response variables in this comparison. The only other positive case is that the score response is slightly above the base case. This indicates that chasing edible ghosts has resulted in enough ghosts being consumed to account for the lower scores in the other response variables. Yet focusing on consuming and collecting edible ghosts does not appear to provide enough time to benefit other response variables.

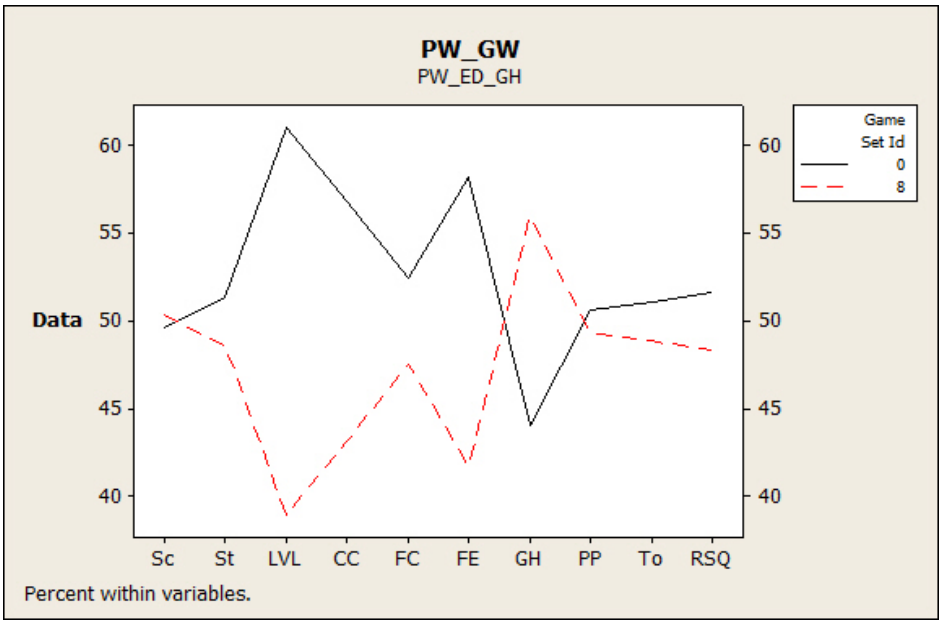


Figure 5.49: Review of the PW_GW game session with low (ID_0) and high values (ID_8) of PW_EDIBLE GHOST

Factor PW_BADGHOST(ID_16) prioritizes avoiding predator ghosts above all other rules. In the PW_FLOCK algorithm, the PW_BADGH factor was the most significant factor. The PW_GW scenario demonstrates a great improvement in nearly all response scores. A decrease in the number of fruits collected was observed, well below the base when we consider the rate of fruit creation.

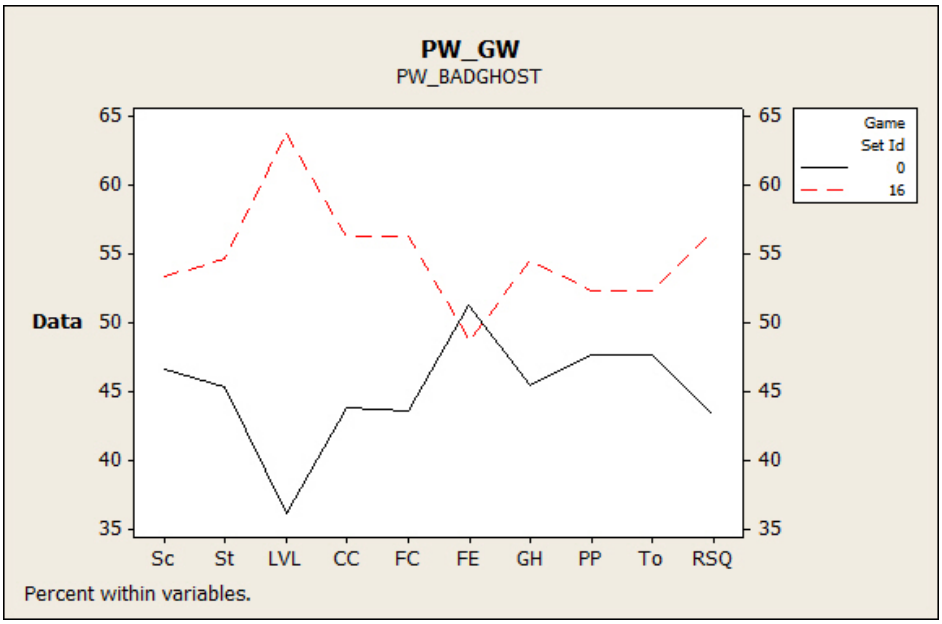


Figure 5.50: Review of the PW_GW game session with low (ID_0) and high values (ID_16) of PW_BADGHOST

Factor PW_BADGHOST_CENTER(ID_32) prioritizes avoiding the center of the ghosts position. Figure 5.51 illustrates a reduced performance in all response variables except the number of fruits collected. Although most responses have decreased in value, the change has been minimal. The diminished value in the number of levels completed may be the result of avoiding important but protected areas. This figure demonstrates a every different picture from the PW_BADGH_CTR against the flocking algorithm. The flocking algorithm showed reduced performance but this reduction was caused by an increase in close calls and the resulting repeated squares. A possible explanation for the difference is the higher level of challenge from the GW algorithm, which halted Pac-Man's progress before similar results could occur.

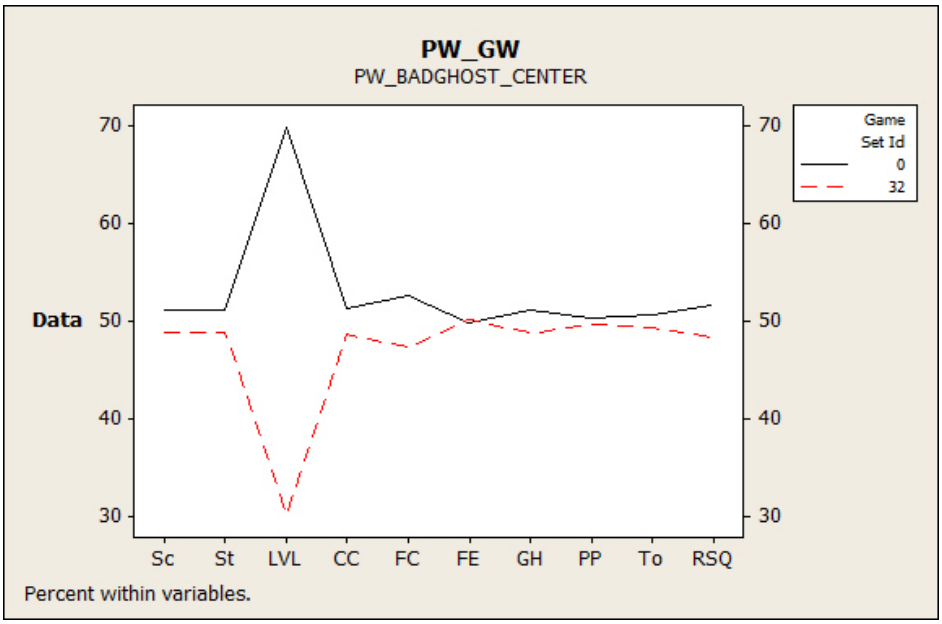


Figure 5.51: Review of the PW_GW game session with low (ID_0) and high values (ID_32) of PW_BADGHOST_CENTER

Factor PW_ITEM_CTR(ID_64) prioritizes moving toward the centroid of the item's position. This factor results in reduced response variables values, although we can observe that the number of close calls and tokens collected are in similar ratio to the number of steps. Comparing the results of the PW_ITEM_CTR factor against the flocking algorithm, we saw results similar to the diminished ability to collect power-pellets and edible ghosts. A potential reason is that power-pellets tend to occur at the outskirts of the levels and would not be in a prioritized area until a portion of the level has been cleared. Similar to the comparison of the PW_BADGHOST_CENTER, we see the PW_ITEM_CTR has reduced response variables values for both the flocking and the weighted algorithm, but that the weighted algorithm seems more competitive and did not allow Pac-Man enough time to increase the number of close calls or repeated steps.

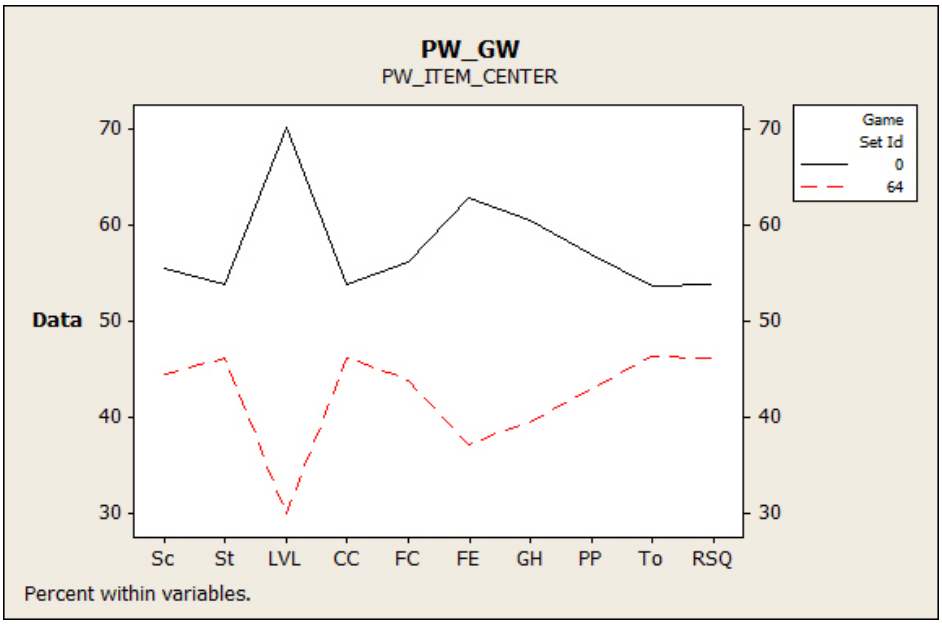


Figure 5.52: Review of the PW_GW game session with low (ID_0) and high values (ID_64) of PW_ITEM_CTR

Factor FRUIT_TIME (ID_128) increases the amount of time that the fruit was available for consumption by Pac-Man. The results demonstrate minimal differences in all response variables. The largest increase was the number of fruits collected which could be expected given the factor change. The results indicate that going for the fruits is increasing the life span of Pac-Man while potentially increasing the number of repeated squares.

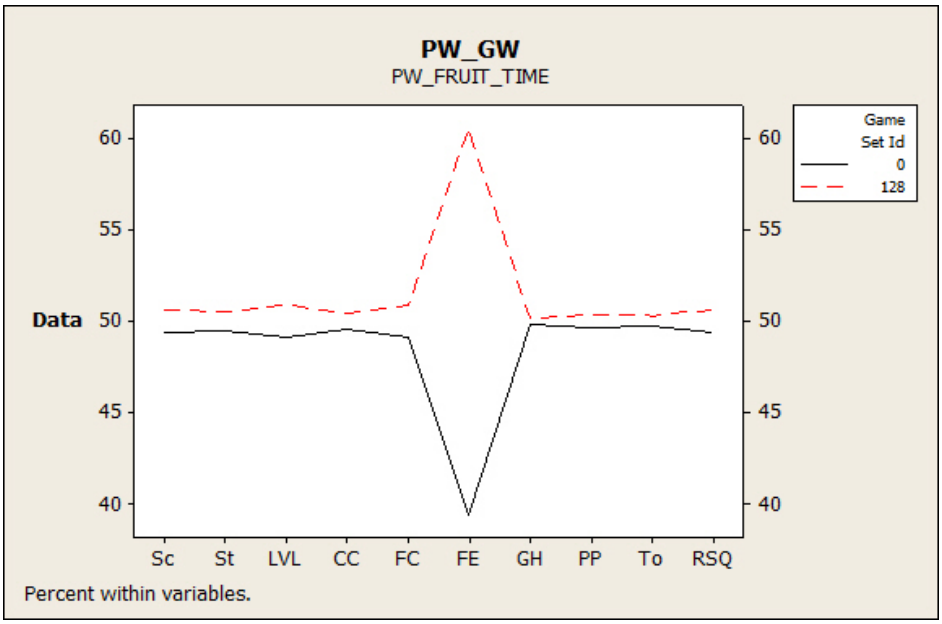


Figure 5.53: Review of the PW_GW game session with low (ID_0) and high values (ID_128) of PW_FRUIT_TIME

Factor FRUIT_FREQ (ID_256) influences the rate of fruit creation. As expected, we see an increase in fruits created and fruits collected. However, we begin to observe a larger dispersion between the number of levels completed and steps. Similar to the results of other algorithm pairs when the frequency of fruit creation has been increased, we observe little to no increase in the number of ghosts, power-pellets or tokens acquired and a score variable value that is only slightly above that of the lower fruit frequency. In this case, the additional fruits are decreasing the chances of Pac-Man completing the level. This is unlike other algorithm comparisons which caused an the increase in number of levels completed as it was thought the additional fruits allowed Pac-Man to lure ghosts away from uncompleted areas of the level.

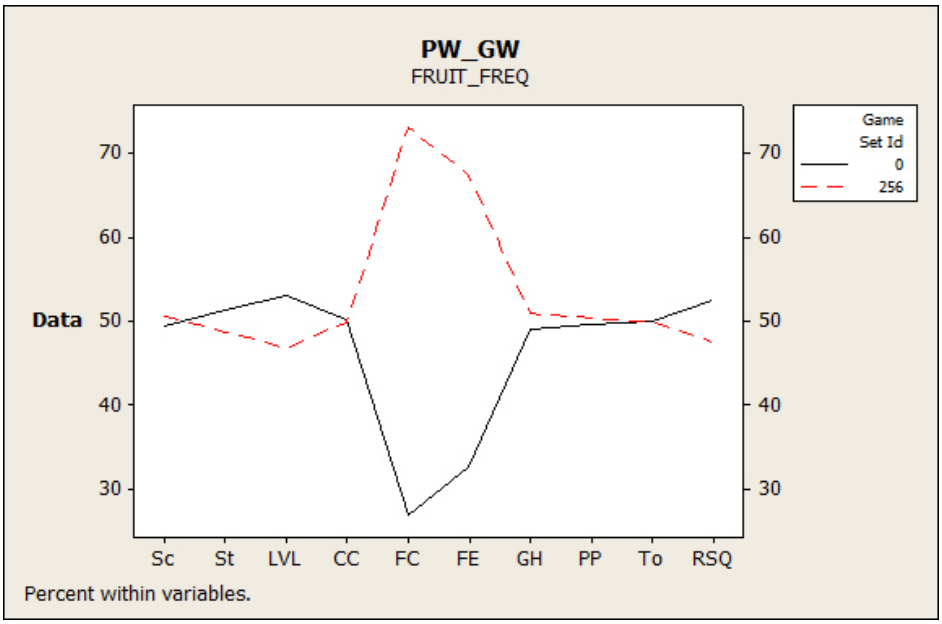


Figure 5.54: Review of the PW_GW game session with low (ID_0) and high values (ID_256) of PW_FRUIT_FREQ

Factor PAC_VIS_LEN (ID_512) increases Pac-Man's range of vision and is generally included as one of the factors in the term analysis. This factor's effect on each response variable varies greatly. The pattern in Figure 5.55 demonstrates a preference towards tokens, power-pellets and thus the completion of levels. This focus minimizes the number of fruits collected and ghosts eaten which are worth higher point values and thus may account for the nearly equal level in terms of the score response. Figure 5.55 illustrates a disproportional reduction in the number of repeated squares to the decrease in the number of steps. This along with the increase in close calls indicates that Pac-Man is utilizing his steps more efficiently, even if this results in Pac-Man being placed in scenarios with higher levels of danger.

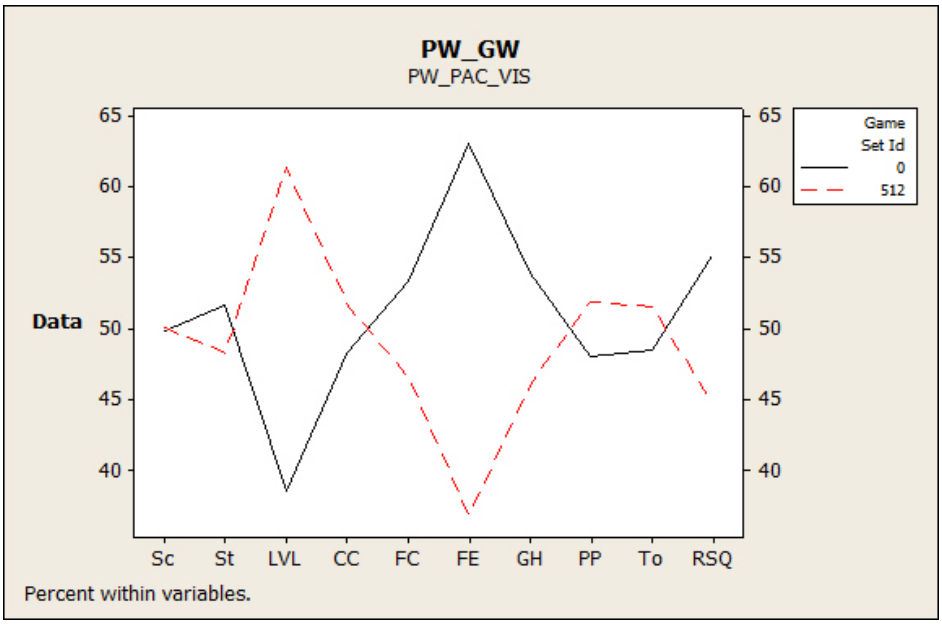


Figure 5.55: Review of the PW_GW game session with low (ID_0) and high values (ID_512) of PW_PAC_VIS

5.3.5 Comparison of Algorithm Performance

This section compares each of the algorithms performance against both sets of opposing algorithms to investigate the global performance of the algorithm.

5.3.5.1 Comparison of SSS-AB* and Ghost Algorithms

Reviewing the performance of the SSS_GW algorithm against the SSS_FLOCK algorithm provides three direct factors for comparison; the flee time, death time and vision range. The GW algorithm has parameters GW_AWAY_GH and GW_TO_GH that perform functions similar to that of the flocking algorithm's separation and cohesion rules respectively. Between both sets of algorithms the vision parameters produce the largest effects, however in the SSS_GW the factors related to the the length of the ghosts' flee and death state times become much more prominent. In terms of similar factors both separation, or GW_AWAY_GH, and cohesion, or GW_TO_GH, prevail as the dominant main effects of their respective algorithm parameters set. An interesting note is that the factor for hunger in flocking algorithm played a large

role, while in the GW algorithm the factor GW_PAC which essentially weights the rule to chase Pac-Man's position while still significant did not contribute to the same extent. An explanation for this result is that with a larger number of rules, each rule contributes less to the final decision. Certain rules help to predict areas that Pac-Man is required to go to accomplish level tasks, as Pac-Man completes portions of the level the number of areas the ghosts have to protect diminishes and provides a greater opportunity for the ghosts to anticipate Pac-Man next area of attack.

The GW algorithm is specifically tuned for the Pac-Man game, while the flocking algorithm is a generalized movement pattern. As such we expected a higher level of competition between the SSS-AB* player and the GW ghosts. In Figure 5.56, we illustrate the results of the SSS-AB* algorithm against the flocking algorithm (FL_BASE) and GW_* for the GW algorithm with low fruit frequency.

This figure illustrates that the SSS-AB* algorithm performs significantly better against the flocking algorithm than against the GW algorithm. A couple of exceptions are the number of close calls and the number of ghosts eaten. These two exceptions indicate that Pac-Man was capable of avoiding ghosts whether being chased or while in predator mode. In comparing the SSS algorithm and the GW factors, we observe that the results are quite similar between GW_SSS_FT, GW_SSS_PAC_VIS_LEN and the GW_SSS_FT. The only major difference is the number of fruit eaten which as expected increases the longer the fruit is available on screen. We observe smaller increases in the number of fruit eaten when Pac-Man's vision range is increased or when the perceived value of the fruit is increased.

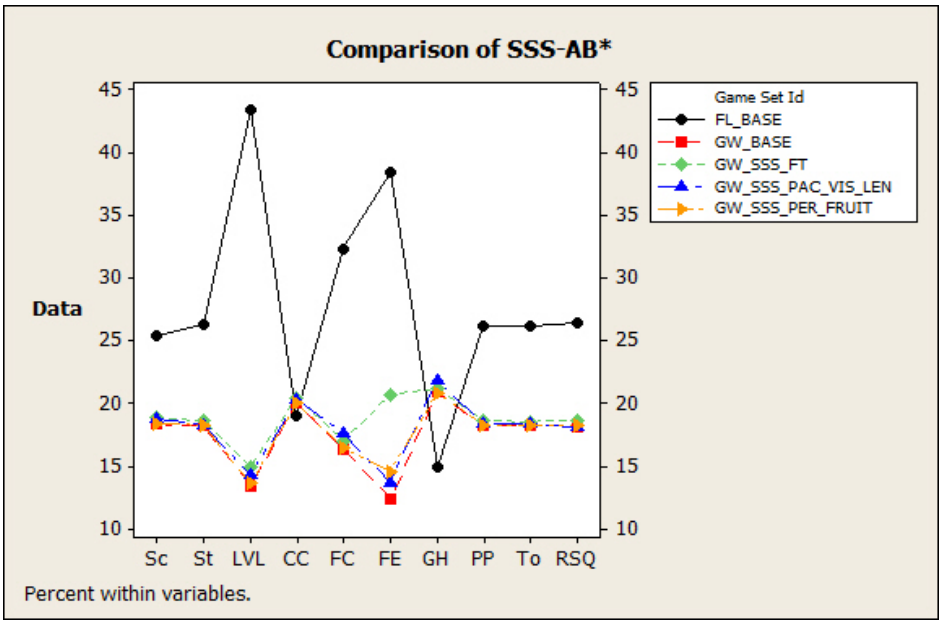


Figure 5.56: Comparison of SSS game sessions results the flocking algorithm and the ghost weighted algorithm. The flocking algorithm (FL_BASE) has only one case as both where near identical except for number of fruit created. The Ghost Weighted algorithm has base (GW_BASE) and factors SSS_PERCEIVED_FRUIT (GW_SSS_PER_FRUIT), FRUIT_TIME (GW_SSS_FT), and PAC_VIS_LEN (GW_SSS_PAC_VIS_LEN).

5.3.5.2 Comparison of Flocking and Player Algorithms

When comparing the performance of the flocking algorithms in SSS_FLOCK and PW_FLOCK, we can compare the four flocking factors, the state factors, the vision parameters and the fruit time. Figure 5.57 displays the average performance of Pac-Man against the flocking algorithm, with each ID representing a factor that was split during the analysis phase. The IDs are similar to previous sections but now include the names of both algorithms used in the experiment, to differentiate between different algorithm set. The PW_BASE represents the base case for the PW_FLOCK algorithm thus all factors are set to their low level value, while the SSS_FLOCK_BASE is the base case for the SSS_FLOCK algorithm. The FRUIT_FREQ factor has been excluded from this figure for clarity and to provide

comparable game settings. Figure 5.57 illustrates that the SSS-AB* algorithm outperforms each session of the PW algorithm for the majority of response variables. One PW session, session number PW_FLOCK_BG produces comparable results in the number of steps and fruit eaten, but produces a higher number of repeated squares. The factor PW_BADGHOST, represented by PW_FLOCK_BG, increases the control against moving towards ghosts. The results indicate that PW_FLOCK_BG outperformed all other PW sessions while maintaining only a slightly higher number of close calls despite a large jump in playing time. It is apparent that the added precaution of avoiding ghosts above the normal level of avoidance resulted in higher achievement in each response variable and a larger number of repeated steps. As expected the SSS_FLOCK (SSS_FLOCK_BASE) proved more efficient in its use of steps than the PW_FLOCK_BG, and as such despite taking a similar number of steps outperformed the PW session on score and level completion while maintaining an comparable number of repeated steps to other ID's.

Interestingly, factor PW_ITEM_CTR (PW_FLOCK_ITEMC), which adds additional weight to moving towards the center of all items, produces fairly poor results in comparison to other ID's, outperforming only PW_TOK_FOR (PW_FLOCK_TOK). Increasing the priority of movement to the center of items greatly increased the number of close calls, and while the number of steps remained similar to other IDs, it had an adverse affect on the number of levels completed.

A review of tables shows that the PW Pac-Man was capable of surviving for a large number of steps, but was having greater trouble accomplishing goals than the SSS-AB* algorithm. The number of close calls is significantly reduced by the SSS-AB* algorithm. This indicates that the Pac-Man was unable to avoid the ghosts with the same level of skill as the SSS-AB* Pac-Man or that the SSS-AB* algorithm simply maintained a greater amount of space between Pac-Man and the ghosts.

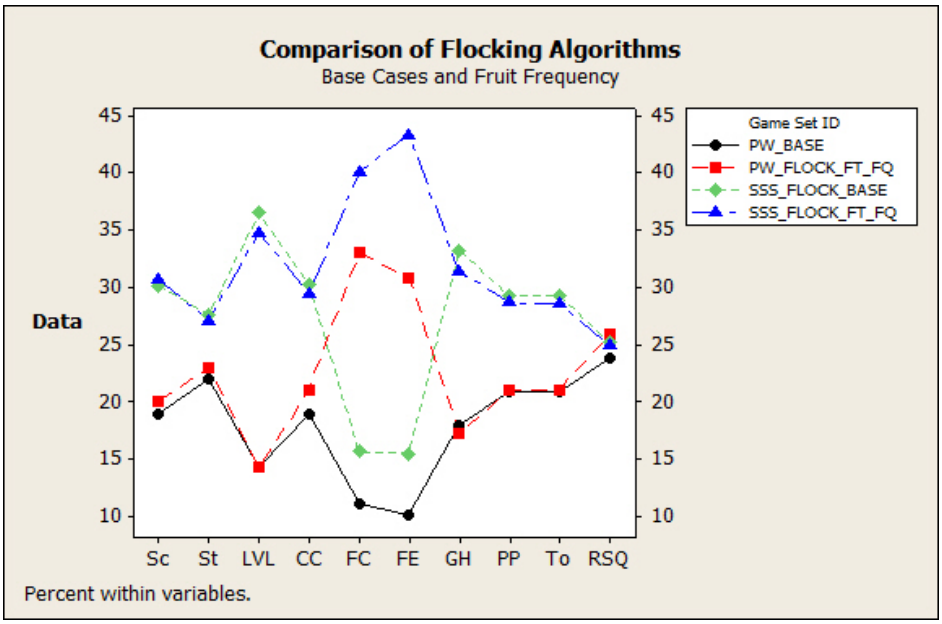


Figure 5.57: Comparison of Flocking game session results between the SSS-AB* and Pac-Man weight algorithm. The SSS-AB* algorithm has case with fruit frequency at a low level(SSS_FLOCK_BASE) and one at the highest level (SSS_FLOCK_FT_FQ). The results of two cases are nearly identical except for number of fruit created (FC) and eaten (FE). The PW_FLOCK pair the base case (PW_BASE) and a case with a high value of fruit frequency (PW_FLOCK_FT_FQ). The increased frequency of fruit produce similar results to the low level value of fruit frequency. We observe a slight increase in the number of close calls (CC) and repeated steps (RSQ), in addition to the increase in fruit creation (FC) and fruit eaten (FE).

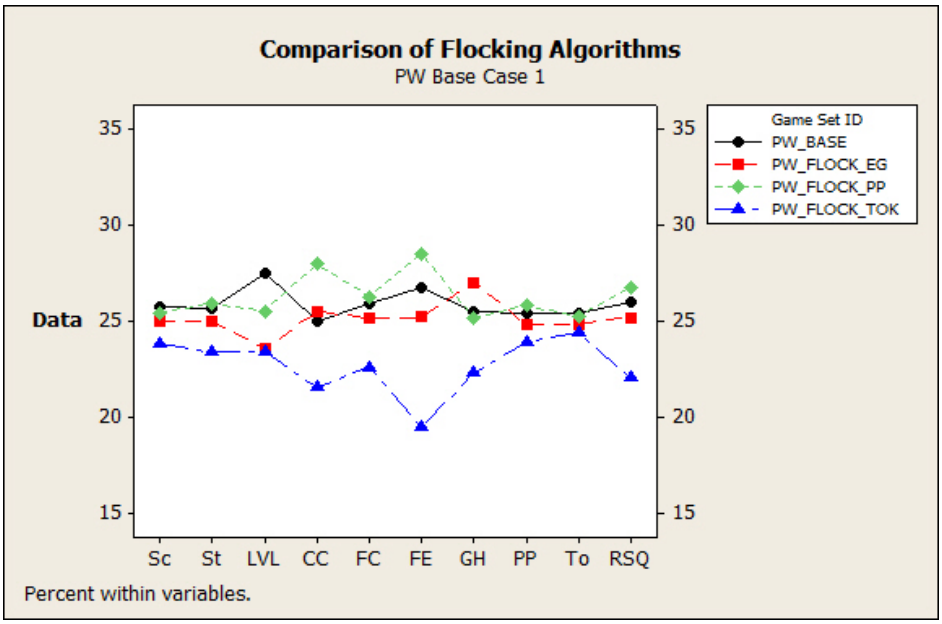


Figure 5.58: Comparison of Flocking game session results Pac-Man weight (PW) algorithm. The PW_FLOCK algorithm has been split into two cases, this case contains the base case level PW_BASE where all factors are at their lowest levels. In addition, we compare the results of the PW_TOK_FOR(PW_FLOCK_TOK), PW_PP_FOR(PW_FLOCK_PP), PW_EDIBLE_GH(PW_FLOCK_EG) factors.

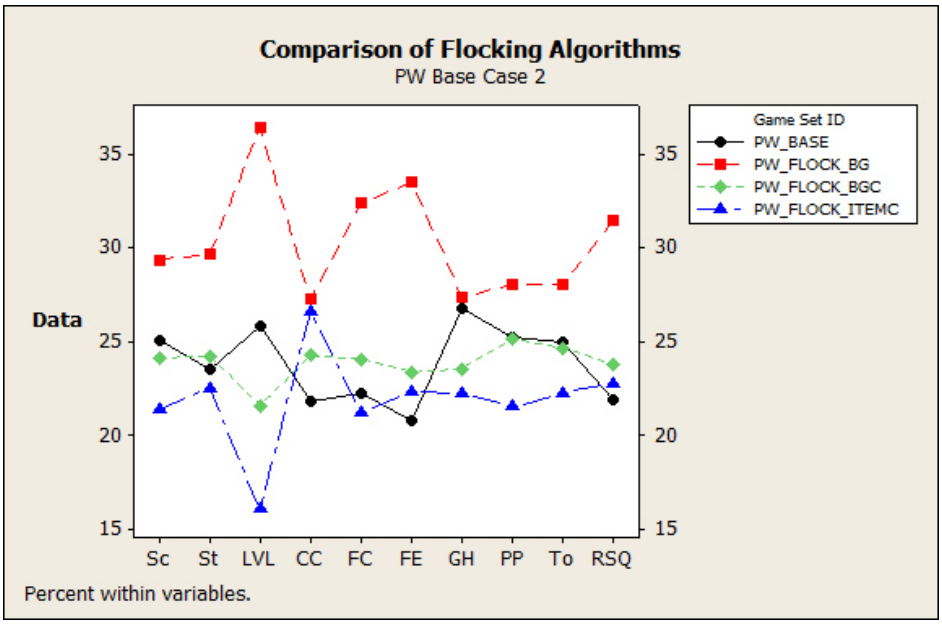


Figure 5.59: Comparison of Flocking game session results Pac-Man weight (PW) algorithm. The PW_FLOCK algorithm has been split into two cases, this case displays the base case level PW_BASE where all factors are at their lowest levels. In addition, we compare the results of the PW_BADGH(PW_FLOCK_BG), PW_BADGH_CTR(PW_FLOCK_BGC) and the PW_ITEM_CTR (PW_FLOCK_ITEMC).

5.3.5.3 Comparison of Pac-Man Weighted (PW) and Ghost Algorithms

In this section we compare the performance results of the PW algorithm's against the flocking and GW algorithms. In Figure 5.60 we illustrate a comparison of the overlapping factors for the PW_FLOCK and PW_GW algorithms. Each ID corresponds to a factor of the PW algorithm. Overall, the largest response improvements are produced by the PW_BADGH factor illustrated by (PW_FLOCK_BG). Figure 5.60 highlights several patterns in the PW algorithm performance, the first being the largest variation occurring in the number of levels completed which appears to be nearly evenly split between a high and low value. The score, number of steps, power-pellets, tokens and repeated squares responses have fairly consistent ranges of values. The result illustrated in Figure 5.60 shows that the response variables for number of levels completed, fruits and ghosts eaten are responsible for the largest variation among the games, indicating a good potential for adaptation for the response score. If we can identify pivotal points that greatly affect those responses, we can adapt factors to potentially broaden the resulting range of scores. However, a response variable such as the number of steps displays a reduced range with minimal opportunity for improvement, thus it may not be a good candidate for adaptation.

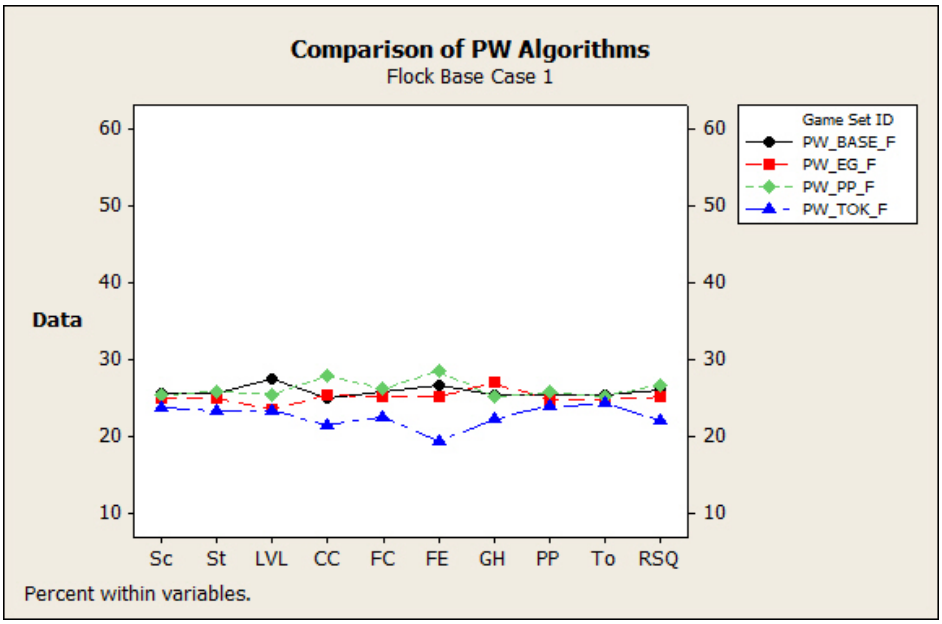


Figure 5.60: Comparison of PW algorithm game session results against the Flocking algorithm. This figure illustrates the base case of flocking and PW algorithm(PW_BASE_F). The factors for the PW algorithm comparison are: PW_EDIBLE_GH (PW_EG_F), PW_POWER_PILL (PW_PP_F) and PW_TOKEN (PW_TOK_F).

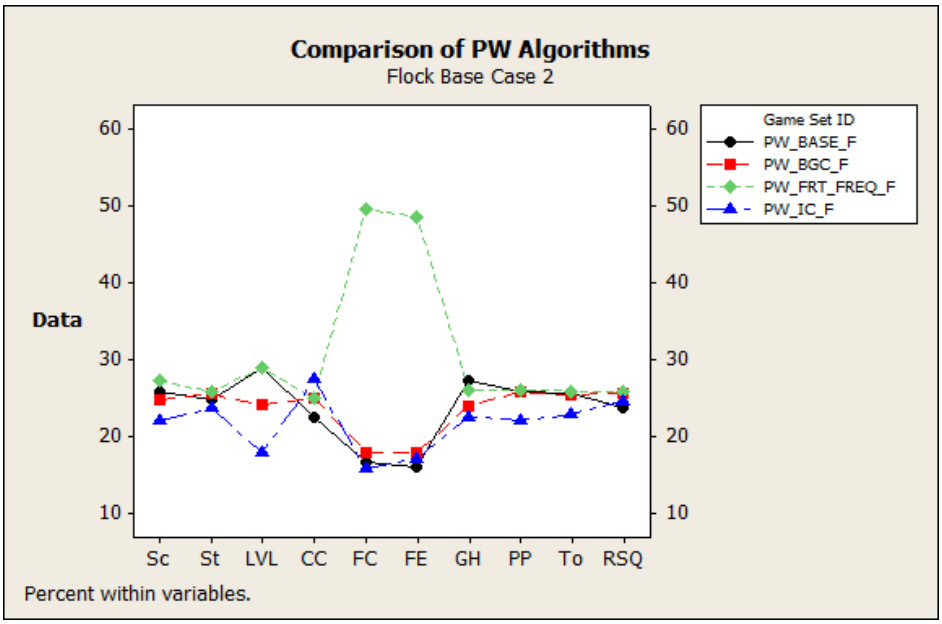


Figure 5.61: Comparison of PW algorithm game session results against the Flocking algorithm. This figure illustrates the base case of flocking and PW algorithm (PW_BASE_F) and the following factors: PW_BADGH_CTR(PW_BGC_F), PW_ITEM_CTR (PW_IC_F), and the fruit frequency (PW_FRT_FREQ_F).

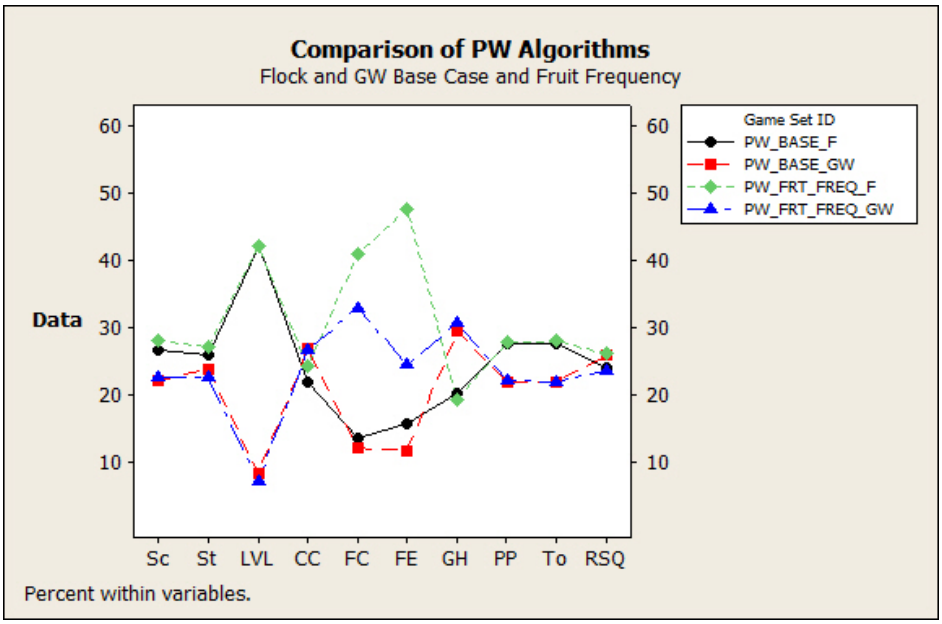


Figure 5.62: Comparison of PW algorithm game session results against the Flocking and Ghost weighted heuristics algorithm. Flocking factors are labeled via PW_FLOCK_*, while ghost weighted heuristics algorithm factors are la belled PW_GW_*. The base cases are FLOCK_BASE for flocking and GW_BASE for ghost weighted heuristics algorithm. The factors for the PW algorithm comparison are: PW_BADGH(PW_BG), PW_BADGH_CTR(PW_BGC), PW_EDIBLE_GH (PW_EG), PW_ITEM_CTR (PW_IC), PW_POWER_PILL (PW_PP) and PW_TOKEN (PW_TOK).

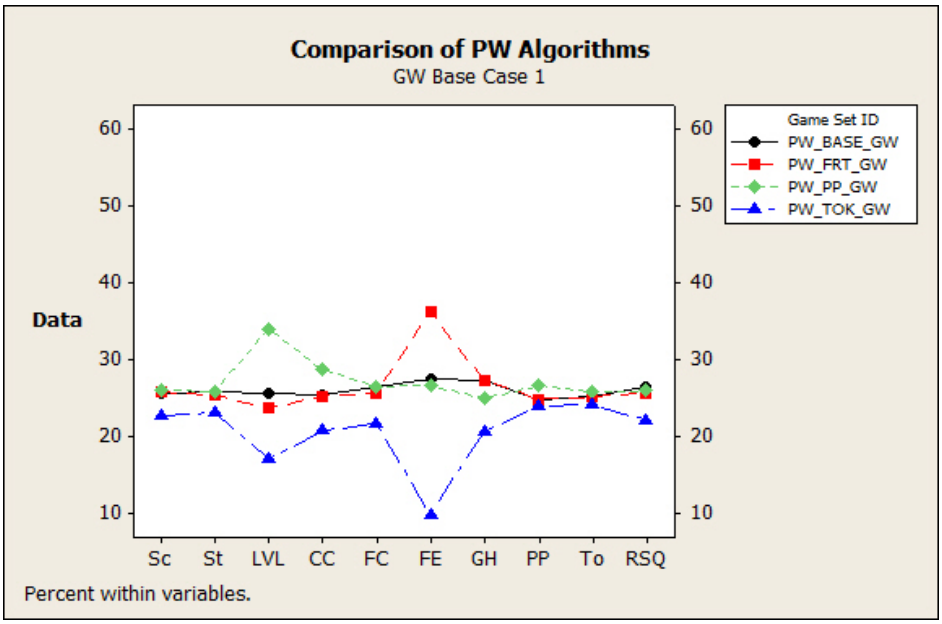


Figure 5.63: Comparison of PW algorithm game session results Ghost Weighted (GW) algorithm. This figure illustrates the base case of GW and PW algorithm(PW_BASE_GW).The factors for the PW algorithm comparison are: PW_FRT (PW_FRT_GW), PW_POWER_PILL (PW_PP) and PW_TOKEN (PW_TOK).

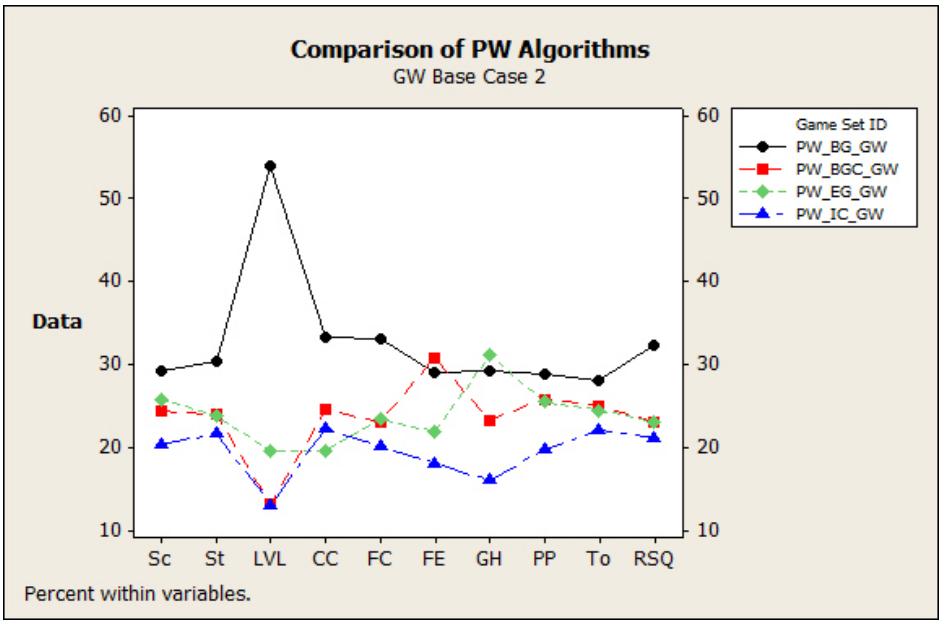


Figure 5.64: Comparison of PW algorithm game session results against the Flocking and Ghost weighted heuristics algorithm. The factors for the PW and GW algorithm are: PW_BADGH(PW_BG_GW), PW_BADGH_CTR(PW_BGC_GW), PW_EDIBLE_GH (PW_EG_GW) and PW_ITEM_CTR (PW_IC_GW).

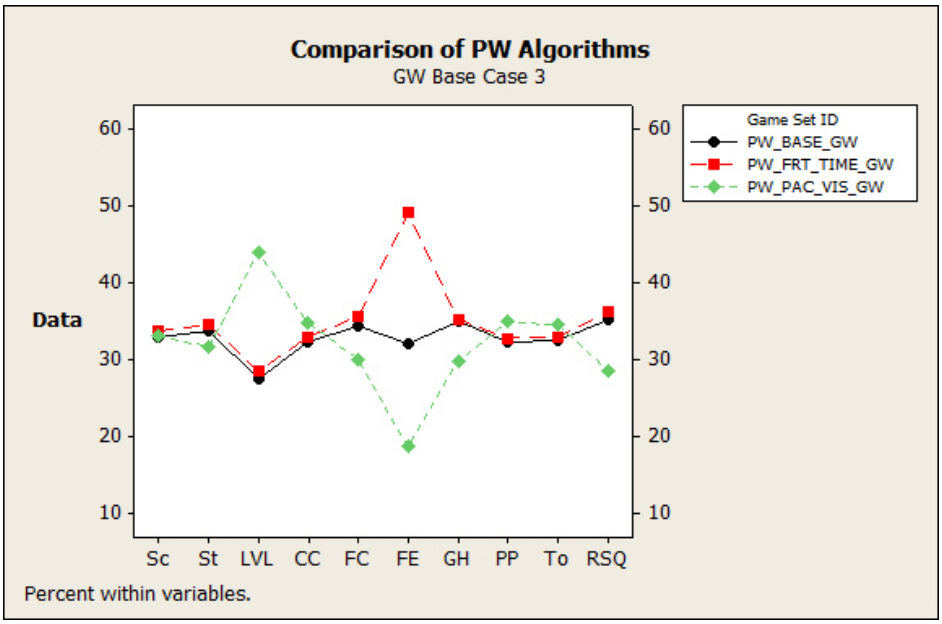


Figure 5.65: Comparison of PW algorithm game session results against the Ghost Weighted (GW) algorithm. This figure illustrates the base case of GW and PW algorithm(PW_BASE_GW). The factors for the PW algorithm comparison are: PW_FRT_TIME (PW_FRT_TIME_GW) and PW_PAC_VIS (PW_PAC_VIS_GW).

5.3.5.4 Comparisons to Ghost Weighted (GW) and Player Algorithms

Figure 5.67 compares the performance of the GW algorithm against the SSS-AB* algorithm and the PW algorithm. The SSS-AB* algorithm performs efficiently and scores uniformly higher for the three factors with a similar number of steps. The number of steps response variable produces similar results for both algorithms which may indicate that the GW algorithm produces a level of challenge beyond the scope of both players. Figure 5.67 indicates that the PW algorithm put a higher priority on the fruits and was capable of obtaining the fruits during game play. While the SSS-AB* focused less on the fruit and more on the eating ghosts. Figure5.67compares the base case of both algorithms with increased vision range for Pac-Man. The SSS-AB* algorithm performance stays consistent or slightly improves for all response variables except fruit eaten. The range of vision aids the PW algorithm perform better in completing additional level tasks, at the expense of bonus tasks but overall fails to vastly improve the score. Increasing the vision range of Pac-Man greatly decreases the number of repeated squares for the PW algorithm. The number of repeated squares demonstrated improved results when the vision range increased for the PW algorithm, yet the SSS-AB* remained nearly the same.

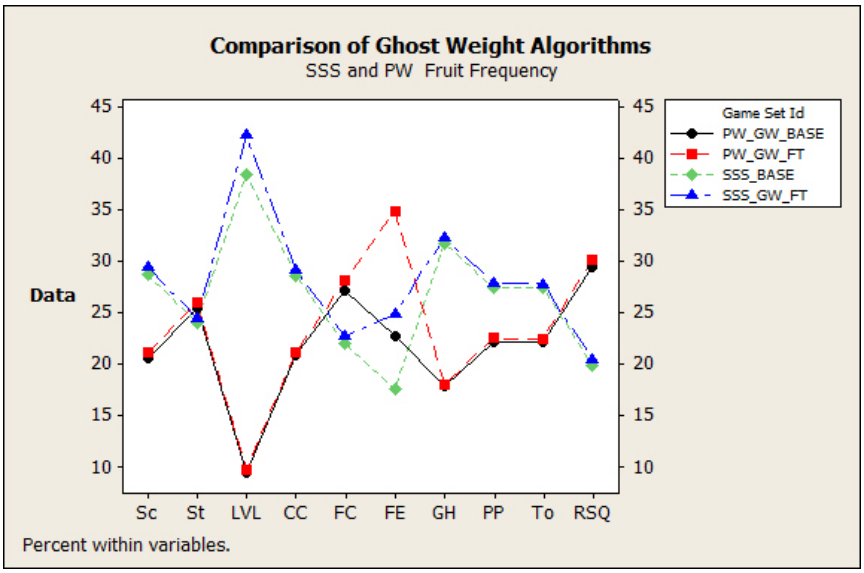


Figure 5.66: Comparison of results from GW algorithm game session against the SSS-AB* and Pac-Man weighted algorithm. The factor's algorithm are indicated by PW_GW_* for the Pac-Man weight algorithm sessions SSS_GW_* for the SSS-AB* sessions. The base cases are identified by SSS_BASE and PW_BASE, and indicate all factors at their lowest level . The factors are length of time the fruit is available (PW_GW_FT and SSS_GW_FT).

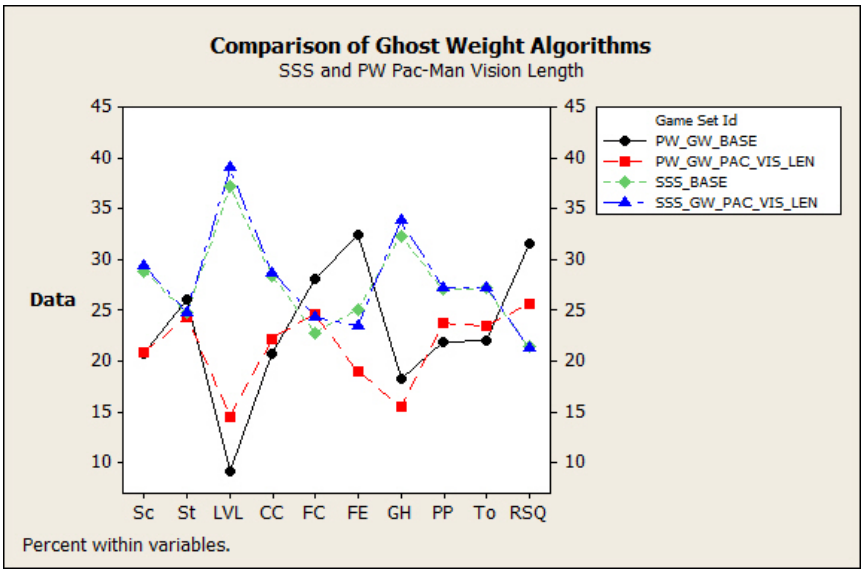


Figure 5.67: Comparison of results from GW algorithm game session against the SSS-AB* and Pac-Man weighted algorithm. The factor's algorithm are indicated by PW_GW_* for the Pac-Man weight algorithm sessions SSS_GW_* for the SSS-AB* sessions. The base cases are identified by SSS_BASE and PW_BASE, and indicate all factors at their lowest level . The factors are the range of Pac-Man's vision (PW_GW_PAC_VIS_LEN and SSS_GW_PAC_VIS_LEN).

5.4 Experimental Environment Summary

The following sections summarize the results from the offline experiments described throughout the previous sections of this chapter.

5.4.1 Model Evaluation Summary

In the model evaluation portion of the experiment we calculated the R-Sq values for each algorithm pair. The R-Sq value provides an assessment of the amount of variance described by the model, given a maximum of 127 terms per model. The means of R-Sq(adj) values ranged 53-74% for the GW algorithms and 68-82% for the flocking algorithms. The lowest value occurred during the PW_GW algorithm, while the highest occurred for the SSS_FLOCK algorithm. The model evaluation is

a portion of the process which potentially could be omitted from the implementation process, however it does provide two useful pieces of information. The first relates to the number of terms required for the adaptive system. If our model explains a large portion of the variance with only 127 terms and the lack-of-fit is not significant there is a possibility that no additional terms are required. This could prove beneficial for storage cost. Additionally this portion of the experiment indicated which response variables should be easier to control. In the results in Section 5.2.3 we identified that adapting the game to control the collection of fruit or the number of levels completed would be the most difficult. Although this section indicates difficulty in predicting and explaining specific response variables results, the PW_GW results indicate that the response variables score and number of close calls are two of the more difficult variables to control, yet we produced fairly positive results in controlling them.

The lack-of-fit testing determined whether the terms excluded from the model played a significant role in the variation of the experiment. As part of our experiment we separated the factorial design into small cases, so that the analysis could be completed. As mentioned, each case contained 10 factors which limited the maximum number of terms per model to 1024, but increased the number of models. Our testing indicated that none of the lack-of-fit tests for any of the models for any algorithm proved to be statistically significant. This demonstrates that our model could not be greatly improved by the inclusion of additional terms. The lack-of-fit testing proved the commercial limit of 127 term did not play a significant role in limiting the results of our separated case models, although the term limit was the cause of separating the models based on large number of factors. As we observed in attempting to reconstruct the SSS_GW models was that the lack-of-fit values were statistically significant.

5.4.2 Factorial Analysis Summary

The factorial analysis portion of the experiment identified factors which played a significant role in the variation of the response variables in the simulation. The first portion of this study identified statistically significant terms that contributed the largest term effects to the response variables. Due to the large number of terms in

the experiment, we presented only the top results for this analysis. This method was expected to identify and quantify the terms effects, some of which intuitively appeared obvious before the analysis. However, other important applications of this analysis is the identification of insignificant terms and factors, unexpected significant factors and interactions which can explain previously unexpected modifications to the level of difficulty. Results indicated that the vision parameters consistently had the largest effect on the majority of the response variables. As expected, Pac-Man's results improved with increased vision range and decreased with the ghosts' vision range. Each algorithm had a unique set of factors; a number of the factors implemented similar functionality. Across the ghost algorithms we observed a pattern that the most significant factors dealt with the separation and cohesion of the group.

After identifying the statistically significant terms, our analysis reviewed the terms' effect sizes to analyze the practical significance of the terms. The practical or game significance of the terms provides additional information for our prediction methods on how each term will alter the results of the response variables. The game significance is more useful to game designers than the statistical information. If a game designer wants to modify an aspect of the game, they would review the factor effect sizes and consider what has practical value for their game, rather than review only the statistical significance.

The third part of the analysis performed a simple comparison of the high and low levels of factors which were omitted from the term effect calculations due to size constraints. This method of analysis provided a useful tool for observing the results of the factors without the term effect size. This form of analysis is well-suited for level designers wishing to experiment with values for two or more factors. Although the interaction information is not accessible for this comparison, it provided interesting insight into how each of the main factors was effecting each of the response variables. The ability to graphically visualize the results highlighted the trade-offs between the different response variables. The frequency of the fruit creation was the only common factor for each of the algorithm pairs and provided interesting results varying from only minimal increases to score and fruit totals to a larger role that altered the

length of the player's life and drastically altered the number of close calls and levels completed.

The final analysis task assessed the global effects of comparable factors between the algorithms. As expected, we identified the flocking algorithm as the weaker defensive algorithm of the pair. The weighted ghost algorithm had a consistently strong performance against both the SSS-AB* and the PW algorithm, indicating a high level of performance or challenge. The SSS-AB* algorithm performed better for the majority of response variables. It focused less on the bonus items and more on collecting ghosts which added to the longer life span. On occasions which the two player algorithms had nearly equivalent life spans, the SSS-AB* algorithm was more efficient, utilizing the same number of steps to produce slightly higher values in the scores.

Chapter 6

The Adaptive System

One of the goals of this thesis is to build a prototype system that adapts game parameters based on the outcomes from the factorial analysis in the previous chapter. This final chapter describes the implementation of an adaptive system which attempts to control the progress of the game. First we discuss the additional calculations and information that helped build the player models for the adaptive system in Section 6.1. Next, in Section 6.2 we discuss our selection of response variables to measure and observe during the adaptation and the goals for the adaptive system. After selecting the response variables to control, we discuss creating and tuning performance heuristics in Section 6.3. Finally, we present the results in Section 6.4 for each of the game sessions being controlled by the adaptive system. We will conclude this section by presenting our success rates for controlling the response variables within a specific target range of results.

6.1 Term Loading for Adaptive Pac-Man

As previously mentioned, one of the limitations of commercial software is the inability to load a large number of terms into the model. This limitation is less of an issue in the adaptive system, as we can specify the maximum number of terms to be loaded. The commercial software limit on the number of loadable terms provides a reasonable estimate of the number of terms required to be included and the expected explanation using only 127 terms. If we are producing adequate R-Sq(adj) values with 127 terms and the lack-of-fit is not significant, given that our models were loaded with the largest factor effects first, including additional terms may not improve results or be needed if space requirements are tight.

The initial step of the adaptive system experimentation was to create effects tables for each of the response variables. In the initial part of the experiment discussed in the experiments in Chapter 5, we identified significant factors and calculated effects for all of those terms. This information is used to build the effects tables. An effects table could be considered a list of all terms and the size of their effects. However, because we are only adapting to factor levels that previously appeared in the experiment, we can precalculate the full effects of each factor setting. Thus, rather than performing a real time calculation to determine a game factor's possible effect on the game every time we wish to make a modification, we precalculate and store the sum of each factor interaction into a table.

Impact Size	Factor IDs	Factor Setting
-10.665	A*B*C*D*E*F*G*H*I*J	1010000001

Table 6.1: An example effect table row. The Factor Setting identifies which of the 10 factors (A- J) should use a high value and the total impact of these adjustments would be -10.665. In this case, the term would be A*C*J would use their high values.

Table 6.1 is an example row as it would appear in the effects table. The first number is the total impact for this factor setting. We will use the terminology impact size to indicate the sum of all factors and interactions, using the level value corresponding by the factor settings. The second string is a list of all potential factor IDs. Finally the third number is a bit map indicating which factors are using their high level value. In this case A*C*J would be activated, -10.665 is the sum of A, C, and J's main effects and their interacting effects, A*C, A*J, C*J and A*C*J at their high level and the rest of the terms at their low level effect size. This optimization means that our effects tables contains 2^k impact sizes. Our implementation has consistently used k = 10 or 1024 total effects in the table. This restricts the number of factor settings to which we could potentially adapt the game to 1024.

The effect table provides important information about the expected results of performing each adaptation. Each effect table provides an ordered list of each factor setting's impact on the game, providing us with an estimate of the maximum and minimum values or interval of adaptation that we can expect to occur in the the

game. This interval is an estimate of the results as the combination of continual modifications could produce unexpected results outside of this interval. The effect table includes other potentially useful pieces of information such as the maximum and minimum adaptation stride, which could be used to control the granularity of adjusting the game. The effect table could also be used to identify ranges where insufficient information is available. For instance if the factor settings adjacent to the current settings have large differences in the impact size, we could identify this situation as one needing extra precaution in making adjustments.

The next step in developing the adaptive system included collecting and calculating the coefficients of the regression equation. These were calculated during the analysis of term significance. A coefficient is the mean of a response variable through all of the runs and repetitions. The sum of the coefficient value and the factor setting effect size from the effect table indicates the total expected value of a response variable for that factor setting over the course of a full game. The sum of the coefficient and the effect size of the factor setting is the value that will be compared to the value produced by the heuristic to assess the difference between the current difficulty and the difficulty required to produce results within the expected interval.

6.2 Response Variable Selection

The next phase of the adaptive system experiment included selecting response variables to adapt during the game and setting target interval for the final results. We selected a game session from each algorithm pair to experiment with a diverse range in the adaptation process. From the SSS_FLOCK algorithm we selected the game session with low fruit frequency. As both game sessions were highly similar this selection was straight forward. For the SSS_GW algorithm we selected the session where FRUIT_TIME used its high level. The FRUIT_TIME was selected for the SSS_GW algorithm due to the variance in the number of fruits collected response, while producing similar results in other response variables. For the PW_FLOCK and PW_GW algorithms we selected the game sessions with the PW_BAD_GHOST pa-

rameter using its high level value. We utilized the PW_BAD_GHOST parameter because it proved to have the most positive influence of all the comparisons for the PW algorithm.

Once the game sessions were chosen, we selected response variables to attempt to control. For the SSS_FLOCK algorithm we would attempt to control the score response. In the SSS_GW and PW_FLOCK algorithm the game would adapt to control the resulting number of steps. Finally, for the PW_GW algorithm we selected to control two response variables: the score and the number of close calls. We selected the score response to adapt, because the player perceives the score as a marker for success and failure. However, the score can be difficult to control due to its progression being non-linear in nature and because Pac-Man has a variable rate of points he can obtain each life. As an example Pac-Man can be revisiting squares in an attempt to collect a fruit or eat a ghost, as such, no additional points would be awarded but there is no increase in difficulty either. Thus, we selected the number of steps as a second response variable to control as it provides a more consistent view of the progress of the player. Finally, we wished to experiment with controlling multiple response variables to demonstrate potential future usage. Here we included the score and number of close calls response. These two response variables were selected because we felt that players utilize the score as an important measure of success while the number of close calls could alter the perceived level of challenge while potentially achieving similar scores.

The next set of Tables 6.2, 6.3, 6.4 and 6.5 illustrate the expected maximum interval for adaptation for a response variable. They illustrate that the adaptation process is easier for some player than for others, as some tables have a large absolute values for the maximum but not the minimum. The numbers in these tables indicate the largest positive and negative impact sizes for each algorithm and the coefficient for the controlled response variable in the regression equation. The largest impact sizes in conjunction with the coefficient help support our selection for the target interval. As the coefficient is the mean of all repetitions of experimental runs, it provides an estimate of the expected results. Our goal is to select the target interval within the

bounds of the absolute maximum impact sizes, preferably a relatively safe distance from the boundaries of the impact size so as to provide the greater number of options for adaptation. If we select target intervals near the very outskirts of the effect interval we risk several issues during the experiment. The first issue is that adapting to the outskirts is that the opportunities for adaptation are heavily reduced due to minimal factor setting selection defined in that set of values. The second issue is that the stride of adaptation increases toward the boundaries of the impact interval, as such we may be unable to match or converge upon the required level of granularity for the level of change and it may result in our target goals being near impossible achieve.

Negative Score Effect	Positive Score Effect	Regression Coefficient
-561.7	985.8	1911.8
-528.4	813.3	
-522.6	789.1	

Table 6.2: SSS_FLOCK Largest positive and negative Score effect

Negative Steps Effect	Positive Steps Effect	Regression Coefficient
-58.583	124.417	263.166
-56.4165	120.083	
-54.083	118.417	

Table 6.3: SSS_GW Largest positive and negative Step effects and the regression equation coefficient.

Negative Steps Effect	Positive Steps Effect	Regression Coefficient
-105.769	190.398	374.871
-103.102	184.231	
-97.2691	178.564	

Table 6.4: PW_FLOCK Largest Factor Setting Effect

For each game session and controlled response we selected target zones towards which we attempted to direct the result of the game. Table 6.9 illustrates our selections for the target interval for each response, as well as the number of runs which occurred above, below or in the target range before the adaptive process was implemented. Our selection process for the response variable intervals attempted to cover a basic scope of adaptive situations. For the first two cases we selected an interval

Score Impact Size	Close Call Impact Size
-257.291	-68.5687
-255.624	-65.7354
-241.457	-65.0687

Table 6.5: The lowest negative score and close calls impact sizes for the PW_GW algorithm.

Score Impact Size	Close Call Impact Size
696.044	266.264
642.709	191.765
607.709	187.264

Table 6.6: The highest positive score and close calls impact sizes for the PW_GW algorithm.

in the middle of the results to replicate a slight increase or decrease in the level of difficulty to have a balanced game. In the final two cases we selected intervals which was tilted towards the high end or low end of results. This situation was selected to replicate a scenario where a large number of alterations were required to drastically alter the results, that is where a player model is required to drastically alter the current performance to produce desired results.

6.3 Heuristics

In building our heuristics to predict Pac-Man's performance we utilized two methods. For the SSS algorithms we built the heuristic functions using portions of our performance measures. For the PW algorithms we utilized regression equations along with minor adjustments. An important issue arose from developing our system is that the term effects were calculated over the duration of a full game, however our adaptation process performs adjustments at set interval times during each life. Thus, the first step in developing our heuristics was to isolate the performance of Pac-Man for each life and provide an expected value for each of the response variables. For our selected control response variables, we observed that the score response variable decreased over the three lives, with the first life on average producing the highest score. While the first life produced higher values for the number of close calls and

Score Coefficient	Close Call Coefficient
1144.58	144.804

Table 6.7: PW_GW Score and Close Call Coefficients.

Score Coefficient	Close Call Coefficient
1144.58	144.804

Table 6.8: PW_GW Score and Close Call Coefficients.

steps, the performance remained fairly consistent over the second and third life, with the third life value even producing higher values for the PW algorithm. The mean performance for each life for each algorithm can be found in Appendix C.

The first heuristic was designed for predicting the steps of the SSS_GW game session. The reason for choosing to develop this heuristic first where: designing for the number of steps is a simpler task than designing for the score, and the result for the number of steps provides a view of continuity and consistency. After reviewing the results for this algorithm, we identified that the number of power-pellets remaining played an important role in the number of steps the player would complete during a life. We designed the heuristic to use the mean number of steps per power-pellet based on the mean performance of the current life of Pac-Man multiplied by the number of power-pellets left on the board. The mean performance of the current life, would use the means from all simulations of the experiment and separate the statistics based on which life points where collected. So, if Pac-Man is on his first life the mean performance would only compare information that is achieved during the first life in the experimental runs. We performed similar calculations for the mean steps per token. Since the number of tokens is quite high, we utilized a dampening factor for the mean steps per token, which starts at 4 and decreases to 1.

The second heuristic has been designed to predict the score for the SSS_FLOCK algorithm. As previously mentioned, the score is more difficult response to control. If the player collects a bonus item or a couple of additional ghosts the results can easily go awry. An additional area of difficulty is that a heuristic that estimates the score likely has to estimate the number of steps as well. Initially the creation of this heuristic utilized the heuristic created for the SSS_GW steps simulation. In the

Algorithm	Response	Target Interval
SSS_FLOCK	Score	1600-2000
SSS_GW	Steps	220-265
PW_FLOCK	Steps	320-400
PW_GW	Score	900-1050
	Close Calls	70-120

Table 6.9: Selected response variables target interval prior to the activating the adaptive system.

Algorithm	Target Interval	Below Interval	In Interval	Above Interval
SSS_FLOCK	1600-2000	333	279	420
SSS_GW	220-265	327	285	418
PW_FLOCK	320-400	563	252	210
PW_GW	900-1050	199	277	555
	70-120	234	262	532

Table 6.10: Results from the experimental phase we have calculated the number of results to occur in the target interval prior to the activating the adaptive system.

GW algorithm the player died slightly faster, thus the dampening factors had to be made to alter in larger increments. In addition we tweaked the SSS_GW heuristic by creating an additional catch all for any estimates below the current number of steps. Once we had an estimate for the number of steps, we compared the current rate of scoring per step in relation to the mean rate of scoring for this specific Pac-Man life. During the early portion of the life we would utilize the mean rate of scoring per life. As the life progressed we would utilize the player's current rate of scoring to make adjustments. Using the mean rate of score per life during the early portion of the life was to avoid over-correcting for the natural progression of scoring which begins very high and decreases throughout game play.

For the third heuristic we develop utilized a regression equation to predict the number of steps for the PW_FLOCK algorithm. The regression equation was built based on the regression coefficient and the player's current score, the number of levels completed, the number of close calls and the number of fruit, ghosts and power-pellets collected. For this heuristic we actually built three separate regression equations one for each of Pac-Man lives. Information for the regression equation can be found in D.

The fourth and final set of heuristics were designed for the PW_GW algorithm

to predict the score and the number of close calls. We previously mentioned the difficulties in predicting the response variable score. Predicting the number of close calls proved to be equally elusive. Even our regression equation produced low prediction rates for the number of close calls. Ultimately, we used the expected number of close calls from the results for the current factor settings. The second heuristic developed for this section was used to predict the score. We utilized a regression equation to predict the score. However, unlike the previous example, we used only one regression equation for the entire game instead of the one per life. In addition, we populated the regression equation with the maximum of the current value or the mean life value to calculate the result. Thus, initially the game would adapt based on the mean, but if the current game results progressed above the mean we would adapt to that information. The regression equation for the score for the PW_GW algorithm used the following response variables to predict the score: the number of steps, levels complete, close calls, ghosts collected, power-pellets, and repeated steps. Information for the regression equation can be found in D.

6.4 Results

Each algorithm with its selected factor settings was rerun with the adaptive system activated. The adaptive system attempted to control the pace such that the results of the response variables would finish within preset target intervals. In the previous section we selected those response variables and control intervals. This section compares the results of the experiment runs with the adaptive system using the heuristics described in the previous section.

Utilizing each of the heuristics we were capable of improving the results of each of the target response variables. Our goal was to increase the number of game sessions within the target interval for one of the response variables by at least 5 percentage points. This goal was accomplished for each algorithm set. For the SSS_FLOCK algorithm we improved the number of games in the target interval from 27.03% to 35.19%. The greatest improvement came from the first heuristic we developed which

initially had 27.87% of game sessions in the target interval and adapted to 39.61%. The PW_FLOCK adaptive session controlled the score response and showed improvement from 24.59% to 30.93%. Finally, the PW_GW adaptive game session attempted to control both the score and the number of close calls improved from 26.87% to 28.81% for the score and 25.48% to 33.59% for the number of close calls. Interestingly for the final case, we could have selected heuristics which would have better control over the score response variable. However, the final heuristic proved capable of maintaining the same interval of scores while increasing the number of close calls. This illustrates the ability to increase the perceived difficulty, while allowing the player to achieve results within the same score interval.

Algorithm	Response	Target Interval
SSS_FLOCK	Score	1600-2000
SSS_GW	Steps	220-265
PW_FLOCK	Steps	320-400
PW_GW	Score	900-1050
	Close Calls	70-120

Table 6.11: Results after the adaptive system has allowed adjustments to occur, the number of game session in targeted results range has increase for each of the response variables.

Algorithm	Target Interval	Below Interval	In Interval	Above Interval
SSS_FLOCK	1600-2000	440 (+107)	373 (+107)	247 (-173)
SSS_GW	220-265	230 (-97)	408 (+123)	392 (-26)
PW_FLOCK	320-400	348 (-215)	317 (+65)	360 (-150)
PW_GW	900-1050	283 (+84)	286 (+9)	463(-92)
	70-120	222 (-12)	344 (+82)	458 (-74)

Table 6.12: Results after the adaptive system has allowed adjustments to occur, the number of game session in targeted results range has increase for each of the response variables.

6.5 Adaptive System Summary

The adaptive system portion of the experiment used the results from the factorial analysis phase to actively adapt the game at set time intervals. This part of the

study relied on two main design issues, the first being the creation of the heuristic to predict the current level of success for a selected response variable and the second being the adaptation of a game setting to direct the player's success towards the targeted interval. Since we designed a simplified version of the adaptive system, we set our goals to be within an achievable interval given the number of key omissions from the adaptive system. The development of our heuristics used regression equations on the assumptions of effective performance measures. The creation of these types of heuristics maybe a more suitable task for machine learning algorithms, as producing the four heuristics required trials and experimentation.

Overall the results from the adaptive system were positive. We set the goal of controlling the game sessions to increase the number of sessions in the targeted interval by 5% percentage points. We accomplished the desired level of result, improving the number of games within the interval by 5% - 12% points. Although we could have tuned the heuristics further we felt these results conclusively indicated the potential of the system. It is important to remember that individual cases are not comparable. Our results indicate that a greater number of cases of the hand-crafted heuristics finished within the target interval. However, each adaptive simulation is a unique case, containing unique models of the player's performance and different response variables. As an example, explaining the variation with the PW_GW algorithm set is more difficult due to the larger number of factors. Since each case represents different algorithms, player models and response variable, we are indicating that we can be adaptive for a variety of cases. but this should not be viewed as promoting hand-crafted heuristics above regression equations.

In particular the results for the PW_GW algorithm which controlled for the score and the number of close calls proved to be quite promising. Although we produced only a small improvement in controlling for the score, we had tremendous success in improving the number of close calls in the interval. This particular case was also of interest because it pitted the benefits of two response variables against each other. In certain scenarios, the adaptive system could be unable to select a factor setting which improved the results of both response variables, similarly it may be

forced to select factor setting which comprised the optimal selection of both response variables to accommodate a cumulative improved response. The final point of interest for adaptation is that current research has already begun having success mapping multiple response variables to player's emotional states, such as fun or frustration [57].

Chapter 7

Conclusion and Future Work

This chapter highlights the contributions of our experiment and methodology, in addition we discuss logical methods of progression and potential applications for future work from the the results of this thesis.

7.1 Contributions

This thesis investigated a methodology of identifying and quantifying significant factors relating to the difficulty of a video-game. We utilized a factorial analysis approach which focused on understanding how different factors influenced a set of response variables. The set of the response variables could potentially describe the challenge of the game, faced by each of the algorithms. Quantifying each of the term's effect size for each response variable was the key step performed during the factorial analysis. This step allowed the adaptive process to select and adapt appropriate factor settings to meet the target control interval. After processing the term effect size for each response variable, we built a simple prototype of an adaptive game system to test the results. Although, we implemented an uncomplicated version of the adaptive system we managed to produce positive results. Our adaptive system managed to control the results of selected response variables, significantly increasing the number of game sessions within target intervals by 5% - 12% points.

The methodology used throughout this thesis for identifying the effect sizes of factors for a number of different response variables provides an important intermediate and potential preprocessing step between the current existing commercial software and the goal of a truly adaptive game system. As a preprocessing step this methodology can be used to identify factors with little or no significant effect on the results.

These factors could be removed from inclusion in learning algorithms and ignored by level designers. Our prototype of the adaptive game system illustrates the potential as an intermediate step, as it allows game factors to be adapted in an online setting but only to a predefined interval of values. This provides the advantage that commercial games are searching for in pre-release testing, while allowing an adaptive system to tune to the player's preferences. The adjustment occurred dynamically during game play and if we increased the number of levels for each factor this method could account for a additional difficulty levels. In addition, the greater the number of factor levels the larger the number of situations the system could adapt too. The additional factor levels would come at an expense of memory usage, but would improve the granularity of modifications and allow a larger selection of players be to represented.

In addition our research is unique in creating a set of response variables for which adaptation could take place. The majority of research has focused on using a single response such as score to perform adaptation. Recent research has demonstrated that the future of adaptive game systems will require multiple sets of response variables to predict emotional states [32, 57] such as frustration, as such adaptive game systems will require the ability to adapt for these sets of response variables. Yannakakis and Hallam's research showed that emotional states can be predicted given a number of response variables, the combination of our research projects could provide a positive step toward adaptation for emotional states simply on the basis of response variables, and being able to control those response variables by altering game factors.

The obvious limitation of this methodology is based on the number of factors, as the cost of analysis grows exponentially for each additional factor. The limit of the commercial software was 7-8 factors for a full factorial analysis, however in cases where minimal interaction between factors is expected a fractional factorial analysis provides the ability to perform analysis for a higher number of factors. Our analysis was capable of utilizing properties of the full factorial analysis to go beyond the range provided by commercial software to 15 factors, although we utilized groups of 10 factors. The commercial limitation of the number of factors for the factorial analysis and the number of terms for the model in creating R-Sq values limited the scope of

our analysis. However these steps are not necessary for building this adaptive system, but may provide useful information for space requirements and the ease of adapting for response variables.

A potential limitation of calculating term effects is that players' abilities and skills are dynamic and players tend to become better at games as they play more. Thus, gradually throughout gameplay as the player's skill increases, the term effects size will drift from the original accurate results to inaccurate unless the player model adapts to reflect this player progression.

7.2 Future Work

This section discusses potential future work for the area of adaptive gaming systems. The discussion reviews the factorial analysis approach, as well as future work within the prototype of the adaptive game system.

7.2.1 Factorial Analysis

As previously mentioned the largest limitation of the factorial analysis methodology is the exponential growth of calculations that occurs by including additional factors. To effectively integrate the information obtained from the factorial analysis into the adaptive system, the process should identify and eliminate insignificant terms. Factors should only be removed from the analysis process once shown that they produce minimal statistical or game significant contributions to a spectrum of players, as they may be insignificant to some players while important to others.

The limitation of utilizing a factorial analysis for simple game parameters such as vision range would quickly become overwhelmed if implemented in a larger game setting. Areas, in which this analysis method could prove useful occur in specific small reoccurring scenarios of games, such as platform jumping or target aiming. In addition, this method could prove effective with higher level or generalized factors such as player aggression or stealth, instead of very low-level values like vision range. Generalized factors could encompass a larger set of factor or even be represented by

different algorithms.

Initially, during an exploratory stage of our research we tested multiple commercial software statistical packages. During, this exploratory stage we tested Minitab, SPSS and R statistical packages. Our testing, found Minitab to have an improved user-interface and was computationally faster for a factorial design, than SPSS. The R statistical program along with the factorial design package AlgDesign, showed to initially be unable to handle data sets of the magnitude of our experiment. Due this fact and the steeper learning curve of R, we ultimately selected Minitab as our statistical package for our experimental analysis. However, a recent commercial R library released in 2011 claims to be capable of handling statistical experiments of this size [4]. The Revolution R Enterprise is specifically designed for scalable data set, and provides a video demonstration of a linear regression and prediction on a 13GB and 120 million-row data set [4]. Further, experimentation of this Revolution R program may ease the methodology of this size of experiment, the practically of implementation and ease the presentation of the results.

7.2.2 Online User Observation

In our research, the player actions were simulated via a static algorithm, the tracking information is basically equivalent in an online setting. The process should be moved to an online setting to gain a larger spectrum of players and test cases. In addition, observing the gameplay of real players would allow players to be clustered based on attributes and response results. That information could be used to create generalized understanding of performance, as well as define player types for the game and possibly tracking progression from beginner to experienced players. Potentially, a player profile would be created for all players, performance could be logged and the information stored and analyzed offline. In an offline setting, a factorial analysis could be used to identify game significant factors, which could be stored and re-evaluated over the duration of their play. If this information was available, it could be utilized in conjunction with information from the player profile, such as reaction speed, gender or experience with the game genre. From this information of player preferences and

factor game significance, we could produce informative player models. These player models would have adequate information to adjust the game based on general player psychological preferences, but also based on an individual factor level.

The ultimate goal of an adaptive system is to produce a state of optimal immersion or Flow, by understanding the relationship between our factors effect on response variables and the relationship of those response variables to emotional states. In doing so, we can give developers the additional tools needed to promote the emotional reaction they want to achieve. Our research has focused on the within-subject design, meaning we investigate sections of the player's gameplay under a variety of situations. To help players achieve and retain a state of Flow during their game session, the adaptive system requires a method with knowledge of transitioning the player to new game elements and challanges. Future work with online players should also address between-subject design, as the transition of the player's emotional state and their own natural progression will play an important role in predicting future player models and how those models will adapt.

A potential approach to integrate our methodology with the transitional progression of the player is to consider all information as unique static player models within a database. Storing the player factor information at regular intervals provides an additional method of performing prediction, about how a player might transition. Each time we evaluate the player's state we explore player profiles that are similar to the current player either in player ability or effect on response variables. Clustering these player models allows a comparison between players with similar game significant terms and/or between the player's topology. All of this information progresses towards a system which is capable of predicting based on profile and game play. Player models could then predict factors to modify or reevaluate their accuracy based on the transitional path of other players in their cluster.

Online user observation is the first step toward integrating this methodology with research that maps response variables to emotional states of players and producing improved player models. Future work could integrate the research predicting emotional state from questionnaires [32, 57]. This progression could ideally produce an

improved understanding between emotional states and a set of game factors. Additionally, the combination of each methodology would allow greater control in creating and altering specific emotional responses. Identifying emotional state is and important area for future research as we need to be able validate whether our player models are contributing to produce the correct emotional states. The emotional state of the player should be integrated with the research of player typologies to identify any potential patterns from players during specific emotional states.

7.2.3 Adaptive Game System

In the development of the adaptive system prototype one of the key elements was the ability to assess the player skill level and accurately predict performance. Predicting the performance allowed the correct adjustment to be made to reach the targeted response goals. Our research used two methods to produce heuristics to predict performance; the first was hand crafted heuristics based on performance measures and the second used regression equations from the simulation information. These methods produced adequate results for our requirements, although future work should investigate this task by implementing machine learning algorithms. This problem is well suited to machine learning due to the high complexity of these functions, as well as the dynamic aspect of a player's natural experience progression.

The progression of adaptation is another area for potential work within the adaptive game system. We developed a method of identifying and quantifying a term's effect on a response variable, little research exists which investigates the rate of which adaptation should occur. The rate of adaptation becomes a predominant aspect in the adaptive system because it would limit the opportunities of mistakes from over-correcting the difficulty and would maintain the sense of believability for the player who may notice drastic alterations to the level of difficulty.

Our prototype adapted the game to settings previously viewed in the simulation phase of the experiment which is ideal for level designers and game testers, as dynamic aspect of the game can be tested but also provide a range of of challenge. Increasing the number of factor levels slightly increases the complexity of the analysis, but if the

factorial design included 3 levels it would allow a game designer to analyze factors in the traditional easy, medium and hard scenario. To develop a truly adaptive game system, our system would need to interpolate and extrapolate the data to include unobserved factor level values. Future research may consider performing multiple runs of the simulation phase with different factor levels to reduce the range of observed data which could potentially decrease errors while calculating unseen values.

7.2.4 Commercial Implementation

This research focused on producing results which could be applicable to both academic and commercial areas of gaming. Particularly, during the analysis and prototype phases we emphasized how our design could easily integrate into current commercial games. The methodology proposed in our research could be used several ways to improve commercial game development. First, the factorial analysis phase can be used to test new game mechanic and additionally to balance selected mechanics or levels. Additionally, the mechanics of factorial analysis create a within-subject snapshot of how the factors effect the player's performance on a set of response variables. The methodology of this research, can be used to test new features or balance the game but the game must be within a stable state, as parameters which implicitly alter the results must remain consistent for the results to be useful. The generalization, modularity and independence of the analysis system allows for quick integration for testing and easy removal before product release.

In terms of the commercial software pipeline our research methodology provides greater use toward the completion of the project as a larger number of game mechanics and design decisions become fixed. Portions of the adaptive game system should use machine learning techniques to adapt to the players. As previously mentioned, this methodology could be used to generate snap-shots of the player's performance at certain intervals; these snapshots are only effective if the algorithms used remain relatively consistent with their decision process throughout the analysis phase. In addition, the analysis for any machine learning algorithms used for the NPC behaviour should be analyzed after the learning phase and before setting the final behaviours

for release.

7.3 Conclusions

Our research investigated a methodology to identify game factors which altered the simulated player's performance on a set of response variables. The overall objective of our research is to contribute to the practical implementation of the AGS model by understanding how to adapt the game to particular individuals and by measuring the effectiveness of those adaptations. Understanding how modifications of the game affect the player's performance is the first step toward understanding how modifications of the game affect the player's emotional state and investment. Future research within this area will need to focus on the relationship between game factors and emotional states to engage and promote highly immersive states. Our research has demonstrated the ability to understand how each factor effects the player's performance. Our results indicated that a small consistent set of factors played prominent roles in altering the performance of the player for all response variables. Although, intuitively, a number of these factors could be predicted as being prominent factors it is important to rank these factors. In addition, the strength of this analysis is not only to identify important factors but also to demonstrate which factors have minimal impact on the game.

The results of the Pac-Man analysis indicates that the vision of the ghosts and Pac-Man along with the flee and death times played the most prominent roles. However, we also identified that the perceived value of the fruit played a minimal role for the SSS-AB* player, while the perceived value of the fruit was of greater importanace to PW algorithms based players. The analysis provided interesting results for the interaction of game factors and which algorithms produced emergent behaviour. This type of information can be difficult to discover during testing and can be valuable for game designers to understand high levels of unintended difficulty. The analysis also highlighted interesting properties relating to the perceived level of challenge; when the player engaged in bonus tasks it often resulted in a diminished overall performance

rather than additional points scored by the player. This type of information could help designers strategically place bonus items or tasks in areas with lower levels of interest rather than challenging areas.

As previously discussed, the effectiveness of this methodology is highly dependent on the ability to compute a large number of factors efficiently. The large number of games required for simulation helps with testing and identifying imbalances within the game. Commercially, this methodology is effective for testing new features and tuning final factor setting for release. It can be easily integrated into commercial products during the testing phase and due to the independence and modularity of this sub-system it can be easily removed before the product is release. In addition, the analysis phase is completely independent of the factors level values, so it can be used without modification for a wide variety of games. Due to the fact, that either NPC or level factors can be modified and observed with this analysis this methodology offers the ability to make design decisions based on overall game design issues or temporary issues such as opponent behaviour.

Finally, the strength of this analysis method is based on the implicitly defined aspect of the player model and the factor effects on each response variable. This methodology will improve researchers' ability to compare, cluster and generalize performance of different player types. This type of information will aid designers to drawing conclusions about the effects of their design decisions and predicting impact on different player types. The analysis method we experimented with during this thesis has shown to be successful toward understanding and adapting the game's factors. The trade-offs for this experiment have been the difficulty in working with large number of factors and the offline nature of the experiment. However, we feel the methodology has shown to be a positive progressive step toward improving and understanding game balancing and adaptive gaming. Adaptive gaming still has a large number of difficult issues to address for future research, however building on the results of our research we can now progress toward connecting the emotional state of the player to modifications of the game. This information brings us one step closer to the goal of creating targeted emotional responses from the modification of a few game

factors. Understanding and creating the precise intended emotional state of the player helps engage the player and allows the game to meet the emotional requirements of the player which is vital to creating the optimal state of Flow.

Appendix A

Factor Effects

A.1 SSS_FLOCK Factor Effects

This section provides a detailed view of a maximum of 127 terms included in each SSS_FLOCK response variable model. The first table presents the size of the confidence intervals for each response variable. If the absolute value of the regression coefficient is below these value it was not statistically significant. The models used a 95% confidence interval or $\alpha = 0.05$ to test statistically significant values. Each section provides two tables, the first table presents the main effects and their effect size. The second table illustrates the effect sizes of the remaining terms that were included in the model.

Response	Statistical Significant Effect Size
Score	21.33
Steps	3.38
Levels Complete	0.01
Close Calls	2.08
Fruit Created	0.06
Fruit Eaten	0.02
Ghosts Eaten	0.04
Power-Pellets Collected	0.05
Tokens Collected	1.90769
Repeated Squares	1.51895

Table A.1: SSS_FLOCK Confidence Intervals value

A.1.1 SSS_FLOCK Close Calls Model Terms

Main Effects	Effect Size	Interactions	Effect Size
FLEE_TIME (A)	-18.71	A*E*F*I	14.20
DEATH_TIME (B)	3.00	G*J	14.03
FRUIT_TIME (C)	-2.07	A*D*E*F	-12.24
FLOCK_SEP (D)	-10.03	B*D*E*F*I	-11.98
FLOCK_ALI (E)	-9.20	D*G*J	11.97
FLOCK_COH (F)	-9.20	E*J	-11.84
FLOCK_HUNGER (G)	36.27	F*J	-11.84
SSS_PERCEIVED_FRUIT (H)	0.99	A*B*I	-11.67
PAC_VIS_LEN (I)	7.01	B*I	11.53
GH_VIS_LEN (J)	-20.36	G*I*J	11.15
		B*E*G	-10.71
		B*F*G	-10.71
		A*E*F*I*J	10.64
		D*E*F*I	-10.58
		B*G	-10.27
		A*B*I*J	10.18
		A*B*D*J	-9.73
		E*F*G*J	8.93
		B*D*G*I*J	-8.78
		E*G	-8.63
		F*G	-8.63

Table A.2: The main effects and a portion of the largest effect sizes included in SSS_Flock model for the response variable Close Calls.

A.1.2 SSS_FLOCK Fruits Eaten Model Terms

Main Effects	Effect Size	Interactions	Effect Size
FLEE_TIME (A)	0.01	D*E*F*G*I*J	-0.11
DEATH_TIME (B)	0.06	A*E*F*G*I*J	0.10
FRUIT_TIME (C)	0.25	A*B*J	0.10
FLOCK_SEP (D)	0.08	A*E*F*J	0.10
FLOCK_ALI (E)	0.02	I*J	-0.10
FLOCK_COH (F)	0.02	A*G*J	0.08
FLOCK_HUNGER (G)	0.00	D*E*F	-0.08
SSS_PERCEIVED_FRUIT (H)	0.08	D*E*F*G	0.08
PAC_VIS_LEN (I)	0.08	B*D*E*F*I	-0.08
GH_VIS_LEN (J)	-0.18	A*D*E*F*G*I*J	0.08
		A*G*I*J	-0.07
		A*B*E*F*G	-0.07
		B*E*F*G*I*J	-0.07
		A*D	-0.07
		B*E*G*J	-0.07
		B*F*G*J	-0.07
		A*B*D*E*F*J	-0.06
		E*F*J	0.06
		D*E*I*J	-0.06
		D*F*I*J	-0.06
		A*D*I*J	0.06
		E*F	-0.06

Table A.3: The main effects and a portion of the largest effect sizes included in SSS_Flock model for the response variable Fruit Eaten.

A.1.3 SSS_FLOCK Ghosts Eaten Model Terms

Main Effects	Effect Size	Interactions	Effect Size
FLEE_TIME (A)	0.73	D*G*J	0.44
DEATH_TIME (B)	-0.05	A*E*F	0.26
FRUIT_TIME (C)	0.03	A*E*F*I	0.25
FLOCK_SEP (D)	0.23	B*G*I	0.23
FLOCK_ALI (E)	-0.13	B*D*G*J	-0.21
FLOCK_COH (F)	-0.13	E*F*J	0.21
FLOCK_HUNGER (G)	-0.05	D*E*F*G*J	-0.20
SSS_PERCEIVED_FRUIT (H)	0.10	A*D*G*I*J	-0.19
PAC_VIS_LEN (I)	-0.33	D*I	0.19
GH_VIS_LEN (J)	-0.18	B*E*F*I	-0.18
		D*E*F*I	-0.18
		B*D*G	-0.18
		B*E*G*I	0.18
		B*F*G*I	0.18
		A*D*E*F*G*I	0.18
		D*E*F*I*J	-0.17
		A*B*D*E*F*G*J	0.17
		A*E*I	0.17
		A*F*I	0.17
		D*E*F*J	0.17
		B*I	0.16
		B*E*F*G*J	0.16
		D*G	0.16

Table A.4: The main effects and a portion of the largest effect sizes included in SSS_Flock model for the response variable Ghosts Eaten.

A.1.4 SSS_FLOCK Levels Completed Model Terms

Main Effects	Effect Size	Interactions	Effect Size
FLEE_TIME (A)	0.07	A*E*F*I	0.10
DEATH_TIME (B)	0.05	E*F	-0.08
FRUIT_TIME (C)	-0.01	B*D*E*F*I*J	-0.08
FLOCK_SEP (D)	0.02	E*F*G*I	-0.07
FLOCK_ALI (E)	0.02	A*D*G*I*J	-0.07
FLOCK_COH (F)	0.02	A*D*E*F*G*I	0.07
FLOCK_HUNGER (G)	0.06	D*I	0.07
SSS_PERCEIVED_FRUIT (H)	0.00	B*D*I*J	0.07
PAC_VIS_LEN (I)	0.15	E*J	-0.06
GH_VIS_LEN (J)	-0.19	F*J	-0.06
		D*E*F*I	-0.06
		I*J	-0.06
		A*E*G*J	-0.06
		A*F*G*J	-0.06
		B*D*E*F*I	-0.06
		D*I*J	0.06
		A*E*G	-0.06
		A*F*G	-0.06
		B*D*E*F	-0.05
		B*D*E	-0.05
		B*D*F	-0.05
		D*G*I	0.05

Table A.5: The main effects and a portion of the largest effect sizes included in SSS_Flock model for the response variable Levels Completed.

A.1.5 SSS_FLOCK Power-Pellets Collected Model Terms

Main Effects	Effect Size	Interactions	Effect Size
FLEE_TIME (A)	0.36	D*E*F*I	-0.33
DEATH_TIME (B)	0.20	E*F*G*I	-0.32
FRUIT_TIME (C)	0.00	A*E*F*I	0.32
FLOCK_SEP (D)	0.03	D*I	0.31
FLOCK_ALI (E)	0.04	D*G*I*J	0.31
FLOCK_COH (F)	0.04	I*J	-0.30
FLOCK_HUNGER (G)	0.14	E*F	-0.29
SSS_PERCEIVED_FRUIT (H)	-0.01	D*E*F*J	0.29
PAC_VIS_LEN (I)	0.65	A*D*G*I*J	-0.26
GH_VIS_LEN (J)	-0.67	A*E*F*I*J	0.25
		B*D*E*F*I*J	-0.24
		B*E*F*G*J	0.24
		E*J	-0.24
		F*J	-0.24
		A*E*F*J	0.24
		E*F*G*J	0.21
		A*D*G	-0.21
		D*G*J	0.21
		A*B*D*E*F*G	0.20
		A*D*I*J	0.20
		G*I*J	0.20
		B*D*E	-0.20
		B*D*F	-0.20
		D*G*I	0.20

Table A.6: The main effects and a portion of the largest effect sizes included in SSS_Flock model for the response variable Power-Pellets Collected.

A.1.6 SSS_FLOCK Repeated Squares Model Terms

Main Effects	Effect Size	Interactions	Effect Size
FLEE_TIME (A)	7.52	D*G*J	12.46
DEATH_TIME (B)	4.55	B*D*E*F*I*J	-11.03
FRUIT_TIME (C)	-0.14	A*E*F*I	10.57
FLOCK_SEP (D)	3.20	I*J	-9.73
FLOCK_ALI (E)	0.16	D*G*I*J	9.41
FLOCK_COH (F)	0.16	B*D*I*J	9.00
FLOCK_HUNGER (G)	7.35	B*J	-8.88
SSS_PERCEIVED_FRUIT (H)	-0.78	E*J	-7.94
PAC_VIS_LEN (I)	11.13	F*J	-7.94
GH_VIS_LEN (J)	-20.17	A*E*F*J	7.84
		A*D*E*F*G*J	7.78
		A*D*E*F	-7.46
		D*E*F*I	-7.35
		A*B	6.84
		A*B*D*E*F	-6.61
		G*J	6.40
		A*B*D*J	-6.07
		A*D*G*I*J	-6.00
		A*B*I	-5.73
		B*D*E*F	-5.57
		D*E*F*J	5.48
		D*I	5.44

Table A.7: The main effects and a portion of the largest effect sizes included in SSS_Flock model for the response variable Repeated Squares.

A.1.7 SSS_FLOCK Score Model Terms

Main Effects	Effect Size	Interactions	Effect Size
FLEE_TIME (A)	194.39	A*E*F*I	144.63
DEATH_TIME (B)	79.46	A*D*G*I*J	-126.66
FRUIT_TIME (C)	30.52	D*I	119.98
FLOCK_SEP (D)	52.56	D*G*J	118.58
FLOCK_ALI (E)	12.05	E*F	-116.43
FLOCK_COH (F)	12.05	D*E*F*I	-116.38
FLOCK_HUNGER (G)	69.31	E*F*G*I	-116.08
SSS_PERCEIVED_FRUIT (H)	0.86	B*D*E*F*I*J	-108.71
PAC_VIS_LEN (I)	242.68	I*J	-107.21
GH_VIS_LEN (J)	-304.10	D*G*I*J	98.84
		D*E*F*J	96.39
		E*J	-93.41
		F*J	-93.41
		A*E*G	-93.27
		A*F*G	-93.27
		A*E*G*J	-91.46
		A*F*G*J	-91.46
		A*D*E*F*G*I	91.24
		A*E*F*J	89.27
		B*D*E	-87.52
		B*D*F	-87.52
		B*E*F*G*J	86.06

Table A.8: The main effects and a portion of the largest effect sizes included in SSS_Flock model for the response variable Score.

A.1.8 SSS_FLOCK Steps Model Terms

Main Effects	Effect Size	Interactions	Effect Size
FLEE_TIME (A)	22.87	A*E*F*I	23.08
DEATH_TIME (B)	13.01	D*G*J	22.17
FRUIT_TIME (C)	-0.46	B*D*E*F*I*J	-21.79
FLOCK_SEP (D)	4.15	I*J	-20.86
FLOCK_ALI (E)	2.64	D*G*I*J	20.26
FLOCK_COH (F)	2.64	D*E*F*I	-18.63
FLOCK_HUNGER (G)	15.65	E*J	-17.86
SSS_PERCEIVED_FRUIT (H)	-1.49	F*J	-17.86
PAC_VIS_LEN (I)	35.38	A*D*G*I*J	-17.81
GH_VIS_LEN (J)	-48.58	D*I	17.56
		A*E*F*J	17.43
		B*J	-16.78
		B*D*I*J	16.74
		E*F*G*I	-15.97
		D*E*F*J	15.43
		E*F	-14.40
		A*B*D*E*F	-14.23
		G*I*J	13.72
		B*D*E*F	-13.30
		B*D*E	-13.22
		B*D*F	-13.22
		A*B*D*E*F*G	13.04

Table A.9: The main effects and a portion of the largest effect sizes included in SSS_Flock model for the response variable Steps.

A.1.9 SSS_FLOCK Tokens Collected Model Terms

Main Effects	Effect Size	Interactions	Effect Size
FLEE_TIME (A)	14.99	A*E*F*I	12.19
DEATH_TIME (B)	8.26	D*I	11.81
FRUIT_TIME (C)	-0.32	E*F*G*I	-11.66
FLOCK_SEP (D)	0.92	A*D*G*I*J	-11.56
FLOCK_ALI (E)	2.43	D*E*F*I	-10.95
FLOCK_COH (F)	2.43	I*J	-10.83
FLOCK_HUNGER (G)	8.15	D*G*I*J	10.55
SSS_PERCEIVED_FRUIT (H)	-0.69	B*D*E*F*I*J	-10.52
PAC_VIS_LEN (I)	23.61	E*F	-10.20
GH_VIS_LEN (J)	-27.75	E*J	-9.67
		F*J	-9.67
		D*E*F*J	9.66
		D*G*J	9.50
		A*E*F*J	9.36
		G*I*J	8.20
		A*B*D*E*F*G	8.04
		A*E*F*I*J	8.01
		B*D*E	-7.85
		B*D*F	-7.85
		A*E*G*J	-7.75
		A*F*G*J	-7.75
		B*J	-7.72

Table A.10: The main effects and a portion of the largest effect sizes included in SSS_Flock model for the response variable Tokens collected.

A.2 SSS_GW Factor Effects

This section provides a detailed view of the statistically significant terms included in each SSS_FLOCK response variable model. The first table, Table A.11 presents the size of the confidence intervals for each response variable. If the absolute value of the regression coefficient is below these value it was not statistically significant. The models used a 95% confidence interval or $\alpha = 0.05$ to test statistically significant values. Each section provides a table that presents the statically significant main effects and 2-factor interactions and their effect sizes.

Response	Statistical Significant Effect Size
Score	5.21
Steps	0.79
Levels Complete	0.003
Close Calls	0.87
Fruit Created	0.01
Fruit Eaten	0.004
Ghosts Eaten	0.01
Power-Pellets Collected	0.01
Tokens Collected	0.42
Repeated Squares	0.40

Table A.11: SSS_GW Confidence Intervals value

A.2.1 SSS_GW Close Calls Model Terms

Main Effects	Effect Size	2-Factor Interactions	Effect Size
GW_TOKEN (A)	*	A*D	1.81
GW_PP (B)	*	A*F	1.17
GW_FT (C)	-1.33	A*G	1.41
GW_PAC (D)	1.86	B*D	1.21
GW_PAC_DIR (E)	4.99	B*G	-2.29
GW_AWAY_GH (F)	-13.74	B*H	1.86
GW_TO_GH (G)	-3.76	B*I	3.89
FLEE_TIME (H)	-6.04	B*J	-3.27
DEATH_TIME (I)	4.23	C*D	-1.06
GH_VIS_LEN (J)	20.96	C*I	-1.47
		D*E	-0.94
		E*J	-2.30
		F*I	-2.50
		F*J	-5.10
		G*H	-1.88
		G*I	-1.84
		H*I	3.80
		I*J	4.04

Table A.12: The statistically significant Main Effects for SSS_GW Close Calls. Effects marked with * are not statistically significant but are included because their term interacts in a statistically significant way with another factor.

A.2.2 SSS_GW Fruits Eaten Model Terms

Main Effects	Effect Size	2-Factor Interactions	Effect Size
GW_TOKEN (A)	*	A*H	0.00
GW_PP (B)	*	B*E	0.00
GW_FT (C)	*	B*F	0.00
GW_PAC (D)	*	B*G	-0.01
GW_PAC_DIR (E)	0.00	B*H	-0.01
GW_AWAY_GH (F)	-0.01	C*D	0.00
GW_TO_GH (G)	-0.01	C*G	0.00
FLEE_TIME (H)	0.01	D*G	0.00
DEATH_TIME (I)	0.01	E*J	0.00
GH_VIS_LEN (J)	-0.05	F*G	0.01
		F*H	0.00
		F*I	-0.01
		G*H	-0.01
		I*J	-0.01

Table A.13: The statistically significant Main Effects for SSS_GW Fruit Eaten. Effects marked with * are not statistically significant but are included because their term interacts in a statistically significant way with another factor.

A.2.3 SSS_GW Ghosts Eaten Model Terms

Main Effects	Effect Size	2-Factor Interactions	Effect Size
GW_TOKEN (A)	-0.04	B*E	0.02
GW_PP (B)	-0.02	B*F	-0.08
GW_FT (C)	*	B*G	-0.09
GW_PAC (D)	-0.11	B*H	-0.03
GW_PAC_DIR (E)	-0.08	B*I	0.04
GW_AWAY_GH (F)	-0.31	B*J	0.03
GW_TO_GH (G)	-0.12	D*E	0.02
FLEE_TIME (H)	0.18	D*F	-0.01
DEATH_TIME (I)	-0.03	D*G	0.05
GH_VIS_LEN (J)	0.58	D*H	0.02
		D*I	-0.01
		D*J	-0.05
		E*G	0.02
		E*H	-0.03
		E*J	-0.01
		F*G	-0.06
		F*H	-0.05
		F*I	0.05
		F*J	-0.16
		G*H	-0.02
		G*I	-0.02
		H*J	-0.17
		I*J	-0.05

Table A.14: The statistically significant Main Effects for SSS_GW Ghosts Eaten. Effects marked with * are not statistically significant but are included because their term interacts in a statistically significant way with another factor.

A.2.4 SSS_GW Levels Completed Model Terms

Main Effects	Effect Size	2-Factor Interactions	Effect Size
GW_TOKEN (A)	0.008	A*B	0.007
GW_PP (B)	0.01	A*D	0.005
GW_FT (C)	*	A*F	0.004
GW_PAC (D)	0.004	A*G	0.003
GW_PAC_DIR (E)	*	B*D	0.006
GW_AWAY_GH (F)	-0.03	B*F	0.006
GW_TO_GH (G)	-0.01	B*G	-0.005
FLEE_TIME (H)	0.02	B*I	0.004
DEATH_TIME (I)	0.02	B*J	-0.011
GH_VIS_LEN (J)	-0.04	C*D	-0.005
		C*F	-0.004
		C*G	-0.004
		C*I	-0.004
		D*G	0.004
		D*H	0.005
		D*I	0.004
		D*J	0.003
		E*J	-0.007
		F*I	-0.005
		F*J	-0.004
		G*H	-0.006
		G*I	-0.013
		G*J	0.007
		H*I	0.012

Table A.15: The statistically significant Main Effects for SSS_GW Levels Completed. Effects marked with * are not statistically significant but are included because their term interacts in a statistically significant way with another factor.

A.2.5 SSS_GW Power-Pellets Collected Model Terms

Main Effects	Effect Size	2-Factor Interactions	Effect Size
GW_TOKEN (A)	-0.02	A*B	0.02
GW_PP (B)	0.03	A*D	0.01
GW_FT (C)	*	A*F	0.02
GW_PAC (D)	*	A*G	0.01
GW_PAC_DIR (E)	0.01	A*H	-0.01
GW_AWAY_GH (F)	-0.13	B*D	0.01
GW_TO_GH (G)	-0.05	B*G	-0.02
FLEE_TIME (H)	0.11	B*I	0.02
DEATH_TIME (I)	0.10	B*J	-0.06
GH_VIS_LEN (J)	-0.18	C*D	-0.01
		C*G	-0.01
		C*I	-0.01
		D*G	0.01
		E*I	-0.01
		E*J	-0.02
		F*J	-0.01
		G*H	-0.02
		G*I	-0.04
		G*J	0.02
		H*I	0.01
		I*J	0.03

Table A.16: The statistically significant Main Effects for SSS_GW Power-Pellets Collected. Effects marked with * are not statistically significant but are included because their term interacts in a statistically significant way with another factor.

A.2.6 SSS_GW Repeated Squares Model Terms

Main Effects	Effect Size	2-Factor Interactions	Effect Size
GW_TOKEN (A)	*	A*B	0.72
GW_PP (B)	1.85	A*D	0.72
GW_FT (C)	-0.60	A*G	0.87
GW_PAC (D)	*	A*J	-0.50
GW_PAC_DIR (E)	0.41	B*D	0.69
GW_AWAY_GH (F)	-3.13	B*G	-0.94
GW_TO_GH (G)	-1.14	B*I	1.16
FLEE_TIME (H)	3.58	B*J	-2.90
DEATH_TIME (I)	3.49	C*D	-0.48
GH_VIS_LEN (J)	-9.43	C*I	-0.77
		D*G	0.90
		E*J	-1.21
		F*I	-0.56
		F*J	-1.05
		G*H	-1.09
		G*I	-1.06
		H*I	1.58

Table A.17: The statistically significant Main Effects for SSS_GW Repeated Squares. Effects marked with * are not statistically significant but are included because their term interacts in a statistically significant way with another factor.

A.2.7 SSS_GW Score Model Terms

Main Effects	Effect Size	2-Factor Interactions	Effect Size
GW_TOKEN (A)	-12.84	A*B	8.09
GW_PP (B)	13.47	A*D	7.32
GW_FT (C)	*	A*F	5.91
GW_PAC (D)	-7.67	A*G	5.80
GW_PAC_DIR (E)	-6.59	B*D	6.70
GW_AWAY_GH (F)	-78.11	B*G	-19.02
GW_TO_GH (G)	-29.70	B*H	-7.93
FLEE_TIME (H)	55.57	B*I	12.48
DEATH_TIME (I)	31.55	B*J	-17.74
GH_VIS_LEN (J)	-20.78	C*D	-7.16
		C*G	-5.48
		C*I	-7.81
		D*G	12.92
		D*H	7.31
		E*J	-13.36
		F*H	-8.87
		F*J	-20.35
		G*H	-15.00
		G*I	-21.15
		G*J	8.84
		H*I	15.80
		H*J	-16.42

Table A.18: The statistically significant Main Effects for SSS_GW Score. Effects marked with * are not statistically significant but are included because their term interacts in a statistically significant way with another factor.

A.2.8 SSS_GW Steps Model Terms

Main Effects	Effect Size	2-Factor Interactions	Effect Size
GW_TOKEN (A)	*	A*B	1.51
GW_PP (B)	3.86	A*D	1.43
GW_FT (C)	-0.82	A*F	0.95
GW_PAC (D)	*	A*G	1.48
GW_PAC_DIR (E)	*	B*D	1.22
GW_AWAY_GH (F)	-8.24	B*G	-1.74
GW_TO_GH (G)	-2.65	B*I	2.21
FLEE_TIME (H)	8.22	B*J	-5.87
DEATH_TIME (I)	7.59	C*D	-1.01
GH_VIS_LEN (J)	-18.98	C*I	-1.57
		D*G	1.69
		E*I	-0.87
		E*J	-2.40
		F*I	-0.84
		F*J	-1.46
		G*H	-2.28
		G*I	-2.84
		G*J	0.86
		H*I	3.07
		I*J	1.10

Table A.19: The statistically significant Main Effects for SSS_GW Steps. Effects marked with * are not statistically significant but are included because their term interacts in a statistically significant way with another factor.

A.2.9 SSS_GW Tokens Collected Model Terms

Main Effects	Effect Size	2-Factor Interactions	Effect Size
GW_TOKEN (A)	-0.84	A*B	0.76
GW_PP (B)	1.98	A*D	0.69
GW_FT (C)	*	A*F	0.65
GW_PAC (D)	*	A*G	0.59
GW_PAC_DIR (E)	*	B*D	0.50
GW_AWAY_GH (F)	-4.98	B*F	0.44
GW_TO_GH (G)	-1.46	B*G	-0.76
FLEE_TIME (H)	4.52	B*H	-0.43
DEATH_TIME (I)	4.00	B*I	1.02
GH_VIS_LEN (J)	-9.36	B*J	-2.90
		C*D	-0.51
		C*G	-0.44
		C*I	-0.78
		D*G	0.77
		E*I	-0.49
		E*J	-1.16
		G*H	-1.16
		G*I	-1.73
		G*J	1.09
		H*I	1.47
		I*J	0.72

Table A.20: The statistically significant Main Effects for SSS_GW Tokens Collected. Effects marked with * are not statistically significant but are included because their term interacts in a statistically significant way with another factor.

A.3 PW_FLOCK Factor Effects

This section illustrates the values calculated for the confidence intervals for the PW_FLOCK models. The models used a 95% confidence interval or $\alpha = 0.05$ to test statistically significant values.

Response	Statistical Significant Effect Size
Score	1.69
Steps	0.39
Levels Complete	0.001
Close Calls	0.29
Fruit Created	0.006
Fruit Eaten	0.0017
Ghosts Eaten	0.003
Power-Pellets Collected	0.004
Tokens Collected	0.15
Repeated Squares	0.28

Table A.21: PW_FLOCK Confidence Intervals value

A.4 PW_GW Factor Effects

This section illustrates the values calculated for the confidence intervals for the PW_FLOCK models. The models used a 95% confidence interval or $\alpha = 0.05$ to test statistically significant values.

Response	Statistical Significant Effect Size
Score	0.41
Steps	0.14
Levels Complete	0.0002
Close Calls	0.125
Fruit Created	0.0023
Fruit Eaten	0.0006
Ghosts Eaten	0.0014
Power-Pellets Collected	0.001
Tokens Collected	0.03
Repeated Squares	0.12

Table A.22: PW_GW Response variables Confidence Intervals values.

Appendix B

Model Error

This Appendix section presents the results of the lack-of-fit test and the residual and pure error terms for the each model. Each table presents the results for all separated game sessions for a single response variable. Information presented in tables are displayed in the following format: the GameID, the separated factor, the sum of squares Lack of Fit score (SSLOF), the sum of squares Residual Error(SSRE), the sum of squares Pure Error(SSPE), the f-value and the p-value.

B.1 SSS_FLOCK Model Error

The separated factor for the SSS_FLOCK algorithm is the frequency of the fruit creation (FRUIT_FREQ). The GameID indicates whether a factor used a high or low level for the experimental run. In this case GameID 0 indicates low level value was used while GameID 1 indicates a high level value was used in the experiment.

GameID	SSLOF	SSRE	SSPE	F-Value	P-Value
0	1814721	21515666	19700944	0.21	1
1	2600082	23060107	20460026	0.29	1

Table B.1: Model Error and Lack of Fit Results for algorithm SSS_FLOCK and response variable Close Calls (CC).

GameID	SSLOF	SSRE	SSPE	F-Value	P-Value
0	113.77	792.44	678.66	0.38	1
1	276.26	2336.93	2060.66	0.31	1

Table B.2: Model Error and Lack of Fit Results for algorithm SSS_FLOCK and response variable the number of fruit eaten (FE).

GameID	SSLOF	SSRE	SSPE	F-Value	P-Value
0	613.80	9348.46	8734.67	0.16	1
1	848.40	8622.40	7774.00	0.25	1

Table B.3: Model Error and Lack of Fit Results for algorithm SSS_FLOCK and response variable the number of fruit eaten (FE).

GameID	SSLOF	SSRE	SSPE	F-Value	P-Value
0	78.86	912.86	834	0.22	1
1	106.74	1006.75	900	0.27	1

Table B.4: Model Error and Lack of Fit Results for algorithm SSS_FLOCK and response variable the number of levels completed (LVL_COMP).

GameID	SSLOF	SSRE	SSPE	F-Value	P-Value
0	1033.73	12496.4	11462.67	0.21	1
1	1392.92	12970.3	11577.33	0.28	1

Table B.5: Model Error and Lack of Fit Results for algorithm SSS_FLOCK and response variable the number of power-pellets eaten(PP).

GameID	SSLOF	SSRE	SSPE	F-Value	P-Value
0	988545.74	11551580	10563034.62	0.21	1
1	1143948.26	12037831	10893883.06	0.24	1

Table B.6: Model Error and Lack of Fit Results for algorithm SSS_FLOCK and response variable the number of repeated squares (RSQ).

GameID	SSLOF	SSRE	SSPE	F-Value	P-Value
0	177835365.59	2162864245	1985028878.92	0.20	1
1	255245093.24	2504227491	2248982397.75	0.26	1

Table B.7: Model Error and Lack of Fit Results for algorithm SSS_FLOCK and response variable the player's score (Sc).

GameID	SSLOF	SSRE	SSPE	F-Value	P-Value
0	4681461.58	57523688	52842226.24	0.20	1
1	5838768.61	59039345	53200576.62	0.25	1

Table B.8: Model Error and Lack of Fit Results for algorithm SSS_FLOCK and response variable the player's score (St).

GameID	SSLOF	SSRE	SSPE	F-Value	P-Value
0	1512354.35	18324393	16812039.03	0.21	1
1	1936172.67	18969312	17033139.05	0.26	1

Table B.9: Model Error and Lack of Fit Results for algorithm SSS_FLOCK and response variable the number of tokens collected (TO).

B.2 SSS_GW Model Error

The four separated factors for the SSS_GW algorithm are: the perceived value of the fruit (SSS_PERCEIVED_FRUIT), the frequency of the fruit creation (FRUIT_FREQ), the time the fruit was available on screen (FRUIT_TIME) and finally the range of Pac-Man's vision (PAC_VIS_LEN). The GameID indicates whether a factor used a high or low level for the experimental run. As an example, GameID 5 would convert to 0101 in binary, indicating that factors 1 and 3 used low values represented by 0, while 2 and 4 used high values indicated by a value of 1.

GameID	SSLOF	SSRE	SSPE	F-Value	P-Value
0	5139897.74	32809474	27669575.97	0.42	1
1	5162482.66	31911121	26748638.75	0.44	1
2	4496450.89	33691388	29194936.62	0.35	1
3	4435268.54	33034938	28599669.076	0.35	1
4	7074655.49	41155052	34080396.096	0.47	1
5	6808527.68	39557849	32749321.58	0.48	1
6	6721028.18	39229836	32508807.43	0.47	1
7	6111007.23	37505412	31394404.65	0.44	1
8	3458452.86	29517950	26059497.01	0.30	1
9	3561765.86	30233351	26671585.62	0.30	1
10	4137606.82	28834309	24696702.26	0.38	1
11	4315342.06	29219810	24904468.14	0.40	1
12	4963847.08	35642828	30678981.31	0.37	1
13	4868573.30	34937354	30068780.7450	0.37	1
14	4966175.46	30723939	25757763.9084	0.44	1
15	4814670.55	30020742	25206071.1000	0.44	1

Table B.10: Model Error and Lack of Fit Results for algorithm SSS_GW and response variable Close Calls (CC).

GameID	SSLOF	SSRE	SSPE	F-Value	P-Value
0	36.14	239.48	203.33	0.41	1
1	41.46	276.13	234.67	0.40	1
2	70.56	461.23	390.67	0.41	1
3	81.43	546.09	464.67	0.40	1
4	171.09	921.75	750.67	0.52	1
5	194.04	1022.04	828.00	0.54	1
6	261.59	1416.26	1154.67	0.52	1
7	305.87	1652.53	1346.67	0.52	1
8	40.64	303.31	262.67	0.35	1
9	44.44	355.78	311.33	0.33	1
10	69.44	498.10	428.67	0.37	1
11	79.79	550.45	470.67	0.39	1
12	154.75	926.08	771.34	0.46	1
13	165.41	1028.74	863.34	0.44	1
14	285.30	1519.96	1234.67	0.53	1
15	311.55	1729.55	1418.00	0.50	1

Table B.11: Model Error and Lack of Fit Results for algorithm SSS_GW and response variable number of fruit eaten (FE).

GameID	SSLOF	SSRE	SSPE	F-Value	P-Value
0	1302.14	11066.1	9764.00	0.30	1
1	1275.41	10828.8	9553.33	0.30	1
2	1453.06	11315.1	9861.99	0.33	1
3	1396.52	11529.9	10133.33	0.31	1
4	2004.42	10579.8	8575.33	0.53	1
5	2005.82	10418.5	8412.66	0.54	1
6	1974.15	12034.2	10059.99	0.44	1
7	1979.19	11793.2	9813.99	0.46	1
8	1573.26	11475.9	9902.66	0.36	1
9	1516.32	11359.0	9842.66	0.35	1
10	1677.90	11603.2	9925.33	0.38	1
11	1754.02	11520.7	9766.66	0.41	1
12	2115.69	14845.0	12729.33	0.37	1
13	1909.46	14431.5	12522.00	0.34	1
14	1833.63	12889.0	11055.33	0.37	1
15	1864.43	12658.4	10793.99	0.39	1

Table B.12: Model Error and Lack of Fit Results for algorithm SSS_GW and response variable number of ghosts eaten (GH).

GameID	SSLOF	SSRE	SSPE	F-Value	P-Value
0	64.77	422.10	357.33	0.41	1
1	69.24	426.58	357.33	0.44	1
2	80.70	477.36	396.66	0.46	1
3	75.70	457.70	382.00	0.45	1
4	86.72	488.06	401.33	0.49	1
5	85.51	479.51	394.00	0.49	1
6	83.86	467.19	383.33	0.50	1
7	75.80	437.81	362.00	0.47	1
8	60.35	431.02	370.66	0.37	1
9	61.04	439.70	378.66	0.36	1
10	72.72	472.72	400.00	0.41	1
11	78.37	474.37	396.00	0.45	1
12	81.38	493.38	412.00	0.45	1
13	75.01	494.34	419.33	0.40	1
14	77.91	445.24	367.33	0.48	1
15	75.51	414.18	338.66	0.50	1

Table B.13: Model Error and Lack of Fit Results for algorithm SSS_GW and response variable levels completed (LVL_COMP).

GameID	SSLOF	SSRE	SSPE	F-Value	P-Value
0	755.86	5354.53	4598.66	0.37	1
1	787.90	5345.90	4558.00	0.39	1
2	925.74	5833.08	4907.33	0.43	1
3	835.53	5658.21	4822.66	0.39	1
4	1053.99	5863.33	4809.33	0.50	1
5	1023.27	5726.61	4703.33	0.49	1
6	970.77	5634.78	4664.00	0.47	1
7	906.30	5413.64	4507.33	0.45	1
8	686.75	5474.76	4788.00	0.32	1
9	689.76	5545.10	4855.33	0.32	1
10	864.76	5663.43	4798.66	0.41	1
11	890.28	5606.28	4716.00	0.43	1
12	988.40	6665.07	5676.66	0.39	1
13	945.83	6505.84	5560.00	0.38	1
14	870.73	5454.74	4584.00	0.43	1
15	838.98	5136.98	4297.99	0.44	1

Table B.14: Model Error and Lack of Fit Results for algorithm SSS_GW and response variable number of power-pellets collected (PP).

GameID	SSLOF	SSRE	SSPE	F-Value	P-Value
0	887877.89	7173890	6286011.70	0.32	1
1	896728.94	7142870	6246141.33	0.32	1
2	1079141.10	7692743	6613601.63	0.37	1
3	1091616.22	7655808	6564191.86	0.38	1
4	1386154.47	8261387	6875232.53	0.46	1
5	1365221.92	7960347	6595125.03	0.47	1
6	1433994.04	8230351	6796356.50	0.48	1
7	1392054.81	7904170	6512115.36	0.48	1
8	792089.17	6323164	5531074.81	0.32	1
9	827151.28	6465292	5638140.81	0.33	1
10	979103.29	6530968	5551864.91	0.40	1
11	990976.13	6370626	5379649.96	0.42	1
12	1056547.23	7386059	6329511.41	0.38	1
13	1040837.32	7200228	6159390.54	0.38	1
14	1129797.81	6825617	5695819.66	0.45	1
15	1104286.99	6602094	5497806.76	0.45	1

Table B.15: Model Error and Lack of Fit Results for algorithm SSS_GW and response variable the number of repeated squares (RSQ).

GameID	SSLOF	SSRE	SSPE	F-Value	P-Value
0	151728361.89	1046972253	895243891.26	0.38	1
1	158747180.06	1048942737	890195557.04	0.40	1
2	190373290.05	1171455663	981082372.51	0.44	1
3	170163363.10	1151157454	980994091.38	0.39	1
4	216552776.05	1192252185	975699409.16	0.50	1
5	212869299.04	1176690044	963820745.10	0.50	1
6	222620836.49	1262139005	1039518168.38	0.48	1
7	211677133.33	1229109899	1017432765.94	0.47	1
8	142003417.64	1091429582	949426163.87	0.34	1
9	146302461.76	1114516309	968213847.28	0.34	1
10	169387405.45	1143582873	974195467.46	0.39	1
11	182334550.63	1137611884	955277332.99	0.43	1
12	218062860.42	1474067813	1256004952.07	0.39	1
13	204951291.04	1447334895	1242383604.08	0.37	1
14	208192327.80	1286210993	1078018664.80	0.44	1
15	195944111.45	1215233074	1019288962.14	0.43	1

Table B.16: Model Error and Lack of Fit Results for algorithm SSS_GW and response variable player's score (Sc).

GameID	SSLOF	SSRE	SSPE	F-Value	P-Value
0	3502776.90	26917232	23414455.52	0.34	1
1	3638341.61	26773066	23134724.77	0.35	1
2	4233108.63	28928507	24695398.48	0.39	1
3	3939135.38	28101483	24162347.46	0.37	1
4	4944868.40	29706998	24762129.99	0.45	1
5	4918815.73	28908736	23989920.08	0.46	1
6	4933101.09	28790278	23857176.88	0.47	1
7	4578868.48	27308652	22729783.74	0.46	1
8	3099907.87	25665106	22565197.69	0.31	1
9	3150211.33	26214127	23063915.40	0.31	1
10	4031275.20	26556554	22525278.94	0.40	1
11	4209986.19	26221447	22011460.93	0.43	1
12	4475710.45	31744069	27268358.75	0.37	1
13	4381520.99	31069720	26688199.09	0.37	1
14	4420155.79	27172988	22752831.84	0.44	1
15	4176477.64	25496758	21320280.56	0.44	1

Table B.17: Model Error and Lack of Fit Results for algorithm SSS_GW and response variable number of steps (St).

GameID	SSLOF	SSRE	SSPE	F-Value	P-Value
0	1050854.47	7283667	6232812.89	0.38	1
1	1096403.77	7223303	6126899.41	0.40	1
2	1225914.44	7947792	6721877.74	0.41	1
3	1092003.90	7565352	6473348.24	0.38	1
4	1389906.66	7994863	6604955.93	0.48	1
5	1364659.43	7841928	6477268.78	0.48	1
6	1288196.67	7431407	6143209.93	0.47	1
7	1169026.03	7052466	5883440.03	0.45	1
8	919549.55	7359691	6440141.02	0.32	1
9	935858.03	7514456	6578598.36	0.32	1
10	1185911.01	7629175	6443263.92	0.42	1
11	1244553.65	7552756	6308202.55	0.45	1
12	1367296.91	9271345	7904048.28	0.39	1
13	1343819.32	9114256	7770437.06	0.39	1
14	1250179.20	7584069	6333890.26	0.45	1
15	1169860.57	7011349	5841488.26	0.45	1

Table B.18: Model Error and Lack of Fit Results for algorithm SSS_GW and response variable number of tokens collected (TO).

B.3 PW_FLOCK Model Error

This Appendix section presents a condensed form of the results of the lack-of-fit test, the residual and pure error terms for the each model. Due to the large number of game sessions and response variables we condensed the results. The results demonstrate the interval to which all of the model's F-Values occur within for each response variable. In addition, we display the P-Value indicating whether the lack-of-fit was significant (α<0.05) or non-significant(α>0.05).

Responses	Mean F-Value	Minimum F-Value	Maximum F-Value	P-Value
CC	0.33	0.15	0.52	1.0
FE	0.39	0.25	0.54	1.0
GH	0.3	0.15	0.49	1.0
LVL_COMP	0.36	0.2	0.53	1.0
PP	0.33	0.19	0.48	1.0
RSQ	0.32	0.16	0.51	1.0
Sc	0.33	0.19	0.49	1.0
ST	0.31	0.14	0.51	1.0
TO	0.33	0.19	0.52	1.0

Table B.19: The F-Values for all game sessions calculated using the residual error, pure error and lack-of-fit.

B.4 PW_GW Model Error

This Appendix section presents a condensed form of the results of the lack-of-fit test, the residual and pure error terms for the each model. Due to the large number of game sessions and response variables we condensed the results. The results demonstrate the interval to which all of the model's F-Values occur within for each response variable. In addition, we display the P-Value indicating whether the lack-of-fit was significant (α<0.05) or non-significant(α>0.05).

Responses	Mean F-Value	Minimum F-Value	Maximum F-Value	P-Value
CC	0.40	0.19	0.61	1.0
FE	0.44	0.26	0.67	1.0
GH	0.37	0.19	0.53	1.0
LVL_COMP	0.48	0.18	0.67	1.0
PP	0.39	0.15	0.62	1.0
RSQ	0.42	0.19	0.64	1.0
Sc	0.42	0.21	0.62	1.0
ST	0.41	0.2	0.6	1.0
TO	0.40	0.17	0.65	1.0

Table B.20: Model Error and Lack of Fit Results for algorithm PW_GW and response variable close calls (C). Part 1

Appendix C

Proof of Concept Statistics

The tables presented in this Appendix demonstrate the mean performance for all response variables over the game session selected for use in the adaptive game system prototype. Each table is composed of 3 rows indicating, each row presents the mean results for that life.

C.1 SSS_FLOCK Mean Performance Per Life

The SSS_FLOCK algorithm used the game session 0, indicating that none of the factors used a high level value.

ALGO_ID	LIFE	SCORE	nSteps	nLevCom	nCC	FC
7	0	748.6	138.5	0.07	43.5	1.59
7	1	609.3	118.1	0.26	50.4	1.30
7	2	571.7	111.6	0.3	50.2	1.18

Table C.1: The mean performance of the SSS_FLOCK for each life over the experiment.

ALGO_ID	LIFE	FE	GE	PPE	TOKE	RepStep
7	0	0.24	1.35	2.87	100.61	35.00
7	1	0.22	0.88	1.64	66.78	49.71
7	2	0.20	0.72	1.35	56.96	53.32

Table C.2: The mean performance of the SSS_FLOCK for each life over the experiment.

C.2 SSS_GW Mean Performance Per Life

The SSS_GW algorithm used in the adaptive game session included all factors at their low level except for fruit time.

ALGO_ID	LIFE	SCORE	nSteps	nLevCom	nCC	FC
6	0	590.15	91.78	0.01	36.25	0.90
6	1	446.53	86.95	0.06	55.27	0.83
6	2	400.80	80.76	0.15	66.42	0.74

Table C.3: The mean performance of the SSS_GW for each life over the experiment.

ALGO_ID	LIFE	FE	GE	PPE	TOKE	RepStep
6	0	0.2	1.95	2.04	71.00	18.74
6	1	0.1	1.32	1.26	50.83	34.86
6	2	0.1	1.08	0.88	37.26	42.63

Table C.4: The mean performance of the SSS_GW for each life over the experiment.

C.3 PW_FLOCK Mean Performance Per Life

The PW_FLOCK algorithm used in the adaptive game session included all factors at their low level except for PW_BAD_GHOST.

ALGO_ID	LIFE	SCORE	nSteps	nLevCom	nCC	FC
1	0	548.74	113.22	0.01	25.28	1.23
1	1	349.31	95.44	0.08	37.05	0.98
1	2	312.72	98.35	0.15	40.20	1.03

Table C.5: The mean performance of the PW_FLOCK for each life over the experiment.

ALGO_ID	LIFE	FE	GE	PPE	TOKE	RepStep
1	0	0.12	0.79	2.20	83.16	27.85
1	1	0.13	0.43	1.16	44.96	49.33
1	2	0.15	0.31	0.83	33.31	64.21

Table C.6: The mean performance of the PW_FLOCK for each life over the experiment.

C.4 PW_GW Mean Performance Per Life

The PW_GW algorithm used in the adaptive game session included all factors at their low level except for PW_BAD_GHOST.

ALGO_ID	LIFE	SCORE	nSteps	nLevCom	nCC	FC
0	0	495.4	91.8	0.00	30.4	0.91
0	1	313.9	81.8	0.02	45.5	0.77
0	2	222.2	91.5	0.04	45.0	0.95

Table C.7: The mean performance of the PW_GW for each life over the experiment.

ALGO_ID	LIFE	FE	GE	PPE	TOKE	RepStep
0	0	0.1	1.1	1.8	70.2	19.8
0	1	0.1	0.8	1.0	38.3	42.5
0	2	0.2	0.5	0.6	21.9	69.0

Table C.8: The mean performance of the PW_GW for each life over the experiment.

Appendix D

Regression Equations (Prototype)

The regression equations where utilized in the adaptive game system prototype as the heuristics for two cases: PW_FLOCK and PW_GW algorithms. These regression equations where used to calculate the expected value of the response variables, number of steps and score respectively.

D.1 PW_FLOCK Regression Equation

The regression equation for estimating the number of steps for the PW_FLOCK algorithm adaptive game session including three separate equations, one for each Pac-Man life. Presented in the Table D.1 are the coefficients for each performance measures used during each life.

LIFE	Coefficient	Score	nLevCom	nCC	FE	GE	PPE
0	10.1	0.18	-96.4	0.5	-5.7	-18.9	25.1
1	3.6	0.36	-184	0.3	-19.0	-40.7	-6.6
2	62.7	0.3	-157	0.2	-7.1	-37.3	0.5

Table D.1: The regression equation coefficients to estimate the steps of the PW_FLOCK algorithm in the adaptive game system prototype.

D.2 PW_GW Regression Equation

The regression equation for estimating the score for the PW_GW algorithm adaptive game session included one equation for the entire life. The coefficients for the regression equations are listed for each performance measure.

Coefficient	Step	nLevCom	nCC	GE	PP	RSQ
31.6	31.6	574	-0.02	101	17.3	-3.4

Table D.2: The regression equation coefficients to estimate the score of the PW_GW algorithm in the adaptive game system prototype.

Appendix E

Response Variables Descriptive Statistics

Within this Appendix we present descriptive statistics of the response variables over each algorithm. This information was used to select target ranges for adaptive game and helped build the heuristics.

E.1 SSS_FLOCK Descriptive Statistics

This Appendix presents some descriptive statistics over all games played with the SSS_FLOCK algorithm.

SSS_FLOCK	SCORE	nSteps	nLevCom	nCC	FC
Mean	1929.6	368.2	0.7	144.2	4.1
StDev	1017.0	161.1	0.7	104.6	3.2
Min	385	82	0	5	0
Max	6655	994	3	729	19
Range	6270	912	3	724	19
Sample Size	6144	6144	6144	6144	6144

Table E.1: SSS_FLOCK Response Variables Descriptive Statistics

SSS_FLOCK	FE	GE	PPE	TOKE	RepStep
Mean	0.7	3.0	5.9	224.4	138.0
StDev	0.9	2.0	2.4	91.5	72.3
Min	0	0	1	70	3
Max	7	13	16	609	453
Range	7	13	15	539	450
Sample Size	6144	6144	6144	6144	6144

Table E.2: SSS_FLOCK Response Variables Descriptive Statistics

E.2 SSS_GW Descriptive Statistics

This Appendix presents some descriptive statistics over all games played with the SSS_GW algorithm.

SSS_FLOCK	SCORE	nSteps	nLevCom	nCC	FC
Mean	1437.5	259.5	0.2	158.0	2.5
StDev	704.1	108.7	0.4	120.2	2.2
Min	300	58	0	4	0
Max	7255	993	3	1092	19
Range	6955	935	3	1088	19
Sample Size	49152	49152	49152	49152	49152

Table E.3: SSS_GW Response Variables Descriptive Statistics

SSS_FLOCK	FE	GE	PPE	TOKE	RepStep
Mean	0.3	4.4	4.2	159.1	96.2
StDev	0.6	2.3	1.5	57.1	55.2
Min	0	0	0	52	3
Max	6	19	16	598	460
Range	6	19	16	546	457
Sample Size	49152	49152	49152	49152	49152

Table E.4: SSS_GW Response Variables Descriptive Statistics

E.3 PW_FLOCK Descriptive Statistics

This Appendix presents some descriptive statistics over all games played with the PW_FLOCK algorithm.

SSS_FLOCK	SCORE	nSteps	nLevCom	nCC	FC
Mean	1210.8	307.0	0.3	102.5	3.2
StDev	674.8	158.0	0.5	111.8	2.9
Min	140	32	0	3	0
Max	5790	1050	3	1751	21
Range	5650	1018	3	1748	21
Sample Size	393216	393216	393216	393216	393216

Table E.5: PW_FLOCK Response Variables Descriptive Statistics

SSS_FLOCK	FE	GE	PPE	TOKE	RepStep
Mean	0.4	1.5	4.2	161.4	141.4
StDev	0.7	1.3	1.6	60.8	111.2
Min	0	0	0	28	0
Max	8	13	14	587	889
Range	8	13	14	559	889
Sample Size	393216	393216	393216	393216	393216

Table E.6: PW_FLOCK Response Variables Descriptive Statistics

E.4 PW_GW Descriptive Statistics

This Appendix presents some descriptive statistics over all games played with the PW_GW algorithm.

SSS_FLOCK	SCORE	nSteps	nLevCom	nCC	FC
Mean	1031.5	265.1	0.1	120.7	2.6
StDev	482.6	173.4	0.2	140.8	3.1
Min	145	34	0	0	0
Max	6190	1051	2	2061	21
Range	6045	1017	2	2061	21
Sample Size	3145728	3145728	3145728	3145728	3145728

Table E.7: PW_GW Response Variable Descriptive Statistics

SSS_FLOCK	FE	GE	PPE	TOKE	RepStep
Mean	0.4	2.3	3.4	130.4	131.3
StDev	0.7	1.7	1.1	38.9	150.5
Min	0	0	0	29	0
Max	12	18	12	473	960
Range	12	18	12	444	960
Sample Size	3145728	3145728	3145728	3145728	3145728

Table E.8: PW_GW Response Variables Descriptive Statistics

Appendix F

Effects Descriptive Statistics

F.1 Close Calls Factor Effects Descriptive Statistics

In Table F.1 descriptive statistics of the effect sizes are presents for the response variable close calls.

	N	Absolute_Mean	StdDev	Min	Max
SSS_FLOCK	2048	2.34	4.14	-31.61	38.25
SSS_GW	16384	3.08	4.49	-39.20	58.25
PW_FLOCK	131072	2.67	4.16	-49.43	51.25
PW_GW	1048576	3.49	5.20	-96.0	88.6

Table F.1: Close Calls Response Variable Descriptive Statistics

F.2 Fruit Eaten Factor Effects Descriptive Statistics

In Table F.2 descriptive statistics of the effect sizes are presents for the response variable fruit eaten.

	N	Absolute_Mean	StdDev	Min	Max
SSS_FLOCK	2048	0.19	0.03	-0.25	0.37
SSS_GW	2048	0.02	0.02	-0.17	0.16
PW_FLOCK	16384	0.01	0.02	-0.36	0.57
PW_GW	1048576	0.02	0.03	-1.40	0.43

Table F.2: Fruit Eaten Response Variable Descriptive Statistics

F.3 Ghosts Eaten Factor Effects Descriptive Statistics

In Table F.3 descriptive statistics of the effect sizes are presents for the response variable ghosts eaten.

	N	Absolute_Mean	StdDev	Min	Max
SSS_FLOCK	2048	0.04	0.07	-0.36	0.87
SSS_GW	16384	0.06	0.09	-0.99	1.46
PW_FLOCK	131072	0.03	0.05	-0.56	0.95
PW_GW	1048576	0.04	0.06	-0.90	1.45

Table F.3: Ghosts Eaten Response Variable Descriptive Statistics

F.4 Levels Completed Factor Effects Descriptive Statistics

In Table F.4 descriptive statistics of the effect sizes are presents for the response variable levels completed.

	N	Absolute_Mean	StdDev	Min	Max
SSS_FLOCK	2048	0.01	0.02	-0.18	0.17
SSS_GW	16384	0.012	0.015	-0.154	0.087
PW_FLOCK	131072	0.01	0.01	-0.22	0.22
PW_GW	1048576	0.006	0.008	-0.138	0.076

Table F.4: Levels Completed Response Variable Descriptive Statistics

F.5 Power-Pellets Collected Factor Effects Descriptive Statistics

In Table F.5 descriptive statistics of the effect sizes are presents for the response variable power-pellets collected.

	N	Absolute_Mean	StdDev	Min	Max
SSS_FLOCK	2048	0.05	0.08	-0.72	0.69
SSS_GW	16384	0.04	0.056	-0.689	0.362
PW_FLOCK	131072	0.03	0.06	-0.88	1.21
PW_GW	1048576	0.02	0.04	-0.98	0.47

Table F.5: Power-Pellets Collected Response Variable Descriptive Statistics

F.6 Repeated Squares Factor Effects Descriptive Statistics

In Table F.6 descriptive statistics of the effect sizes are presents for the response variable repeated squares.

	N	Absolute_Mean	StdDev	Min	Max
SSS_FLOCK	2048	1.55	2.60	-20.85	18.72
SSS_GW	16384	1.41	2.02	-27.88	13.41
PW_FLOCK	131072	2.55	4.04	-57.20	60.28
PW_GW	1048576	3.52	6.11	-216.50	81.05

Table F.6: Repeated Squares Response Variable Descriptive Statistics

F.7 Score Factor Effects Descriptive Statistics

In Table F.7 descriptive statistics of the effect sizes are presents for the response variable score.

	N	Absolute_Mean	StdDev	Min	Max
SSS_FLOCK	2048	22.03	36.72	-321.69	255.96
SSS_GW	16384	18.37	25.46	-268.73	178.0
PW_FLOCK	131072	15.98	25.15	-355.16	416.72
PW_GW	1048576	11.87	18.03	-489.11	287.81

Table F.7: Score Response Variable Descriptive Statistics

F.8 Steps Factor Effects Descriptive Statistics

In Table F.8 descriptive statistics of the effect sizes are presents for the response variable steps.

	N	Absolute_Mean	StdDev	Min	Max
SSS_FLOCK	2048	3.48	5.82	-49.02	36.91
SSS_GW	16384	2.79	4.01	-57.57	28.30
PW_FLOCK	131072	1.42	2.24	-32.57	41.73
PW_GW	1048576	4.04	7.03	-241.59	92.50

Table F.8: Steps Response Variable Descriptive Statistics

F.9 Tokens Collected Factor Effects Descriptive Statistics

In Table F.9 descriptive statistics of the effect sizes are presents for the response variable tokens collected.

	N	Absolute_Mean	StdDev	Min	Max
SSS_FLOCK	2048	2.00	3.35	-27.95	24.51
SSS_GW	16384	1.48	4.00	-57.56	28.29
PW_FLOCK	131072	1.42	2.24	-32.57	41.73
PW_GW	1048576	0.94	1.45	-43.66	16.13

Table F.9: Tokens Collected Response Variable Descriptive Statistics

Bibliography

[1] E. Adams. Balancing Games with Positive Feedback. *Article published in Gamasutra and is available online at http://www.gamasutra.com/features/20020104/adams-01.htm*, 2002.

[2] A. Agogino, K. Stanley, and R. Miikkulainen. Online Interactive Neuroevolution. *Neural Processing Letters*, 11(1):29–38, 2000.

[3] David Aha, Matthew Molineaux, and Marc Ponsen. Learning to win: Case-based plan selection in a real-time strategy game. In Héctor Muñoz-Ávila and Francesco Ricci, editors, *Case-Based Reasoning Research and Development*, volume 3620 of *Lecture Notes in Computer Science*, pages 5–20. Springer Berlin / Heidelberg, 2005.

[4] Revolution Analytics. R is Ready for Business. *Article published at http://www.revolutionanalytics.com/products/revolution-enterprise.php*, 2011.

[5] G. Andrade, G. Ramalho, H. Santana, and V. Corruble. Challenge-sensitive action selection: an application to game balancing. In *Intelligent Agent Technology, IEEE/WIC/ACM International Conference on*, pages 194–200. IEEE, 2005.

[6] M.V. Aponte, G. Levieux, and S. Natkin. Scaling the level of difficulty in single player video games. *Entertainment Computing–ICEC 2009*, pages 24–35, 2009.

[7] C. Bailey and M. Katchabaw. An experimental testbed to enable auto-dynamic difficulty in modern video games. In *Proceedings of the 2005 GameOn North America Conference*, pages 18–22, 2005.

[8] W. Barendregt, MM Bekker, DG Bouwhuis, and E. Baauw. Identifying usability and fun problems in a computer game during first use and after some practice. *International Journal of Human-Computer Studies*, 64(9):830–846, 2006.

[9] R. Bartle. Presence and flow: Ill-fitting clothes for virtual worlds. *Techné*, page 39, 2007.

[10] R.A. Bartle. Designing Virtual Worlds. 2003.

[11] C.M. Bateman and R. Boon. *21st century game design*. Charles River Media, 2006.

[12] D.M. Bourg and G. Seemann. *AI for Game Developers*. O'Reilly, 2004.

[13] D. Charles and M. Black. Dynamic Player Modelling: A Framework for Player-Centered Digital Games. *International Conference on Computer Games: Artificial Intelligence, Design and Education, Microsoft Campus, Reading, Nov*, pages 8–10, 2004.

[14] D. Charles, M. McNeill, M. McAlister, M. Black, A. Moore, K. Stringer, J. Kücklich, and A. Kerr. Player-centred game design: Player modelling and adaptive digital games. 285, 2005.

[15] D. Charles, M. McNeill, M. McAlister, M. Black, A. Moore, K. Stringer, J. Kücklich, and A. Kerr. Player-Centred Game Design: Player Modelling and Adaptive Digital Games. *Proceedings of DiGRA 2005 Conference: Changing Views–Worlds in Play*, 2005.

[16] J. Chen. Flow in games. *Master's thesis, University of Southern California*, 2006.

[17] K. Cheng and P.A. Cairns. Behaviour, realism and immersion in games. *Conference on Human Factors in Computing Systems*, pages 1272–1275, 2005.

[18] B. Cowley, D. Charles, M. Black, and R. Hickey. Using decision theory for player analysis in pacman. *Optimizing Player Satisfaction in Computer and Physical Games*, page 41.

[19] B. Cowley, D. Charles, M. Black, and R. Hickey. User-System-Experience Model for User Centered Design in Computer Games. *LECTURE NOTES IN COMPUTER SCIENCE*, 4018:419, 2006.

[20] M. Csikszentmihalyi. *Flow: The Psychology of Optimal Experience*. Harper & Row, 1990.

[21] L. Ermi and F. Mäyrä. Fundamental Components of the Gameplay Experience: Analysing Immersion. *Changing views: worlds in play, DiGRA conference, Vancouver*, 2005.

[22] M.A. Federoff. *Heuristics and Usability Guidelines for the Creation and Evaluation of Fun In Video Games*. PhD thesis, Indiana University, 2002.

[23] D. Grammenos. Game over: learning by dying. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1443–1452. ACM, 2008.

[24] W.G. Hopkins. Probabilities of clinical or practical significance. *Sportscience*, 6:201, 2002.

[25] R. Hunicke and V. Chapman. Ai for dynamic difficulty adjustment in games. In *Challenges in Game Artificial Intelligence AAAI Workshop*, pages 91–96, 2004.

[26] B. Ip and E. Adams. From Casual to Core: A Statistical Mechanism for Studying Gamer Dedication. *Article published in Gamasutra and is available online at http://www.gamasutra.com/features/20020605/ip-pfv.htm*, 2002.

[27] R. Jain. *The art of computer systems performance analysis*, volume 182. John Wiley & Sons New York, 1991.

[28] L. Laufer and N. Bottyan. Unbeatable program: skin signals betray a gamer's moves. *Article published in Newscientist and is available online at http://www.newscientist.com/blog/technology/2007/08/skin-signals-betray-gamers-moves. html*, 2007.

[29] N. Lazzaro. Why we play games: Four keys to more emotion without story. *Design*, 18:1–8, 2005.

[30] S. Miller. Auto-Dynamic Difficulty. *Published in Scott Miller Game Matters Blog (http://dukenukem. typepad. com/game-matters/2004/01/autoadjusting. html)*, 2004.

[31] I. Millington. *Artificial Intelligence for Games*. Morgan Kaufmann, 2006.

[32] C. Pedersen, J. Togelius, and G.N. Yannakakis. Modeling player experience for content creation. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(1):54–67, 2010.

[33] A. Plaat, J. Schaeffer, W. Pijls, and A. Bruin. SSS*= AB+ TT. 1995.

[34] M. PONSEN, H. MUNOZ-AVILA, P. SPRONCK, and D.W. AHA. Automatically generating game tactics through evolutionary learning. *The AI magazine*, 27(3):75–84, 2006.

[35] M. Ponsen, P. Spronck, H. Muñoz-Avila, and D.W. Aha. Knowledge acquisition for adaptive game AI. *Science of Computer Programming*, 67(1):59–75, 2007.

[36] P. Rani, N. Sarkar, and C. Liu. Maintaining optimal challenge in computer games through real-time physiological feedback. In *Proceedings of the 11th International Conference on Human Computer Interaction*, pages 184–192, 2005.

[37] C.W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH Computer Graphics*, 21(4):25–34, 1987.

[38] S.J. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1995.

[39] B. Schwab. *AI Game Engine Programming*. Charles River Media, 2004.

[40] Bart Simon. Wii are Out of Control: Bodies, Game Screens and the Production of Gestural Excess. *SSRN eLibrary*, 2009.

[41] P. Spronck. A Model for Reliable Adaptive Game Intelligence. *IJCAI-05 Workshop on Reasoning, Representation, and Learning in Computer Games*, pages 95–100, 2005.

[42] P. Spronck and Others. Adaptive game AI. *SIKS dissertation series*, 2005.

[43] P. Spronck, M. Ponsen, I. Sprinkhuizen-Kuyper, and E. Postma. Adaptive game AI with dynamic scripting. *Machine Learning*, 63(3):217–248, 2006.

[44] P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma. Difficulty scaling of game ai. In *Proceedings of the 5th International Conference on Intelligent Games and Simulation (GAME-ON 2004)*, pages 33–37, 2004.

[45] K.O. Stanley and R. Miikkulainen. Efficient Reinforcement Learning through Evolving Neural Network Topologies. *Network (Phenotype)*, 1(2):3.

[46] R.S. Sutton and A.G. Barto. *Introduction to Reinforcement Learning*. MIT Press Cambridge, MA, USA, 1998.

[47] P. Sweetser and P. Wyeth. GameFlow: a model for evaluating player enjoyment in games. *Computers in Entertainment (CIE)*, 3(3):3–3, 2005.

[48] P. Sweetser and P. Wyeth. GameFlow: a model for evaluating player enjoyment in games. *Computers in Entertainment (CIE)*, 3(3):3, 2005.

[49] J. Sykes and S. Brown. Affective gaming: measuring emotion through the gamepad. *Conference on Human Factors in Computing Systems*, pages 732–733, 2003.

[50] I. Szita and A. Lorincz. Learning to play using low-complexity rule-based policies: Illustrations through Ms. Pac-Man. *Journal of Artificial Intelligence Research*, 30:659–684, 2007.

[51] J. Togelius, R. De Nardi, and S.M. Lucas. Towards automatic personalised content creation for racing games. In *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, pages 252–259. IEEE, 2007.

[52] KW Wong, CC Fung, A. Depickere, and S. Rai. Static and Dynamic Difficulty Level Design for Edutainment Game Using Artificial Neural Networks. *LECTURE NOTES IN COMPUTER SCIENCE*, 3942:463, 2006.

[53] G.N. Yannakakis and J. Hallam. Evolving opponents for interesting interactive computer games. *From animals to animats*, 8:499–508, 2004.

[54] G.N. Yannakakis and J. Hallam. A generic approach for generating interesting interactive pac-man opponents. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 94–101, 2005.

[55] G.N. Yannakakis and J. Hallam. Modeling and augmenting game entertainment through challenge and curiosity. *International Journal of Artificial Intelligence Tools*, 16(6):981–1000, 2007.

[56] G.N. Yannakakis and J. Hallam. Real-time Adaptation of Augmented-Reality Games for Optimizing Player Satisfaction. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 103–110, 2008.

[57] G.N. Yannakakis and J. Hallam. Real-time game adaptation for optimizing player satisfaction. *Computational Intelligence and AI in Games, IEEE Transactions on*, 1(2):121–133, 2009.

[58] R. Yeung and J. Gumberg. Video Games and Internet to Remain Fastest-Growing Industry Segments; Digital Distribution and Mobile Music Will Reinvigorate Recorded Music Industry. *Published in PriceWaterHouseCOOPERS (http://www.pwc.com/extweb/ncpressrelease.nsf/DocID/9F2A3D2CFD45635D85257027006F2* 2005.

[59] D. Zapf, F.C. Brodbeck, M. Frese, H. Peters, and J. Pr
"umper. Errors in working with office computers: A first validation of a taxonomy for observed errors in a field setting. *International Journal of Human-Computer Interaction*, 4(4):311–339, 1992.

[60] R. Zhao. *Applying agent modeling to behaviour patterns of characters in story-based games*. PhD thesis, University of Alberta, 2010.

Vita

NAME:	Jamey Fraser
POST-SECONDARY EDUCATION AND DEGREES:	University of Western Ontario London, Ontario, Canada 2007-2011 M.Sc. (Computer Science) Saint Mary's University Halifax, Nova Scotia, Canada 2002-2006 B.Sc. (Honours Computer Science)
RELATED WORK EXPERIENCE:	Video Game Programmer Big Blue Bubble London, Ontario, Canada 2008 - 2011
TEACHING ASSISTANTSHIPS	University of Western Ontario London, Ontario, Canada Computer Science Fundamentals II University of Western Ontario London, Ontario, Canada Introduction to Software Engineering University of Western Ontario London, Ontario, Canada Computer Science Fundamentals I Saint Mary's University Halifax, Nova Scotia, Canada Introduction to Calculus