

Skill-based Progression Model for Smash Time

João Carlos Borges Pereira Catarino

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Dr. Carlos António Roque Martinho

Co-Supervisor: Nuno Miguel Ávila Toscano Monteiro

Examination Committee

Chairperson: Prof. Dr. António Manuel Ferreira Rito da Silva

Supervisor: Prof. Dr. Carlos António Roque Martinho

Member of the Committee: Prof. Dr. José Alberto Rodrigues Pereira Sardinha

June 2018

"Let's optimize for player experience rather than what we think will make more money."

Ron Carmel, World of Goo

To my parents and my brother, for their unconditional love and support through every step of my life.
To Carla, my life partner, for her love and support from the beginning of this work to its conclusion.

Acknowledgements

Firstly, I would like to express my gratitude to my supervisor Prof. Dr. Carlos Martinho, for the continuous support throughout this project. He was always available to share his knowledge and his guidance and helped me in all the time of research, development and writing of this thesis. I could not have imagined having a better supervisor and mentor for my Master's thesis.

I want to thank all the people that helped testing the work developed in this project. They have also contributed to the conclusion of this thesis, and therefore to the conclusion of my Masters Course.

I would like to thank my colleagues at Bica Studios, particularly to Nuno Monteiro for all the hours he has spent brainstorming ideas with me in stimulating discussions and for following the research, development and writing of this thesis. Like my supervisor, he was always available to share his insights on this project with me.

I would like to thank my entire family for all the support during all these years, for always making me feel loved and for encouraging me to follow my dreams. Also I thank my close friends for sharing their friendship with me and for motivating me to finish this thesis and my Masters Course.

Carla, thank you for being my best friend, my confident, my life partner and for always being there whenever I needed you. You were the first person to which I talked, when I first thought about starting a new Masters' thesis and get back to the Masters Course, and you were the first person to motivate me to do it. Words cannot describe the positive impact you had in my life since we started sharing this journey called life. I will always be eternally grateful to you.

Last but not the least, it is time to thank my mother, my father and my brother. Thank you for your love, for never doubting me and for always being able to provide everything I ever needed to make me what I am today. Thank you for everything!

Abstract

Gone are the days the content of a game was completely created manually and the gameplay was pre-scripted before the game release. Nowadays an increasing number of games use procedurally generated content to provide immersive and engaging gaming experiences. Most of the endless single player games generate content just based on the difficulty of the challenges combined with the duration of the current game session. Usually the difficulty setting is achieved through the adaptation of the challenges to a sample of the players population, which leads to a similar gameplay experience to every player and even to the same player in different game sessions.

In this dissertation, we address the problem of keeping the players engaged in a game for longer periods of time generating game content that is modeled according to the player's performance, to improve the gameplay experience.

In order to increase the overall gameplay experience of a game, we should increase the feeling of progression. To solve this problem we propose a progression model that creates content based, not just on the difficulty of the challenges, but also based on the player skill and performance to overcome those challenges. We also propose to control the variety of the challenges to create a better gameplay experience. We theorize that, by providing a progression that is adapted to the player based on the player skill, keeping the variety of the challenges will lead to more engaging and fun gameplay experiences, increasing the replayability in single player games.

We propose a progression model that uses player skill and challenges' variety, in the endless level of the mobile game *Smash Time*, a smasher game with more than 250.000 downloads on the iOS, Android and Windows Phone platforms.

With this dissertation, we believe to have created a skill-based progression model that is robust and dynamic enough to be used in different types of games and that excludes the need of using preset difficulty settings and/or adaptation rules.

The results from playtests with users suggest that the developed skill-based progression model is able to increase the number and duration of the game sessions. The results also suggest that the progression model has the potential to increase player immersion, and consequently to create more engaging gameplay experiences. With the potential to create longer and more engaging gameplay experiences as well as to increase their frequency, comes the possibility to, ultimately, increase the overall lifetime of the game itself.

Keywords: player skill, content variety, progression, procedural content generation, dynamic difficulty adjustment, player modelling

Resumo

Longe vão os tempos em que o conteúdo de um jogo era completamente criado manualmente e a jogabilidade era pré-programada antes do lançamento do jogo. Hoje em dia um número crescente de jogos usam conteúdo gerado procedimentalmente para fornecer experiências de jogo imersivas, envolventes e cativantes. A maioria dos jogos infinitos para um jogador geram conteúdo baseando-se apenas na dificuldade dos desafios combinada com a duração da sessão de jogo. A configuração da dificuldade é normalmente definida através da adaptação dos desafios a uma amostra da população de jogadores, o que leva a experiências de jogo semelhantes para todos os jogadores e inclusive para o mesmo jogador em sessões de jogo diferentes.

Nesta dissertação, abordamos o problema de manter os jogadores cativados e envolvidos num jogo por períodos de tempo mais longos, gerando o conteúdo do jogo, modelado de acordo com a habilidade e desempenho dos jogadores para melhorar a experiência de jogo.

Para melhorar a experiência global de um jogo, devemos aumentar o sentimento de progressão. Para resolver este problema, propomos um modelo de progressão que cria conteúdo baseado, não apenas na dificuldade dos desafios, mas também baseado na habilidade e desempenho do jogador a ultrapassar esses desafios. Propomos também controlar a variedade dos desafios para criar uma melhor experiência de jogo. Nós teorizamos que, fornecendo uma progressão que é adaptada ao jogador baseada nas suas habilidades, mantendo a variedade dos desafios leva a melhores, mais cativantes, envolventes e divertidas experiências de jogo, aumentando a repetibilidade dos jogos para um jogador.

Propomos um modelo de progressão que usa a habilidade do jogador e a variedade dos desafios, no nível infinito do jogo de telemóvel *Smash Time*, um jogo do género smasher com mais de 250.000 downloads nas plataformas iOS, Android e Windows Phone.

Com esta dissertação, acreditamos ter criado um modelo de progressão baseado na habilidade do jogador, que é robusto e dinâmico o suficiente para ser usado em diferentes tipos de jogos e que exclui a necessidade do uso de configurações pré-definidas de dificuldade e/ou de regras de adaptação do conteúdo.

Os resultados dos testes com utilizadores sugerem que o modelo de progressão desenvolvido, baseado na habilidade do jogador, é capaz de aumentar o número e a duração das sessões de jogo. Os resultados também sugerem que o modelo de progressão tem o potencial de aumentar a imersão do/a jogador/a, e consequentemente de criar experiências de jogo mais envolventes. Com o potencial de criar mais longas e envolventes experiências de jogo, assim como de aumentar a sua frequência, vem a possibilidade de, em última análise, aumentar a vida útil do próprio jogo.

Palavras-Chave: habilidade do jogador, variedade de conteúdo, progressão, geração procedimental de conteúdo, ajustamento automático de dificuldade, modelação do jogador

Contents

List of Tables	xi
List of Figures	xiii
1 Introduction	3
1.1 Motivation and Problem	3
1.2 Hypothesis and Approach	3
1.3 Contributions	5
1.4 Organization	5
2 Related Work	7
2.1 Procedural Content Generation	7
2.2 User Experience	8
2.3 User Skill	9
2.4 Discussion Overview	12
3 Testbed Game	15
3.1 Smash Time	15
3.2 Game components	16
3.2.1 Enemies	16
3.2.2 Animals	16
3.2.3 Score System	18
3.2.4 Enemy Sequence	19
3.3 Arena Mode	19
4 Progression Model	21
4.1 Challenges	22
4.2 Obstacles	22
4.3 Tags	23
4.4 Game Cycle	25
4.5 Content Generation Model	25
4.6 Player Performance Model	27
4.6.1 Player Performance Analyzer	27
4.6.2 Player Performance Predictive System	30
4.7 Content Variety Model	30

5	Evaluation	37
5.1	Preliminary Evaluations	37
5.1.1	Procedure	37
5.1.2	Results and Changes	38
5.2	Final Evaluation	38
5.2.1	Procedure	39
5.2.2	Demographic Results	40
5.2.3	Quantitative Evaluation Results	40
5.2.4	Qualitative Evaluation Results	42
5.2.5	Discussion	43
5.3	Summary	44
6	Conclusion	45
6.1	Future Work	46
	Bibliography	48
A	Testbed Game: Smash Time	53
B	Progression Model Pseudo Code	55
C	Playtest Procedure Guideline	63
D	Playtest Game Data Collected Guideline	65
E	Game Experience Questionnaire	67
F	Game Data Collected Questionnaire	69
G	Scoring Guideline Game Experience Questionnaire	71
H	Quantitative Evaluation Results	73
I	Qualitative Macro Evaluation Results	75
J	Qualitative Micro Evaluation Results	77

List of Tables

3.1	Smash Time Enemies' De-evolution Phases.	17
3.2	Smash Time Enemies' Evolution Phases.	18
5.1	Game data collected (average values) during the playtests: Old Arena VS New Arena. . .	41

List of Figures

1.1	Csikszentmihalyi's Flow Channel.	4
1.2	Smash Time mobile game.	4
2.1	Example of a progression graph enabling several mechanics and challenges.	11
3.1	Smash Time promo screens.	15
3.2	Smash Time tap mechanic to smash one enemy.	16
3.3	Smash Time enemy eating an animal and evolving.	17
3.4	Smash Time game over by player smashing an animal.	18
3.5	Smash Time Arena Timer States.	19
3.6	Smash Time Arena HUD.	20
4.1	Arena Challenge Library.	22
4.2	Challenge prefab with 3 obstacle waves.	23
4.3	Arena obstacles' backward evolution phases	23
4.4	Tag categories.	24
4.5	Challenge Editor Window: Challenge with 2 Game Designer Tags assigned.	24
4.6	Game cycle with the Progression Model Architecture.	26
4.7	Game Content Generation (Challenge Library + Content Generation Model).	31
4.8	Player Performance Curve.	32
4.9	Player Performance Model Data.	33
4.10	Example of the player performance dealing with a challenge generated by the Content Generation Model.	34
4.11	Content Variety Model Data.	35
4.12	Content Variety Curve.	36
5.1	Play Games Frequency: Old Arena VS New Arena.	41
5.2	Game Genre Familiarity: Old Arena VS New Arena.	41
5.3	Played Smash Time: Old Arena VS New Arena.	42

Chapter 1

Introduction

1.1 Motivation and Problem

The processes used in game development are constantly evolving since the beginning of videogames as they play each day a more important role in our lives. Millions of people have daily short gameplay experiences with mobile games like Candy Crush Saga [1], Clash of Clans [2] and Clash Royale [3], as well as long entertainment sessions with PC and console games such as World of Warcraft [4], League of Legends [5] and Minecraft [6]. According to the U.S Entertainment Software Association (ESA), in 2017, 67 percent of American households own at least one device to play video games and the average game player age is 35 [7]. With more people joining the videogames world and more games being developed over the years, the games themselves need to adapt and evolve to meet the expectations of the players. Due to their human nature, players will always want to be challenged and surprised when playing a videogame and that is why generating immersive and engaging gaming experiences can be the ultimate goal to achieve in modern game design and development.

In the past, most of the game content was created manually by the game designers and the gameplay has been pre-scripted before the game release. Although this method is still used nowadays, an increasing number of video games are using procedurally generated content to provide more immersive and engaging gaming experiences.

In this dissertation, we address the problem of keeping the players engaged in a game for longer periods of time generating game content that is modeled according to the player's skill and performance and needs, to improve the gameplay experience. More specifically, we attempt to give answers to the following question: how to generate game content to create more engaging gameplay experiences that may lead to longer gameplay experiences.

1.2 Hypothesis and Approach

Most of the endless single player games generate content just based on the difficulty of the challenges and usually the difficulty setting is achieved through the adaptation of the challenges to a sample of the players population. This method leads to a similar gameplay experience on every player and even on the same player in different game sessions, which can result in less fun and engaging experiences reducing the replay value of the games. In this dissertation we address the problem of how can we generate game content to create more engaging gameplay experiences.

In order to increase the overall gameplay experience of a game, we should increase the feeling of progression. Csikszentmihalyi[8] concluded that a person's skill and the difficulty of a challenge can

create different emotional states on that person. He found that people become bored when dealing with an easy task if their skill level is high and, alternatively, they become anxious when dealing with a difficult task if their skill level is too low. Following Csikszentmihalyi studies and his definition of "Flow Channel" (Figure 1.1), one can conclude that a good and engaging game flow can be achieved by connecting, in the same proportion, the player ability and the game difficulty by challenging the players according to their skill. To solve this problem we propose a progression model that creates content based, not just on the difficulty of the challenges, but also on the player skill to overcome those challenges.

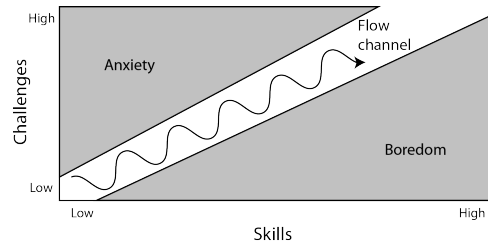


Figure 1.1: Csikszentmihalyi's Flow Channel.

As an extra element to create a better experience, we also propose to control the variety of the challenges. The variety of a challenge represents its novelty, i.e., whether the challenge characteristics have been previously presented to the player. This is achieved with the classification of the challenges' pace, number of obstacles, game designer challenge description and type of obstacles that compose the challenge.

We theorize that, by providing a progression that is adapted to the player based on his/her skill, keeping the variety of the challenges, may lead to more engaging gameplay experiences, creating a stronger connection between the player and the game, and possibly resulting in longer gameplay experiences.

We propose a progression model that uses player skill and content variety, in the endless level of the mobile game *Smash Time* [9], a smasher game with more than 250.000 downloads on the iOS, Android and Windows Phone platforms (Figure 1.2). This game already has a progression model to generate content, but like most the single player games it just analyses the difficulty of the challenges. More specifically, this game uses the classification of the challenges set by the game designer as easy, medium or hard and has different probabilities to choose the next challenges' difficulty according to the time length of the gameplay session.



Figure 1.2: Smash Time mobile game.

With this dissertation we hope to create a skill-based progression model that is robust and dynamic enough to be used in different types of games and that excludes the need of using preset difficulty settings and/or adaptation rules.

1.3 Contributions

The work presented in this dissertation surveys the state of the art of research on player skill modeling and Procedural Content Generation. This work describes a progression model for endless single player video games based on two different dimensions, the performance of the player relative to the challenges that the game creates autonomously and the variety of the generated content. This model was implemented in a commercial mobile video game, available on the Apple, Google Play and Windows Phone stores. Finally, the implemented progression model was tested with users in order to validate the proposal presented in this study.

This work reinforces the importance of integrating the skill dimension into the way Procedural Content Generation is used in video games, through the autonomous generation of content based on the player performance dealing with the challenges created and adapted by the game to a specific player at specific moments on the play session. This work intends to demonstrate, that combining the skill dimension with the variety of the generated content will provide more engaging gameplay experiences in single player endless video games, increasing the duration of the gameplay experiences.

1.4 Organization

This dissertation is organized into 6 chapters and 10 appendices as follows:

Chapter 1, "Introduction", is this one and serves to introduce the reader to the problem that was considered in this dissertation, our approach to deal with that problem and our contributions. It also has the purpose to motivate the reader to know more about this dissertation and to serve as a guide through this document.

Chapter 2, "Related Work", reviews the state of the art of the different research fields related to the work presented in this dissertation.

Chapter 3, "Testbed Game", introduces the testbed game used to implement the progression model developed on this research work.

Chapter 4, "Progression Model", describes the proposed skill-based progression model, its components, concepts and implementation in the testbed game.

Chapter 5, "Evaluation", investigates the efficiency of the implemented progression model, describing the evaluation process done with real players and the results that were gathered from this evaluation.

Chapter 6, "Conclusion", summarizes the thesis main achievements and contributions and discusses the proposed progression model current limitations. This chapter also describes future research steps beyond the limits of this dissertation to improve the capabilities of the presented progression model.

Appendix A, "Testbed Game: Smash Time", contains extra information about the testbed game used to implement the progression model.

Appendix B, "Progression Model Pseudo Code", contains the main pseudo code of the progression model.

Appendix C, "Playtest Procedure Guideline", was the guideline used to conduct the playtest sessions.

Appendix D, "Playtest Game Data Collected Guideline", was the guideline used to implement the code that would track and register the game data of the playtest sessions.

Appendix E, "Game Experience Questionnaire", was the questionnaire used to know how players felt while playing the testbed game with the proposed and implemented progression model.

Appendix F, "Game Data Collected Questionnaire", was the questionnaire used by us to gather the registered data by the game during the evaluation sessions with the users.

Appendix G, "Scoring Guideline Game Experience Questionnaire", contains the 7 components, and their items, used to evaluate qualitatively in a macro point of view the results of the playtests.

Appendix H, "Quantitative Evaluation Results", summarizes the quantitative results from the playtests with users.

Appendix I, "Qualitative Macro Evaluation Results", summarizes the qualitative results, in a macro point of view, from the playtests with users.

Appendix J, "Qualitative Micro Evaluation Results", summarizes the qualitative results, in a micro point of view, from the playtests with users.

Chapter 2

Related Work

This chapter starts with a brief description of what is Procedural Content Generation (PCG) and some of its goals followed by a survey on previous work on PCG and its use in the gaming industry. We finish the current chapter with an overall discussion about the related work bearing in mind the focus of this work.

2.1 Procedural Content Generation

PCG is the process in which, computer software, algorithmically generates on the fly, game content with limited or indirect user input [10] [11]. The algorithmically generated game content can be anything from parts of a level or a map to complete levels and maps, game rules, 2D textures and 3D models, characters and items, music, stories and side quests, etc. In this section we will make a quick survey through some of the most iconic and important games which have paved the way PCG can be used in games and some interesting games that were groundbreaking on the way they used PCG to generate its content.

One of the first games to use PCG was Rogue [12], to generate the game dungeons where the players could navigate. Also used to generate dungeons, there is the popular Diablo series with Diablo I [13], Diablo II [14] and Diablo III [15]. In role-playing games like Diablo and others, PCG is also used to generate the items the player can catch and the enemies that he/she will encounter. In Borderlands [16] unique items and weapons are created and found in chests, on the ground, dropped by enemies, sold at vendors in the game and got as rewards in quests, using PCG techniques. The algorithms used in Borderlands change properties like the firepower, rate of fire and accuracy of weapons and were able to generate over 3.000.000 different weapons [17]. PCG can be used to create game worlds that are virtually infinite like the one in Minecraft [6], that is generated as players explore it. The world map is generated using a map seed that is obtained from the system clock at the time the world was created. The generation of such an amount of content is possible due to the splitting of the world into smaller sections called chunks that are only created or loaded when a player is nearby.

Bethesda Softworks developed Radiant AI, a technology for Elder Scrolls IV: Oblivion [18] later expanded and improved for Elder Scrolls V: Skyrim [19]. The Radiant AI technology is composed by the Radiant AI system, that generates goals to the NPCs and deals with NPC interactions and behavior to let them determine how to achieve those goals, and by the Radiant Story system, a quest system that procedurally generates infinite quests to give the players new tasks based on their progress level in the game. In dreesps: Alarm Playing Game [20], PCG is used to generate AI behavior to allow the game to adapt to the player life rhythm. The first-person shooter video game .kkrieger [21] creates all the texture

assets of the gameplay during the loading phase, resulting in longer loading times but allowing the entire game to use a little amount of disk space compared to other games. SpeedTree [22] is a 3D Procedural vegetation middleware software with the ability to generate 3D models of trees and plants to be used in video games. Proteus [23] uses PCG to generate the music in the game, it analyses the player's location and movements and adapts the soundtrack to it, for instance, it may be silent when the player is at the top of a hill and become more intense as he/she travels down the hill.

Spore [24] is an example of the potential that PCG can offer to video games due to the extensive use of PCG algorithms in different areas. In Spore, the developers used PCG techniques to generate planets, textures, animations and music. Spore uses user-generated content, with the Spore Creature Creator in-game tool, to procedurally generate player-driven textures, materials and animations. When creating a new creature, players first shape the torso of the creature, then they have the possibility to add parts such as eyes, noses and mouths, arms and hands, legs and feet. The choices made can just have a visual impact on the creature but may also affect the creature's abilities like the speed, strength and other stats. Once the creature is formed, PCG techniques are used to, depending on the topology of the creature, procedurally apply textures, overlays, colors and patterns according to the painting choices of the players, as well as to determine how the creature should walk, swim, dance, drag an object, eat, etc., and procedurally generate animations for these behaviors based on the creature model that was created by the player.

An example of how PCG can be used to create game content can be found in No Man's Sky [25]. If we analyze closely how PCG was used in this game, we can say that there are no limits of what can be done using PCG techniques. Stars, planets and their ecosystems (flora, fauna and their behavioral patterns), artificial structures, alien factions, their spacecraft and encounters with the players are created through procedural generation. This means that nearly all elements of the game are procedurally generated, leading to there being over 18 quintillion planets to explore within the game. This content is generated using deterministic algorithms and random number generators from a single seed number, hence very little data is stored on the game's servers, as all the elements of the game are created through deterministic calculations when the player is near them. This method assures that other players will see the same elements by travelling to the same location in the galaxy. Temporary changes on the planets, like mining resources, are not tracked once the player leaves that vicinity, but major changes, like destroying a space station, are tracked for all players on the game's servers. Additionally, in No Man's Sky, missions and the game's audio are also procedurally generated.

There are a lot of PCG uses in commercial games in the video games industry as shown above. PCG can also be used to analyze the player experience and skill to generate game content as we will show in the sections 2.2 and 2.3 of this chapter.

2.2 User Experience

The player is the main element when we talk about the experience created by the game. In order to develop good games with mechanisms that boost the experience we need to have a good understanding about the player, like their: motivations such as needs, preferences, interests, expectations, values, fears and dreams; limitations; capabilities; knowledge; and the context in which they play a game like, with who, where and when they play a game. Gathering this information, we are able to create a player profile that will be useful to create better game experiences [26]. As Chen [27] pointed out, each player is different and experiences the same games in different ways, due to their personality, skills and expectations when playing a game. This suggests that to satisfy different types of players, the game should be able to adapt itself to the preferences of the player.

Focusing in the player experience and in order to generate effective and meaningful content, Yanakakis and Togelius [28] proposed a framework for PCG driven by computational models of user experience based on the personalization of user experience through affective and cognitive modeling combined with real time adjustment of the content according to user needs and preferences. Also focusing in the player experience, Dias and Martinho developed a personality-based framework to adapt videogame content to the player [29]. Their framework infers the player type, based on the player's Myers-Briggs personality type (conqueror, manager, wanderer and participant), from his/her behavior and decides how the content should be managed and presented to the player based on the inferred player type. In this research a videogame was developed, called Grim Business, to show that being aware of the player type can improve the player's experience, based on difficulty management, presentation and control depth over certain aspects of the game, resulting in higher player immersion and enjoyment. Shaker et. al [30] used Super Mario Bros [31] to demonstrate how to collect players' data to accurately model player experience and tailor game content generation according to the player behavior. They collected data from hundreds of players playing Infinite Mario Bros [32], related to content features, gameplay features and reported player experience to tailor player experience in real time through automatic game content generation, based on computational models of in-game player experience.

There are some videogames that also focus their content generation in the user-experience. Spore [24], is an example about user-experience driven PCG as it extensively records and analyses players actions to generate content that have a huge impact on the players' gameplay experience of the game. Spore is separated into 5 stages: the Cell Stage; the Creature Stage; the Tribal Stage; the Civilization Stage; and the Space Stage. Each stage offers a different type of experience with different goals to achieve. The players have the option to advance to the next stage once the main goal is achieved, or to continue playing the current stage as long as they wish. The outcome of one stage affects the initial conditions of the next stage. When a player progress to the next stage, the player's actions are analyzed and used to assign a characteristic to the player's creature. Each stage has some characteristics, based on how aggressively or peacefully the stage was played, that determine how the creature will start the next stage and which abilities it will have, to be later used in the game.

2.3 User Skill

Focusing in the player, Cook [33] described the player model as "The player is entity that is driven, consciously or subconsciously, to learn new skills high in perceived value.". In this context, a skill is a behavior that a person uses to manipulate the world. Some skills are physical, such as making a sculpture while others are on the conceptual domain, such as observing a map and planning the best path to go from one point to another. Cook states that, when players learn something new and can use that knowledge to successfully manipulate the environment for the better, they experience joy and gain pleasure for that achievement. Furthermore, to create enjoyable gameplay experiences, the game should demand full concentration from the player because when a person needs most of his/her skills to deal with a challenging situation, his/her attention is completely absorbed by the activity in question leaving no excess attention and focus to process anything else besides that activity [8].

There are several ways a game can provide different difficulty levels, adapting itself to different players with different skill levels. Changing the difficulty of game can be achieved, for example, increasing enemies' attributes in a first person shooter game, such as shooting accuracy, damage, amount of life and protection gear, resulting in a decrease of the player survivability. We can also tweak the opponents' car stats in a racing game such as power, top speed, acceleration and shift time to increase the challenge the players have to reach the top positions in a race. The simplest way to do this, is to ask the player,

in the beginning of the game, which is his/her skill level, usually with the options: beginner, intermediate and expert, corresponding to the difficulty levels: easy, medium and hard. This method can lead to not so good gameplay experiences since players may not be completely aware of their skill level. Some games try to understand the player skill level tracking and recording their actions in training levels. In the beginning of Call of Duty: Modern Warfare 2 [34], players have access to a training camp to learn how to play the game, followed by a course test called The Pit, where the player has to clear enemy locations with minimal civil casualties in the shortest time possible. The Pit is used to test players' skill level, tracking their actions and analyzing players' statistics such as: total time; civilians killed; and shooting accuracy. After running The Pit, the game suggests a difficulty for the player based on their skill and performance, but the player can choose to continue on any difficulty.

Another way to know the player skill level, and therefore adapt the difficulty of the game to his/her skill level is to give the player several tasks and set the difficulty level according to the success rate performing those tasks. Quake Live [35] use this technique, providing a training center when the game starts, where the player has 3 portals to enter: Beginner; Intermediate; and Expert. At start, just the beginner portal is available, and the player needs to use a technique familiar to players of previous Quake games' like the famous Quake III Arena [36], called *rocket jump*, where the player uses the explosion from a rocket combined with a jump to reach high and/or far places, to reach the Intermediate portal. After entering the Intermediate portal, an advance challenge, easily done by expert Quake players, is given to the player, where he/she needs to perform a series of fast jumps in a limited amount of time to reach the Expert portal. After entering the portal, the player needs to fight a non-player character (NPC) controlled by the game AI. While he/she plays against the NPC, the opponent AI will, if needed, reclassify the player skill and automatically adjust the difficulty setting in order to better face the player's skills. Quake Live uses this tutorial to assess the player skill level, so that it gets online games against players with a similar skill level, through the game skill-based matchmaking system.

Different players have different skill levels. Even if two players start playing the same game at the same skill level, they will develop their skill at different rates, which means that the same game can become frustratingly difficult for some players and boringly easy for others. One of the ways to deal with this problem is to use Dynamic Difficulty Adjustment (DDA), where the game adapts itself during play in response to players' skill and performance evolution.

Since this dissertation proposes a progression model to an endless single player game mode, we will now focus on some examples on how to progressively increase the game difficulty in endless single player video games. This progression can be achieved by tracking the duration of the game session and making some changes to the gameplay such as increasing the speed of the gameplay and changing some features of the game content. Subway Surfers [37] is an endless runner for mobile platforms, where the player controls a character that is running from a policeman in a railway track. The game increases its speed as the player progresses through the railway track, giving progressively less time for the player to react and avoid the obstacles that appear on the way and thus increasing the difficulty of the game as time goes by. In Smash Time [9], another mobile game, there is the arena game mode, an endless level where the player needs to tap and smash enemies that appear on the screen trying to attack a hero, at the bottom of the screen, that teams up with the player. Besides progressively increasing the speed of the enemies that come in waves, the game also changes the generated content type as the game session advances, as it sends enemies that spawn other enemies after being killed, and thus resulting in more taps needed to overcome an enemy wave. The negative side of both approaches is that they only relate the difficulty of the procedurally generated content with the duration of the game session, not bearing in mind player's skill dealing with the given obstacles (content). As in both cases the game uses DDA techniques that are based on the game designer intuition instead of reflecting the actual player skill level, both progression techniques end up generating similar gameplay experiences for

different players and even for the same player as the player skill increases through several game sessions.

Polymorph [38] is a 2d platformer game, that combines machine learning with level generation techniques to understand the level difficulty and the player skill and dynamically construct levels with continually suited challenge. As Jennings-Teats et al. claim, “the difficulty in 2d platformer levels is related to the combinations of adjacent level components more than to the presence of a particular level component”. Hence, Polymorph’s machine learning model of difficulty uses level segments combined with a dynamic model of the player’s current performance to generate the level in front of the player. It attempts to give the players the appropriate level of challenge while trying to avoid making them bored or frustrated, if the game is too easy or too hard, respectively.

Pereira [39] developed a progression modeling tool to understand the player skill evolution in order to provide appropriate challenges in a 2d platformer game. This tool uses a library of challenges combined with a skill-based player model that is based on the player mastery level (e.g. uninitiated, partially mastered, mastered) for specific challenges (e.g. wall, hole) and game mechanics (e.g. jump, slide, double jump) to know when to unlock new and more difficult challenges and mechanics. In this progression model, a challenge becomes mastered after several challenges of that type are successfully overcome and a mechanic becomes mastered when its used several times to overcome a challenge. The progression is guided by preset adaptation rules created by the game designer, that specify how the progression of the game should evolve according to the player’s skill evolution. The game designer creates a progression graph that specifies the dependencies and constraints between all the challenges the game can generate and the mechanics that can be used by the player. The game starts with few or just one mechanic and challenge unlocked to the player and, as the player skill evolves the game starts unlocking new and more difficult challenges and mechanics, according to the progression graph defined by the game designer. An example of a progression graph can be seen on Figure 2.1.

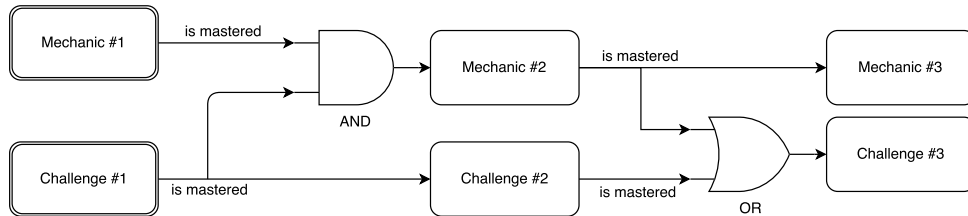


Figure 2.1: Example of a progression graph enabling several mechanics and challenges.

Zook et al. [40] proposed a model for skill-based mission generation that tries to solve 2 problems: *challenge tailoring*, or in other words “the problem of matching the difficulty of skill-based challenges over the course of a game to match player abilities”; and *challenge contextualization*, related to the fact that the game should provide appropriate motivating story context for the skill-based challenges and thus resulting in the creation of story content that motivates game play in between challenges. To deal with the *challenge tailoring* problem, the model must find a sequence of challenges that produce a given progression of predicted player performance. To achieve this, the game designer specifies a performance curve that determines the wanted progression of the player’s performance over the course of a mission. As a result of this study, they propose a five criteria data-driven player model: predictive power; accuracy; efficiency; generative sufficiency and temporality to guide skill-based mission generation combined with story events.

2.4 Discussion Overview

Inspired on the work of Chen et al. [27] and Martinho et al. [26] about the importance of the player in the development of videogames, we will try to value each player individually, with his/her own capabilities and limitations, in order to provide a different and personal gameplay experience to each one of them trying to satisfy different types of players with the same game. Furthermore, we will create a player model, that will be updated over the course of time and play sessions, trying to give each player a different and appropriate experience every time he/she plays the game according to the evolution of his/her skills.

We believe the DDA techniques used in Call of Duty: Modern Warfare 2 [34] and Quake Live [35] are a good approach to adapt the levels to different players with different skills, although we want to avoid, in one hand, asking the players to guess and define their skill level so that the game can adapt its difficulty to their skill level and, in the other hand, to force the players to play a training level to allow the game to understand their skill and abilities. In order to avoid these situations, we want to focus in DDA techniques where the game adapts itself during play in response to players' skill and performance rather than in the game designer's intuition of what challenges are appropriate to a specific skill level or the player's intuition on which skill level he/she has.

Using the concept of Polymorph [38] of continuous level difficulty we will create a player model that records the player skill in response to the last N-challenges presented. In this point of view the difficulty of a challenge and its obstacles is not separate from the previous challenges and their obstacles. This approach allows the progression of the game to match the progression of the player's skills dynamically in order to avoid making the player bored if the game is too easy or frustrated if too hard.

We will base our progression model in the skill-based mission generation model of Zook et al. [40] with two components, *challenge tailoring* and *challenge contextualization*. The first component will work in the same way, since the model must find a sequence of challenges that produce a given progression of predicted performance. This will be done also using a game designer specified performance curve that determines the progression of a player's performance over the course of a game session. The second component will be slightly adapted since the context of the challenges will be done according by the type of obstacles that compose the challenges, their patterns and paces to provide a good variety of the challenges given to the players.

Like in the work developed by Pereira [39], we will try to understand how the player skill evolves, also using the concept of game content being presented in the way of challenges, to provide a challenging and engaging game progression to the players. In contrast with that progression model, that uses a progression graph of dependencies and constraints between challenges, preset by the game designer to unlock new content, all of the content in our progression model is unlocked at the start of the game. The approach in the presented progression model, is that is going to be the player's skill evolution to show the game that the player is ready for more challenging content, according to the defined performance curve defined by the game designer. With this approach we want to avoid, as the game dimension and complexity increases, the task of the game designer to create new challenges not to become too difficult and time-consuming, due to the fact that the challenges do not need to be interrelated, as it happens with a progression graph and all its dependencies and constraints between the challenges. Another reason for the exclusion of the progression graph is that, without it, we also remove the game designer's intuition factor, about what would be the difficulty level of each challenge and the decision of all the possible connections and dependencies between the challenges. Instead, the game designer has a simpler task, that is to define a performance curve that will allow the game itself to choose which content to provide to the player. The proposed progression model also uses a library of challenges in which each challenge can be created in a modular way without dealing with its relation with all the other

challenges already existing in the challenges' library. This leads to a less time-consuming and easier challenges' creation task for the game designer. Finally, with the exclusion of the progression graph, the game is able to provide different gameplay experiences each time a player plays the game, instead of providing always the same progression evolution for all the players in all game sessions.

Another two examples of games that also provide always the same progression, which leads to a rigid gameplay evolution and with that, similar gameplay experiences for all the players in all game sessions, are the progression models used in Subway Surfers [37] and Smash Time [9]. To avoid this, we will, like explained before, remove the game designer's intuition factor about the challenges' difficulty and also about how should the speed of the game increase progressively, to focus on the real player skill, and its evolution, as he/she plays the game.

Bearing what was addressed in this chapter and discussed in this section, we created a progression model that can successfully create challenging, engaging and varied gameplay experiences.

Chapter 3

Testbed Game

This chapter presents Smash Time, the video game used as testbed game to implement and test the progression model described in this work. Smash Time^{1 2} is a smasher game developed by me and my colleagues at Bica Studios, with Unity [41], for smartphones and tablets.

3.1 Smash Time

Smash Time has fast gameplay mechanics, that result from the combination of elements from classic games like Whac-a-Mole and Space Invaders, mixed with puzzle mechanics. There are two game modes available in the game, the Campaign and the Arena.

In Smash Time, the world is being devoured by alien blobs that came from the outer space and rapidly started eating everything in their way making them grow bigger and evolving to stronger creatures. They must be stopped in order to save the Universe, and to help the player in this task there is Bica, a cosmic guardian with the power to unite worlds through the player's smartphone or tablet (Figure 3.1).



Figure 3.1: Smash Time promo screens.

In this chapter we are going to explain in detail the game components, the Arena game mode³ and its gameplay, focusing on the information related to the scope of this work.

¹iOS version available on the App Store: <https://itunes.apple.com/pt/app/smash-time/id948782612?mt=8>

²Android version available on Google Play: <https://play.google.com/store/apps/details?id=com.bicastudios.smashtime>

³Information about the Campaign game mode, the other game mode of Smash Time, can be found in Appendix A

3.2 Game components

Smash Time characters are enemies, animals and heroes ⁴, that coexist in the same world. With the help of a team of heroes, the player must save the animals by defeating the enemies. This section contains a description of the characters and game components that are relevant in the scope of the developed progression model.

3.2.1 Enemies

The enemies have different colors, sizes, movement and attack mechanics. Enemies enter the screen from the top and both sides and try to attack, both the hero that is, at the bottom of the screen, helping the player and the animals that are trying to escape from them to survive. The player goal is to smash the enemies and clear all the incoming waves (groups of enemies). To smash one enemy and receive one or more points for it, the player must tap on it with a finger, one tap for normal enemies and multiple taps for the bosses. An example of an enemy being tapped and smashed by the player is presented in Figure 3.2.



Figure 3.2: Smash Time tap mechanic to smash one enemy.

There are 4 normal enemies on Smash Time and some of them spawn a new enemy when killed by the player, as the Table 3.1 shows.

3.2.2 Animals

As said before, one of the main goals of the player is to save as much animals as possible from the hungry enemies. The animals are running around like crazy, they enter and exit the screen from the top and both sides and some of them also run away in the direction of the hero, exiting the screen from its bottom. Some of the enemies eat animals if they have the chance to get close enough to them, evolving into a new type of enemy as shown in Figure 3.3.

The evolution states of the enemies follow the opposite way of the enemies' de-evolution order shown previously (Table 3.2).

⁴Additional information about the heroes of Smash Time can be found in Appendix A
















Enemies' Devolving Phases			
Enemy	Player Mechanic	Game State Changes	
		Killed Enemy	Dead Enemy Spawns New Enemy
Red	TAP	Smashed Red Enemy	Spawned Green Enemy
			
Green	TAP	Smashed Green Enemy	Spawned Blue Enemy
			
Blue	TAP	Smashed Blue Enemy	Spawned Purple Enemy
			
Purple	TAP	Smashed Purple Enemy	No Enemy Is Spawned
			—

Table 3.1: Smash Time Enemies' De-evolution Phases.



Figure 3.3: Smash Time enemy eating an animal and evolving.

Hence, the player's task is to defeat the enemies to stop them before they get too close to the animals. The player must be careful to not tap on an animal because, the animals can also be smashed if the player taps on them, resulting in a game over situation (Figure 3.4).











Enemies' Evolving Phases		
Enemy	Enemy Mechanic	Game State Changes
		Enemy Evolves Into New Enemy
Purple	Eat Animal	Purple Enemy Evolved To Blue Enemy
		
Blue	Eat Animal	Blue Enemy Evolved To Green Enemy
		
Green	Eat Animal	Green Enemy Evolved To Red Enemy
		
Red	Red Enemy Does Not Eat Animals	Red Enemy Does Not Evolve
	—	—

Table 3.2: Smash Time Enemies' Evolution Phases.



Figure 3.4: Smash Time game over by player smashing an animal.

3.2.3 Score System

The score multiplier value goes up and down according to the player actions and has the minimum value of 1 and does not have a maximum value. Each smashed enemy gives 1 point multiplied by the score multiplier value. As seen in Figure 3.2 the game rewarded the player, for smashing one enemy, with 2 points because the score multiplier had the value of 2. However, when the player misses one tap on an enemy and touches the floor of the scenario, the value of the score multiplier decreases one unit. The score multiplier can be increased by completing enemy sequences.

3.2.4 Enemy Sequence

The puzzle component of the game is presented by a secondary goal that the player can also focus on. A sequence of enemies is shown to the player representing the order that the enemies should be smashed to increase the score multiplier value.

Players can smash any enemy, at any moment at their will, always being rewarded with points for each successfully killed enemy. The advantage of paying attention to the enemy sequence and trying to follow it, is that at each enemy smashed corresponding to the next enemy on the sequence, the players gets closer to complete the enemy sequence. As mentioned above, when the player completes one enemy sequence, the score multiplier value is incremented by one unit. Finally, with higher score multiplier values, the player increases the chances of achieving higher score results and better rewards.

3.3 Arena Mode

There are online events, where players compete, to reach the highest positions in the leaderboards and with that achieve better rewards, called Arenas. An Arena run starts with 60 seconds (Figure 3.5a), and ends when the timer gets to zero seconds (Figure 3.5c) or when the hero gets attacked by the enemies as shown on Appendix A.

In this game mode, players try to play as much time as possible, to reach the highest score possible, on an infinite level with a timer that can be extended to the maximum of 99 seconds. According to the player's performance against killing the enemies of the enemies' sequences, that are generated by the game, they are rewarded with the possibility to extend the timer, by tapping and smashing a Golden Bouncini, one special type of enemy that appears more often the larger the number of enemies' sequences completed by the player. Figure 3.5b shows a player killing a Golden Bouncini to get a time extension reward of 10 seconds.

This game mode does not include animals running from the enemies, to increase the tap frenzy effect and remove the special attention that the players need to pay to the different types of obstacles (enemies and animals) and if they can tap on them or not.



(a) Start State (Time = 60s).

(b) Time Extension (Time = Time + 10s).

(c) Final State (Time = 0s).

Figure 3.5: Smash Time Arena Timer States.

Figure 3.6 shows the heads-up display (HUD) of the arena game mode with: an enemy sequence combo given to the player; the score multiplier, the score the player reached so far, the arena timer and the next player on the leaderboard according to the score achieved up to that point. The numbers in the figure have the following meanings:

1. Arena timer showing the time left on the current game session;
2. Sequence of enemies to smash;
3. The score multiplier. Each enemy sequence has a time limit to be completed, this is shown by the timer bar behind the multiplier number that slowly empties until being completely empty, generating a new enemy sequence at that point. If the player is able to successfully smash all the enemies of the sequence inside the time limit, the score multiplier increases one unit;
4. Score achieved so far;
5. Avatar and score of the next player on the leaderboard.



Figure 3.6: Smash Time Arena HUD.

As a result of this work, a new progression model was created and replaced the original progression model of the arena game mode. The old progression model that has originally been used in the arena game mode of the testbed game is described below.

An arena game session was regulated according to the duration of the arena run itself. There were 3 difficulty settings: easy; medium; and hard. For each difficulty there were enemies' waves preset. Each 30 seconds the difficulty phase of the arena was updated to the next one, which means that after 30 seconds the game changed to the medium difficulty phase and after another 30 seconds the game changed to the hard difficulty phase, where it would stay until the end of the game session.

For each difficulty phase, the game designer had to set the probability of the difficulty of the enemy waves (groups of enemies) that were randomly selected. Besides the type of enemy waves that were selected, according to each difficulty phase probabilities, the speed of the enemy waves was constantly increasing in relation to the elapsed time since the beginning of the arena run. This means that one enemy wave could be used several times in the same difficulty phase (easy, medium or hard) with different speeds, according to the elapsed time so far in the arena run.

Chapter 4

Progression Model

In this dissertation thesis, we propose a skill-based progression model for endless single player videogames and the present chapter will serve to explain our proposal in detail. The progression model developed in this dissertation thesis was implemented, using Unity engine, on top of the Arena game mode of Smash Time [9], the testbed game discussed in the previous chapter.

The main goal of our progression model is to provide immersive and engaging gameplay experiences that will make players want to play more often and for longer periods. To achieve this, we kept in mind these principles:

1. the game should allow and support player skill development;
2. the game should be constantly challenging, with care to avoid not becoming too difficult, and therefore stressful for the players, nor too easy, and consequently boring for the players;
3. the game should match the player's skill level at all times throughout a game session, increasing the level of challenge as the player moves forward through the game and increases their skill level;
4. the game should provide different levels of challenge for different players;
5. the game should vary its content and provide new challenges at an appropriate pace.

Following these principles, we propose a skill-based progression model that is composed by the following components:

- Concepts:
 - Challenges;
 - Obstacles;
 - Tags.
- Models:
 - Player Performance Model;
 - Content Variety Model;
 - Content Generation Model.

In this chapter, we will analyze the progression model concepts and components. More specifically, how the *Challenges*, *Obstacles* and *Tags* are used by the three main components that compose the core of the developed progression model: the *Player Performance Model*; the *Content Variety Model*; and the *Content Generation Model*.

4.1 Challenges

Challenges are formations created by the game designer with one or more paths (or wave paths) that guide the movement of the obstacles. The game designer creates a library of challenges that are stored as Unity prefabs (game objects with components and properties that can be used as templates to create new object instances in real time) (Figure 4.1), to be used by the progression model later in real time.

A challenge is composed by a set of waves that are going to spawn a set of obstacles that will move with a certain speed (challenge pace) in a specific wave path. A wave path is defined by a set of waypoints (intermediate points that are connected to form the path that will be followed by the obstacles of the wave) that are specified by the game designer. The quantity and type of each wave's obstacles, as well as the challenge pace are defined by the progression model in real time. An example of a challenge composed by 3 different waves (each with the wave path composed by 3 waypoints) can be seen in Figure 4.2.

The progression model has the task to generate new challenges as the game progresses, while the player has the task to overcome those challenges.

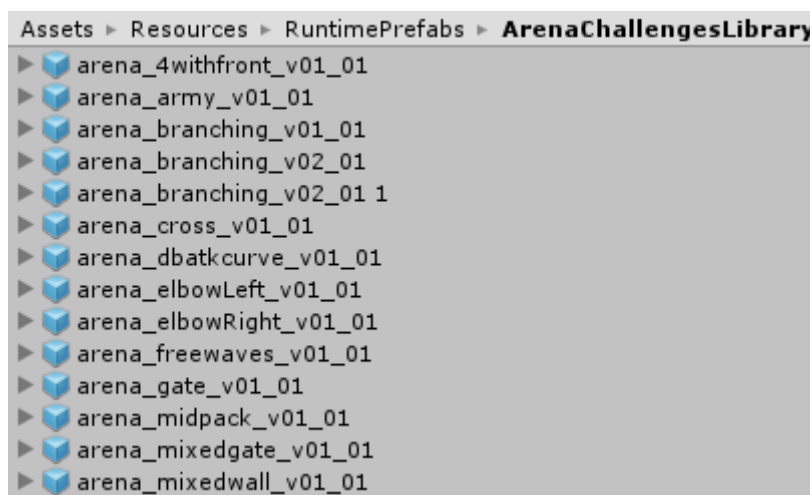


Figure 4.1: Arena Challenge Library.

4.2 Obstacles

An obstacle can be something that requires the player to use his/her skills to be overcome. In the context of the testbed game, the obstacles of the Arena game mode can be: a Red Enemy; a Green Enemy; a Blue Enemy; and a Purple Enemy. To overcome each obstacle, the player must tap on it. When the player taps on an enemy, it smashes and kills that enemy. Some enemies generate another enemy after being killed by the player. The Figure 4.3 illustrates the arena obstacles' backward evolution phases when the player smashes one enemy with a tap.

Even though each obstacle is a different enemy and all of them have one life, meaning this that they die with a single tap from the player, as some enemies generate other enemies, we consider that an obstacle is completely overcome just when the Purple and last enemy in the de-evolution order is smashed and killed. On the basis of this premise, we can distinguish an enemy that was spawned inside one challenge wave and an enemy that was spawned from another enemy that was already alive.

Therefore, let's look into the following example: A challenge has one wave that spawns a red enemy obstacle, the player taps on the red enemy obstacle and it spawns a green enemy obstacle, then the

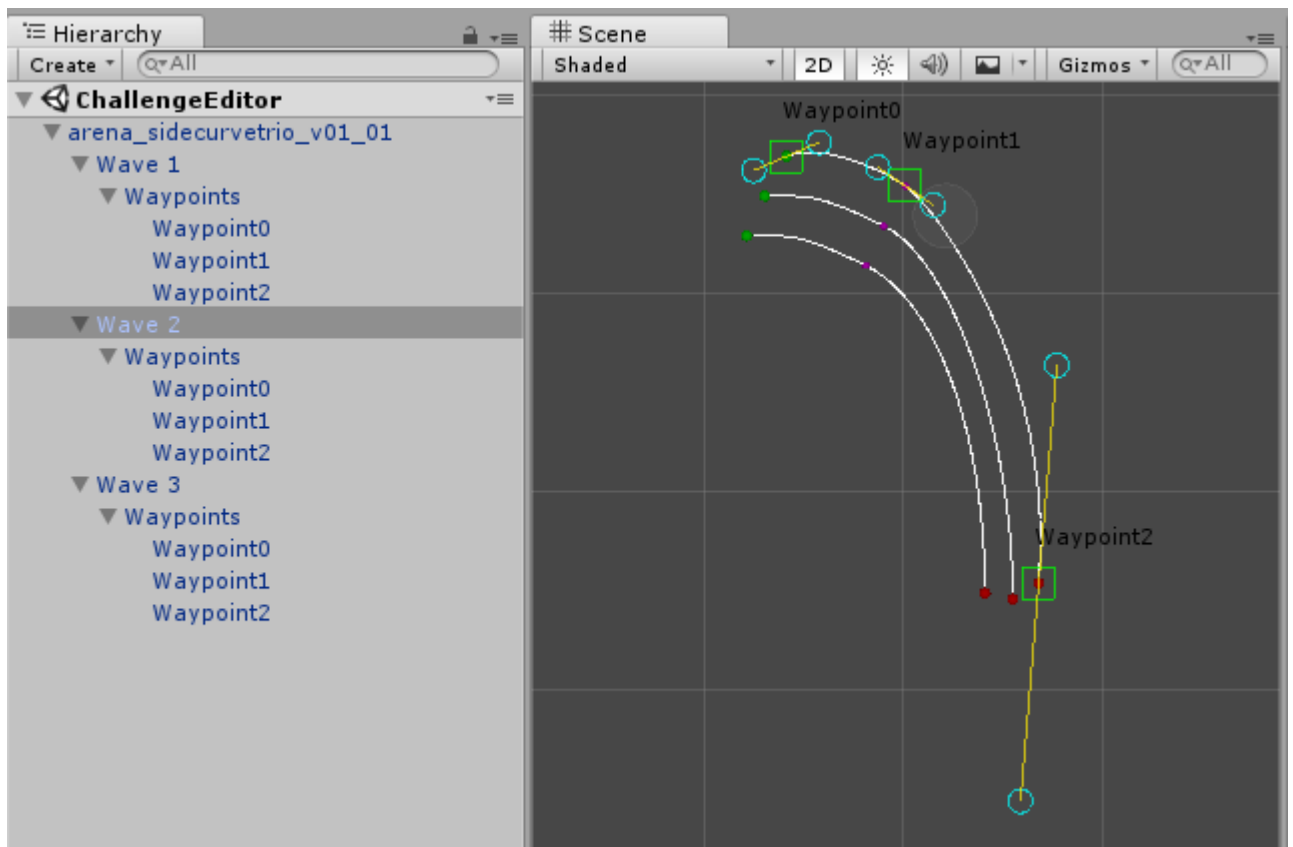


Figure 4.2: Challenge prefab with 3 obstacle waves.









Obstacle	Mechanic	Obstacle	Mechanic	Obstacle	Mechanic	Obstacle	Mechanic	Obstacle
Red Enemy	→ tap →	Green Enemy	→ tap →	Blue Enemy	→ tap →	Purple Enemy	→ tap →	-
								-

Figure 4.3: Arena obstacles' backward evolution phases

player taps on the green enemy obstacle and it spawns a blue enemy obstacle, when the player taps this blue enemy obstacle a new purple enemy obstacle is spawned, and finally when the player taps on the purple enemy obstacle no more enemies are spawned. Hence, when that happens, we consider that the original red enemy obstacle was completely overcome with the smashing of the red, green, blue and purple enemy obstacles. With all obstacles overcome, the progression model considers that the challenge was also overcome.

The quantity and types of the enemy obstacles that are spawned by one challenge are defined in real time, by the progression model.

4.3 Tags

The proposed progression model uses a set of tags to give context to the game content, more precisely to give context to the challenges and obstacles of the game. The context of a challenge and its obstacles describes and classifies the challenge, using a set of tags that are assigned, both by the game designer and by the progression model. The tags used by the progression model to describe each challenge, have the purpose to provide a way to assign, each given challenge to the player, a performance value

and a variety value, and are organized into four separated groups as shown in Figure 4.4.

TAGS			
OBSTACLE	PACE	TAPS	GAME DESIGNER
Purple	Slow	1-3	TargetHero
Blue	Moderate	4-6	Escape
Green	Fast	7-9	Gate
Red		10-12	Elbow
		13-15	DownLine
	

Figure 4.4: Tag categories.

These tags are defined and assigned in two different phases:

The first phase takes place during the creation of the challenge's library, when the game designer creates the challenges and manually assigns a set of tags, the Challenge Game Designer Tags, through the Challenge Editor Window on Unity (Figure 4.5), to each challenge after creating it. These tags can be used to describe: the context situation created by the obstacles (e.g. *AttackHero* - when the wave paths make the enemies go down the screen to attack the hero, *Escape* - when the enemies escape by both sides of the screen); the formation of the obstacles (e.g. *BlockAttack* - when the waves form a block and spawn enemies in a group, *ZigZagAttack* - when the wave paths of a challenge are zig zag lines); or anything the game designer wants to describe in the challenge.

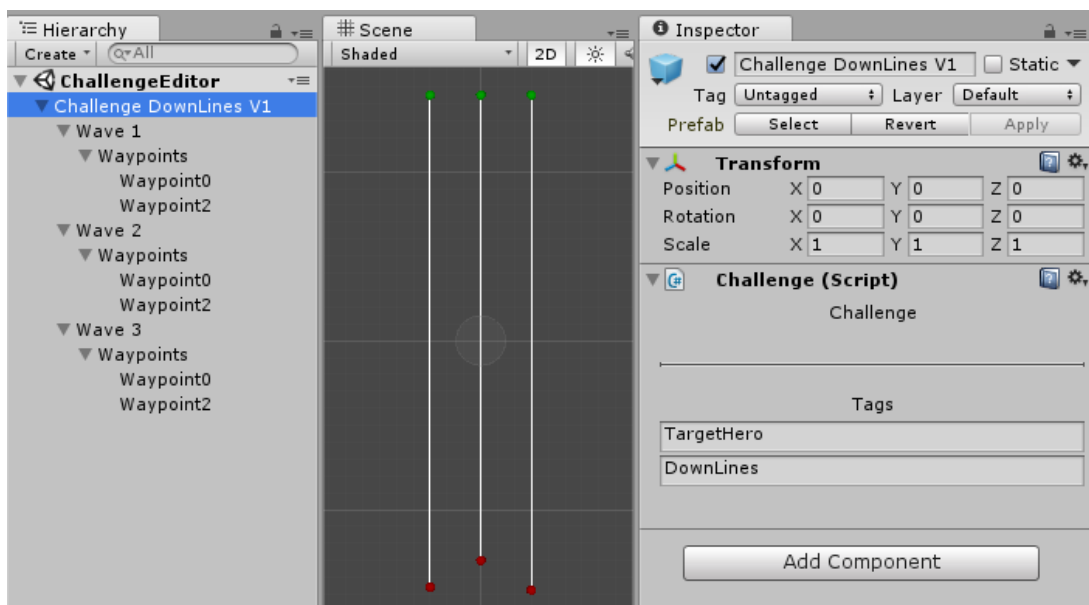


Figure 4.5: Challenge Editor Window: Challenge with 2 Game Designer Tags assigned.

The second phase takes place in real time, when the progression model autonomously creates a new challenge, with random content (quantity and type of obstacles and their pace), and assigns a new set of automatic tags that describe: the number of taps needed to overcome the challenge, the Challenge Taps Tags, ¹ (e.g. *Taps1-3*, *Taps4-6*, ...); the type and name of the obstacles that the challenge and its waves are going to spawn, the Obstacle Tags, (e.g. *RedObstacle*, *GreenObstacle*, *BlueObstacle* and *PurpleObstacle*); and the challenge pace, the Challenge Pace Tags, that represent the speed of all the challenge obstacles that move at the same pace (e.g. *SlowPace*, *ModeratePace*, *FastPace*).

The proximity between two challenges, from the challenges' context point of view, is calculated by the amount of common tags between the two challenges, relative to the total amount of tags of both challenges.

4.4 Game Cycle

This section presents the game cycle with the progression model integrated. To better understand the game cycle, the progression model architecture diagram is visible in the Figure 4.6. This diagram represents the game cycle of the game with the progression model which is composed by the following steps, that are repeated in a loop:

1. Generate a new challenge (game content) to present to the player, using:
 - the Player Performance Predictive System from the Player Performance Model;
 - the Content Variety Data from the Content Variety Model;
 - the Challenge Library.
2. Register the player response dealing with the obstacles that compose the generated challenge;
3. Analyze the player performance through the recorded player actions relative to the generated challenge;
4. Register the player performance data in the Player Performance Model;
5. Register the challenge variety data in the Content Variety Model;
6. Go to step 1.

4.5 Content Generation Model

The Content Generation Model uses both the *Player Performance Model* and the *Content Variety Model* to be able to generate engaging and challenging game content throughout a game session. Figure 4.7 shows the game content generation process, focusing in the connection between the Challenge Library and the Content Generation Model, and type of content that is used and generated by the Content Generation Model.

The Content Generation Model is used to generate a new challenge in the beginning of a new run and, from that moment, every time there is only 1 obstacle left from the last generated challenge. This model is composed by the following phases:

¹The first version of the Challenge Taps Tags was created with one tag for each number of taps (e.g. *Taps1* = 1 tap; *Taps2* = 2 taps; *Taps3* = 3 taps; ...) but, as explained in the section 5.1, it was later changed to groups of 3 taps for each tag (e.g. *Taps1-3* = [1,3] taps; *Taps4-6* = [4,6] taps; *Taps7-9* = [7,9] taps; ...).

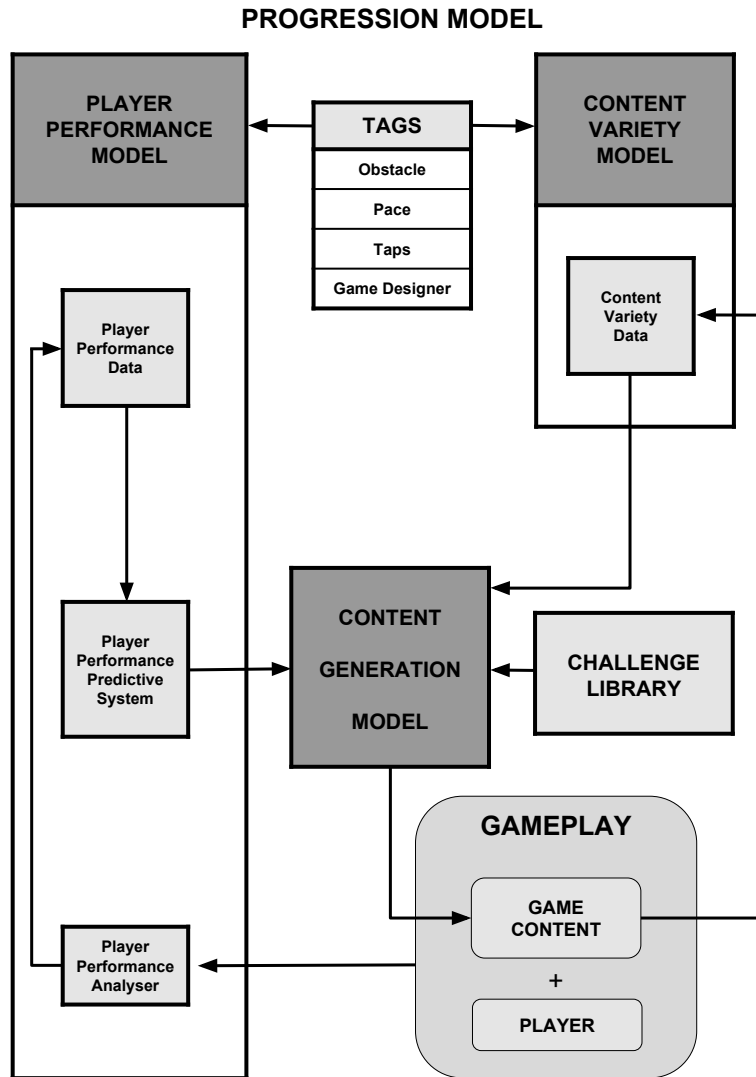


Figure 4.6: Game cycle with the Progression Model Architecture.

1. Generate a new population of 50 random challenges: This population of new challenges is, in fact, a population of metadata challenges that will contain only the data needed to generate a new challenge. This data includes the list of the various tags that will be assigned to the challenge, as well as the predicted player performance value and the predicted content variety value that are both used to calculate the overall challenge utility value. Instead of generating a population of Challenge prefabs, that would take a big amount of time and memory to be instantiated, especially given the fact that the testbed game runs on mobile devices, this model generates a population of metadata challenges. The class Challenge Data was created for this purpose, as it serves as a container to store the data of a possible challenge, allowing the creation of lots of possible challenges and thus, optimizing the efficiency and performance of the progression model. After the generation of the metadata challenges' population, each Challenge Data is connected to a, randomly selected, challenge prefab in the challenges' library ².
2. Populate the new population of metadata challenges: each Challenge Data is populated with random content for each challenge's wave: the wave obstacles' quantity; the wave obstacles' types;

²For more information, see the algorithm 2 in the Appendix B

and the wave obstacles' order. After setting the obstacles that will be spawned, the corresponding Obstacle Tags are assigned to each Challenge Data. The next step is to count the number of taps needed to overcome all the obstacles that will be spawned and the obstacles that will appear from the originally spawned obstacles of the Challenge and assign the correspondent Challenge Taps Tag to the Challenge Data. A random pace is set for the challenge and the corresponding Challenge Pace Tag is assigned to the Challenge Data ³. The challenge pace is then assigned to each wave of the challenge, to be later set to each obstacle when it is spawned. Finally, the Challenge Game Designer Tags are copied from the original challenge from the challenges' library to the Challenge Data, as well as to each wave of the challenge to be later set on each obstacle when spawned. ⁴.

3. Select the next best challenge candidate to be generated: Run through all the populated metadata challenges and calculate their utility values using a heuristic evaluation function that combines the predicted content variety value of a challenge with its predicted player performance value. In the end of this phase, the Content Generation Model defines which one is the most useful challenge to be generated next and presented to the player. To get the best challenge candidate, the heuristic evaluation compares: the next desired player performance value, from the performance curve defined by the game designer, with the predicted player performance value of each metadata challenge; and the next desired content variety value, from the variety curve also defined by the game designer, with the predicted content variety value of each metadata challenge ⁵;
4. Generate and activate a new challenge: after choosing the next best challenge candidate, from the metadata challenges' population, the Content Generation Model copies all the data from the chosen Challenge Data to the corresponding challenge prefab from the library and activates the new challenge.
5. Clean all the data from the last generated and populated metadata challenges' population and reuse again this population of metadata challenges on the step 1.

4.6 Player Performance Model

The following Player Performance Model was designed to evaluate the player's skill level, in order to allow and support the game to generate content that matches the player's skill level and to keep constantly challenging the player, as the player progress through the game. Since this Player Performance Model is adapted with data collected every time the player plays the game, it also allows the game to adapt itself and provide different levels of challenge for different players.

To achieve this, the game designer specifies a performance curve, that will shape the progression of the player's performance as it determines the difficulty flow of a game session. Figure 4.8 shows the used performance curve in this progression model as an example of a possible performance curve defined by the game designer.

4.6.1 Player Performance Analyzer

The performance of a challenge represents the player's dexterity to overcome the obstacles that compose the challenge. The value of the player performance relative to a generated and presented challenge is calculated using the combination of all its tags (Challenge Game Designer Tags, Challenge

³The obstacles' speed varies from 3 to 4,5: [3;3,5] = Slow Pace; [3,5;4] = Moderate Pace; [4;4,5] = Fast Pace

⁴For more information, see the algorithm 3 in the Appendix B

⁵For more information, see the algorithms 4, 8, 9 and 10 in the Appendix B

Pace Tags, Challenge Taps Tags, Obstacle Tags) performance values considering their categories' performance weights ⁶.

The Player Performance Model stores performance data relative to each tag of the progression model and relative to every challenge generated and presented to the player. The tags have a performance history of the last 10 performance values and the challenges' performances are recorded since the beginning of each game run as shown in Figure 4.9. When the game is installed by a new user, each tag starts with a bootstrap performance value, that was obtained during play testing sessions with both new users to the game and experienced users. With each arena run session this bootstrap performance values will be replaced by real player performance values obtained through the gameplay data.

While the Obstacle Tags' performances are analyzed individually to get more granular and accurate data, the Challenge Tags' (Pace, Taps, Game Designer) performances are calculated in a macro point of view, with all the challenge obstacles contributing to those tags' performance, using a performance metric called *Challenge Taps Score*, that is calculated from the counting of all the taps needed to overcome all the obstacles that will be spawned in a challenge and the taps successfully done on obstacles, calculated according to the following formula:

$$ChallengeTapsScore = \frac{TapsDone}{TapsNeeded} \quad (4.1)$$

The performance of an obstacle reflects if the player was able to overpass it (enemy smashed) or not (enemy escaped or attacked the hero). After one obstacle is overcome or not by the player, a performance value is assigned to that obstacle, and recorded in the performance history of the correspondent Obstacle Tag assigned to the challenge to which the obstacle belongs to, with one of the following values:

$$Performance(ObstacleOvercome) = 1 \quad (4.2)$$

$$Performance(ObstacleNotOvercome) = 0 \quad (4.3)$$

As explained in the Section 4.2, an obstacle is considered to be overcome when all the obstacles that are spawned from it, if there are any, are overcome.

The Figure 4.10 shows an example of a challenge with the obstacles overcome by the player marked with a green check mark and the obstacles not overcome marked with a red cross. The figure also shows the tags assigned to the challenge and the Challenge Tap Score value calculated from the player performance.

Every time a challenge is deactivated, its tags' performances are calculated and registered on the Player Performance Model, and used to assign an overall performance value to the challenge. The Challenge Taps Score value is calculated and assigned to the Pace Tag, Taps Tag and Game Designer Tags performance values to be afterwards registered in each tag performance history in the player performance data (Figure 4.9). The final performance value of each Obstacle Tag of the challenge is calculated with the average of all the recorded obstacle performances with the same tag. For this challenge and the corresponding player performance, shown in the figure, the Player Performance Analyzer would do the following calculations:

• *Game Designer Tags Performance :*

⁶For more information, see the algorithm 2 in the Appendix B

$$Performance(TargetHero) = 0,64 \quad (4.4)$$

$$Performance(DownLines) = 0,64 \quad (4.5)$$

$$Performance(GameDesignerTags) = \frac{TargetHero + DownLines}{2} = 0,64 \quad (4.6)$$

• *Pace Tag Performance :*

$$Performance(SlowPace) = 0,64 \quad (4.7)$$

• *Taps Tag Performance :*

$$Performance(Taps10 - 12) = 0,64 \quad (4.8)$$

• *Obstacle Tags Performance :*

$$Performance(RedObstacle1) = 1 \quad (4.9)$$

$$Performance(RedObstacle2) = 0 \quad (4.10)$$

$$Performance(BlueObstacle) = 0 \quad (4.11)$$

$$Performance(PurpleObstacle) = 1 \quad (4.12)$$

$$Performance(ObstacleTags) = \frac{RedObstacle + RedObstacle + BlueObstacles + PurpleObstacle}{4} = 0,5 \quad (4.13)$$

Finally, the performance assigned to the challenge is calculated using the following formula:

$$Performance(Challenge) = Performance(GameDesignerTags) * GameDesignerWeight \quad (4.14)$$

$$+ Performance(PaceTag) * PaceWeight \quad (4.15)$$

$$+ Performance(TapsTags) * TapsWeight \quad (4.16)$$

$$+ Performance(ObstaclesTags) * ObstacleWeight \quad (4.17)$$

4.6.2 Player Performance Predictive System

When the Player Performance Model needs to estimate what would be the player performance against a new challenge, it looks at the estimated performance of all the tags that are assigned to that challenge, according to each tag category weight, like when the player performance is analyzed and calculated as explained above ⁷. The estimated performance of each tag is calculated by the average value of the last 10 player performance values recorded.

4.7 Content Variety Model

The variety of a challenge represents its novelty and is defined by the variety of its tags. Challenges have variety values between 0 and 1. The variety of one specific tag is calculated by the frequency of its appearance in the game compared to the total count of already used tags. This means the more often a tag was used, the closer to 0 its variety value is and on the opposite side the less a tag was used, the closer its variety value is to 1. Hence, when a player starts a new game run, all the tags start with a variety value of 1, because they were never presented to the player. Figure 4.11 shows the data stored relative to the game content variety.

In the same way the game designer has the task to define a performance curve, he/she also has the task to define a variety curve, that will shape and guide the progression of the gameplay in terms of variation of the game content that is generated by the Content Generation Model. Figure 4.12 shows the used variety curve in this progression model as an example of a possible variety curve defined by the game designer.

The variety of a tag is calculated using the counting of the used tags in the already generated challenges, through the data stored in the Content Variety Model as shown if the following formula:

$$Variety(Tag) = \frac{TagUsageCount}{TotalTagsUsageCount} \quad (4.18)$$

Using each tag's individual variety value, the Content Variety Model is able to assign a variety value to a challenge using the following formula:

$$Variety(Challenge) = Variety(GameDesignerTags) * GameDesignerWeight \quad (4.19)$$

$$+ Variety(PaceTag) * PaceWeight \quad (4.20)$$

$$+ Variety(TapsTags) * TapsWeight \quad (4.21)$$

$$+ Variety(ObstaclesTags) * ObstacleWeight \quad (4.22)$$

In this progression all the tags' categories have the same variety weight but there is the possibility, for example, in order to strengthen the game designer role relevance in the variation of the game content, to increase the weight of the Game Designer Tags' variety.

⁷For more information, see the algorithm 2 in the Appendix B

GAME CONTENT GENERATION

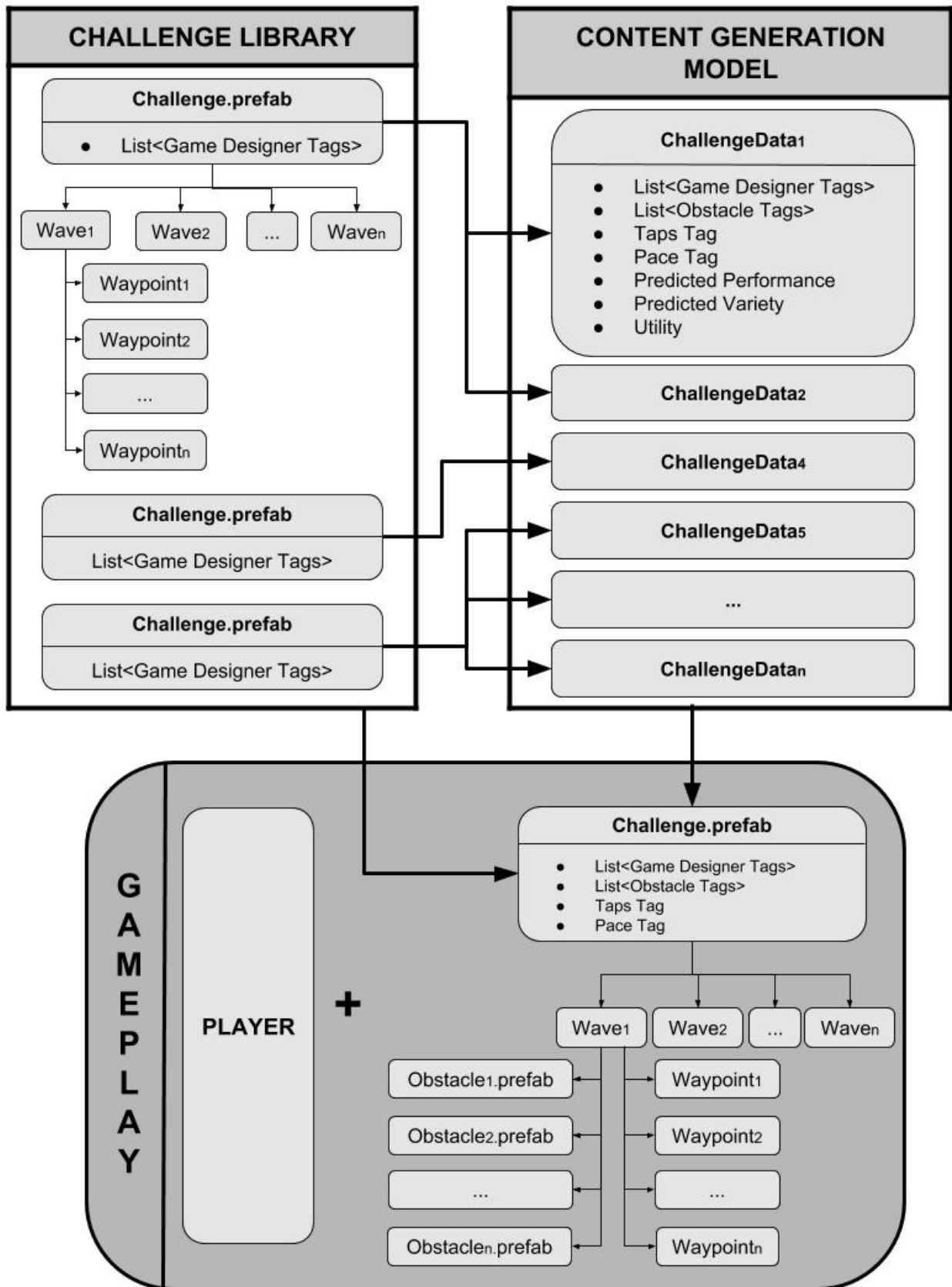
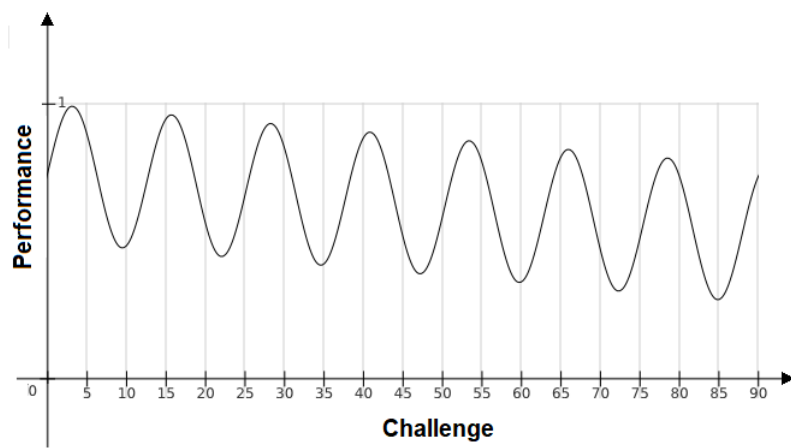


Figure 4.7: Game Content Generation (Challenge Library + Content Generation Model).



$$\text{Performance}(\text{Challenge}) = (\sin(\text{Challenge} / 2) - (0.01 * \text{Challenge}) + 3) / 4$$

Figure 4.8: Player Performance Curve.

Player Performance Model													
Tags		Recorded Performances											
		Bootstrap (Start Value)	Last 10 Obstacle Performances										
			1	2	3	4	5	6	7	8	9	10	
OBSTACLE	Purple	P ₀	P ₁	P ₂	P ₃	P ₄	P ₅	P ₆	P ₇	P ₈	P ₉	P ₁₀	
	Blue	P ₀	P ₁	P ₂	P ₃	P ₄	P ₅	P ₆	P ₇	P ₈	P ₉	P ₁₀	
	Green	P ₀	P ₁	P ₂	P ₃	P ₄	P ₅	P ₆	P ₇	P ₈	P ₉	P ₁₀	
	Red	P ₀	P ₁	P ₂	P ₃	P ₄	P ₅	P ₆	P ₇	P ₈	P ₉	P ₁₀	
PACE	Slow	P ₀	P ₁	P ₂	P ₃	P ₄	P ₅	P ₆	P ₇	P ₈	P ₉	P ₁₀	
	Moderate	P ₀	P ₁	P ₂	P ₃	P ₄	P ₅	P ₆	P ₇	P ₈	P ₉	P ₁₀	
	Fast	P ₀	P ₁	P ₂	P ₃	P ₄	P ₅	P ₆	P ₇	P ₈	P ₉	P ₁₀	
TAPS	1-3	P ₀	P ₁	P ₂	P ₃	P ₄	P ₅	P ₆	P ₇	P ₈	P ₉	P ₁₀	
	4-6	P ₀	P ₁	P ₂	P ₃	P ₄	P ₅	P ₆	P ₇	P ₈	P ₉	P ₁₀	
	7-9	P ₀	P ₁	P ₂	P ₃	P ₄	P ₅	P ₆	P ₇	P ₈	P ₉	P ₁₀	
	...	P ₀	P ₁	P ₂	P ₃	P ₄	P ₅	P ₆	P ₇	P ₈	P ₉	P ₁₀	
GAME DESIGNER	TargetHero	P ₀	P ₁	P ₂	P ₃	P ₄	P ₅	P ₆	P ₇	P ₈	P ₉	P ₁₀	
	Escape	P ₀	P ₁	P ₂	P ₃	P ₄	P ₅	P ₆	P ₇	P ₈	P ₉	P ₁₀	
	Gate	P ₀	P ₁	P ₂	P ₃	P ₄	P ₅	P ₆	P ₇	P ₈	P ₉	P ₁₀	
	Elbow	P ₀	P ₁	P ₂	P ₃	P ₄	P ₅	P ₆	P ₇	P ₈	P ₉	P ₁₀	
	DownLine	P ₀	P ₁	P ₂	P ₃	P ₄	P ₅	P ₆	P ₇	P ₈	P ₉	P ₁₀	
	...	P ₀	P ₁	P ₂	P ₃	P ₄	P ₅	P ₆	P ₇	P ₈	P ₉	P ₁₀	
CURRENT RUN CHALLENGES			C ₁	C ₂	C ₃	C ₄	C ₅	C ₆	C ₇	C ₈	...	C _N	
RECORDED PERFORMANCES			P ₁	P ₂	P ₃	P ₄	P ₅	P ₆	P ₇	P ₈	...	P _N	

Figure 4.9: Player Performance Model Data.

- Challenge Game Designer Tags = { " Target Hero " , " Down Lines " }
- Challenge Pace Tag = " Slow "
- Challenge Taps Tag = " 10-12 "
- Obstacles Tags = { " Red ", " Blue ", " Purple " }

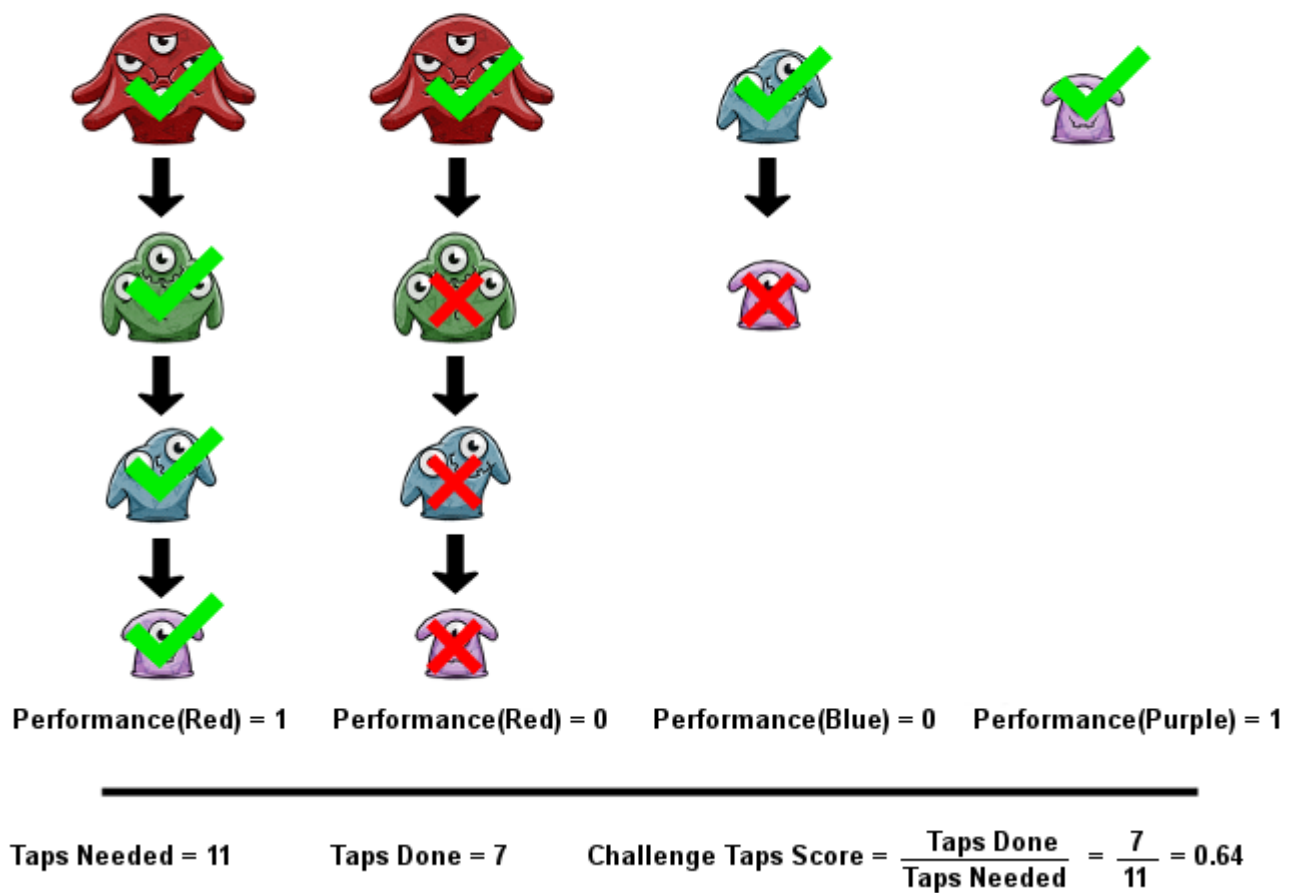
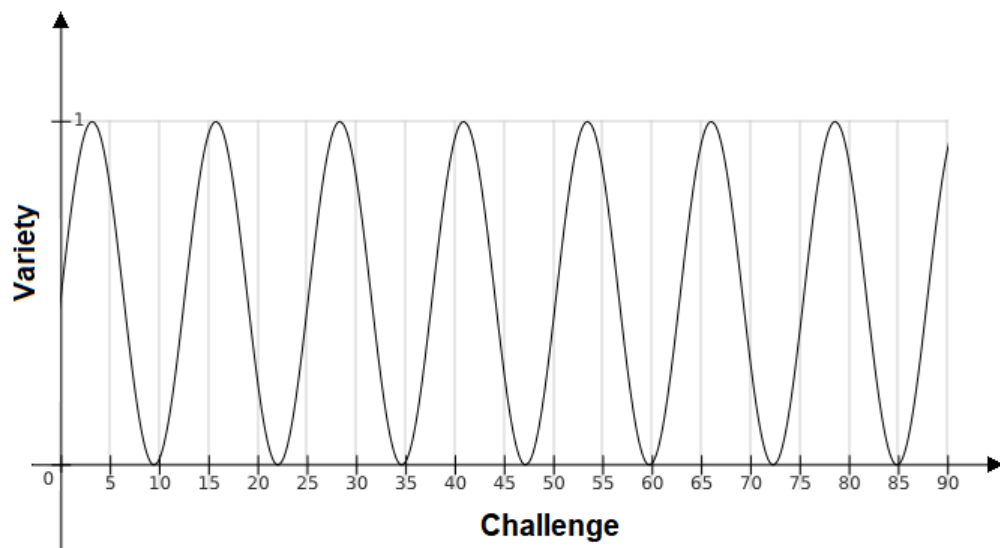


Figure 4.10: Example of the player performance dealing with a challenge generated by the Content Generation Model.

Content Variety Model											
TAGS		Current Run Challenges Tag Use Count									
OBSTACLE	Purple	X									
	Blue	X									
	Green	X									
	Red	X									
PACE	Slow	X									
	Moderate	X									
	Fast	X									
TAPS	1-3	X									
	4-6	X									
	7-9	X									
	...	X									
GAME DESIGNER	TargetHero	X									
	Escape	X									
	Gate	X									
	Elbow	X									
	DownLine	X									
	...	X									
CURRENT RUN CHALLENGES		C ₁	C ₂	C ₃	C ₄	C ₅	C ₆	C ₇	C ₈	...	C _N
RECORDED VARIETIES		V ₁	V ₂	V ₃	V ₄	V ₅	V ₆	V ₇	V ₈	...	V _N

Figure 4.11: Content Variety Model Data.



$$\text{Variety}(\text{Challenge}) = (\sin(\text{Challenge} / 2) + 1)/2$$

Figure 4.12: Content Variety Curve.

Chapter 5

Evaluation

In this chapter, we report the testing made on the progression model presented in the previous chapter, presenting the procedures, results, changes and insights for each moment of the evaluation. In order to validate if the generated player-adapted content improves the player experience we evaluated the quality and potential of the implemented progression model with playtesters, in several phases during the development of the proposed solution. The first part of the evaluation with real users, novice and expert players of the testbed game, was made to adjust the parameters of the progression model and to verify its potential. Next, we will describe in detail the qualitative and quantitative evaluations made to validate our approach. To conclude this chapter, we will present the insights gained from the different evaluations that were made.

5.1 Preliminary Evaluations

Preliminary evaluations were made several times through the development of this progression model. These evaluations were made with real users, both novice and expert players of the testbed game. These evaluations had the following objectives:

- Test the performance of the progression model;
- Test the reaction time of the progression model to the evolution of the player skill level;
- Test players' reactions obtained from the gameplay experience with the progression model;
- Get the bootstrap values of the player performance to be used in the final version of the Player Performance Model;
- Refine the performance curve that defines the wanted player skill evolution curve;
- Refine the variety curve that represent the wanted content variety progression;
- Fine tune every parameter of the progression model components.

5.1.1 Procedure

These evaluations were made without any formal playtest guideline, it was based on informal playtesting the new arena followed by an unstructured interview to the participants to collect all type of feedback without telling the participants what they were testing in order to not bias the gameplay experience and

the participants feedback. In the end of each playtest session the values recorded by the Player Performance Model, accessible through the SRDebugger tool on the mobile, were collected and gathered on the progression model being developed.

5.1.2 Results and Changes

One of the goals of these preliminary evaluations was to get the bootstrap values of player performance of the several tags that are stored in the Player Performance Model. This process was done through several iterations, each of them with changes on the values collected from the users' playtests, in order to tune those bootstrap values to the point that they represent a good starting point to new users. After having tuned the bootstrap player performance values that compose the Player Performance Model of the progression model, we were ready to do qualitative and quantitative evaluations of the implemented solution.

As these experiments were repeated several times through the development of the progression model, the number of recorded player performance entries for each tag changed several times. In the first version used on these preliminary evaluations, the Player Performance Model had saved data related to all challenges and obstacles presented to the player since the game was installed. This first experiments revealed that this approach was not appropriate because on the one hand, the player data become too large and the game started to slowdown a little bit after some game sessions and on the other hand, as all the times a new challenge is generated by the Content Generation Model, the Player Performance Model takes into account everything the player did since the beginning, which means that we were not giving more relevance to the most recent actions of the player. To solve both of these problems the recorded player actions window was updated to keep track of the last 20 obstacles presented to the player. As a result of later experiments and in order to decrease the reaction time of the progression model to the player skill evolution, this value was later changed to the last 10 player performance records for each tag. With a smaller performance history window, with the most recent player performance data, the progression model is able to adapt more easily to the player skill progression not being tied up to what happened a long time ago and that is no longer relevant to the current player skill level.

One of the changes that were made after these experiments, was to change the Challenge Taps Tags to have each tag representing a group of number of taps (e.g. Taps1-3 = [1,3] taps; Taps4-6 = [4,6] taps, Taps7-9 = [7,9] taps; ...) instead of each tag matching a specific number of tags (e.g. Taps1 = 1 tap; Taps2 = 2 taps; Taps3 = 3 taps; ...). With this change we hope that in most arena runs, the player performance data is updated, with recent data, for the largest possible number of the existing Challenge Taps Tags in the Player Performance Model. If the tags only represented one value of taps, there was the possibility that several arena runs would pass, until one or more tags' performance data history would be updated, due to no challenges being generated with some values of taps needed, what would lead the progression model to keep old player performance data that, once again, could no longer be relevant to the current player skill level.

After having the bootstrap values of the progression model and all parameters adjusted, we were ready to qualitatively and quantitatively test the implemented solution.

5.2 Final Evaluation

The qualitative and quantitative evaluations made to validate the developed progression model are described bellow. The playtest procedure guideline used by the observer of the playtest sessions can be consulted in the Appendix C.

5.2.1 Procedure

For this study, we looked to find people from all types:

- gamers and non-gamers;
- with and without experience in games of the same genre;
- with and without previous experience in the used testbed game;
- females and males;
- young and old.

The objective of this evaluation was to test if the progression model is able to adapt itself to any type of player (according to the previous categories) to provide enjoyable and challenging gameplay experiences every time a player starts a new game session.

To validate these hypothesis we compared the developed progression model in this study with the previous progression model already being used in the testbed game. Hence, we tested two different versions of a progression model on the arena endless level of Smash Time:

- Smash Time O - Old progression model that is on the game on the stores at this moment, that takes into account the difficulty of the challenges combined with the duration of the game session;
- Smash Time N - New progression model created during this dissertation that analyses the performance of the player facing the challenges combined with the control of the variety of the generated content.

The playtest started with a brief presentation of the testbed game, without mentioning what was going to be tested, after it was said to the participants that they would have to answer a questionnaire in the end, again without mentioning what was going to be asked, to not influence the gameplay experience and the answers collected in the questionnaires.

After the presentation, the participants played the first three campaign levels that serve as game tutorials to present the game and its content and to teach the players how the gameplay mechanics work, during this time the participants were able to ask any questions if they were not understanding something.

Following this initial contact with the game, through the tutorial levels, we made a short demonstration of the arena level, focusing on the time component, on how it works and how could the players extend the time to achieve better scores.

After the arena game mode demonstration, the participants had the opportunity to ask any remaining questions before starting the real playtest. There was no minimum nor maximum number of arena runs for the playtest, so that the participants could play as much as they wanted and during the time they wanted.

During this phase of the playtest there was no intervention by the observer, except when requested by the participants.

When the participants decided to stop playing, the Game Experience Questionnaire (Appendix E) was given to them, followed by an unstructured interview in order to gather some additional feedback from the participants.

After finishing the participants part in the playtest, the observer then had to collect the game data that was automatically tracked and registered by the game, relative to: the total arena gameplay duration; the number of arena starts; the number of times the Hero was attacked by an enemy; the number of

times the session ended with time out; and the number of times the user quit an arena run (Appendix D) and fill the Data Collected Questionnaire (Appendix F).

As mentioned before, we used two different questionnaires, for each version (old and new) of the progression model, to collect data from the playtests with the participants, relative to their experience and the game data about what happened in the game during the playtests:

- Game Experience Questionnaire (Appendix E) - Answered by the users at the end of the playtest session. The game experience questionnaire is the core module part of a bigger questionnaire called "The Game Experience Questionnaire". [42];
- Game Data Collected Questionnaire (Appendix F) - Answered by the person responsible for conducting the playtest session, without the knowledge of the users, with the data automatically tracked and collected by the game and accessible through the SRDebugger Tool [43], that was checked inside the game, on-device, in the end of each playtest session.

This resulted in four different questionnaires:

- Game Experience Questionnaire - Old Arena;
- Game Experience Questionnaire - New Arena;
- Game Data Collected Questionnaire - Old Arena
- Game Data Collected Questionnaire - New Arena

The Game Experience Questionnaire will try to answer if the new progression model changed the gameplay experience and the Data Collected Questionnaire will try to show if the players spent more time playing the game or not, and if they play more times than with the previous progression model.

5.2.2 Demographic Results

In this sub section we will report the demographic results of the gameplay tests. The user tests group was composed by 32 people, 16 for each arena version with the respective progression models.

Next, there is a descriptive analysis of the relevant demographics data: Figure 5.1 shows the relation the participants have with videogames and the frequency they play them; Figure 5.2 presents the participants familiarity with the smasher genre on mobile videogames; and Figure 5.3 indicates if the participants had played the used testbed mobile videogame, Smash Time, before the playtest.

5.2.3 Quantitative Evaluation Results

Table 5.1 presents a set of game data tracked and collected during the playtests with users, it shows the average values collected, since the users start playing the arena game mode during the playtests (after the playtest introduction, playing the tutorial levels and the arena mode demonstration), relative to: the overall duration of the arena playtest, in seconds; the count of starts of a new arena run; the number of times the users got to an arena game over due to 1) the hero being attacked by enemies and 2) the time running out; and the number of times the users had quit a run by themselves, both for the old arena version and the new arena version developed during this dissertation with the progression model proposed in this study.

In this table, it is possible to confirm that the overall gameplay duration, as well as the number of starts of a new run has increased from the old to the new arena versions.

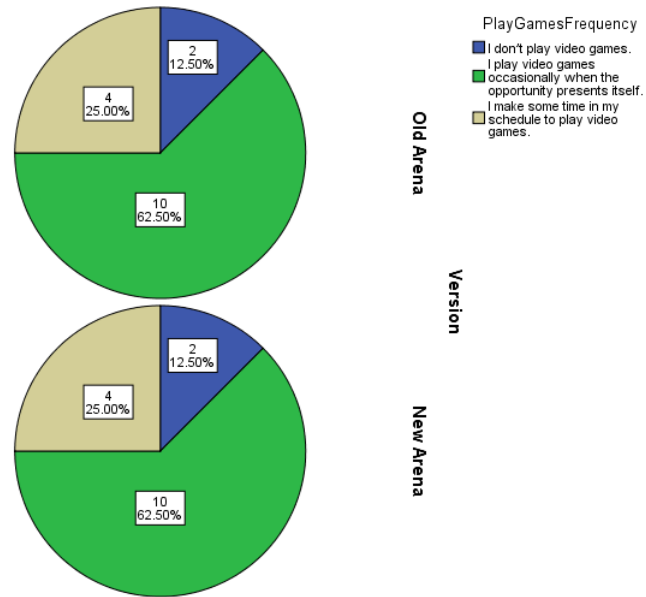


Figure 5.1: Play Games Frequency: Old Arena VS New Arena.

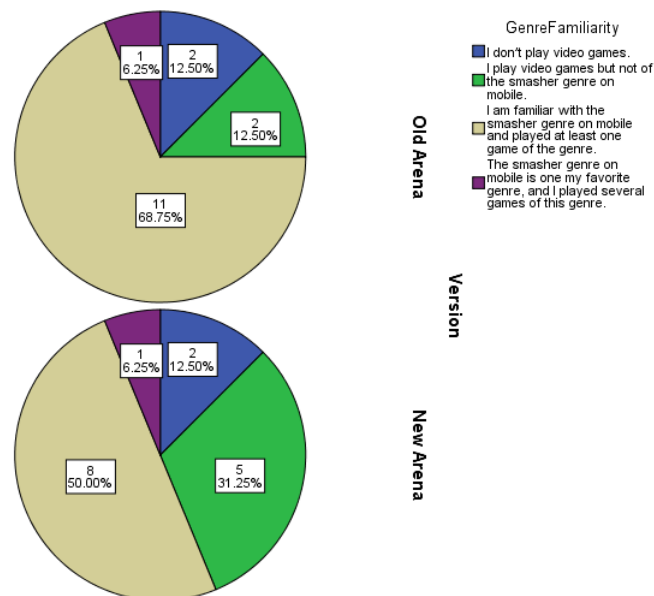


Figure 5.2: Game Genre Familiarity: Old Arena VS New Arena.

Arena Version	Playtest Duration (seconds)	Arena Start Count	Hero Attacked Count	Time Out Count	User Quit Count
Old	446	2.2	0.6	1.6	0
New	906	5.8	1.0	4.8	0

Table 5.1: Game data collected (average values) during the playtests: Old Arena VS New Arena.

None of the variables passed the Shapiro-Wilk normality test, hence, we used non-parametric statistical test. Regarding the Mann-Whitney U statistical test on the above mentioned variables, there was statistical significance on the following variables: *PlayTestDuration*; *ArenaStartCount*, and *TimeOutCount*, in all cases with $p < 0.001$, which means there is a clear difference in both versions, whereas

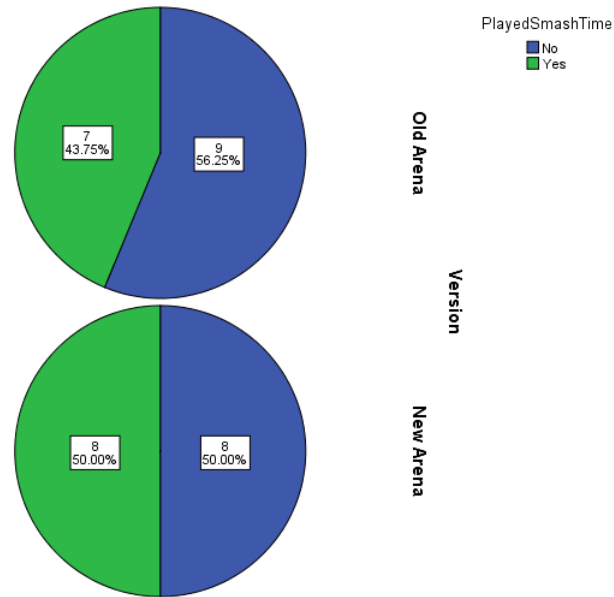


Figure 5.3: Played Smash Time: Old Arena VS New Arena.

the new arena version with the developed progression model on this dissertation got better results than the old arena version. More data results from this evaluation can be consulted in the Appendix H.

5.2.4 Qualitative Evaluation Results

In this section we will present the results gathered from the Game Experience Questionnaire given to the participants.

Firstly, we will do a macro evaluation composed by 7 components, each representing a group of variables from the original Game Experience Questionnaire. These variables and respective analysis were based on the scoring guidelines for the Game Experience Questionnaire Core Module [42], and were grouped as shown in the Appendix G. From these 7 aggregating variables, the variables *SensoryAndImaginativeImmersion* and *Flow* follow normal distributions on the Shapiro-Wilk normality test. Also, on the Mann-Whitney U test, none of these variables have shown statistical significance.

Except in the case of the variables *FeltCouldExplore* (Item: I felt that I could explore things.) and *PutLotEffort* (Item: I had to put a lot of effort into it), none of the other variables passed the Shapiro-Wilk normality test. Regarding the non-parametric Mann-Whitney U tests, the variables *WasTiresome* (Item: I found it tiresome.) and *WasImpressive* (Item: I found it impressive.) have statistical significance with $p < 0.05$. These results suggest the participants felt the new arena version was more impressive but also more tiring in comparison with the old arena version. More data results from this evaluation can be consulted in the Appendix I.

After doing the playtest, and answering the Game Experience Questionnaire, an unstructured interview was conducted in order to gather some additional feedback by the participants.

Some of the most relevant observations made by the participants were, in a positive point of view, that the game seemed to become more difficult as time was passing by and that they had the feeling that there were some moments where the game got more demanding taking their skills to the limit, sometimes a little too much becoming a bit exhaustive, opposing with more relaxed moments where they could rest for a while, before engaging again in a more challenging phase providing an interesting experience by making the players go through various states of mind and experiencing different moods while playing. On a not so positive point of view some of them claimed that the game got too hard very

fast and they couldn't really follow that increase in the difficulty of the content that was being given to them, which means the progression curve and the variety curve should be tweaked.

After answering the unstructured interview, it was explained what the implemented progression model is used for, what were the users testing and what was the propose of the playtests. The reactions were of surprise with the confirmation of the feeling that the game was progressively becoming more difficulty (challenging) over time.

5.2.5 Discussion

The comments from the participants suggest that the players' acceptance was quite good, as they have had the perception of progression, as the game kept providing challenging content as their skill level was increasing, getting the feeling that the game was reacting to what they were learning and doing, and trying to correspond giving them more challenging content. These comments are also complemented by the data collected during the playtest sessions for the qualitative and quantitative evaluation. In an overall picture based on these results, we can say that they are very encouraging to keep developing this progression model believing that it has what it needs to achieve its main objectives, that we first set out for the progression model, like providing challenging and more engaging gameplay experiences, with the potential to increase the duration of each game session in endless single player video games.

After the informal interview, it was explained to the participants that they were testing a progression model that generates game content in real time while they were playing based on their skill level, as the game was constantly evaluating their performance overcoming the challenges that were being given to them.

The reactions varied from surprise, when knowing that the game was capable of understanding how they were playing, to the confirmation of the feeling that the game was progressively becoming more difficult and challenging over time.

Some of the participants said that they had played several endless games that invariably got extremely difficult and frustrating at some point after some minutes of gameplay leading to the feeling that the game was not fair because it increased its speed too much, or increased the amount and difficulty of obstacles or enemies to a point where it was not possible to overcome and keep playing, almost forcing the players to get a game over and start all over again from the beginning in a new game run.

Concluding these observations, some participants suggested that those games should have used a progression model similar to the one that was developed and integrated in the testbed game and maybe with that they would have played those games for some more time before putting them aside.

On the one hand, some participants, mostly the ones that had already experience playing the testbed game, were getting better and better from run to run while, on the other hand some of participants with no experience in the game, had the best performance (and score) in the first runs, that could be a sign that the progression model is reacting to the player skill and evolving too rapidly for the players to follow that progression. One possible change that can be made on the Player Performance Model is to smooth the game difficulty evolution, in order to not demand too much from the novice players, that is to increase the number of last performances the progression model keeps tracking for each tag, in order to take longer to react and evolve to the player skill. Therefore, the progression model gives the players more time to learn how to play but also to get really good at the current challenge level of the game before the game rises up its challenge level again. This could be done, for instance, by changing the recorded player performance data for each tag on the Player Performance Model from the current 10 last performance values being used, to 20 performance values for each tag. This change means that the progression model will need more challenges and time to react to the player skill evolution, providing a smoother player skill progression.

Another possible change to try to mitigate this steep and fast evolution on the challenge level is to tune the performance curve formula to be smoother over time. Although the performance curve had been tweaked several times during the preliminary evaluations, due to the playtests with real players, as reported above in Section 5.1 of this chapter, it looks like it still needs more fine tuning to get closer to the perfect balance.

One thing that we believe that does not need more tweaking are the bootstrap performance values of each tag, that were also tweaked several times during the preliminary evaluation, because according to the results it looks like that the game provides a good starting gameplay experience to both novice and expert players of the testbed game even using the same bootstrap performance values to all players.

In the first testing phase, we should have used both expert and novice players of the testbed game with the same proportions instead of focusing more in the expert players feedback, because as the results from the playtest sessions suggests, the bootstrap values are good for every player, novice or expert, but the performance curve was more adapted for expert players or players with experience in the game but on the other hand, the progression model challenge level, based on the performance curve, evolved too fast for novice players.

5.3 Summary

According to the evaluations done to the developed progression model and the results that we got from those evaluations, we believe that the approach to model the player skill progression was a good approach and has a great potential to be used in a future update to the testbed game, as well as to be applied to other endless single player video games.

Players did play the game more times, effectively, and for longer periods of time, which was the two main objectives of our work. The feedback received through the questionnaires, answered after the playtest sessions, also suggest that players found this approach more impressive and challenging than the previous progression model version that just increased the difficult of the game, through preset rules, based on the game designer intuition combined with the game session duration.

Therefore, and to complete the analysis of the evaluation of the potential and success of our approach, we believe that the approach to develop a progression model that takes into account the real player skill and the content variety, was successfully implemented into a robust and dynamic skill-based progression model.

Chapter 6

Conclusion

In this dissertation, we presented a skill-based progression model for endless single player videogames based on two main concepts: player performance and content variety. While implementing this progression model, we focused on addressing the main research objective of this dissertation, more specifically: creating more challenging and engaging gameplay experiences, increasing the duration of the game sessions on endless single player videogames. To achieve this goal, we have used *Smash Time*, a mobile smasher videogame, as a testbed game for our experiments.

Given that the main reason videogames are created is to give enjoyable and challenging gameplay experiences to the players, we focused in the player role while playing a game since the beginning of this study. To that extent, we decided to create a player model to understand better the player, more specifically, a player model that would assess the player skill and its evolution as he/she progresses in the game. The other focus of attention was the variety of the generated game content, that should present new and varied content at an appropriate pace, letting the player recognize some patterns while at the same time, be constantly presenting new and mixed content. Starting from these premises, we used the concepts of *Challenges*, *Obstacles* and *Tags* to develop the three main components of the developed progression model: the *Player Performance Model*; the *Content Variety Model*; and the *Content Generation Model*.

An experiment, with both novice and experienced players to the testbed game, was conducted to find the best bootstrap player performance values to start the progression model to adapt the game content. After finding a good starting point to the Player Performance Model, both quantitative and qualitative experiments were done through playtest sessions with real players. In the end of the playtest session, each player answered a game experience questionnaire. The data collected from the questionnaires was later combined with game data automatically tracked and collected by the game while the playtesters were trying the testbed game with the developed progression model.

The main objectives of this evaluation were to test if the progression model is able to adapt itself to any type of player providing enjoyable and challenging gameplay experiences every time a player starts a new game session and to confirm if the progression model is able to increase the number of times the game is played as well as to increase the duration of each game session.

As the results suggest, we were able to successfully increase, both the duration of each game session and the number of starts of a new game run. We believe to have managed to take the players faster to their gameplay flow channel, in order to make the game interesting, fun and challenging for longer periods of time, by preventing the players to get through a phase of an easy difficulty level in the beginning of every game run, that eventually will make the game boring in the beginning. Besides that, by allowing the players to stay in their gameplay flow channel for longer periods, we also believe to have prevented the game to become unsustainably difficult after a while and, thus, becoming, at some point,

frustrating to the players.

By not using preset game flow rules, that lead to linear gameplay experiences, like many games do and like the previous progression model used in the testbed game did, we believe to have managed to create more dynamic gameplay experiences that, ultimately, also increase the replayability and life time of the game.

We believe this is a good progression model that allows to reduce the game designer amount of work, such as to define a dependency flow chart to guide the PCG, as it happens in some games, or to define the probabilities of specific content with a difficulty level associated in relation to the duration of the game session as it was happening in the previous progression model that is currently being used in the official version of the testbed game and in many other endless games. This progression model supports the game designer's work, carrying the task to evaluate the player skill level and decide the game challenge level at any given moment.

Finally, we believe to have developed a promising skill-based progression model that procedurally generates game content based on the player skill and content variety. An appropriate estimator of player skill and a robust progression model were designed, implemented and tested in this dissertation. Hence, we argue that the presented approach to model progression in an endless single player video game, has a great potential to be applied successfully to other endless games and even to videogames in general. This progression model is data persistent and can be used in the normal levels (not endless) of the campaign mode of a videogame since the player performance model may be carried forward as the player progresses through the game. In order to generalize the progression model to other videogames, the *Challenges*, *Obstacles* and *Tags* concepts must be adapted to the specific context of the videogame content. Lastly and, once the player skill and performance evaluation were quantitatively modelled in the *Player Performance Model*, one just needs to adapt the performance measurement method to what is most suitable to the game in which is being implemented. Additionally, the progression model proposed showed promising results that also demonstrate the potential for extensibility to other games.

6.1 Future Work

To conclude this dissertation study, we propose to extend the functionality of 2 components of the developed progression model, the *Player Performance Model* and the *Content Generation Model* as explored bellow:

1. Extend Player Performance Model:

- Refine the information used to calculate the player performance with the inclusion of the player's reaction time to an obstacle. If the player is not able to overcome an obstacle the associated performance value will still be 0, but when the player is able to overcome an obstacle its performance value can decrease from 1, as soon as the obstacle appears, to 0 as the reaction time tends to infinity.
- Create a new depth level in the Player Performance Model by combining the existing player skill data with one of the existing models of the player type, like the Myers-Briggs personality type (conqueror, manager, wanderer and participant).

2. Extend Content Generation Model:

- Use previously collected data about the player performance and content variety to influence the generation of the new challenges. A part of the new challenges data populations or the whole population that is generated every time a new challenge must be created and presented

to the player can use some of the information already stored in the progression model, in order to leverage the utility value of the new challenges.

- Extend the Challenge component to support the functionality of composed challenges. The progression model could be able to create completely new challenges, in real time, from the combination of previously created challenges by the game designer that compose the challenges' library. Additionally, these new composed challenges created in real time, could be stored in the challenges' library to be later used in future game sessions.

Bibliography

- [1] King, “Candy Crush Saga.” [Digital game] King, 2012.
- [2] Supercell, “Clash of Clans.” [Digital game] Supercell, 2012.
- [3] Supercell, “Clash Royale.” [Digital game] Supercell, 2016.
- [4] Blizzard Entertainment, “World of Warcraft.” [Digital game] Blizzard Entertainment, 2004.
- [5] Riot Games, “League of Legends.” [Digital game] Riot Games, 2009.
- [6] Mojang, “Minecraft.” [Digital game] Mojang, Microsoft Studios, Sony Computer Entertainment, 2011.
- [7] Entertainment Software Association, “Essential Facts About the Computer and Video Game Industry.” http://www.theesa.com/wp-content/uploads/2017/09/EF2017_Design_FinalDigital.pdf, 2017.
- [8] M. Csikszentmihalyi, *Flow: The Psychology of Optimal Experience*. Harper & Row, 1990.
- [9] Bica Studios, “Smash Time.” [Digital game] Bica Studios, 2015.
- [10] J. Togelius, E. Kastbjerg, D. Schedl, and G. N. Yannakakis, “What is procedural content generation?: Mario on the borderline,” in *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*, p. 3, ACM, 2011.
- [11] J. Togelius, A. J. Champanand, P. L. Lanzi, M. Mateas, A. Paiva, M. Preuss, and K. O. Stanley, “Procedural content generation: goals, challenges and actionable steps,” in *Dagstuhl Follow-Ups*, vol. 6: Artificial and Computational Intelligence in Games, pp. 61–75, 2013.
- [12] M. Toy, G. Wichman, K. Arnold, and J. Lane, “Rogue.” [Digital game], 1980.
- [13] Blizzard North, “Diablo.” [Digital game] Blizzard Entertainment, Ubisoft Entertainment, Electronic Arts, 1996.
- [14] Blizzard North, “Diablo II.” [Digital game] Blizzard Entertainment, Sierra Entertainment, 2000.
- [15] Blizzard North, “Diablo III.” [Digital game] Blizzard Entertainment, 2012.
- [16] Gearbox Software, “Borderlands.” [Digital game] 2K Games, 2009.
- [17] R. Fruhstick, “‘Borderlands’ has 3,166,880 different weapons.” <https://web.archive.org/web/20090731095451/http://multiplayerblog.mtv.com/2009/07/28/borderlands-has-3166880-different-weapons/>, 2009.
- [18] Bethesda Game Studios, “Elder Scrolls IV: Oblivion.” [Digital game] Bethesda Softworks, 2K Games, 2006.

- [19] Bethesda Game Studios, "Elder Scrolls V: Skyrim." [Digital game] Bethesda Softworks, 2011.
- [20] H. Hiraoka, D. Watanabe, and K. Fujita, "dreeps : Alarm Playing Game." [Digital game], 2015.
- [21] Farbrausch, ".kkrieger." [Digital game], 2004.
- [22] Interactive Data Visualization, Inc, "SpeedTree." [Software] <https://store.speedtree.com>, 2002.
- [23] E. Key and D. Kanaga, "Proteus." [Digital game], 2013.
- [24] Maxis, "Spore." [Digital game] Electronic Arts, 2008.
- [25] Hello Games, "No Man's Sky." [Digital game] Hello Games, 2016.
- [26] C. Martinho, P. Santos, and R. Prada, *Design e Desenvolvimento de Jogos*, ch. 4. Lisboa, Portugal: FCA, 2014.
- [27] J. Chen, "Flow in games (and everything else)," in *Communications of the ACM*, vol. 50, No. 4, pp. 31–34, ACM, 2007.
- [28] G. N. Yannakakis and J. Togelius, "Experience-driven procedural content generation," in *IEEE Transactions on Affective Computing*, vol. 2, Issue. 3, pp. 147–161, IEEE, 2011.
- [29] R. Dias and C. Martinho, "Adapting content presentation and control to player personality in videogames," in *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology*, ACE, 2011.
- [30] N. Shaker, G. N. Yannakakis, and J. Togelius, "Towards player-driven procedural content generation," in *Proceedings of the 9th conference on Computing Frontiers*, pp. 237–240, ACM, 2012.
- [31] Nintendo, "Super mario bros." [Digital game] Nintendo, 1985.
- [32] M. Persson, "Infinite mario bros." [Digital game] Markus Persson.
- [33] D. Cook, "The chemistry of game design." https://www.gamasutra.com/view/feature/129948/the_chemistry_of_game_design.php, July 2007.
- [34] Infinity Ward, "Call of Duty: Modern Warfare 2." [Digital game] Activision, 2009.
- [35] Id Software, "Quake Live." [Digital game] Bethesda Softworks, 2010.
- [36] Id Software, "Quake III Arena." [Digital game] Activision, 1999.
- [37] Kiloo and SYBO Games, "Subway Surfers." [Digital game] Kiloo and Microsoft, 2012.
- [38] M. Jennings-Teats, G. Smith, and N. Wardrip-Fruin, "Polymorph: A model for dynamic level generation," in *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE, 2010.
- [39] P. Pereira, "Modelling progression in video games." MSc Thesis, Instituto Superior Técnico, University of Lisbon, 2016.
- [40] A. Zook, S. Lee-Urban, M. R. Drinkwater, and M. O. Riedl, "Skill-based Mission Generation: A Data-driven Temporal Player Modeling Approach," in *Proceedings of the The third workshop on Procedural Content Generation in Games*, ACM, 2012.
- [41] Unity Technologies, "Unity." [Game Engine] <https://unity3d.com>, 2005.

- [42] W. IJsselstein, Y. Kort, and K. Poels, "The game experience questionnaire." https://pure.tue.nl/ws/files/21666907/Game_Experience_Questionnaire_English.pdf, 2013.
- [43] S. Robot], "[srdebugger - console & tools on-device]." [Software] <https://www.stompyrobot.uk/tools/srdebugger>, 2015.

Appendix A

Testbed Game: Smash Time

Heroes

Before starting defeating the enemies, players have to choose one hero to team up (Figure 1). Although after choosing the hero team mate, players can, before starting a new level, change the hero that teams up with them to better face the challenges ahead according to the hero super power (For example: Some heroes spend less energy on their home land scenario). Every time one enemy is smashed by the player, the hero throws a magic ball to rescue the animals that were eaten by that enemy. This means that the hero becomes a target to the enemies, that will constantly try to attack him/her. If the hero is attacked in the arena game mode it is game over and, if the hero is attacked in the campaign game mode, the player has a limited time to save the hero and if that does not happen, it is game over as seen in Figure 2.



Figure 1: Smash Time Heroes.



Figure 2: Smash Time energy attacking the hero and Game Over.

Campaign Mode

This game mode is composed by different areas, each with 15 levels with a boss to fight at the final level of each area (Figure 3). While playing the campaign, players can unlock several heroes to team up with. To move forward and progress in the Campaign mode, players need to get the minimum score points to unlock the first star of each levels, although they can unlock until 3 stars to get more and better rewards.



Figure 3: Smash Time Campaign Mode Map.

Appendix B

Progression Model Pseudo Code

Algorithm 1: Arena's algorithm to manage the procedural content generation of the game and record player's performance.

1 ArenaUpdate ();

Input : void

Output : void

/ A new challenge is generated and activated when there is still one obstacle active from the previous challenge to avoid long idle times between a new challenge activation, the spawning of its obstacles and their entrance in the screen. */*

2 **if** ActiveObstaclesList.Count = 1 **then**

3 var newChallengePopulation = GenerateChallengePopulation();

4 var nextChallenge = SelectNextChallenge(newChallengePopulation);

5 ActivateChallenge(nextChallenge);

6 **end**

*/*Analyze player and obstacles' actions+Calculate player performance+Store player performance data.*/*

7 **if** player smashes one obstacle **then**

8 **if** obstacle == PurpleEnemy **then**

9 Save performance of original obstacle tag with 1;

10 **end**

11 Increment the taps done on the obstacle challenge by one;

12 **end**

13 **if** obstacle runs away from the player and escapes with success **then**

14 Save performance of original obstacle tag with 0;

15 **end**

16 **if** obstacle attacks the hero **then**

17 Save performance of original obstacle tag with 0;

18 **end**

19 **foreach** challenge activeChallengei ∈ ActiveChallengesList **do**

20 **if** activeChallengei.ActiveObstaclesList.Count == 0 **then**

21 DeactivateChallenge(activeChallengei);

22 **end**

23 **end**

Algorithm 2: Generate Challenge Data Population algorithm to create a new population of metadata challenges with content from the Challenges in the library and extra random content.

```
1 GenerateChallengeDataPopulation ();
```

Input : void

Output : ChallengeDataPopulation

```
2 var newChallengeDataPopulation = previousChallengeDataPopulation;  
3 for ( i = 0; i < ChallengePopulationCount; i + + ) {  
4     var randomChallenge = SelectRandomChallenge(ChallengeLibrary);  
5     PopulateChallengeData(randomChallenge, newChallengeDataPopulation[i]);  
6 }
```

Algorithm 3: Extract data from a Challenge from the library and populate the Challenge Data object with the content from the original Challenge and extra random obstacles' data.

```
1 PopulateChallengeData ();
```

Input : Challenge, ChallengeData

Output : void

```
2 foreach wave  $w_i \in$  Challenge do  
3     Create new wave on ChallengeData;  
4     Copy wave waypoints to the new wave;  
5     Set wave obstacles' quantity;  
6     Set wave obstacles' type;  
7     Set wave obstacles' order;  
8     Add wave obstacles' Tags to ChallengeData;  
9 end
```

```
10 Count the number of taps needed to overcome all the spawned obstacles and the obstacles that will  
    appear from the originally spawned obstacles of the ChallengeData and classify the ChallengeData with  
    the correspondent Challenge Taps Tag;
```

```
11 Copy the challenge game designer custom tags to the ChallengeData object;
```

```
12 Set a random pace to the ChallengeData and classify the ChallengeData with a Pace Tag;
```

Algorithm 4: Select Next Challenge algorithm that analyzes one challenge data population and selects the best possible challenge to be activated next.

1 SelectNextChallenge ();

Input : ChallengeDataPopulation

Output : ChallengeData

```

2 var wantedChallengeUtility = (NextChallengePerformance(calculatedChallengeUtilityCounter)
    + NextChallengeVariety(calculatedChallengeUtilityCounter))/2;
3 float wantedChallengeUtility = NextChallengeWantedPerformance *
    UTILITY_PERFORMANCE_WEIGHT
    + NextChallengeWantedVariety * UTILITY_VARIETY_WEIGHT;
4 ChallengeData closestChallenge = ChallengeDataPopulation[0];
5 float closestChallengeDistance = 1f;
6 foreach ChallengeData challengeDatai ∈ ChallengeDataPopulation do
    /* Use an heuristic function that analyses the ChallengeData tags to classify the utility of the
    challenge regarding its predicted performance and variety. */
7     float challengeUtility = CalculateChallengeUtility(challengeDatai);
8     float challengeUtilityDistance = System.Math.Abs(challengeUtility - wantedChallengeUtility);
9     if challengeUtilityDistance < closestChallengeDistance then
10         closestChallenge = challengeDatai ;
11         closestChallengeDistance = challengeUtilityDistance;
12     end
13 end
14 return closestChallenge;

```

Algorithm 5: Activate Challenge method that activates each wave in the Challenge and spawns its obstacles.

1 ActivateChallenge ();

Input : Challenge

Output : void

```

2 foreach Wave wavei ∈ Challenge do
3     ActivateWave(wavei);
4     foreach Obstacle obstaclei ∈ wavei do
5         ActivateObstacle(obstaclei);
6     end
7 end

```

Algorithm6: Deactivate Challenge method.

1 DeactivateChallenge ();

Input : Challenge

Output : void

/ Calculate Player Performance */*

2 CalculatePlayerPerformance(Challenge);

Algorithm 7: Deactivate Challenge method to calculate the final player performance to the Challenge.

1 CalculatePlayerPerformance ();

Input : Challenge

Output : float

/ Calculate the tap score of the challenge */*

2 float ChallengeTapsScore = Taps Done/Taps Needed;

3 Assign Challenge Taps Score to the performance value of the Challenge Tags (Game Designer, Pace and Taps) performance history;

4 Calculate obstacles tags performance;

/ Calculate the global challenge player performance value*/*

5 ChallengePerformance = GameDesignerTagsPerformance * GameDesignerTagsPerformanceWeight
+ PaceTagPerformance * PaceTagPerformanceWeight
+ TapsTagPerformance * TapsTagPerformanceWeight
+ ObstacleTagsPerformance * ObstacleTagsPerformanceWeight;;

Algorithm 8: Calculate Challenge Utility method estimates an utility value to allow the decision of the selection of the possible next best challenge to be generated.

1 CalculateChallengeUtility (ChallengeData);

Input : ChallengeData

Output : float

```

2 float challengeUtility = PredictChallengePerformance(challengeData)
    *.UTILITY_PERFORMANCE_W EIGHT
    + PredictChallengeVariety(challengeData)
    *.UTILITY_VARIETY_WEIGHT;

```

Algorithm 9: Predict Challenge Performance method estimates what would possibly be the performance value of the player against this challenge based on the recorded player performance data built on the Player Performance Model.

```

1 PredictChallengePerformance (ChallengeData);

```

Input : ChallengeData

Output : float

```

/* Calculate Challenge Pace Tag performance. */

```

```

2 float paceTagPerformance = PlayerData.TagsPerformance[challengeData.PaceTag];

```

```

/* Calculate Obstacle Name Tags performance. */

```

```

3 float purplePerformance      = PlayerData.TagsPerformance[EOBSTACLE_NAME.PURPLE];
4 float bluePerformance        = PlayerData.TagsPerformance[EOBSTACLE_NAME.BLUE];
5 float greenPerformance       = PlayerData.TagsPerformance[EOBSTACLE_NAME.GREEN];
6 float redPerformance          = PlayerData.TagsPerformance[EOBSTACLE_NAME.RED];
7 int purpleObstaclesCount     = challengeData.ObstaclesCount[EOBSTACLE_NAME.PURPLE];
8 int blueObstaclesCount       = challengeData.ObstaclesCount[EOBSTACLE_NAME.BLUE];
9 int greenObstaclesCount      = challengeData.ObstaclesCount[EOBSTACLE_NAME.GREEN];
10 int redObstaclesCount        = challengeData.ObstaclesCount[EOBSTACLE_NAME.RED];
11 int totalObstaclesCount      = purpleObstaclesCount + blueObstaclesCount
    + greenObstaclesCount + redObstaclesCount;
12 float obstacleTagsPerformance = ((purplePerformance * purpleObstaclesCount
    + bluePerformance * blueObstaclesCount
    + greenPerformance * greenObstaclesCount
    + redPerformance * redObstaclesCount)/totalObstaclesCount);

```

```

/* Calculate Challenge Taps Tag performance. */

```

```

13 float tapsTagPerformance = TagsPerformance[challengeData.TapsTag];

```

```

/* Calculate Challenge Game Designer Tag performance. */

```

```

14 float gameDesignerTagsPerformance = 0f;
15 foreach Wave gameDesignerTagi ∈ challengeData.GameDesignerTags do

```

```

16     gameDesignerTagsPerformance += PlayerData.TagsPerformance[gameDesignerTagi ];
17     gameDesignerTagsPerformance /= challengeData.GameDesignerTags.Count;
18 end
19 float predictedChallengePerformance =
    paceTagPerformance * PACE_TAG_PERFORMANCE_WEIGHT
    + tapsTagPerformance * TAPS_TAG_PERFORMANCE_WEIGHT
    + obstacleTagsPerformance * OBSTACLE_TAGS_PERFORMANCE_WEIGHT
    + gameDesignerTagsPerformance * GAME_DESIGNER_TAGS_PERFORMANCE_WEIGHT;
20 return predictedChallengePerformance;

```

Algorithm 10: Predict Challenge Variety method estimates what would be the variety value of the argument challenge based on the recorded tags' usage data built on the Content Variety Model.

```

1 PredictChallengeVariety (ChallengeData);

```

Input : ChallengeData

Output : float

/ Calculate Challenge Pace Tag variety. */*

```

2 float paceTagVariety = GetTagVariety(challengeData.PaceTag);

```

/ Calculate Obstacle Name Tags variety. */*

```

3 float purpleVariety    = GetTagVariety(EOBSTACLE_NAME.PURPLE);
4 float blueVariety      = GetTagVariety(EOBSTACLE_NAME.BLUE);
5 float greenVariety     = GetTagVariety(EOBSTACLE_NAME.GREEN);
6 float redVariety       = GetTagVariety(EOBSTACLE_NAME.RED);
7 int purpleObstaclesCount = challengeData.ObstaclesCount[EOBSTACLE_NAME.PURPLE] ;
8 int blueObstaclesCount  = challengeData.ObstaclesCount[EOBSTACLE_NAME.BLUE];
9 int greenObstaclesCount = challengeData.ObstaclesCount[EOBSTACLE_NAME.GREEN];
10 int redObstaclesCount  = challengeData.ObstaclesCount[EOBSTACLE_NAME.RED];
11 int totalObstaclesCount = purpleObstaclesCount
    + blueObstaclesCount
    + greenObstaclesCount
    + redObstaclesCount;
12 float obstacleTagsVariety = ((purpleVariety * purpleObstaclesCount
    + blueVariety * blueObstaclesCount
    + greenVariety * greenObstaclesCount
    + redVariety * redObstaclesCount)/totalObstaclesCount);

```



```
/* Calculate Challenge Taps Tag variety. */
```

```
13 float tapsTagVariety = GetTagVariety(challengeData.TapsTag);
```

```
/* Calculate Challenge Game Designer Tag variety. */
```

```
14 float gameDesignerTagsVariety = 0f;
```

```
15 foreach Wave gameDesignerTagi ∈ challengeData.GameDesignerTags do
```

```
16     gameDesignerTagsVariety += GetTagVariety(gameDesignerTagi);
```

```
17     gameDesignerTagsVariety /= challengeData.GameDesignerTags.Count;
```

```
18 end
```

```
19 float predictedChallengeVariety = paceTagVariety * PACE_TAG_VARIETY_WEIGHT
```

```
    + tapsTagVariety * TAPS_TAG_VARIETY_WEIGHT
```

```
    + obstacleTagsVariety * OBSTACLE_TAGS_VARIETY_WEIGHT
```

```
    + gameDesignerTagsVariety * GAME_DESIGNER_TAGS_VARIETY_WEIGHT;
```

```
20 return predictedChallengeVariety;
```


Appendix C

Playtest Procedure Guideline

Smash Time Playtest

Procedure Guideline

1 - Playtest and testbed game presentation	5 minutes
2 - Tutorial levels: Levels 1,2,3 Campaign mode	10 minutes
3 - Arena gameplay demonstration	3 minutes
4 - Questions and answers	2 minutes
3 - Play test on the Arena level	Undefined
4 - Questionnaire presentation and answering	10 minutes
Total playtest duration	30+ minutes

Appendix D

Playtest Game Data Collected Guideline

Smash Time Playtest

Game Data Collected Guideline

Track and register game data:

- Total arena gameplay duration
- Number of arena starts of a new run
- Number of times the hero gets attacked by an enemy (Game Over)
- Number of times the game session ended with time out (Game Over)
- Number of times the user quit a game session by himself

Appendix E

Game Experience Questionnaire

Thank you for participating in the playtest of the Arena game mode of Smash Time.

Please fill this quick survey and let us know how you felt while playing the game (your answers will be anonymous).

* Required

1. What is your gender? *

2. What is your age? *

3. How often do you play video games? *

Mark only one oval.

- ☐ I don't play video games.
- ☐ I play video games occasionally when the opportunity presents itself.
- ☐ I make some time in my schedule to play video games.

4. Are you familiar with mobile smasher video games (e.g. "Smash Time", "Ant Smasher", "Smash Hit", etc.)? *

Mark only one oval.

- ☐ I don't play video games.
- ☐ I play video games but not of the smasher genre on mobile.
- ☐ I am familiar with the smasher genre on mobile and played at least one game of the genre.
- ☐ The smasher genre on mobile is one my favorite genre, and I played several games of this genre.

5. Have you played the game Smash Time before? *

Mark only one oval.

- ☐ No
- ☐ Yes

6. Please indicate how you felt while playing the game for each of the items, on the following scale: *

Mark only one oval per row.

	0 - Not at all	1 - Slightly	2 - Moderately	3 - Fairly	4 - Extremely
I felt content	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt skilful	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I was interested in the game's story	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I thought it was fun	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I was fully occupied with the game	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt happy	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
It gave me a bad mood	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I thought about other things	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I found it tiresome	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt competent	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I thought it was hard	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
It was aesthetically pleasing	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I forgot everything around me	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt good	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I was good at it	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt bored	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt successful	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt imaginative	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt that I could explore things	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I enjoyed it	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I was fast at reaching the game's targets	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt annoyed	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt pressured	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt irritable	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I lost track of time	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt challenged	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I found it impressive	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I was deeply concentrated in the game	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt frustrated	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
It felt like a rich experience	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I lost connection with the outside world	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt time pressure	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I had to put a lot of effort into it	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Powered by



Appendix F

Game Data Collected Questionnaire

* Required

1. Playtest Duration? *

2. Arena Start Count? *

3. Hero Attacked Count? *

4. Time Out Count? *

5. User Quit Count *

Powered by



Google Forms

Appendix G

Scoring Guideline Game Experience Questionnaire

Competence:

- I felt skilful
- I felt competent
- I was good at it
- I felt successful
- I was fast at reaching the game's targets

Sensory and Imaginative Immersion:

- I was interested in the game's story
- It was aesthetically pleasing
- I felt imaginative
- I felt that I could explore things
- I found it impressive
- It felt like a rich experience

Flow

- I was fully occupied with the game
- I forgot everything around me
- I lost track of time
- I was deeply concentrated in the game
- I lost connection with the outside world

Tension/Annoyance

- I felt annoyed
- I felt irritable
- I felt frustrated

Challenge

- I thought it was hard
- I felt pressured
- I felt challenged
- I felt time pressure
- I had to put a lot of effort into it

Negative affect

- It gave me a bad mood
- I thought about other things
- I found it tiresome
- I felt bored

Positive affect

- I felt content
- I thought it was fun
- I felt happy
- I felt good
- I enjoyed it

Note: Scoring guidelines from the “The Game Experience Questionnaire Core Module” [41].

Appendix H

Quantitative Evaluation Results

Quantitative Evaluation

		Tests of Normality ^{c,d}					
		Kolmogorov-Smirnov ^a			Shapiro-Wilk		
	Version	Statistic	df	Sig.	Statistic	df	Sig.
Playtest Duration (s)	Old Arena	.219	16	.039	.852	16	.015
	New Arena	.149	16	.200*	.927	16	.219
Arena Start Count	Old Arena	.302	16	.000	.790	16	.002
	New Arena	.270	16	.003	.812	16	.004
Hero Attacked Count	Old Arena	.314	16	.000	.750	16	.001
	New Arena	.313	16	.000	.725	16	.000
Time Out Count	Old Arena	.292	16	.001	.861	16	.020
	New Arena	.227	16	.026	.790	16	.002

*. This is a lower bound of the true significance.

a. Lilliefors Significance Correction

c. UserQuitCount is constant when Version = Old Arena. It has been omitted.

d. UserQuitCount is constant when Version = New Arena. It has been omitted.

Appendix I

Qualitative Macro Evaluation Results

Qualitative Evaluation

Tests of Normality

	Version	Shapiro-Wilk		Shapiro-Wilk
		Statistic	df	Sig
Competence	Old Arena	.879	16	.038
	New Arena	.926	16	.213
Sensory And Imaginative Immersion	Old Arena	.945	16	.416
	New Arena	.960	16	.668
Flow	Old Arena	.973	16	.888
	New Arena	.966	16	.767
Tension And Annoyance	Old Arena	.718	16	.000
	New Arena	.787	16	.002
Challenge	Old Arena	.964	16	.738
	New Arena	.878	16	.037
Negative Affect	Old Arena	.704	16	.000
	New Arena	.883	16	.043
Positive Affect	Old Arena	.835	16	.008
	New Arena	.953	16	.533

*. This is a lower bound of the true significance.

a. Lilliefors Significance Correction

Mann-Whitney Test

Ranks

	Version	N	Mean Rank	Sum of Ranks
Competence	Old Arena	16	16.34	261.50
	New Arena	16	16.66	266.50
	Total	32		
Sensory And Imaginative Immersion	Old Arena	16	14.09	225.50
	New Arena	16	18.91	302.50
	Total	32		
Flow	Old Arena	16	14.56	233.00
	New Arena	16	18.44	295.00
	Total	32		
Tension And Annoyance	Old Arena	16	14.41	230.50
	New Arena	16	18.59	297.50
	Total	32		
Challenge	Old Arena	16	14.94	239.00
	New Arena	16	18.06	289.00
	Total	32		
Negative Affect	Old Arena	16	13.31	213.00
	New Arena	16	19.69	315.00
	Total	32		
Positive Affect	Old Arena	16	15.06	241.00
	New Arena	16	17.94	287.00
	Total	32		

Test Statistics^a

	Competence	Sensory And Imaginative Immersion	Flow	Tension And Annoyance	Challenge	Negative Affect	Positive Affect
Mann-Whitney U	125.500	89.500	97.000	94.500	103.000	77.000	105.000
Wilcoxon W	261.500	225.500	233.000	230.500	239.000	213.000	241.000
Z	-.096	-1.456	-1.172	-1.342	-.947	-2.049	-.888
Asymp. Sig. (2-tailed)	.924	.145	.241	.179	.344	.040	.375
Exact Sig. [2*(1-tailed Sig.)]	.926 ^b	.149 ^b	.254 ^b	.210 ^b	.361 ^b	.056 ^b	.402 ^b

a. Grouping Variable: Version

b. Not corrected for ties.

Appendix J

Qualitative Micro Evaluation Results

Qualitative Evaluation

Mann-Whitney U Test

Ranks

	Version	N	Mean Rank	Sum of Ranks
Felt Content	Old Arena	16	16.44	263.00
	New Arena	16	16.56	265.00
	Total	32		
Felt Skilful	Old Arena	16	16.13	258.00
	New Arena	16	16.88	270.00
	Total	32		
Interested In Story	Old Arena	16	15.56	249.00
	New Arena	16	17.44	279.00
	Total	32		
Was Fun	Old Arena	16	14.00	224.00
	New Arena	16	19.00	304.00
	Total	32		
Was Fully Occupied	Old Arena	16	16.72	267.50
	New Arena	16	16.28	260.50
	Total	32		
Felt Happy	Old Arena	16	16.56	265.00
	New Arena	16	16.44	263.00
	Total	32		
Gave Bad Mood	Old Arena	16	15.00	240.00
	New Arena	16	18.00	288.00
	Total	32		
Thought Other Things	Old Arena	16	16.06	257.00
	New Arena	16	16.94	271.00
	Total	32		
Was Tiresome	Old Arena	16	12.91	206.50
	New Arena	16	20.09	321.50
	Total	32		

Felt Competent	Old Arena	16	17.56	281.00
	New Arena	16	15.44	247.00
	Total	32		
Was Hard	Old Arena	16	14.47	231.50
	New Arena	16	18.53	296.50
	Total	32		
Was Aesthetically Pleasing	Old Arena	16	15.69	251.00
	New Arena	16	17.31	277.00
	Total	32		
Forgot Everything Around	Old Arena	16	15.09	241.50
	New Arena	16	17.91	286.50
	Total	32		
Felt Good	Old Arena	16	15.47	247.50
	New Arena	16	17.53	280.50
	Total	32		
I Was Good	Old Arena	16	15.63	250.00
	New Arena	16	17.38	278.00
	Total	32		
Felt Bored	Old Arena	16	16.50	264.00
	New Arena	16	16.50	264.00
	Total	32		
Felt Successful	Old Arena	16	16.97	271.50
	New Arena	16	16.03	256.50
	Total	32		
Felt Imaginative	Old Arena	16	15.84	253.50
	New Arena	16	17.16	274.50
	Total	32		
Felt Could Explore	Old Arena	16	15.59	249.50
	New Arena	16	17.41	278.50
	Total	32		
Enjoyed	Old Arena	16	15.09	241.50
	New Arena	16	17.91	286.50
	Total	32		
Fast Reaching Targets	Old Arena	16	17.06	273.00
	New Arena	16	15.94	255.00
	Total	32		

Felt Annoyed	Old Arena	16	14.44	231.00
	New Arena	16	18.56	297.00
	Total	32		
Felt Pressured	Old Arena	16	15.06	241.00
	New Arena	16	17.94	287.00
	Total	32		
Felt Irritable	Old Arena	16	16.63	266.00
	New Arena	16	16.38	262.00
	Total	32		
Lost Time Track	Old Arena	16	15.84	253.50
	New Arena	16	17.16	274.50
	Total	32		
Felt Challenged	Old Arena	16	14.50	232.00
	New Arena	16	18.50	296.00
	Total	32		
Was Impressive	Old Arena	16	13.13	210.00
	New Arena	16	19.88	318.00
	Total	32		
Was Deeply Concentrated	Old Arena	16	14.50	232.00
	New Arena	16	18.50	296.00
	Total	32		
Felt Frustrated	Old Arena	16	13.72	219.50
	New Arena	16	19.28	308.50
	Total	32		
Was Rich Experience	Old Arena	16	14.50	232.00
	New Arena	16	18.50	296.00
	Total	32		
Lost World Connection	Old Arena	16	14.56	233.00
	New Arena	16	18.44	295.00
	Total	32		
Felt Time Pressure	Old Arena	16	16.19	259.00
	New Arena	16	16.81	269.00
	Total	32		
Put Lot Effort	Old Arena	16	16.59	265.50
	New Arena	16	16.41	262.50
	Total	32		

Test Statistics^a

	Felt Content	Felt Skilful	Interested In Story	Was Fun	Was Fully Occupied
Mann-Whitney U	127.000	122.000	113.000	88.000	124.500
Wilcoxon W	263.000	258.000	249.000	224.000	260.500
Z	-.044	-.241	-.607	-1.718	-.153
Asymp. Sig. (2-tailed)	.965	.809	.544	.086	.879
Exact Sig. [2*(1-tailed Sig.)]	.985 ^b	.838 ^b	.590 ^b	.138 ^b	.897 ^b

	Felt Happy	Gave Bad Mood	Thought Other Things	Was Tiresome	Felt Competent
Mann-Whitney U	127.000	104.000	121.000	70.500	111.000
Wilcoxon W	263.000	240.000	257.000	206.500	247.000
Z	-.044	-1.432	-.366	-2.568	-.739
Asymp. Sig. (2-tailed)	.965	.152	.714	.010	.460
Exact Sig. [2*(1-tailed Sig.)]	.985 ^b	.381 ^b	.809 ^b	.029 ^b	.539 ^b

	Was Hard	Was Aesthetically Pleasing	Forgot Everything Around	Felt Good	I Was Good
Mann-Whitney U	95.500	115.000	105.500	111.500	114.000
Wilcoxon W	231.500	251.000	241.500	247.500	250.000
Z	-1.289	-.554	-.871	-.713	-.613
Asymp. Sig. (2-tailed)	.197	.580	.384	.476	.540
Exact Sig. [2*(1-tailed Sig.)]	.224 ^b	.642 ^b	.402 ^b	.539 ^b	.616 ^b

	Felt Bored	Felt Successful	Felt Imaginative	Felt Could Explore	Enjoyed
Mann-Whitney U	128.000	120.500	117.500	113.500	105.500
Wilcoxon W	264.000	256.500	253.500	249.500	241.500
Z	.000	-.319	-.417	-.574	-.951
Asymp. Sig. (2-tailed)	1.000	.749	.676	.566	.341
Exact Sig. [2*(1-tailed Sig.)]	1.000 ^b	.780 ^b	.696 ^b	.590 ^b	.402 ^b

	Fast Reaching Targets	Felt Annoyed	Felt Pressured	Felt Irritable	Lost Time Track
Mann-Whitney U	119.000	95.000	105.000	126.000	117.500
Wilcoxon W	255.000	231.000	241.000	262.000	253.500
Z	-.373	-1.650	-.902	-.099	-.411
Asymp. Sig. (2-tailed)	.709	.099	.367	.921	.681
Exact Sig. [2*(1-tailed Sig.)]	.752 ^b	.224 ^b	.402 ^b	.956 ^b	.696 ^b

	Felt Challenged	Was Impressive	Was Deeply Concentrated	Felt Frustrated
Mann-Whitney U	96.000	74.000	96.000	83.500
Wilcoxon W	232.000	210.000	232.000	219.500
Z	-1.324	-2.156	-1.260	-1.905
Asymp. Sig. (2-tailed)	.186	.031	.207	.057
Exact Sig. [2*(1-tailed Sig.)]	.239 ^b	.043 ^b	.239 ^b	.094 ^b

	Was Rich Experience	Lost World Connection	Felt Time Pressure	Put Lot Effort
Mann-Whitney U	96.000	97.000	123.000	126.500
Wilcoxon W	232.000	233.000	259.000	262.500
Z	-1.279	-1.203	-.197	-.058
Asymp. Sig. (2-tailed)	.201	.229	.844	.954
Exact Sig. [2*(1-tailed Sig.)]	.239 ^b	.254 ^b	.867 ^b	.956 ^b

a. Grouping Variable: Version

b. Not corrected for ties.

