Time Balancing of Computer Games with Adaptive Time-Variant Minigames

By Amin Tavassolian
Department of Computer Science
University of Saskatchewan

Supervisors:
Dr. Carl Gutwin
Dr. Kevin G. Stanley

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

ABSTRACT

Game designers spend a great deal of time developing well-balanced game experiences. However, differences in player ability, hardware capacity (e.g. network connections) or real-world elements that impose fixed temporal constraints on the game (as in mixed-reality games) make it difficult to balance games for all players in all conditions. In this research *adaptive time-variant minigames* (ATMs) are introduced as a way of addressing the problems of time balancing. These minigames are parameterized to allow both a guaranteed minimum play time (to address fixed temporal constraints), and dynamic adaptability (to address temporal variances caused by individual differences). We also introduce three time adaptation algorithms and analyze the interaction between adaptive algorithm, game mechanic, and game difficulty in controlled experiments. The studies showed that there are significant effects and interactions for all three factors, confirming our intuition that these processes are important and linked. We further find that finer temporal granularity leads to less-perceptible adaptation and smaller deviations in game completion times. The results also provid evidence that adaptation mechanisms allow accurate prediction of play time, that the minigames were valuable in helping to balance temporal asymmetries in a real mixed-reality game, and that they did not detract from the overall play experience.

CHAPTER ONE
INTRODUCTION

The history of video games began in the 1940s, when Thomas T. Goldsmith, Jr. and Estle Ray Mann filed a United States patent for an invention they described as a "cathode ray tube amusement device." [1]. Video gaming would not reach mainstream popularity until the 1970s and 1980s, when arcade video games, gaming consoles, and home computer games were introduced to the general public. Video gaming has since become a popular form of entertainment and a part of modern culture in many parts of the world. Many games are very successful – eg. *World of Warcraft* has made over 10 billion dollars since 2004 [2].

Video games attract players with many different skill levels - from casual gamers to tournament champions - and *game balance* has received considerable attention (eg [3, 4]). There are two main types of game balance that are relevant to this work: *outcome balancing*, and *player balancing*. Outcome balancing is concerned with ensuring that players of equal skill have an equal opportunity to win the game, regardless of their starting orientation or the setup of the game.

Truly balanced multi-player games are rare. Chess is a well-known example of a balanced game because both players start the game with identical resources, and also start with similar positions. Interestingly, there is one aspect of the game that is unbalanced: there is an unavoidable asymmetry of the game mechanic where one player has to play first.

Several aspects of games have been investigated as potential means for accomplishing balance, such as the available strategies for different character types in *Neverwinter Nights* 2 [5]. *Neverwinter Nights* is a Massive Multi-player Online Role Playing Game (MMORPG) in which players create customized characters to represent themselves and have the opportunity create a group with their characters to finish a set of missions. There are several different strategies in the game that players can perform given their avatars. Each strategy is unique in terms of visual effects and the way it should be performed, but the consequences of these strategies are all reasonably balanced to prevent a player receiving a huge advantage based on avatar choice. The allocation of initial resources in *Age of Empires* [6] and level of powers and damage in *Mortal Kombat* [7], are other examples of different methods of game balancing. In Age of Empires, a player selects a specific tribe/race prior to the start of the game and receives a section of the game map, which offers a limited set of resources. Game designers can use these differences and

resources for balancing purposes. In Mortal Kombat, players select their character before the game starts. Regardless of the character's personality and features, game elements such as the level of power when hitting opponents and the amount of damage received from others are reasonably balanced.

Although outcome balance is an important aspect of game design, it can also lead to problems in situations where players do not have equal skill levels. The "equality" in outcome balancing refers to equality (or near-quality) of game resources and game opportunities for players with same the skill level. However, one of the factors that make multi-player games interesting is the unavoidable differences in individual experience and skill. Hence, in such games, the balancing methods of the game should be able to compensate for these differences and prevent players from receiving any preference. When players with different skills try to play a multi-player game, the experience can be problematic, because players with more experience and expertise win proportionally more often, potentially making those with less experience unhappy. This situation may not be desirable for expert players either, because they can win games too easily.

In general, games should be balanced not only in terms of *fairness*, in that players with greater skill should usually prevail, but also in terms of *competitive flow*, in that the game should provide an engaging and competitive experience for all players even if they have different skill levels.

The second type of balancing is player balancing. The player balancing makes sure that the game remains competitive even if two players have different skill levels. Some games have implemented mechanisms for player balancing. For example, real-world games such as golf or racing use handicaps or head starts to balance different skill levels. In the video game *MarioKart* [8], powerups are allocated unequally to players based on their standing in the race: players at the back of the pack will receive more, and better, powerups that help them to stay competitive.

However, there are relatively few mechanisms for balancing players that are non-obvious and that do not interfere with the gameplay experience. Another example is *Diablo3* [9] which strength and number of enemies surrounding the player varies based on the skill of player: Usually for an experienced player with good performance in the game, enemies are more in terms of quantity and are stronger. Conversely, for novice players, fewer weak enemies surround the player for fight.

However, there are relatively few mechanisms for balancing players that are non-obvious and that do not interfere with the gameplay experience. For example in NeverWinter Nights, some of

the items are rudimentary unavailable at some situations in the game for expert players to make the game harder. Variable loading time of the weapons is another example of player balancing in Neverwinter Nights. As the player progress in the game, the level of the player's character is increased resulting in more powerful weapons with shorter loading time. But during the game play, the loading time varies based on the payer's performance and in some situations the same weapon is loaded faster for a novice player than an expert. Although this approach adjusts the balance of the game, it does not satisfy expert players.

In this thesis, I explore one possible mechanism that can provide new opportunities for player balancing – *time.* My focus is in the manipulation of *time* – that is, the amount of time needed for players to complete activities in the game, such as obtaining resources, building units, moving to different locations or defeating an enemy, as a mechanism for balancing multi-player games.

Time-based activities can be seen in many games: in race-based games such as MarioKart, in games requiring synchronized motion between heterogeneous agents [10], in games employing rates of production such as *StarCraft 2* [11], and in games with 'cooldown' mechanics such as *World of Warcraft* [12]. In this thesis I introduce a novel game balancing method that uses *minigames* as adaptable units, which can manipulate the timing of larger tasks and actions in games and deliver a balanced solution for certain design goals.

I focus on the time element for balancing because for game mechanics with a significant temporal component, the time taken for different activities is an obvious way that more-skilled players differentiate themselves from less-skilled players. For example, in Age of Empires, a professional player is able to quickly create an empire and start fighting with other nations, while a similar process takes much more time for a novice player. As a result, the less-skilled player will be beaten before getting a chance to build sufficient forces.

Some issues of time balancing can be dealt with in game design (e.g., ensuring that faster units are less powerful), but two particular situations cannot be completely solved in design, leading to temporal asymmetries that must be addressed during play. First, individual differences in experience or skill mean that two players will take different amounts of time to complete particular tasks; this situation affects a wide variety of multi-player games. Second, some games – e.g., *mixed-reality games* [10] and *pervasive games* [13] – involve aspects of the real world that impose fixed temporal constraints. For example, the amount of time it takes for a player to run from one game area to another is determined by the size of the real-world game space, and

cannot be changed in the design of the game; once the game rules are set and players take roles, it is not possible to dynamically change those rules determined by players in the middle of the game to address the unforeseen temporal inequalities that are raised during the gameplay. If the game space is fixed, the only way to balance the timing of tasks is to make the players faster, which is not usually possible.

The time-based mechanisms and actions in the main game could be manipulated to balance players of different skill levels; however, directly manipulating the time or timing parameters of main game activities can be disruptive for the player, and complex mechanics could be rendered unstable by the feedback loop created by the adaptation algorithm.

An alternative approach is to manipulate time through activities that are outside the main game – such as through *minigames* that appear at various points within the game, but whose (usually simple) mechanics are different from the main game activities. Minigames are simple activities contained within a larger game, and are common in commercial titles (e.g., *Mario Party*, *Sid Meier's Pirates!*, and *Assassin's Creed 2*). Minigames can help designers balance temporal aspects because they can add time to a player's main game task or reduce the time of the specific task in a mission. Figure 1 shows some of the four-player minigames in Mario Party.

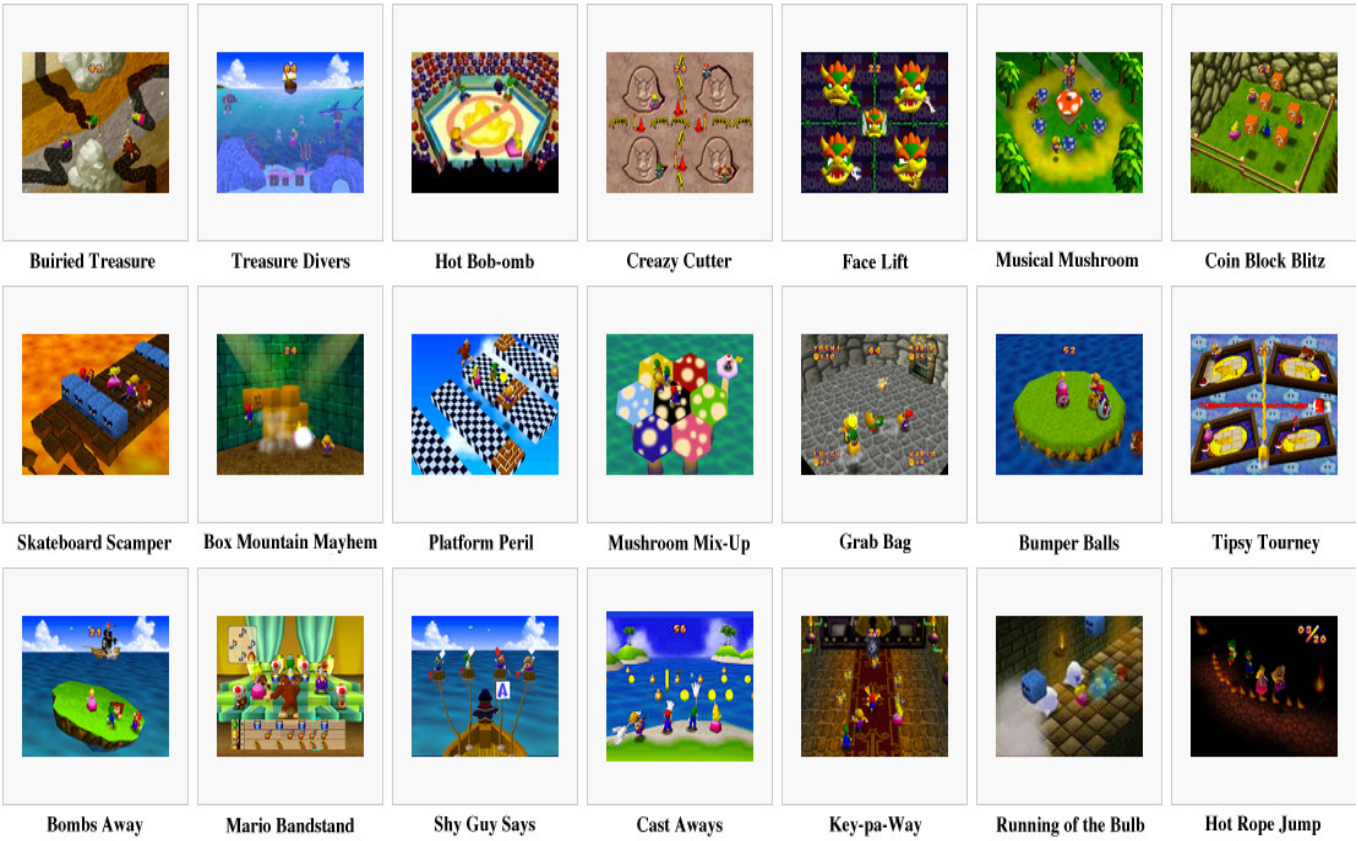

Figure 1: List of minigames in Mario

In this research I introduce a novel way of carrying out time balancing through the use of *adaptive time-variant minigames* (ATMs). ATMs are simple minigames contained within a larger game that balance temporal flow by adding varying amounts of time to a player's main-game task or mission. For example, a player might have to complete a lock-picking minigame to break into a building – and the amount of time taken can be controlled by appropriate parameterization of the lock picking activities.

Adaptive time-variant minigames provide designers with considerable flexibility: in an ATM, the minigame is parameterized over a range of completion times, based on the game state and player skill. Minigames can be started as part of traditional game mechanics, such as when a character casts a spell in *World of Warcraft* or when a production order is issued to a building in StarCraft 2. The minigame would then spawn as part of the main-game mechanics. In order for the primary task to be completed, the minigame must be completed successfully.

The ATM approach has several strengths: it decouples the balancing activity from the primary game play; it allows the creation of specific minigame-based interactions to mask the temporal adaptation; and it provides the designer with two primary mechanisms to alter balance: the initial difficulty level (often based on main game state), and dynamic elements of the minigame adjusted during gameplay (often based on player performance in the minigame).

To test the efficiency of ATMs as tools for balancing game time, I developed four different minigames with three different balancing algorithms (Discrete balancing, Continuous balancing and State balancing) and carried out three studies using these games. The first laboratory study examined whether the minigames were able to manage time correctly in isolation. In this experiment I used the simplest form of balancing algorithm which adapted the minigames at only one point during the game play (*Discrete*). The second study tested the real-world effectiveness of ATMs in a real mixed-reality game called *Stealth Hacker*. In the third study I investigated the effect of temporal adaptation granularity and game genre on time balancing abilities of ATMs. In this experiment I used all the balancing algorithms (*Discrete, Continuous, State*) and compared their performance in terms of accuracy and user experience. Although this research was a limited trial, my results show that the adaptive time-variant minigames were able to provide temporal balance without detracting from the main game. My experiences with ATMs suggest that the underlying principle can be used more generally to assist designers with time balancing in a wide variety of multi-player games.

This work provides three main contributions. First, it provides evidence that adaptive time-varying minigames are effective tools for time balancing. To show the effectiveness, I analyzed the result of the experiments in terms of enjoyment level and accuracy of completion times of the minigames. Second, it demonstrates the feasibility of ATMs in a real mixed-reality location-based game and that they were able to manage the time balancing of different tasks and activities in the game. Third, it shows the differences between three adaptive approaches with different adaptation granularities, and shows that the type and difficulty of the minigame had a substantial effect on the adaptation. Moreover, it demonstrates that Continuous balancing performed best both in terms of time manipulation and perceptibility. The results of this work provide new and valuable information for multiplayer game developers on the design, deployment, and evaluation of minigame-based techniques for time balancing.

In the following chapters, I will describe time balancing in games and will explain the research methodology and experiments.

- In Chapter two, I will present a survey of related work, which forms the foundation of this thesis. First, the general concept of game balance will be discussed; second, I briefly discuss different approaches in game balance of games; and third, different time balancing methods in games are discussed.
- In Chapter three, I discuss time balancing in computer games. First, I explain the concept of time in computer games. Second, I identify the parameters in games that are related to game balance. Third, I explain the concept of time balance in detail and review some of the common issues of time balancing. Fourth, I introduce minigames as separable, manageable games that are independent from the main game. Fifth, the idea of time balancing of game by minigames will be discussed.
- In Chapter four, I will introduce a model to record player progress during the game. I will present three different balancing algorithms and will review their specification and compare them.
- Chapter five is dedicated to the evaluation phase of the research. In this chapter I provide performance information about the balancing algorithms in chapter four. In this chapter the main focus is the accuracy of the balancing algorithms and the noticeability of the different balancing methods and players' experience.
- Chapter six presents a discussion of the most important outcomes of this work, in particular chapter five. Higher-level implications of my findings and issues related to the work as a whole are addressed.
- Chapter seven summarizes the research presented in this thesis, discussing the main contributions of my work and highlighting avenues of future work that have been opened as a result of this thesis.

CHAPTER TWO
RELATED WORK

Unlike many other types of media, video games are designed to generate interactive and engaging experiences [14], [15] and now constitute a major branch of the entertainment industry [16]. A game is a form of art in which participants, termed *players*, make decisions in order to manage resources through game tokens in the pursuit of a goal [17]. Each game is, in fact, a system which has several components - such as players, objectives, procedures, rules, and resources - that are interacting together to reach a goal [18], [19].

As Rollins and Adams state in *On Game Design*, "you need to keep the players in the balance sweet spot for as long as is practical in order to keep the game fun and let the underdogs have a chance to catch up. However, the major factor that determines winners should be player skill" [20]. In essence, the goal of game balancing is to allow the best player to finish first, but to maintain a competitive margin between players.

When players with different skill levels play games, they lose "flow" because of feelings of incompetence or lack of challenge. Flow represents the feeling of complete and energized focus in an activity, with a high level of enjoyment and fulfillment [21]. One aspect of *flow* [22] is the degree to which a game provides an experience for players that has an appropriate level of challenge: if the game's challenges exceed the player's ability, it leads to frustration; if challenges are lower than the player's skill, the player becomes bored [23], [24], [25], [26], [27]. Challenge is a crucial part of every game [28]. Crispini [29] has discussed criteria to make a simple online game enjoyable and appealing. The human survey-based outcome of his work presents challenge, diversity and unpredictability as primary criteria for an enjoyable game.

The research domain of this thesis is balancing the timing of tasks and activities in multiplayer games using. To reach this goal, the following areas of research must be discussed:

1. Time balancing of computer games is a subset of different types of game balancing, so I first define game balancing and identify its parameters and types. Different terms and definitions, described by other researchers, provide a detailed knowledge about what game balance means and why it is important.

2. Game balancing methods vary in terms of game genre, number of players, frequency of balancing algorithm and, other parameters. Different game balancing studies can provide insight into game balancing methods, best practices and key parameters that should be considered while designing a new game balancing method.
3. In this thesis, a secondary focus is on time balancing in multi-player location-based mixed-reality games. There are many successful multi-player mixed-reality games that have exploited different game balancing methods to synchronize digital and real players in mixed-reality games. In this section I will discuss previous multi-player mixed reality games and will review different approaches in game balancing for these games.

2.1 Game Balance in Computer Games

Computer game balance is recognized as a design issue that has profound effects on enjoyment – mutually influencing challenge and user satisfaction [30], [31]. This issue is a common topic in every game regardless of the number of players and genre. Previous research divides the game design process into several sections, and game balance is declared as an early stage of the design process [32], [20]. If balancing issues cannot be addressed during the game design process, it will be postponed to the later phases and will be more difficult to be dealt with. For example in CatchBob! [33], the balance of the game was affected by lack of sufficient lines of sight in the game's location, which could have been addressed earlier in the design section of the game.

In general, regardless of the method of balancing, previous methods of managing game complexity and balance, such as tuning the difficulty of static pre-defined levels, are often labor-intensive [34]. Bateman et al. [35] divided game balancing into gameplay balancing and player balancing to emphasis on the role of players' skill and experience in game balance and suggested three different approaches: *Matchmaking* (grouping players by their abilities)*, Asymmetric Roles* (assign different roles to different players based on their skill level and experience) and *Difficulty Adjustment* (Adaptively adjust the difficulty of the challenges in the game). In fact, maintaining optimal game balance often needs to be a dynamic process because of the evolution of the player's behavior and skill [36]. A good example of this is the constant upgrading in World of Warcraft [12] to patch newly discovered/ created exploits.

2.2 Different Mechanisms in Game Balancing

A game balancing approach can be considered from two different aspects: the degree of adaptability of the overall approach of game balancing and actual game balancing algorithms employed to achieve the approach. *Static game balancing* and *dynamic game balancing* are the two primary approaches for the adaptability, which are considered here. Many game balancing algorithms are possible once an approach has been determined, and key implementations are discussed here.

2.2.1 Dynamic Game Balancing vs. Static Game Balancing

A primary issue in competitive games is that the different teams or players should have equal chances to win the game based on rules and starting positions [20]. Balancing fairness can involve manipulations of different game elements – for example, the capabilities and initial resources allocated to player types such as *Orcs* and *Humans* in *WarCraft* [12]. This type of balancing (called 'static balancing') is often carried out through repeated playtesting of the game mechanics and parameters [34], such as tuning the capabilities of individual weapons or units [20].

The idea of balancing a game dynamically during game play is not new [36]. Dynamic balancing, considers a fully continuous spectrum of play, from the starting point of the game to its end. Dynamic balancing differs from static balancing because the interaction of the player or players with the game should be considered, and different units and parameters in the game configuration should be adapted based on the current state of the game [37] rather than at the start of play based on player models. Variable frequency of enemies in Diablo 3 and variable power of enemies in *Assassin's Creed 4: Black Flag* [38] are examples of dynamic balancing during game play.

2.2.2 Game Balancing using Playtesting

One traditional way of balancing games is playtesting. The playtesting is performed by iteratively refining the value of penalties, awards, setting thresholds and other important game parameters until the game is deemed balanced [39]. In playtesting, game designers select a

statistical population (players) to play the game and iteratively refine important parameters of the game to reach an optimum static value. These optimum values can be set based on either statistical analysis of test results or by players' answers to questionnaires.

2.2.3 Game Balancing using Artificial Intelligence

High quality game AI has become an important selling point of computer games in recent years [40]. However, game players often still prefer to play against human controlled opponents (via a network) rather than computer controlled ones [41]. Olesen has explored neuro-evolution methodologies to generate intelligent opponents in Real-Time Strategy (RTS) games and tried to adapt the challenge generated by the game opponents to match the skill of a player in real-time [42]. Several previous approaches focused on the game's AI and probabilistic methods to address dynamic balancing. In Knock'Em [43], reinforcement learning techniques has been employed to build intelligent agents that adapt their behavior in order to provide dynamic game balancing. Hunicke [26] has explored a computational and design requirements for a dynamic difficulty adjustment system using probabilistic methods based on *Half Life* game engine [44].

2.2.4 Player Balance in Multi-Player Games

There are four main ways that designers can balance competition in multiplayer games (also called player balancing [35]). First, a few methods exist for balancing competition without changing the game – for example, ranking systems and ladder tournaments help match players with opponents who have similar skill levels. Lida's work on measurement of entertainment in board games was the first attempt to provide rigour for ranking [45]. He introduced a general metric of entertainment for variants of chess games depending on average game length and possible moves.

Second, games can be designed so that a stronger player is given an explicit disadvantage, such as handicapping in golf or a head-start in playground games. In computational environments, games can also be designed with asymmetric roles, placing the stronger player at a disadvantage [46]. Although this strategy can be successful, the balancing mechanism is readily apparent to the players, potentially reducing the sense of fairness, which is a primary goal of a balancing scheme.

Third, some games naturally evolve in such a way to make winning more difficult as the game progresses. For example, in 8-ball billiards, the leader has fewer balls to aim at, and more of their opponent's balls to avoid [20].

Fourth, some player balancing techniques dynamically alter the characteristics of game elements during play to even out the competition. This approach was used in a version of *Pong* that was intended to allow parents and children to play together: the game automatically adjusted a player's capabilities (paddle size and movement speed) based on the current score [25]. A similar capability adjustment is seen in the 'Fatboy' mod of *Unreal Tournament*, which adjusts the width of a player's avatar based on their kill-to-death ratio, making it easier to hit better players [47].

A third example is a system which provides differential targeting assistance using techniques such as target gravity or sticky targets [35]. The amount of assistance given to players is based on the score differential: as a player falls further behind, their targeting cursor becomes more attracted to the targets. A study of this technique showed that it increased competiveness, and that neither the strong nor weak players noticed the adaptation.

2.2.5 Game Balancing using Player Satisfaction

Cognitive user models of playing experience promise significant potential for the design of digital interactive entertainment systems such as augmented reality games. Quantitative modeling of entertainment or satisfaction as a class of user experiences may reveal game features or user features of play that relate to the level of satisfaction perceived by the player. That relationship can then be used to adjust digital entertainment systems according to individual user preferences to optimize player satisfaction in real time [48]. Some of the previous methods have considered "user satisfaction" as the key element to deal with game balance and have categorized different game balance methods based on it [30].

2.2.6 Game Balancing using Time

The dynamic player-balancing techniques described above all act on player *capabilities*; fewer techniques have explored adjustments to the *time* required for different player actions and tasks. One game genre that does frequently use time balancing is the racing genre – many racing games

implement ‘catch-up’ or ‘rubber-band’ effects [35] in which a slower player receives a speed boost. For example, *Mario Kart* provides the ‘Bullet Bill’ power-up only to players who are far behind the leaders, which dramatically increases speed without the need to steer.

2.3 Game Balancing in Multi-Player Mixed-Reality (MMR) Games

Multi-player Mixed-reality (MMR) games, which incorporate real and virtual play simultaneously, face particularly acute time balancing issues. Generally, the physical portion of the game relies on existing infrastructure such as buildings, roads, and bridges, and is difficult to modify; similarly, the behavior of real world participants is dictated by physics and human physiology and cannot be altered. The majority of time balancing must therefore take place in the virtual portion of the game.

Balancing a mixed-reality game is naturally harder than previously mentioned genres because game designers have to synchronize virtual and real worlds and balance the game on each world. There are many parameters that should be considered while balancing a mixed-reality game especially those that are imposed from the real world. In Treasure [49] players should pick up coins scattered around an urban area and put them into a virtual chest. Results of the game showed that the chance to load up the found coins was higher with a better network connection, resulting in an unbalanced advantage for a group of players with better network devices. In general, the level of the player’s knowledge about the physical terrain of a mixed-reality game affects the balance of the game [50].

Several approaches have been proposed to balance mixed-reality games. For example, online players may play on a scaled-down representation of the real playground with speeds adjusted proportionally to be appropriate for this scale [10]. *NetAttack* [51] divided players based on their roles and balanced play, but not timing based on role. In *Manhattan Story Mashup* [52] static minigames have been employed to implicitly manage game balance. Players were given a clue as a part of their ‘mission’ and were then asked to take a picture of the most related object within a cool-down timer, but the timer in the minigame is fixed, and variations in skill or the surrounding context do not change the duration.

Most solutions to time balancing in MMR games have presumed that virtual interfaces are point-to-point mapped to the real world – that is, that virtual players play in simulacrums of the real playground. Timing is implicitly addressed by setting virtual locomotion speeds to be

approximately equivalent to expected real world locomotion speed [53]. While straightforward and easy to implement, this assumption is overly limiting and constrains the design space for MMR games.

I introduce a new type of time-balancing mechanism that can be used in a wider variety of game types. This new mechanism uses *adaptive time-variant minigames* (ATMs) to adjust the time taken for main-game tasks that incorporate a minigame as part of the overall action. As stated in [54], "minigames are particularly attractive for time balancing because they are intended as short-duration activities, and can unobtrusively and selectively delay specific players without unduly disrupting the overall gaming experience".

In this section I reviewed some of the previous works in the game balancing domain. As most of these works suggested, game balance is a crucial issue on every type of game, which should be addressed in early stages of the game development process. There are many game balancing approaches that have positive and negative points. I categorized the general trend of game balancing methods into two major classes: Static game balancing and Dynamic game balancing. Previous literature shows that dynamic game balancing has been successful in many game balancing scenarios. By reviewing the possible approaches to perform the dynamic game balancing, I determined that in none of the previous works except one case [52], minigames have been employed for game balancing purposes. I also found that most of the previous works did not use time as the primary element for game balancing.

CHAPTER THREE
GAME BALANCE AND TIME

The idea of time balance is based on the phenomenon that activities in many games (particularly in digital games) take specified amounts of time. Players perform different activities toward the narrative of the game to finish it. If there was a way to calculate the completion time of activities in games, it would be possible to use this completion time as a parameter to balance different activities in games. In general, the total completion time of an activity is influenced by parameters such as gaming skill, the game interface, the complexity of the game input, complexity of the game environment (difficulty of the game) and the underlying mechanics. In this chapter, I will discuss these parameters in detail.

The area of the game where time will be manipulated is that of minigames. Minigames are generally short, self-contained play experiences within a larger game framework, but with their own internal logic, game state, and mechanics [55]. Because minigames have their own internal mechanics, they can be configured independently of the main narrative or action, making them an attractive alternative for dynamic balancing because the initial and goal game states can be made contingent with the overall game state. Minigames are particularly attractive for time balancing because they are intended as short-duration activities, and can unobtrusively and selectively delay specific players without unduly disrupting the overall gaming experience. In this chapter I will introduce a novel solution to balance the timing of different activities in games using minigames.

In general, time balancing using minigames has four main steps:

1- Identify the type of the game and potential issues relating to the balance of the game. The type of the game includes a set of specifications of the game such as the number of players (single player or multiplayer), game genre (fighting, maze, shooter, etc.), game mechanics (match pattern, find signal, etc.) and so on. Each game balance method has a set of parameters that let the balancing algorithm manipulate challenges and total difficulty level of the game. For example, in a multi-player first person shooting game, resources such as weapons are shared and players compete to earn them, while in a

multiplayer racing game players compete to finish the race as quickly as possible. Hence, the first step is to determine the type of the game and the preferred balancing method.

2- Design minigames that fit with the context of the game. Minigames are independent and can incorporate mechanics and design elements independent of the main game; however, immersion will likely suffer if there is design and mechanics inconsistency. For instance, it is not reasonable to put a silly minigame into a horror genre, since this would adversely affect the overall mood; however, it does not mean that the type and genre of the minigame and the main game must necessarily be the same. One advantage in using minigames for game balancing is the required time for players to complete minigames, which can be used as a parameter to balance the main game. For example, a minigame can be started any time that the main game needs to be balanced [56].

3- Identify manipulable elements in the minigames and their relationships. As will be discussed later in this chapter, each game has several constitutive components that interact together through the game mechanics. When the game mechanic is chosen, we can use mathematics to calculate the required time for each component and eventually calculate the total required time for given scenarios.

4- Find specific situations in the game state of the main game from which the chosen minigames should be triggered. Depending on the type of game and the balancing algorithm, it is possible to define *trigger points* in the main game, then select and run a minigame when the player reaches to these points. A *trigger point* is a specific situation in the main game, definable in the context of the main game and repeatable if the prerequisite situation is reached. Prior to starting the minigame, all the balancing variables, which are required to balance the main game, are passed to the selected minigame and the minigame loads the appropriate difficulty level based on received variables. Finally, the player returns to the main game after the minigame has finished.

In the next sections I investigate the above steps in detail and reveal the role of time in balancing. I will also discuss the concept of timing in games and will provide some examples of commercial games and different time parameters that game designers have used to manipulate the difficulty level of the game. Finally, I investigate the relation between time balancing of the games and minigames.

3.1 The Concept of Time in Computer Games

Play time represents the actual time taken to perform a specific activity in a game. In abstract games such as Checkers or Tetris, players play the game in real time and different moves can be thought of as happening instantaneously. The mathematical model for these games is based on finite-state machines where players start the game from an initial state and try to either reach the final state (or force the opponent to reach it) faster. Technically, most games are discrete finite-state machines. For example in Chess, the position of the pieces defines the state of the game. The initial state of the game is when all the pieces are arranged at their first positions in the chessboard. Then players try to proceed in the game by moving pieces to reach the final state as quickly as possible.

In first person shooter games like *Quake III Arena* [57] or *Unreal Tournament* [58] players experience *duality*: the player exists in the real world and as a character in the game world [10]. As Juul [59] has suggested, I use term *event time* to indicate the time of events happening in the game world distinct from the actions the player takes in the real world.

In most action games and in traditional arcade games, the relation between play time and event time is presented as identical. For example in Quake III Arena, pressing the fire key or moving the mouse appears to immediately affects the world inside the game. In fact there is a small delay beyond human perception where the input alters the digital game state. *SimCity* [60] – an open-ended city-building game - provides another example of the concept of play time and event time. Game events, such as building houses, happen faster than in the real world, and minutes of real world playtime might equal to a year in the game world. The relationship between play time and event time can be described as a *mappings*; meaning that the play time and event time are projected into a game world. In fact, the play time is mapped to the event time relative to the speed of the game. For example, constructing a house may takes two days in the game (event time) while it takes one hour in the real world (play time).

Most action games tend to have a direct mapping of the play time to the event time to facilitate the feeling of urgency and action pacing. In some games such as *The Sims* [61] – a strategic life simulation game -, the player can select the game speed, which specifies the relation between playtime and event time, for example to increase the speed at night when their avatar is asleep. The real-time strategy game StarCraft – a military science fiction real-time strategy game, is set in space, and the player doesn't have strong expectation for the speed of the different

events in the game (for example, an activity in the game that is expected to be done in two days in the real world, only takes one hour in the game's world). As a result, the play time can be mapped to event time with different relations.

The capacity to map play time to event time is critically important in games with different game worlds, particularly mixed-reality games. In all of the above examples, the play time denotes the actual time that players spend performing an activity in the game and event time is the time taken for specific events in the game. Mixed-reality games also must account for the time that a player spends to complete game tasks in the real world. Mixed-reality is a term indicating games that include both *real* and *virtual* components at the same time, and consequently game events and tasks are divided into two groups, *Real events* and *Virtual events*.

Real events are those activities that a player performs in the real world, such as moving from one location to the other or taking a photo, and *Virtual events* are actions that are only defined in the game world, such as killing a virtual enemy, constructing a building or training an army. Game designers have almost absolute control over the timing of events that occur in the virtual world but almost no control over those that occur in the real world. In mixed-reality games, unpredicted events during the game play can possibly break the balanced connection between the real and virtual worlds and consequently collapse the whole game because either the real world's or virtual world's players (or both) cannot proceed in the game.

A fundamental requirement of MMR games is the ability to synchronize events in the two worlds (Real and Virtual) and map their events and activities to a shared component in the game. As I will mention later in this chapter, there are several different ways to manipulate time of events in games, such as speeding up/down the movement of certain pieces, but time in the real world is not manipulable by game designers and depends on players' individual skills.

To address this issue, many game designers try to design a virtual world similar to the real world, meaning that they try to provide direct mapping between the virtual event time and real event time: for example, if a player moves from one location to another location in real world, the player's representation in the virtual world of the game moves similarly but with a different speed. For example in Can You See Me Know [10] the virtual world is a simplified map, which is directly mapped from the real location of the game in the real world and the speed of the avatars of players in the game is equal to a fraction of the actual speed of the players in the real world.

It is possible to map the player's movement in the real word to a different action in the virtual world of the game, but still what matters here is the mapping of these two worlds to each other. For example in Pop&Dodge [62] when players jump in the real world, their avatars in the virtual world dodge the balls by sliding to right and left. Finding shared components between real and virtual worlds and identifying the appropriate mapping between the required time for real events and virtual events is a complicated process. Game designers try to reduce the complexity of the game activities and constrain them into a limited set of basic actions in the real world, such as moving in a playground or pressing a button. This simplification limits the creative scope of the game, because designers are limited to simple actions. Moreover, they are forced to use similar game worlds in both real and virtual modes. Using ATMs is a novel way to address this issue. Using ATMs not only makes the game more interesting and fun, but also solves the unavoidable temporal asymmetries that exist in any type of game, especially mixed-reality games.

3.2 Identification of Parameterizable Game Elements

In computer science, the *Time Complexity* of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input [63]. But in computer games, *Complexity* is the number of steps needed to solve an instance of a puzzle in a game. Consequently, when I talk about the time complexity of games, I refer to the total time that is required to take all the necessary steps of a solution in a game. Time complexity of computer games varies with the state-space of the game, the possible set of states reachable from the current state.

Games are usually complex activities that are dividable into several simpler tasks, allowing the measurement and manipulation of the total complexity by modifying the complexity of their constitutive tasks. My definition of time complexity in computer games proposes that the completion times in games can be measured by finding the number of required steps to reach the goal and summing the time of each step [64].

As described in the related work section, substantial research has been performed on using parameter manipulation or selection to generate games of a specific difficulty [65] or for balancing player abilities [66]. These parameterizable game elements can also have varying effects on how long the game takes to play. In *Pong*, for example, the speed of the ball has a relatively straightforward effect on the time needed to reach a set score, but the speed of the

paddle has a more complex relationship with game time, as a faster paddle allows the player to reach more shots and extend the rally, but may also increase the number of player errors.

In general, *game size* is one of the parameters that affects game time. *Game size* refers to the size of the game in terms of number of simultaneous ways of doing a specific task (*Action Width*) and the length of each task (*Action Length*). *Action Width* implies the number of available ways to perform an action in a game. For example in Neverhood Chronicles [67] – a point and click adventure game - the rat puzzle (Figure 2) offers several solutions simultaneously to the player. The goal of the game is to guide the mouse to the cheese and the number of possible paths directly affects the difficulty of the game. The *Action Length* implies the time that an atomic action in the game takes to be completed. In one of the scenes in the game, there is a very long route that the player has to take to reach an important object in the game (Figure 3), requiring an unavoidable minimum round trip time of 6 minutes irrespective of players' actions.

Figure 2: Neverhood Chronicles: The cheese and rat puzzle

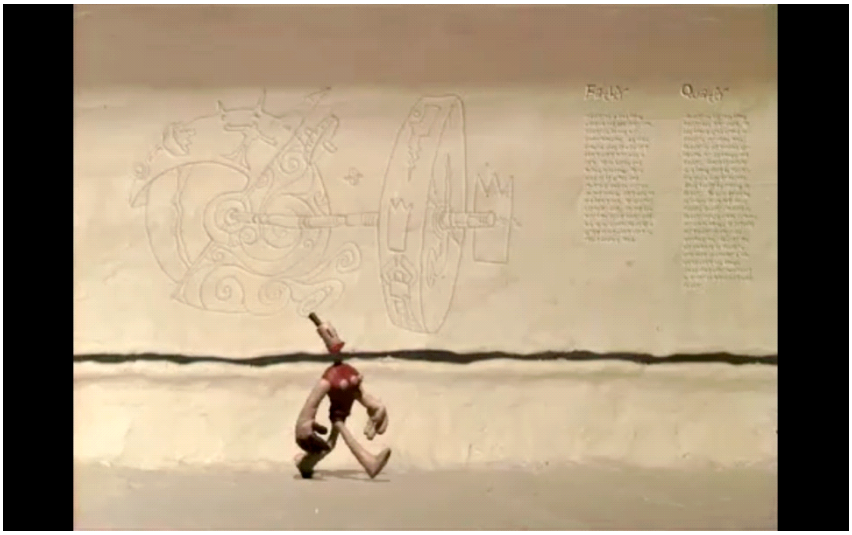

Figure 3: Neverhood Chronicles: Long path to find a hidden object which is required to complete the game.

There are also situations in which game designers use repetitive simple tasks to increase the difficulty of the game. Figure 4 shows the final battle of Sonic II [68] – a platform game in which the player characters are two hedgehogs. The game player has to jump over and beat the giant robot. The giant receives damage on every jump. In this example killing the giant is the overall mission and is done by repeating a smaller task (jumping repeatedly over the giant). When the player jumps over the giant, the game allocates damage and calculates the remaining life of the giant. Then it returns a value representing the remaining life of the giant and the color of the giant is changed to show the player how much progress has been made. The number of jumps is a fixed value which game designers, considering the desired difficulty, set before the play starts. Another example is Lost Planet [69] (Figure 5) – a third person shooter game that happens on a fictional planet - where the player has to kill the giant worm by shooting its energy sources (big yellow dots). Every time the worm jumps out of the ice, a cool-down timer is started and the player has to shoot the worm's energy sources, otherwise when the timer is up, the worm returns to the ice and those resources that are not destroyed completely will be reset, prolonging the battle. In this level of the game, the worm receives a wound in every shot of the player. The game receives the worm's damage and calculates the level of the damage by accumulating all the worm's wounds while it has been above the ice. It then breaks the worm's resources to show the player how much progress has been made. If the total level of damage is more than the total

health level of the worm, it is killed and player wins the battle. On the other hand, if the total level of damage is less than the total health level of the worm, and the cool-down timer is up, the game resets the damage level of the worm and it returns to the ice.

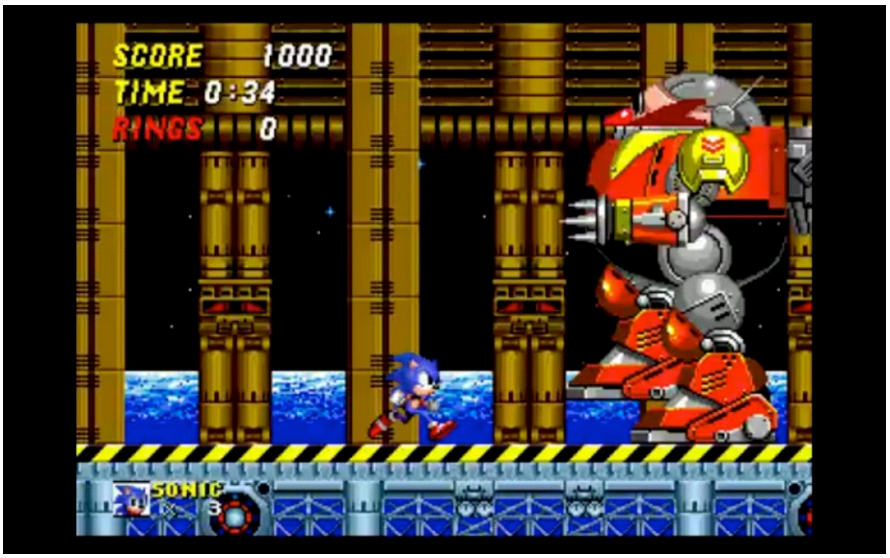

Figure 4: Sonic 2 (Final boss scene)

In this example the main action is killing the giant worm, which is divided into a number of subtasks, shooting the worm's energy sources. Each subtask must be finished in a given time (while the worm is out of the ice). It is not specified that how many times the player has to shoot at the worm to kill it and complete the overall action, as this depends on the skill of the player. However, it is possible to specify a minimum time for a perfect marksman to complete the task, introducing the concept of a minimum completion time, something I will draw on in ATMs.

Figure 5: Lost Planet 1 (Gigantic worm scene)

In general, the level of control affects the difficulty level of the game. The control level measures how much control the player has over the character in the game. For example in Classic Mario [70] – a single player platform game - the possible actions of Mario are limited to moving right, moving left, jumping and shooting (Figure 7). On the other hand, in Neverwinter Nights 2 the player controls one character in the game (Figure 6) with numerous options. In this game players receive a set of actions, skills and weapons for their avatars. In every fight scene of the game, players are capable of choosing between several actions, skills and weapons, each of which has a specified level of damage. Each weapon has a specified amount of damage and speed, which lets game designers manipulate the total power of the weapon. Although a player controls only one character in the game, having several options for play increases the complexity of the game.

Figure 6: Neverwinter Nights 2

Figure 7: Classic Mario

Most of the time, modifying one parameter of a complex action in game will affect other aspects. Whenever timing of tasks in a game is crucially important, it is hard to calculate the complex actions' total time. Minigames, which typically have simple mechanics, can provide a method for specifying precise timing control within a larger game.

3.3 Game Balance as a General Concept

As previously mentioned, game balance is a technical term representing the fairness of the game and should not be confused with subjective measures such as fun. The first step of game balancing is to recognize the balancing methods of the game. For example in a single-player game, the term balance is mostly used to indicate whether the challenge level of the different tasks in the game is appropriate for the current players, whereas in multiplayer games balance indicates the overall fairness between players.

Setting the challenge level is a fundamental game balance problem. Although it is possible to state that a challenge level should be higher for skilled players than novice players, it can be difficult to specify what is easy and what is hard for a particular game or mechanic. The standard way to address challenge balancing is playtesting, because it reveals different players' behaviors, but even playtesting is not a comprehensive solution because not all players are exactly the same and not all strategies can be explored by playtesters for a complete game.

One approach to this issue is to test the game with a wide range of test players. Usually playtesting gives game designers a chance to analyze players' performance and create progress charts and statistics, but there is still one question that remains unsolved: because not all players are the same, how should different players be ranked with respect to the result of playtests.

In multi-player games, players play together and the balancing concept changes. Multi-player games are naturally asymmetric which means different players of the game are not equal in terms of skill and experience, making multiplayer games harder to balance. Game designers often employ different game components to adjust the fairness, for example by changing the starting point, certain resources, or character's state. In multi-player games where more than one strategy exists, the advantage of each strategy should be clearly balanced. Similar concepts work for resources in games as well; in a balanced game the cost and benefit of resources in the game are fair, so controlling a particular resource would not destabilize the game balance. Two resource pools are balanced when they have similar cost and benefit for the players.

In general there are four ways to balance games:

1) By using the experience and instinct of the game designer; in this way the game designer tries to play the game several times until it feels right. Unfortunately this method is not reliable because, while an expert, the designer is biased.

2) By calculating the relationships between the components of the game to ensure that every entity in the game has the appropriate cost and benefit. Although this method of balancing is reliable in terms of the correctness of the formulas, it is hard to calculate all the possible situations of the most games. Additionally, any errors in assumptions underlying the formulation can cause the balance to fail catastrophically when violated.
3) By playtesting the game. Similar to the first method, the designer keeps testing the game until most of the players have a reasonably good and fair experience. One drawback of this system is that playtests are usually time consuming. Additionally, the outcome of the experiments is dependent on how representative the test players are of the overall population.
4) By dynamic balancing during game play. In this method the game starts with an initial setting, which has been acquired via one of the above approaches, and the rest of the balancing parameters are adjusted dynamically.

3.4 Minigames as Separated Units in Games

As mentioned previously, to use time balancing in games, a game designer should be able to assign completion time to a set of activities in the game, implying that there should be a time chart that shows how long specific activities in the game take. Obviously, better players should be able to complete the task faster than weaker players, but the variance should be modest to increase competitiveness, and the mean should coincide with the duration the designer desires. Although time manipulation can be used for many design elements in games (e.g., to artificially synchronize player action [15]) my focus here is on time manipulation as a player-balancing tool. Minigames have several parameters that can be manipulated to speed up and slow down the playtime. Although there are several ways to manipulate the playtime such as changing the game size, these variations have side effects, meaning that if I manipulate one component of the game, all the other components of the game that are related to the modified component will be affected. For example, suppose that I have two games: one with numerous simple activities and the other with a few hard activities. It would be difficult to compare the playtimes of these games without empirical data.

The simplest example involves controlling the scope of a repetitive task, such as shooting asteroids or aliens, where the number of times the task must be repeated changes. Another simple

example is the manipulation of game physics (or physics analogues) to increase or decrease the speed of active components: for example, increasing the speed of falling bricks in Tetris can allow faster completion times because the blocks cross the screen faster.

3.5 Minigames and Time Balance

As a part of time balance using minigames, the designer must identify elements and mechanics in the minigame that affect completion time, and must determine the parameterization of those elements. Also, the designer must determine which elements should be adapted at the start of the minigame, and which can be adjusted dynamically during play.

Minigame-based time balancing can be divided into two phases: a static phase and a dynamic phase. The *static phase*, which occurs before the minigame starts, sets the minigame's parameters and mechanics to satisfy an anticipated time constraint –potentially determined by the main game state. For example, elements such as the size of the game, the number of levels to complete, or the starting difficulty can all be set before the minigame begins. To do this, one of the above mentioned methods, such as experience of the game designer or playtesting, can be employed to determine this initial setting. In the *dynamic phase*, dynamic balancing is achieved by periodically comparing game state to an a priori desired state, and adjusting one or more parameters of game elements such that the completion time of the minigame will approach the desired time.

Although time balancing of games with minigames sounds simple, it has several complexities that a game designer should address. Figure 8 shows the general concept of time balancing using minigames. As shown, two players with different skill play a multi-player game together. Suppose that player A is much more skilled than player B, and the game starts at the same time for both players. The game is a simple running match and players have to finish the path as fast as possible. Every red point in the paths represents a station where players have to stop and rest.

At each station, player will be given a minigame to play. Minigames start with an initial setting, the small rectangles tagged as "Static", which have been set prior to the minigame. This is what the game designer has set based on experience, playtesting or mathematical calculations. This amount is consistent until the game is finished and can be used as initial difficulty level of the game. The dynamic part of the minigames, the rectangles with variable size and tagged as "Dynamic", will be activated during the game play to help the underdog player and make the

game harder for strong players. By changing the difficulty of the game dynamically, the game will be more challenging for the stronger player, and less frustrating for the novice player. It is worth mentioning that the above setting is still subject to change. For example, if the game would be too easy for the underdog player, the adaptation mechanism makes it harder again.

Although the setting of the minigames in the above example is dynamically modifiable, it is also possible to change the frequency of the minigames during the main game. For instance, the game could trigger more minigames for the stronger player. To be able to change the difficulty level and the frequency of minigames, game designers should be able to address the two following questions:

1- How frequently should minigames be used?

2- How much change in the difficulty level of the minigames is appropriate?

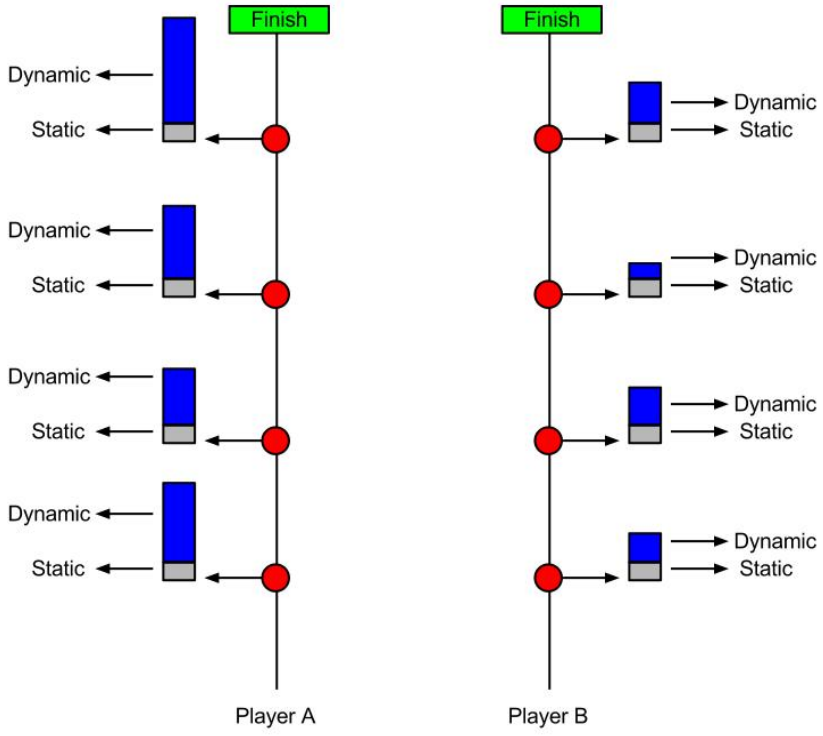

Figure 8: General concept of game balancing using minigames

There are several ways these adaptation decisions can be made which will be discussed in the next chapter.

CHAPTER FOUR
ATM: ADAPTIVE TIME-VARIANT MINIGAMES

Two main issues regarding game time balancing using minigames are addressed here: First, if minigames are used to perform the time balancing in a game, how often should minigames be updated? Second, how much should time vary within a single minigame?

The following experiments were performed to evaluate these issues:

1- To find the optimum frequency of adaptation, three different adaptation speeds: *Discrete (One-shot) Balance*, *State Balance* and *Continuous Balance* were investigated.

2- To find the appropriate amount of manipulation in minigames' game mechanics, parameters that affect player experience were tabulated. In this research I focused on Aggressiveness, Number of Elements, and Interaction against noticeability of the modification.

3- A progress-vs.-completion model called *Temporal Exemplar* was created to acquire a reusable model of players' experience while playing my minigames.

In this chapter the *Temporal Exemplar Model* is discussed first. Next, The effective parameters on players' experience and introduce the three different balancing frequencies is examined next. Finally, four different minigames that are compatible with time balancing algorithms are presented and I discuss their performance when different balancing algorithms are employed.

4.1 Temporal Exemplar

As mentioned previously, one of the issues common to every balancing algorithm is that the game designer does not know what constitutes on easy, medium and hard difficulty for the game. To address this issue and to be able to deliver a particular completion time, the system must have a model of how long the minigame should take. This model can be as simple as a single completion time value or more complex if techniques such as continuous adaptation are to be used (discussed later in this chapter).

To find how different players progress in the minigames, I developed an exemplar model for each minigame: by asking several people to play the minigames without any adaptation, and creating a time-vs.-progress model from the averaged data (see example in Figure 9). Players were asked to finish the game as quickly as possible, using as fewer resources as possible.

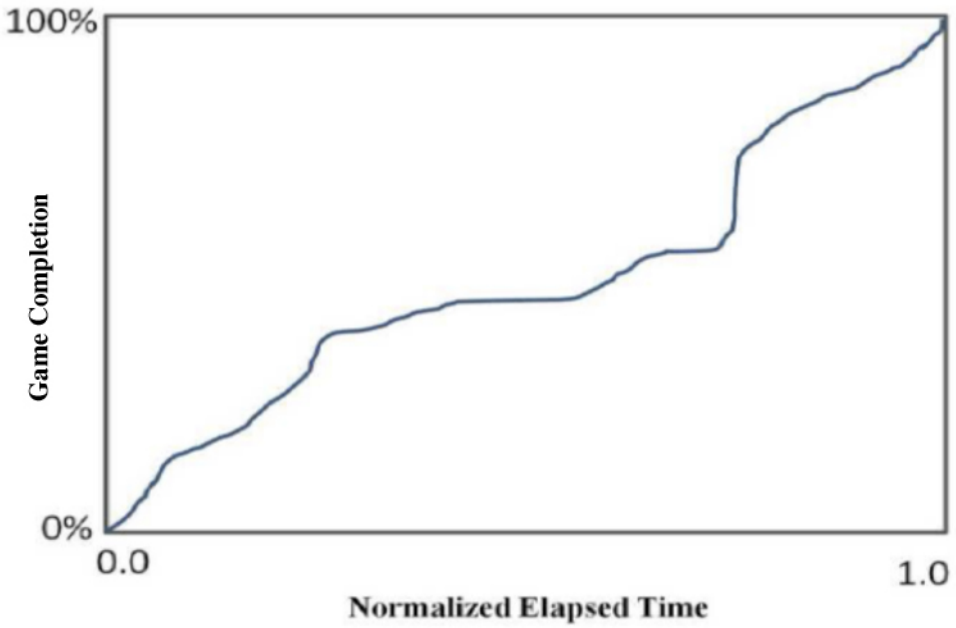

Figure 9: Sample Time-vs.-Progress Model

Using the time-vs.-progress model, every point of the adaptation algorithm to the model can be mapped, allowing for the entire or a subsection of the minigame to be estimated. Expert design or mathematical derivation could also have been employed, but exemplar data was chosen as the least likely to confound subsequent experiments.

4.2 Important Parameters in Noticeability of Adaptation Algorithms

The adaptive algorithm controls the type and magnitude of adaptations. These algorithms compare the player's current performance to some model of desired performance. Within this general class, adaptive algorithms can still vary across several characteristics. For example:

- *Aggressiveness*: the algorithm can be more or less aggressive in correcting a disparity between the player and the ideal. For example, Bateman and colleagues noted that

cautious adjustments were sometimes not able to make up a disparity before the game finished [35].

- *Number of elements*: algorithms can change a single parameter of a single game element at a time, or can change several simultaneously. Changing multiple elements can reduce the noticeability of adaptation in game, but can also be more difficult to model.
- *Interaction with game narrative or appearance*: algorithms may attempt to make their adaptations less noticeable by interacting with the game narrative – a change to an element's parameter could be explained through additional narrative elements (e.g., there are more enemies to defeat because reinforcements have arrived; the ball is moving slower because a penalty brick was hit).

4.3 Frequency of Adaptation Algorithm

In addition to the characteristics of adaptation algorithm mentioned in previous section, the *frequency* at which adaptation decisions are made is a critical part of the adaptive algorithm. Game state adjustment could be continuous, such as the continuous adjustment of traffic load in *Need For Speed: The Run* [71] to adapt the challenge level, or could be discrete, as with the preferential distribution of power-ups in MarioKart. *Frequency* of adaptation plays a major role in this process, because the granularity of adaptation can dramatically affect noticeability.

4.3.1 Discrete (One-Shot) Balance

In this method, players play the game with the starting parameters until a preset duration is exceeded, then a single immediate adjustment in balancing parameters occurs. The preset duration can be any value chosen by game designers. In this research I have evaluated this balancing method with two different targets: minimum completion time and average completion time.

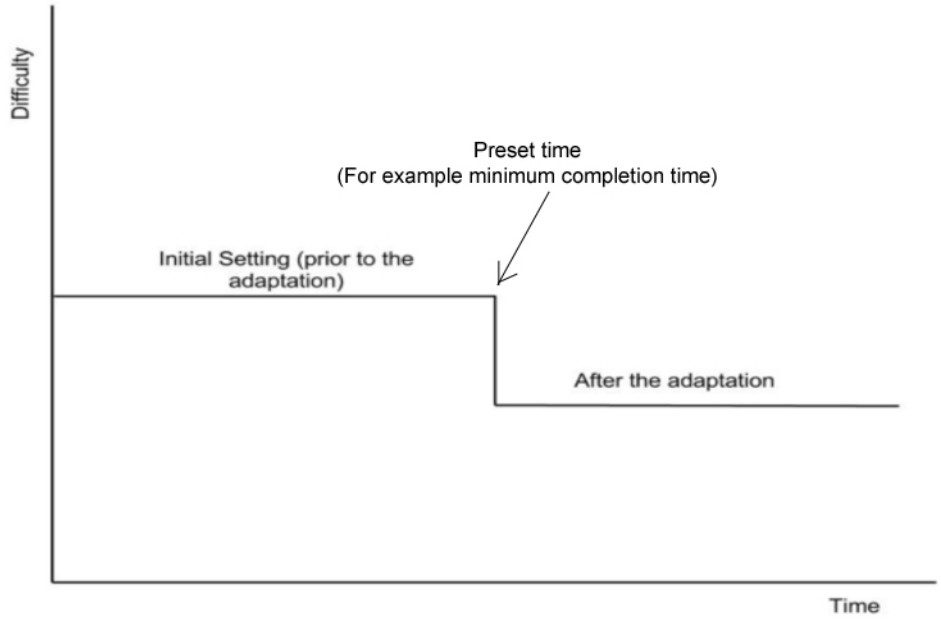

Figure 10: Variation of game's configuration based on time

The adaptation algorithm is employed only once, and it dynamically changes other components of the game to compensate for the latency of players (Figure 10). The adaptation algorithm starts when the minimum time (or any time which is set prior the game starts) is passed. Some of the advantages of this method are:

1- The Discrete Balancing is simple and easy to implement. It requires the least amount of information about the game state.
2- It can provide a minimum completion time in conjunction with game mechanics because adaptation will not occur until a minimum time is reached.

Although this method does provide a degree of dynamic balancing, it has the following shortcomings:

1- The minigame takes a minimum amount of time to complete, which would reduce flexibility if a minimum time is undesirable.
2- The adjustment may be too coarse, making it difficult for novice players, and also more likely to be noticed by players.

These issues can be addressed by employing a fully dynamic adaptation that modifies the adaptable elements of the game as play proceeds. In the next two balancing methods, State balance and Continuous Balance, I employed temporal graining and removed the minimum time constraint.

4.3.2 State Balance

In State balance, a player's performance is compared with an exemplar every time a particular game state changes (e.g., a subtask is completed). For example, in a puzzle game, players should assemble all the pieces successfully to finish the game. Putting each piece of puzzle into its correct place is a subtask. In a state-based update, balancing parameters would be recalculated after every puzzle piece was placed.

Using collected times for each state change, a progress-vs.-time model can be implemented (see section 4.1). The total completion time is then the sum of the completion time for each subtask.

For example, in Figure 11 each milestone represents a successful piece placement in a puzzle game in the time-vs.-progress model and the black line shows how a new player has played the game. When the player reaches to the first milestone, the State balance algorithm is called, which checks whether the players is ahead the time-vs.-progress model or not. If the player is faster than the reference, the State balance algorithm manipulates the game components and makes it harder, for example decreasing the mouse speed. In this case, the player was slower than the model, the State balance algorithm makes the game easier to let the player progress faster. This process repeats at each milestone and tries to make the new player's total completion time as close as possible to the total completion time recorded in the progress-vs.-time model.

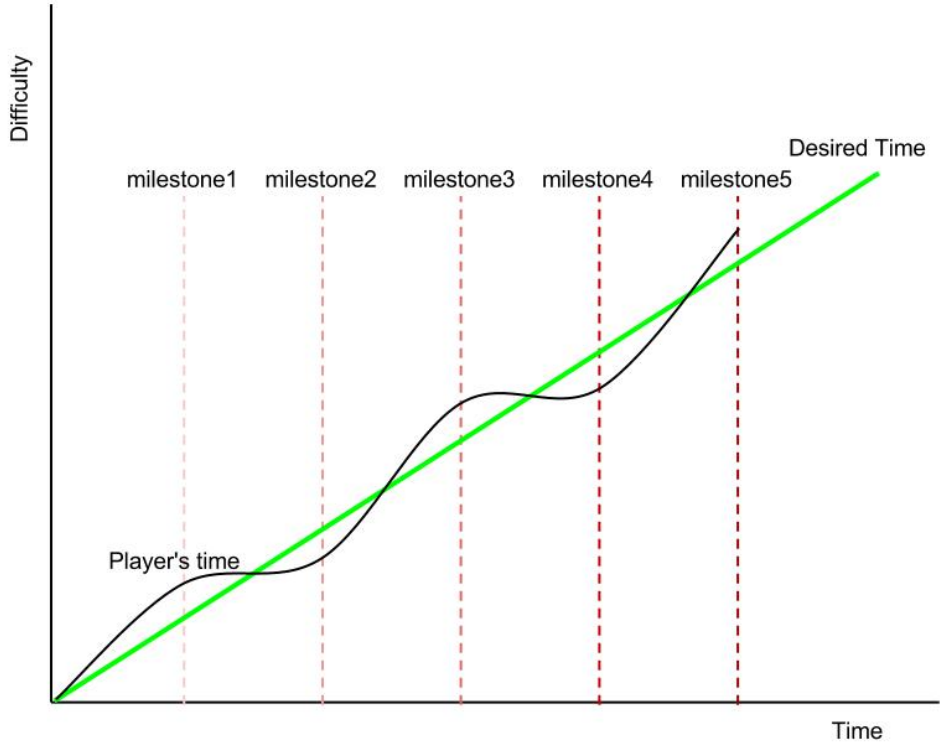

Figure 11: State balance time-vs.-progress exemplar: The player's time swings around the desired time that the designer has set in design stage of the game

4.3.3 Continuous Balancing

In Continuous balance, a player's performance is compared with an exemplar at regular intervals (usually a factor of the game's heartbeat) (Figure 12). The continuous method is a balancing method with a finer granularity, able to detect a change in game state smaller than a subtask. I called this balancing method "Continuous" because the intervals employed were much smaller than human perception, appearing continuous to the player. In this method, similar to State balance, I use a progress-vs.-time model to decide how to balance the game.

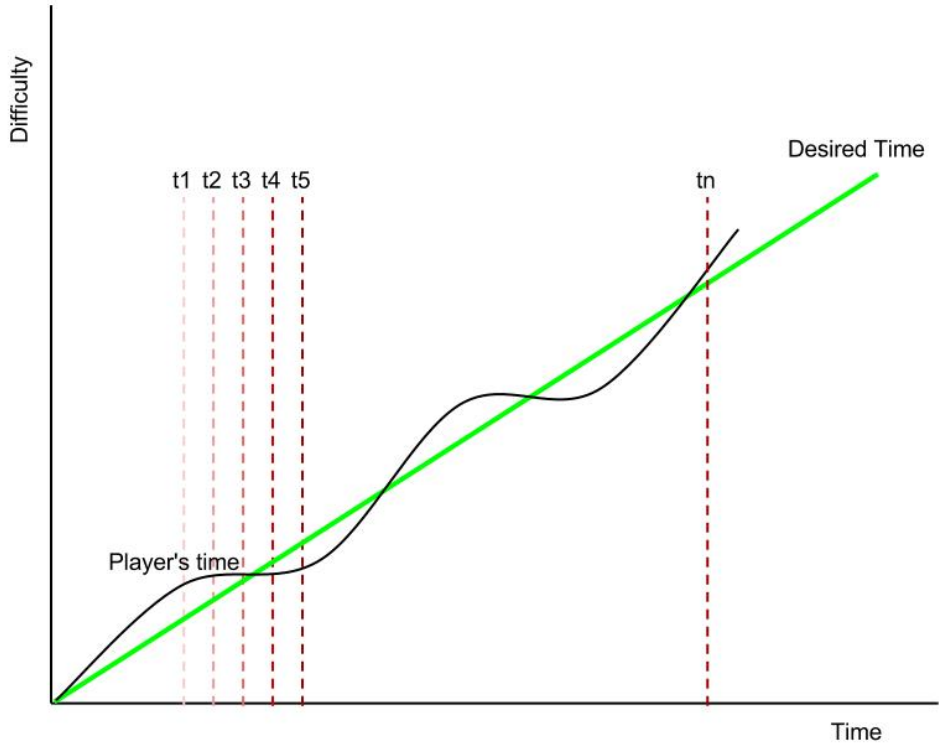

Figure 12: Continuous balance time-vs.-progress exemplar

There are some differences between Continuous balance and State balance:

1- In State balance, the granularity of the progress-vs.-time model (milestone in Figure 11) is equal to every activity in the game that changes a subtask level element, for example placement a piece in it's place in a puzzle game. The granularity in Continuous algorithm is usually a multiple of the game heartbeat, and balance is recalculated regardless of subtask state.

2- In State balance, the balancing algorithm tries to make the game state as close as possible to progress-vs.-time model at the same milestone independent of total completion time. In Continuous balance algorithm, compares the current state of the game with the in progress-vs.-time model as frequently as possible and, if the difference is more than a threshold, the balancing algorithm manipulates the balancing parameters to compensate.

.

4.4 Four Example Minigames

I used four minigames to test the efficacy of my time balancing algorithms: Spinning Puzzle, Electris, Click-And-Hack, and BrickOut. Each game has both static and dynamic balancing mechanisms. While the primary purpose was to evaluate the dynamic balancing algorithms, I also evaluated two static balancing settings (deployed as two difficulty levels) for each game to ensure that the dynamic algorithms' performance was not specific to a given starting configuration.

All the games were implemented in C# using the XNA framework. Continuous updates were tied to the XNA game heartbeat, of 16.7 millisecond. Each game has manipulable components which game designers can use to modify the total completion time of the minigame. In order to mask change in games were implemented mechanic of the minigames, I tried to reduce the intensity of the change and reflect the change gradually in the minigame. For example, if any change in the speed of a component in a minigame is required, it was performed linearly over a two-second period.

4.4.1 Click and Hack

Click and Hack is a variant of the fairground game "Whack-a-mole." Players must click on the "Hack" button and then quickly click on a computer image that appears at a seemingly random location on the screen (Figure 13). Click and Hack is essentially a Fitts' Law task [72], where the difficulty of the challenge is proportional to the size of the target and the distance from the Hack button.

Figure 13: Click-And-Hack minigame: In this image, the player has clicked the 'HACK' button and a new target (computer) has appeared.

Static and dynamic elements: The static balancing mechanism is the number of targets that must be clicked to complete the game. The combined distance-size tradeoff – generally termed the index of difficulty in Fitts' law studies – is the dynamic balancing mechanism.

Dynamic adaptation method: I used target size and the distance between the "Hack" button and targets as the adjustable parameters. In Discrete balancing, I tried two different methods:

1- During normal game play, the computer can appear anywhere on the screen at a fixed size. After the minimum time is reached, the size of the target will be increased and the game gets easier for players.

2- During the normal game play, the computer can appear anywhere in the "Middle area" (Figure 14). When the player is progressing more quickly than the exemplar and the average completion time of the minigames is reached, I draw targets from a distribution biased to provide more distant targets. When the player is slower than the exemplar and

the average completion time of the minigame is reached, I draw targets from a distribution biased to provide closer targets.

Dynamic adaptation could be also accomplished by reducing the number of targets. For example, when the minimum time is reached, the game finishes and the player is led to believe that the goal has been accomplished; however this method was not investigated here.

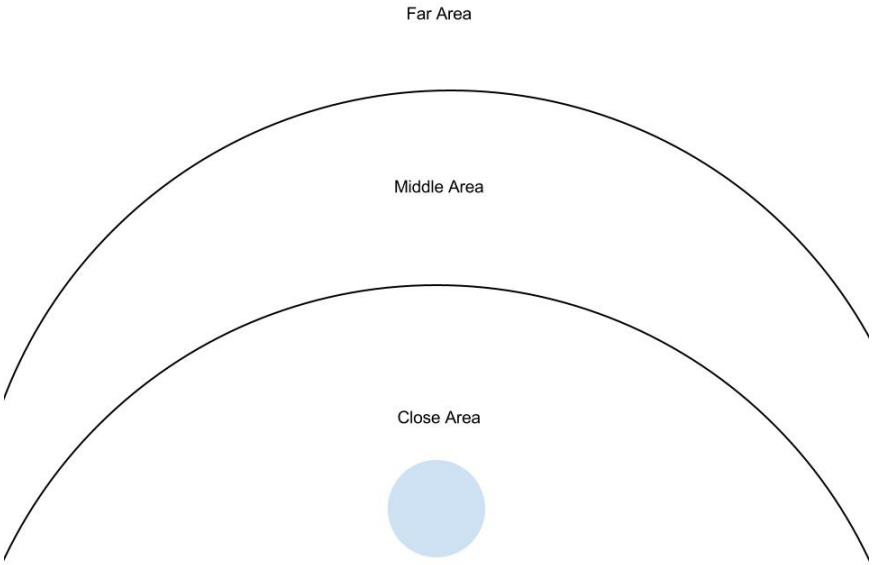

Figure 14: Click-And-Hack: There are 3 different areas; “Close”, “Middle” and “Far”

In State balancing algorithm, clicking on each computer changes the state of the game and causes the State balancing algorithm to compare the player’s progress with the exemplar model. If the player is slower than the recorded time in the model, new target will appear in the “Close area” (shown in Figure 14) to make the game easier. If the player is faster than the corresponding sample in the exemplar model, the next target will appear in the “Far area”, causing the game to be harder.

The Continuous Balancing mode, is similar to State balancing in this case, because it is not possible to modify the game balance any faster than State balancing algorithm without employing mouse trajectory modeling. To adjust the game balance in Click-And-Hack I use the size of the targets, or the distance of the target from the fixed Hack button, which can only update once a subtask has been completed.

4.4.2 Spinning Puzzle

In the Spinning Puzzle game, players must align a series of disks to make a continuous path from a chip to a cooling fan. There is only one solution, so the game poses a similar gameplay challenge to a physical geometric puzzle (Figure 15).

Static and dynamic elements: The static balancing mechanism is the number of disks in the puzzle. The dynamic balancing mechanism is the rotational speed of the pieces.

Dynamic adaptation method: In this game, I modify the rotational speed of the disks dynamically in the game to adjust the balance. In all cases the speed begins at 12°/s (Normal speed) and when needed, I increase it to 18°/s (Fast speed) or decrease it to 8°/s (Slow speed).

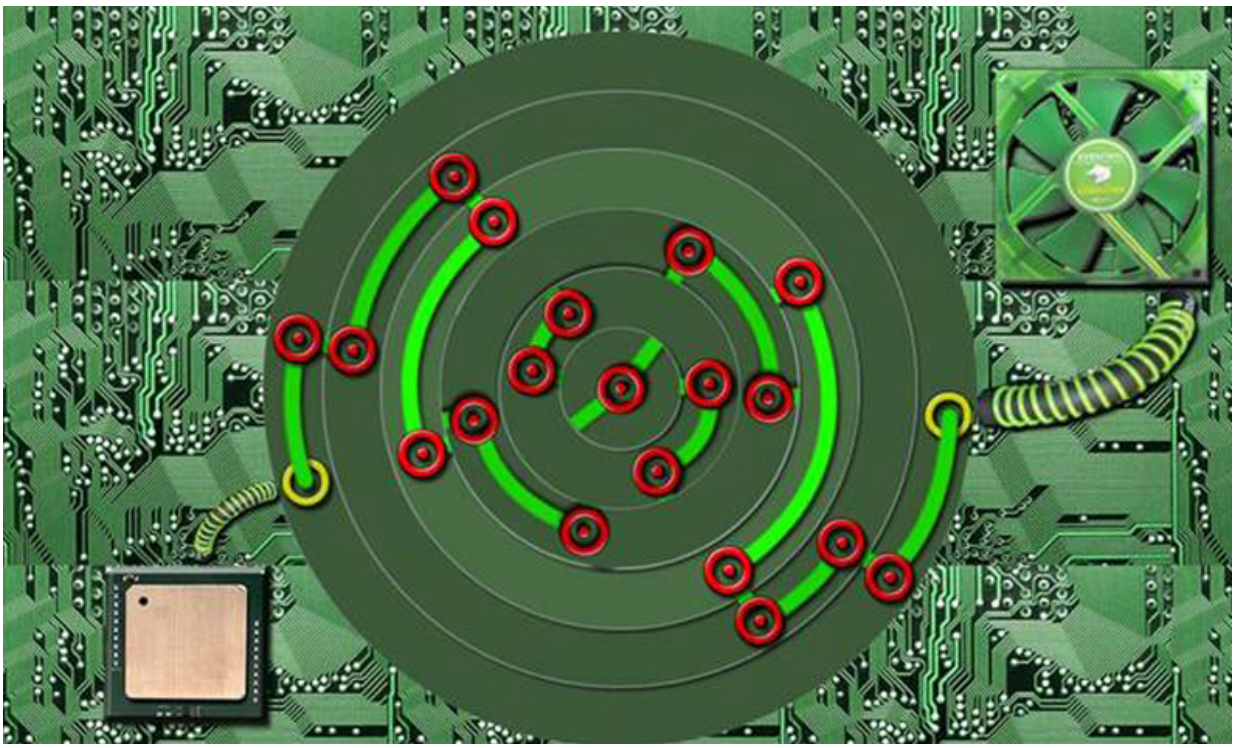

Figure 15: Spinning puzzle: the player turns disks to plug the chipset on the left side to the cooling fan on the top right side. Red circles in each disk represent the pluggable point and yellow circles show the start and end points of the path.

Two methods of dynamic balancing were employed:

1- During the normal game play, the rotational speed of the disk is Normal. When the minimum completion time of the minigame is reached, the rotation speed will be set to Fast speed to make the game easier.
2- During the normal game play, the rotational speed of the disk is Normal speed. When the average completion time of the minigame is reached, the rotation speed will gradually be set to Fast speed to make the game easier, to hide this change from player.

In the State balancing algorithm, the rotation speed is recalculated every time a player moves a disk to the correct location. The number of recalculations for the State algorithm is therefore equal to the number of disks minus one (assuming that players do not move a correctly placed disk to an incorrect position). If the difference of the current game play time for the correctly located disk and the recorded time for the same disk in the exemplar model exceeds the threshold (1 second), the rotation speed will be set to Fast speed for the next disk. If the player is slower than the exemplar, the rotation speed will be set to Slow speed for the next disk.

The Continuous algorithm measures the game state every 16 ms (equal to the heartbeat of the game) and updates the rotation speed at the same rate if necessary. The algorithm compares the current player's game play time with the recorded time in the exemplar model and if this difference exceeds the threshold (one second), it changes the rotation speed. In this case, the State and Continuous cased are different because game State and subtask States are distinct.

4.4.3 Electris

Electris is a variant of falling brick games such as Tetris or Bejeweled. Electrical components fall from the top of the screen down changing their appearance sequentially with every downward step. The player must match a particular electric circuit shown at the top of the screen. Like most falling brick games, the bricks fall at a set rate from top to bottom. While the component falls, the player can move it to left and right with the arrow keys, and can commit the component by pressing spacebar, which causes the component to stop cycling and fall at a faster rate (Figure 16).

Electris is distinct from the other games I studied because there is a substantial error cost. Once a component has been played it cannot be removed easily. To remove an incorrect component the

player has to put another similar component next to the incorrect component to remove it, which may not be possible. If there is no room beside the incorrect component, it negates an entire row and requires the player to finish filling the row so they can begin a new row on top.

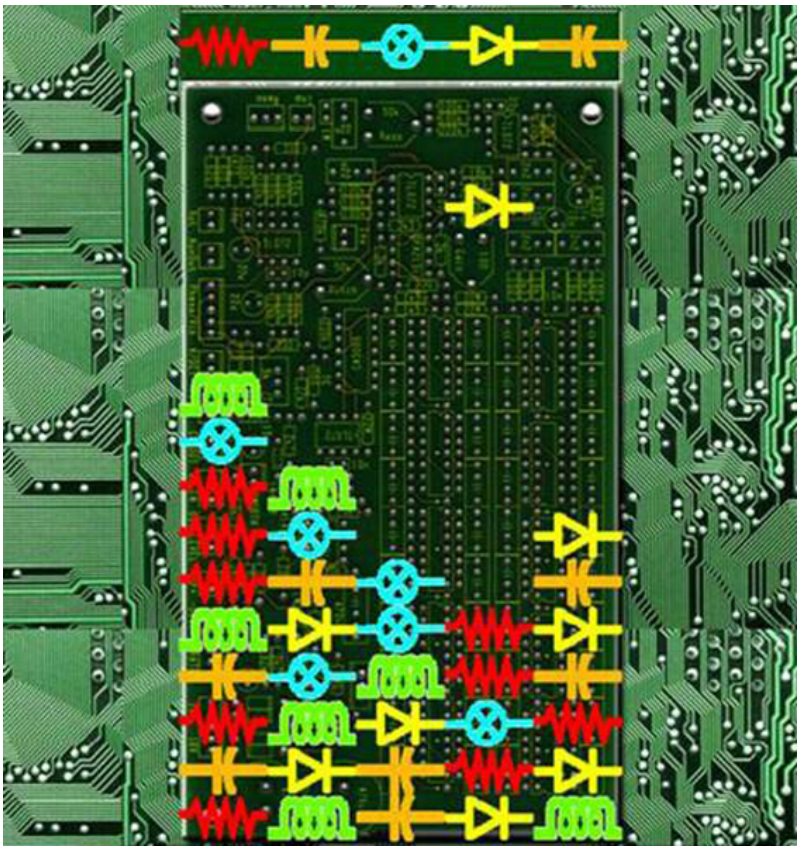

Figure 16: Electris: The player is trying to create the same pattern as shown on top of the screen, but has made several mistakes and lost the very first rows.

Static and dynamic elements: The primary static balancing mechanism is the number of rows that must be completed. The primary dynamic balancing mechanism is the speed at which pieces fall after the spacebar is pressed.

Dynamic adaptation method: In this game, I changed the falling speed to adjust the balance of the game. In all cases pieces fall at a rate of 120 pixels/second (Normal Speed) and when it is required, I increased this rate to 360 pixels/second (Fast Speed) to let players to progress faster, or decreased to 80 pixels/second (Slow Speed) to prevent players from finishing the game too quickly.

Similar to the previous minigames, in Discrete balancing two different methods were employed:

1- In the normal setting, pieces fall at Normal speed until the minimum completion time of the minigame is exceeded, and then pieces fall at a Fast speed.
2- In the normal setting, pieces fall at Normal speed until the average completion time of the minigame is exceeded, and then pieces fall at Fast speed. To intertwine the adaptation with the narrative, the color of the background varies (red for Fast, green for Normal, blue for Slow – Figure 17). The goal of this visualization is to hide the adaptation process from the player's perspective. Players were told that the variation in falling speed is a random event in the minigame.

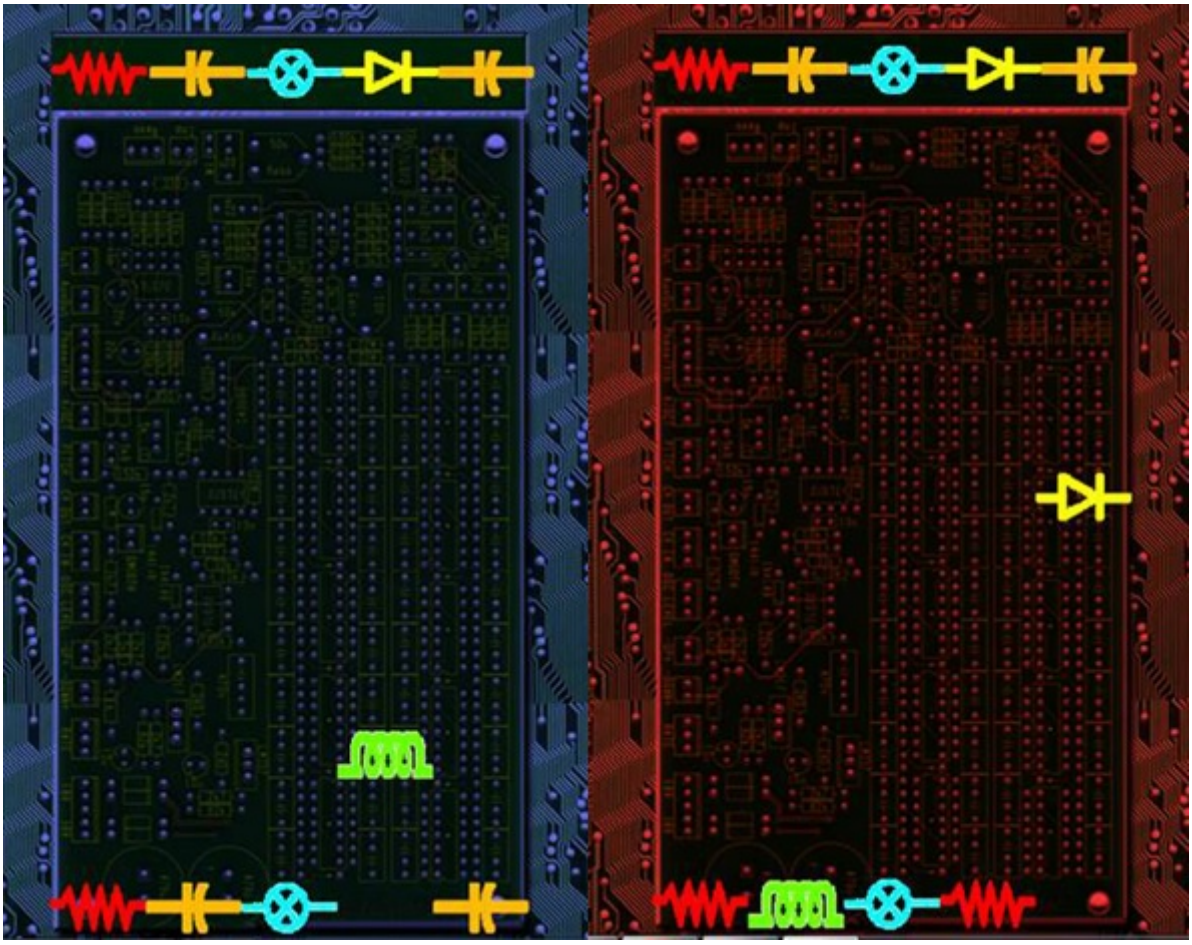

Figure 17: Electris: On the left side the red background shows that the game's speed has increased, while the blue background represents the slow mode of the game

In the State balancing algorithm, the falling speed is set based on the number of correct pieces placed when compared with the exemplar. Every time the player places a piece correctly, the State balancing algorithm compares the current play time of the game with recorded time in the exemplar model. If the player is slower than the model, the next rate of the falling for the next piece will be set to Fast speed. But if the player is faster than the exemplar model, the rate of falling for the next piece will be set to Slow speed.

In Continuous balancing algorithm, the current play time of the game is compared with the recorded time in the exemplar model on every 16 ms and the falling speed will be adjusted based on the difference between these two times to one of the Fast, Normal, or Slow values. In State balance mode, the rate of the falling cannot be changed between placements of pieces. If a player places a piece slower than the exemplar model, the rate of falling will be set to Fast and it remains consistent until the next piece is placed correctly. In Continuous mode, based on whether the player is faster or slower than the exemplar model, the rate of falling will be adjusted.

4.4.4 BrickOut

In BrickOut, the player must guide a bouncing ball such that it hits a series of bricks at the top of the screen. Bricks disappear when struck, and the game is complete once all the bricks have been eliminated (Figure 18). BrickOut represents a baseline for the other styles of adaptation because the brick count (and therefore the total distance travelled) and ball velocity represents the most direct mapping to total time as the ratio of distance and speed.

***Static and dynamic elements*:** The static balancing mechanism is the number of rows of bricks. The dynamic balancing mechanism is the speed of the ball.

***Dynamic adaptation method*:** In this game I change the speed of the ball to be able to adjust the balance of the game. Normally, the ball moves at 20 pixel/second (Normal speed). The Slow speed was 10 pixel/second and the Fast speed was 30 pixel/second.

Figure 18: BrickOut minigame: The player guides the ball to bricks to eliminate them as fast as possible.

In Discrete balancing two methods were employed:

1- When the game starts, the ball moves at Normal speed and when the player exceeded the minimum completion time the algorithm increases the speed of the ball to let the player progress faster.

2- The initial speed of the ball is set at Normal speed. When the player exceeded the average completion time of the minigame, the ball's speed gradually increased to Fast. The ball's color changes when the speed is adapted using the same scheme as the Electris background (Figure 19).

In the State balancing algorithm, the balance is updated to integrate the change in to the narrative. Every time a brick is hit and the speed of the ball is changed on the rebound. If the player is slower than the recorded time in the exemplar model, the ball speed will be set at Fast, if the player is faster than the recorded time in the exemplar model, the speed of the ball will be set at Slow. All the speed variations are reflected gradually in the game to hide the balancing process from the player's perspective.

In Continuous balance mode, the speed of the ball is continuously and gradually adjusted based on the difference between the current play time and the recorded time in the exemplar model. In the State balance algorithm, the speed of the ball is adjusted when a brick is hit and remains consistent until the next brick is hit. But in Continuous balance mode, every time that the difference of the current play time and the corresponding recorded time in the exemplar model exceeds the threshold (one second), the speed of the ball is adjusted.

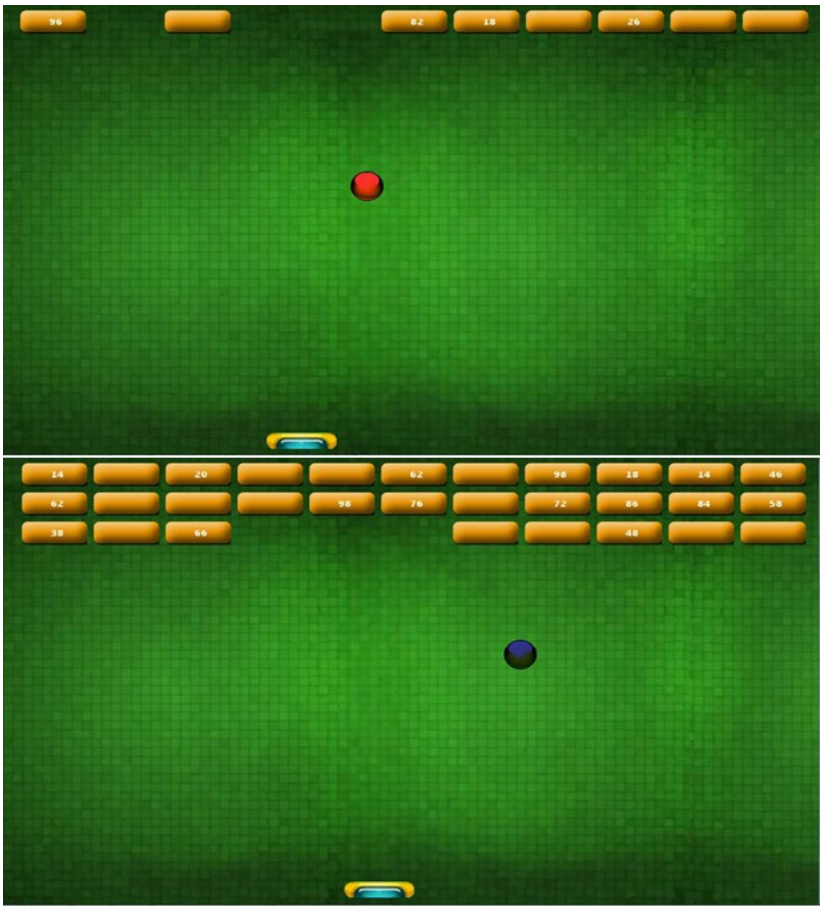

Figure 19: Brickout minigame: in the top game the ball's color has changed to red to show the increased speed, while in the bottom image the ball is moving relatively slowly and its color is blue.

4.5 Minigame Completion Times and Game Mechanics

As mentioned earlier, each minigame includes two different types of elements: static elements and dynamic elements. Static elements of the minigames help game designers adjust the balance of the initial condition of games. For example if the game designer decides to use the Spinning puzzle minigame, the number of disks can be used to set the difficulty of the minigame. I used "static" term for these elements because they are set initially and are independent of a player's subsequent performance in the game.

On the other hand, the dynamic elements can be changed during game play and therefore the difficulty level of the minigames is adjustable. Table 1 shows all the static and dynamic elements of the minigames, elements that are used for initial settings and mathematic formulas that are required for all calculations.

In principle, other components as diverse as *play area* or *number of avatars* or *simultaneity of tasks* could be utilized as manipulable parameters, but I only list those parameters that were actually implemented.

Game	Manipulable Components	Initial Settings	Adaptive Component	Minimum Time
Click And Hack	Mouse speed, target size, number of targets, distance	Target size, # of targets	Target size, distance	$\sum_{i=1}^{n} \frac{2x_i}{v}$
Electris	Piece speed, number of lines	Number of lines	Piece speed	$N \sum_{i=0}^{n} \frac{h_i}{v}$
Spinning Puzzle	Rotation Speed, number of disks, min. # turns required	Number of disks	Rotation speed	$\sum_{i=1}^{n} \frac{\theta_i}{\omega}$
Brickout	Number of bricks, speed of ball	Speed of ball	Number of bricks	$\sum_{i=1}^{n} \frac{2h_i}{v}$

Table 1: Minigame parameters and formulas

4.6 Chapter summary

In this chapter I introduced Temporal Exemplar Model (progress-vs.-time model). I also investigated the effective parameters on players' experience and introduced three different

balancing algorithms: Discrete balancing, State balancing and Continuous balancing, which presented three different approaches in frequency of balancing algorithms. Finally, I presented four different minigames that were compatible with the introduced time balancing algorithms and discussed their performance when different balancing algorithms are employed. In the next chapter, I will evaluate the performance of these balancing algorithms with the four different minigames introduced in this chapter.

CHAPTER FIVE
EVALUATION

In this chapter I provide performance information for Adaptive Time-Variant minigames (ATMs). In previous chapters, I introduced the concept of game balancing in multiplayer game, I identified the key elements in time balancing and reviewed design parameters to be able to have a minigame with manipulable completion time.

In these experiments I investigate the following questions:

1- Were the minigames were playable?
2- How fun was the players' experience?
3- Were the minigames able to deliver the desired time constraints using the three time balancing algorithms?
4- Were the minigames able to balance timing of tasks in a real game?

To answer above questions the evaluation was divided into three phases:

1- A laboratory study to examine whether it is possible to manipulate the completion time of minigames using a time balancing algorithms. The simplest of balancing algorithm, Discrete balancing, was evaluated for all four minigames in terms of performance and user experience.
2- An integrated study tested the performance of balanced games embedded within a larger game. I was interested to see whether the larger game is still fun to play when balanced using minigames.
3- The effect of intensity and frequency of the balancing algorithms crossed with different genres of game was evaluated by comparing all three balancing algorithms (Discrete, State and Continuous) together in terms of performance, noticeability of adjustments and perceived enjoyment of the minigames.

5.1 Testing Discrete Balancing Algorithm using Adaptive Time-Variant Minigames

Understanding whether time balancing algorithms are able to manipulate the completion time of the minigames is important because if time balancing algorithms cannot manipulate the completion time of the minigames, the impact of the type of adaptations is of limited intrest.

5.1.1 Goal

The main goals of this phase of the experiment are as follow:

1- Identify the performance of the Discrete balancing algorithm in controlling the total completion time of the minigames.
2- Determine the differences between different minigames in balance performance.
3- Determine the relationship between difficulty level and the performance of the balancing algorithm.

5.1.2. Method

I asked 15 participants (10 male and 5 female, aged 22 to 33) to play all the minigames. The experiment ran on a Dell 6500 laptop (Intel Core 2 Duo, 2.53 GHz) with a 15-in 1800x1200 display, and using a standard keyboard and mouse. Players trained on each minigame at each difficulty level once to reduce training effects, and then further played each minigame once for every difficulty level both with and without adaption. Difficulty levels for each game are shown in Table 2. The difficulty parameters are the static elements of the minigames that were consistent during the game play. By initializing the value of these parameters prior to start of the minigames, I was able to set the difficulty, and therefore, the minimum completion time of the minigames.

As shown in Table 2, the number of difficulty levels for Spinning puzzle and Click-And-Hack is not the same as for Electris, because I didn't want to bias my results by setting the difficulty too high. As discussed in chapter 4.4.3, the penalty for error associated with Electris is high and setting the difficulty at a high level could lead to negative play experiences.

Game	Difficulty Parameter	Value of Parameter
Spinning Puzzle	Number of rings	4,5,6,7,8
Electris	Number of rows	1,2,3
Click-And-Hack	Number of targets	10,20,30,40,50

Table 2: Difficulty level of the minigames. The table shows the difficulty of the minigames with respect to their configuration. For example Spinning Puzzle with 4 disks (rings) is the easiest and 8 disks is the hardest configurations of this game.

Participants were told that minigames would be used in a larger game in the future. I did not say anything about the adaptation algorithm. To hide the adaptation process from players, I told participants that the environments of the games are imaginary and the story of the main game will be in a mysterious, unknown world. I told them that these minigames would not necessarily follow the natural laws of physics. For example, physics and game mechanics could be changed during the game play.

5.1.3 Analysis and Result

This study was intended to verify that time of completion is controllable through the Discrete balancing algorithm using the static and dynamic elements of the minigames. I found that all the completion times were faster with adaptation, and linear with difficulty, albeit with a smaller slope. Results for the three tested games are shown in Figure 20.

These results demonstrate three important properties of my adaptive minigames:

- Minigames have linearly increasing mean time of completion with difficulty. The more difficult the game is, the more time it takes for the player to finish. This fact, at first, sounds obvious but it is important for design. Furthermore, the linear increase means that the manipulation of initial estimated or minimum completion time is straightforward and predictable.

  Although increasing the number of disks in a puzzle makes the game longer, the role of other parameters such as the initial offset degree for each disk should not be underestimated. These charts (Figure 20) indicate the change in completion time when changing one adaptive element of the minigame and keep the rest constant.

- There is a game-dependent decrease in completion times with adaption. In Figure 20, the lower line, marked with red circles shows the average completion time of the minigames for different difficulty levels. Although this decrease is not the same for different minigames, it establishes the functionality of the adaptation algorithms.

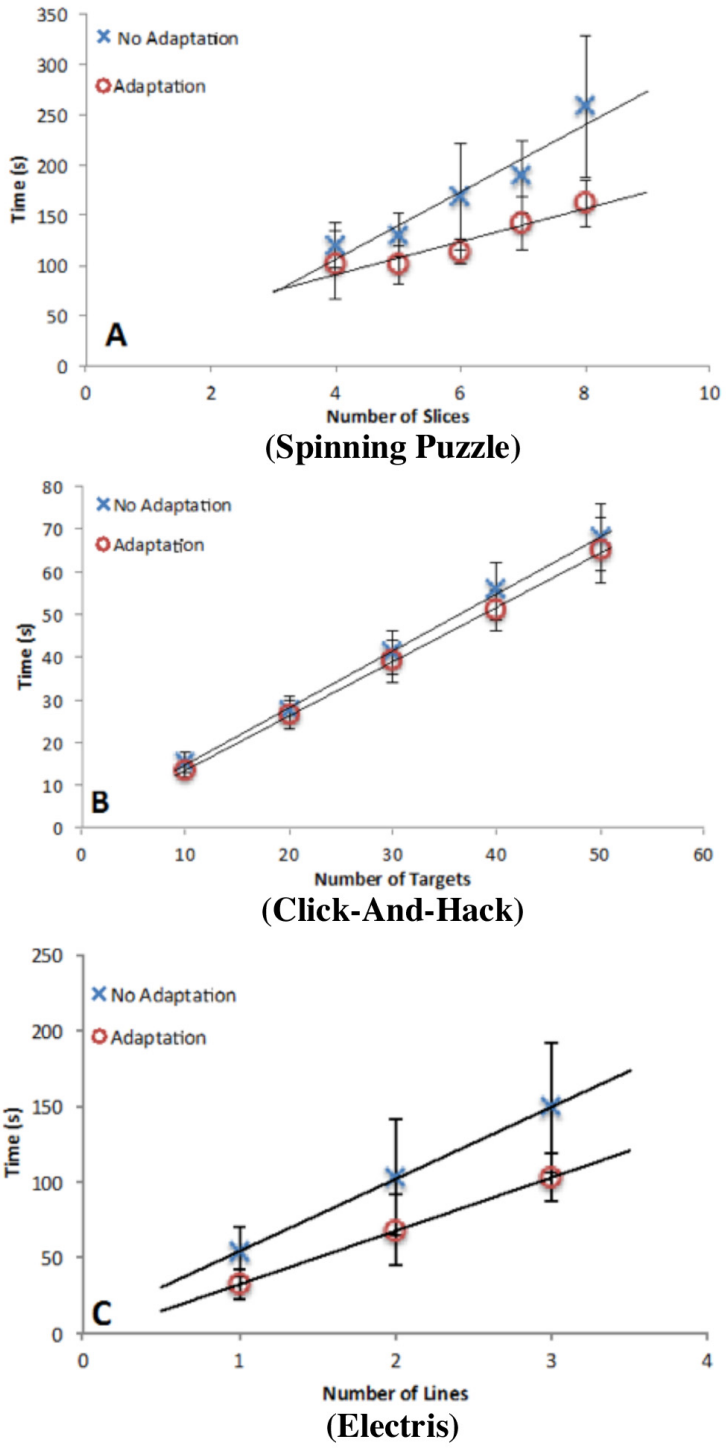

Figure 20: A: Average completion times for Spinning Puzzle for 5 different difficulty levels. B. Average completion times for Click-And-Hack for 5 different difficulties. C. Average completion times for Electris for 3 different difficulties.

- The means and variances of completion time of each of the games are different. The mean shows the expected completion time, since it is calculated over a range of players. Variance measures how far the completion times are spread from the mean value and indicates the degree of heterogeneity amongst players. The variance shows the differences in different players' skills and experience. It is indicates that the completion times of different players vary based on such parameter other than the level of the difficulty. For example, in Click-And-Hack, this range changes linearly with the difficulty level of the game, while in Spinning Puzzle it does not occur. In fact, Click-And-Hack is a repetitive task, which is hidden within the narrative of the game, every subtask is a smallest and simplest possible activity in the game and the penalty of player's mistakes is minimum. These reasons decrease the variation of completion time of the different players. On the other hand, Spinning Puzzle requires some thought and state of mind, which is highly dependent on the individual's skills and experience.

5.2 Testing Discrete Balancing using A Multi-Player Location-Based Mixed-Reality Game: Stealth Hacker

As a part of the second evaluation phase I ran an experiment on a MMR game. I implemented a mixed-reality location-based game called Stealth Hacker and exploited my minigames to balance the timing of its components. Evaluation of the time balancing algorithms using a larger game essential to establish whether the time balancing using adaptive time-variant minigames can impact overall game balance.

5.2.1 Goal

The main goal of this phase of the evaluation was to evaluate the performance of the adaptive algorithms in a larger game, in particular the user experience in terms of perceived enjoyment level of the game (and minigames), noticeability of the balancing algorithm, in both minigames and the main game, and the overall efficiency of the minigames at maintaining balance.

5.2.2 Method

5.2.2.1 A Multi-Player Location-Based Mixed-Reality Game: Stealth Hacker

Stealth Hacker is a mixed-reality location-based game inspired by the playground game *Cops and Robbers*, played with several cops and a single hacker. The shared playground is a network of computers, which the hacker attempts to infiltrate. The cops navigate this playground physically, moving from computer to computer and scanning them with smartphones (Figure 21.A). The hacker, fittingly, moves from computer to computer virtually, by navigating a simple avatar around a network diagram (Figure 21.B).

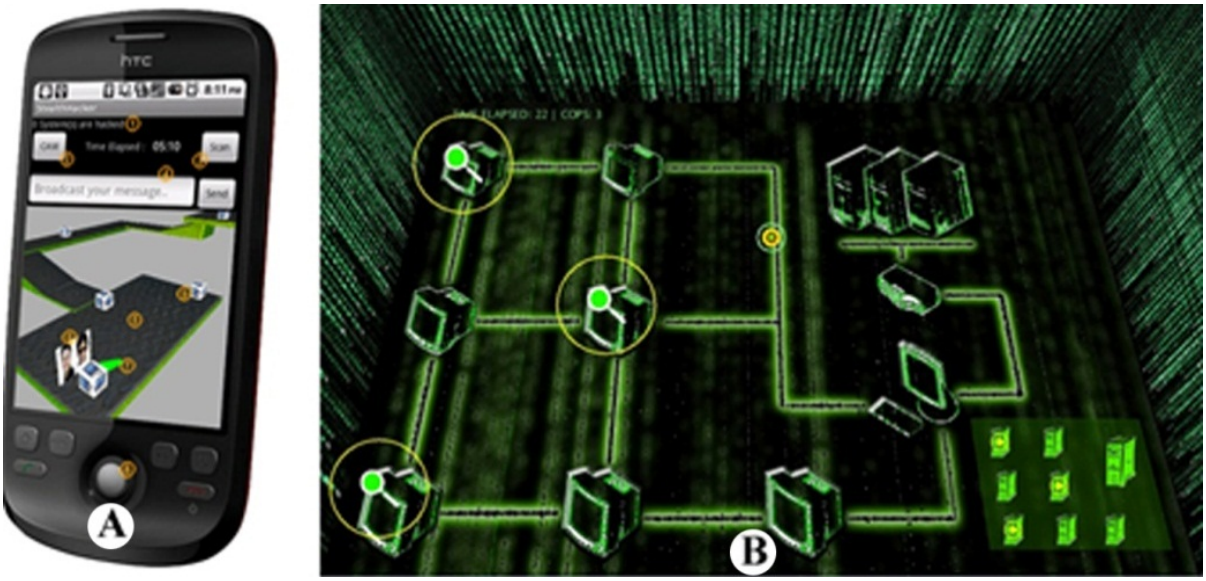

Figure 21: Stealth Hacker interface: The real-world players' interface on smartphone (A) and the virtual-world player's interface on a standalone PC (B)

The movement speed of the hacker, fitting the narrative, is on the order of seconds, providing the feeling of zipping across the network from computer to computer. Cops, in contrast, move from computer to computer on foot, with elapsed times on the order of tens of seconds, fitting the Newtonian physics that governs motion in the real world. This asymmetry of spatial representation and navigation speed creates an interesting timing dichotomy: in the real world, the cops predominantly spend time moving between nodes, but spend little time at each node,

while the hacker can transit between nodes quickly, and therefore must spend more game time at network nodes to maintain time balance.

The cops' interface provides them with information on the location of their partners (both current location and planned movements), the last known location of the hacker, and a chat interface. When the cops scan a computer, the program records the computer's Bluetooth MAC address, and transmits it to the server wirelessly. The scanned computer does not actually contribute anything other than its Bluetooth address, because the server manages the game state, and the hacking and scanning are simulated as minigames on the smartphones or the hacker's PC. The hacker tries to hack every computer in the network. Minigames are launched when the hacker attempts to infiltrate a computer, and these minigames provide dynamic balance through guaranteed minimum and expected mean and maximum times at each node.

In *Stealth Hacker*, one of three minigames (Hack-A-Computer, Electris, or Spinning Puzzle) is allocated to an individual node when the hacker arrives and attempts to break in to the computer at that node. The game choice and its initial complexity are based on the average real-world distance from the attacked computer to the two nearest cops, based on an estimated foot speed of 4.95 ft/s [73]. I chose two cops - first to motivate the cop players to together to arrest the Hacker, and second, to balance the powers of the Hacker and the Cops. The appropriate equation in Table 1 is used to calculate parameter settings for each game and that will provide a target completion time that matches the estimate of the cops' travel time (from the distance and speed heuristic). The "Discrete" adaptation to balance the game, which means that once the hacker has played the minigame for the minimum time, the game adapts to allow the hacker to finish whatever tasks remain as quickly as they are able, by increasing the speed of the adaptive component listed in Table 1.

The Brickout game is given a special role. It is triggered if the hacker is caught by one of the cops. Catching the hacker occurs if a cop arrives at the same physical location as the hacker's virtual location, and 'scans' the computer. For every cop that 'scans' the hacker during a single instance of the Brickout game, an additional row of bricks appears, making the game more difficult to complete. If the hacker completes the game before a timer runs out, they escape back into the network. If the hacker fails to complete the minigame, they are captured and the cops win.

The minimum game completion time for Brickout, as set by the ball speed, is slightly longer than the average physical transit time between any two physically adjacent nodes in the network. Well-organized teams of cops therefore have the chance to 'gang up' on the hacker by ensuring that reinforcements are sufficiently close. This special case demonstrates that minigames can be tuned to manipulate game balance based on user input as well as initial game state.

5.2.2.2 Game Balance in Stealth Hacker

As mentioned previously, game balance in a mixed-reality game is critical, and I used ATMs as the main balancing mechanic, both to balance the timing of the tasks in the game, and to improve the enjoinment of the game for the players. In Stealth Hacker, cops play the game in the real world while the Hacker plays in a virtual world. Obviously there is huge difference between these two types of worlds that should be considered in the game balance:

- Cops can freely move in the real world and change their location, while the Hacker is limited to the virtual world of the game.
- Cops can move with their desired speed (as fast as or as slow as they want) but the Hacker moves with a constant speed that the game designer has set prior to the game.
- The Hacker moves from one computer to another without any obstacles while it is possible for cops to be trapped by the potential obstacles of the real world such as dead ends, lack of signal coverage and locked doors.

  Stealth Hacker was implemented as a test bed to evaluate the performance of the ATMs. Although managing technical issues in MMR games is a challenging task, it is possible to address many of the timing issues by exploiting ATMs. The goal here is to make the gameplay enjoyable for players. It would be easy to force a virtual player, the Hacker, to wait for the real players by arbitrarily pausing at nodes; however, watching loading bars is not generally regarded as a recreational experience. In Stealth Hacker, it is possible to check the game state continuously while the Hacker is in the middle of the hacking process and change the minigame's game mechanic to reach balance.

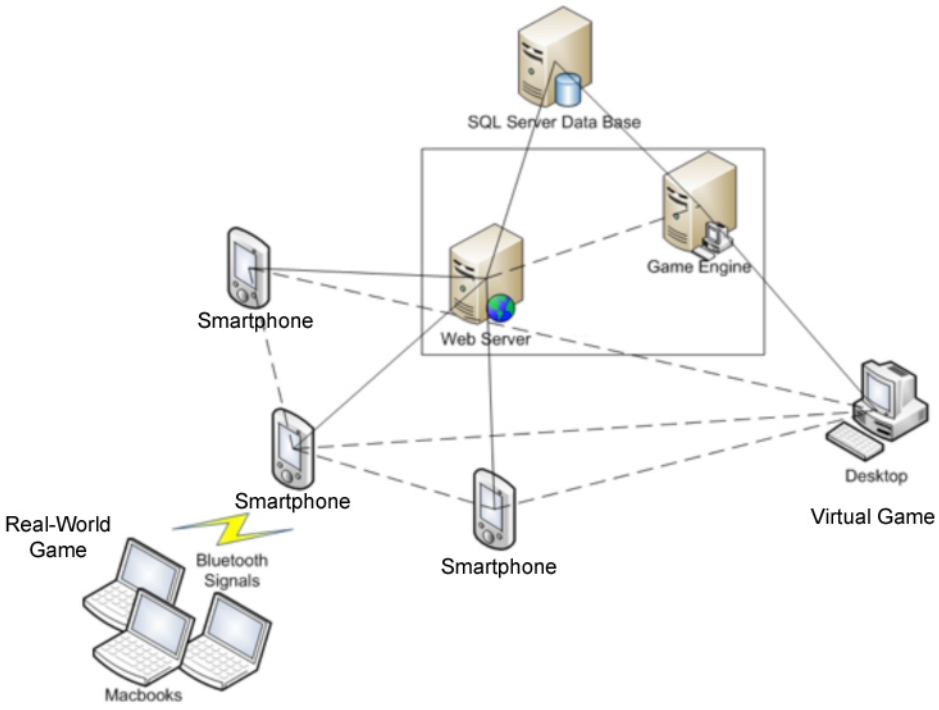

Figure 22: Infrastructure of Stealth Hacker

5.2.2.3 Implementation

Stealth Hacker is implemented with C# .NET using Visual Studio 2010 and Android using OpenGL ES and Eclipse Helios. Stealth Hacker contains more than 12000 lines of code for all game components. Figure 22 represents the infrastructure of the game. As shown in the figure, the game has two different sections; *real* and *virtual*, where cops and the hacker play. The mixed-reality engine of the game is responsible for executing the game play and synchronizing the *real* and *virtual* sides. When a cop gets close enough to one of the computers, the cop's device detects the presence of a new location via the Bluetooth signal of the computers. The cop's device sends a request to the server including the state of the game - location of players - and asks for the latest update on the hacker's position. The server receives the cop's request and reflects it to the current state of the game in the database server of the game (based on Microsoft SQL Server) and updates the state of the game. The server then sends the cop the updated game state. The hacker's system also frequently asks the server for the latest game state and represents it in the hacker's interface.

Since time management and synchronization in multiplayer games is one of the most important factors that affect the game play, it was crucial to handle the timing of tasks in both the real-world and the virtual-world. Moreover, location-based games are usually vulnerable to stochastic confounds such as interruption in network coverage.

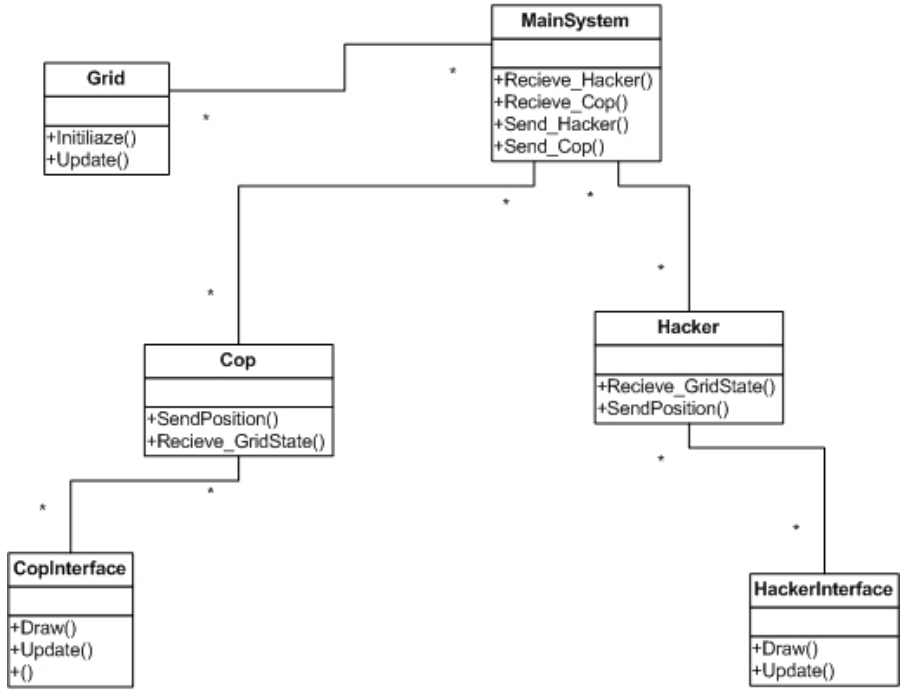

Figure 23: Class diagram of Stealth Hacker

The system minimizes data transfer by eliminating worthless data transfers from the player-server communication (Figure 23). For example, whenever a cop asks the server for the latest update, the server checks the latest update time of the game state package received from the cop with the most recent update of the game state in the server and answers to the cop's request only if these two states are different. The code below shows a sample of the XML message that is transferred between a cop and the server:

```
<Inspectors>
       <inspector id="0" position="0">
       </inspector>
       [...]
       <hacker position ="0"></hacker>
       <scan request="0"></scan>
       <victory flag="0"></victory>
```

```
</Inspectors>
```

5.2.2.4 Experiment

A group of players aged from 25 to 39 years played Stealth Hacker 8 times. Each time a different participant played the hacker, meaning each player played the hacker twice. Prior to the real experiment, participants played a practice round to make sure that the system worked smoothly and players understood the narrative and mechanics of the game. The experiment was run in the Thorvaldson building of the University of Saskatchewan. I arranged computers in two different floors of the building:

- 3 computers in a second floor laboratory
- 1 computer in the third floor corridor
- 3 computers in a third floor laboratory

5.2.3 Evaluation of the Stealth Hacker MMR Game

Three outcomes were observed: that players had positive experiences playing the game as the hacker, that the game remained balanced in opportunity if not outcome, and that the minigame timing reflected the game state at the time of instantiation.

To evaluate the subjective user experience during play, I administered a survey. Players felt that the minigames were fun (mean rating 3.6 out of 5), and added to the overall game (4 Yes, 0 No), which was also seen as fun (mean rating 4.25 out of 5). I also asked participants to rate the percentage of time they spent playing minigames (mean 62.5%), which was substantially less than the value measured from the logs (mean 79.8%), which could indicate that players were absorbed by the minigames.

I examined the balance of opportunity and outcome by determining the number of hacked systems per game and plotting an annotated node occupancy diagram for one of the shorter games played. The ‘number of hacked systems’ metric is a measure of overall balance because if the number is too small, it indicates that the hacker had little chance of winning; if too large, it indicates dominance by the hacker. The hacker hacked all seven systems 3 times, winning the game, but still managed to hack at least 3 and an average of 5.75 systems in the 5 losses. The dynamic timing balance achieved by the adaptive minigames is shown for a single game in Figure 24.

In this figure, the dark boxes represent the hacker playing a minigame and the numbered light boxes represent the three cops while each number refers to one of the cops. The length of each box represents the time that each player has spent in a location. The y-axis is the node location (one of the seven computers). Early in the game the cops were near the hacker, and the minigame engine spawned three relatively easy games. Once the hacker moved to a more distant node, a much more difficult game was spawned, which the hacker successfully completed. In the final game, a more difficult game was also spawned, as the cops were initially far away, but rapidly converged on the location of the hacker and trapped him with three consecutive rows of Brickout, shown by the occupancy of location 3 at the end of the game.

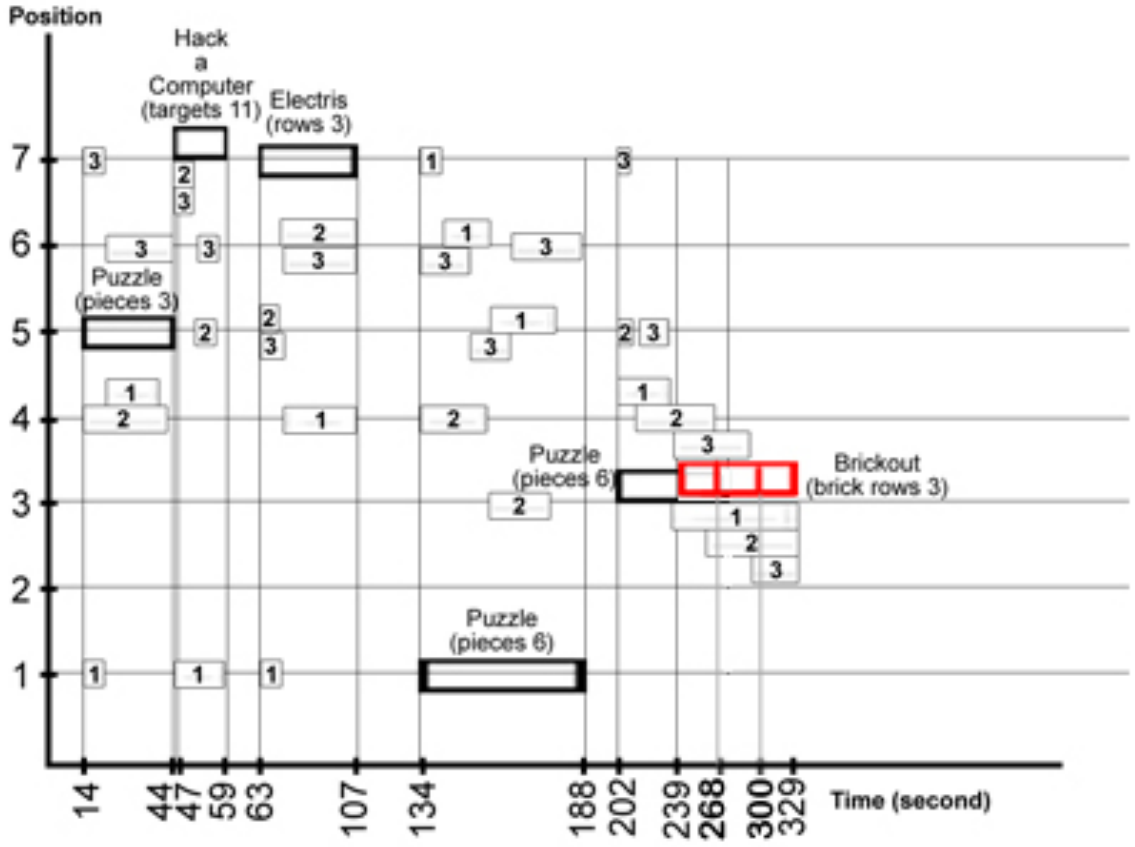

Figure 24: Player locations and actions in a single Stealth Hacker game

To verify that the minigame duration reflected the game state as the game evolved, I carried out the analysis showed in Figure 24, which shows that in a single instance at least, the cops were often proximate to the hacker while the hacker played the minigame. However, a single game does not provide compelling evidence of efficacy.

Figure 25 shows every minigame played over all 8 conditions (each column represents one round of the game while one of the players was the Hacker). In this chart, each point represents a specific time while a minigame is being played:

- The red points are the "Actual Time" which show the time that has been taken for the player to finish the minigame.
- The blue points are the "Estimated Time" which is calculated by the game engine and represents the minimum time that it takes for two cops to reach the Hacker.
- The green points are the "Minimum Time" which represents the minimum completion time for the current minigame.

As shown in the chart, the estimated average time for two cops to reach the hacker closely tracks the minimum calculated completion time of the minigame, demonstrating that my techniques have sufficiently high temporal resolution to capture variable game states. The actual time of completion follows the minimum values and shows variability both within and between subjects demonstrating the techniques provides game balance control without artificially limiting the game, by still allowing for player expertise and chance to play a role.

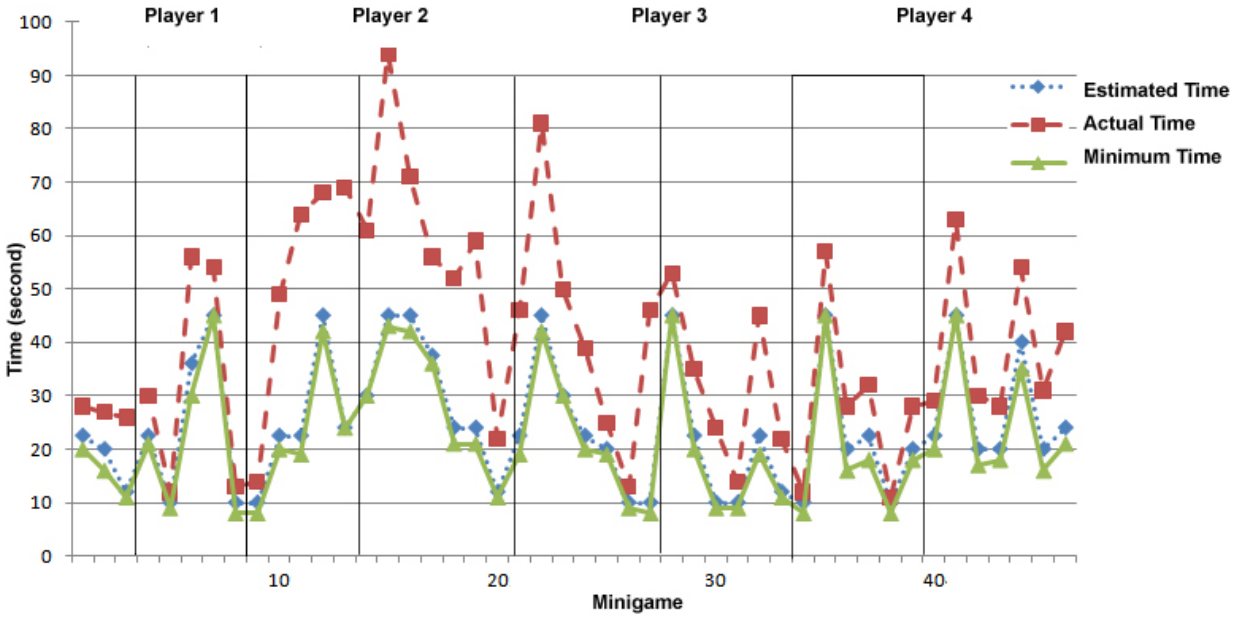

Figure 25: Minimum completion times for all minigames in the experiment

5.3 Testing the Effect of Temporal Adaption Granularity and Game Genre on Abilities of Time Balancing Algorithms

Only the Discrete balancing algorithm has been presented thus far, both individually in embedded within a larger world with a multi-player Mixed-Reality game. In the third study tested the performance of other the balancing algorithms – State and Continuous – and compare their results. To test State and Continuous balancing algorithms I need to have a time-vs.-progress model (see section 4.1). I decided to divide the third study into two sections:

1- In the first section I ran an experiment to record players' data while playing minigames separately. The result of this experiment was used to create as temporal exemplar models for each minigame.
2- In the second step, I evaluated the balancing algorithms for all minigames using a different pool of participants.

5.3.1 Exemplar models

The time-vs.-progress model represents the player's thinking and reacting behavior, and generally, the way that players are playing a game. Since the model is a step-by-step recorded of progress over time, it can be used to investigate the general balance of the game.

5.3.1.1 Goal

The main goal of this section of the study was to create the progress-vs.-time model from players' data for each minigame. To be able to perform the Continuous balancing and State balancing algorithms, it is important to have a model that describes players' experience as a form of progress over time. The goal of this study was to precisely record players' progress and to represent it as a function of time.

5.3.1.2 Method

Eight volunteer participants were recruited to play each of the game conditions. The average performance of the eight players within each game for the exemplars, where 'performance' was defined differently for the different games: time per disk in Spinning Puzzle, time per brick in

Breakout, time per targeting action for Click-And-Hack, and time per piece for Electris. The performance data was used to create time-vs.-progress models similar to those of Figure 11 in section 4.1.

5.3.2 Testing the Effect of Temporal Adaptation Granularity and Game Genre on Abilities of Time Balancing Algorithm

The goal of this to experiment is to investigate two main issues with time balancing algorithms:

1. Accuracy in managing completion time -which contains the following questions:
   Question 1: are the adaptive approaches more accurate than the non-adaptive condition?
   Question 2: which adaptive approach is most accurate?
   Question 3: does game type or difficulty level affect accuracy?
2. Player experience – which contains the following questions:
   Question 4: Were there differences in the players' perception of the different adaptive approaches?
   Question 5: Did differences in adaptation alter the players' enjoyment of the game?

5.3.2.1 Method

For this study, 24 test subjects were recruited (12 male and 12 female, average age of 27 years) from the university community. Participants were all experienced with mouse-and-windows software, and had a wide range of experience with video games (18 played games rarely – less than 3 hours a week, 5 played regularly – between 3 to 10 hours a week, and 1 played frequently – more than 10 hours a week). The study was carried out in a controlled environment using two systems, on a Windows 7 PC with a 1920x1080 screen and a dual core laptop with 1280x800 resolution. Minigames were run full-screen, and were all controlled with a standard two-button optical mouse. The study software recorded all performance measures and questionnaire data was gathered using online forms.

Participants played two versions of each of the four minigames described in section 4.4 (Click-And-Hack, BrickOut, Electris, and Spinning Puzzle). One version had 'easy' starting difficulty, and therefore a lower expected completion time, and one version had 'medium' difficulty and a

longer expected time. The specific starting values for easy and medium were dependent on the type of game, and are shown in Table 3.

	Easy	Medium
Click and Hack	30 targets	50 targets
Spinning Puzzle	5 rings	6 rings
Electris	1 row	3 rows
Breakout	1 row	3 rows

Table 3: Minigame configuration for different difficulty levels

Fatigue may lead to biased results so the difficulty of the minigames was set such that the experiment would be completed in a reasonable time. Participants played the eight different minigames (four game types and two difficulty levels) under the four different adaptation approaches described in section 4.3 (No balancing, Discrete balancing, State balancing and Continuous balancing).

Each player was briefed on the different games and the procedure of the study. Players were told that I was testing different game configurations, but not what the differences between the conditions were or the ordering of the conditions. Similar to the first study – Testing Discrete balancing, I told players that minigames were going to be used in a bigger game and that I was looking for the best parameter settings, so there could be some differences in the games. The players then played the eight minigames shown in Table 3 (four game types and two difficulty levels) with each of the four balancing algorithms. Players played all of the difficulties and balancing algorithms within each game in a different order, based on a Latin square design.

After every game, participants were given a short questionnaire to determine their play experience and their impression of the perceptibility of the algorithms. I asked the participants to complete the questionnaire right after the experiments to make sure that the participant's memory of the game was fresh. Game state and all parameters associated with the time balancing

algorithms were logged. Once the participants had completed all games, they were given a final questionnaire on their experience and a brief demographic survey.

5.3.2.2 Evaluation

The study used a factorial within-participants design, with three factors:

- *Adaptation Algorithm*: No-Balance, Discrete balancing, State balancing, Continuous balancing
- *Difficulty*: Easy or Medium starting difficulty
- *Game*: Click-And-Hack, Electris, Spinning Puzzle, BrickOut.

The order of presentation of the games, and the order of presentation for the difficulty and adaptation conditions within each game, were balanced using Latin square designs. The main dependent measures were game completion time and game performance (progress over time was also recorded for the adaptation mechanism and is also used in my analysis). Timing data gathered from computer logs were analyzed with three-way ANOVA tests; post-hoc tests were conducted using Tukey's HSD. Survey results were analyzed using Friedman's ANOVA for related samples. For all tests, α was set at 0.05.

5.3.2.2.1 Overall Completion Time

The ultimate goal of ATMs is to provide game designers with the ability to deploy situation-dependent time-balancing minigames within a larger game, and maintain tight control over the minigame completion time by using dynamic adaption to move individual performances toward an exemplar. Figure 26 shows the completion time distributions for all conditions in the second experiment.

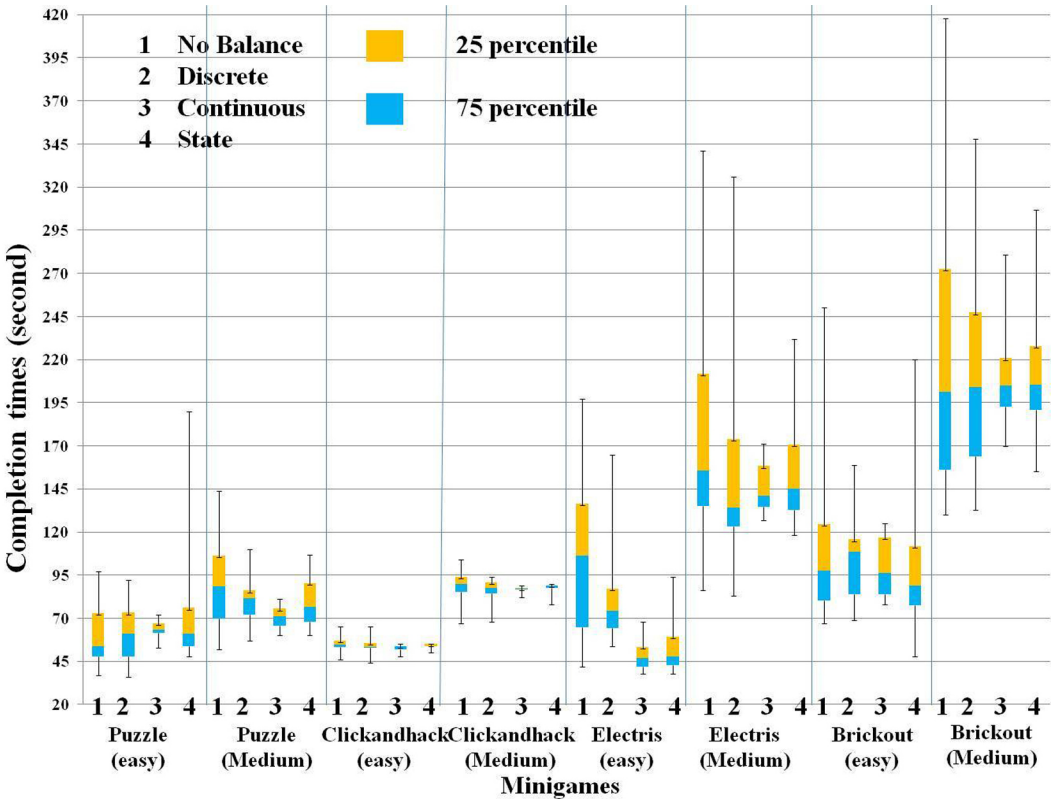

Figure 26: Completion time distributions for all minigames and all conditions with minimum, 25 percentile, median, 75 percentile and maximum values

Within each game category, the Continuous adaptation is usually at minimum in time and variance, and No-Balance adaptation case is usually at maximum. The exception is Spinning Puzzle in easy mode (5 disks), which was dominated by a few notable outliers in the State and Continuous cases, where completion time was dominated by the difficulty of the puzzle, not the speed of the disks. In all cases the quartiles (represented by the extent of the box) and the 95% confidence interval (represented by the whiskers) are smallest for the Continuous case, indicating that player performance more closely adhered to the exemplar.

5.3.2.2.2 Accuracy in Managing Completion Time

Accuracy was determined by subtracting the completion time for each different game from the desired time indicated by the exemplar model; this provides an error for each minigame. Given the four adaption scenarios examined – No-Balance, Discrete, State and Continuous – The adaptive cases – Discrete, State and Continuous – should converge toward the exemplar, and the

No-Balance case depart from the exemplar. Given the nature of the games, there should be differences in the relation between the balancing algorithms and the completion time for different games and difficulty levels.

The ANOVA showed significant main effects of all three primary factors on error amount (Algorithm: $F_{3,69}$=14.67; Game: $F_{3,69}$=48.27; Difficulty: $F_{1,23}$=16.13, all $p<0.001$). A summary of mean error amounts for these factors is shown in Figure 27.

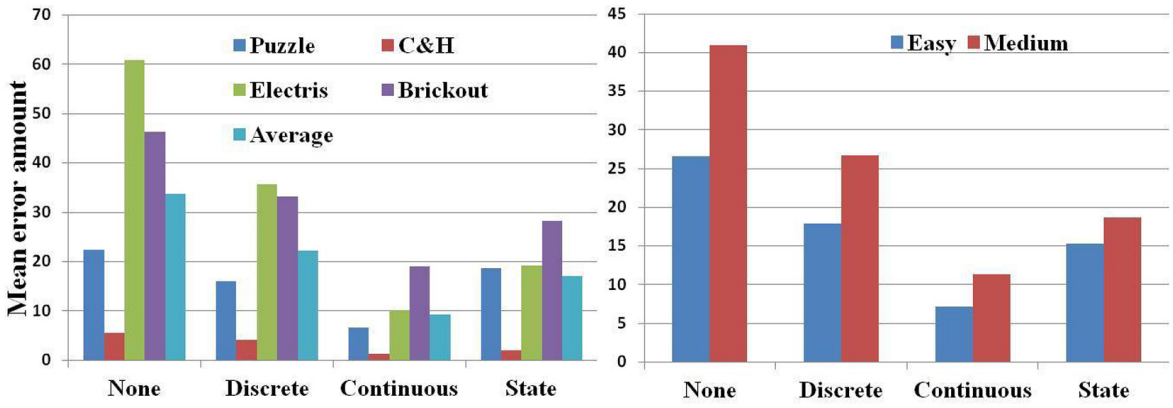

Figure 27: Left: mean error, by game and algorithm (note that the overall average for each algorithm is shown in the final bar of each group). Right: mean error, by difficulty and algorithm

My primary interest in following up these main effects was to find whether the adaptive approaches were more accurate than non-adaptive case (Q1) and to explore the most accurate adaptive balancing algorithm among all the adaptive algorithms (Q2). A Tukey's HSD test showed that there were significant differences between the balancing algorithms (all $p<0.05$): all of the adaptation conditions had significantly lower error amounts than the No-Balance condition, and the Continuous algorithm had significantly lower error than Discrete and State; no other differences were found.

The ANOVA test also showed significant interactions between Algorithm and Game ($F_{9,207}$=7.20, $p<0.05$), and between Algorithm and Difficulty ($F_{3,69}$=4.19, $p<0.05$). Figure 29 summarizes these differences; as the figure indicates, the different algorithms performed differently on different games and difficulty levels. In particular, all algorithms performed better on Click-And-Hack than on the other games; and for some games (Spinning Puzzle and BrickOut), differences between the algorithms were larger with the more difficult starting conditions, whereas for others (Electris), the differences were larger with the easy version of the

game. These findings confirmed that there are many hidden elements in games that should be considered during the balancing process as discussed in section 3.2. Some of these variables are adjustable by game designers, but there are several factors that are out of designers' hands. For example, the reaction speed of the players, or the time it takes for different players to solve a puzzle in a game are outside of a designer's control.

Based on these results (Figure 26 and 27), I conclude that adaptation algorithm does have an effect on the results of balancing (Q1). Moreover, it is obvious that the choice of adaptation algorithm does affect accuracy, with Continuous having significantly lower error amount than other approaches (answer to Question 2). However, these results depend to some degree on both the type of game and the difficulty level (answer to Question 3).

5.3.2.2.3 Player Performance under Adaptation

Figure 28 shows the error times (Actual completion times minus baseline exemplar time) for all of the adaptation algorithms for Spinning Puzzle (medium), and Brickout (medium). Figure 29 shows the performance and exemplar of a single player for the same pair of games.

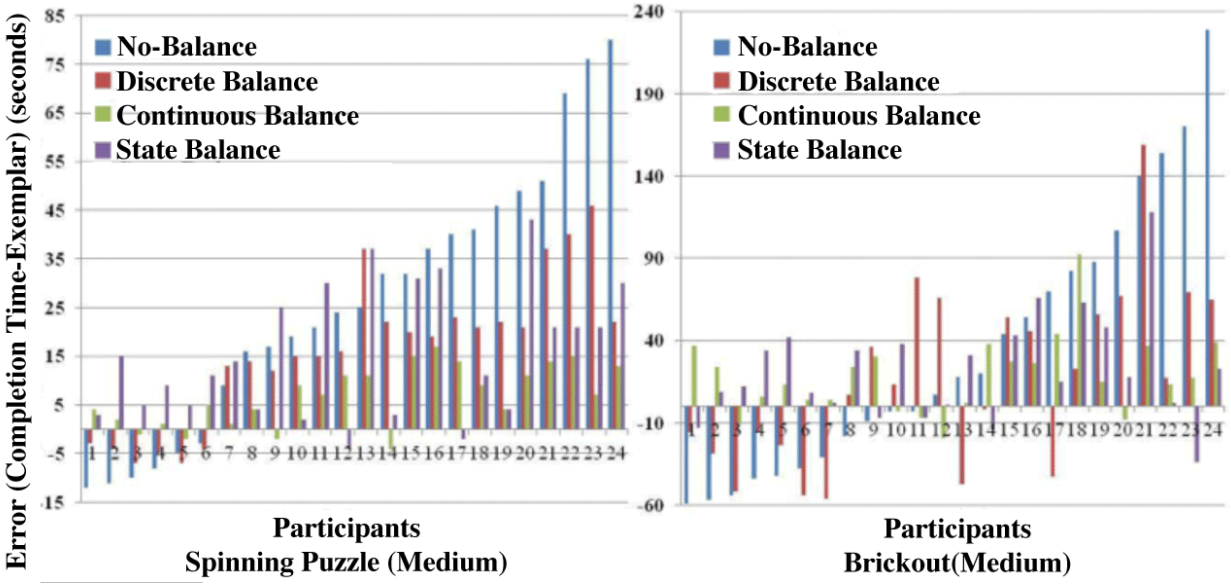

Figure 28: Error times for Spinning Puzzle - medium (left) and Brickout - medium (right) by person

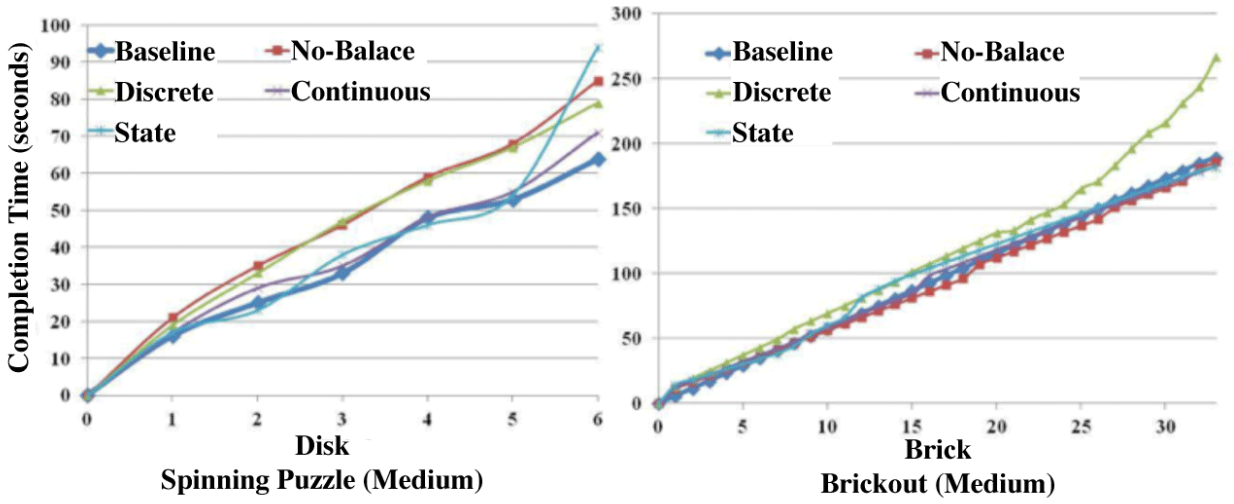

Figure 29: Performance and exemplar for a single example player for Spinning Puzzle - medium (left) and Brickout - medium (right). The bold line indicates the exemplar

Each graph in Figure 28 shows the absolute error performance of an individual participant for the given game and level combination. Players are sorted by completion time in the No-Balance case. Several notable outliers are evident in the State adaptation case for the Spinning Puzzle game. These outliers are primarily due to feedback effects and the low frequency of State updates in the Spinning Puzzle game, which only calculates balance once a disk has been correctly positioned. Players who performed particularly well on a particular piece are unduly punished with a speed reduction on the next piece, potentially dramatically increasing completion time.

This performance oscillation is evident in Figure 29 (left), which shows the game performance for a single player overlaid on the exemplar. In the State case, small oscillations in the exemplar and player performance feedback upon each other to drive increasingly larger swings in performance, culminating in a final completion time substantially slower than the exemplar or the Continuous case (Figure 30).

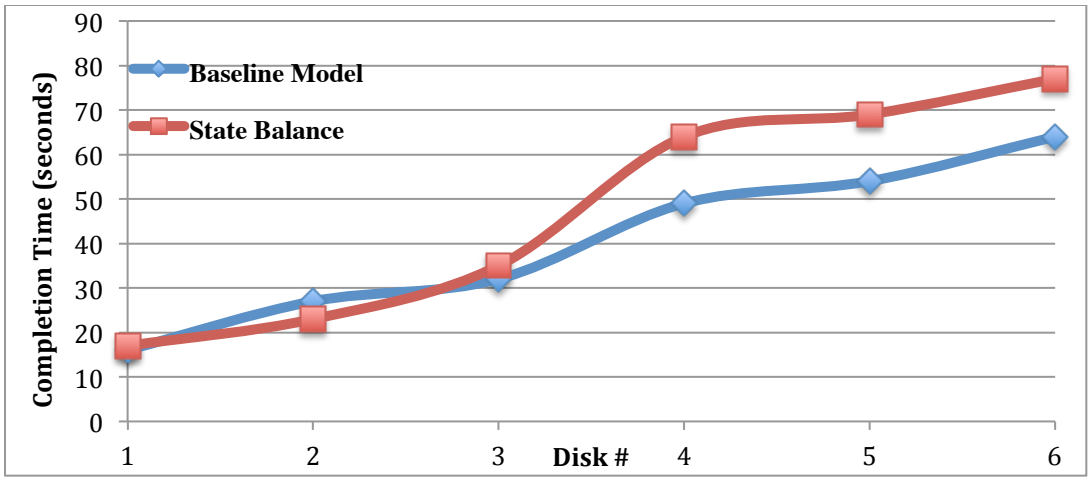

Figure 30: Spinning Puzzle (Medium)

In BrickOut (Figure 29- right) player performance follows the exemplar more closely, except for the Discrete-balancing case. The Discrete algorithm shows a marked departure from the exemplar near the end of the game. This performance lag was due to the player missing the last brick, and having to bounce the ball back and forth over the width of the screen and back again to achieve the correct angle to strike the final brick and end the game, demonstrating that while adaptation can drive the player performance distribution towards a desired shape in aggregate, individual player performance still matters for the outcome of the game.

5.3.2.2.4 Player Experience

The experiment established that minigames have useful properties for the parameterization of adaptation. However, appropriate balancing is of little utility if the adaptation algorithm destroys the game experience. To investigate the effects of different balancing techniques on player experience, I gave participants questionnaires after playing every condition and at the end of the session. Appendix A.1 indicates the questionnaire that was given to participants after each set of minigames and appendix A.3 and A.4 show the final questionnaire that was given to participants at the end of the session.

As a part of the results of the user study, I concluded the below statements:

- Over 59% of the players felt that minigames were fun or very fun (Figure 31) in every condition but State balance, also 12% of players felt that minigames were not fun in every condition but State balance. In State balance case, 46% of the players stated that the games were fun but 29% have felt that games were not fun.
- A Friedman test of the responses to this question (see question 3 from appendix A.1) indicated no significant differences in level of fun between the different games, indicating either that adaptation algorithms did not affect player enjoyment of the game, or that my instrument was insufficiently precise to find the differences (Answer to Question 5).

Figure 31: The fun level of games by adaptation algorithm

I also wanted to determine the relative perceptibility of the adaptation algorithm in each game. At the end of the experiment participants were asked "Did you notice a difference in the game mechanics between the four versions of <Minigame>?" and "Did you notice that the game mechanics in <Minigame> would change based on your performance in the game?" Since the first question refers to the different configurations of each game in the experiment, and each configuration represents a unique balancing algorithm, I was interested to discover whether players have noticed the differences between three balancing algorithms and No-Balance method. The answer of this question indicates whether players have noticed that the games were manipulated. The second question targets the relation between the adaptation algorithms and players' performance, which reveals the noticeability of the exploited game mechanics. The yes/no responses to these questions are plotted in Figure 32 A and B. I, intentionally, postponed these two important questions to the end of the session because if I would have asked these

questions after every set of minigames (every adaptation algorithm), players might have noticed that there should be an adaptation mechanism.

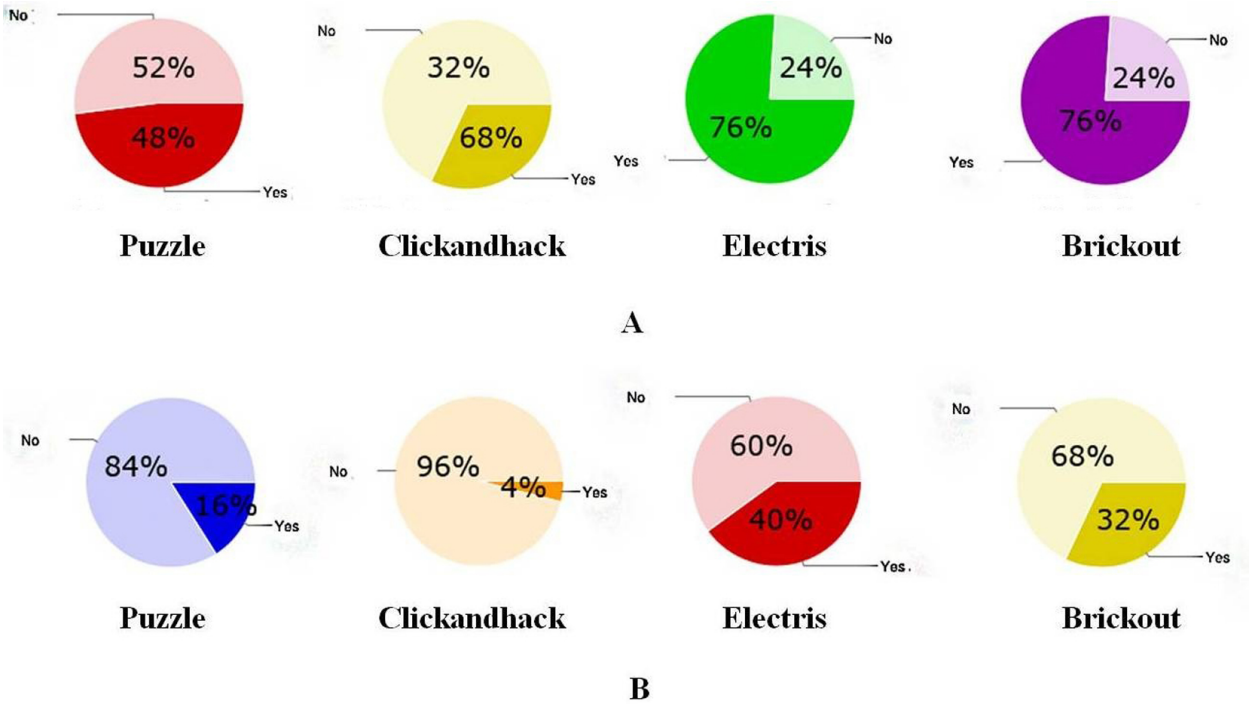

Figure 32: Perceptibility of adaptation algorithms (A). Perceptibility of game mechanics (B)

The majority of participants noticed a difference in game mechanics between the four cases, although this is possibly due to the appearance changes in the games when adaptation was employed.

Participants were also asked after each game condition to comment on whether they noticed any changes within the game. The question was kept intentionally vague to avoid biasing the within-subjects design. For all the games, only three participants (for Continuous and State) and five participants (for Discrete) responded affirmatively. Most of those responses commented on the change in game appearance. No respondents noted that the game mechanics changes seemed to be tied to their performance. In the Continuous case participants actively stated that changes in mechanic were unrelated to their performance. For example for the Continuous cases:

> "The background of Electris changes all the time during the game but it wouldn't affect my performance."

> "The colour changes are fine, but seem to coincide with speed reductions in the parts of the game I do not control."
>
> "The color change in the middle of the Electris game does not have any significant meaning, and was initially misleading."

These comments are distinct from feedback for the State cases where all players noted that the change in display was related to the speed of the game, indicating that the larger, less frequent speed changes in the State case were more noticeable.

> "In Electris: I think the idea that the background colors changed was not bad but the speed kept changing too... made it less predictable."
>
> "The blue is a nice touch, though it seems to indicate slower gameplay, so I found myself looking forward to the red."
>
> "I liked the change of ball color in the BrickOut game, perhaps it indicated the speed of the ball".

Based on the survey results and participants' comments, I conclude that while some players noticed the change in the dynamic adaptation, none perceived that it was tied to their performance, indicating that dynamic adaptation was not noticeable in the experiment (Answer to Question 4).

CHAPTER SIX
DISCUSSION

6.1 Outcomes of the Studies

My evaluations provide evidence for the efficacy of adaptive time-variant minigames as a mechanism for balancing time. In the first laboratory study I used the simplest form of balancing algorithm, the Discrete balancing algorithm, to manipulate the completion time of the minigames. This phase of the evaluation showed that it is possible to manipulate the completion time of minigames using time balancing algorithms. It also revealed that there are differences among different players with different experience and skill. This difference is important because it demonstrates that adaptive minigames alter but do not determine the game outcome.

In the second phase of the experiment, the real world study, I tested the performance of the Discrete balancing algorithm was implemented in a real mixed-reality multi-player game, Stealth Hacker. The result of the study showed that the minigames, using the Discrete balancing algorithm, are enjoyable and are capable of balancing the large game which they embedded within. In the third phase of the study I investigated the effect of the frequency of the balancing algorithms update by using all the balancing algorithms crossed with all the games. The results of this study showed that it is possible to improve the accuracy of the adaptation by changing certain parameters in adaptation algorithms and in particular, I demonstrated the connection between adaptation granularity and perceptibility, and showed that individual differences are still preserved.

6.2 What other types of ATMs are possible?

This work makes several contributions to the design and engineering of adaptive game mechanics. The idea of adaptive minigames can be applied much more widely than just the example systems demonstrated here. For example they can be used as time filters while levels are loading. The core elements of designing ATMs involve analyzing the time requirements for

each game mechanic in the minigame, determining how the game can be parameterized to control completion time, and designing an adaptive algorithm for responding to run-time events.

This process is applicable to a wide variety of game genres. For example, a search minigame (e.g., *Where's Waldo*) involves visual search as the main game mechanic. The time needed for visual search is a function of the number of items that must be searched, and the time needed to evaluate each item, allowing parameterization of items to the visual differences between the target and the distracters.

Many possible game mechanics can be considered: pattern matching - that includes recreating a previously shown pattern, aiming – that includes targeting and shooting, pursuit tracking – that includes purchasing a previously shown track step by step, short-term memory – that includes repeating a set of related or unrelated event, and spatial memory – that includes selecting the location of the previously shown items in the game. The timing profiles of some mechanics have been modeled (e.g., Fitts' Law [72], Hick's Law [74, 75], the Keystroke Level Model [76]), permitting the use of existing models as starting points for analyzing and parameterizing minigames.

The time needed for minigame tasks involving cognition (e.g., calculation, reasoning, or mental rotation) will be more difficult to predict and will be subject to greater individual differences, and so are less useful for use within an ATM. However, even cognitive tasks could be modeled using empirical testing – that is, a mean time and a distribution around that mean can easily be found by asking a sample group to play the game during design and testing, such as the exemplar presented here.

6.3 Explanation about Time-vs.-Progress Model

As I mentioned, The time-vs.-progress model represents the player's thinking and reacting behavior, and generally, the way that players are playing a game. Since the model is a step-by-step recorded of progress over time, it can be used to investigate the general balance of the game - for example if the total completion time of a game is strongly affected by only one element of the game for all players, it indicates that this element may not be functioning properly. This element could be a minigame, which is residing inside of a bigger game, or could be a specific task or activity inside of a game or minigame. Although there are many possible uses of

progress-vs.-time model, I kept the implementation simple, suitable for the preliminary analysis performed here.

6.4 Explanation for the main results

Accuracy of the adaptive methods. The results of the study showed that the adaptive algorithms performed well, and their success is a basic confirmation of my initial premise in this research – that the simpler mechanics of minigames can be analyzed and understood to the point where manipulation of completion time is possible. The overall completion times, mean errors and errors in different conditions and the in-depth examinations of player progress (e.g., Figure 26, 27, 28 and 29) showed that the algorithms were effective in recognizing divergence from the desired time, and effective in altering the games to shift the player's time toward the exemplar.

Differences between game types. The adaptation methods performed differently for the different games. In Click and Hack, there was very little difference between any techniques, including no adaptation at all. In this case I believe that game time is so well described by the underlying Fitts' Law model that setting the static initial parameters may be enough to provide a particular time value. In contrast, time error in Electris was much larger and more varied across the different algorithms. In this game, the gameplay follows a much less linear path than Click and Hack, primarily due to the effects of making errors. These results indicate that the complexity of the game mechanics play a large role in the behavior of dynamic balancing algorithms such that some events in one game lead to a huge delay in completion of the game whereas in other games has much less effect on the overall completion time.

The value of more-frequent adaptation. Making adaptation decisions more frequently (as in the Continuous and State algorithms) was less perceptible and caused fewer oscillations in the players' performance. The significant oscillations evident in some cases (e.g., for some players in Electris) suggests that effective time balancing may only be possible in simple games such as simple minigames. More complex games (i.e., most main games) have much more sophisticated mechanics, and are likely to exhibit non-linear behavior when adaptations are introduced. By constraining the adaptation to games with simple mechanics that response linearly to an input parameter - The risk of complex and difficult-to-control behaviors disrupting game balance is reduced.

Retaining individual differences. The studied also showed that employing an adaptive algorithm does not remove all variability from the games – as stated earlier, it is important to provide

competitive balance but without negating the effects of player skill or game design. In the study results, there were larger variances between games and between difficulty levels than there were between algorithms. This is desirable because it demonstrates that the adaptation is not the dominant factor in determining completion time, and that designers have freedom to create the timing profiles they desire by appropriately choosing the game and difficulty level prior to instantiation. It is also worth noting that the Continuous algorithm did not disrupt game timing when the players' performance was near the exemplar. Overall, the completion times still formed a distribution (albeit with significant variation in mean, and variance between games), with means driven towards the desired values specified by the exemplar.

Cost vs. performance. Although the Continuous balancing is nominated as the most accurate balancing method it is obviously a trade-off between the cost and performance of the balancing method. In specific situations, Continuous balancing is relatively expensive, since it evaluates the player's performance and game state moment by moment.

The performance of the Continuous adaptation. As mentioned earlier, one major step in time balancing using ATMs is to decide which adaptation algorithm should be carried out, for example Continuous or State; therefore, one important questions are the frequency at which the adaptation algorithm should be invoked during game play and the intensity that the adaptation algorithm performs on the adaptive components of the game. Since in each type of adaptation – Discrete, Continuous and State – the current state of the game is compared with a previously calculated exemplar, the frequency of the comparisons can affect the final result of the adaptation. With this in mind, I might conclude that the Continuous adaptation is the best at any situation.

The performance of an adaptation algorithm depends on other parameters such as game type, player type, frequency of adaptation and underlying game mechanics. Moreover, each algorithm should be hidden from the player while managing the game-completion time. In fact, an adaptation algorithm that performs accurately is not necessarily the most desirable if it interferes with player experience. For example, consider an adaptation method which compares the elapsed time with an exemplar and when reaches a certain time, finishes the game suddenly. This method is accurate because the total completion time of the game will be exactly as specified, but the heavy-handed manipulation could destroy the gameplay experience for the player.

Although one of the goals of this research is to investigate the role of temporal adaptation granularity and game genre in time balancing capabilities, the players' game experience is implicitly addressed. While it is a reasonable hypothesis that a higher frequency adaptation leads to improved accuracy, it is still necessary to evaluate the players' experience. In the third study of the evaluation phase of this research I asked 24 participants to play the 4 minigames with different configurations to determine the perceptibility of adaptation algorithms and game mechanics, and consequently to see how enjoyable these algorithms are for players.

The adaptive algorithms were evaluated with respect to accuracy of completion time, and player experience. Player experience was further divided into the enjoyability of the game (fun), and the perceptibility of the adaptation and adaptive mechanic. By considering the results in Figure 26 and Figure 27, the Continuous adaptation is the most accurate method among all other adaptation algorithms. Moreover, it deviated least from the desired completion time of all the algorithms. The more frequent operation of the mechanism dampens oscillation in players' performance (Figure 29) and consequently leads to smaller variations in game mechanics and is therefore less perceptible to the players (Figure 32).

Since Continuous adaptation compares the current progress of the game with a previously obtained model, there should be a model with a sufficiently high temporal granularity and measurement accuracy to serve as a baseline. Generating these exemplar models (progress-vs.-time models) usually requires time and energy. To create exemplar models of games, researchers need to first, find all the effective elements of the game that impact the total completion time of the game, and second, need to run several experiments to record progress of several players during their gameplay and reflect all of them into one model. This research used an exemplar that was derived from empirical data, which may be prohibitively expensive in commercial games.

6.5 Application and Deployment

Minigame-based time balancing can be employed in any game in which there are obvious breaks in pacing where a minigame can be inserted. This is often done in current mainstream games in a non-adaptive way with quick-time events (activities that should be performed in a given time or as quick as possible), where the primary gameplay mechanic is suspended and replaced with a rhythm/pattern-matching mechanic such as pressing a set of buttons on gamepad as quick as

possible or recreate a previously shown pattern as quick as possible. While a more fulsome examination of the applicability of this approach is the subject of future work, an initial discussion is provided here of the applicability of the minigame time balancing mechanic to two general types of game interactions: races, and action timing.

In race games, time is the final mediator. Whoever completes the challenge fastest – whether it is solving a puzzle, building a structure or navigating a maze – is the winner. Significant attention has been paid to providing balanced outcomes in racing games, from subtly increasing the top speed of the weaker player, to providing context sensitive power-ups based on position. Minigames could be used to help provide timing balance if the primary game mechanic provides for a break in the race. This could be a pit stop in a car-racing game, a locked door in a maze racing game, or the scheduled discipline switches in a triathlon. This type of timing intervention is analogous to the Stealth Hacker mixed reality game. In this case, designers should use games like Spinning Puzzle to maximize potential control over the timing, or Electris to provide an additional time penalty for player with too many mistakes in the minigame.

While racing games are based on time-to-completion, many other games, such as first person shooters (FPSs), real time strategy (RTS) and roleplaying games (RPGs) are based to a large extent on relative rates, such as damage-per-second (DPS), or power-to-build-time tradeoffs. Minigames could be spawned during changeover events, such as reloading a weapon or casting a spell to replace the fixed cool-down timers that are explicitly (for an MMORPG spell) or implicitly (through a reloading animation) rendered in existing games. This could be integrated into the game as an additional exercise in skill: players that can cast spells or reload their weapons faster would have a DPS advantage. Because of the tight timelines imposed by these small cool-down timers, designers would likely want to opt for low mean, low variance minigames such as Hack-A-Computer to add small amounts of balance to regularly repeated actions, rather than large mean and variance games suitable for infrequent actions. It is easy to imagine a direct variant of Hack-A-Computer in an MMO context where minigames - such as mystic ruins - would appear and players would have to click them in order to complete the spell.

6.6 Limitations and Future Work

The limitations of this work relate to the relative youth of multiplayer balancing algorithms in general, and time-balancing algorithms in particular. I highlight four primary shortcomings and the future work required to address them.

1. **Scope:** Only a fraction of the proposed balance methodologies described in this work have been tested, which are in turn only a subset of all the possible minigame balancing mechanics. While my research demonstrates the feasibility of the approach, fertile ground remains for examining the breadth of applicability and generalizability of the concept. In particular, many game mechanics are based on psychometric principles (e.g., movement, memory-based recall) that have well-studied models, and could be used to provide a better understanding of how particular kinds of game elements can predict completion time.
2. **Breadth:** The analysis focused on the adaptation mechanics, and on examining the impact of integrating the minigames within a larger gaming context. This simplification was undertaken to control overall game engagement. Given that I have already established the viability of the integrated approach in Stealth Hacker, this was a reasonable experimental methodology. Future work in this area involves consideration of how minigames can be designed to fit into the overall narrative of the main game, and how timing requirements can be identified within the main game and used as the initial conditions of the minigame.
3. **Sample Bias:** As with any experiment involving human subjects, there is the possibility for sample bias. I cannot conclude that my findings will hold for all players equally; in fact it is reasonable to hypothesize that competitive gamers would be more sophisticated at spotting small adjustments in game mechanics than the dedicated but not elite gamers studied here. Broader studies with different games and demographics could extend the results.
4. **Baseline:** I have only examined two simple variants of the exemplar model. In the first experiment and in the real world experiment I examined the case where the timing profile was entirely defined by the designer as the minimum required completion time. In the second experiment, the advanced versions of the adaptation algorithms, I examined the

entirely empirical case where desired average time was based on play-tester performance. In the future, I expect more sophisticated exemplar variants based on the synthesis of designer intuition and empirical metrics garnered during playtesting and by mining play logs after game deployment.

5. **More evaluation with real games in the real world:** The dynamic time adaptation algorithm presented in the real world experiment was somewhat crude, but accepted by players. The more elegant way would be testing the Stealth Hacker game in all situations and with all adaptation algorithms (Continuous and State). It is possible to record partial completion times of the players and use it as an exemplar in adaptation algorithms in a bigger game. Although I have carried an experiment to evaluate this effect with a simple version of the adaptation, the Discrete algorithm, it is still not clear whether the performance of the players will be affected within a bigger game with other adaptation algorithms.

CHAPTER SEVEN
SUMMARY

In this work I described a novel approach for balancing timing in multiplayer games using adaptive time-variant minigames. There are three primary contributions to this research, already published at ICEC 2011 [54] (nominated for the best paper) and the Entertainment Computing Journal 2013 [55]. I categorize the key contributions of my research in the following categories:

- *The concept of time balancing through ATMs.* By instantiating ATMs outside the flow of the regular game it is possible to adapt the timing with strictly controlled mechanics without interrupting the depth of play or narrative of the main game.

  The first phase of the study (testing minigames in situ when the Discrete balancing algorithm was employed) showed that the minimum and expected completion times of the minigames were predictable. The results of this experiment (Figure 20) revealed three important properties:

  1. Minigames have linearly increasing mean time of completion with difficulty.
  2. There is a game-dependent decrease in completion times with adaption.
  3. The means and variances of completion time of each of the games are significantly different indicating the presence of different players' skill and experience.
- *Evidence for the efficiency of the ATMs.* I demonstrated that ATMs can provide a compelling experience for balancing a mixed-reality game, a particularly difficult time-balancing problem since computer players must be balanced against those in the real world. The second study with a real mixed-reality game, Stealth Hacker, showed that the minigames were enjoyable, and provided the balancing effects for which they were designed. This phase of the experiment also revealed the followings:

  1. Players had positive experiences playing the game as the hacker
  2. The game remained balanced in opportunity if not outcome
  3. The minigame timing reflected the game state at the time of instantiation
- *Evidence for the interaction between adaptive algorithm, game mechanic, and game difficulty*. I found significant effects and interactions for all three factors, confirming my

intuition that these processes are important and linked. I further found that finer temporal granularity leads to less-perceptible adaptation and smaller deviations in game completion times. I found that a continuous time-based update strategy, coupled with design techniques meant to integrate or mask the adaptability led to average completion time tending toward the desired value, while minimizing player disruption. In particular I found the following result based on the third phase of the experiment:

1- All of the adaptive algorithms were more effective than the non-adaptive condition in manipulating minigame completion time.
2- The Continuous algorithm was significantly more accurate than all other algorithms, and State-based balancing was more accurate than Discrete.
3- The Continuous algorithm had the lowest standard deviation of all algorithms.
4- Participants noticed some changes to game parameters, but people did not notice the connection between the changes and their performance.
5- The more frequent adaptation algorithms (Continuous and State) appeared to be less noticeable overall.
6- The adaptive methods did not reduce participants' subjective level of fun.

7.1 Conclusion

In this thesis I discussed how these techniques could be integrated into a number of gaming genres, including the popular Racing, RPG, RTS and MMORPG genres. I demonstrated that these games can provide a compelling experience for balancing a mixed-reality game, a particularly difficult time-balancing problem since computer players must be balanced against those in the real world.

In the future, I hope to explore the potential for balancing other game genres using this mechanism. I will investigate additional minigame mechanics, more sophisticated adaptation algorithms and the integration within larger gaming contexts. This work represents a strong foundation for the continued research, development and deployment of time-adaptive minigames.

CHAPTER EIGHT
REFERENCES

[1] Biography of Thomas T. Goldsmoth. Available: http://en.wikipedia.org/wiki/Thomas_T._Goldsmith,_Jr. (Accessed 29 April 2013)

[2] The alternative video game blog. Available: www.digitalbattle.com (Accessed 7 April 2013)

[3] Leigh, R., Schonfeld, J., Louis S. J., Using Coevolution to Understand and Validate Game Balance in Continuous Games, GECCO 2008, Atlanta, Georgia, USA, pp. 12–16.

[4] Ludwig, J., Farley, A., A learning infrastructure for Improving Agent Performance and Game Balance, in Proceedings of the AIIDE 2007, Workshop on Optimizing Player Satisfaction, Technical Report WS-07- 01, G. N. Yannakakis and J. Hallam, Eds. AAAI Press, 2007, pp. 7-12.

[5] The official NeverWinter Nights 2 website. Available: http://nwn2.com/US/index.php (Accessed 12 April 2013)

[6] The official Age of Empires 3 website. Available: http://www.ageofempires3.com/ (Accessed 4 May 2013)

[7] The official Mortal Kombat website. Available: http://www.themortalkombat.com/age-gate?redirect=/ (Accessed 18 April 2013)

[8] The official MarioKart 7 website. Available: http://mariokart7.nintendo.com/ (Accessed 11 May 2013)

[9] Official website for Diablo 3. Available: http://us.battle.net/d3/en/ (Accessed on January 2014)

[10] Benford, S., Crabtree A., Flintham M., Drozd A., Anastasi R., Paxton M.L., Can You See Me Now?, ACM TOCHI, Vol. 13, No. 1, 2006, pp. 100-133.

[11] A portal to play Blizzard StartCraft 2. Available: http://us.battle.net/sc2/en/ (Accessed 23 March 2013).

[12] A portal to play Blizzard World of Warcraft. Available: http://us.battle.net/wow/en/ (Accessed 21 March 2013)

[13] Schneider J., Kortuem G., How to Host a Pervasive Game Supporting Face-to-Face Interactions in Live-Action Roleplaying, In Position paper at the Designing Ubiquitous Computing Games Workshop at Ubicomp, 2001.

[14] Sweetser, P., Wyeth, P., GameFlow: A Model for Evaluating Player Enjoyment in Games, ACM Computers in Entertainment, Vol. 3, No. 3, July 2005, Article 3A.

[15] Killi, K., Digital Game-based Learning: Towards an Experiential Gaming Model, The Internet and Higher Education, Vol. 8, Issue 1, 2005, pp. 13-24.

[16] Yannakakis N. G., Hallam, J., A Scheme for Creating Digital Entertainment with Substance, In Proceedings of the workshop on reasoning, representation and learning in computer games, 19th International Joint Conference on Artificial Intelligence (IJCAI), Ithaca, NY, 31 July, 2005, pp. 119-124.

[17] Costikyan, G., I Have No Words and I Must Design: Toward a Critical Vocabulary for Games, In Proceeding of Computer Games and Digital Cultures Conference, Tampere University Press, Finland, 2002, pp. 9-33.

[18] Fullerton, T., Swain, C., Huffman S., Game Design Workshop: Designing, Prototyping, Playtesting Game, Taylor & Francis US, 2004, pp. 43-80.

[19] Zohoorian, A., Stanley, K. G., Gutwin, C., Tavassolian, A., PLATO: a coordination framework for designers of multi-player real-time digital games, In Proceedings of International Conference on the Foundations of Digital Games, 2012, Raleigh, North Carolina, pp. 141-148.

[20] Rollings, A., Adams, E., on Game Design, New Riders Publication, Ed. 1, 2003, p. 271.

[21] Debold, E., Flow with Soul: An interview with Dr. Mihaly Csikszentmihalyi, What Is Enlightenment Magazine, Spring-Summer 2002, p. 1.

[22] Sweetser, P., Wyeth. P., GameFlow: A Model for Evaluating Player Enjoyment in Games, Computers in Entertainment (CIE), 2005, Vol. 3, No. 3, Article 3A, p. 3.

[23] Koster, R., Theory of Fun for Game Design, Paraglyph Press, Ed. 1, 2004.

[24] Csikszentmihalyi, M., Beyond Boredom and Anxiety: Experiencing Flow in Work and Play, San Francisco: Jossey-Bass Publication, 2000.

[25] Ibáñez-Martínez J., Delgado-Mata C., From competitive to social two-player video games, In Proceeding of 2nd WOCCI, 2009, Cambridge, MA, USA, p. 18.

[26] Hunicke, R., Chapman, V., AI for Dynamic Difficulty Adjustment in Games, Challenges in game artificial intelligence, AAAI Workshop, San Jose, California, US, 2004.

[27] Charles, D., Black, M., Dynamic Player Modeling: A Framework for Player-Centric Digital Games, In Proceeding of the International Conference on Computer Games: Artificial Intelligence, Games and Education, Memphis, Tennessee, USA, pp. 29-35, 2004.

[28] Malone, T., What makes computer games fun? In Proceeding of the Joint Conference on Easier and More Productive Use of Computer Systems. (Part 2): Human Interface and the User Interface, New York, NY, USA, 1981, p. 277.

[29] Crispini, N., Considering the growing popularity of online games: What contribute to making an online game attractive, addictive and compelling, Dissertation, SAE Institute, London, 2003.

[30] Andrade, G., Ramalho, G., Gomes, A., Corruble, V., Dynamic game balancing: an evaluation of user satisfaction, In Proceeding of American Association for Artificial Intelligence, Boston Massachusetts, UBM Tech Publication, 2006, pp. 3-8.

[31] Humble, R., Inside EverQuest, Game Developer Magazine, May 2004, pp. 18–26.

[32] Crawford, C., The Art of Computer Game Design, Washington State University Publication, 1984, Chapter 6, p. 6.

[33] Girardin, F., Blackstock, M., Dillenbourg, P., Finke, M., Jeffrey, Ph., Nova, N., Issues from Deploying and Maintaining a Pervasive Game on Multiple Sites, 2008.

[34] Boll, S., Krosche, J., Wegener, C., Paper Chase Revisited – A Real World Game Meets Hypermedia, In Proceeding of 14th Conference on Hypertext and Hypermedia, Nottingham, UK, 2003, pp. 126-127.

[35] Bateman, S., Mandryk, R.L., Stach, T., Gutwin, C., Target Assistance for Subtly Balancing Competitive Play, In Proceeding of CHI 2011, Vancouver, BC, Canada, pp. 7–21.

[36] Andrade, G., Ramalho, G., Santana, H., Corruble, V., Automatic computer game balancing: a reinforcement learning approach, In Proceeding of AAMAS 2005, pp. 25-29.

[37] Tan, C. H., Tan K. C., Tay, A., Dynamic Game Difficulty Scaling Using Adaptive Behavior-Based AI, IEEE Transactions on Computational Intelligence and AI in Games, Vol. 3, No. 4, 2011, pp. 289-301.

[38] Official website of Assassin's Creed 4: Black Flag. Available: http://assassinscreed.ubi.com/en-ca/home/index.aspx (Accessed January 2014)

[39] Hunicke, R., LeBlanc, M., Zubek, R., MDA: A Formal Approach to Game Design and Game Research, In Proceeding of AAAI Workshop on Challenges in Games, San Jose, California, US, AAAI Press, 2004, pp.4-4.

[40] K. Forbus and J. Laird, AI and the entertainment industry, IEEE 2002, Vol. 17, No. 4, pp. 15–16.

[41] Weibel, D., Wissmath, B., Habegger, S., Steiner, Y., Groner, R., Playing online games against computer- vs. human-controlled opponents: Effects on presence, flow, and enjoyment., Computers in Human Behavoir, Vol. 24, Issue 5, 2008, pp. 2274–2291.

[42] Olesen, J. K., Yannakakis G. N., Hallam, J., Real-time challenge balance in an RTS game using rtNEAT, In Proceeding of Computational Intelligence and Games (CIG), IEEE Symposium On, 2008, pp. 87-94.

[43] Demasi, P., Cruz, A., Online Coevolution for Action Games, International Journal of Intelligent Games and Simulation, Vol. 2, No. 2, pp. 113–120.

[44] Official website for Half Life 2. Available: http://orange.half-life2.com/hl2.html (Accessed 7 November 2013)

[45] Iida, H., Takeshita N., Yoshimura, J., A metric for entertainment of board games: its implication for evolution of chess variants, The International Federation for Information Processing, Vol. 112, 2003, pp. 65-72.

[46] Sinclair, J., Hingston, Ph., Masek M., Considerations for the Design of Exergames, GRAPHITE 2007, In Proceeding of the 5th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia, pp. 289 – 295.

[47] Game review section of Game Revolution website. Available: http://www.gamerevolution.com/review/unreal-tournament (Accessed April 2013)

[48] Yannakakis, G. N., Hallam, J., Real-time Game Adaptation for Optimizing Player Satisfaction, IEEE Transactions on Computational Intelligence and AI in Games, 2009, Vol. 1, No. 2, pp. 121-133.

[49] Barkhuus, L., Chalmers, M., Tennent, P., Hall, M., Bell, M., Sherwood, S., Brown, B., Picking Pockets on the Lawn: The Development of Tactics and Strategies in a Mobile Game, UbiComp 2005, Tokyo, Japan, pp. 358-374.

[50] Matyas, S., Playful Geospatial Data Acquisition by Location-based Gaming Communities, The International Journal of Virtual Reality, 2007, Vol. 6, No. 3, pp. 1-10.

[51] Lindt, I., Broll, W., NetAttack – First Steps Towards Pervasive Gaming, ERCIM NEWS Special Issue on Games Technology 57, 2004, pp. 49–50.

[52] Tuulos, V., Scheible, J., Nyholm, H., Combining web, mobile phones and public displays in large-scale: Manhattan story mashup, In Proceeding of Pervasive 2007, Toronto, Canada, Vol. 4480, pp. 37–54.

[53] Benford, S., Flintham, M., Drozd, A., Anastasi, R., Rowland, D., Tandavanitj, N., Adams, M., Row-Farr, J., Oldroyd, A., Sutton, J., Uncle Roy All Around You:

Implicating the City in a Location-Based Performance, In Proceeding of Advances in Computer Entertainment (ACE) 2004, Singapore, Vol. 21, p. 47.

[54] Tavassolian A., Stanley, K. G., Gutwin, C., Zohoorian, A., Time balancing with adaptive time-variant minigames, International Conference on Entertainment Computing (ICEC), 2011, Vancouver, BC, Canada, pp. 173-185.

[55] Tavassolian, A., Stanley, K, G., Gutwin, C., The Effect of Temporal Adaptation Granularity and Game Genre on the Time Balancing Abilities of Adaptive Time-Varying Minigames, Journal of Entertainment Computing, Elsevier, 25 July 2013.

[56] The official website of The Legend of Zelda: Ocarina of Time 3d. Available: http://www.zelda.com/ocarina3d/ (Accessed April 2013)

[57] Official Quake III Arena website. Available: http://www.idsoftware.com/gate.php?referer=%2Fgames%2Fquake%2Fquake3-arena (Accessed March 2013)

[58] Official Unreal Tournament website. Available: http://www.unrealtournament.com/ (Accessed December 2013)

[59] Juul, J., Intorduction to Game Time, The International Journal of Computer, Vol. 1, Issue 1, 2001.

[60] Official SimCity website. Available: http://www.simcity.com/ (Accessed March 2013)

[61] The official of The Sims website. Available: http://www.thesims.com/ (Accessed March 2013)

[62] Official website of PopAndDodge game. Available:

http://digidointeractive.com/popanddodge (Accessed 12 March 2013).

[63] Sipser, M., Introduction to the Theory of Computation, Course Technology Inc. press, Ed. 3, 2006, p. 273.

[64] Lopez, M., Gameplay Design Fundamentals: Gameplay Progression. Available: http://www.gamasutra.com/view/feature/1771/gameplay_design_fundamentals_.php (Accessed March 2013)

[65] Smith, G., Othenin-Girard, A., Whitehead, J., Wardrip-Fruin, N., PCG-based Game Design: Creating Endless Web, FDG 2012, Raleigh, North Carolina, pp. 188-195.

[66] Flatla, D., Gutwin, C., Nacke, L., Bateman, S., Mandryk, R., Calibration Games: Making Calibration Tasks Enjoyable by Adding Motivating Game Elements, UIST 2011, Santa Barbara, CA, USA, pp. 403-412.

[67] Official Neverhood game website. Available: http://www.neverhood.se/ (Accessed March 2013)

[68] Wikipedia entry for Sonic the Hedgehog 2. Available: http://en.wikipedia.org/wiki/Sonic_the_Hedgehog_2_(8-bit_video_game) (Accessed on March 2013)

[69] Official Capcom page for Lost Planet. Available: http://lostplanetcommunity.com/ (Accessed April 2013)

[70] Official Mario website. Available: http://mario.nintendo.com/ (Accessed April 2013)

[71] The official Need For Speed website. Available http://www.needforspeed.com/the-run (Accessed 4 Nov 2013).

[72] MacKenzie, I. S., Fitts' law as a research and design tool in human-computer interaction, Journal of Human-Computer Interaction, Vol. 7, 1992, pp. 91-139.

[73] Aspelin, K., Establishing Pedestrian Walking Speeds, Portland State University, 2005.

[74] Hick, W., On the rate of gain of information. Quarterly Journal of Experimental Psychology, Vol. 4, Issue 1, 1952, pp. 11-36.

[75] Hyman, R., Stimulus information as a determinant of reaction time. Journal of Experimental Psychology, 1953, pp. 188-196.

[76] Card, S. K., Moran, T. P., Newell, A. The Keystroke Model for User Performance Time with Interactive Systems, Communications of the ACM, 1980, Vol. 23, No. 7, pp. 396-410.

APPENDIX A

A.1 Questionnaire after each set of minigames

How successful were you in these games? *
(Completing the games as fast as possible, with the fewest possible errors)

	1	2	3	4	5	
Unsuccessful	○	○	○	○	○	Successful

Why you were successful or unsuccessful?
(Optional. One or two sentences only.)

How much fun were these games to play? *

	1	2	3	4	5	
Not Fun	○	○	○	○	○	Very Fun

Why were the games fun, or not fun?
(Optional. One or two sentences only.)

Which parts of the game visuals and mechanics did you like, and why? Which parts didn't you like and why? (The 4 games were: Puzzle, Click-and-Hack, Electris, and BrickOut) *
(Please be as specific and detailed as possible)

A.2 Demographic Questionnaire

Section 1 of 3: Demographics

Participant ID: *

Gender: *

- Male
- Female

What is your occupation? (If you are a student, what is your major?) *

What input devices do you usually use? *
(E.g. Keyboard, mouse, trackpad, joystick, touchscreen, etc.)

How many hours a week, on average, do you spend playing computer/video games? *

Which games, or types of games do you usually play? *
(E.g. First-person shooter, role-playing, real-time strategy, etc.)

When playing computer/video games, how much time do you normally spend per session? *
(Playing the game without taking a break)

- Less than 1 hour
- 1 to 2 hours
- More than 2 hours

A.3 The final questionnaire at the end of the third phase of the experiment (page 1)

1: Did you notice a difference in the game mechanics between the 4 versions of Puzzle? *

- Yes
- No

If yes, could you please describe the differences you noticed: (between the 4 versions of Puzzle)
Please be as specific and detailed as possible.

2: Did you notice a difference in the game mechanics between the 4 versions of Click-and-Hack? *

- Yes
- No

If yes, could you please describe the differences you noticed: (between the 4 versions of Click-and-Hack)
Please be as specific and detailed as possible.

3: Did you notice a difference in the game mechanics between the 4 versions of Electris? *

- Yes
- No

If yes, could you please describe the differences you noticed: (between the 4 versions of Electris)
Please be as specific and detailed as possible.

4: Did you notice a difference in the game mechanics between the 4 versions of BrickOut? *

- Yes
- No

If yes, could you please describe the differences you noticed: (between the 4 versions of BrickOut)
Please be as specific and detailed as possible.

A.4 The final questionnaire at the end of the third phase of the experiment (page 2)

5: Did you notice that the game mechanics in Puzzle would change based on your performance in the game? *
Did you notice the game was changing based on how successfully you were playing?

- Yes
- No

If yes, please tell us which of the 4 versions of Puzzle you noticed this in, as well as how and when you noticed.
Please be as specific and detailed as possible!

6: Did you notice that the game mechanics in Click-and-Hack would change based on your performance? *
Did you notice the game was changing based on how successfully you were playing?

- Yes
- No

If yes, please tell us which of the 4 versions of Click-and-Hack you noticed this in, as well as how and when you noticed.
Please be as specific and detailed as possible!

7: Did you notice that the game mechanics in Electris would change based on your performance? *
Did you notice the game was changing based on how successfully you were playing?

- Yes
- No

If yes, please tell us which of the 4 versions of Electris you noticed this in, as well as how and when you noticed.
Please be as specific and detailed as possible!

8: Did you notice that the game mechanics in BrickOut would change based on your performance? *
Did you notice the game was changing based on how successfully you were playing?

- Yes
- No

If yes, please tell us which of the 4 versions of BrickOut you noticed this in, as well as how and when you noticed.
Please be as specific and detailed as possible!