Applying Causal Models to Dynamic Difficulty Adjustment in Video Games

by

Jeffrey Peter Moffett

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

May 2010

APPROVED:

_____

Professor Charles Rich, Thesis Advisor

_____

Professor Joseph Beck, Thesis Advisor

_____

Professor David Finkel, Thesis Reader

_____

Professor Michael Gennert, Head of Department

Abstract

We have developed a causal model of how various aspects of a computer game influence how much a player enjoys the experience, as well as how long the player will play. This model is organized into three layers: a generic layer that applies to any game, a refinement layer for a particular game genre, and an instantiation layer for a specific game. Two experiments using different games were performed to validate the model. The model was used to design and implement a system and API for Dynamic Difficulty Adjustment(DDA). This DDA system and API uses machine learning techniques to make changes to a game in real time in the hopes of improving the experience of the user and making them play longer. A final experiment is presented that shows the effectiveness of the designed system.

Acknowledgments

I would like to begin be thanking both of my advisors, Joseph Beck and Charles Rich. Both have been invaluable resources in their respective domains. Having two advisors that come from such different backgrounds provided for some insightful (albeit heated) discussions about how to proceed.

Under Joe's tutelage, I feel that I should also be graduating with at least a minor in statistics. I don't know how he is able to come up with so many different analysis ideas at once. I also think you single handedly made Excel a viable analysis tool for me.

Chuck, you always seem to find the bright point in everything I do, even total failures. You taught me that there is always something to learn from every experience. In two years, you transformed me from a undergraduate with no research experience into a research paper crazy student. I honestly look at all my projects now to see if I can bust a paper out of them.

Special thanks goes to my reader David Finkel. I wrote this thesis and even I do not want to read it. Thank you for taking the time to analyze and critique my research and writing skills.

Thanks is also due to by fellow Teaching Assistant and band members, Jeremy Denham and Suvesh Pratapa. Graduate life would have been very boring without all the crazy ideas that you two always came up with. I should really be writing my thesis on Wiffle Hall. Can at least get a colloquium out of it.

Finally, I would like to thank the entire WPI Computer Science Department. Thanks is due to the professors, secretaries, and Mike Voorhis for loaning me his headphones (p.s. you are never getting them back!).

Contents

List of Figures

List of Tables

1 Introduction

1.1 Context of the Problem

Recent work on applying machine learning and optimization techniques to computer games, such as first-person shooters [HHSA08] and real-time strategy games [UJG08], has assumed that the goal is to achieve the best possible AI performance. In fact, game design books [AR07] warn against exactly this fallacy. The goal of applying AI to games should be to improve the player's experience. An opponent that is overly dominant is not fun!

However, in order to use AI to balance a game (either dynamically or off line) with respect to the experience of a player, we must need a formal mathematical model of the factors that exist in video games. This formula can then be manipulated to optimize the player's experience in a video game. This is the primary motivation for our research.

1.2 Problem Statement

The core problem of this thesis was to design and implement a model for optimizing the player's experience while playing a video game. To do this, some formal notation which approximates "fun" needs to be quantified. If fun can be defined in terms of a numerical equation, it can be optimized using well known machine learning techniques.

The difficulty lies in converting fun into a numerical equation. Fun itself is an abstract concept. In many situations it is difficult to qualitatively explain the experience of a player, let alone define it quantitatively.

1.3 Application Domain

The domain this thesis concentrates on is video games. This applies to games created for the computer, home consoles, as well as small media devices such as cell phones. While video games exist on a wide array of platforms, similar terms are used in most, if not all, video games.

1.3.1 Terminology

- Avatar: The character (or set of characters) in the game a human player controls.
- Non-Player Character (NPC): All characters not controlled by a human player. These characters are controlled by the inner logic of the video game.
- Environment: The physical representation of the game world. The environment defines how the avatar and NPC are able to navigate in the game. Examples of environmental factors include the physics in the game, the topographical layout of the world, as well as the current weather in the game.
- Spawn: When a new character is created and placed at a location in the environment. Both Avatar and NPC players can spawn. Avatar players usually only spawn at the beginning of a round or after a death. Spawning is usually controlled by the environment.

1.4 Related Work

There has been much research aimed at defining the abstract concept of fun in mathematical terms.

1.4.1 EVE

EVE [Bur07] explains fun in terms of Bayesian information theory. It is comprised of three components: Expectation, Violation, and Explanation. EVE argues that a positive experience (fun) is produced when either a situation matches our expectations or when a situation violates our expectations, but we have an explanation for why the violation occurred. EVE has been applied to the analysis of gambling games, music creation [Bur06] and computer games [Pis06].

1.4.2 MDA

MDA [Hun05] was designed as a way to understand how factors in a video game interact. A game is decomposed into three components: mechanics, dynamics, and aesthetics. The mechanics of the game are the low-level data structures and algorithms. Dynamics are the run-time interactions between the user and the system. Aesthetics are the emotional responses of the player while playing the game.

MDA has been used to create a game-monitoring system, called Hamlet, which attempts to improve the player's aesthetic experience in combat-based computer games. The system dynamically estimates the player's probability of dying based on the experience of previous enemy encounters and intervenes to make the game easier when needed. Initial results showed that Hamlet decreased the number of player deaths while keeping its operation virtually undetected by the player.

1.4.3 GameFlow

The GameFlow model [SW05] provides a qualitative approach to evaluating player enjoyment of computer games, using the well known concept of "flow" [Cs98] as the starting point. Each of the eight elements of "flow" are mapped to typical game elements. GameFlow also adds a ninth element, social interactions, which exists

in some games but is not included in the standard "flow" model. The GameFlow model has been used to analyze player enjoyment in two real time strategy games, Warcraft 3 and Lords of Everquest.

1.4.4 Heuristic Entertainment Value

Yannakakis and Hallam [YH07] developed a heuristic function to predict the level of enjoyment in predator/prey games, such as Pac-Man. By extensively analyzing this type of game, three elements were theorized to directly effect entertainment value: challenge level, predator behavior diversity, and predator spatial diversity. The heuristic function was validated by comparing user surveys of game sessions with the values predicted by the heuristic function. Of the related work mentioned above, this is the most similar to ours.

2 Creating the Causal Model

2.1 Causal Model Definition

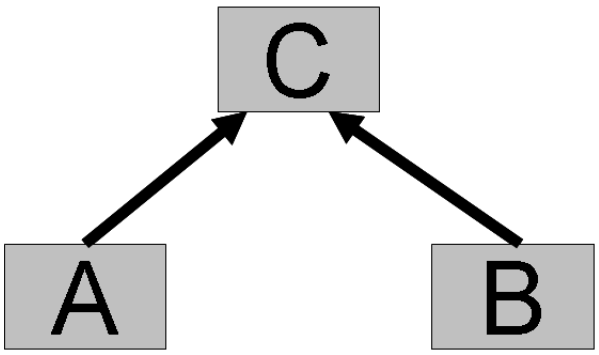

Figure 1: Example Causal Model

A causal model [Coh95] is an directed acyclic graph (DAG) in which nodes represent modeled quantities or factors and each edge indicates that the source

factor directly influences the target factor. Edges are also known as causal links. Causal links can be assigned values that represent the strength of the association between the factors they connect. Using a causal model allows complex relationships to be visualized. Multiple factors can combine to effect a single factor. One factor, on the other hand, can have influence on multiple factors.

In the example in Figure 1, factor A and B both have an effect on C. Changes in either of these factors will have a direct, measurable effect on factor C. This model also states that there is no relationship between factors A and B. Changing the value of factor A does not affect factor B .

While placing a causal link between two factors represents an important relationship between the factors, the lack of a causal link is more powerful. A causal model built with a causal link between every pair of factors (i.e., a complete graph) will perform as well or better than any model built with the same set of factors[Pea00]. Occam's Razor states that the simplest answers should be preferred. Model selection involves the removal of causal links at the price of some level of accuracy. Removing a causal link is a powerful statement; this action strengthens the causal model by lowering its complexity,this adhering to Occam's Razor, while potentially decreasing its accuracy.

2.2 Application to Video Games

Causal models are used to define a set of interactions between factors in a game. These interactions can be broken down until they are defined as a set of manipulated factors. These manipulated factors can then be independently manipulated to achieve a controllable effect on fun.

Our causal model was designed using domain knowledge gathered from multiple sources. The main source of domain knowledge came from an extensive literature

review. This review explored both publications on game design as well as current research on Dynamic Difficulty Adjustment (DDA). The goal of DDA is to perform changes to a game in real time in an effort to prevent a game from becoming too easy or too challenging. DDA is based heavily on the previously discussed idea of "flow"[Cs98]. Knowledge was also obtained from the advisors of this thesis, with one advisor belonging to the Interactive Media and Game Development (IMGD) department at WPI. Finally, knowledge was obtained from previous experience playing video games.

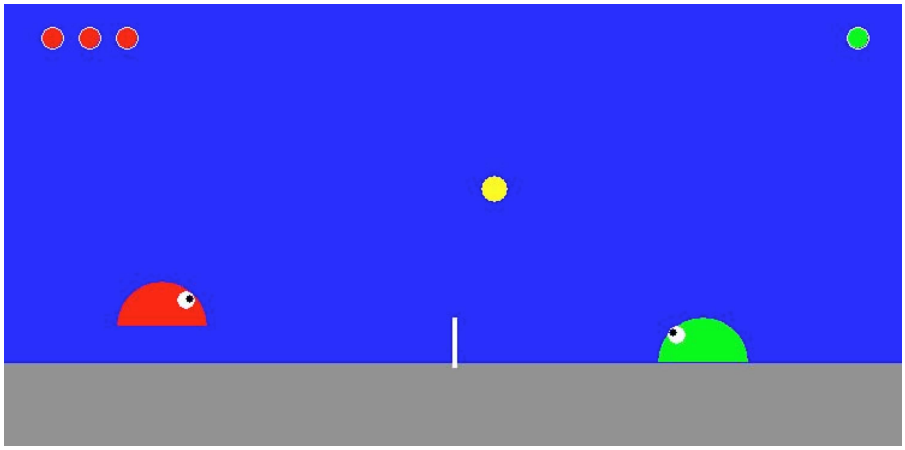

Figure 2: Screenshot from Slime Volleyball

The causal model was applied to the video game Slime Volleyball in parallel with the model's development. Slime Volleyball is an open source Java Applet game originally created by Quin Pendragon for two players. It is a two-dimensional volleyball game (see Figure 2) in which the player controls the creature (slime) on one side of the net. Each creature can move laterally and jump to block and return the ball. As in volleyball, you lose the point if the ball touches the ground on your side of the net. A match ends when either side scores five points. When a version of the model was proposed, we applied it to Slime Volleyball to verify that it fully covered the factors in the game.

To create a causal model in the video game domain, some additional terms are required:

- Game Metric: A factor that can be used to measure the quality of a character relative to other characters in the game. Examples include score, kill/death ratio, and time to complete a level. These values are calculated by the game.

- Performance Factor: A factor that can be used to measure the quality of a character, regardless of the quality of the other characters. In Slime Volleyball, the average distance to the ball is a performance factor since the character can control where on the opponent's side of the field the ball should be directed. While the opponent is able to optimally position itself to return the ball, the action undertaking is controlled completely by the character in possession of the ball. Margin of victory is a game metric since scoring a point relies on the offensive and defensive skill of both players. Similar to game metrics, these values are calculated by the game.

- Manipulated Factor: A factor that can be directly influenced during the game. While game metrics and performance factors are simply observed, manipulated factors can be changed dynamically at run time. An important property of manipulated factors is that they are independent of each other. Changing one manipulated factor has no effect on the other manipulated factors. A manipulated factor will have no incoming causal links in the model. The manipulated factors are used to control the level of enjoyment the user is having.

- Ambiance: The artistic, stylistic and aesthetic factors of the game. Ambiance includes both visual and auditory aspects of a game.

- Perceived Fairness: Player's belief in their chance to be victorious in the game. Perceived Fairness can range from a player believing they have no chance to win a game, regardless of their ability, to believing that they will win regardless of the actions that they take in the game. Values for perceived fairness are not obtained from the game, the player must define this value subjectively.

2.3 Levels of Causal Model

In all studies conducted for this thesis, fun will be measured by surveying the player after a game. Fun could in principle be determined in other ways, such as direct observation of the player by judges, amount of time played, or measurement of biological signals.

The causal model developed can be abstracted at three levels. The levels are shown in Figure 3. The generic level can be applied to all video games. The genre-specific level further refines the model, utilizing knowledge inherent in various game genres. Finally, the game-specific model can be used to directly map the model to actual factors in a game.

2.3.1 First Level: Generic Model

The generic model covers both games in which the player plays only versus the environment and games in which there is a nonplayer character (NPC) as an opponent. The specific game we chose for initial validation of the model involves an NPC. For a game without an NPC opponent, all the NPC-related nodes in the model should simply be ignored.

The first factor directly influencing fun is the player. Players vary in a great many ways, including, age, gender, and gaming experience. In applications of this model, the player is not usually a factor that can be manipulated.

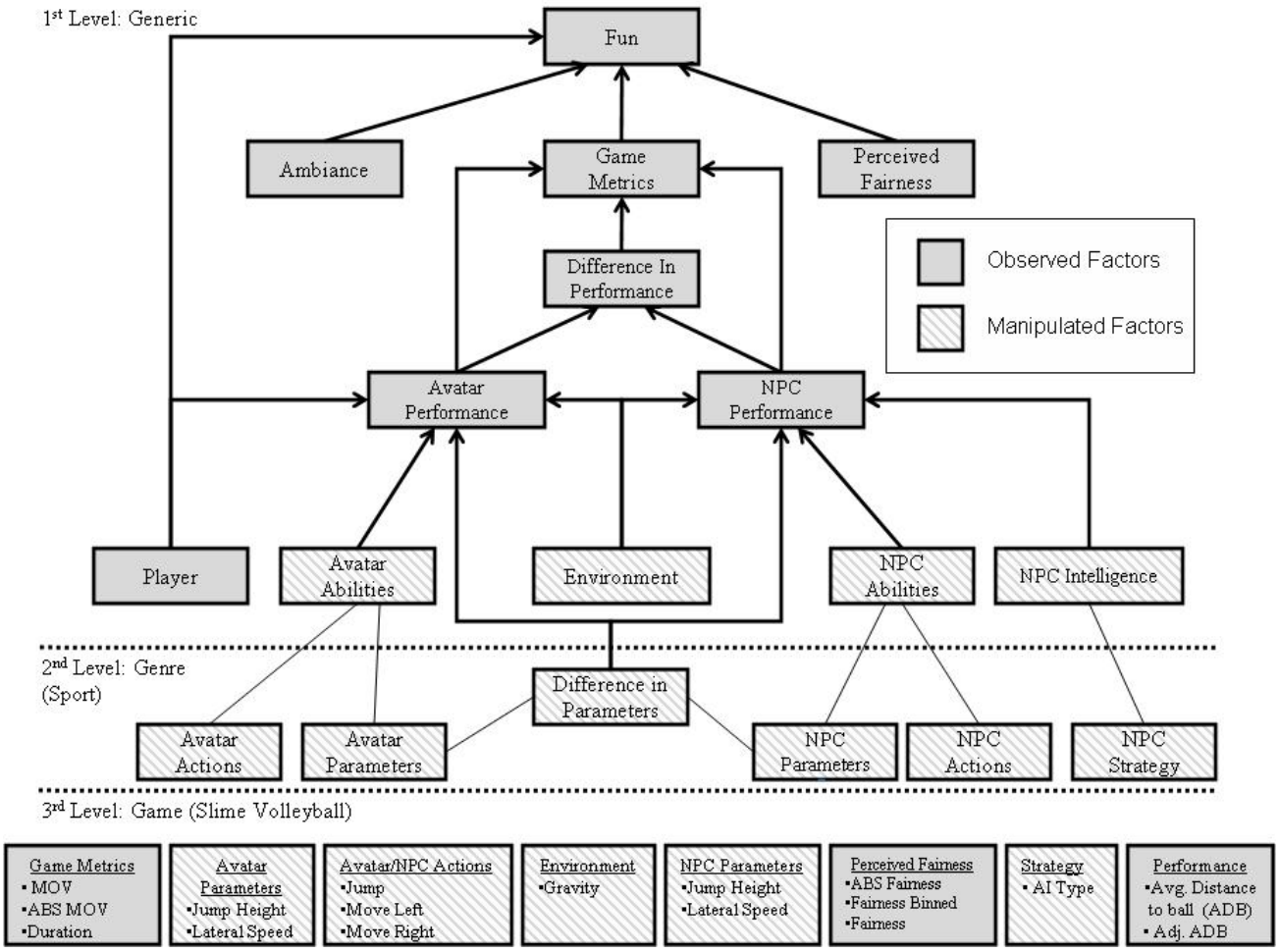

Figure 3: Initial Slime Volleyball Causal Model

The second factor directly influencing fun is the game *ambiance*. Even though these aspects of a game undoubtedly affect how much fun it is to play, this is not an area we explored in Slime Volleyball. Thus there are currently no nodes in the model which directly influence *ambiance*.

The third factor directly influencing fun is the *game metrics*. A player who is winning a game will have a different view of the amount of fun they are having as opposed to a player that is losing.

The fourth factor directly influencing fun is *perceived fairness*. Players want to believe that they have a chance of winning. It's not fun to play a game in which

your opponents have an unfair advantage, or when you have an unfair advantage over them. This factor can be seen as an extension on the concept of "flow" [Cs98]. Just as with *ambiance*, we were unsure about which factors would have a direct influence on *perceived fairness*, so the node has not incoming links. To explore the influence *perceived fairness* has in the game, players were asked to give a value to this factor after playing (see appendix for the *perceived fairness* survey question).

Since *game metrics* are used to judge character performance, two of the direct influences on *game metrics* are the *avatar* (character controlled by the player) *performance* and the *NPC performance*. Also, in many cases, what matters most is not the individual performances of the avatar and NPC, but their relative performance. The *difference in performance* node was added to the model to reflect this influence. *Difference in performance* blurs the distinction between *performance factors* and *game metrics* since it derives its value in relation to the performance of other characters. Since the values come from character *performance factors*, the choice was made to keep these factors in the performance node.

Since the player controls his/her *avatar*, there is an obvious direct influence from the player to *avatar performance*. The performance of the *avatar* is also typically influenced by a set of *avatar abilities*, the details of which depend on the game genre and the specific game.

Symmetrically, the *NPC performance* is directly influenced by the *NPC abilities*. Corresponding to the player's control of the *avatar*, the *NPC intelligence* controls and therefore directly influences *NPC performance*.

Finally, the *environment* node represents everything in the game world that is not an *avatar* or *NPC*. In general, changes in the environment directly influence both *avatar* and *NPC performance*.

2.3.2 Second Level: Genre-Specific Model

The genre-specific model refines the generic model with concepts and distinctions related to a particular game genre, such as first-person shooters, real-time strategy games, or sports games. The refinement process may include both adding additional nodes and edges and expanding/splitting a single node into one or more nodes. Different genres will require a different structure at this level.

Figure 3 shows the refinement of the generic model for the sports game genre. For sports games, the *avatar* and *NPC abilities* nodes are each split into actions and parameters. Actions are the fundamental vocabulary of interactions that a character can have with the game world. Parameters constrain these actions. For example, jumping is an action; how high you can jump is a parameter. In many sports games, the *avatar* and *NPC* have the same actions, so that what is important is the difference in their parameters. We have therefore added a difference in parameters node with direct influences to both *avatar* and *NPC performance*.

2.3.3 Third Level: Game-Specific Model

The game-specific model is different than the generic model and the genre model. In the game-specific model, we instantiate the abstract concepts in the generic or genre model with one or more concrete quantities or factors in a specific game. Each node may contain multiple factors. It is not required that each factor in a node be included in this model. These factors are simply candidates for inclusion in the model. The goal is to find a set of factors and causal links so that there is a path from the *manipulated factors* to fun. Figure 3 illustrates all three levels of the causal model for Slime Volleyball.

Starting on the left of the third level in Figure 3, we see three concrete game metrics candidates: *margin of victory* (avatar score minus NPC score), *absolute*

margin of victory (absolute value of margin) and *duration* (how long the player played the game before quitting).

For *perceived fairness*, there are also three concrete factors. These factors were created by taking the player supplied value for *perceived fairness* and looking at it in different ways. The first is the *raw fairness value*. This is the value provided by the player. Players were asked how fair they believed the game was on a 0-9 scale (where 0 means computer had advantage, 5 was an even match, and 9 means you had the advantage). The second factor for *perceived fairness* was the *binned fairness value*. Here, the *raw fairness value* was grouped into bins (0-3, 4-6, 7-9). This grouping focused on whether they player believed the game was fair, compared with the raw fairness that was concerned with how fair the player believed the game to be. The last factor for *perceived fairness* was *deviation from fair*. This value was calculated by taking the absolute value of the *raw fairness* value minus five). This factor was concerned with how far off from a fair game the player believed the game was, regardless of whether the player had the advantage, or the *NPCs* had the advantage.

Both the *avatar* and the *NPCs* in Slime Volleyball have the same set of actions, namely jump, move left and move right. However, each has its own jump height and lateral speed values.

The two candidates for both *avatar* and *NPC performance factors* were *average distance to ball* (the distance a volley requires the opponent to move in order to block and return it) and *adjusted average distance to ball* (average distance divided by the opponents lateral speed, i.e., how much time the opponent has to block and return the ball).

The only environment factor in Slime Volleyball is *gravity*. This controlled how the ball would move after it was hit.

Finally, the *NPC intelligence* in Slime Volleyball varies over three AI defense levels. At the lowest level, the *NPC* moves simply based on whether the ball is behind it or in front of it. At the medium level, it also takes into account the direction the ball is moving and whether it is falling or rising. At the high level, it calculates the spot where the ball will land.

3 Validating the Causal Model

The causal model described above was validated in a series of experiments. In these experiments, the strengths of the causal links in the model were evaluated using various statistical techniques. A majority of the analysis was performed using one of the following techniques.

- *Correlation:* Measures the relationship between two or more random variables. Correlations are bounded between -1 and 1, with numbers farther from zero implying a stronger relationship. Random variables with a causal relationship will have a strong correlation if the relationship is linear. The converse does not apply (i.e. correlation does not imply causality).

- *Partial Correlation:* Similar to correlation, partial correlation measures the relationship between two or more random variables. The distinction is that partial correlations allow for the effect of other random variables to be accounted for before determining a value.

- *Analysis of Variance (ANOVA):* This is used to compare the effect of a population that has been grouped in a particular way. ANOVA is used to determine whether the variance in each group is similar to the population variance or that the groups are statistically different from each other. ANOVA determines, for

example, if the different AI settings have a significant impact on how a *NPC* performs during the game.

3.1 Validation Study for Slime Volleyball

3.1.1 Experiment Design

As a first step toward validating the causal model in Figure 3, we conducted a study of 136 players of Slime Volleyball. The basic methodology of the study was to vary the values of the *manipulated factors* in the model and to measure the values at the other nodes. Validation of the model was based primarily on direct and partial correlations. For example, our causal model posits that the partial correlation between difference in performance and fun, adjusting for the influence of game metrics, will be close to zero.

The top two layers of the causal model in Figure 3 have nine leaf nodes: player, *ambiance*, *perceived fairness*, *environment*, *NPC intelligence*, *avatar/NPC actions* and *avatar/NPC parameters*. In this study, the *ambiance* and *avatar/NPC actions* were kept constant, and we decided to measure rather than try to control *perceived fairness*, because we were unsure of its direct influences. This left us with five varying leaf nodes: player, *environment*, *NPC intelligence* and *avatar/NPC parameters*.

In addition to validating the edges in the top two layers of the causal model, a second goal of the study was to evaluate alternative instantiation candidates, both for leaf and other nodes in the third Slime Volleyball-specific layer of the model. For example, we wanted to determine whether both or either of jump height and lateral speed were valid direct influences on *NPC performance*, and whether *margin of victory*, *absolute margin of victory* or *duration* were valid *game metrics* in terms of their direct influence on fun.

Slime Volleyball needed to be heavily modified to meet the needs of the study. Among other things, we modified the game to be used by one player versus an AI-controlled *NPC*. Additionally, a logging utility was implemented to record the required data about a play session and store it in a useful format.

The game was made publicly available on a web page hosted on a WPI server. Potential participants were solicited by word of mouth and via various mailing lists and directed to the web page where, before participating, they had to read a brief introduction explaining the game and the data collection procedure (see appendix). After agreeing to participate, players were asked their age, gender and what type of gamer they considered themselves to be. The breakdown of participants is shown in Table 1.

Participants by Age		
	Frequency	Percent
under 20	30	22.1
20 to 25	95	69.9
26 to 30	1	.7
31 plus	10	7.3

Participants by Gender		
	Frequency	Percent
Female	46	33.8
Male	90	66.2

Participants by Gamer Type		
	Frequency	Percent
Casual	58	42.6
Average	35	25.7
Hard Core	27	19.9
Unknown	16	11.8

Table 1: Participants in Slime Volleyball Study

After answering the three questions about themselves, participants were directed to the game. Each participant was assigned one of 54 possible configurations of the

game (see below) according to a predetermined sequence based on order of login. After each match (five points), the participant was asked the following two questions:

- How much fun was this match? (0-9)
- How fair was this match? (0-9 where 0 means computer had advantage, 5 was an even match, and 9 means you had the advantage)

Participants were allowed to play as many matches as they wanted (cf. duration). The 54 possible game configurations were determined as follows:

$$
\begin{aligned}
54 &= AI \times GV \times JH \times LS \\
AI &= 3 \text{ AI Levels} \\
GV &= 2 \text{ Gravity Levels} \\
JH &= 3 \text{ Jump Height Difference Between Players} \\
LS &= 3 \text{ Lateral Speed Difference Between Players}
\end{aligned}
$$

To reduce the total number of configurations, we did not use all possible combinations of avatar and *NPC jump height* and *lateral speed* parameter values. Specifically, we only used the *avatar-NPC difference* combinations high-low, medium-medium and low-high for each parameter.

As the starting point of the analysis, we calculated correlation or partial correlation between every pair of nodes in the top two layers of the causal model. We used a simple correlation for node pairs with either a direct influence or no influence between them in the model. For nodes with a causal path between them, we used a partial correlation, holding the intermediate nodes constant. For each node with multiple candidate quantities in the third layer of the model, the appropriate

correlation was calculated using each candidate. In this way each candidate factor could be individually considered for inclusion into the final, instantiated model.

Then, to consider an edge (direct influence) in the causal model to be validated, we required that it account for at least 10 % of the variance (i.e., $R^2 \geq 0.1$, or $R \geq 0.32$). Given our sample size, this restriction is more stringent than $P \leq 0.05$, so all of the edges in our model are statistically reliable. To determine these values, tests were performed using the statistical software package SPSS.

3.1.2 Experimental Results and Analysis

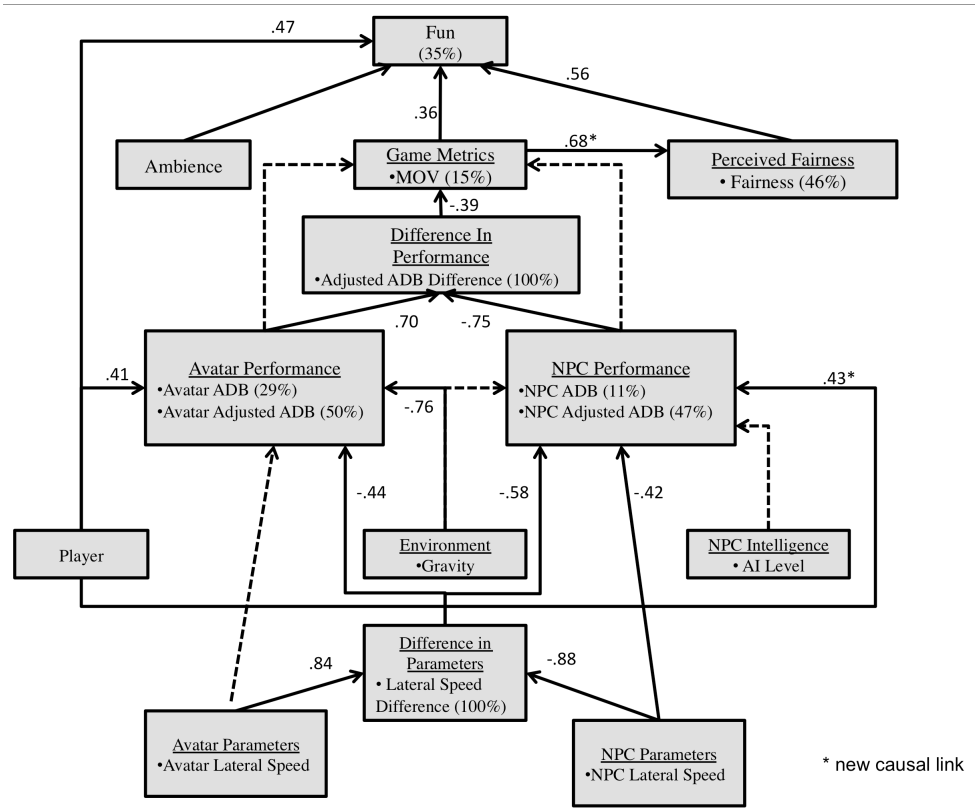

Figure 4: Validated Causal Model for Slime Volleyball

Figure 4 shows the result of applying the validation procedure described above. The solid edges in this figure (except for the two marked with an asterisk) are the

validated edges from Figure 3, with the R values indicated. For nodes with multiple instantiation candidates (*avatar/NPC performance*) the average correlation is reported. The five dotted edges in Figure 4 are influences in Figure 3 that were not supported by this study. The two edges marked with an asterisk are direct influences discovered in this study that were not in our original model. The direct influence of the *margin of victory* game metric on *perceived fairness* is a preliminary, and sensible, answer to our original uncertainty about the source of perceived fairness. The direct influence of the player on *NPC performance* makes sense in Slime Volleyball because a strong player will cause the *NPC* to make more mistakes. The percentages reported within some nodes represents the fraction of the overall variance accounted for by the current model (using ANOVA). To avoid over-fitting, we report the adjusted R^2 (a statistical adjustment to penalize for model complexity) value. For example, it appears that in Slime Volleyball, our model accounts for 35% of the variance in fun.

Finally, it is interesting to note which candidate factors in the Slime Volleyball-specific model turned out to be significant. For *game metrics*, only *margin of victory* was significant; for *perceived fairness*, only the *raw fairness* number; for *avatar/NPC performance*, only the *adjusted average distance to ball*; and for *avatar/NPC parameters*, only the *lateral speed*. These factors can be used to manipulate the level of enjoyment in the game.

3.2 Further Model Validation

While the results of the previous study generally supported the claims made in the causal model, more work was needed before concluding that the model is an effective representation of video game factor interactions. Questions remained about whether the model could be applied to all video games, or even all games in a single genre.

To answer the remaining questions, the work previously conducted using Slime Volleyball would need to be replicated using different video games. By developing a causal model for other games and validating it in a scientific study, we can be more confident in our claim that this model can be applied to most, if not all, video games.

We also sought to expand upon our previous research by applying the causal model dynamically during run-time. The model developed during the study can be used as a basis for designing a dynamic adjustment system that will make changes to the game at run-time. These changes will theoretically increase the amount of fun the player is having while playing. It is also desirable for the changes to prolong how long the player is enjoying the game and hopefully increase the amount of time they play for.

Many factors were considered when choosing the next game to apply the model to. Should the game be created for the study or should a previously developed open-source game be modified? What genre should the game be? How complex should the game be? These issues are expanded upon in the next sub-sections.

3.2.1 Constraints on Game Choice

The selected game would be required for the remainder of the thesis. Development of the dynamic difficulty adjustment system could not begin without an initial study that required the selected game. After determining a time frame for the rest of the project, it was decided that a pre-existing game was the only viable choice. In particular, the game needed to be open-source to allow for required changes to be implemented, such as a log to record informations needed during the analysis.

The complexity of the game was also a concern. Slime Volleyball is a very simple game. There are only two players. Each player only has three possible

actions (left,right,jump). Also, the only environmental factor is gravity. The next game would need to surpass Slime Volleyball in complexity. A more complex game would likely yield a robust model with a higher degree of interactions as well as a larger set of factors that could be manipulated during run-time.

On the other hand, a high complexity game introduces other difficulties. There was a limited set of participants (mostly WPI students) available for each study. If the game were too complex, some interesting factor interactions would not be adequately explored while others would never even be found. The search space of potential models for the game needed to be small enough so that it could be explored thoroughly during the study.

3.2.2 Selecting Appropriate Genre

In selecting the next game for analysis, all game genres [Gam10] were originally considered. Upon analysis of each genre, certain advantages and disadvantages of selecting a game from each emerged. Below is a list of the genres considered, including the advantages and disadvantages associated with them. While not an exhaustive list of genres, this list encompasses most of the major video game genres, as well as some of the important sub-genres.

- *First Person Shooter (FPS)*: In a First Person Shooter (FPS) game, the player's avatar is not in view. The camera is positioned in the first person perspective of the avatar; the player sees what the avatar sees. Characters in FPS games, both the *avatar* character and *NPC*, tend to have a robust set of factors that can be manipulated, including speed, rate of fire, and jump height. The physics of the game can be changed as well as the location and frequency of when enemies appear, or *spawn*. FPS games typically require strong reflexive skill since players are constantly firing at and avoiding fire

from opponents. Study participants without experience in this genre would likely not be able to gain the reflexes required in the time this study allows.

- *Platformer*: Games of this genre are characterized as having a complex terrain which the player must navigate through. An example of a platformer is Super Mario Brothers. Platformer games provide an large set of *environmental factors* which can be manipulated. The number of *NPC factors*, on the other hand, tends to be minimal when compared to other genres. Many platform games have almost no *NPC cognitive factors* at all, *NPCs* simply traverse the screen in predetermined patterns.

- *Racing*: The racing genre encompasses both games that attempt to be realistic and well as highly stylized. The goal of racing games is to navigate a predetermined course faster than your opponent. Depending on the level of realism, racing games will have a large set of environmental factors which can be manipulated, such as factors involving the course's surface condition. Racing games,in particular those that attempt to mimic real life, tend to be hard to control on a standard keyboard and mouse. They work best with a steering wheel peripheral. This peripheral could not be obtained for this study.

- *Role Playing Game (RPG)*: As the name implies, RPGs are games involve games where the player controls a single *avatar* or a small band of characters. Throughout the game, players are allowed to "level up", or customize their characters with new abilities, weapons, or powers. RPG games tend to progress very slowly, something that would not lend itself very well to this study.

- *Real-Time Strategy (RTS)*: In contrast with RPGs, RTS games usually have the player control a large number of avatars at any time. However, rather

than directly controlling the avatars, the player acts as an omnipotent being, directing the *avatars* to perform certain actions. Each *avatar* can have its own set of factors to manipulate. While this can be seen as a benefit, having such a large number of factors would significantly increase complexity of the causal model and any dynamic difficulty adjustment system derived from it.

- *Third Person shooter*: This genre of shooter games differs from FPS games in the player's perspective. Players view the *avatar* at some distance away from it. In this way, *NPCs* can be viewed from every angle around the *avatar*. These shooter games share the same advantages as FPS games without requiring strong reflexive skill. While top down and side scrolling shooters (two sub genres of third person shooters) do tend to have factors for the *avatar*, *NPC*, and *environment* that can be manipulated, the number of factors is usually lower than that of FPS games.

- *Fighting*: Fighting games typically involve two opponents battling each other until the other is incapacitated or disqualified by the game. There will be many factors referring to the *avatar* and *NPC* that can be manipulated. There is typically very little that can be done with factors pertaining to the environment. Fighting games tend to have a large "combo" system, where powerful moves can be performed by pressing a pattern of buttons within a specified interval. Study participants would not have the time to learn this combo system during the time frame of the study.

- *Puzzle*: Puzzle games consist of a set of cognitive challenges for the player. They tend to have neither an *avatar*, nor any *NPCs*. The *environments* in puzzle games usually cannot be altered without making the solution impossible or trivial.

3.2.3 Number of Human Players

A final consideration in choosing a game was the number of human players that would play during a game session. Slime Volleyball was a single player game; only a single human player was involved in the game at any time. Some games involve multiple human players interacting with each other. While it would be interesting to explore this subset of games, multiple human players introduce a level of uncertainty in the model. The human players would interact with each other during the game. Since human players can not be controlled directly, this could introduce a set of interactions that could not be manipulated by an internal system. These interactions would mask some of the causal relations we would be looking for in the study. The choice was therefore made to choose a game involving only one human player at a time.

3.2.4 Selected Game: Project Starfighter

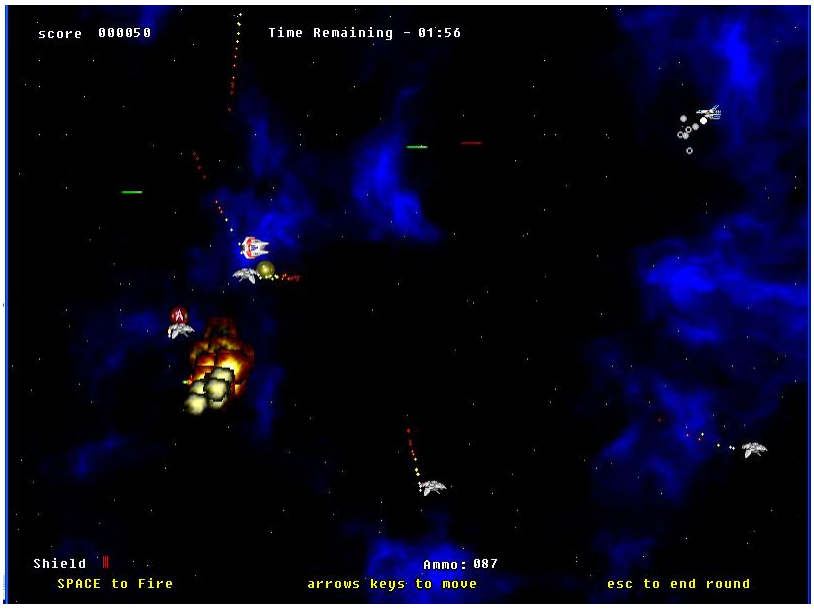

Figure 5: Screenshot from Project Starfigher

After considering all the factors discussed above, Project Starfighter was chosen as the next game to use to validate the causal model. Project Starfighter (see Figure 5), is a two-dimensional shooter game, where the player pilots a spaceship in space. The game is written in C++ by the company Parallel Realities, who released the source to the public. The objective of the game is to fight off a never-ending set of enemy ships. The piloted ship can move both vertically and horizontally. The ship can also fire its weapon. This weapon is limited in the amount of ammunition it has and can also be upgraded during the game. It is also a third person shooter, a genre that appears to have a large set of positive traits and few negative traits.

Participants by Age		
	Frequency	Percent
under 20	19	29.7
20 to 29	39	60.9
30 to 39	4	6.3
40 plus	2	3.1

Participants by Gender		
	Frequency	Percent
Female	15	23.4
Male	49	76.6

Participants by Gamer Type		
	Frequency	Percent
Casual	14	21.9
Average	31	48.4
Hard Core	18	28.1
Unknown	1	1.6

Table 2: Participants in Project Starfighter Study

A second study was performed in order to create a model for Project Starfighter. The design of this study was similar to the one used previously for Slime volleyball. The main difference was that Project Starfighter contained a unique set of factors to explore. Analysis of the results were performed in the same manner as the previous

study.

Participants in the study played six rounds of Project Starfighter. The first round lasts a minute and is used to let the player learn the controls. The five rounds that follow present the player with a random configuration of the game. Each round lasts for a maximum of five minutes. The player is allowed to end the round at any time. Round time can thus be recorded and analyzed to determine the factors that effect it, as well as the relationship that exists between this value and fun. It is theorized that there will be a strong positive relation between persistence (round time) and fun. At the end of each round, players are asked how much fun and how fair they believed the game to be (see appendix for survey questions).

3.2.5 Validation Study for Project Starfighter

The model created in the study is shown in Figure 6. The solid lines represent causal links that are supported by the model. The dashed lines represent possible causal links that were not supported by the model. The factors between these links were found to be highly correlated, even when partialing out other factors in the model.

Seven factors were found that could be directly manipulated to effect the amount of fun in the model. Changing the values of these factors will change the values of the *performance* and *game metrics*, which then in turn will effect the perceived fairness as well as the amount of fun in the game. These factors are the values that will be utilized at run-time to make adjustments to the game. The factors found to have a manipulable effect on fun are:

- Avatar Weapon: The weapon that the player's character is using. Three weapon types are used in the study.
  - Single Shot: Shoots a small laser directly in front of the avatar ship.

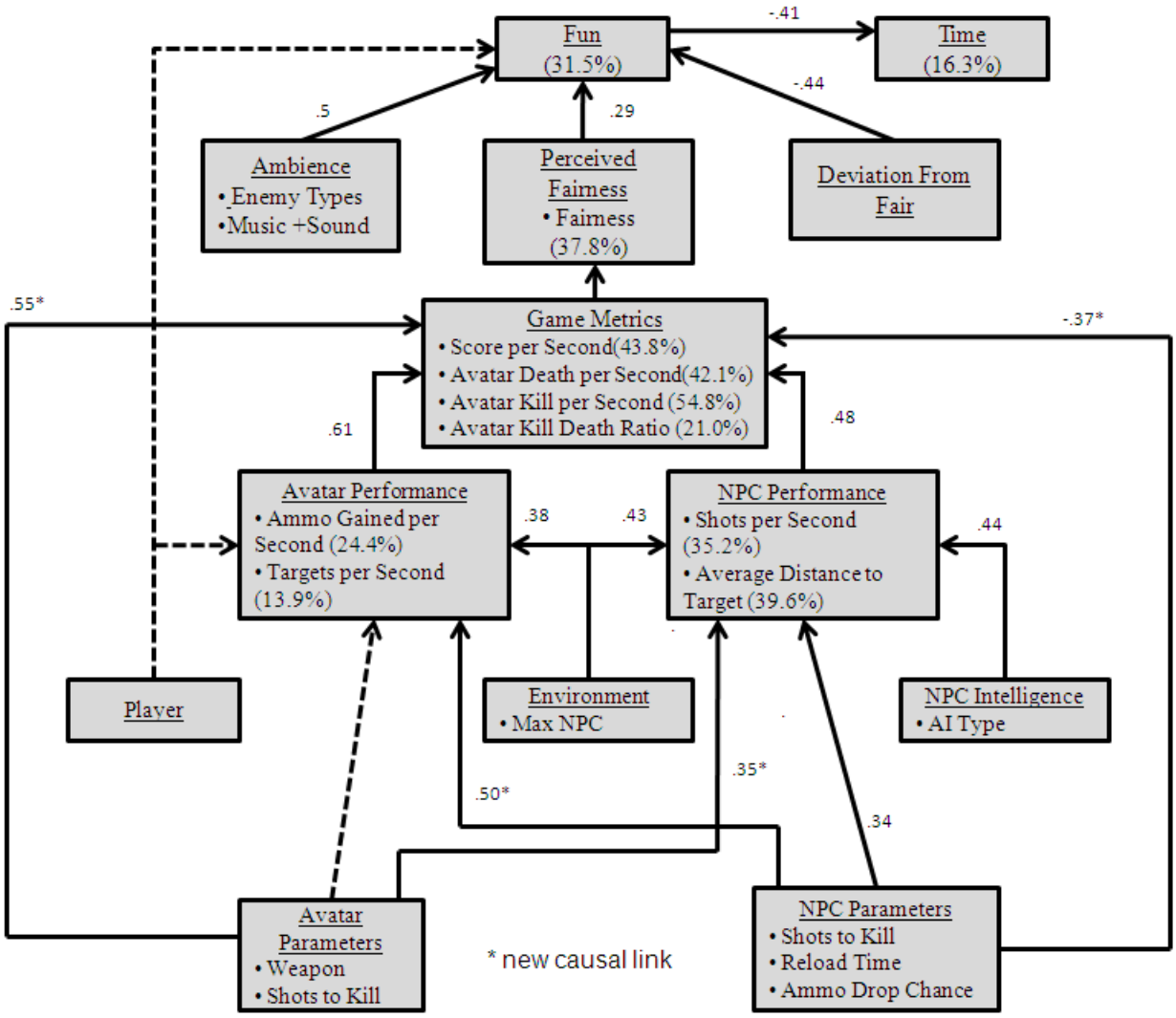

Figure 6: Validated Causal Model for Project Starfighter

  - Triple Shot: Shoots the same laser as the single shot, except that there are now three of them. One shoots directly in front of the player, while the others shoot at an angle above and below the avatar ship.
  - Homing Missile: When shot, this missile automatically finds an NPC ship and adjusts its trajectory to guarantee a hit.

- NPC Artificial Intelligence(AI): The controlling logic for the NPC ships. Three different AIs are used in the study.

  - *Offensive*: NPCs with this AI tend to continually attack the avatar ship. Their trajectory is adjusted so that they remain within a small distance of the avatar ship at all time.
  - *Evasive*: The opposite logic to an offensive AI, NPCs with this AI will attempt to keep a large distance from them and the avatar ship. These NPCs will occasionally attempt to attack the avatar ship, though usually it is up to the avatar to seek out these NPCs.
  - *Defensive*: A compromise between the offensive and evasive AI, the Defensive AI will attempt to navigate the NPC ship it is controlling so that a small distance is kept from the avatar ship. This distance is much smaller than the one used by the evasive AI. The defensive AI will also attempt more frequent attacks on the avatar than the evasive AI.

- *Avatar Shots To Kill*: The number of hits an avatar ship can withstand before it is destroyed.
- *NPC shots To Kill*: The number of hits an NPC ship can withstand before it is destroyed.
- *Maximum NPC On Screen*: The number of NPC ships that can be spawned at any time. NPC ships will always spawn until this number is reached. No new NPC ships will be spawned after this number is reached until one is destroyed.
- *NPC Ammo Drop Chance*: When an NPC is destroyed, there is a chance that it will drop ammo in space that the Avatar can then collect. This factor controls the probability that this occurs.
- *NPC Reload Rate*: Controls how fast an NPC can fire its weapon.

While *ambiance*, *perceived fairness*, *deviation from perceived fairness*, and *round time* all had an effect on fun, no *game metric* had a direct influence on fun. *Round time* had a positive influence on fun. A player that played a round for longer viewed the game as being more fun. Players also enjoyed when there were sounds playing and when the enemy ships varied in size and color (values measured in the *ambiance* node). *Perceived fairness* had a positive effect on fun while *deviation from perceived fairness* had a negative effect. All the influence on fun from the *game metrics* is directed through *perceived fairness*. No *game metrics* had a direct influence on the *deviation from perceived fairness*.

Four *game metrics* had a direct effect on *perceived fairness*. *Avatar deaths per second* had a negative effect on perceived fairness, while the rest had a positive effect. The negative effect of *avatar deaths per second* is not surprising since a player may perceive a game where they are constantly dying as too difficult and thus unfair. The converse explanation can be used for the other three metrics.

There is a complex relationship between the *game metrics* and the four *performance factors* that were found to have a strong causal effect on *game metrics*. The manipulable factors *NPC shots to kill* and *avatar shots to kill* had direct effects on certain *game metrics*, even when controlling for the contributing effects of the *performance factors*.

Some causal links were found that conflicted with the causal model. The study ,however, supports a majority of the assumptions the causal model makes. While the causal model has not been proven to apply to all video games, its value has been shown to cross into multiple game genres.

Both studies, in general, supported the claims made by the causal model. Also, both models supported placing a causal link between *game metrics* and *perceived fairness*. Other causal links were not as well supported. The causal link between

player and fun was only found in the Slime Volleyball Study. *NPC Intelligence* was found to have an effect on *NPC performance* in the Project Starfighter study. This relationship is not found in the Slime Volleyball study. The number of new causal links (ones not expected in the model) was larger in the Project Starfighter study. This is likely due to the increased complexity of the game when compared with Slime Vollyeball.

4 Applying the Causal Model

Utilizing the causal model developed for Project Starfighter, the design and implementation of a dynamic adjustment system is possible. The goal of this system is to utilize the knowledge obtained from the model to modify factors in a way that will positively influence fun.

4.1 Designing the Dynamic Difficulty Adjustment System

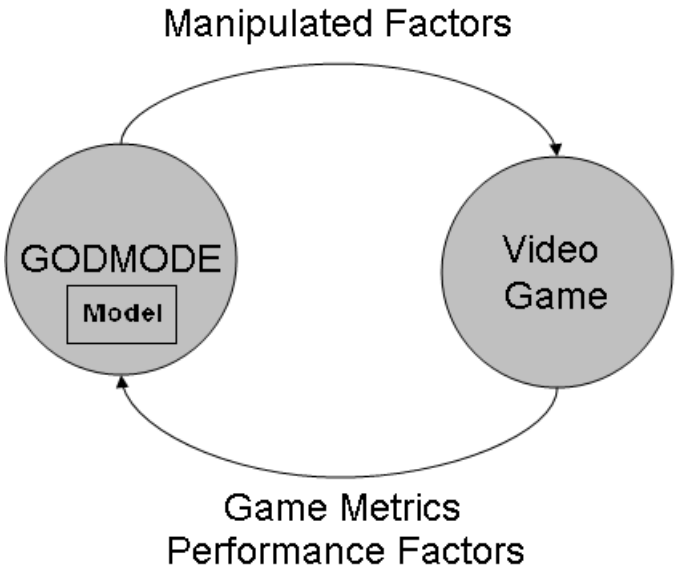

Figure 7: Flow of Data Between GODMODE and a Video Game

The resulting Dynamic Difficulty Adjustment system and Application Programming Interface (API) developed is known as GODMODE (Games Observed Dynamically to Manipulate Observed Difficulty Errors). The API is structured so it can be placed into an existing game with minimal adjustment to the game's code base. A programmer simply needs to make calls to the three functions defined in the API (see below). Using the API, the programmer provides GODMODE with an initial model, read access to the *game metrics* and *performance factors*, and write access to the *manipulated factors*. Figure 7 illustrates how data flows between GODMODE and a video game. The programmer can then ask the API to update the model or determine new values for the *manipulated factors*. The requested changes are then performed and incorporated into the game automatically.

4.1.1 Converting Causal Models to Linear Regression Equations

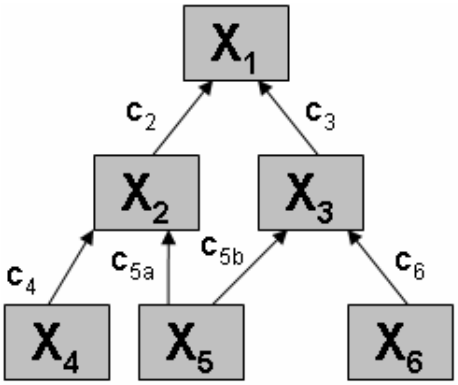

Figure 8: Example Causal Model

To facilitate customization and optimization (discussed below) in the causal model, each node in the model was converted into a linear regression equation. A linear regression equation contains a set of variables, each with a coefficient expressing the influence a variable has on the equation. Each equation also has a constant associated with it. This value is used to capture the influence of all possible factors

not included in the equation. These equations can be used separately to determine an estimate for any node in the model. They can also be collapsed to generate a equation for fun in terms on *manipulated factors* only.

Figure 8 is an example of a causal model. The factors X_{4-6} are *manipulated factors*. In collapsing the model, the goal is to build an equation for X_1 in relation to these *manipulated factors*. Below are the steps to collapse the given model.

$$
\begin{aligned}
X_1 &= C_2 * X_2 + C_3 * X_3 + E_1 \\
X_2 &= C_4 * X_4 + C_{5a} * X_5 + E_2 \\
X_3 &= C_{5b} * X_5 + C_6 * X_6 + E_3 \\
&\quad Therefore \\
X_1 &= C_2 * C_4 * X_4 + (C_2 * C_{5a} + C_3 * C_{5b}) * X_5 \\
&+ C_3 * C_6 * X_6 + C_2 * E_2 + C_3 * E_3 + E_1 \\
&\quad Where \\
C_i &= \text{Regression Coefficient} \\
E &= \text{Regression Constant}
\end{aligned}
$$

This process can be expanded to a causal model of any size, yielding a single equation where all the variables are *manipulated factors*.

The model built in the previous Project Starfigher study reflects the collective input from every participant. This level of contribution was required to validate causal links in the model. The strengths of each link, however, may vary between groups of players, or between every player. A level of specialization was needed to adapt the model to different players.

The simplest approach is to introduce stereotypes into the model [Ric79]. In this approach, groups that have varying models (different set of causal links) are identified. For Project Starfighter, potential stereotypes are the player's gender, age, and gamer type.

The main drawback with this approach is that each model needs to be generated and validated using separate data sets. A small set of participants were available for the study. Every participant would contribute exclusively to a single model in the set. The strength of each model would be drastically reduced compared to generating a single model using all participants.

Even if more participants could be obtained for a study, an even distribution could not be guaranteed for each model. In the Slime Volleyball study, for example, there were approximately twice as many males as there were females. A model built using only male participants could be created utilizing more data than the female model. The validity of the male causal model would be greater than the female causal model. The expected result would be a system that adjusted the game better for males than females.

A second approach is to build one model utilizing the data from all participants in the study. This model is given to GODMODE and is used initially to adjust the game during run-time. At specific intervals, the strength of each causal link in the model is evaluated. Based on the current values of the nodes on either end of the causal link, the coefficients in one of the linear regression equations representing a causal link are updated. The strength of each causal link can be adjusted based on the current game state, yielding a model that has been customized for a specific player. This is the approach that GODMODE takes utilizing the delta rule. This rule seeks to update the value for all the coefficients in the linear regression equations. The model is updated by calling the updateModel() function from the API.

Linear regression equations are stored for every game metric and performance factor in the model. Using these equations, any of these factors can be predicted. GODMODE also has read access to the current *actual* value for each *game metric* and *performance factor*. These two values can be used to determine the current error ϵ for each *performance factor* and *game metric* in the model.

$$\begin{aligned} Y_{Predicted} &= C_1 * X_1 + C_2 * X_2 + C_3 * X_3 + E \\ Y_{Actual} &= \text{Current Value From Game} \\ \epsilon &= Y_{Actual} - Y_{Predicted} \end{aligned}$$

With an error term for each regression equation in the model, the coefficients for each equation can be updated. This is accomplished with the following equation:

$$C_i = C_i + \delta_i \epsilon X_i$$

Here, each coefficient is effected by both the error of the equation and the current value of its corresponding independent variable. Manipulating the coefficients this way allows for them to be weighted by the independent variable they correspond to. If the independent variable is small, it is contributing little to the error. The corresponding coefficient should not change much. If the independent variable is large, it has a significant effect on the equation. The coefficient corresponding to this variable should be manipulated accordingly.

The change in each coefficient is also effected by the δ value. This value (usually very small) is used to control the amount each coefficient can change during a single

update. Multiplying by this value dampens the effect of both the equation error and the value of the independent variable. Without the δ value, the changes to the coefficients may be too large, causing the resulting equation to overcompensate for the error. The equation will never never converge, it will instead "thrash" around, alternatively increasing and decreasing the value of each coefficient.

4.1.2 Manipulating Factors

From a programmer's point of view, the only step in generating new *manipulated factor* values during run-time is to call the updateManipulatedFactors() function from the GODMODE API. Invoking this function will result in the *manipulated factors* given to GODMODE to be updated with values that maximize the model's estimate for fun.

Determining a mechanism to perform updates of the manipulated factors was a key decision in designing GODMODE. A search of similar research yielded the Simplex algorithm, a linear programming technique [Dey09]. This algorithm is guaranteed to find the optimal minimization or maximization of a equation consisting of a set of variables. Furthermore, inequalities can be introduced to restrict the range of values that each variable can take. Consider the following example:

$$\begin{aligned} Y &= C_1 * X_1 + C_2 * X_2 \\ X_1 &\leq R_1 \\ X_2 &\leq R_2 \end{aligned}$$

Here, R_i represents the maximum value that X_i can take. Since all changes Simplex performs are positive, the effective range of values for X_i is 0 - R_i inclusive. The value of R_i represents a step wise restriction imposed by the causal model (i.e.

$R_i = X_i \pm maxchange$). This value promotes small, incremental changes to the game state. Also, the equation for Y can be represented by any of the linear regression equations in the model.

Simplex will generate an optimal answer in a time linear to the number of variables if all variables are continuous. With discrete variables the problem, known as mixed-integer programming, becomes NP-Hard. This is a major issue since solutions are required during the run-time of the game. With computation time needed for both graphics and game logic, few resources can be allocated to GODMODE to perform the desired changes. This implementation is not suitable for equations with more than a few variables as the computation time would grow exponentially.

The other difficulty with the simplex algorithm is that all changes made to variables are positive. By adjusting the sign of the equation, changes can be made to be positive or negative. Both possibilities, however, are never considered in parallel. This restricts the search space of *manipulated factor* values that simplex explores, essentially cutting it in half.

To overcome the efficiency issues with mixed-integer programming, we used hill climbing for problems with integer variables. Hill climbing is a local search algorithm that incrementally finds an optimal solution. For the purposes of GODMODE, hill climbing begins by collapsing the model into a single equation for fun in terms of *manipulated factors*. Hill climbing looks at each of these factors one at a time, increasing or decreasing the current value by one. The change that minimizes the deviation from the desired value for fun is chosen. This process iterates until no change produces a smaller deviation on fun. The maximum change for each variable is controlled by the same step-wise restriction used for Simplex.

Hill climbing addresses both problems associated with using the simplex algorithm. First, hill climbing considers both negative and positive changes for each ma-

nipulated factor. Second, little computation is required to evaluate each state. While hill climbing is not guaranteed to run in polynomial time, the time required is significantly smaller than using simplex due to the smaller set of operations performed on each step. While the simplex algorithm performs numerous matrix operations on each step, hill climbing only performs a small set of additions and subtractions to find a suitable solution. Also, the algorithm generally finds a local optimal solution in a small number of iterations.

$$
\begin{aligned}
Y &= C_1 * X_1 + C_2 * X_2 + C_3 * X_3 \\
Step(X_1) &= 2 \\
Step(X_2) &= 2 \\
Step(X_3) &= 2
\end{aligned}
$$

To illustrate, an equation for the variable Y is given that is composed of three *manipulated factors*. Each factor can increase or decrease by a maximum of 2 (as long as the change is still in the range defined for that factor). For each iteration using this equation, seven equations are considered. The first is the equation as it is, with no factors adjusted. Two equations are considered for each factor, one with the factor increased by one, the other with the factor decreased by one, holding the rest of the factors constant. The equation with the lowest deviation from the desired value of Y is chosen and the factors are set to the corresponding values. This process is repeated until no improvement in deviation from the desired value for Y can be found.

The main drawback of using hill climbing is that it is prone to becoming stuck in local optimal solutions, instead of finding the global optimal solution. This is a minor

issue since the hill climbing algorithm is invoked from GODMODE multiple times during the game. Between invocations, the model will be updated, changing the influence of each *manipulated factor*. These changes can affect the value for both the global and local optimal solutions. Adjustments to the model between invocations of the hill climbing algorithm allow it to get out of local optimal solutions.

4.1.3 Initializing the Model

GODMODE requires two pieces of information before it will be able to perform the desired changes to a game during run-time. First, a causal model must be given to the GODMODE. Read access to the *game metrics* and *performance factors*, as well as write access to the *manipulated factors*, must also be permitted. Both steps are performed by calling the instantiateModel() functions from the API. This functions should only be called once during a game to set up the causal model.

The file given to GODMODE must be of a specific format (see appendix for example input file). The file contains the coefficients for the linear regression equations built for fun, *perceived fairness*, *game metrics*, and *performance factors*. To allow the model to be as general as possible, a coefficient for each factor in the lower levels of the model is included. For example, the equation for *game metrics* has a coefficient for each *performance factor* as well as each *manipulated factor*. The resulting set of equations is sparse, contain many zero coefficients. A zero coefficient corresponds to a factor that does not have a direct effect on the dependent variable. GODMODE removes factors with zero coefficients when performing operations on the model.

The file also contains strings describing the variables each coefficient represents. The descriptive strings are included to ensure coefficients are entered in the appropriate order. These strings are ignored when GODMODE reads the file.

As well as including the coefficients for all equations in the model, the given file must also include a set of restrictions for each of the *manipulated factors*. For each *manipulated factor*, an initial, maximum, and minimum value is specified. The maximum and minimum values specify a range of valid values for each *manipulated factor*. The initial value is the value that each *manipulated factor* will be set to when the initializeModel() function in invoked. Also included is a step-wise constraint on each manipulated factor. This value specifies a maximum absolute value that each factor can change during a single update. This promotes incremental changes that will hopefully be invisible to the player.

Nominal factors can also be encoded into the model. A nominal factor (such as *avatar weapon*) will include a variable for each possible value that it can take. The minimum value for each variable must be zero, while the maximum must be one. Also, the step-wise constraint must be one. In this way, variables representing a nominal factor can either be on (one) or off (zero). Only one variable for each nominal factor should initially be set to one.

GODMODE requires read access to the *game metrics* and *performance factors*. It also requires write access to the *manipulated factors*. *Game metrics* and *performance factors* are used to update the model. The *manipulated factors* are used to dynamically make changes to the game. By storing the location all these variables, GODMODE is able to operate with minimal interaction with the game. Changes made to *game metrics* and *performance factors* are passively incorporated into the model. Changes to the *manipulated factors* are likewise incorporated into the game passively.

Currently, all factors in the model are required to be continuous. This choice was made so that both discrete and continuous factors could be incorporated. The game is required to facilitate the conversion of a continuous value if needed. For

example, Project Starfighter stores all of its *manipulated factors* as discrete values. Write access to a set of matching continuous factors are given to GODMODE. When these factors are updated, the game converts the continuous values it was given into discrete values.

4.2 Applying GODMODE to Project Starfighter

When Project Starfighter is first run, it creates three sets containing the *game metrics*, *performance factors*, and the *manipulated factors* respectively. GODMODE is given the appropriate read/write access to these sets using the initializeModel() function from the API. Also passed to this function is the location of the file containing the model. With this information, GODMODE has everything it needs from Project Starfighter to create a model.

We decided that updates to the model and *manipulated factors* should occur in predetermined intervals. Controlling when *manipulated factors* are updated prevents changes from occurring rapidly and being noticed by the player. Model updates are controlled to allow for a consistent amount of learning to occur throughout the game.

Project Starfighter updates the causal model by calling the updateModel() method once every second. This time interval was chosen since one second is the smallest interval that guarantees that every *game metric* and *performance factor* value would be updated at least once. The updateManipulatedFactors() method is called every fifteen seconds. This interval was chosen through a small series of experiments where the game was played using different intervals of updateManipulatedFactors() invocations. The intervals were qualitatively evaluated based on the ability of the model to adjust to the player without the changes being obvious.

When updateManipulatedFactors() is invoked, some changes to the *manipulated factors* do not occur instantaneously. This is done in an attempt to keep the ad-

justments transparent to the player. The logic that controls when changes occur is located in the game code. If a new weapon is selected for the player, a power-up item is created the next time an enemy ship is destroyed. When the player touches this power-up, their current weapon is replaced with the one the power-up represented. These power-ups lasts fifteen seconds to coincide with the *manipulated factor* update interval. Also, updates to the number of shots it takes to kill the player's *avatar* occur only after the player dies.

Changes to the *NPC* characters only occur if the adjustment would make the *NPC* easier to kill. This is performed in an effort to prevent the scenario where the *NPC* characters have become too powerful for the player to kill. In this scenario, adjustments to the *NPC* characters will instantly decrease their difficulty and allow the player to break out of this sub-optimal scenario. Adjustments that would increase the difficulty of the *NPC* do not occur until a new NPC is spawned. If the *NPC* is easy to kill, it will likely need to respawn quickly, allowing the changes to occur.

5 Experimental Evaluation of DDA

5.1 Experimental Design

To test the effectiveness of the GODMODE, We conducted a second study with Project Starfighter. Like the initial Project Starfighter study, participants start by playing a one minute practice round of the game. They then play five rounds that last a maximum of five minutes. Participants are allowed to end the round for any reason at any time. At the end of each round, the user is asked how fun and fair they believed the round to be. Demographic information on the participants age, gender, and gamer type are also recorded.

For this study, two versions of Project Starfighter were created. The first version of the game acted as the control for the study. Using the equations previously found for fun in Project Starfighter, a set-up was found that predicted the highest possible value for fun. The values for the *manipulated factors* were set to the values below:

$$\begin{aligned} \text{Avatar Weapon} &= \text{Homing Rockets} \\ \text{NPC AI} &= \text{Offensive} \\ \text{NPC Shots To Kill} &= 1 \\ \text{Avatar Shots To Kill} &= 7 \\ \text{Maximum NPC On Screen} &= 8 \\ \text{NPC Ammo Drop Chance} &= 50 \\ \text{NPC Reload Time} &= 5 \end{aligned}$$

The control version of the game set the *manipulated factors* to these values. The values used for each *manipulated factor* remain constant during the entire time a player is playing. It is theorized that participants playing this version will quickly grow tired of this static set-up. They will quickly become experienced with the game and their perception of how fair the game is will increase towards the player having a large advantage. As a result, both fun and *round time* should decrease.

The second version of the game begins identically to the first version. Participants in this experimental group begin with the same static set-up in the practice round. After the practice round, however, the game utilizes GODMODE by periodically invoking the updateManipulatedFactors() and updateModel() functions from the API. This allows changes in the *manipulated factors* to take place during run time. Ideally, these changes will adapt to a player as they gain experience with the game. The participants *perceived fairness* should remain close to the ideal value

throughout the entire play session. We believe that both fun and *round time* should be significantly higher when compared to the control group.

We hypothesize that participants in the experimental group will play for longer and believe the game to be more fun when compared with participants in the control group. We also believe that the participants in the experimental group will perceive the game to be fair. Participants in the control group, by comparison, will view the game to be too easy, especially in the later rounds.

Participants by Age					
Control Group			Experimental Group		
	Frequency	Percent		Frequency	Percent
under 20	0	0	under 20	1	3.6
20 to 29	26	96.3	20 to 29	26	92.8
30 to 39	0	0	30 to 39	1	3.6
40 plus	1	3.7	40 plus	0	0

Participants by Gender					
Control Group			Experimental Group		
	Frequency	Percent		Frequency	Percent
Female	2	7.4	Female	4	14.3
Male	25	92.6	Male	24	85.7

Participants by Gamer Type					
Control			Experiment		
	Frequency	Percent		Frequency	Percent
Casual	2	7.4	Casual	2	7.1
Average	19	70.4	Average	17	60.7
Hard Core	6	22.2	Hard Core	9	32.2

Table 3: Participants in GODMODE. Study

5.2 Results

Figure 9 displays the distribution of participant responses as it pertains to perceived fairness. As hypothesized, a majority of responses from the control group believed

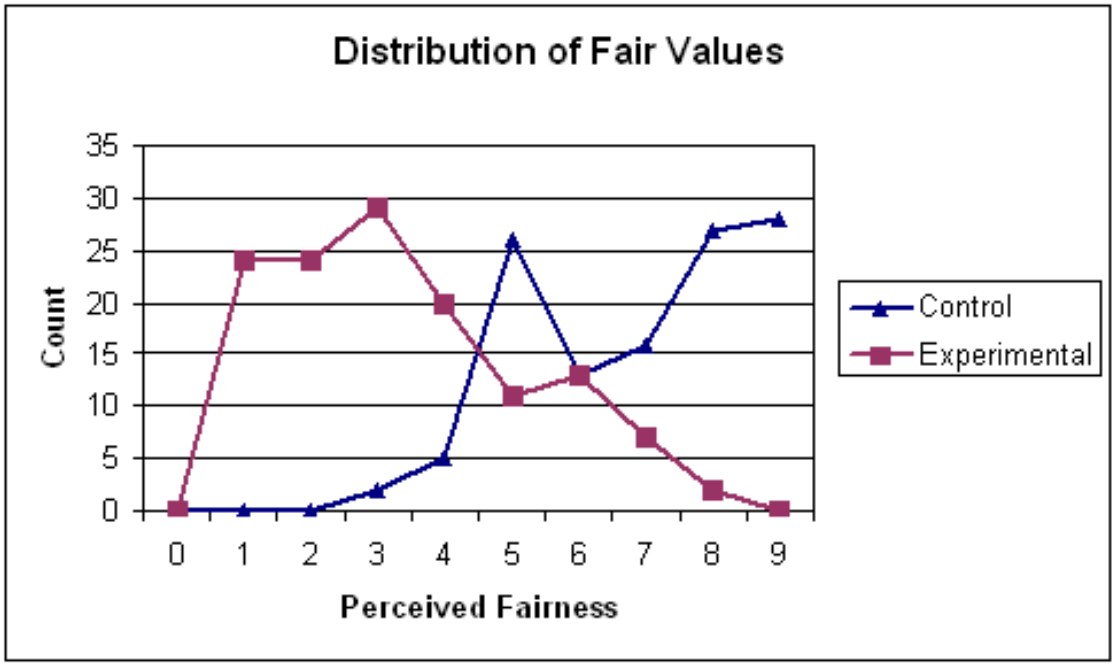

Figure 9: Distribution of Perceived Fairness by Group

that they had an advantage during the game (i.e. response higher than five). Surprisingly, the experimental group consistently believed they were at a disadvantage (i.e. response lower than a five) when playing the game. The hypothesis was that the distribution of this variable would be approximately normal with an average value of five.

Figures 10 and 11 summarize the impact *perceived fairness* had on fun and *round time* for each group. The combined plots of both groups in Figure 10 support previous results of an association between fun and *deviation from fair*. As the value for *perceived fairness* deviates from the middle values, the value of fun rapidly decreases. It should be noted that this association is not centered on the expected middle value of five.

An initial analysis of the data yielded conflicting results with some of our hypothesis. Participants in the control group consistently rated the game as being

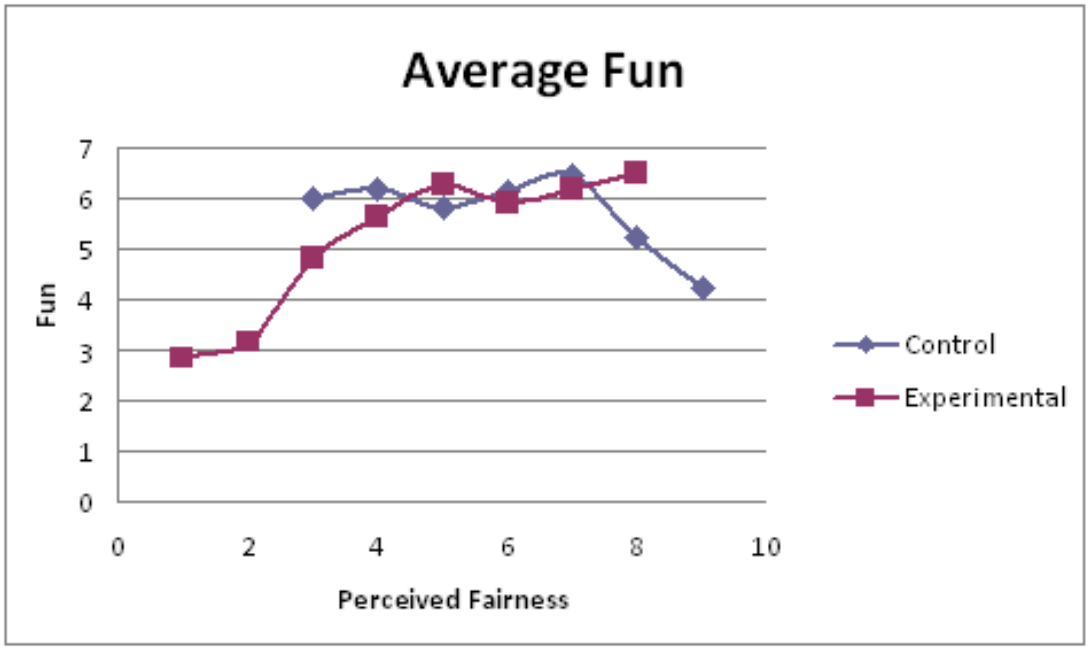

Figure 10: Average Fun for Various Perceived Fairness Values by Group

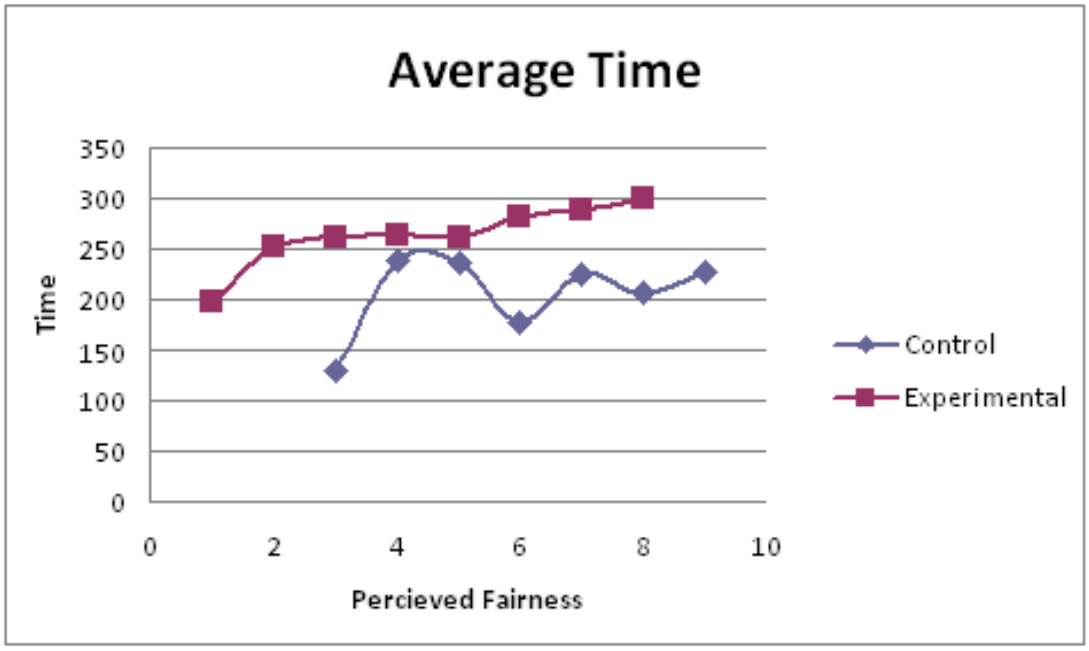

Figure 11: Average Round Time for Various Perceived Fairness Values by Group

more fun when compared to the experimental group. Participants in the experimental group, however, consistently played for a significantly longer amount of time

Analysis of Average Values				
Group	Time	Fun	Perceived Fairness	Deviation From Fair Game
Control	218.76	5.53	6.96	2.10
Experimental	253.26	4.62	3.36	2.15

Table 4: Average Values of GODMODE. Study by Group

per round. By the final round, participants in the experimental group were playing for an entire minute longer than in the control group.

There is also a significant difference in the *perceived fairness* value in each group. The deviation from the ideal *perceived fairness* value of five is insignificant. Both groups vary the same amount from the desired fair game, just in different directions. The control group yielded a game that was too easy. The adjustments made in the experimental group overcompensated for the performance of the player, creating a game that was consistently too hard.

Looking at the demographic data listed in Table 3, some discrepancies between groups emerge. These differences may have had an impact on the results of the study. The larger percentage of hardcore gamer type participants in the experimental group may have negatively affected the average value for fun. It may be the case that this type of gamer would not have been satisfied with any changes made to the game.

To compensate for these difference, values were calculated for each factor using the marginal means. Marginal mean is a statistical value that attempts to compensate for differences in groups. In theory, the marginal mean would be the actual average value for a factor if the distribution of participants in each group was identical.

The effect of analyzing fun and *round time* using marginal means can be seen in Table 5. The impact of version on the amount of fun a player has becomes very insignificant ($p >> .05$). While version's effect on *round time* also loses its

Significance of Group Using Marginal Means			
Factor	Mean Square	F-Value	Significance (P-Value)
Fun	12.224	1.491	.310
Play Time	61457.649	6.886	.054

Table 5: Significance of Version Using Marginal Means on Fun and Round Time

significance (based on a 95% confidence level), the change is much smaller when compared to fun. There is a higher probability that the positive effect on *round time* produced by the experimental group was not a chance occurence when compared with the effect the control group had on fun. In other words, it is very likely that the experimental version of the game did indeed have a positive effect on *round time*. It is less likely that the control version of the game had an equivalent impact on the fun a player had.

5.2.1 Deriving Equation of Fun from Study Data

Both the control and experimental group's *perceived fairness* vary from the desired value by about the same amount. In theory, this equal deviation should yield the same value for fun in each group. The control group, however, reported having a larger amount of fun.

Using the data obtained from the initial Project Starfighter study, an equation was derived using linear regression to calculate fun from *perceived fairness* as well as the *deviation from fair*. The formula calculated is given below:

$$Fun = .31 * Perceived\ Fairness - .582 * Deviation\ From\ Fair + 4.801$$

The dominating term in this equation is *Deviation from Fair*. As suspected, fun is maximized at the expected *perceived fairness* value of five. Since *perceived fairness* has an effect on fun, the equation is not symmetrical about the maximum value. The

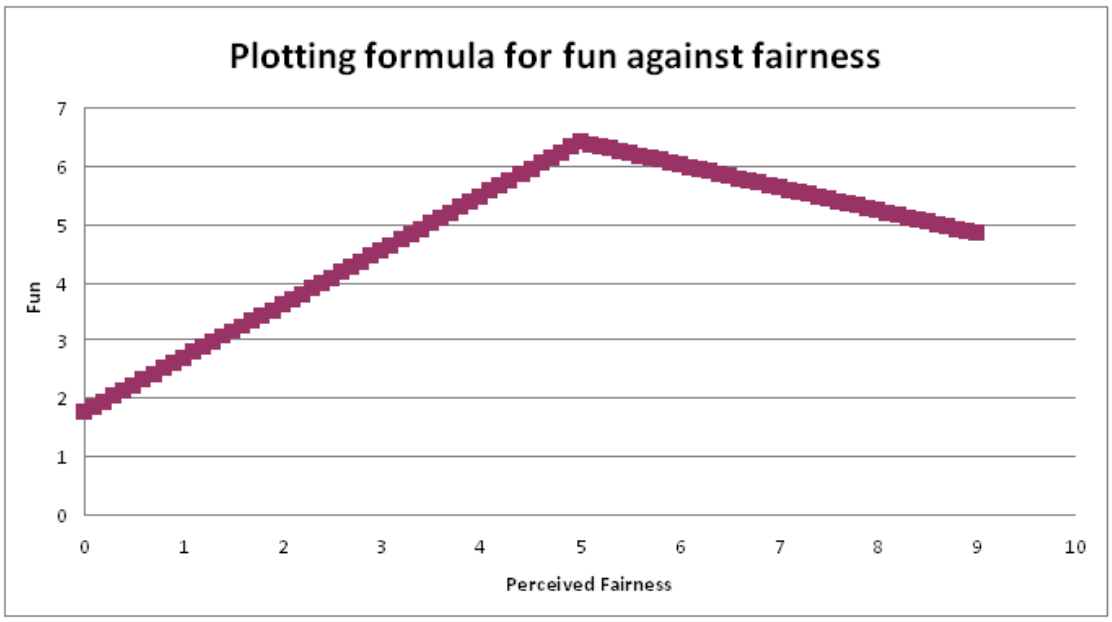

Figure 12: Plotting Fun against Fair Using Study Data

rate at which fun decreases is slower for values larger than five when compared to the rate for values smaller than five. This can be seen in Figure 12. The difference in values for fun between groups makes sense since each is on one side of the equation.

5.2.2 Analysis of Results by Gamer Type

The demographic value with the largest variation between the two groups was gamer type. The data was split by differing gamer type so that differences between the groups could analyzed.

Figure 13 and Figure 14 summarize how fun and *round time* differed by gamer type. Surprisingly, the hard core gamers did not report a difference in fun or *round time* between the control and experimental group. This is especially surprising since this gamer type had the largest difference in the *perceived fairness* between the control and experimental set-ups (see Figure 15).

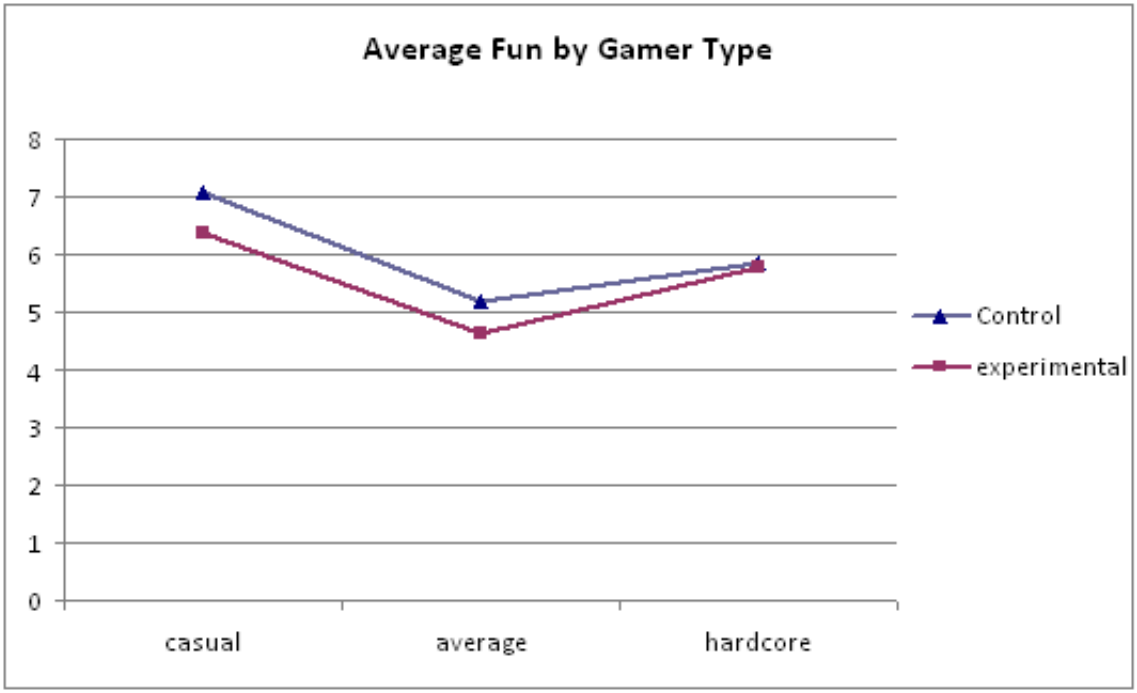

Figure 13: Comparing Effect on Fun by Game Type

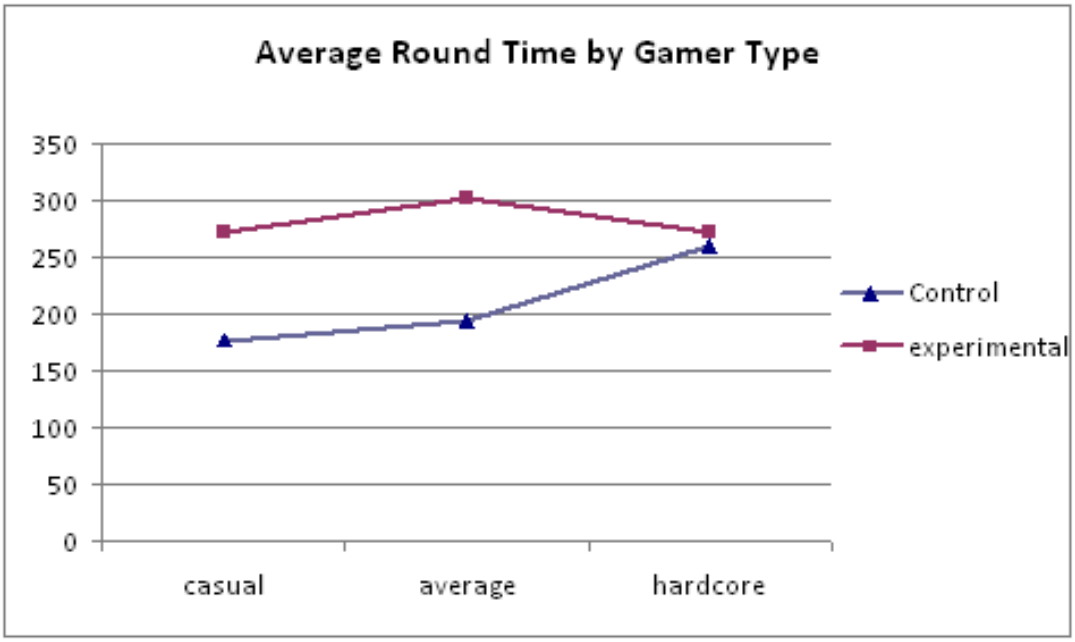

Figure 14: Comparing Effect on Round Time by Game Type

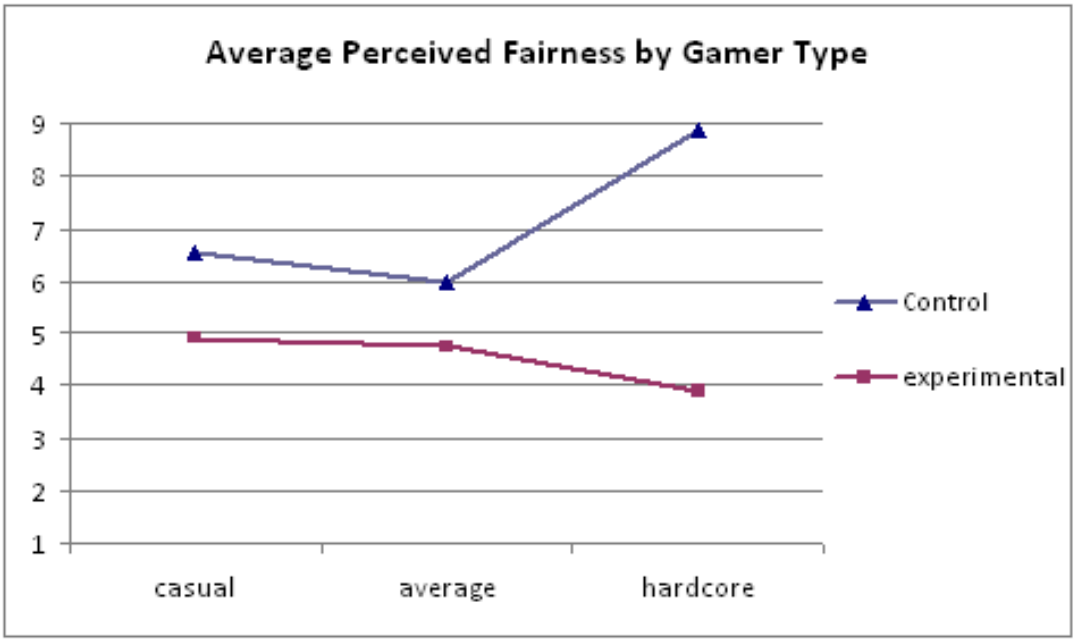

Figure 15: Comparing Effect on Perceived Fairness by Game Type

5.2.3 Analysis of Results by Round

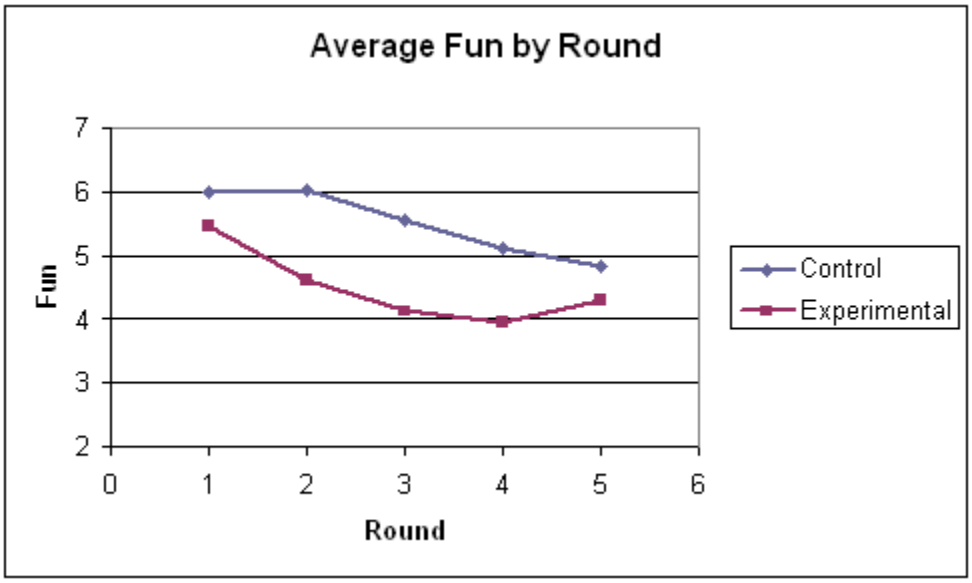

Figure 16: Comparing Effect on Fun by Round

Finally, the data was analyzed by round. The experimental group played for

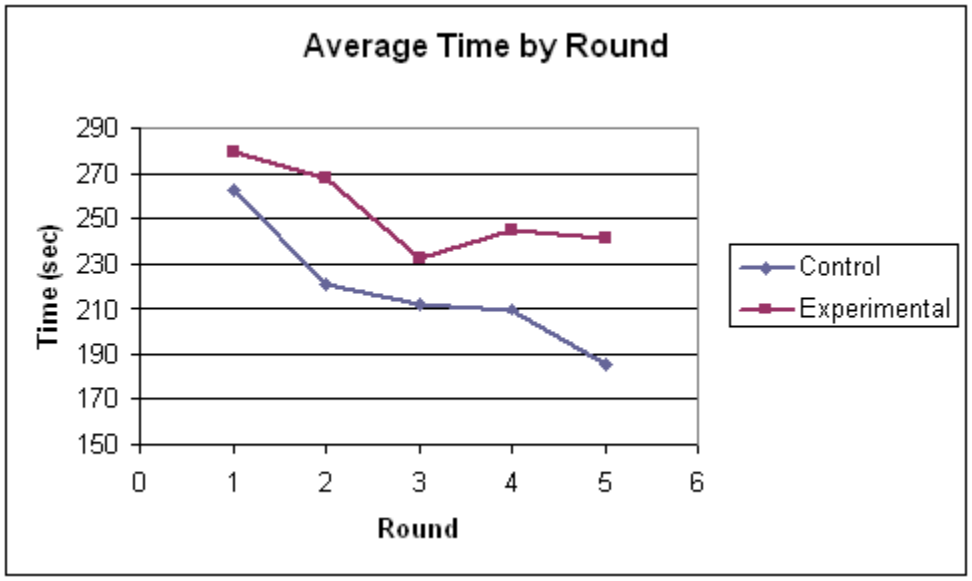

Figure 17: Comparing Effect on Time by Round

a longer time during each round when compared to the experimental group. This change increases in later rounds. The amount of time played in the experimental group is approximately an extra minute when compared to the control group in the last round.

In the first round, both groups reported the same value for fun. The control group reported a significantly larger value for fun for rounds two through four compared with the experimental group. A downward trend for the control group can be seen in Figure 16. In contrast to the control group, the reported value for fun, although lower than the control group, begins to level out and even begins to trend upwards in the later rounds. By the final round, there is little distinction between the two groups. If the experiment had been extended to more rounds, the experimental group may have overtaken the control group in the reported value for fun.

Figure 18 displays the average value for *perceived fairness* in each round of the game. The control group's *perceived fairness* remained relatively consistent over all four rounds. The value varies by about half a point. Between the first two rounds of

the experimental group, the value for *perceived fairness* drops significantly. After the first round, however, the value for *perceived fairness* remains constant. It appears that the GODMODE API was able to converge on a single value quickly. That value, unfortunately, was not the target value.

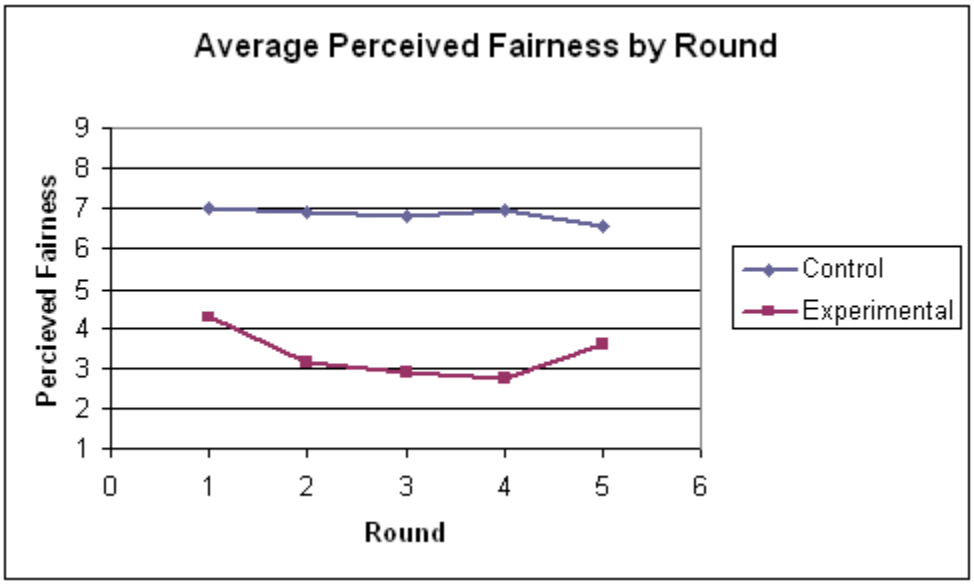

Figure 18: Comparing Effect of Perceived Fairness by Round

The trends for average *perceived fairness* per round mirrors the trend for the average fun per round. In the experimental group, the value for *perceived fairness* initially decreases, but then plateaus and even begins to increase. It may be that additional rounds would have allowed the value for *perceived fairness* to converge to the desired value.

5.3 Analysis of DDA System Failures

The experimental setup of Project Starfighter was designed with a logging mechanism to capture the values in the model every time they were updated. These data are used to analyze the equations in the model. Ideally, all the equations that

make up the model (fun, *perceived fairness*, *game metrics*, and *performance factors*) should all converge over time, yielding the desired setup customized to the player. Non convergence in any of these equations is an issue, since errors at the lower levels of the model will propagate upwards.

Analysis began at the top of the model, with the equation for fun derived from *perceived fairness*. The only time this equation can be validated is at the end of the round, when the player enters their actual value for fun. The models estimate for fun at the beginning and end of a round were compared with the actual value to derive an error estimate. If the equation is correct, the error at the end of a round should be less than at the beginning. Ideally, this error will converge to zero by the later rounds.

No statistical significance was found between the error at the beginning and end of a round. This could mean that the equation derived for fun is not complete. There may be factors that were not considered, or the coefficients for the factors included were incorrect. It is also possible that errors "lower" in the model had a strong effect on the equation.

Analysis of the error on various *performance factor* and *game metric* equations is visualized in Figures 19 and 20. These figures present the average error on each equation every second during the game. Performance factor one (PF1) is the only performance factor equation that seems to be converging. The error on the rest of the equations remains large throughout the entire game. It is likely that this discrepancy negatively affected the accuracy of the model and caused the decrease in fun for the player. While it appears the error on performance factor two (PF2) is relatively low, this can be explained by the large spike in error during certain times in the game, skewing the normalized results.

The performance factor equations likely did not converge due to incorrect levels

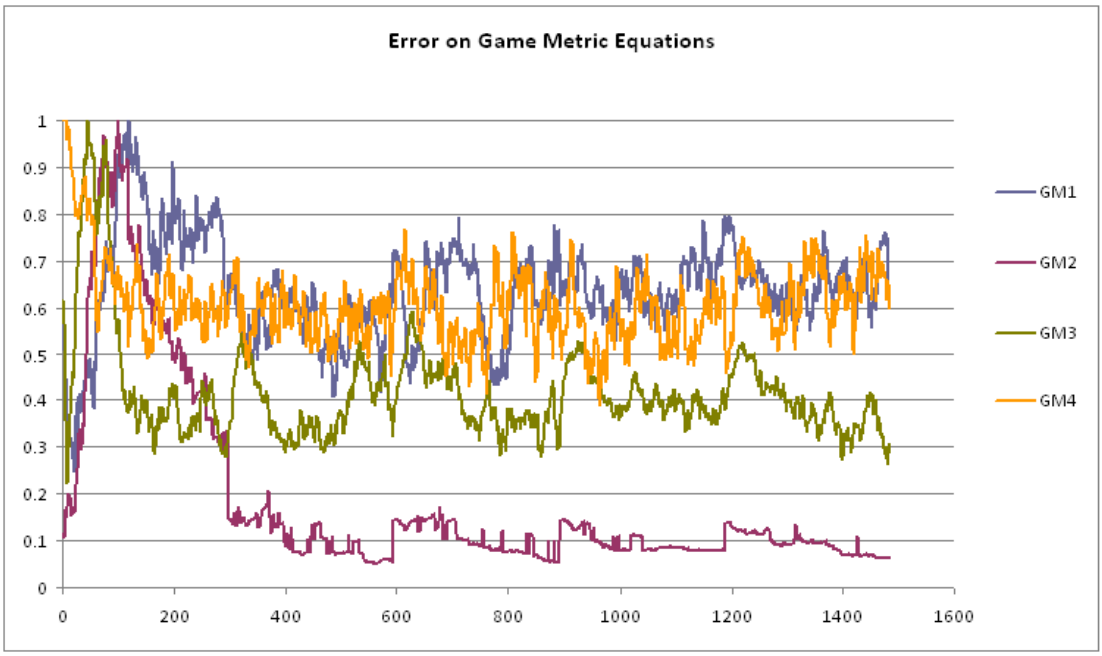

Figure 19: Normalized Error on Game Metric Equations Over Time

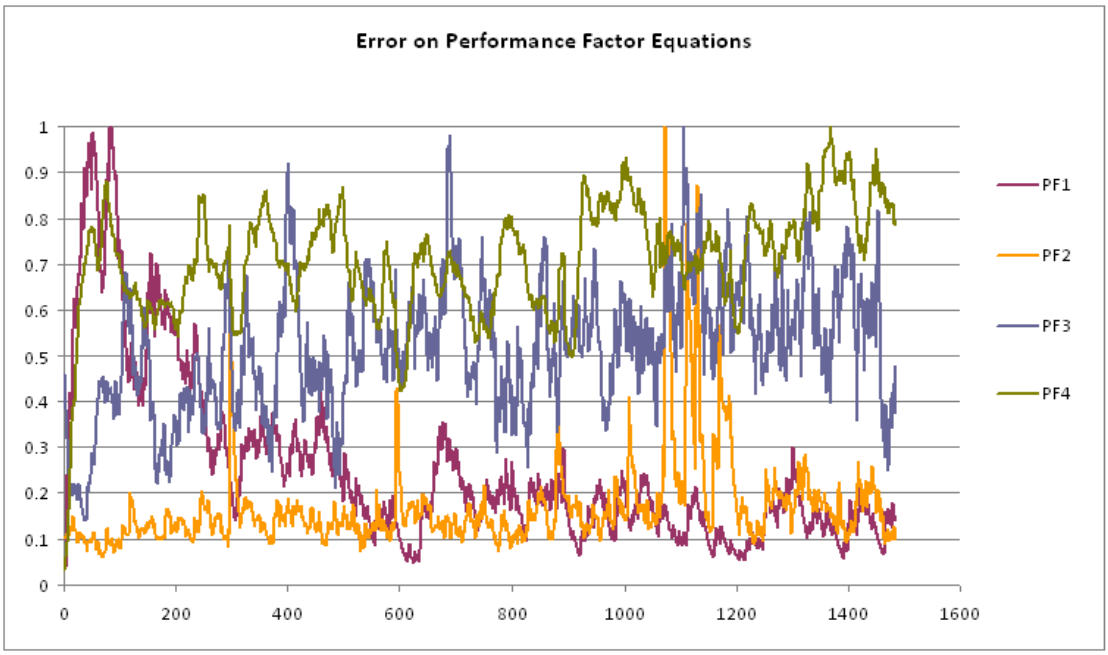

Figure 20: Normalized Error on Performance Factor Equations Over Time

of adjustment for the coefficients in the equation. The change in the coefficients after each update were not of the correct magnitude to account for the error in the

equation. The δ value for each factor in the equation may need to be adjusted to control the rate of change for the coefficients.

The trend of non-convergence continues in the game metric equations. Only game metric two (GM2) appears to be converging. The explanation for the error in the game metric equations is more complex than in the performance factor equations. Like the performance factors, the size of adjustments to the coefficients in the game metric equations may be incorrect. The performance factors also have a direct influence on the game metric equations. Errors in adjusting the performance factor equations propagate upwards, negatively affecting the game metric equations.

On average, PF2 (average distance to the target) represented the largest actual value in the model. While some values in the model were very small ($<$ .01), PF2 frequently had an actual value in the hundreds. Another approach to refine the model would be to normalize the data before generating the linear regression equations that make up the model. Doing so would reduce the effect of factors in the causal model that are many orders of magnitude larger than other factors. Updates to individual factors would be easier to control. This approach would likely prevent large errors like the one in equation PF2 from surfacing.

While the magnitude of the error of an equation is important, the sign of the error must also be taken into consideration. Ideally, the sign of the error should be split relatively even between positive and negative, while the magnitude of the error converges. If the updates to the coefficients (controlled by the δ values) in the equation are too large, however, the equation will never converge. After each update, the change in the coefficients will cause the equation to overcompensate for the error. The magnitude of the error will remain approximately constant. This process can repeat indefinitely in a process known as "thrashing". A "thrashing" equation does not converge. On the other hand, if the updates to the coefficients are

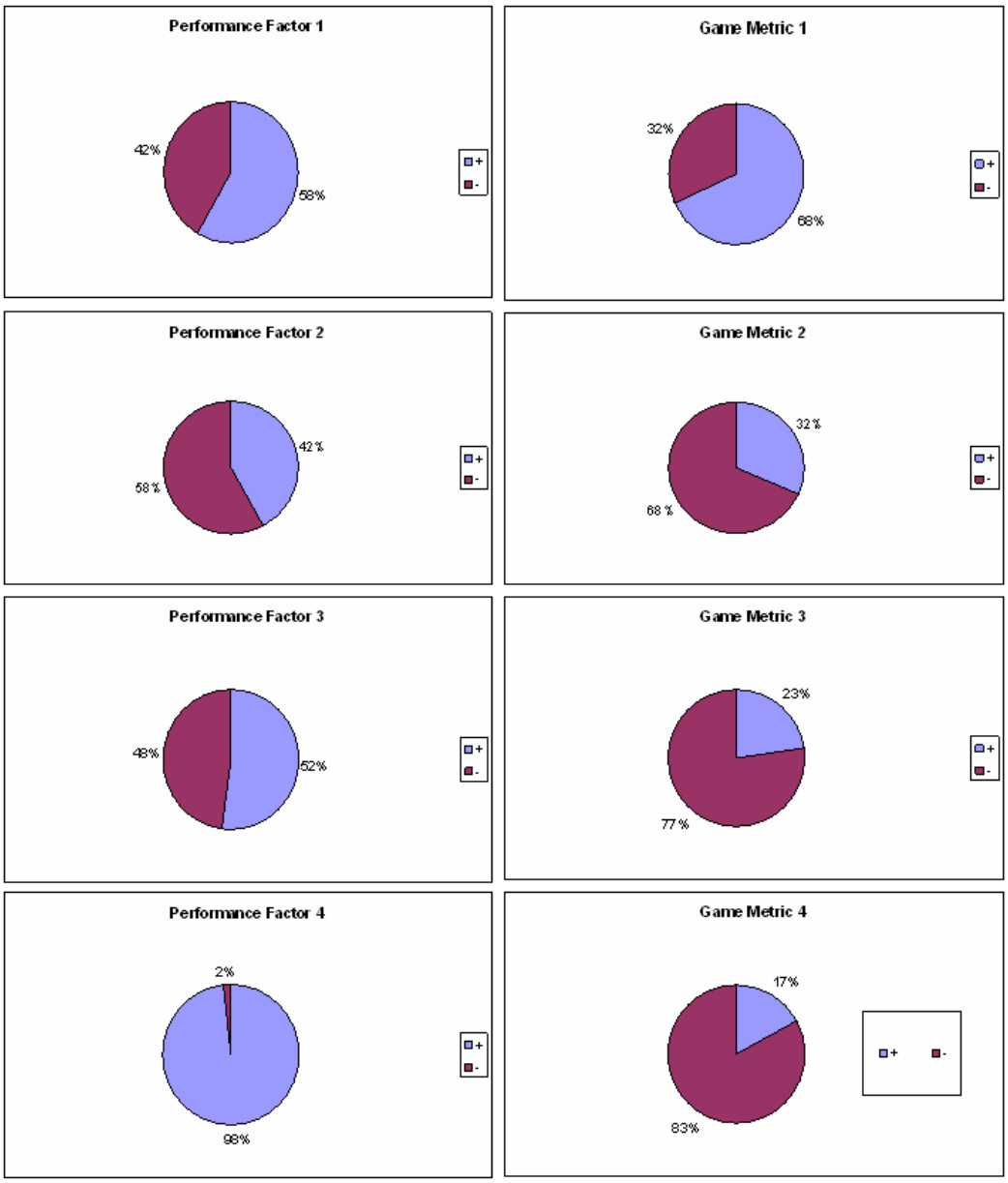

Figure 21: Analysis of Sign of Error on Performance Factor and Game Metric Equations

too small, the equation will not converge and the sign of the equation will remain relatively constant (either positive or negative).

Figure 21 analyzes the amount of time the error of each *performance factor* and *game metric* equation was either positive or negative. The δ values in GM1 and

GM3 likely are too large since the magnitude of the error remains large while the sign of the error is split. The δ values in GM4 will likely need to be increased since the magnitude of the error is large and the sign of the error is mostly negative. Looking at the *performance factor* equations, the δ values in PF2 and PF3 will need to be decreased for the same reasons discussed for the *game metrics*. The same reasoning for increasing GM4 can be used to justify increasing the δ values in PF4.

6 Conclusions

We have shown that a causal model can be used to describe how factors in a video game interact. Our causal model has been validated in two studies. These studies can be easily repeated for any game to construct a causal model that can then be used by game designers or as the basis of a dynamic difficulty adjustment system.

The model we developed expands upon previous work in the field, creating a more detailed understanding of how factors in a game interact. We validated the model by performing multiple studies. In doing so, we provided evidence to the validity of the model as well as provide a template for further work in this field.

The GODMODE DDA system can be easily applied to an existing game, as well as a game in development. By utilizing the three functions in the API, changes to a game can be controlled by the game developer. The API can be utilized by an existing code base with minimal changes.

When incorporating the GODMODE DDA system, Project Starfighter was shown to increase the amount of time a player spent in the game when compared to a control group where the game remained static. There was no significant difference between the groups when it pertains to the amount of fun a player had. Both groups deviated significantly from the desired "fair" game.

6.1 Future Work

While the initial studies described in this thesis help to validate the proposed causal model, more work is required before this causal model can be fully validated. The studies presented need to be repeated for multiple games in different genres. Games in a specific genre usually share similar traits. Looking at different genres allows for a more robust and complete exploration of the video game design space.

Games involving multiple human players should also be explored. Both games presented in this thesis involved only a single human player. Presenting multiple human players into a game substantially increases the complexity of any difficulty adjustment mechanism, as the actions of each player can not be controlled by the system. Players will have a direct impact on each other, hindering the impact of factors that the system can manipulate. Also, an equation for fun will need to be optimized for all players at the same time. It has been shown that different player traits (especially gamer type) have an important impact on how a game is perceived. Adapting to these different player traits at the same time will be difficult. Optimization may likely result in some players increasing their enjoyment of the game, while at the same time decreasing fun for other players.

As stated previously, the process for manipulating factors is different depending on if the set of factors are all discrete, continuous, or mixed. Currently, GODMODE. assumes that all the factors are discrete and will always use hill climbing to optimize the function for fun. All three algorithms for updating manipulated factors will need to be implemented to complete the API. A mechanism for choosing the appropriate control at run time will also need to be implemented.

A Appendix

A.1 Example Input File for GODMODE.

Godmode_data.txt

```
11
4
4
Score_Sec Kill_Death Kill_Sec Death_Sec Ammo_Sec Avg_Dist NPC_Shots_Sec
Targets_Sec W1 W2 W3 A1 A2 A3 NPC_Shots Avatar_Shots Max_NPC
NPC_Collect NPC_Reload Const
Fair .01 .038 .332 -14.438 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4.834
Ammo_Sec Avg_Dist NPC_Shots_SEc Targets_Sec W1 W2 W3 A1
A2 A3 NPC_Shots Avatar_Shots Max_NPC NPC_Collect NPC_Reload Constant
Score_Sec 11.841 -.011 0 3.066 0 0 0 0 0 0 -5.46 2.538 0 0 0 -1.046
Kill_Death 14.639 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2.828
Avatar_Kills .086 0 0 .034 0 0 0 0 0 0 -.048 0 0 0 0 .192
Avatar_Death 0 -.00002634 .023 0 0 0 0 0 0 0 0 -0.008 0 0 0 .076
W1 W2 W3 A1 A2 A3 NPC_Shots Avatar_Shots
Max_NPC NPC_Collect NPC_Reload Const
Ammo_Sec 0 0 0 0 0 0 0 0 0 .008 0 .033
NPC_AVG_dist 0 0 0 -25.435 -38.851 .001 0 -8.674 8.345 0 0 312.262
NPC_Shots_sec .418 -.566 .001 0 0 0 0 0 .163 0 -.028 1.037
Targets_Sec 0 0 0 0 0 0 0 0 .323 0 0 .098
Initial 1 0 0 1 0 0 3 5 5 20 20
Constraints 1 1 1 1 1 1 1 1 1 1 1
Maximum 1 1 1 1 1 1 10 20 20 100 30
```

```
Minimum 0 0 0 0 0 0 1 1 2 10 1
```

A.2 Informed Consent Form For Studies

Informed Consent Agreement for Participation in a Research Study

Investigator: Jeffrey Moffett

Contact Information: jeffmoffett@wpi.edu

Title of Research Study: Towards a Causal Model of Dynamic Game Balancing

Sponsor: Computer Science Department

Introduction (recommended) You are being asked to participate in a research study. Before you agree, however, you must be fully informed about the purpose of the study, the procedures to be followed, and any benefits, risks or discomfort that you may experience as a result of your participation. This form presents information about the study so that you may make a fully informed decision regarding your participation.

Purpose of the study: To collect data necessary to construct a model of an average video game player. This model will be used as the basis for a Dynamic Difficulty Adjustment System.

Procedures to be followed: You will play the game "Project Starfighter" for duration of between 10 and 30 minutes. Initially, survey information about your age, gender, and experience with video games will be collected. You will play a round of the game, pressing the Escape ('esc') key when you are no longer enjoying the game. Upon completing the round, you will be asked to rate how fair and fun you thought the round was. This will continue for 5 rounds of the game.

Risks to study participants: There are no foreseeable risks to participating

Benefits to research participants and others: The data collected here will

be used to build a system that improves upon previous efforts at game balancing. If successful, the study conducted here would provide a template for others to improve the level of enjoyment everyone receives while playing video games.

Record keeping and confidentiality: No identifying information will be kept about you. All information will be stored on a single computer located at WPI. The data will only be used for the specific purposes listed above. There is no danger of being identified as a participant if the data was to be stolen or divulged.

Compensation or treatment in the event of injury: "You do not give up any of your legal rights by signing this statement."

For more information about this research or about the rights of research participants, or in case of research-related injury, contact:

Jeffrey Moffett (Email: Jeffmoffett@wpi.edu)

IRB Chair (Professor Kent Rissmiller, Tel. 508-831-5019, Email: kjr@wpi.edu) and the University Compliance Officer (Michael J. Curley, Tel. 508-831-6919, Email: mjcurley@wpi.edu).

Your participation in this research is voluntary. Your refusal to participate will not result in any penalty to you or any loss of benefits to which you may otherwise be entitled. You may decide to stop participating in the research at any time without penalty or loss of other benefits. The project investigators retain the right to cancel or postpone the experimental procedures at any time they see fit.

By signing below, you acknowledge that you have been informed about and consent to be a participant in the study described above. Make sure that your questions are answered to your satisfaction before signing. You are entitled to retain a copy of this consent agreement.

Study Participant Signature

_____Date: _____

Study Participant Name (Please print)

_____Date: _____

Signature of Person who explained this study

A.3 In-game Survey Questions for Project Starfighter

Figure 22: Survey Question on Gender of Participant

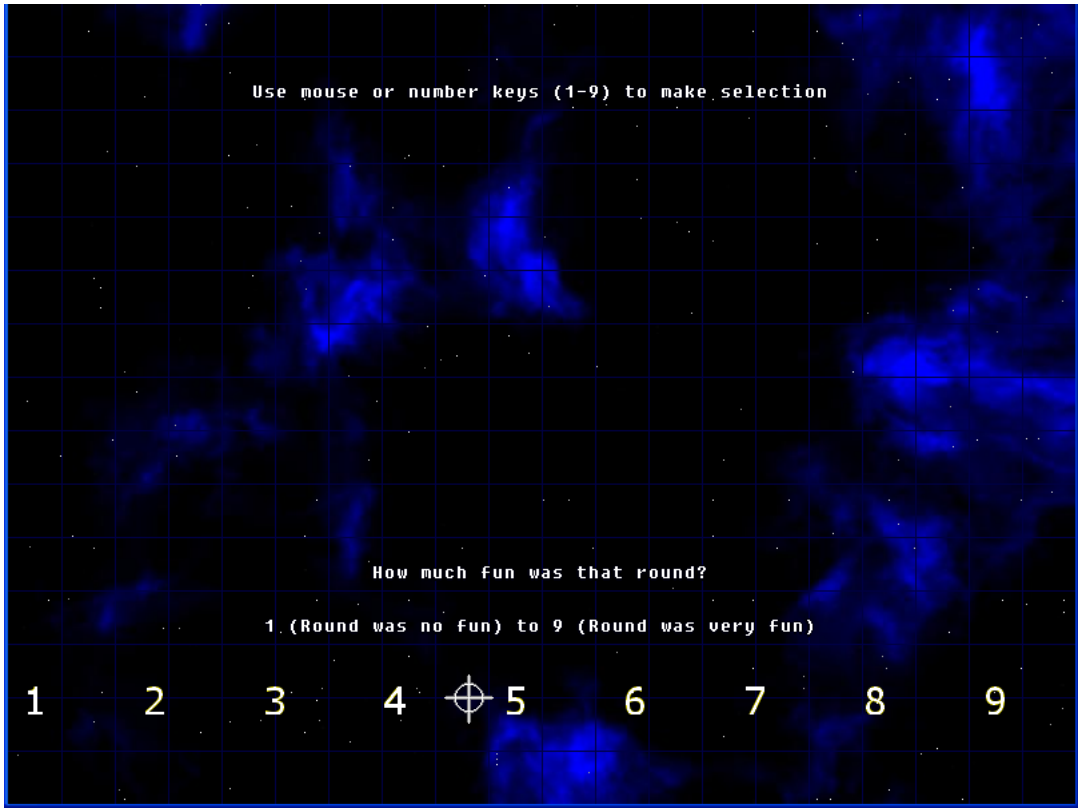

Figure 23: Survey Question on Fun of Round Presented to Participant

Figure 24: Survey Question on Perceived Fairness of Round Presented to Participant

References

[AR07] Ernest Adams and Andrew Rollings. *Game Design and Development: Fundamentals of Game Design*. Pearson Prentice Hall, Upper Saddle River, New Jersey, 2007.

[Bur06] Kevin Burns. In *Bayesian Beauty: On the ART of EVE and the Act of Enjoyment*, CIG-06, Boston,Massachusetts, JUL 2006. AAAI Workshop.

[Bur07] Kevin Burns. *EVE's Entropy: A Formal Gauge of Fun in Games*, volume 71 of *Advanced Intelligent Paradigms in Computer Games*. Springer, 2007.

[Coh95] P. Cohen. *Empirical Methods in Artificial Intelligence*. MIT Press, 1995.

[Cs98] M. Cskszentmihlyi. *Finding Flow: The Psychology of Engagement With Everyday Life*. New York,New York, 1998.

[Dey09] Tamal Dey. Simplex algorithm, April 2009. `http://www.cse.ohio-state.edu/ tamaldey/course/794/simplex.pdf`.

[Gam10] Moby Games. Genre definitions, March 2010. `http://www.mobygames.com/glossary/genres`.

[HHSA08] A. Hefny, A. Hatem, M. Shalaby, and A. Atiya. In *Cerberus: Applying Supervised and Reinforcement Learning Techniques to Capture the Flag Games*, Fourth Artificial Intelligence and Interactive Digital Entertainment Conference, Stanford,California, 2008. AIIDE.

[Hun05] R. Hunicke. *The case for dynamic difficulty adjustment in games*. ACE '05. New york, New York, 2005.

[Pea00] Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, New York, New York, 2000.

[Pis06] Paulo Piselli. *Relating cognitive models of computer games to user evaluations of entertainment*. Master's thesis. Worcester Polytechnic institutel, Worcester, Massachusetts, 2006.

[Ric79] Elaine Rich. User modeling via stereotypes, 1979.

[SW05] P. Sweetser and P. Wyeth. *GameFlow: A Model for Evaluating Player Enjoyment in Games*. 2005.

[UJG08] P. Ulam, J. Jones, and A. Goel. In *Combining Model-Based Meta-Reasoning and Reinforcement Learning For Adapting Game-Playing Agents*, Fourth Artificial Intelligence and Interactive Digital Entertainment Conference, Stanford,California, 2008. AIIDE.

[YH07] G.N. Yannakakis and J. Hallam. *Towards Optimizing Entertainment in Computer Games.* Applied Artificial Intelligence. 2007.