TÉCNICO LISBOA

Holiday Knight: a Videogame with Skill-based Challenge Generation

João Filipe Lopes Pardal

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Professor Carlos António Roque Martinho

Examination Committee

Chairperson: Professor Miguel Nuno Dias Alves Pupo Correia
Supervisor: Professor Carlos António Roque Martinho
Member of the Committee: Professor Rui Filipe Fernandes Prada

May 2019

Acknowledgments

I want to acknowledge my dissertation supervisor Prof. Carlos Martinho for his insight, support, patience, immense amount of help and sharing of knowledge that has made this Thesis possible.

I would like to thank my parents, brother and grandmother for their friendship, encouragement and caring over all these years, for always being there for me through thick and thin and without whom this project would not be possible.

Last but not least, to all my friends and colleagues that helped me grow as a person and were always there for me during the good and bad times in my life.

To each and every one of you – Thank you.

Abstract

Challenges in video games tend to be created in a 'one challenge for all players' fashion, which creates different experiences for different players given that they do not all possess the same skills required to overcome said challenges. Some games offer the choice between a few difficulty settings like the well known easy, normal and hard format.

But what if instead of one difficulty for everyone or making the player choose the difficulty he wants, the game could adjust its challenges to suit each player in a way that would make the experience felt by all players similar?

Based on a previous work [1] that proved that, for the game used, such a model increased both time spent playing and number of times the game was replayed, when comparing an adaptive model with a conventional way of creating challenges, the work presented in this document investigated how the adaptation of the challenges affected the experience of the player where both versions of the game used the same method to create the challenges, but where one of them would adapt the challenges to the player and the other would not.

The conclusions were that there was no significant difference in time played nor in how many times the game was replayed, but the challenge felt by the players of the adaptive version was more homogeneous, meaning the challenge felt by testers was very similar, and that no loss of competence was felt by these players when compared to the static version.

Keywords

Procedural Content Generation; Player Skill Monitoring; Engagement; Adaptive Challenges.

Resumo

Os desafios em video jogos tendem a ser criados numa mentalidade de 'o mesmo desafio para toda a gente', o que gera diferentes experiências para diferentes jogadores já que estes não possuem todos o mesmo nível de competência nas capacidades necessária para ultrapassar os ditos desafios. Alternativamente, alguns jogos oferecem a escolha entre algumas dificuldades como os tão conhecidos: fácil, normal e difícil, mas, e se, em vez de ter uma dificuldade para todos os jogadores ou perguntar ao jogador que dificuldade ele acha que quer, o jogo fosse capaz de adaptar os seus desafios a cada jogador, de modo a que a experiência sentida por todos fosse semelhante?

Baseado num trabalho anteriormente desenvolvido [1] que provou que tal modelo aumenta tanto o tempo que um jogador passa a jogar como o núemro de vezes que joga quando comparado com um método tradicional de criar desafios, o trabalho apresentado neste documento investiga se a semelhante conclusão se pode chegar num jogo onde ambas as soluções usadas em testes com utilizador forem as mesmas, com a diferença de que numa há adaptção ao jogador e na outra não, e como é que esta diferença afeta a experiência do jogador.

As conclusões foram que não houve diferenças significativas na duração de uma sessão de jogo, nem houve um aumento no número de vezes que o jogo foi jogado, mas que o desafio sentido pelos jogadores que jogaram a versão em que os desafios eram adaptados foi mais homogéneo, o que significa que o nível de desafio sentido entre estes jogadores foi muito semelhante, não se verificando perda de competência quando comparados com os jogadores que jogaram a versão sem adaptação.

Palavras Chave

Geração Procedimental de Conteúdo; Monitorização de Habilidade; Envolvimento; Desafios Adaptados.

Contents

List of Figures

Acronyms

PCG Procedural Content Generation

EDPCG Experience Driven Procedural Content Generation

GEQ Game Experience Questionnaire

1

Introduction

Contents

1.1 Motivation

Nowadays most of the digital content a person consumes is somewhat adapted to them, whether it is the music we hear on Spotify [1], the videos we watch on Youtube [2], the shows we watch on Netflix [3] or video games on Steam [4]. While using these services, they gather data relating to what content is engaging for each user, and with it they predict content that he would like to consume, with the objective of showing the most engaging content to each individual user, hoping that that will make people satisfied with the service they are being provided, making them use the service often.

The premise is: if we know what content is engaging to a user, we can adapt the content we show them to maximize satisfaction on the service we are providing.

Video games being a type of digital content as well, much like the services stated previously, aim to give their users (players) an engaging experience. At their very core, video games can be seen as a combination of tasks or challenges the game tells the player he can/must do to progress. So if engagement is to be maximized on video games, the tasks they give to the player are the feature to adapt to each player's taste.

As Csikszentmihalyi [2] observed, how engaged someone is while performing a task is related to 2 variables: (1) the difficulty of the task and (2) the skill the person has to perform it. It then seems safe to assume that in order to make a video game as engaging as possible, for each individual player there needs to be a way to measure the skill of the player on a given task or challenge, and adapt, within reason, its difficulty to the player's skills.

Games like Left 4 Dead [3], Resident Evil 4 [4] and Rocksmith [5] all do this with the challenges they present to the player. In Resident Evil's [6] and Left 4 Dead's [7] cases, these games analyze the performance of the player based on factors such as health lost during encounters with enemies and shot accuracy to raise or lower the difficulty of the next wave of enemies, by increasing the damage each enemies does, or the number of enemies spawned.

In Rocksmith's case, the game promises its user to teach him how to play songs on a real electric guitar, no matter the player's familiarity with the instrument, by adapting how many notes the game shows the player at any given time. If the player can hit all the notes in a section of a song, next time that section comes along, it will come with more notes, until all the notes in the original song are being played, while on the other hand, if the game concludes that the player is struggling with a certain section of a song, it can decrease the number of notes it has, and even suggest to the player to try to play that section in isolation.

[1] https://www.spotify.com
[2] https://www.youtube.com
[3] https://www.netflix.com
[4] https://store.steampowered.com

1.2 Problem

People have different needs based on their tastes and skills. While some people can overcome a certain challenge easily, if another person is asked to overcome the exact same challenge they might have more trouble doing it. Whether they take longer to do it, if they can even complete it or how they will perform it varies from person to person.

Most video games do not take into consideration the player's skill to determine the kind of challenges that will keep him engaged. Some games, like Grand Theft Auto V [8] just have one difficulty: every player has the same challenges. Though the game's difficulty curve varies throughout the game, there is no difficulty setting the player can choose. Other games offer the standard choice of difficulty: easy, normal or hard, or some variation of this.

Neither of these are taking the player's skill into consideration, at most, it takes the player's opinion, based on games he considers similar that he played before, or asks him to choose the level of challenge he thinks he wants from a sentence that describes the different difficulties, like "Suited for beginners", "This difficulty is the intended experience of the game", or something of the sort.

In either case the problem persists, the game is not adapting to the player's skill, and by having static difficulties, each player's experience will be different, depending on their mastery level of the challenges the game presents to them and the pace at which the player learns the necessary skills to master the game, which can cause the player to quit the game, because the experience is being frustrating or boring. The question is: how can we create challenges that represent engaging gameplay experiences to the player, in a video game?

1.3 Hypothesis

A definition for how to measure engagement, while performing a task or trying to overcome a challenge, is given by the psychologist Csikszentmihalyi [2]: the state a person is in while engaged is called *cognitive flow* state, and it depends on two parameters: (1) the challenge's difficulty, and (2) the skill the person trying to overcome that challenge has, to overcome it.

This principle is one of the fundamental pillars of designing the experience of a game: in order to keep a player in cognitive flow, the game should present challenges to the player with a difficulty roughly suited for their skill level. If the difficulty of the challenge is too high compared to the player's skill he gets frustrated, if the difficulty is too low, compared to the player's skill, he gets bored.

João Catarino [1], studied this problem as well and proposed a model for the game *Smash Time* [9] where the challenges for the game would be generated based on the *cognitive flow* concept: monitoring and using the skill the player was showing in overcoming the challenges presented to them to generate the next challenges. He concluded the model had significant impact on the experience of the players.

This work will be based on Catarino's model that uses *cognitive flow* as a basis, where Catarino's solution will be applied to a different style of game that will be tested with players to understand if they will have a more engaging experience.

For this, a game will be developed from scratch and in the end two versions of this game will be compared: both versions of the game will pick challenges progressively harder for the player, where one of them will be monitoring the performance of the player against said challenges and using this data to pick the next best challenge to present to the player and the other version will not do this adaptation, more details on this in section 5.2. This will minimize the difference between the two versions tested to just the adaptation of the challenges, to test how impactful it will be on the experience of the player.

1.4 Contribution

This work intends to demonstrate that by generating challenges on a game to suit each player's skill level (within reason), and therefore creating more engaging experiences, that will make the player play the game for longer periods of time and/or enjoy the game more.

This work should reinforce how important taking the player's skill into consideration is when creating the challenges of a game.

Additionally, this work will try to understand what dimensions of the experience of the player are being affected by this adaptation.

1.5 Organization of the Document

Here is presented a summary of this document organization and topics:

- In Chapter 1 a presentation of the motivation for this work is given, as well as the problem it tries to solve, how the work will try to solve the problem presented and its expected contribution;
- In Chapter 2, the related work that informed the solution is presented;
- In Chapter 3 a description of the game used as a testbed game to this work is made;
- In Chapter 4 the implemented model is described in detail, with all its components and how they influence each other;
- In Chapter 5 a description on how the model was evaluated is made, explaining the different test phases, their objectives and their outcome;
- In Chapter 6 a summary of this work's main achievements is made, as well as future work.

2

Related Work

Contents

In this chapter concepts necessary for developing this work like the *Cognitive Flow* state and *Procedural Content Generation (PCG)* will be introduced, along with a work developed to create a progression model that is dependent on the player's performance. There will be a brief discussion after each concept to explain why they are important to this work.

2.1 Measuring Engagement - Cognitive Flow

Cognitive Flow is the mental state achieved while performing a task that is characterized by [10]:

- extreme focus on a task;
- a sense of active control;
- merging of action and awareness;
- loss of self-awareness;
- distortion of the experience of time;
- the experience of the task being the only necessary justification for continuing it;

According to Csikszentmihalyi [11], how engaged a person is while doing a task can be measured by relating 2 dimensions: challenge of a task and skill of the person performing it. In fig. 2.1 we can see the relation between these dimensions.

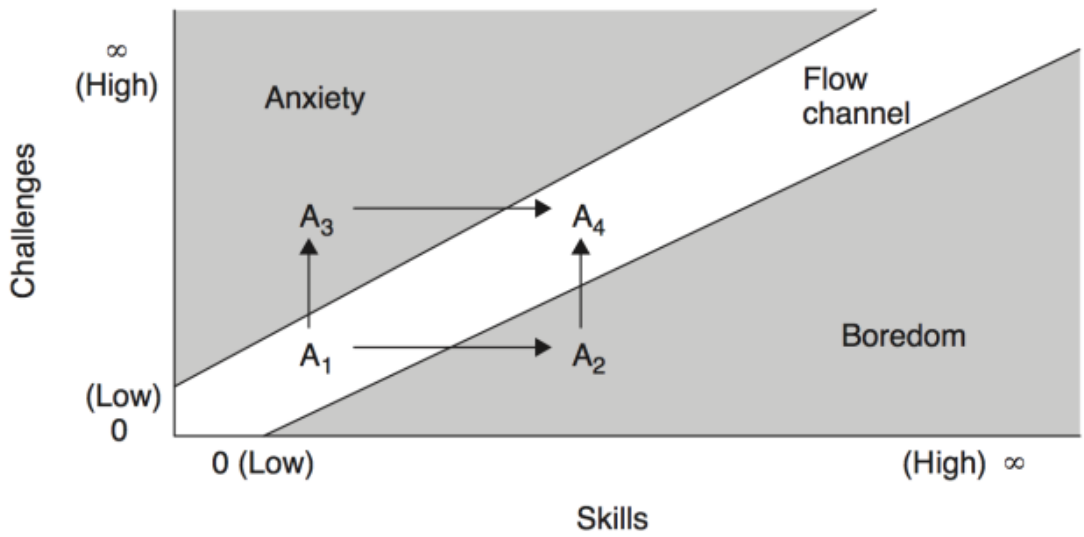


Figure 2.1: Flow Channel representation according to Csikszentmihalyi

In the figure we can see four possible states that a person might be in, while performing a task: A1, A2, A3 and A4. A person is in the cognitive flow state if the challenge of the task at hand is roughly proportional to the person's skill to accomplish it, represented in fig. 2.1 by A1 and A4.

If the challenge of the task is too high compared to the person's skill, the person will feel anxious with the challenge, represented by A3 in fig. 2.1.

Finally, if the challenge of the task is too low compared to the current skill level, the person will feel bored, represented in fig. 2.1 by A2.

In these last two cases the person is not fully engaged, and this might lead to the person quitting the task.

With this, it is possible to conclude that throughout a task, the person's skill level and the task's challenge level should increase proportionally to keep the person in cognitive flow, from A1 to A4, hence keeping the person engaged.

2.1.1 Discussion

This work's objective is to keep the player engaged to play. Understanding cognitive flow is important because it makes clear the two concepts that affect the engagement of the player while playing: the challenge (or difficulty) of a task, and the skill of the player in performing said task, along with how they should be balanced to keep the player engaged.

2.2 Procedural Content Generation (PCG)

According to Togelius et al [12], PCG is the algorithmic creation of game content with limited or indirect user input. In other words, it refers to computer software that can create game content on its own, or together with human players or designers.

PCG methods are developed and used for a number of different reasons, including saving development time and costs, increasing replayability, allowing for adaptive games, assisting designers and studying creativity and game design.

The created content can range from entire levels to textures, songs, character's models, weapons, vehicles, game rules, etc. Examples of what is not considered content in this definition are game engines and non-playable character's behaviour [13].

Game related specific knowledge such as design, affordances and constraints, should be taken into account by the content generation system, so that the generated content makes sense to that game.

2.2.1 Desirable properties for a Procedurally Generated solution

Since PCG solutions can be used in a significantly different array of problems, a list of properties can be defined that can measure how adequate the solution is to the problem presented. Some of the desirable properties of PCG solutions are:

1. speed - how urgently should the system present a solution;
2. reliability - how critical the solution is, a level without an entrance is catastrophic, whereas a patch of grass that looks weird is not game breaking;
3. controllability - properties that can be used to adjust or control the solution presented;
4. diversity - how different should different solutions be from each other;
5. believability - does this solution look like it was made by a human, rather than by an algorithm?

Tradeoffs are necessary in some scenarios, for example, if the grass is not to look weird within 50 milliseconds, maybe reliability and diversity will have to be sacrificed.

However, for most of the generated content the most important property is that it should be playable - a weapon should be usable as such, terrain should be walkable if created with that purpose, it should be possible to finish a generated level and so on.

2.2.2 Taxonomy of PCG

Due to the various problems that can be solved with PCG, there is a need to categorize them based on dimensions that each problem has. These dimensions are, based on a revised version presented by Togelius et al. [14]:

- **Online vs Offline**

  PCG techniques can be used to generate content online, as the player is playing the game, allowing the generation of endless variations, making the game infinitely replayable and opening the possibility of generating player-adapted content, or offline during the development of the game or before the start of a game session. The use of PCG for offline content generation is particularly useful when generating complex content such as environments and maps.

  An example of the use of online content generation can be found in *Left 4 Dead* [3], a first person shooter game that provides dynamic experience for each player by analyzing player's behaviour on the fly and altering the game state accordingly, using PCG techniques.

- **Necessary vs Optional**

PCG can be used to generate necessary game content that is required for the completion of a level, or it can be used to generate auxiliary content that can be discarded or exchanged for other content. The main distinctive feature between necessary and optional content is that necessary content should always be correct while this condition does not always hold true for optional content.

An example of a game where both are used is *The Binding Of Isaac* [15], a roguelike where each level is generated randomly with a certain amount of rooms per level. Each level has 4 core rooms that are mandatory, and a bunch of non-core rooms, that the player can explore [16].

- **Degree and Dimensions of Control**

  The generation of content by PCG can be controlled in different ways. The use of a random seed is one way to gain control over the generation space.

  Random seeds are used when generating the worlds in *Minecraft* [17], which means the same world can be regenerated if the same seed is used [1]. Another way to control the generation of content is to use a set of parameters that control the content generation along a number of dimensions. A vector of content features was used in *Shaker et al.* [18] to generate levels for *Infinite Mario Bros.* [19], a mod for the platformer *Super Mario Bros.* where levels are procedurally generated every time a game starts, that satisfy a set of feature specifications. In this work, some of the features used where: number of gaps, average width of gaps and gap entropy.

- **Generic vs Adaptive**

  Generic content generation refers to the paradigm of PCG where content is generated without taking player behaviour into account, as opposed to adaptive, personalized or player-centered content generation where player interaction with the game is analyzed and content is created based on a player's previous behaviour. Most commercial games tackle PCG in a generic way, while adaptive PCG has been receiving increasing attention in academia recently. A recent extensive review of PCG for player-adaptive games can be found in [20].

  Left 4 Dead [3] is an example of the use of adaptive PCG in a commercial game where an algorithm is used to adjust the pacing of the game on the fly based on the player's emotional intensity. In this case, adaptive PCG is used to adjust the difficulty of the game in order to keep the player engaged [7].

  Adaptive content generation can also be used with another motive such as the generation of more content of the kind the player seems to like.

  This approach was followed in the *Galactic Arms Race* [21] game where the weapons presented to the player are evolved based on his previous weapon use and preferences.

[1] http://www.minecraftwiki.net/

- **Stochastic vs Deterministic**

  Deterministic PCG allows the regeneration of the same content given the same starting point and method parameters as opposed to stochastic PCG where recreating the same content is usually not possible.

  The regeneration of worlds in *Minecraft* [17] is an example of the deterministic use of PCG.

- **Constructive vs Generate-and-Test**

  In constructive PCG, the content is generated in one pass, as commonly done in roguelike games. Generate-and-test PCG techniques, on the other hand, alternate generating and testing in a loop, repeating until a satisfactory solution is generated.

  Yavalath [2] is a two-player board game generated completely by a computer program using the generate-and-test paradigm [22].

- **Automatic Generation vs Mixed Authorship**

  Until recently, PCG has allowed limited input from game designers, who usually tweak the algorithm parameters to control and guide content generation while the main purpose of PCG remains the generation of infinite variations of playable content. However, a new interesting paradigm, has emerged that focuses on incorporating designer and/or player input through the design process. In this mixed-initiative paradigm, a human designer or player cooperates with the algorithm to generate the desired content.

  Pedro Lucas [23] developed a level design tool for *Legend of Grimrock 2* [24], where its interface can be used by the level designer to preview generated levels and orient the tool's behavior, being the created level a mix of what the designer wanted and what the program calculated to be an adequate level, given certain parameters. Another example is *Tanagra* [25], a system where the designer draws part of a 2D level and a constraint satisfaction algorithm is used to generate the missing parts while retaining playability.

2.2.3 Experience Driven Procedural Content Generation (EDPCG)

According to Yannakakis et al [20], recent years have seen both a boost in the size of the gaming population and a demographic diversification of computer game players [26]. Twenty years ago, game players were largely young white males with an interest in technology; nowadays, gamers can be found in every part of society [27]. This means that skills, preferences and emotion elicitation differ widely among prospective players of the same game. In order to generate the same gameplay experience, very different game content will be needed, depending on the player's skills, preferences and emotional

[2] https://boardgamegeek.com/boardgame/33767/yavalath

profile [28]. Therefore, the need for tailoring the game to individual playing experience is growing and the tasks of user modeling and affective-based adaptation within games becomes increasingly difficult. Procedural mechanisms that are able to adjust elements of the game to optimize for the experience of the player will be a necessary milestone.

Game content is viewed as building blocks of games, and games as potentiators of player experience. Since a game is synthesized by game content that, when played by a particular player, elicit player experience, one needs to assess the quality of the content generated (linked to the experiences of the player), search through the available content, and generate content that optimizes the experience for the player. The components of EDPCG are:

- Player Experience Model: player experience is modeled as a function of game content and player (the player is characterized by his playing style, and his cognitive and affective responses to gameplay stimuli);
- Content Quality: the quality of the generated content is assessed and linked to the modeled experience of the player;
- Content Representation: content is represent accordingly to maximize efficacy, performance and robustness of the generator.
- Content Generator: the generator searches through content space for content that optimizes the experience for the player according to the acquired model.

2.2.4 Discussion

In section 2.2, Togelius et al [12], define what PCG is, the different games' systems and games' production phases in which it can be implemented, as well as describing some of the desirable properties for a PCG solution, along with a taxonomy describing different properties that describe a PCG solution.

Using this taxonomy, the work presented in this document can be categorized as online (the content is being generated as the player is playing), necessary (the enemies are the base of this game) and adaptive (the behaviour of the player will have a direct impact in what is generated).

In section 2.2.3, Yannakakis et al [20] explain a more specific type of PCG, EDPCG. This article starts by justifying that the growing need to adapt the content of a video game to the player arises from the diversity of people playing games compared to a few years ago, and in order to create the same gameplay experience for everyone, there is a need to adapt the content of the game for each individual's skill, preferences and emotional profile. The authors then explain how games are potentiators for player experience and how the player needs to be accounted for when trying to build that experience.

This is what this work hopes to accomplish, trying to provide the same gameplay experience to everyone that plays the game by adapting its content (in this work's case, the challenges) to the player.

After this, the authors give a blueprint of the architecture necessary to create a PCG solution that is based on the player, with all the different components needed to achieve this.

2.3 Progression Model

In [1], Catarino et al. propose a skill-based progression model for video game *Smash Time* [9], with the objective of providing an engaging gameplay experience that will make players play this game more often, and for longer periods.

To do this, the progression model needs to have the following properties/ characteristics:

- the game should allow and support player skill development;
- the challenges of the game should keep the player in cognitive flow (the challenges should not be too difficult or too boring);
- the game should match the player's skill level at all times throughout a game session, increasing the level of difficulty of the challenges as the player's skills get better;
- the game should provide different challenges for different players, based on their skill level;
- the game should vary its content and provide new challenges at an appropriate pace.

For this to be possible, two functions were created to control the challenges' properties in this progression model: a challenge function and a variety function.

To measure and use these properties, Catarino define a set of models for his work:

1. *Player Performance Model*;
2. *Content Variety Model*;
3. *Content Generator System*.

The *Content Generator System* created for this progression model uses both the *Player Performance Model* and the *Content Variety Model* to be able to generate interesting and challenging game content throughout a game session.

A list of concepts were defined to be used by the above mentioned models:

1. *Obstacle*;
2. *Challenge*;
3. *Tags*;

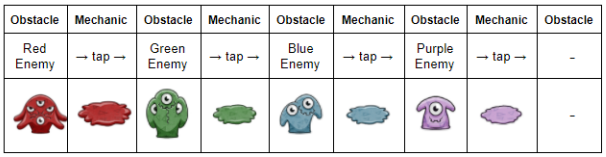


Figure 2.2: Obstacles in Catarino's work

2.3.1 Obstacles

Obstacles in this work are different types of enemies faced throughout the game. To overcome an obstacle the player has to tap on each of the obstacles, shown in Figure 2.2 a certain number of times in order to defeat them. Some obstacles generate new obstacles when they are killed.

An obstacle is considered to be overcome when all the obstacles that are generated from it, if there are any, are overcome.

2.3.2 Challenge

A challenge is defined by a group of obstacles and their properties, such as speed, their type, number of obstacles necessary to overcome the challenge and how they will move.

The quantity and types of obstacles that are spawned by one challenge is defined in real time, by the progression model. While the specific way a challenge moves is defined by the game designer, the other properties are defined by the progression model in realtime.

The progression model as a whole has the task of generating new challenges as the game progresses, while the player has the task of overcoming the obstacles of those challenges. The progression model generates a population of challenges from the challenges' library, and fills each challenge with a random quantity and type of obstacles, each time a new challenge is created.

2.3.3 Tags

The progression model assigns tags to both the challenges and obstacles of the game as a way to measure the performance value of the player for a given challenge as well as a variety value for the challenges being presented. These concepts are explained in *Player Performance Analyzer* and in *Content Variety Model*, respectively.

The tag assignment is done by both the game designer and the progression model.

These tags are grouped in the following four categories:

1. Challenge Game Designer Tags;

2. Challenge Pace Tags;

3. Challenge Taps Tags;

4. Obstacle Name Tags;

and can be found in Figure 2.3.

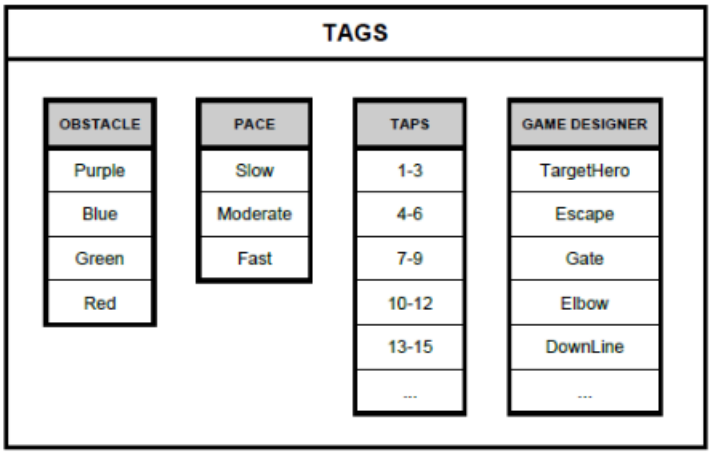


Figure 2.3: Example of tags used in Catarino's work

The Challenge Game Designer Tags are defined by the game designer, while the other three are defined autonomously, in real time, by the progression model. Tags are defined and assigned in two phases:

1. The first phase occurs offline, during the creation of the challenge's library, when the game designer creates and assigns a set of manual tags to each challenge, after it had been created. These tags are used to describe the type of objective the obstacles will present, the type of the obstacles' formation or anything the game designer wants to describe in the challenge.

2. The second phase takes place in real time when the progression model creates a new challenge with random content (quantity, type and pace of the obstacles) and assigns this values to the challenge's tags.

The proximity between two challenges, regarding their content, is calculated by the amount of common tags between them, relative to the total amount of tags of both challenges.

2.3.4 Player Performance Model

This component is responsible for evaluating the skill of the player so that it can be used, later on, to generate challenges that are fitting for his skill level.

The game designer chooses the intended performance function, as figure 2.4 suggests, that will shape the intended performance of the player throughout the game's session.

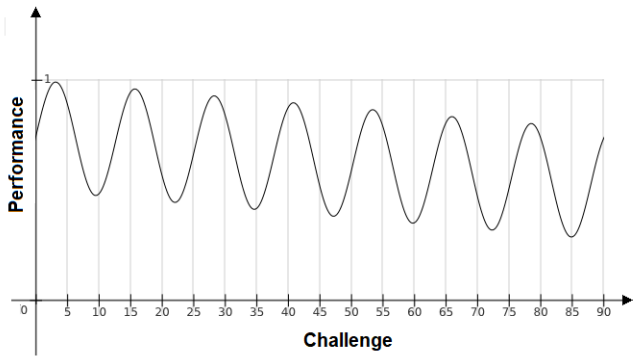


Figure 2.4: Example of an intended performance function, defined by the game designer.

To achieve this, two systems were implemented: *Player Performance Analyzer* and *Player Performance Predictive System*.

2.3.5 Player Performance Analyzer

The performance of the player in a challenge is measured by the player's dexterity to overcome the obstacles that compose the challenge.

The *Player Performance Model* stores performance data relative to the last 10 performance values of each tag of the progression model presented to the player.

While the *Obstacle Tags*' performances are analyzed individually to get more granular and accurate data, the *Challenge Tags*' (Pace, Taps, Game Designer) are all calculated using a single performance value: *ChallengeTapsScore*.

The performance of an obstacle is binary, either the player smashed the obstacle which indicates success, or the obstacle escaped or attacked the hero indicating failure. An example of the player's performance and *ChallengeTapsScore* calculation can be seen in figure 2.5.

At the top of the figure it is possible to see the tags assigned to the challenge, and at the bottom the *ChallengeTapsScore* for that challenge. A green check on the obstacle means it was overcome, and a red check means it was not.

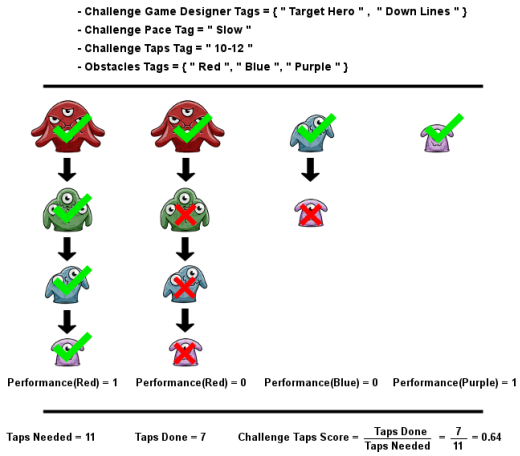


Figure 2.5: Example of calculation for *ChallengeTapsScore*.

To calculate the tags' values:

1. When all the obstacles in a challenge are overcome, each of the *Pace Tag*, *Taps Tag* and *Game Designer Tags* that the obstacles of the challenge had are updated in the player performance history. To see the exact calculations made to determine this value and all values mentioned below, refer to Catarino's work [29].

2. Each obstacle in the challenge will have its performance updated in the player performance history (1 for success or 0 for failure). Then, *Obstacle Tags* value , *Pace Tag*, *Taps Tag* and *Game Designer Tags* are calculated and are afterwards registered in each tag's performance history in the player performance data.

3. Finally, with the performance values for *Obstacle Tags*, *Pace Tag*, *Taps Tag* and *Game Designer Tags* the performance of the challenge is calculated and then it is stored in the *Player Performance Model*.

2.3.6 Player Performance Predictive System

When the *Player Performance Model* needs to estimate the player's performance to compare how fit a generated challenge is, it looks at the estimated performance of all the tags that are assigned to that challenge, according to each tag's category weight, like when the player performance is analyzed and calculated as explained above. The estimated performance of each tag is calculated by the average value of the last 10 player performance values recorded.

2.3.7 Content Variety Model

The variety of a challenge represents its novelty when compared to the most recent challenges.

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:

$$Variety(Tag) = TagUsageCount/TotalTagsUsageCount$$

As with the performance function, we will use a variety function that will shape the variation of content that is generated by the *Content Generation Model*. This function should be defined by the game designer.

Using each tag's individual variety value, *the Content Variety Model* is able to assign a variety value to a challenge the same way *Player Performance Analyzer* does to performance.

2.3.8 Game Cycle

In Figure 2.6 we can see the architecture that comprises the progression model proposed by Catarino. The game cycle is composed of a loop with 6 steps:

1. Generate a new challenge (game content) to present to the player, using:
   (a) the *Player Performance Predictive System* from the *Player Performance Model*;
   (b) the *Content Variety Data* from the *Content Variety Model*;
   (c) the *Challenge Library*.
2. Register the player's response dealing with the obstacles that compose the generated challenge;
3. The *Player Performance Analyzer* calculates the tags' values given the player's performance recorded from the generated challenge;
4. Register the player's performance data in the *Player Performance Model*;
5. Register the challenge variety data in the *Content Variety Model*;
6. Go to step 1.

2.3.9 Content Generator Model

The *Content Generation Model* is used to generate a new challenge. This model is composed by the following phases:

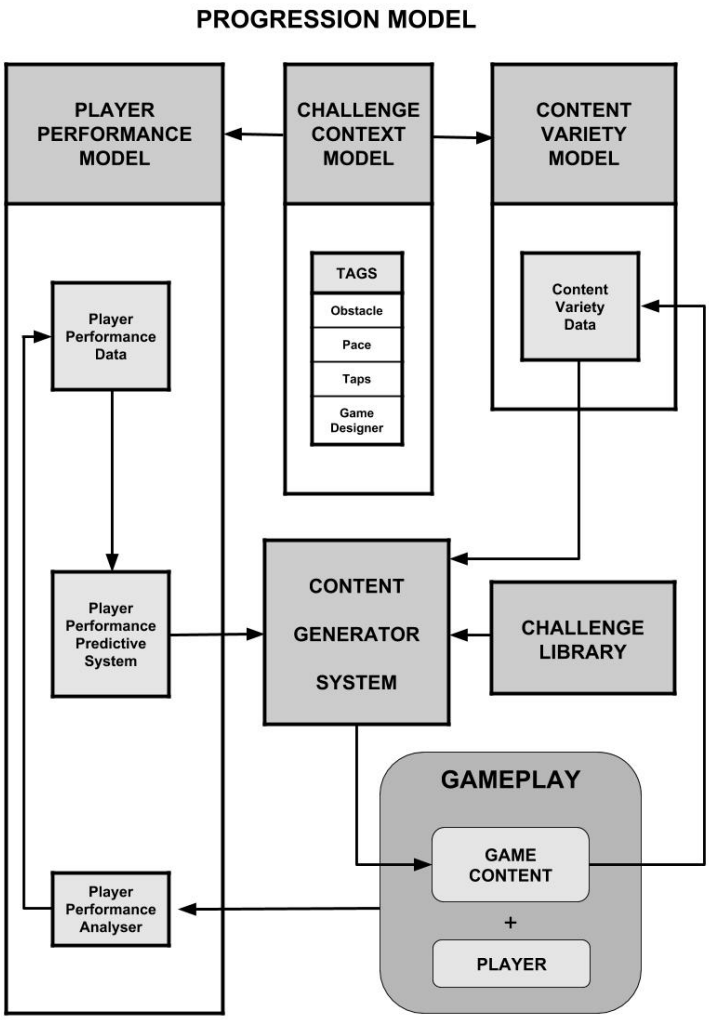


Figure 2.6: Progression model architecture used in Catarino's work, showing how the different models interact with each other

1. Generate a new population of 50 random challenges: This data includes the list of the various tags that will be assigned to the challenge, as well as the predicted player performance value calculate by the *Player Performance Predictive System* and the predicted content variety value that are both used to calculate the overall challenge utility value.

   After this *meta challenge* population has been created, it is assigned to a randomly selected template of a challenge.

2. Populate the new population of *meta challenges*:

   (a) each *meta challenge* is populated with random content for each challenge's wave: the wave obstacles' quantity; the wave obstacles' types; and the wave obstacles' order.

(b) After setting the obstacles that will be spawned, the corresponding *Obstacle Tags* are assigned to each *meta challenge*.

(c) The next step is to count the number of taps needed to overcome all the spawned obstacles and assign the correspondent *Challenge Taps Tag* to the *meta challenge*.

(d) A random pace is set for the challenge and the corresponding *Challenge Pace Tag* is assigned to the *meta challenge*.

(e) Finally, the *challenge Game Designer Tags* are copied from the original challenge, from the challenges' library, to the *meta challenge*, as well as to each wave of the challenge to be later set on each obstacle when spawned.

3. Select the next possible best challenge to be generated: Run through all the populated *meta challenges* and calculate their *utility values* using an heuristic evaluation function that combines the *predicted content variety value* of a challenge with its *predicted player performance value*. To get the best challenge candidate, the heuristic evaluation compares: the next *intended player performance value*, from the performance function defined by the game designer, with the *predicted player performance value* of each *meta challenge*; the same is then made for the *variety value* of each *meta challenge*.

   After this is done, the *Content Generation Model* knows what is the best challenge in the population to be generated and presented to the player.

4. Generate and activate a new challenge: after choosing the best possible challenge from the *meta challenge's* population, the *Content Generation Model* copies all the data from the chosen *meta challenge* to the corresponding challenge template from the library and activates the new challenge.

5. Clear all the data from the last generated and populated *meta challenges'* population and reuse it in phase 1.

2.3.10 Discussion

Catarino proposes a skill-based progression model where the skill that the player is displaying throughout the session is used, along side with intended performance and variety functions, to choose the best challenge in a pool of possible challenges with the objective of creating a more engaging experience to each player.

In EDPCG presented by Yannakakis, the components needed to implement this type of model were: (1) a *Player Experience Model*, (2) *Content Quality*, (3) *Content Representation* and (4) *Content Generator*.

In Catarino's work, the *Player Experience Model* is implemented via the performance and variety functions, that will model the player experience as a function of game content and what the player is doing.

The *Content Quality* is the information stored in *Player Performance Model* and *Content Variety Model*, that will measure how good the generated content is, comparing it to what we want the experience of the player to be.

The *Content Representation* is implemented using *Tags* and *Challenge Data*, in order to maximize the performance of the generator.

Lastly, the *Content Generator* is the entire architecture of the work, but the final result comes from the implementation of the *Content Generator Model*.

This work was described in detail since it will serve as a base to the work presented in this document. The model described in this section will be applied to a different game type, and with an evaluation method different than that used by Catarino: In Catarino's work, the two versions of the game he used on the final evaluation chose the challenges to be presented based on different algorithms, both based on the fact that a run should be 90 seconds long:

1. one used the model described in this section;
2. and in the other version the challenges were chosen based on how much time that run was going on for, where there were 3 difficulty settings, starting on the lowest, and it would increase every 30 seconds and every difficulty had a set of preset challenges.

With his evaluation he concluded that players played for longer periods of times and the amount of times they played was higher as well on the version that had the model.

The work described in this document will try to understand how impactful it is to adapt the challenges of the game by comparing two versions of the same model, one where there is adaptation and one where there is not, since in Catarino's work the challenges on the two versions of the game used in the final evaluation had different models to create the challenges.

3

Testbed Game

Contents

In this chapter the testbed game used for the implementation of this work, Holiday Knight, is described. The game's mechanics and components are explained. This game was created from scratch specifically for the model of this work to be implemented. An explanation is given for the creation of the game from scratch rather than modding an already existing game.

3.1 Choice of the Testbed Game

Initially this work meant to create a mod for the game Dark Souls [30] with the same objective: "Will players be more engaged while playing the game if the challenges are personalized?" , But a few weeks into the development of said mod problems started to arise with generation of content at runtime, limitations of what the modding tools could do at the time, and overall access do data necessary to be read or written for the model to work.

After dropping Dark Souls, some games like Binding of Isaac [15] and Legends of Grimrock II [24] were tested with the same intention, to create a mod that would implement the model this work proposes, but due to limitations as to what could be done at runtime with the available tools and access to certain information that was needed, these games where dropped as well.

Finally to make sure that there were no problems of the nature describe before, the decision was to create a game from scratch, where all information was accessible and the tools would allow creation of entities at runtime without problems. This game was made using Unity game engine[1], since it is an engine that the author had already experience in using.

3.2 Holiday Knight

Holiday Knight is a game heavily inspired in Enter the Gungeon [31] and Binding of Isaac [15], where the main character fights a number of enemies by shooting at them while at the same time dodging their shots, also known as *Bullet Hell* or *Shoot'em Up* Games, but applying two twists to these games: (1) asides from shooting weapons, they can be thrown to deal damage to the enemies and (2) the player has a shield that will protect him from incoming fire for a short period of time, and give him some bullets back.

These actions will be explained in further detail in section 3.4.

[1] https://unity.com/

3.3 Game Loop

The game consists in several rooms with enemies, in order to go to the next room the player needs to clear all enemies in the current room.

The objective is to clear twenty rooms before the character's life reaches zero. If the character's life reaches zero, the game resets, meaning he is put back in the first room and the progress of cleared rooms is set to zero.

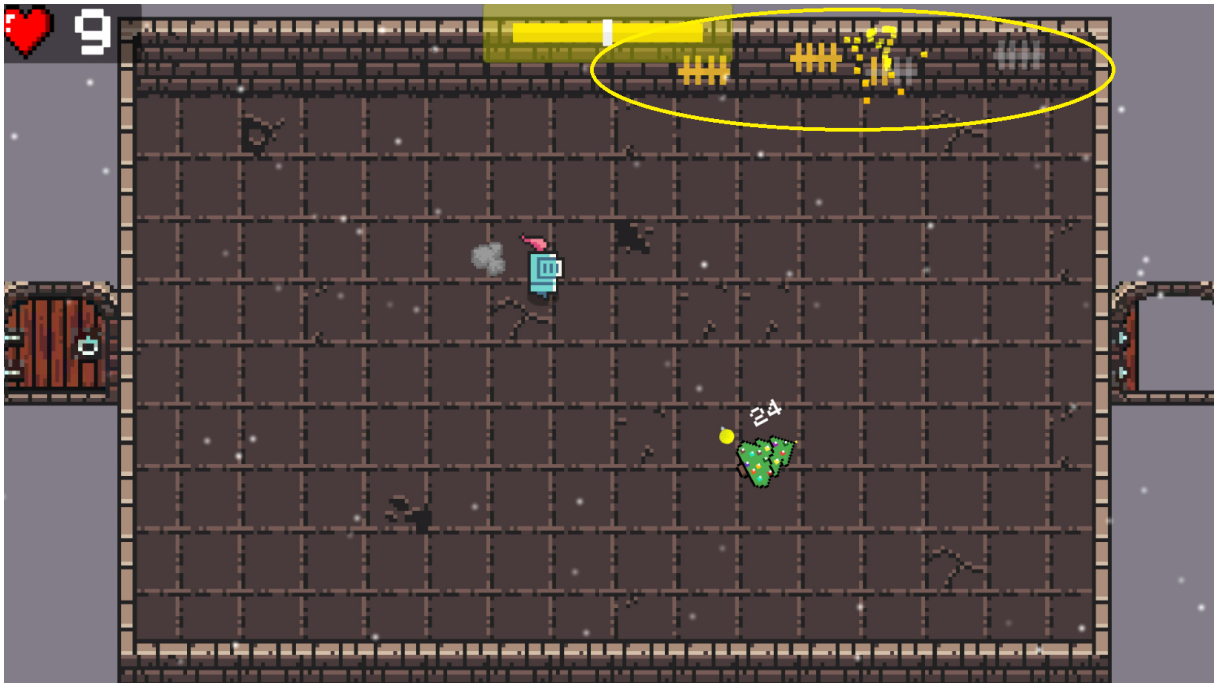


Figure 3.1: Indication of the player's current progress towards the goal of the game

3.4 Character Actions

The Actions the character can do are:

1. Walking: move from one room or position to the other, dodge enemy shots and picking up weapons are all done by walking;
2. Shooting a weapon: enemies' health needs to be dropped to zero to kill them, this is one of the ways to do so;
3. Throwing a weapon: this is the other way of damaging enemies;
4. Aiming weapons or throws: to hit an enemy with shots or throws the player needs to aim at them before executing one of those actions;

5. Shielding: this creates a shield around the player that will absorb enemies' shots for a few seconds, the shield can only be used when it is fully charged, and to charge it the player needs to damage enemies. Additionally, after the shield is depleted, for each shot the shield absorbed, a bullet is dropped for the player's gun. Contrary to the other mechanics, the player doesn't start with this mechanic unlocked, it is unlocked by progressing in the game;

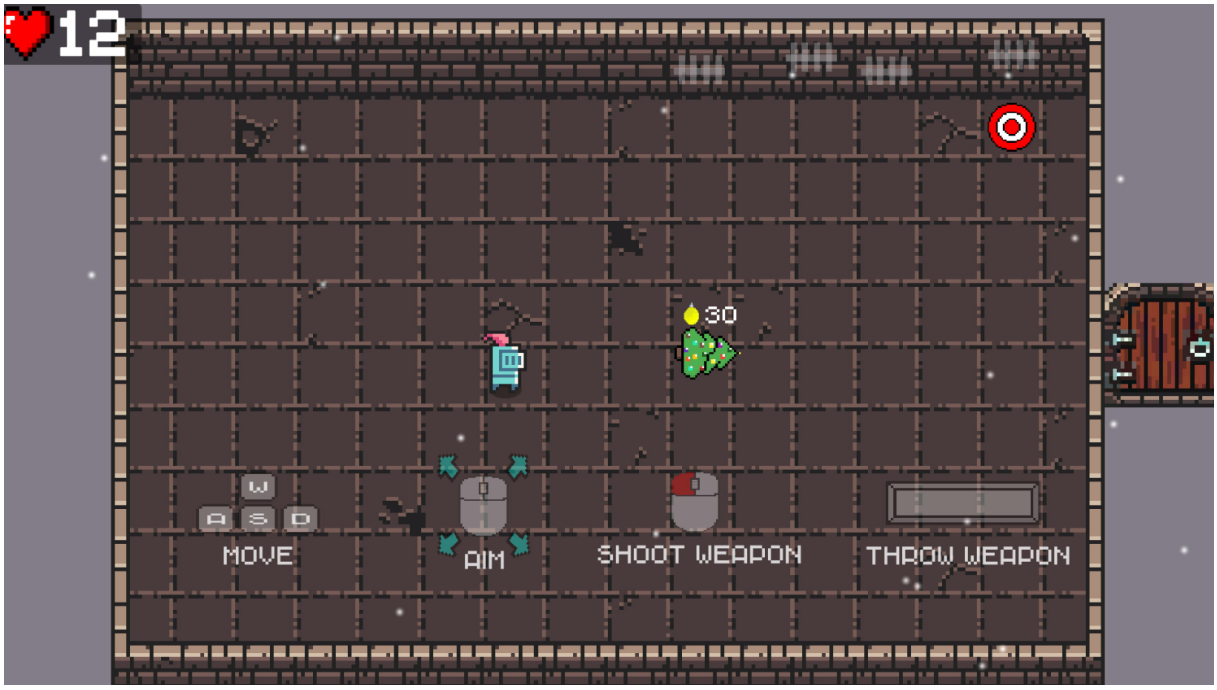


Figure 3.2: First room of the game where the controls are layed out, as well as the actions the character can do.

3.5 Enemies

Enemies are of paramount importance in the game and the model described in this document, the player succeeds or fails the game based on his performance against enemies. Enemies are characterized by:

- Their sprite and animations: the art that visually represents them, fig. 3.3, and the animations that controls their visual aspect;
- Their stats: how many health points they have, how close to the player they get and their size;
- Their shooting pattern or behaviour: each enemy has a distinct way of shooting bullets, some examples of this patterns can be seen in fig. 3.4.

To kill an enemy the player must bring the enemy's life to zero, by shooting them or throwing the weapon at them.

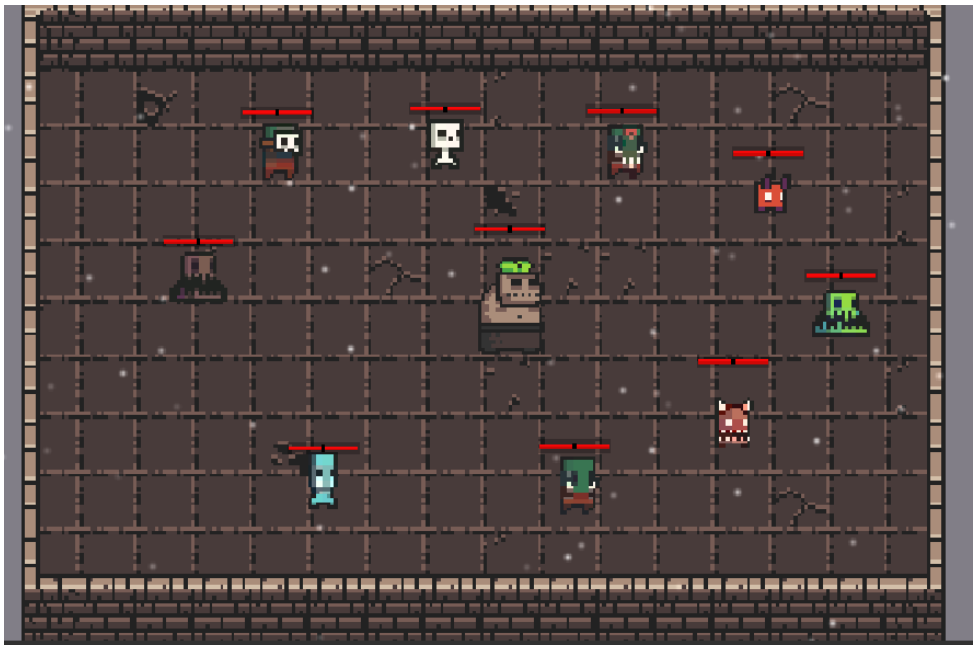


Figure 3.3: The ten default enemies in the game.

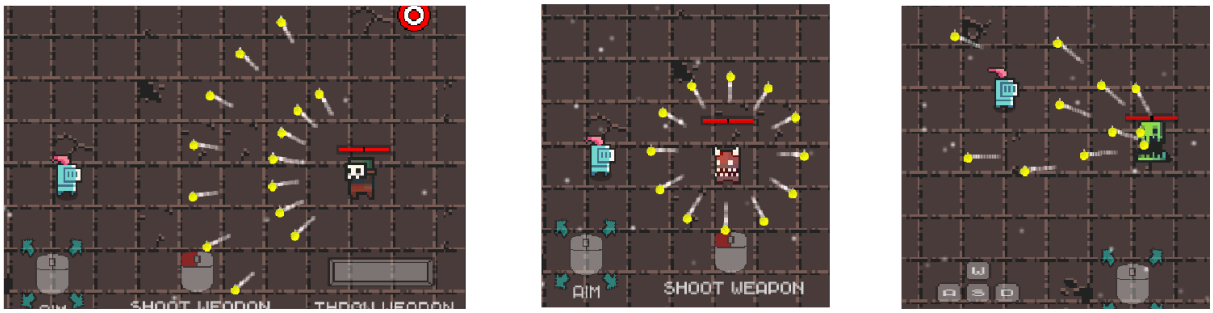


Figure 3.4: Examples of shooting patterns for some enemies

Additionally, when an enemy is killed there is a chance that he will drop bullets that the player can pick up if he is holding a weapon, or a new weapon.

The decision of what enemies to generate in the next room is what the model described in this document will influence.

3.6 Enemies' Mutations

To artificially increase the number of enemies, mutations were created. Mutations are stat or behaviour changes that are applied to the enemies presented in section 3.5, as well as a change in those enemies

visuals, as seen in fig. 3.5. With these mutations slightly different enemies were created to the ones that already exist, in one of the following ways:

1. Spawn a grenade when an enemy dies, represented by the yellowish enemies in fig. 3.5;
2. Have slightly more health points, represented by the redish enemies in fig. 3.5;

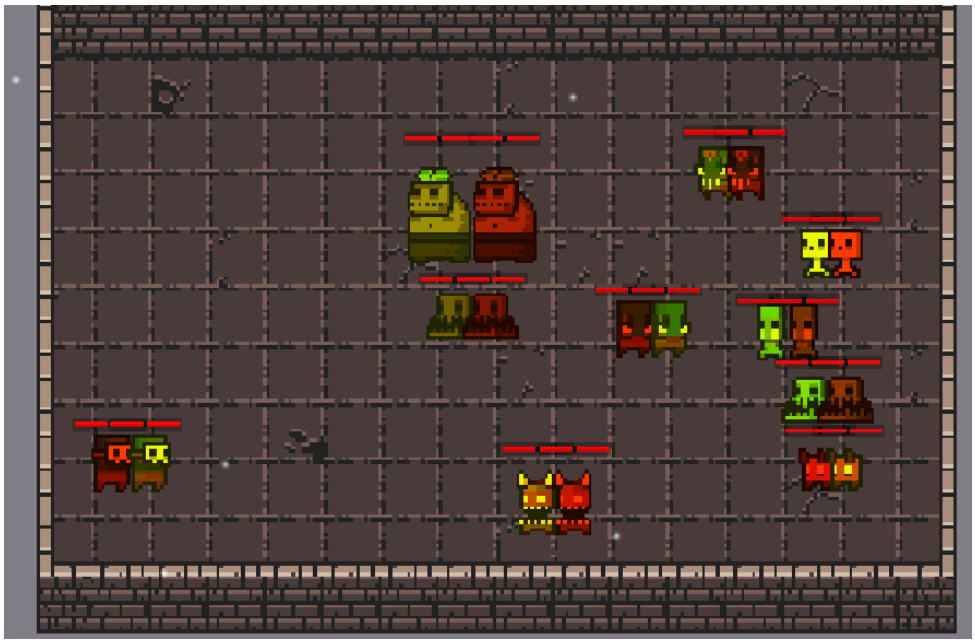


Figure 3.5: Mutated enemies, red ones have increased health points while the yellow ones spawn a grenade when they die.

This gives the game a bigger combination of possible enemies, increasing variety, which is one the two variables the model from this work is trying to optimize when looking for the best enemies to present to the player.

3.7 Weapons

The game has two holiday inspired weapons, one inspired by Christmas, and one inspire by Halloween, as seen in fig. 3.6

A weapon is characterized by:

- Its sprite: the art that visually represents it;
- The type of bullets it fires: Christmas tree decorative balls, a single bullet that hits one enemy, or a pumpkin, that explodes like a grenade damaging everyone in its radius;
- Its throwing damage: how much damage it does when it is thrown against enemies;

- Fire rate: how short is the time between shots to be able to shoot again, high fire rate means this time is small, while low fire rate means this time is bigger;
- The number of available bullets it has: when the character shoots once, the number of available bullets decreases by one.

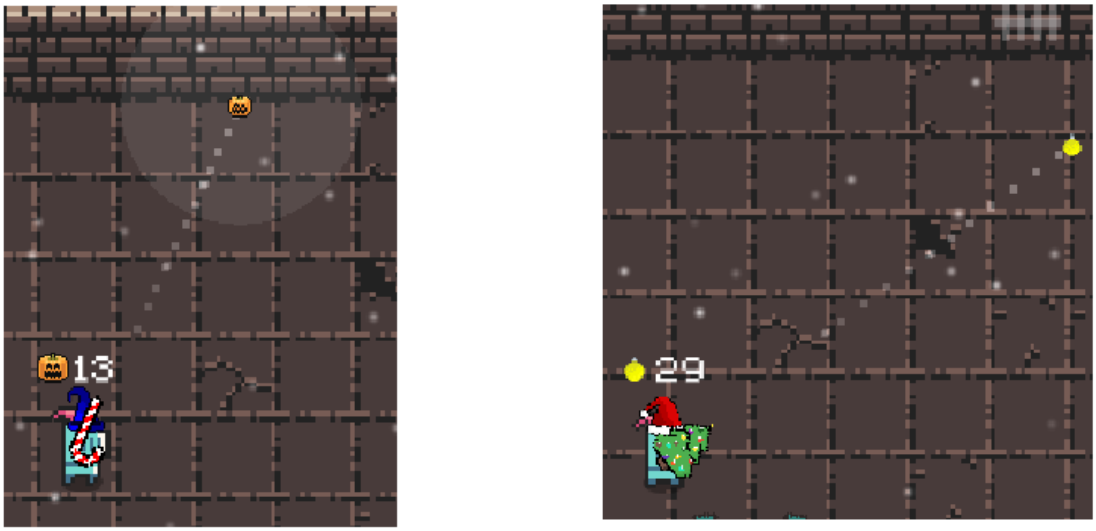


Figure 3.6: The two weapons of the game and their bullets

The decision to make two different weapons came from wanting to give the player variety when it comes to strategizing how they damage enemies: one weapon shoots bullets that only hit one enemy at a time, the damage is immediate, has high fire rate and does a lot of damage when thrown, and the other that hits multiple enemies at once, it takes a few seconds to explode, has low fire rate and doesn't do a lot of damage when thrown.

4

Implementation

Contents

In this section the various components that make up the progression model that supports this work are described in detail, as well as how they interact with the testbed game.

The model was developed in Unity [1], on top of the game described in chapter 3.

This model was developed with the intention of creating a more engaging experience to the player, more precisely in the challenges the game presents to the player with the intention of having longer playing sessions and or this sessions being more enjoyable.

For this, as discussed in section 2.1, the model needs to generate challenges that are not too difficult or too easy for each player. To accomplish this, it is necessary to measure how well the player is doing against the different challenges the game has and give that performance a value so that it is possible to compare the performance of the player on the different challenges.

Additionally, repetition of challenges is undesired, since trying to overcome the same challenges can become boring, so asides from the performance of the player against the challenges, the novelty of a challenge will be taken into consideration as well when choosing the best challenge to present to the player.

Given this, we can define three components that will make the model, and that will be explained in detail in this section:

1. Content Generation Model, section 4.5;
2. Player Performance Model, section 4.6;
3. Content Novelty Model, section 4.7;

4.1 Progression Model Cycle

The cycle for the progression model is as follows, these steps will be explained in further detail throughout this chapter:

1. Generate a new challenge to present to the player, using the *content generation model*;
2. Update the *content novelty model* for the obstacles in the challenge;
3. Record the performance of the player against each obstacle of that challenge;
4. Update the *player performance model* with the performance values of each of the obstacles overcame;
5. Repeat from step 1.

[1] https://unity.com/

4.2 Challenges

Challenges in the context of this work are characterized by:

- A number of obstacles, between 2 and 4;
- The types of each obstacle, explained in section 4.3;
- Its *novelty* value;
- The player's *predicted performance* value against this challenge;

All of the above characteristics are defined by the progression model and a challenge is considered overcome when all the obstacles in it are overcame.

4.3 Obstacles

Obstacles are the singular entities that carry meaning to the model and they are domain (game) based. The obstacles for this game are each enemy that will be part of a challenge that the player needs to overcome. These enemies are described in detail in section 3.5.

To overcome an obstacle in the testbed game, the player needs to reduce its life to zero, by shooting the enemies or throwing weapons at them.

The way the obstacles are given meaning to the model is by adding a set of tags that characterize them. Tags are described in section 4.4.

4.4 Tags

For the model to work, there needs to be a way of associating meaningful characteristics of an obstacle to a performance and a novelty value.

The way this is done is to associate tags to obstacles that have the characteristics that we want to base the performance and novelty on. For example, if the game has strong and weak enemies, fast and slow, enemies that shoot weapon from afar and enemies that can only damage players when they are close, these are all characteristics that can serve as tags. If the game has an enemy that has high attack stats, can only attack with a sword and moves slowly, we can give it tags such has {*highDamage, meleeRange, slowMovement*}.

With this, the model supports different enemies having commons tags, one such example is fig. 4.2, that will be used throughout this section to illustrate how different parts of the model work.

In this specific work, each enemy is only characterized by a tag with its name, rather than its characteristics, as seen in fig. 4.1. To keep the example used throughout this section as generic as possible, the tags used by the different components of the model will be the ones from fig. 4.2.

After overcoming said enemy, each of these tags will be given a performance value. More on this in section 4.6.

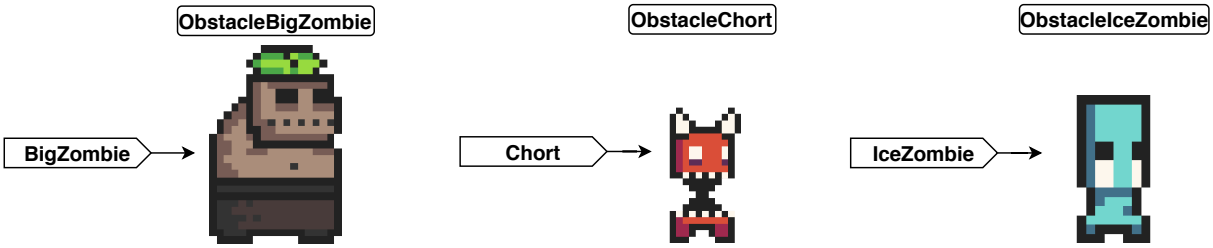


Figure 4.1: Examples of tags used in this work.

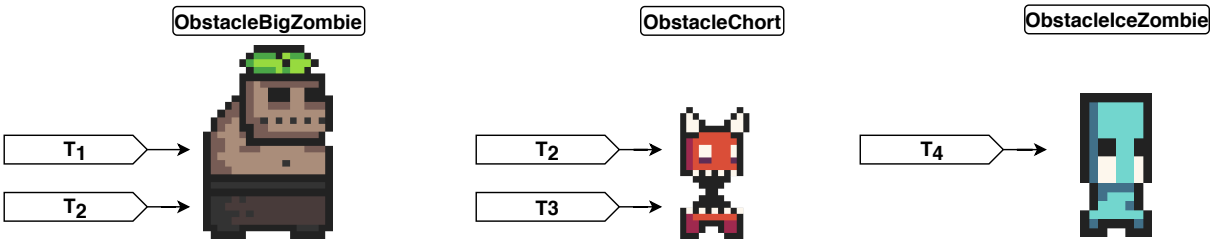


Figure 4.2: Example of tags for different obstacles that will be used throughout this section.

4.5 Content Generation Model

This section will describe the *content generation model*, the core of this work. This component is the one responsible for connecting the *player performance model*, *content novelty model* and the game content.

Every time the player overcomes a challenge, this is the component responsible for triggering the mechanisms necessary for generating the next possible challenges, and picking the best one to present to the player.

To start, it will generate a population of 50 random challenges that represent the pool of possible challenges that the model will choose the best solution from to present to the player. After this generation the work of this component is cyclical and its steps are as follows:

1. Of the information stated in section 4.2, that comprises a challenge, this component starts by attributing, to one possible challenge, a *number of obstacles* and the *type of each obstacle* at random.

After this, the *predicted performance* value, see section 4.6.3, and *novelty* value, see section 4.7.1, are calculated for this possible challenge. These values need to be compared to some kind of intended values for the next challenge to be generated, so that the model knows how useful this possible challenge is.

So, the *content generation model* asks for the *intended performance* and *intended novelty* values for the next challenge to be generated, more on these values in section 4.6.4 and section 4.7.2 respectively, and calculates how close the predicted values are to the intended ones, as stated in eq. (4.1) for performance, and in eq. (4.2) for novelty. Both *performance proximity* and *novelty proximity* values range from 0 to 100.

$$\begin{aligned} performanceProximity = &|predictedPerformanceForPossibleChallenge \\ &- intendedPerformanceForNextChallenge| \end{aligned} \tag{4.1}$$

$$\begin{aligned} noveltyProximity = &|noveltyForPossibleChallenge \\ &- intendedNoveltyForNextChallenge| \end{aligned} \tag{4.2}$$

At this point the model knows the proximity of this possible challenge to the intended challenge, but it still needs to take into account the weight given to each of these parameters. These weights are set by the game designer, and range from 0 to 100. With this, the *total utility* of a challenge is given by eq. (4.3).

$$\begin{aligned} totalUtilityIn\% = 100 - &performanceProximity * performanceWeight \\ &+ noveltyProximity * (100 - performanceWeight) \end{aligned} \tag{4.3}$$

This value represents the accuracy, between 0 and 100%, of how good this possible challenge is, when compared to what the designer intended the next challenge's performance and novelty to be;

2. After calculating the *total utility* for a challenge, this value is compared to the best possible challenge's *total utility* the *content generation model* found so far. If the *total utility* of this challenge is higher than that of the best possible challenge so far, it becomes the best;
3. Steps 1 and 2 are repeated until all possible challenges have been evaluated;
4. When all possible solutions have been evaluated, the *content generation model* holds the best challenge;

5. The *content novelty model* is updated so that the tags for all obstacles in the best challenge have their counts updated;

6. The best possible challenge is created in the game as the next challenge for the player to overcome.

4.6 Player Performance Model

This model serves four purposes:

1. Storing performance values for each tag;

2. Analyzing the player performance for each challenge that is generated, section 4.6.2;

3. Predicting the player's performance against a possible challenge, section 4.6.3;

4. Defining and informing *content generation model* what the intended performance for the next challenge is, section 4.6.4;

4.6.1 Heuristic for Performance

The success of the overall work presented in this document is strongly impacted by the choice of the heuristic that is chosen to represent performance, and this is the part of the model that is most domain based.

This describes what variables are being taken into consideration to calculate performance, the metrics that represent how the player's skill will be measured in the model.

For a shooter game, the performance might be related to the number of bullets the player hit on the enemy related to how many bullets he shot, the number of shots enemies hit on the player or the overall time he took to kill an enemy, to name a few.

In this work the decision on which heuristic to use was based on being as simple and domain free as possible, so the choice was to use time to overcome each obstacle in a challenge as the heuristic. The reasoning for this was three folded: (1) if the results of the study prove this model can support our hypothesis for the specific game genre of the testbed game with such a simple heuristic, future work for this specific genre can be to find better heuristics that can be more domain based when compared to this one, (2) in this work part of the focus is to see if the model works for the testbed game genre, it is not to find the best heuristic, and (3), to try to create a model as generic as possible for this work, so that the results are not domain based, and that people can apply this model as a baseline in other game genres.

In a first iteration, the time started counting as soon as the player entered a room, and the time for each obstacle would be stopped when the player overcame it, the problem was that obstacles that were left to be overcome lastly took longer to overcome, hence the player would have a worse performance against said obstacle, but the reality was that usually a player would take down the most dangerous obstacles first, and this was not suppose to influence the time count. To mitigate this, the decision was to only start to count the time, for each obstacle, after the player hit him the first time, and the time would stop being counted after the obstacles was overcame.

4.6.2 Player Performance Analyzer

This is the component responsible for giving a performance value to all the tags for all the obstacles in a challenge, after the player overcame it.

For this, this component needs information related to the *tag performance history*, that stores the last 10 performance values of the player for each tag, and *tag performance average*, that is a single value that represents the average of the data in *tag performance history*, and it is the value used to calculate the *predicted performance* value in section 4.6.3.

As described in section 4.6.1 the heuristic being used to calculate the performance is the time it takes the player to overcome each obstacle. The higher the time, the worst the performance against said obstacle. After the challenge is overcome, all obstacles on it have an associated time it took to clear them, as shown in fig. 4.3.

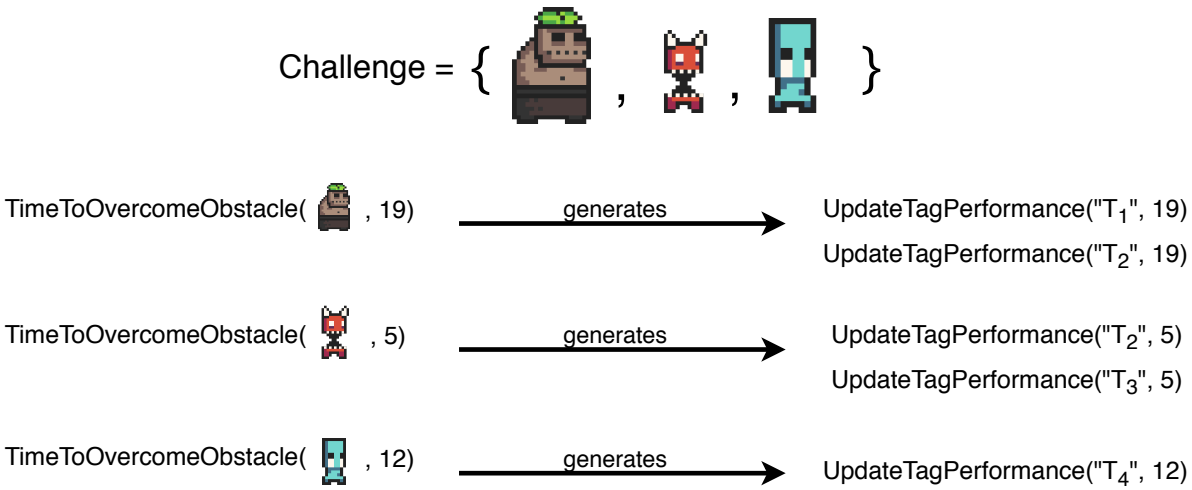


Figure 4.3: Time it took to overcome each of the obstacles in this challenge.

Then, the analyzer is responsible for updating the *tag performance* values for all tags in the challenge's obstacles in the following way, example in fig. 4.4:

- The time to beat an obstacle is added to the *tag performance history* of the obstacle's tag(s), in

the form of a pair <*tag(s) for the obstacle, time to beat obstacle*>;

- The *tag average performance* is updated to all tags that had their *tag performance history* updated;

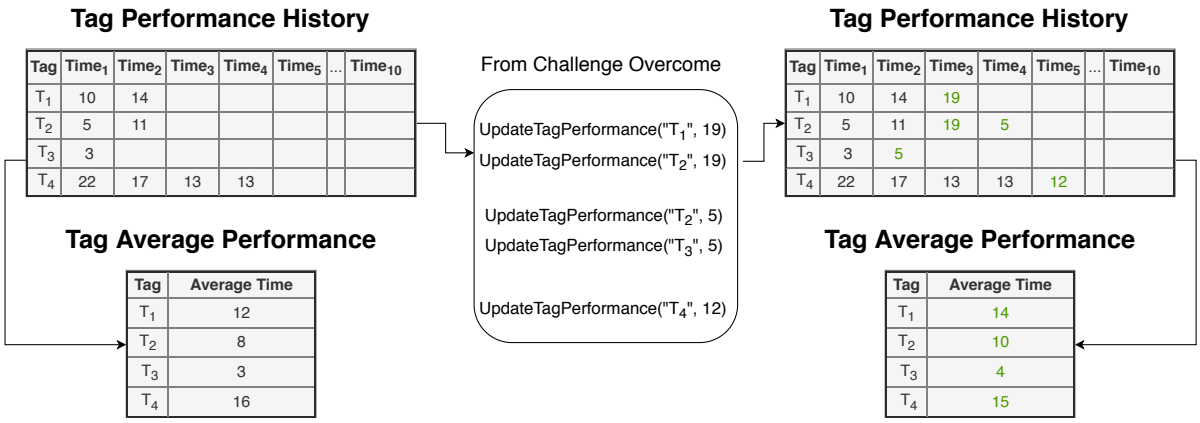


Figure 4.4: Player performance analyzer updating performance data after a challenge has been overcome.

When the game is initialized the performance values associated with each tag are bootstrap values that were calculated empirically, see section 5.1 for more information on this, these values are added once per tag in *tag performance history*, so that the model starts with some idea of the expected performance against each tag. The more a certain obstacle appears with that tag, the less these bootstrap values influence the model, until they are disregarded completely after the *tag performance history* is full, and the older performance values are discarded to make room for the new ones.

4.6.3 Player Performance Predictor

In order to know how good or bad a possible challenge is, it is necessary to have an estimate on the player's performance for that possible challenge so that it can be compared with the intended performance for the next challenge.

This component is responsible for calculating the *predicted performance* value for a given challenge. For each possible challenge generated, this component goes through the tags of the obstacles that compose the challenge, calculates its *predicted performance* value and informs the content generation model of said value.

The values used to calculate the *predicted performance* value for a challenge are : (1) how many obstacles the challenge has and (2) the average of all obstacles' in that challenge *tag performance average*. A visual example of the procedure described next to calculate this value can be seen in fig. 4.5.

The number of obstacles a challenge has needs to influence the *predicted performance* of the challenge, or else this value would be the same for a challenge where there are 2 obstacles with tag *T1* and

a challenge where there are 4 obstacles with the same tag, and in the case of this specific game, this is not true, as 4 enemies will make the challenge harder, hence potentially decreasing the performance of the player, so, to mitigate this, the formula eq. (4.4) is used to calculate the *predicted performance* value for a possible challenge,

$$\begin{aligned} predictedPerformanceIn\% = \\ avgPredictedPerformanceForObstaclesInChallengeAs\% * performanceWeight \\ + numberOfObstaclesAs\% * (100 - performanceWeight) \end{aligned} \tag{4.4}$$

$$numberOfObstaclesAs\% = \frac{numberOfObstaclesInThisChallenge - minAmountOfObstaclesInAChallenge}{maxAmountOfObstaclesInAChallenge - minAmountOfObstaclesInAChallenge} * 100 \tag{4.5}$$

where *performanceWeight* is a value between 0 and 100 that determines how much each of the two values will influence the overall predicted performance for a challenge. In this work, the value used for *performanceWeight* was 65.

As mentioned in section 4.5, the *predicted performance* has to be a value between 0 and 100 so that it is comparable to the *novelty*, and in a more generic sense, to create an abstraction between the heuristic value used to calculate performance and the rest of the model.

For this the player performance analyzer will transform the time value in a comparative performance value, where 0 will be associated with the tag the player took the longest to overcome, and 100 to the tag that he took the shortest amount of time, with the formula eq. (4.6). It is important to remember that this performance value is supposed to be a value that indicates how well the player will do against this specific tag, 100 means the player has been having the best performances against obstacles with this tag, and 0 means the player has been having the worst performances against obstacles that have this tag.

When the challenge is complete, the *tag performance average* is recalculated for all obstacles, transforming this time into percentages:

- The maximum and minimum time to beat any tag is recorded throughout this part, storing the tag that had the highest and the lowest time to beat, by default maximum value is set to 100, and minimum to 0. It is extremely important to refer that this maximum is refereeing to the performance, which is inverse to the time, so maximum performance refers to the minimum time, and minimum performance refers to maximum time;

- For each tag, *tag performance average* is calculated, using all values in the *tag performance history*

for that tag, and the amount of times that tag was used;

- Then, the *tag performance average* is compared to the minimum and maximum times, and if the average is bigger than the minimum, the average is now the minimum, if the average is smaller than the maximum, it is now the new maximum;

- Lastly, after these averages have been calculated for all tags, the model associates a performance of 0% to the maximum value, 100% to the minimum value, and calculates all other performance values in between by linearly interpolating between these two values using:

$$performanceForTagTin\% = \frac{performanceForTagT(secs) - maxPerformance(secs)}{minPerformance(secs) - maxPerformance(secs)} * 100 \tag{4.6}$$

The value calculated by formula eq. (4.4) is then returned as the *predicted performance* value for that possible challenge.

4.6.4 Intended Performance Function

This function is defined by the game designer and it represents how the designer intends the performance of the player to vary throughout the game. The output of this function is influenced by the amount of challenges the player has overcome. Examples of 2 possible functions can be seen in fig. 4.6.

4.7 Content Novelty Model

This model serves three purposes:

1. Storing novelty values for each tag;

2. Calculating the novelty of a possible challenge, section 4.7.1;

3. Defining and informing *content generation model* of what the intended novelty for the next challenge is, section 4.7.2;

When the *content generation model* chooses the next challenge to be presented to the player, the novelty values for the tags on the obstacles of that challenge are updated by incrementing their *tag usage count*. But, as in *predicted performance* value, *tag usage count* needs to be a value between 0 and 100, so that it is comparable to the *performance* of a possible challenge, and in a more generic sense, to create an abstraction between the way the novelty value is calculated and the rest of the model.

To store this information, a *novelty percentage* value is created, storing the current novelty value for each tag, each with a value ranging from 0 to 100, where 0 is attributed to the most used tags, and 100

to the least used one in the challenges presented to the player. All other *novelty percentage* values are linearly interpolated between these two values using eq. (4.7), and an example can be seen in fig. 4.7:

$$noveltyForTagTin\% = 100 - \frac{usageCountForTagT - leastUsedTagCount}{mostUsedTagCount - leastUsedTagCount} * 100 \tag{4.7}$$

Tag novelty values are updated every time a challenge is presented to the player.

4.7.1 Content Novelty Calculator

In order to know how good or bad a possible challenge is, it is necessary to know the novelty for that possible challenge, so that it can be compared with the intended novelty for the next challenge.

This component is responsible for calculating the novelty of a possible challenge.

To do this, for each possible challenge generated, this component goes through the tags of the obstacles that compose the challenge, it averages their *novelty percentage* values, and informs the content generation model of said value.

4.7.2 Intended Novelty Function

This function is defined by the game designer and it represents how the designer intends the novelty of the challenges to vary throughout the game. The output of this function is influenced by the amount of challenges the player has overcome. Examples of 2 possible functions can be seen in fig. 4.6.

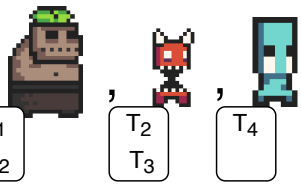
eq. (4.4)

PredictedPerformanceIn% = AvgPredictedPerformanceForObstaclesInAChallengesAs% * performanceWeight
+ numberOfObstaclesAs% * (100 - performanceWeight)

Tag Average Performance —eq. (4.6)→ **Tag Average Performance In %**

Tag	Average Time
T_1	14
T_2	10
T_3	4
T_4	15

Tag	Performance In %
T_1	9
T_2	45
T_3	100
T_4	0

Challenge = { 

}

AvgPredictedPerformanceForObstaclesInAChallengesAs% = (9 + 45 + 45 + 100 + 0) / 5
= 40

eq. (4.5) → maxAmountOfObstaclesInAChallenge = 4
minAmountOfObstaclesInAChallenge = 2

numberOfObstaclesAs% = (3 - 4) / (2 - 4) * 100
= 50

PredictedPerformanceIn%For {  } = 40 * performanceWeight
+ 50 * (100 - performanceWeight)

Figure 4.5: Example of how *predictedPerformanceIn%* is calculated for a challenge.

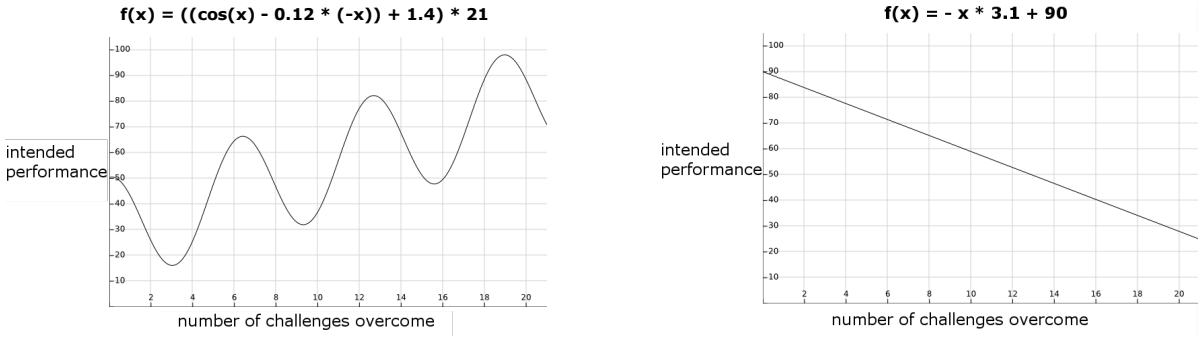


Figure 4.6: Example of intended performance functions, to the left the performance of the player is suppose to, overall, increase the more challenges he overcomes (challenges getting easier), and to the right his performance should decrease(challenges getting harder). The function on the right was used in this model.

Tag	Usage Count
T_A	2
T_B	11
T_C	3
T_D	5
T_E	9

Tag	Novelty in %
T_A	100
T_B	0
T_C	89
T_D	67
T_E	22

Figure 4.7: Converting *tag usage count* to novelty percentage.

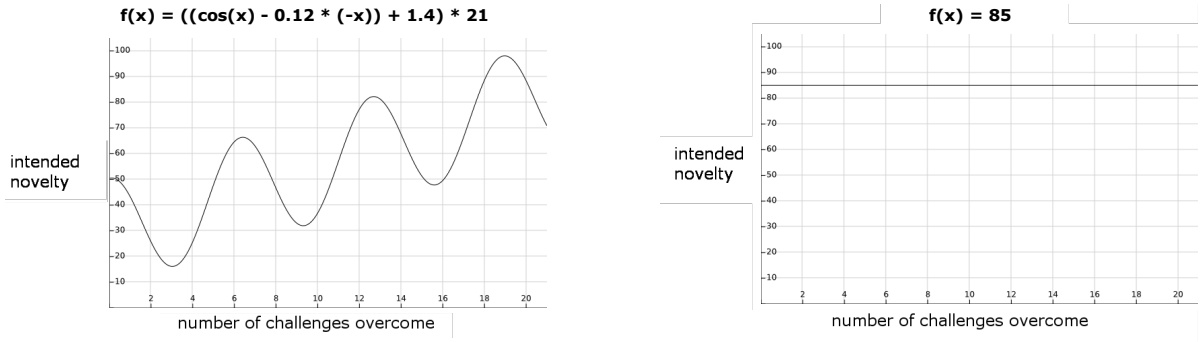


Figure 4.8: Example of intended novelty functions, to the left the novelty of the challenges is suppose to, overall, increase the more challenges a player overcomes, and to the right, the novelty is supposed to be the same throughout the game. The function on the right was used in this model.

5

Evaluation

Contents

In this section the 2 evaluation phases of this work are described: (1) the preliminary evaluation, where the main objective was to gather data from players, relating to the obstacles of the game, to be used in the model described in chapter 4, and (2) the final evaluation where the main objective was to test if the model implemented supported our hypothesis.

Objectives, procedures and results are stated for both of this evaluations that were done with testers.

5.1 Preliminary Evaluation

5.1.1 Objectives

The preliminary evaluation served 3 purposes:

1. Get bootstrap values for each tag's performance: Initial values to replace the default value for each tag's performance are needed, bootstrap values that will be used when the player first plays the game, and the model does not have data about the player's performance, since when the game starts, all tag's performance values are set to their default value (-1, to inform that no performance data exists for a tag).

   These values will be the performance for each of the tags when the game is started and they are progressively replaced with the player's performance while he plays, until the model has collected enough data about the player's performance for each tag to just use that data and forget about these bootstrap values;

2. Get a performance value to be attributed to a tag when the player dies and find a maximum value for the time the player can take to overcome an enemy: There were two important decisions that needed to be made: (1) what value should be given to the tags when the player doesn't successfully overcome a challenge? and (2) should there be a limit of time to defeat a single enemy?

   Since in these tests time for each tag was being tracked, the decision was that these questions would be answered based on the data we collected;

3. Feedback about the game mechanics and identify bugs: Since the game has been developed from scratch and it hasn't been tested by players for a few weeks prior to this test, this test served to make sure that bugs were found and dealt with before the final tests, to make sure the players understood the controls, objective and mechanics of the game, to understand if all the feedback that the player was expecting from the game was implemented, and overall to understand if the testers were enjoying the experience of playing the game.

5.1.2 Procedure

To get the bootstrap values for each tag's performance, a preliminary evaluation was made with 6 people, one of them working in the game development industry, where the model was adjusted so that the best possible solution for each room was the one with the highest novelty utility, ignoring performance, to make sure the biggest amount of data for each tag was recorded.

For this data to be recorded a new functionality was added to the game to record all performance values for each tag, instead of only keeping the last 10 values per tag, and write those values on a file that would not be destroyed after the game stopped running.

Testers were asked to play the game for a few minutes and ask questions if they felt like something was not clear or working as they expected. While the session was occurring we were observing the game being played to spot any undesired behaviour the game might had that needed to be fixed.

5.1.3 Results and Changes

After these sessions, we had the necessary data to create the bootstrap values for each tag's performance. The performance data for each tag was averaged and that value was assigned to the corresponding tag.

As for the value that should be given to a tag when the player doesn't overcome a challenge, the problem that needs to be solved is: what value should be given to the tag of an enemy when the player doesn't overcome a challenge and that enemy is still alive? For this, using the data collected from these tests, the 95th percentile of each tag was calculated, to mitigate the effect of outliers, and the highest value of all tags was the one used, in this case 40. This means that every time the player doesn't overcome a challenge, all enemies' that are alive in the room get a tag update with value *time to beat obstacle*, section 4.6.2, of 40, where in average, during this evaluation, these value ranged from 8 to 19 seconds, depending on the enemy.

This raised another problem: if there is a value that is assigned to a tag when the player doesn't overcome the challenge, which should indicate the worst performance case, what does a value higher than that represent? If the player takes longer to overcome a challenge, should he be penalized with a value higher than that associated with him failing to complete the challenge? For this reason the decision was to give a threshold to the value a tag can have. To choose this threshold, it was done so in a way where the problem stated previously would not exist, so this threshold was set to the same value as the penalty for when a player doesn't overcome the challenge (in this case 40). With this decision, there is no difference, performance wise, to the player failing to overcome a challenge and taking a long time to overcome one.

It is important to note that the data extracted from this evaluation was not changed to accommodate

this rule, meaning that all values that were above the aforementioned threshold stayed that way, this threshold was only imposed in the final tests.

Another important remark is that out of the 809 saved performance values across this evaluation, 54 occurrences where higher or equal to the previously established value of 40 (corresponding to 6,7% of the total values).

5.2 Final Evaluation

5.2.1 Objectives

Part of the contribution of this work is to understand if by creating more engaging challenges to the player (1) he plays the game for longer periods and (2) if his enjoyment of the time spent playing is greater and to understand the impact of that adaptation on the player's experience.

To do so this evaluation aimed to gather significant data from people that played the game in a way that would either backup or refute these ideas.

5.2.2 Procedure

To test the premise of the players spending more time playing when the challenges are adapted to him, the decision was to create logs that automatically saved the time the tester spent playing the game, as well as the total number of runs, i.e. times the tester played the game and how many rooms he cleared in each run.

To be able to evaluate the enjoyment of his time playing, and how the tester's experience was affected by the adaptation of the challenges, after the tester finished playing the game he was asked to fill the Game Experience Questionnaire (GEQ)'s [32] core questionnaire. This tool assesses game experience as scores on seven components for each tester:

1. Immersion;
2. Flow;
3. Competence;
4. Positive Affect;
5. Negative Affect;
6. Tension;
7. Challenge.

Each of this components has a set of questions associated with it, where the answer is in a Likert scale format, where the answer's values range from *not at all* with a value of 0, to *extremely* with a value of 4. The way the score of each component is calculated is by averaging the values of each answer in that component. The scoring guideline can be found in appendix A.

With this in mind, the final evaluation consisted of 2 phases, where each tester was asked, firstly, to play the game for as long as he wanted, and then to fill a questionnaire, the full questionnaire is in appendix B, where GEQ was used, alongside with some other questions relating to his video game playing habits, and the specific genre of the testbed game.

In the first part of the test, two versions of the game were used:

- One where the tag's performance values were being updated throughout the test, with the model described in chapter 4, *with model* will be the designation for this version throughout the rest of the section;
- And another where even though the challenges were picked using the cycle described in section 4.5 as the previous version, the performance for the tags of the obstacles that constitute a challenge are never updated, these values, calculated in the preliminary tests, are the same throughout the game. *Without model* will be the designation for this version throughout the rest of the section.

During the test there was no intervention or observation by the people developing this work unless a tester asked for it.

5.2.3 Demographic Results

These tests were made with 40 testers, 20 for each version of the game. To describe this population, questions where asked to each tester after playing the game, and logs where being automatically saved during the play session.

In fig. 5.1 it is shown how each tester allocates his time to playing video games in his everyday life, in fig. 5.2 data about the familiarity with the testbed's game genre is shown, if the testers participated in the preliminary tests fig. 5.3, how many testers finished the game can be seen in fig. 5.4, and how many testers finished the game in their first try in fig. 5.5.

The sample of testers used in each version is relatively homogeneous, both in time allocation to play video games and the game's genre familiarity. The ratio of testers that were able to complete the game is very similar as well.

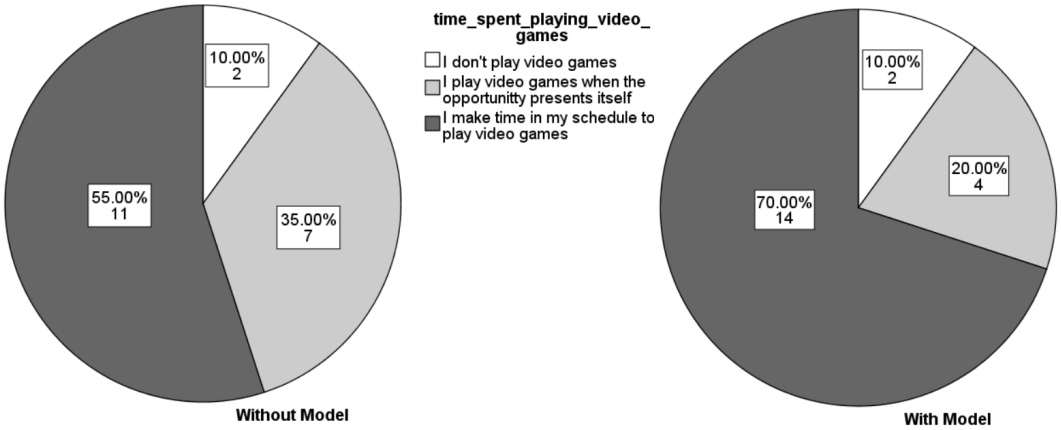


Figure 5.1: Testers' allocation of time to playing video games.

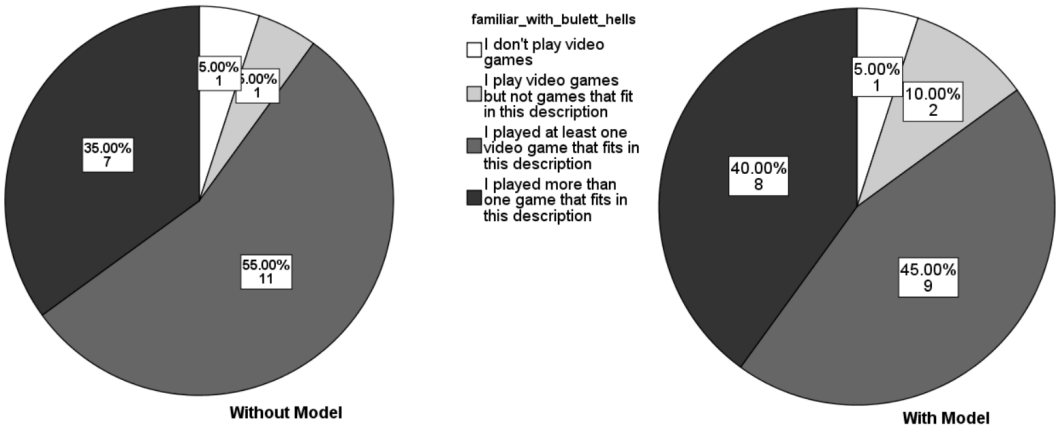


Figure 5.2: Testers' familiarity with the testbed's game genre.

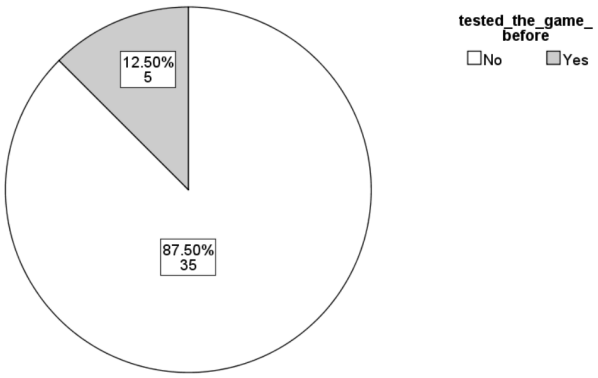


Figure 5.3: Ratio of tester's that participated in the preliminary tests.

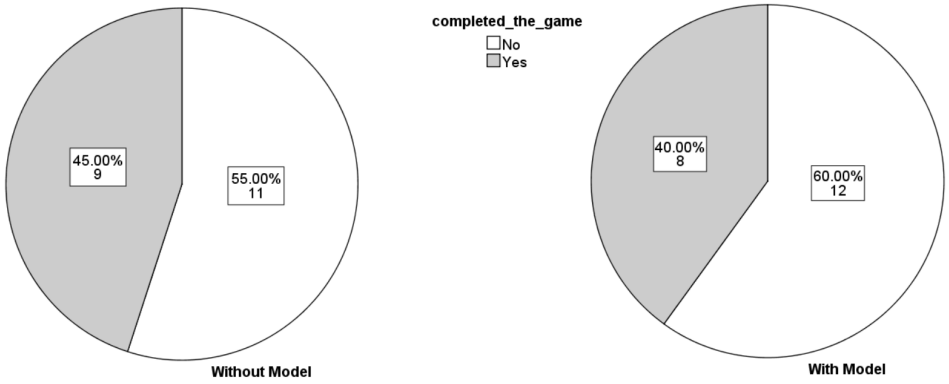


Figure 5.4: Ratio of testers that completed the game.

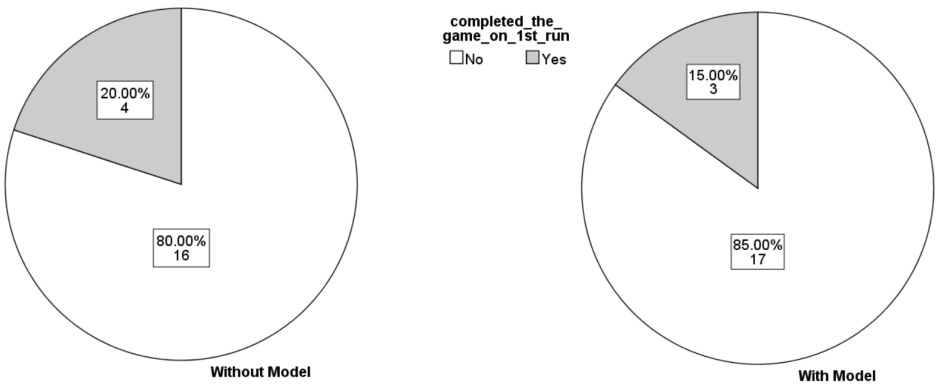


Figure 5.5: Ratio of testers that complete the game on the first try.

5.2.4 Results

Firstly each of the 7 dimensions of GEQ were put through the non parametric Mann-Whitney U Test, since each model only had 20 participants, to investigate if any of the dimensions had statistical significance.

The table containing this data can be seen in fig. 5.6, where the dependent variables are the 7 dimensions of GEQ and the independent variable is the version of the game (with or without the model).

Test Statistics[a]

	Competence	Sensory_and _Imaginative_ Immersion	Flow	Positive_Affect	Challenge	Negative_Affect	Tension_Annoyance
Mann-Whitney U	168.500	135.500	154.500	194.500	129.000	195.500	155.500
Wilcoxon W	378.500	345.500	364.500	404.500	339.000	405.500	365.500
Z	-.857	-1.752	-1.236	-.151	-1.935	-.125	-1.255
Asymp. Sig. (2-tailed)	.391	.080	.216	.880	.053	.900	.210
Exact Sig. [2*(1-tailed Sig.)]	.398[b]	.081[b]	.221[b]	.883[b]	.056[b]	.904[b]	.231[b]

a. Grouping Variable: game_version

b. Not corrected for ties.

Figure 5.6: Man-Whitney U test for GEQ's dimensions for the whole dataset.

For this test, a dimension has statistical significance if the row *Asymp. Sig. (2-tailed)* in fig. 5.6 is smaller than 0.05.

As we can see, no dimension satisfies this criteria, but the *Challenge* dimension is close. After some analysis on the data for this dimension, that can be seen in fig. 5.8, there is a single participant, in the version of the game where the model was used (red bars), that reported a much higher level of challenge than the rest of the testers of that version, making him a possible outlier.

In fig. 5.7, we can see the effect of removing this possible outlier from the results: both *Challenge* and *Sensory and Imaginative Immersion* now have a value of *Asymp. Sig. (2-tailed)* smaller than 0.05, making these 2 dimensions significantly different with and without the model.

Test Statistics[a]

	Competence	Sensory_and _Imaginative_ Immersion	Flow	Positive_Affect	Challenge	Negative_Affect	Tension_Annoyance
Mann-Whitney U	151.500	117.500	136.500	187.000	109.500	188.000	137.000
Wilcoxon W	341.500	307.500	326.500	377.000	299.500	378.000	327.000
Z	-1.088	-2.046	-1.511	-.086	-2.280	-.058	-1.557
Asymp. Sig. (2-tailed)	.277	.041	.131	.932	.023	.954	.119
Exact Sig. [2*(1-tailed Sig.)]	.283[b]	.041[b]	.134[b]	.945[b]	.022[b]	.967[b]	.141[b]

a. Grouping Variable: game_version

b. Not corrected for ties.

Figure 5.7: Man-Whitney U test for GEQ's dimensions without the possible outlier.

If analyzing the graph for the *Challenge* dimension without the outlier (red bar where challenge =

19), fig. 5.8, it is noticeable that the version of the game with the model cuts off the higher values of challenge, whereas the version without the model apparently felt more challenging.

As seen before in this work, to give the most engaging possible challenges to the player, those challenges shouldn't be too hard, in this case, the tester shouldn't feel like the challenge is too high, or too easy, characterized by low challenge values. It appears that the model was able to remove the hardest challenges.

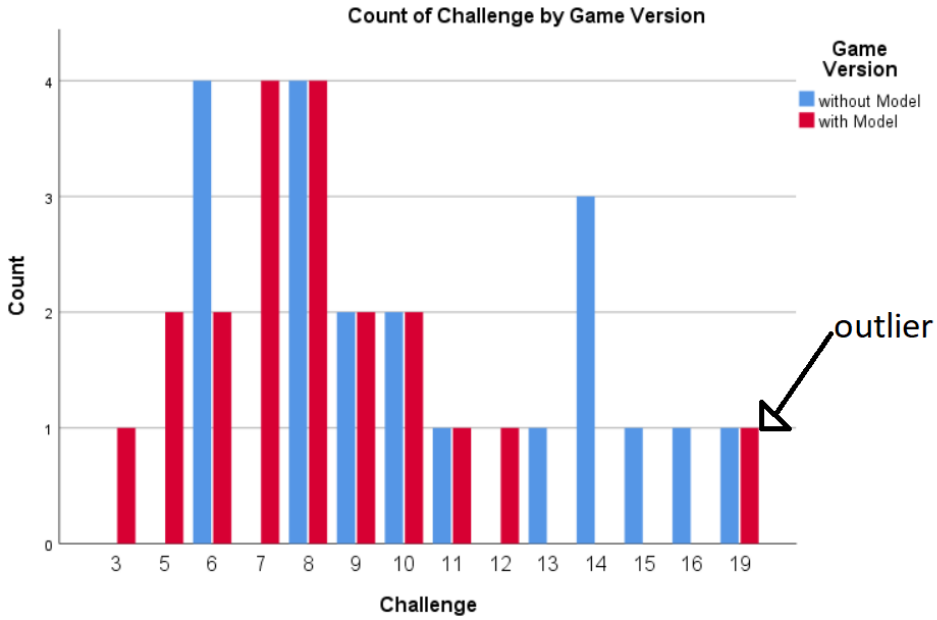


Figure 5.8: Count of testers per challenge grade.

In the *Challenge* dimension, when looking at the questions that compose it, fig. 5.9, in the "Test Statistics" table, it is possible to see that question 32 ("*I felt time pressure*") and question 33 ("*I had to put a lot of effort into it*") are the ones with statistical significant results, where the table "Ranks" helps us see that for both questions, the values are lower ("Mean Rank") for the version with the model, indicating that the testers of this version did not feel like they had to put as much effort in overcoming the challenges throughout the game as the testers of the version without the model, being safe to assume that it was a smoother experience, where they felt that the difficulty was not increasing greatly from challenge to challenge, and that they didn't feel as time pressured as testers that played the game version without the model, even though *Competence* was not affected.

Ranks

	Game Version	N	Mean Rank	Sum of Ranks
Q11	without Model	20	22.38	447.50
	with Model	19	17.50	332.50
	Total	39		
Q23	without Model	20	22.10	442.00
	with Model	19	17.79	338.00
	Total	39		
Q26	without Model	20	21.80	436.00
	with Model	19	18.11	344.00
	Total	39		
Q32	without Model	20	23.00	460.00
	with Model	19	16.84	320.00
	Total	39		
Q33	without Model	20	23.35	467.00
	with Model	19	16.47	313.00
	Total	39		

Test Statistics[a]

	Q11	Q23	Q26	Q32	Q33
Mann-Whitney U	142.500	148.000	154.000	130.000	123.000
Wilcoxon W	332.500	338.000	344.000	320.000	313.000
Z	-1.405	-1.230	-1.095	-2.199	-2.037
Asymp. Sig. (2-tailed)	.160	.219	.274	.028	.042
Exact Sig. [2*(1-tailed Sig.)]	.184[b]	.247[b]	.322[b]	.095[b]	.061[b]

a. Grouping Variable: Game Version

b. Not corrected for ties.

Figure 5.9: Man-Whitney U test for the questions in the *Challenge* dimension, where Q11 = ”I thought it was hard”, Q23 = ”I felt pressured”, Q26 = ”I felt challenged”, Q32 = ”I felt time pressure” and Q33 = ”I had to put a lot of effort into it”.

For the *Sensory and Imaginative Immersion* dimension, the question that had statistical significance, fig. 5.10, was question 19 of GEQ, (”*I felt that I could explore things*”), we can see in the ”Ranks” table that testers that played the game with the model felt they could not explore as much as the other testers, which seems to indicate that even thought the challenges were more adapted to each tester, that adaptation came at the cost of their sense of exploration, which can be explained by the heuristic used.

Ranks

	Game Version	N	Mean Rank	Sum of Ranks
Q3	without Model	20	21.35	427.00
	with Model	19	18.58	353.00
	Total	39		
Q12	without Model	20	21.53	430.50
	with Model	19	18.39	349.50
	Total	39		
Q18	without Model	20	22.63	452.50
	with Model	19	17.24	327.50
	Total	39		
Q19	without Model	20	24.58	491.50
	with Model	19	15.18	288.50
	Total	39		
Q27	without Model	20	21.33	426.50
	with Model	19	18.61	353.50
	Total	39		
Q30	without Model	20	21.15	423.00
	with Model	19	18.79	357.00
	Total	39		

Test Statistics[a]

	Q3	Q12	Q18	Q19	Q27	Q30
Mann-Whitney U	163.000	159.500	137.500	98.500	163.500	167.000
Wilcoxon W	353.000	349.500	327.500	288.500	353.500	357.000
Z	-.801	-.933	-1.521	-2.654	-.828	-.699
Asymp. Sig. (2-tailed)	.423	.351	.128	.008	.408	.484
Exact Sig. [2*(1-tailed Sig.)]	.461[b]	.396[b]	.141[b]	.009[b]	.461[b]	.531[b]

Figure 5.10: Man-Whitney U test for the questions in the *Sensory and Imaginative Immersion* dimension, where Q3 = ”I was interested in the game’s story”, Q12 = ”It was aesthetically pleasing”, Q18 = ”I felt imaginative”, Q19 = ”I felt that I could explore things”, Q27 = ”I found it impressive”, Q30 = ”It felt like a rich experience” .

For the hypothesis that people would play for longer periods of time the version with the model, the table ”Test Statistics” in fig. 5.11 shows that there was no statistical significance in the data collected.

Time played by itself, might not be a good comparison of a better experience on a game session, so to further this part of the study, other data was collected during the tests, to help measure this: the *number of runs* each tester did, as the more runs a player does might indicate a bigger interest in the game.

Ranks

	Game Version	N	Mean Rank	Sum of Ranks
time_played_secs	without Model	20	20.28	405.50
	with Model	19	19.71	374.50
	Total	39		

Test Statistics[a]

	time_played_secs
Mann-Whitney U	184.500
Wilcoxon W	374.500
Z	-.155
Asymp. Sig. (2-tailed)	.877
Exact Sig. [2*(1-tailed Sig.)]	.879[b]

a. Grouping Variable: Game Version

b. Not corrected for ties.

Figure 5.11: Man-Whitney U test for *time played in secs*.

A graph showing the number of runs each tester did can be seen in fig. 5.12. Additionally, it is not only important to know if the testers played more runs, but how far they went on each run, since the number of runs alone might not inform about the quality of a run.

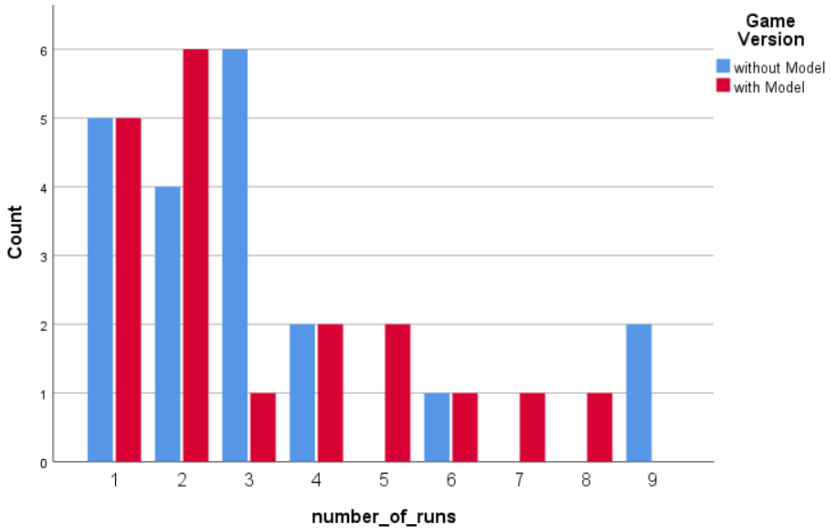


Figure 5.12: Tester count per number of runs made.

For this, the information about how many challenges were overcame (rooms cleared) by run was recorded during the test, and after analyzing if they were significantly different from one version to the other, fig. 5.13, the conclusion is they were not, meaning that the *number of runs* alone could be a good indicator if the testers played one version more than the other. Moreover, this result implies that testers were not necessarily showing a greater competence on any version, which is supported by the questionnaire responses for questions relating to the *Competence* dimension.

Ranks

	Game Version	N	Mean Rank	Sum of Ranks
total_number_of_rooms_cleared	without Model	20	19.78	395.50
	with Model	19	20.24	384.50
	Total	39		

Test Statistics[a]

	total_number_of_rooms_cleared
Mann-Whitney U	185.500
Wilcoxon W	395.500
Z	-.127
Asymp. Sig. (2-tailed)	.899
Exact Sig. [2*(1-tailed Sig.)]	.901[b]

a. Grouping Variable: Game Version

b. Not corrected for ties.

Figure 5.13: Man-Whitney U test for *number of rooms cleared*.

5.2.5 Discussion

In general, after finishing the questionnaire the testers were curious to know what the purpose of the test was. Some of them had a hunch it had to do with how enemies (challenges) were being selected, and were curious to know more about the model that lead to some interesting discussions, for example relating to the heuristic used to calculate the player's performance.

Both testers that played the version with the model and those who played the version without it felt like the challenges were being different enough from one run to the other.

It is possible to see that 11 out of the 40 testers only played one run, which might have impacted the results of this test, especially in the version with the model that needed time to get enough data about the player performance to output the best challenges. Out of these 11 testers 6 of them were playing the version of the testbed game with the model and of those 6, 3 finished the game, as seen in fig. 5.5.

Only 1 of the 17 players that finished the game replayed it after completing the objective of the game.

After analyzing the *number of runs* of each version, fig. 5.14, it is safe to say that this variable was not affected by the usage of the model.

From fig. 5.8, we can see a concentration of testers in the lower part of the grades, indicating that most people felt the challenge was easier rather than harder. If this was to be changed, one simply had to modify the *intended performance function* in section 4.6.4, to a function that would demand a higher performance of the player in each challenge, and perhaps make a version of the model with that more demanding function to compare the results with these ones to see how different the results would be in

Ranks

	Game Version	N	Mean Rank	Sum of Ranks
number_of_runs	without Model	20	20.20	404.00
	with Model	19	19.79	376.00
	Total	39		

Test Statistics[a]

	number_of_runs
Mann-Whitney U	186.000
Wilcoxon W	376.000
Z	-.115
Asymp. Sig. (2-tailed)	.909
Exact Sig. [2*(1-tailed Sig.)]	.923[b]

a. Grouping Variable: Game Version

b. Not corrected for ties.

Figure 5.14: Man-Whitney U test for *number of runs*.

that case.

Around the 15 testers mark of this evaluation phase, it was clear that asking testers to play for as long as they wanted without establishing a minimum amount of time might have been a mistake that might have had influence on the final results of this work. Part of this problem was that when a tester finished the game he would not replay it. Even thought part of the hypothesis was that players would play for longer periods of time, this could have been tested even if there was a minimum amount of time to be played.

5.2.6 Summary

Differently to Catarino's model, the two versions used in the final evaluation of this work, to choose the best challenge, were based on the same algorithm, minus the steps of updating the player's performance on one of them, so the results of the final evaluation showed the impact of adapting the challenges vs not adapting, using the same model to pick the best solution, where the time played and how many times the game was replayed were not significantly different, but the challenge felt by the testers was more homogeneous in the adaptive version, which supports the initial idea that the same level of difficulty can represent different challenges for different people, which is what this model should accomplish, and even thought the challenge felt was lower for the adaptive version, the competence dimension did not show significant differences.

Another important conclusion came from the fact that testers that played the adaptive version felt they could explore significantly less than the other testers, which might indicate that the intended difficulty function should have highs and lows, more of a sinusoidal form, rather than the straight line that was used, to let the player have more relaxed challenges where he feels like he can explore new tactics and behaviours in a less challenging environment.

Testers that only played for a few rooms might not have explored the game mechanics well enough to get a good grasp of how to play. This was a problem with the way the game was made, there was no tutorial/practice room where the tester could fight enemies in a more relaxed scenario. Even thought they could explore move, throwing the weapon and aiming in this room without any 'penalty', a comment that some players did, was that in the first room bullets should not be spent so that they could get a better feel for how the weapon shot without immediately loosing bullets.

6

Conclusion

Contents

6.1 Conclusions

In this document a possible approach to adapting the challenges of a game to the player is presented. Keeping in mind that different players have different skill levels, this work focused on how to create a model that would analyze performance while the player is playing and use that data to pick what the model believes to be the best possible challenge, within a random population, comparing it to functions created by the game designer on how the performance of the player should evolve throughout the game, as well as the novelty of the content.

A game was created from scratch to serve as the testbed game for the model, *Holiday Knight*, a *Bullet Hell/ Shoot'em Up* game, where progress is made by killing all enemies in a room to go to the next and the model described in this document was used to choose what enemies to spawn in each room.

By analysing the performance of the player against said enemies, and using it on the decision of what the next challenge should be, the hypothesis was that players would play for longer periods of time and/or replay the game more. Since this idea had already been tested by Catarino [1], this work was based on his, but applied to a different type of game, and the final evaluation of this work had the additional purpose of understanding how and what dimensions of the experience were affected by the fact that the challenge was adapted to the player.

For this, in the final evaluation of this work, two versions using the same model to generate the challenges of each room were used, where in one of them the player's performance was dynamically updated accordingly to the performance he was showing against each enemy, and on the other version these values were static.

The final results showed no significant differences in the duration of a play session nor on the amount of times the game was played. This seems to indicate that the difference in Catarino's work of comparing two completely different ways of creating challenges made his adaptive version more engaging, even thought the duration of a run in his game was shorter, around 90 seconds, while in the case of this work, each run could take from a few seconds to around 12 minutes, depending on the skill of the player, but when it comes to testing just using the model with adaptation of performance values vs static, the difference is not significant.

Other results showed that the population that played the adaptive version felt a more homogeneous level of challenge than those with the static version, with no loss of competence. Additionally the population that played the adaptive version felt that they could not explore as much as the static version.

On a game designer note, this model was helpful in empirically setting a difficulty for each enemy created without the need of complex formulas. The enemies were created and their variables tweaked to a level where they didn't feel unfair or boring, and on the preliminary evaluation, done with 6 users, all of the encounter's performance values were being recorded, and after doing some data manipulation, difficulty values were found for each enemy. This way, the game designer could just focus on creating

the enemies and the model took care of finding a difficulty value suited for each enemy.

Another important consideration is that with this model there was no need to create individual challenges by hand, all of it was automatized, the model was fed all the information needed about how the progression should be made, performance and novelty wise, and it picked the best challenges based on that. The game had 20 rooms, but if it had 100 the game designer would have no extra work, he just needed to guarantee that the intended performance and novelty functions would accomodate those many rooms, and that they would still be interesting.

6.2 Future Work

To conclude this document, here are a few topics that can be interesting additions/modifications to the work described:

1. Use a solution distribution that will ensure the biggest possible coverage of the solution space: In the phase where the pool of possible challenges is created, there is no distribution being enforced on the randomizer, which might leave areas of the solution space unexplored where the best solutions can be;

2. Instead of using random generation for the possible challenges, create solutions based on the intended values: In this work a random population of possible challenges is being generated and then each of this solutions is compared to intended values of novelty and performance to pick the best solution among that population, but since this intended values are known, it could be interesting to have a model that would create the solutions based on this information, for instance, using genetic algorithms;

3. Find better heuristics: a study on how different heuristics could affect the experience of the player could be interesting;

4. Using different intended functions throughout the session: in this work the intended performance and novelty functions are the same throughout the entire session. It can be interesting to test how using multiple functions affect the experience of the player;

Bibliography

[1] J. Catarino and C. Martinho, "Procedural Progression Model for Smash Time," in *Proceedings of the 1st IEEE International Conference on Games*. IEEE Press, 2019.

[2] M. Csikszentmihalyi, *Flow and the Foundations of Positive Psychology: The Collected Works of Mihaly Csikszentmihalyi*. Springer, 2014.

[3] "Left 4 Dead," Valve South, Microsft Windows, 2008.

[4] "Resident Evil 4," Capcom, GameCube, 2005.

[5] "Rocksmith 2014," Ubisoft - San Francisco, Microsft Windows, 2013.

[6] N. Summers, "Resident Evil 4 secretly adjusted its difficulty for you," 2015. [Online]. Available: https://www.engadget.com/2015/06/03/resident-evil-4-difficulty/

[7] M. Booth, "The AI systems of Left 4 Dead," in *Artificial Intelligence and Interactive Digital Entertainment Conference at Stanford*, 2009.

[8] "Grand Theft Auto V," Rockstar North, Playstation 3, 2013.

[9] "Smash Time!" Bica Studios, Android, iOS, 2016.

[10] S. Baron, "Cognitive Flow: The Psychology of Great Game Design," 2012. [Online]. Available: https://www.gamasutra.com/view/feature/166972/cognitiveflowthepsychologyof.php

[11] J. Nakamura and M. Csikszentmihalyi, *The Concept of Flow*. Dordrecht: Springer Netherlands, 2014, pp. 239–263.

[12] J. Togelius, N. Shaker, and M. J. Nelson, "Introduction," in *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*, N. Shaker, J. Togelius, and M. J. Nelson, Eds. Springer, 2016, ch. 1.

[13] ——, "Introduction," in *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*, N. Shaker, J. Togelius, and M. J. Nelson, Eds. Springer, 2016, ch. 1.1.

[14] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, “Search-based Procedural Content Generation,” in *European Conference on the Applications of Evolutionary Computation*. Springer, 2010, pp. 141–150.

[15] “The Binding of Isaac,” Nikalis Inc., Microsft Windows, 2011.

[16] E. McMillen, “The Binding of Isaac Gameplay explained,” 2011. [Online]. Available: http://edmundmcmillen.blogspot.pt/2011/09/binding-of-isaac-gameplay-explained.html

[17] “Minecraft,” Mojang, Microsft Windows, 2011.

[18] N. Shaker, G. Yannakakis, and J. Togelius, “Towards automatic personalized content generation for platform games,” in *Sixth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2010.

[19] M. Persson, “Infinite Mario Bros,” *Online Game. Last Accessed: December*, vol. 11, 2008.

[20] G. N. Yannakakis and J. Togelius, “Experience-Driven Procedural Content Generation,” *IEEE Transactions on Affective Computing*, vol. 2, no. 3, pp. 147–161, 2011.

[21] E. J. Hastings, R. K. Guha, and K. O. Stanley, “Evolving content in the Galactic Arms Race Video Game,” in *2009 IEEE Symposium on Computational Intelligence and Games*. IEEE, 2009, pp. 241–248.

[22] C. Browne and F. Maire, “Evolutionary Game Design,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 1, pp. 1–16, 2010.

[23] P. Lucas and C. Martinho, “Stay Awhile and Listen to 3Buddy, a Co-creative Level Design Support Tool,” in *ICCC*, 2017.

[24] “Legend of Grimrock II,” Almost Human Games, Microsft Windows, 2014.

[25] G. Smith, J. Whitehead, and M. Mateas, “Tanagra: A mixed-initiative level design tool,” in *Proceedings of the Fifth International Conference on the Foundations of Digital Games*. ACM, 2010, pp. 209–216.

[26] J. Juul, *A Casual Revolution: Reinventing Video Fames and their Players*. MIT press, 2010.

[27] T. Taylor, *Exploring Online Game Culture*. Cambridge, Mass: MIT Press, 2006.

[28] C. Bateman and R. Boon, *21st Century Game Design (Game Development Series)*. Charles River Media, Inc., 2005.

[29] J. Catarino and C. Martinho, *Procedural Progression Model for Smash Time*, 2018, ch. 4.6, 4.7.

[30] “Dark Souls,” From Software, Playstation 3, 2009.

[31] “Enter the Gungeon,” Dodge Roll, Microsft Windows, 2016.

[32] W. IJsselsteijn, Y. De Kort, and K. Poels, “The Game Experience Questionnaire,” *Eindhoven: Technische Universiteit Eindhoven*, 2013.

A

Scoring Guideline for Game Experience Questionnaire

Competence:

- I felt skillful
- 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” [32].

B

Post Game Questionnaire with GEQ

Game Experience Questionnaire

Thank you so much for playing Holiday Knight and helping me with my thesis!

Before you go please answer these questions about your experience while playing the game.
It is important that you answer honestly.

*Required

1. **Did you choose Cake or Ice Cream? If you don't remember you can check it by pressing "ESC" on the dungeon.** *
   Mark only one oval.
   - Cake
   - Ice Cream

2. **What is your assigned number in the game? You can check this number by pressing "ESC" on the dungeon (If you are testing remotely answer 99).** *

   _____

3. **Have you tested this game in the previous 2 weeks?** *
   Mark only one oval.
   - Yes
   - No

4. **How often do you play video games?** *
   Mark only one oval.
   - I don't play video games.
   - I play video games when the opportunity presents itself.
   - I make some time in my schedule to play video games.

5. **How familiar are you with games in which the protagonist combats a large number of enemies by shooting at them while dodging their fire, like Enter the Gungeon, Binding of Isaac, Ikaruga, Cuphead, Nuclear Throne?** *
   Mark only one oval.
   - I don't play video games.
   - I play video games but not games that fit in this description.
   - I have played at least one game that fits in this description.
   - I have played more than one game that fits this description.

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

Mark only one oval per row.

	not at all	slightly	moderately	fairly	extremely
I felt content					
I felt skilful					
I was interested in the game's story					
I thought it was fun					
I was fully occupied with the game					
I felt happy					
It gave me a bad mood					
I thought about other things					
I found it tiresome					
I felt competent					
I thought it was hard					
It was aesthetically pleasing					

7. *

Mark only one oval per row.

	not at all	slightly	moderately	fairly	extremely
I forgot everything around me					
I felt good					
I was good at it					
I felt bored					
I felt successful					
I felt imaginative					
I felt that I could explore things					
I enjoyed it					
I was fast at reaching the game's targets					
I felt annoyed					

8. *

Mark only one oval per row.

	not at all	slightly	moderately	fairly	extremely
I felt pressured					
I felt irritable					
I lost track of time					
I felt challenged					
I found it impressive					
I was deeply concentrated in the game					
I felt frustrated					
It felt like a rich experience					
I lost connection with the outside world					
I felt time pressure					
I had to put a lot of effort into it					