Personalized adaptive dentist game

Bachelor Thesis

B. Evers
O. Huang
R.A.N. Starre

Delft University of Technology

TU Delft Delft University of Technology

Challenge the future

PERSONALIZED ADAPTIVE DENTIST GAME

BACHELOR THESIS

by

Björn Evers
Owen Huang
Rolf Starre

in partial fulfillment of the requirements for the degree of

Bachelor of Science
in Computer Science

at the Delft University of Technology,
to be defended publicly on Friday July 1, 2016 at 09:30.

Supervisor:	Dr. ir. R. Bidarra
Client:	Dr. ir. R.E. Kooij
Thesis committee:	Dr. ir. R. Bidarra
	Dr. ir. R.E. Kooij
	Dr. ir. M.A. Larson

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft Delft University of Technology

PREFACE

This thesis concludes the Bachelor project as part of the Bachelor of Science programme Computer Science at the Delft University of Technology. During the project students are required to demonstrate their ability to successfully carry out a project commissioned by a client.

Over the past ten weeks, we have carried out the Personalized Adaptive Dentist Game project in assignment of the Delft University of Technology. We have chosen this project, because the novelty in Procedural Content Generation (PCG) and the application of these concepts to a serious game appealed to us. In the span of these ten weeks, we have designed and tested a dentist game using PCG approaches building upon a previous prototype of the game.

Furthermore, this thesis aims to inform the reader of the design process of the project up to the completed work and concludes by giving recommendations concerning future work on this project.

We would like to thank everyone who helped us in realizing this project. In particular, we thank our supervisor Rafael Bidarra who has helped us tremendously during the design and development process and for the valuable insights in making this project a success. We would also like to thank our client Rob Kooij for making it possible to carry out this project. We thank the Bachelor project coordinators Felienne Hermans, Martha Larson and Otto Visser for the organisation of the Bachelor project and for always being available to answer our questions about the organisation of the project. Furthermore, we thank Dylan Nagel for the interesting discussions and the feedback for the design of the game. We also want to thank Sander Roeleveld and the dental practice Mondhygiënisten Sweelinckplein for the opportunity to test our game with their patients. We thank Marloes Adank for the opportunity to present our game at the International Festival of Technology. Finally, we want to thank everyone who participated in testing the game, which helped making this project a success.

Björn Evers
Owen Huang
Rolf Starre
Delft, June 2016

SUMMARY

The goal of this project was to develop a personalized adaptive game for use at the dentist or other dental treatment settings. Before the project started, a prototype game utilising the Unity game engine was already available. The game's purpose is to make dental treatment more comfortable by playing it during the treatment. The game screen is viewed through VR goggles that the patient has to wear during the treatment.

The prototype game was 'static' and lacking in content. There were only two levels and always the same. As a result, the game did not suit different types of players well. Additionally, players would lose interest in the game when playing longer periods of time, because the game had little content.

Our goal for this project was to transform the existing prototype game into a personalized, adaptive version of the game. The game experience should vary on the type of player who is playing the game, e.g. with respect to the difficulty. Furthermore, the game should be playable by a very broad audience: from non-gamers to hardcore-gamers, and from young children to the elderly.

To achieve this, we developed a game prototype that uses gameplay characteristics of the player to construct a model representing the player using a set of parameters. This player model is used to construct level generation parameters, which in turn are used by a level generator to generate an appropriate level.

To determine what techniques were required to build the game, we conducted research on serious gaming and investigated several methods for player modelling and PCG in games. We found that it is important to keep a player engaged during a serious game in order for the game to have an optimal effect. We also discovered that using an event based player model construction method was the best option for this game. Finally, we chose to use a chunk based level generation mechanism to keep the level generation both varied and easy to control.

The main objective of the game remains setting a high score by avoiding the obstacles in the cave and collecting coins. The game also features an oxygen meter which the player should keep filled by collecting oxygen bubbles.

The adaptation of the game to the player is based on game events. These events are processed by a second module which generates player parameters representing the skill of the player based on the logged events. The player parameters are used by another module to generate level generation parameters. Finally, the level generation parameters are used by the level generator to generate level sections that are adapted to the players up-to-date skill level.

As a result, separating all the aforementioned modules allows for easy extendibility and ease of maintenance of the system. It also simplifies the comprehensibility of the system's concepts.

Levels are generated using many different level chunks which are placed one after another to create an endless level. Placement of the different level elements is done based on the player parameters and is also randomized to ensure that the player navigates a unique cave during each game session.

The game has been tested twice. The first time, at a dental practice with patients of a dental hygienist. The patients found that the game was indeed distracting them from the treatment. Because of a loss of eye contact with the patients (due to the VR goggles), the hygienist suggested that a different method of communication should be in place for emergencies e.g. a panic button the patient could press.

The second time, the game was tested at the International Festival of Technology (IFoT) in Delft. Around 40 people played the game and most indicated that they enjoyed the game. However, people were divided on whether the game was challenging.

Scrum was followed as development methodology. Sprints lasted one week, but in hindsight two weeks would have been more suitable because of the overhead one week sprints created.

We can conclude that the project has succeeded; a personalized adaptive dentist game has been developed. We designed a player model and used it to generate suitable level generation parameters. Also, a controllable level generator was created to facilitate the adaptive experience. The system structure is flexible and allows easy addition, deletion, or modification of system components.

It has to be stated that the game is still a prototype and is not fully ready for the market. Additional testing is required and the applied adaptation techniques can be extended and fine tuned to create a better experience. Finally, more gameplay elements could be added to make the game even more interesting for longer periods of time.

Contents

List of Tables

List of Figures

1

INTRODUCTION

Fear of the dentist is an issue for large groups of patients. The Delft University of Technology has developed a prototype game to tackle the issue of discomforting and distressing experiences at the dentist by means of distraction through an immersive and interactive game [1]. However, the prototype game was not suitable for long dental treatments due to a lack of content. Furthermore, the prototype would not be suitable for every patient's needs. Some patients may enjoy a more challenging game because they are more skilled, while other patients may struggle with the very basics. This may be detrimental to the immersion of the game which results in a loss of effectiveness in distraction, because the game is experienced to be too easy or difficult [3].

This project addresses these issues by designing a player model which captures relevant player data that will steer the elements of the game through a PCG approach. This will improve the enjoyability of the game for more patients and in turn the effectiveness of distracting patients, because the game creates a personalized experience.

This thesis outlines the process of research to evaluation, starting with a concrete description of the problem and an analysis in Chapter 2. This chapter also includes the requirements that needed to be met in order to make this project a success. In Chapter 3, proposed solutions to the problem are more thoroughly explored, which contributed to the design of the game. This design is fully elaborated upon in Chapter 4. In Chapter 5, the final product of the project is presented. In Chapter 6, the implementation of all the components of the game and the way the player model and level generator communicate with each other are discussed. In Chapter 7, a description of the various methods of testing that were conducted is given and the results of the field tests are analysed. Chapter 8 describes the organisation of the project and evaluates the overall process during the different stages of the project. Finally, we conclude with the conclusions and recommendations for future work in Chapters 9 and 10.

2

PROBLEM DEFINITION & ANALYSIS

The problem at hand needs to be concretely defined and thoroughly investigated. In Section 2.1 the definition of the problem is given and Section 2.2 proceeds to analyze it elaborately by reflecting on the old prototype of the game. In Section 2.3, the requirements for this project are defined. These requirements project what criteria have to be met in order to bring success to the project using a MoSCoW approach[1].

2.1. PROBLEM DEFINITION

Fear of visiting the dentist is a prevalent issue in large groups of patients. This fear commonly originates from previous unpleasant and discomforting experiences at the dentist [1]. As a result, specialized dental anxiety clinics aim to assist patients who abhor a trip to the dentist. One way to put a patient at ease during treatment is by distracting the patient through an interactive game [1]. The aforementioned game has already shown to achieve positive reactions among patients, however it is still in its early stages for actual deployment in a dental clinic.

Every patient is different and therefore has different needs. Some patients seek a bigger challenge in the game, while others may find the game too difficult. To account for the broad spectrum of needs, the game requires a level of personalization and the ability to adapt to these needs. The prototype of the game has only two levels (an easy one and a hard one) and is unsuitable for application in long treatments (up to an hour). Similarly, the prototype is lacking a personalized experience.

2.2. PROBLEM ANALYSIS

During the treatment the dentist should be able to maintain contact with the patient at all times, e.g. to ask questions or request movements. In addition, the game should not require a lot of time to set-up since this could interfere with the schedule of the dentist.

To keep the patient distracted during the entire treatment, sufficient gameplay content has to be provided. However, for long treatments simply providing more content does not necessarily make it suitable. While there is a lack of content for long treatments, short treatments also require attention. The game has to address both a short and long game, while creating a satisfying ending to the game right when the treatment is about to finish.

Keeping the patient distracted also includes keeping them calm. Therefore, given the context of a dental setting the content also has to adhere some defining characteristics to be effective. These characteristics include *relaxation* of the player and *simplicity* of controls [1].

Achieving the personalization of the game plays a big role in the effectiveness of the game in distracting the patient. But to create a personalized experience the player's preference and needs have to be defined first. As a result, one of the challenges of this project is modelling the player. This model requires capturing relevant player data, such as controller actions, gameplay specific objects, but also context dependent information (e.g. time). Another challenge is providing the content to address the aforementioned lack of it. Procedural content generation is a technique that can help solve this, but it has to be adapted to the different needs of patients to remain appealing. In other words, content should be generated depending on the player model.

[1] The MoSCoW approach categorizes requirements in four sections ('Must, Could, Should, Won't have') each being in descending order of importance.

This project focuses on achieving personalization of the game to account for every patient through capturing relevant player data to construct a personalized player model. As a result, another goal of the project is to increase the overall effectiveness of the game in distracting the patient. Content is procedurally generated utilizing this player model, while addressing the issue of the lack of content for longer treatment.

2.3. REQUIREMENTS

The requirements for this project are listed in Tables 2.1 and 2.2 in a prioritizing manner by definition of the MoSCoW approach. The functional requirements are separated from the non-functional. The 'Must have' requirements are essential to the success of the project, these have the highest priority. Following, in descending order of importance, the 'Should have' requirements are just as important as the 'Must have' requirements, but not essential for the success of the project. Next, 'Could have' requirements make a nice addition to the project given there is time left. Lastly, the 'Won't have' requirements will not be incorporated in this project as they are either not feasible within the scope of this project or regarded as an unimportant addition.

Table 2.1: Functional requirements using the MoSCoW approach.

Functional requirements
Must have
• Relevant player data is captured and managed by a player model manager.
• Levels must be procedurally generated using parameters from the constructed player model.
• Contextual time constraints must be able to influence the game play.
Should have
• A game session should have a smooth 'ending'.
• The game should be pausable.
• The game should have clear goals.
• Game elements that influence gameplay should be clearly distinguished from elements that dont influence gameplay (e.g. background).
Could have
• Leaderboard, high scores are being kept inside the leaderboard.
• Continuous feedback loop after each level. The feedback states how well you performed after each level.
• The game could allow for different styles of play.
• The player could be rewarded for correct play.
Wont have
• Save games, the game will not save individual game sessions in a save file.

Table 2.2: Non-functional requirements using the MoSCoW approach.

Non-functional requirements
• The game must be implemented in C# using Unity.
• The game must not use complex controls (no more than a few buttons).
• The game controls should be intuitive.
• The game must contain support for XBox 360 controller.
• The game should be runnable on a basic computer and the Vuzix 1200 VR goggles.
• Should be able to quickly start playing.
• The game should not contain any surprises that may scare a patient during treatment.
• The game should be a fun and calming experience.
• Game should distract from fear of dental treatment.
• Dentist must be able to communicate with the player at all times during the treatment.
• The gameplay should be tested thoroughly with the addition of user testing inside a dentist clinic.
• It is not allowed to use the headtracking of the VR goggles.

3

RESEARCH

The first two weeks of this project were dedicated to orientation and research of the proposed problem. Previously in Chapter 2 the project's scope was thoroughly defined and explored. Now, in order to fully explore the possibilities for fulfilling each proposed requirement Sections 3.1 and 3.3 shed light on the current existing techniques that can aid in solving the problem.

3.1. SERIOUS GAMING FOR DENTAL ANXIETY

In this section we will take a look at our research into serious gaming. We started by looking into the concept of serious gaming and its applications, trying to find answers to questions like how can we define serious gaming and how is it currently used. While serious games are not purely for entertainment it is still important to look into the gameplay in order to deliver a good product because you want patients want to play the game. Therefore we also did some research into ways to make games fun and engaging, which can be combined with the PCG and player modelling which we will discuss in other sections. Looking into how distraction is caused we focused on the relation between immersion and flow and how to integrate this in a game. This has led to some research into concepts of playability, usability and testing thereof using heuristics and user tests.

3.1.1. SERIOUS GAMING

Serious gaming is becoming more accepted as a field and is becoming increasingly popular, where a quick Google-search on 'serious gaming' resulted in about 1.090.000 hits on 01-03-2007 [19] today this same search results in 34.200.000 hits (27-04-2016). While some authors only see games that focus on education or training as serious games [8], others see serious games as all games that serve purposes other than only entertainment [19].

There are many areas in which serious games have been developed. They have of course been developed in the aforementioned areas of education and training, such as military training or surgical training, as well as in areas such as physical fitness (Wii Fit Plus), diagnosing mental disorders (e.g. ADHD, PTSD) [19], and as a means of distraction (e.g. from pain [25], or from anxiety in a dental practice [1].

Since we consider the game we are developing as a serious game we adhere to the definition of a serious game as "a game that has a primary goal other than entertainment". We are mainly interested in distracting patients and reducing perceived pain during dental treatments. Previous research has already shown that it is possible to use games or Virtual Reality as a means of reducing pain and anxiety [10, 25]. It has been speculated that the immersion into these virtual worlds is what causes this distraction [1, 25], and a concept that could relate to this is 'Flow'. This concept is elaborated in 3.1.2.

3.1.2. FUN & FLOW

While entertainment is not the primary purpose of serious games in general according to our definition, it is still an important aspect in the current project. Since people go to the dentist regularly, the game should achieve a certain level of replayability and the patient should also be willing to play the game. If the patient is not interested in playing anymore, after just one visit we might not consider the product to be very successful. Therefore, it is important to incorporate gameplay design in our research.

Having a game that is entertaining is also likely to help immerse the patient into this game, which should

help with reducing perceived pain and provide a distraction from the treatment. Being in a state of Flow, “the feeling of complete and energized focus in an activity, with a high level of enjoyment and fulfillment" [3], is likely to induce a feeling immersion. We will give a short introduction into Flow and some research into what makes games fun.

According to [14], to be in a state of Flow one needs to feel like he is challenged at a level that fits his level of proficiency and there should be clear short term (fast) goals and feedback about progression of the player. To stay in this Flow zone one has to be challenged according to their abilities. With an increase in abilities the challenge should also increase, high level of skill compared to the required skill leads to boredom while an activity where the required skills far exceed the level of skill leads to anxiety [3, 14] (see Figure 3.1). According to [3] there are also inter-individual differences in this, some people need or prefer more challenge relative to their skills than others. Therefore to make a game fun for as many people as possible they propose that game design should allow the player choices that give them some control over the flow of the game without interrupting the gameplay, thereby allowing the player to stay in their Flow zone.

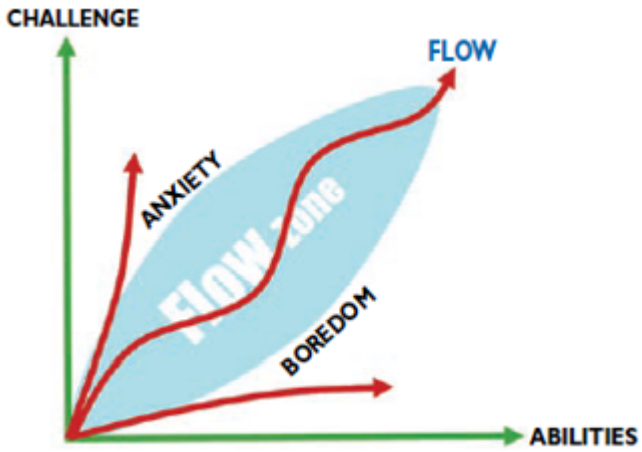


Figure 3.1: Depiction of the ‘flow’ in terms of challenge and ability [3].

According to this research not being in a state of Flow can lead to anxiety or boredom, so we can relate Flow to fun. Although there are obviously other ways to have fun other than partaking in a challenging activity, it is one of the aspects that makes a game fun [12]. [12] presents four different ways to play with the difficulty of achieving goals: by varying the difficulty level, by having multiple goals in a levels; by having hidden information, and by having elements of randomness in the game.

According to [12] there are two additional aspects: fantasy and curiosity. Fantasy has to do with living out a fantasy; e.g. being a hero that saves a princess, wielding powerful magic, or piloting a submarine in a large submerged cave system. About curiosity he says “one of the most important features of intrinsically motivating environments is the degree to which they can continue to arouse and then satisfy our curiosity”. However, not everyone may agree with these categories. According to other results curiosity was not that important for gamers and it might be immersion rather than fantasy that is stimulating to players [6]. In another study challenge was found to be one of the most important factors for a game to be a favorite game [6].

3.1.3. GAMEPLAY EVALUATION AND HEURISTICS

In order to find out if the goals of the game are reached gameplay wise, some kind of testing or evaluation has to be done. To gain insight into this subject we tried to find some research related to this area. According to [13] traditional usability testing is not enough, because they cannot be directly applied to all the aspects of games. In addition, we should also take the experience of the gameplay and for serious games the specific goal of the game other than the entertainment into account [13].

[13] uses the model in Figure 3.2 to model how the game and the game designer interact with the individual players. The game designer works on the game which shapes the game system experience which then affects the experience of the individual players. A way in which the game designer can try to directly influence this individual experience is by implementing player modelling and/or adaptive content based on player actions [13].

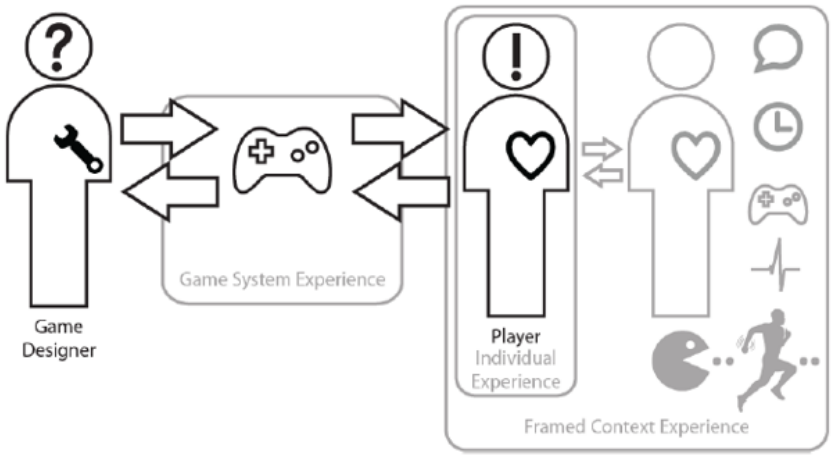


Figure 3.2: Planes of gameplay experience [13].

There are many different ways in which gameplay experience can be evaluated. An example is measuring physiological properties (e.g. heart rate, brain activation, or electrodermal activity), while players are playing the game, measuring the responses of the player in this way can give an indication of their emotion and arousal during the game. Some other ways to help evaluating player experience is by analysing what a player does during the game, by creating of model of player behaviour that can you can 'testdrive' game elements with, by using interviews and questionnaires, or by using playability heuristics [13]. Different methods of evaluation can also be combined to provide additional insight into the player experience to help model the game, for example [11] combined the use of questionnaires with analysis of gameplay metrics to see how specific complaints or praise correlated to the gameplay of the individuals.

In our project the context experience is mostly about the interaction between the dentist and the player/patient and the effect the game has on the distraction from anxiety/pain during the treatment, which can also be tested in a controlled environment. Feedback from both the dentist and the player can prove to be valuable in assessing the success of this element. One practical way to evaluate the playability of a game is by using design heuristics [13], these provide guidelines for designing the game and are very useable earlier in the design process when there is not yet enough gameplay to warrant user testing. There has been a lot of research that proposes various kinds of gameplay heuristics [4, 6, 9, 12, 13].

[9] used a model in which they identify three different important aspects for the playability (Figure 3.3); i.e. gameplay, game usability, and mobility. They included mobility because they inspect games playable on a mobile phone.

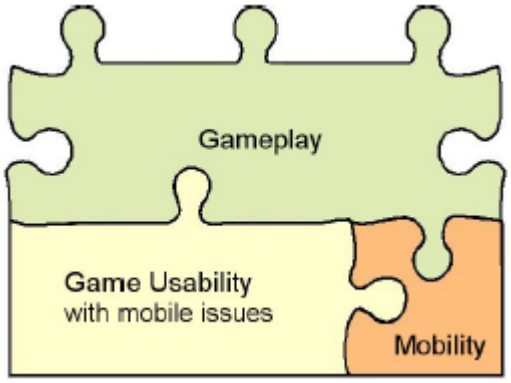


Figure 3.3: Aspects of playability in mobile games [9].

Some of the aspects of mobility also relate to our case at the dentist, we have to take into account interruptions (when the dentist requires attention of the patient or the patient needs to move) and the duration of the game should fit different lengths (some dental treatments last a lot longer than others) so the game should be easy to pick up and play. [9] first created an initial model and then used expert evaluators to im-

prove on this initial try. They included game usability as a separate module because this was also found to be very important for the gameplay experience. These heuristics can be found in Figure 3.4, Figure 3.5, and Figure 3.6.

No.	Gameplay Heuristics
GP1	The game provides clear goals or supports player-created goals
GP2	The player sees the progress in the game and can compare the results
GP3	The players are rewarded and rewards are meaningful
GP4	The player is in control
GP5	Challenge, strategy, and pace are in balance
GP6	The first-time experience is encouraging
GP7	The game story supports the gameplay and is meaningful
GP8	There are no repetitive or boring tasks
GP9	The players can express themselves
GP10	The game supports different playing styles
GP11	The game does not stagnate
GP12	The game is consistent
GP13	The game uses orthogonal unit differentiation[4]
GP 14	The player does not lose any hard-won possessions

Figure 3.4: Gameplay heuristics [9].

No.	Game Usability Heuristics
GU1	Audio-visual representation supports the game
GU2	Screen layout is efficient and visually pleasing
GU3	Device UI and game UI are used for their own purposes
GU4	Indicators are visible
GU5	The player understands the terminology
GU6	Navigation is consistent, logical, and minimalist
GU7	Control keys are consistent and follow standard conventions
GU8	Game controls are convenient and flexible
GU9	The game gives feedback on the player's actions
GU10	The player cannot make irreversible errors
GU11	The player does not have to memorize things unnecessarily
GU12	The game contains help

Figure 3.5: Usability heuristics [9].

No.	Mobility Heuristics
MO1	The game and play sessions can be started quickly
MO2	The game accommodates with the surroundings
MO3	Interruptions are handled reasonably

Figure 3.6: Mobility heuristics [9].

3.2. PLAYER MODELING

In general, a player model is a set of data about a player that can be used in a game or during game development for a certain purpose. In other words, a player model models the behavior of a player in a game in a way that is useful for improving the game in some way.

For the dentist game, we plan to use player models in order to personalize the game for specific patients at the dentist. The player models can be used as, among others, an input to the procedural content generator in order to generate levels that are ideal for the player at a specific point in time.

3.2.1. TYPES OF PLAYER MODELS

As stated, player models can be used for a lot of different purposes. Player models that will be used for different purposes are constructed differently. For example, a player model that is used for Dynamic Difficulty Adjustment (DDA) to control the difficulty of a game over the course of one play session will generally contain generalized attributes related to the player's performance in the current play session. On the other hand a player model that is used to control an Artificial Intelligence (AI) character that has to simulate the player has to collect very precise data about what the AI character should do in a lot of different situations.

One could list a lot of other kinds of player models besides the two just mentioned. With so many types of player models, confusion may arise about the type of player model that one talks about. For that purpose [18] proposes a taxonomy in which they try to include all kinds of player models and introduce a terminology for them.

They classify player models based on four categories:

Domain Can be either *Game Actions* or *Human Reactions*. Game Actions refers to actual details of the specific rule system of the game, while Human Reactions refers to the actions of a player caused by playing the game.

Purpose Can be either *Generative* or *Descriptive*. A generative model aims to replace a human in detail, while a descriptive model is of a higher level in its description.

Scope The value of this category can be either *Individual*, only applying to a single player, *Class*, applying to a part of the player group, *Universal*, applying to all players of the game, or *Hypothetical*, not applying to any specific player but to a hypothetical player or class of players.

Source The source can be either *Empirical* or *Theoretical*. An empirical player model is constructed based on behavior that is actually recorded, while a theoretical model is based on some theory or derived from the game's rules.

The player model that will be constructed for the dentist game can be classified as follows. It belongs in the game actions domain, because it will record actual actions the player records in the game. Its purpose is generative, because it has to be fairly detailed and record actual actions in order to draw conclusions about the performance of the player. The scope will be either individual or class based, because it will apply to a single player or a group of players playing the dentist game. Finally, the source of the model will be empirical, because is is constructed from actual data from a player playing the game.

3.2.2. CONSTRUCTION OF A PLAYER MODEL

Player models can be constructed in a variety of different ways. The most important and relevant methods will be discussed here.

First of all, the easiest way to construct a player model might be asking the player himself to provide data for the player model. For example, at the beginning of the game, the player could be given the option to select a difficulty for the game. Such an approach is suggested in [2], where a framework is suggested for developing player-centered games based on player models.

For the dentist game, this approach may or may not be a good idea, depending on the point of view. On the one hand we can expect the players of the dentist game to know the best how difficult they want the game to be, at least initially. On the other hand, the dentist may be able to make a better judgment about how anxious or frustrated a patient may become during the treatment.

In [27], tensor (multidimensional matrix) algebra is used to infer the performance of players in coming stages of a game, when some stages have been played. This method could be useful in the dentist game because it takes progression over time into account and can be used in real time.

Head movements and facial expressions are used in [17] to derive relations between gameplay features and emotions of players. This data can also be used for constructing player models. Because video capturing devices and advanced image processing methods are necessary for this to succeed, it is probably outside of the scope of this project, for now.

Another possibility is to use an Artificial Neural Network (ANN) to model the behavior of a player. This has been done in [20] to model the racing behavior of a real human. Several strategies for using ANNs in player modelling are also discussed in [2]. The advantage of using ANNs is that they can capture rather complicated relations in data. In our dentist game, they could be capable of processing a lot of gameplay attributes at once in order to produce a very detailed model. However, the fact that ANNs can capture complex concepts also leads to their main disadvantage: they are hard to understand for humans. Only the simplest of ANNs can be comprehended by a human.

A concept that is often mentioned to be important for a player model is progress of the player performance over time [2, 20, 27]. These studies show that performance of a player is always evolving and that at the moment of adaptation, the actual performance of the player could already be different. In [20], it is noted that a race track constructed for a specific player with a specific skill level is already outdated the second time the player races on the track. In other words, the player has already improved after the first race. In [27], the tensor that is constructed includes a time dimension to signify the performance improvement over time of a player. This can be used to predict how much a player will improve in the next timestep. This knowledge could be used in the dentist game to predict the right performance change.

The type and complexity of data is also an important aspect to keep in mind when constructing a model. In [20] they show three different granularities of data collection. In the finest granularity, they collect data about the specific action that the player performs at each game timestep. At the second finest granularity, they only collect data about specific game attributes, like the number of times the player crashed into a wall, or the average speed. In the roughest granularity, they only collect the time taken to reach each waypoint along the track.

In [11], only very basic data is collected about player performance using a limited list of attributes, like the second granularity level just described. It is useful to think about the level of granularity we want to use for data collection in the dentist game. A higher granularity potentially leads to more accurate results, but it may be more difficult to process. We need to find a good trade-off between those two factors.

3.2.3. USING A PLAYER MODEL

One can use a player model in a variety of ways. For example, one could input the data into an ANN in order to get some useful data as an output. For example, in [15], ANNs are used to discover what the strength of an emotion, like anxiety or frustration, of a player would be when certain content generation parameters would be combined with the current player model of the player. In [20], computer controlled vehicles are used to check how a real player would perform on a specific track using an ANN that has been trained on real player racing data and that gives as its output speed an steering commands. The player model can also be processed by regular programming code, like in [11]. For the dentist game, ANNs will probably not be feasible to construct. Regular programming code will most likely be used to process the player model.

There is also the question of how specific one should be when processing the player model. One could classify a player in a group based on its player model, and generate specific content for that group, or one could generate content specifically for one player. The first option is probably easier to implement in the dentist game than the second option, but the second option may lead to a more personalized game. Assigning players to groups is part of the framework for developing player-centered games in [2]. In [11], content is being generated specifically for each player. In the game, the player has to balance on a snowball that rolls to the right and the player has to dodge obstacles. If a player hits more obstacles, more obstacles are placed, but if a player at the same time manages to balance on a snowball, that part of the game would become more difficult. In other words, the difficulty is very personalized instead of being based on difficulty classes.

3.2.4. PURPOSES OF A PLAYER MODEL

In the dentist game, we could use player models for two different purposes. First, one could use player models to dynamically generate game aspects to personalize a game for a specific player, like in [27] and [15]. Second, one could use player modelling for testing purposes. For example, one could generate a player model that is used to simulate a player in a level in order to efficiently test the level. Or one could use the test player model to construct a level that is optimized for a specific player, like in [20]. For the dentist game, the most prominent use of the player model will be for PCG. Testing may be done using a model, but we do not expect

it.

One could also use the player model to predict the strength of the player's emotions, like anxiety and frustration, as has been done in [15]. This may be very relevant for our dentist game, because we would like the players of the dentist game to be in a calm emotional state. We would not like them to become anxious or frustrated.

The features that are important for a player model to have depend on what the player model is going to be used for. For example, if the player model is going to be used for online content generation (see Section 3.3.1), it should be simple enough for a computer to process the model in real-time. This will be the case for the largest part for our dentist game.

Also, some applications require the model to be more accurate than others. If we use the model for simulating a real player, like in [20], we want the behavior resulting from the model to be very similar to the real player. On the other hand, if we only want to broadly know how well a player is doing in a game, like in [27], we may settle for about 70 percent accuracy.

The amount of information that is needed for the model to be accurate enough may also be relevant. For example, if we are using the dentist game for a treatment of 5 minutes, and the game has collected enough information only after 4 minutes, there is only 1 minute of personalized play time left. In other words, we want the calibrating phase in our dentist game to be short and so we need a player model that allows us to achieve that.

3.2.5. EMPIRICAL RESEARCH

For some of the player model types, empirical research is needed to use them. For example, in [15], to construct the ANNs to predict the strength of several emotions, data from play sessions from 120 people has been used, together with answers to questions about the game from those same people. According to [15], the ANNs could not have been constructed reliably with much less data. In [17], they even used video data about head movements and facial expressions to construct ANNs for predicting the strength of the player's emotions. It is probably not feasible to collect as much data for the development of the dentist game.

As the amount of time and number of opportunities to collect data for the dentist game will probably be limited, it would probably be better to choose player modelling methods that do not require as much data collection. In [11], for example, not as much data has been collected, but the result was still quite good.

3.3. PROCEDURAL CONTENT GENERATION IN GAMES

PCG has applications within many fields and is becoming more popular in the field of games. Creating content in games is usually done manually, which takes a lot of time resulting in dedicating entire teams to the development of content [26]. PCG refers to algorithmically generating content [16, 24, 26].

The interactivity between the game and the player through adaptation of the game based on player skill is the concept of DDA. A general overview of what this concept constitutes is given in Section 3.3.2. The game also requires to be interconnected with the player model, so in Section 3.3.3 and Section 3.3.4 two approaches to PCG that account for the player model are discussed.

3.3.1. ONLINE AND OFFLINE PCG DISTINCTION

There is a clear distinction to be made between generating game content either during a game session or before [24]. Offline PCG refers to the application of PCG before the start of a game session or even during development of the game. Such examples are generating levels before the player starts the game or even generating levels during the development of the game in which an algorithm or even the developer (manually) picks levels suitable for actual gameplay.

Online PCG is the exact opposite in the sense that the content is generated during the game session, i.e. while the player is playing the game, content is generated at the moment in time. An example of the online PCG application is when a level dynamically changes as a result of a player model. This change may also be a direct application of DDA that is discussed in Section 3.3.2.

3.3.2. DYNAMIC DIFFICULTY ADJUSTMENT

Every player is different in terms of skill. The notion of adapting a game by their difficulty is not a new one. When a player performs well, the game gets more difficult. The goal of changing the difficulty according to the performance of the player is to create a more adaptive game that caters to more people. When a game becomes easy to the point that it gets boring, the game adapts itself by raising the difficulty to suit the skill set

of the player. Similarly, when a game becomes too frustrating because it is too difficult, the game becomes easier [7, 24].

3.3.3. EXPERIENCE-DRIVEN PROCEDURAL CONTENT GENERATION IN GAMES

Experience-Driven Procedural Content Generation (EDPCG) models the player experience with a player model and adaptively generates content based on the player model [16, 26]. The player model in EDPCG consists of gameplay related data, such as how well a player objectively performs. Based on the data of the player model, content can be generated e.g. stochastically by even a most simple form of a pseudo-random number generator. This means that it is possible to apply EDPCG for generating random levels. In [26] three different types of Player Experience Modelling (PEM) are identified.

The first being subjective data from the players themselves, which the players can for instance provide by indicating what their current level of satisfaction is.

The second is objective data inferred from the players. This data can be captured by creating real-time video recordings and analyzing what emotional response players have to certain events.

Lastly, the most obvious type of PEM is modeling based on gameplay data. There is also a distinction in model-based and model-free approaches to PEM or a mix of them [22, 26]. The first relies on a theoretical framework, while the latter does not. Eventually, evaluating the generated content can be achieved by using evaluation functions [26]. These functions may be a direct mapping to some kind of score, simulation-based through running simulations with an agent and evaluating how the agent performs or even through an interactive approach with the player during a game session.

3.3.4. CONTEXT-DRIVEN PROCEDURAL CONTENT GENERATION IN GAMES

A much more rarely touched approach is the Context-Driven Procedural Content Generation (CDPCG). This approach generates content based on the context of the player rather than constructing a player model. The context dependency of this approach can be applied to e.g. time constraints [11]. In the setting of a dental clinic, gameplay duration is bounded by time because of time dependency of different kind of treatments. In this case, the CDPCG approach can also actively adapt the gameplay in either an online or offline manner. This approach can also be combined with other approaches such as EDPCG.

3.3.5. SEARCH-BASED PROCEDURAL CONTENT GENERATION IN GAMES

Using the previous case of generating levels, this can be done in a constructive or generate-and-test manner [21]. The first refers to generating content in a way such that by specifying specific rules, it ensures that the content is correct and also usable. Therefore, the generation is also only done once. The generate-and-test method randomly generates content and uses a fitness function to evaluate if the content satisfies all the requirements afterwards. Search-based Procedural Content Generation (SBPCG) is an application of the generate-and-test approach [21]. The fitness functions are similar to the ones previously mentioned in the EDPCG.

3.3.6. GRAMMAR CONSTRUCTION FOR PROCEDURAL CONTENT GENERATION IN GAMES

Grammars can help generate different types of structures that can be used to construct a system in a schematic manner. Grammars consist of a set of rules that transforms objects into something new by explicitly following those rules [23]. In [5, 23] it is shown that a grammar can be helpful to construct levels and missions in a game. In [23] this is done by representing the structure of missions as a graph. Applying the grammar to the graph can then create more complex structures, while preserving the correctness of the constructed content by definition of the grammar. The real challenge lies in interconnecting the generation of the level to the mission. Missions need to be carried out in the generated space. [23] proposes three strategies that can help solving this issue:

- The first strategy is to construct the mission first and directly construct the space accordingly.
- The second is to build a set of instructions from the constructed mission in order to guide the construction of the space.
- Lastly, is to build an abstract representation of the space and generate missions afterwards.

4

DESIGN

In this chapter, we elaborate on the overall design of the game. The complete design of the game is divided in three parts. First, we discuss the general game design in Section 4.1, which depicts the ideas behind the gameplay concepts. Following in Section 4.3, we discuss the design for the PCG which shows how the content of the game is generated. Lastly, in Section 4.2 the design for the player model which influences the gameplay elements is given.

4.1. GENERAL GAME DESIGN

Following the research phase, we discuss the design of the game in terms of its game play experience. In Section 4.1.1 we provide an overview of the target audience of the game. In Section 4.1.2 the setting of the game is thoroughly described, followed by the gameplay in Section 4.1.3. In the sections following, we describe the art style for the game, the sound and music used in the game, the user interface design and the controls of the game in sections (sections 4.1.4 to 4.1.7), respectively.

4.1.1. TARGET AUDIENCE

The game will be played at the dentist, so there may be a large discrepancy in age between the patients. Because of this diversity in types of patients, their skill levels vary a lot as well. Therefore, the game will be tailored to specific patients based on their skill. We make three distinctions in groups based on skill: 'newbies', 'casual players' and 'hardcore players'. The 'newbies' are players who rarely play video games. 'Casual players' play games on a regular basis, but are not necessarily skilled contrary to 'hardcore players'. Because of the diversity of skill in patients, the game will primarily target the 'newbies' and 'casual' players as these are most likely to be the majority. The game should be enjoyable for even a patient who has never player a video game before. The largest group of patients have likely played a video game before. These 'casual' players are therefore the main target audience. Finally, the 'hardcore' players are likely few in number, so the game is not specifically designed for them.

4.1.2. SETTING AND THEME

The game is set underwater. Patients should feel relaxed at the beginning of the game, so an underwater theme fits the game because it provides a soothing feeling. The player controls a yellow submarine and does not know anything about the current situation or why the submarine is underwater. The player has to navigate through caves, exploring the depths of the ocean. It is made clear throughout the game (by passing 'tranquil zones') that there is an end to these caves. Additionally, there is a symbolism of the enclosed caves during exploration indicating the initial fear for the dentist. Advancing in the caves is indicated by more brighter colors and more light eventually ending in an open space. This open space marks a sense of relief which coincides with the end of a treatment.

4.1.3. GAMEPLAY

We will give a general overview of the game in terms of its gameplay. This includes the objectives, general overview of the timeline, the scoring system and the other key features.

OBJECTIVES

The main objective of the game is to get out of the cave, while trying to survive. The player will ascend to the surface, while trying to obtain a high objective based score. The player must actively engage in steering the submarine to prevent crashing into the cave walls. Crashes will cause loss of oxygen and may lead to dying when oxygen levels drop dangerously low. This main objective provides the player with more context in why they are controlling the submarine and gives them an incentive to do so. Secondary objectives are collecting as many coins as possible for a higher score and also completing objective based missions. These objective based missions include the collection of special letter objects to form words that are fitting the underwater theme. Completing these missions will grant the player a score bonus. The scoring system will intuitively provide motivation for the player to try to perform well by completing objectives and collecting as many collectables as possible.

TIMELINE OF THE GAME

The player starts in an open, light and wide area of the cave to calmly introduce the player to the area. This will help with introducing new players to the gameplay. Players can experiment more freely with the controls and are given more space to observe dangerous enemies. Furthermore, the open area marks the start of the game and makes it clear to the player that this is the beginning to a (long) journey through the caves. The starting area's bright light is also consistent in the recurring 'tranquil' zones in the midst of the game which indicate that the player is given a brief window to relax.

The view continuously moves to the right and the player remains inside this view at all times. The player may not go off the top and bottom of the screen and will be pushed forward at the left edge of the screen.

During the game, the difficulty will (slowly) go up. The speed of this increase in difficulty is dependent on the skill of the player as indicated by the player model (section 4.2). At specific marks of traveled distance, 'tranquil zones' will be introduced to the player. These zones will be repeated in fixed intervals as shown in Figure 4.1 (i.e. when the player passes the defined specified distances). During a tranquil zone, the difficulty of the game is lowered by a large margin (e.g. enemies are not present or more oxygen to prevent death is given). After the player exits the tranquil zone, the difficulty starts lower than the the level it was at the moment the player entered the tranquil zone. By using tranquil zones this way, the game maintains a certain rhythm and makes the intense moments feel more intense. At the tranquil zones, small markers are placed with a remaining distance to the end. This gives a more direct sense of progress throughout the game.

Approaching the end of the level, the player will slowly approach the exit of the cave and eventually end in at surface. More light shines from above and brighter colors are used to indicate ascension. After reaching the surface, the player is given a summary of how he performed. Afterwards, the player may roam in the ocean freely for indefinite amount of time viewing the scene of a sunset. The dentist indicates how long the treatment will take beforehand.

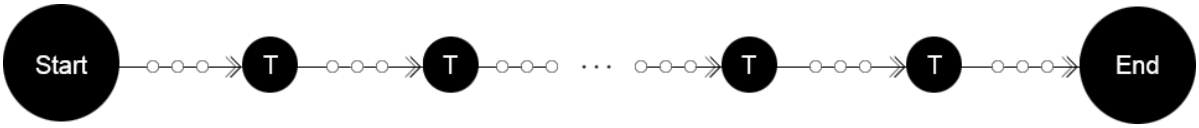


Figure 4.1: At fixed intervals, 'tranquil zones' are placed. They are indicated by a 'T'

SCORING SYSTEM

The player's score is influenced by several variables:

Collectable treasure Various types of treasures can be found throughout the level. Bronze, silver and gold coins give a score increase of 50, 100, and 200, respectively. Lower valued coins are generally more common than the higher valued ones. Other, more rare treasures, including pearls and diamonds, valued 1000 and 5000, respectively, can sometimes be found in places that are difficult to reach.

Mission objectives Each letter found awards a score increase of 1000. Upon completion of a word an additional 10000 is awarded.

Death Every time the player dies a -25% score deduction based on the current score is applied. A percentage based death penalty feels more forgiving towards lower skilled players rather than a deduction of a fixed amount of points.

Combo modifier The score the player gains by collecting treasures is affected by a multiplier that rises as the player collects treasures in succession. This modifier starts at 1 and is capped at 5. The modifier drops when the player fails to collect treasures (i.e. missing them).

KEY GAME ELEMENTS

Up until here, many game elements have already been identified. We summarize the most important game elements once more and provide some more detail to them. A more comprehensive list with motivation behind the game elements can be found in Appendix E.

The game consists of an 'endless level', in which the view continuously moves to the right. The actual length of the level will be determined based on the amount of time the treatment requires (this is something the dentist will determine at the start of the game).

The submarine controlled by the player is affected by a gravitational force which pulls it towards the bottom of the view, potentially causing a crash with the cave wall. The player is supposed to keep the submarine away from the cave walls by steering accordingly.

During the game, the player will encounter collectables in the form of coins, pearls and diamonds which differ in rarity and reward. Missions appear in the form of collecting letters to form a word.

Inside the caves, dangerous elements such as turrets and mines are present. The player should avoid getting hit by these elements.

The difficulty of the game is adjusted to how well the player has performed so far. This is a continuous process of evaluation.

Finally, progress is indicated by the 'tranquil zones', marking the distance left until the end of the game.

4.1.4. ART STYLE

The game is set underwater and mainly uses shades of blue color for the cave. The dark color of the walls indicate that it is closer to the view, while brighter colors are used for the background in the far distance. Water is presented as a blue-tinted gradient. The overall style is minimalistic (i.e. few details used for the different game elements).

4.1.5. SOUND AND MUSIC

The patient will not be equipped with headphones during treatment, as this will impede communication with the dentist. Additionally, due to restrictions of audibility caused by the distance to the speakers of the machine running the game, the volume of the sound would need to raised. This may disturb the dentist, causing loss of concentration required for the treatment. In order to prevent unnecessary disturbances sound will be present in a minimal fashion. Sounds are therefore only used to give necessary feedback (i.e. when items are collected and taking damage).

4.1.6. USER INTERFACE

An oxygen level meter is displayed in the bottom left corner of the screen. The meter is displayed as a percentage going from 100% to 0%, in whole percent points. The color of the meter changes depending on the oxygen level:

- Green = 76-100%
- Yellow = 36-75%
- Red = 0-35%

The current score is displayed in the bottom right corner as: 'Score: x' ('x' is the current score as a whole number). The combo modifier is shown in the top right corner as: 'Combo: y' ('y' is the current combo modifier as a number ranging from 1.0 to 5.0).

4.1.7. CONTROLS

During treatment patients will be facing their head towards the ceiling at all times. Patients also cannot see the controller, because they are equipped with a pair of VR-goggles which functions as the monitor. As a result, patients are not able to look down to their controller, so simplistic controls are essential. For the entire game, only one joystick will be used for the movement of the player's submarine.

4.2. PLAYER MODEL DESIGN

We will discuss the logging of events in Section 4.2.1 and the design of the player parameters in Section 4.2.2.

4.2.1. EVENT LOGGING

During gameplay, several event types are being logged. An event type is defined as some sequence of events during gameplay. Several event types have been defined. For example, the event in which a player collides with a cave wall and the event where the player collects a collectable are being logged.

The logged events are used to determine what the values of the player parameters should be. This process is described in Section 4.2.2.

The events that have been logged are never deleted during a game session. This is beneficial, because the whole history of events can be used for determining player parameters, which results in more accurate player parameter values. The memory space that the events take up is not a relevant constraint, because the logged events are very small in size and their number is so small that the memory space they occupy is negligible.

The information that is recorded for each event type consists of a number of attributes. This set of attributes is the union of a set of global attributes that are recorded for each event type and a set of attributes that are recorded for the specific event type.

All global attributes along with a description are shown in Table 4.1. All events, along with a description, their specific attributes, and a description of their specific attributes are shown in Table 4.2.

Attribute	Description
Time since game start	Time since the start of the game session. This is the pure playing time, so time in pause mode, for example, will not be included.
Score	Score of the player before the event occurred. For example, if the player collects a collectable that increases the score upon collection, the score before increasing is recorded.
Oxygen level	Oxygen level of the player before the event occurred. For example, if the player hits a mine and loses oxygen, the oxygen level before the collision is recorded.
Level generation parameters	Level generation parameters that were in effect before the event occurred. The level generation parameters are described in Section 4.3.

Table 4.1: Global event attributes. These attributes are recorded for all event types.

4.2.2. PLAYER PARAMETERS

During a game session, several player parameters are maintained. These parameters represent some aspect of a player, which can change throughout a game session. All player parameters indicate how well a player is doing in one aspect of the game. The defined player parameters include obstacle evasion, wall evasion and collectable collecting. Their meaning is defined as follows:

Obstacle evasion Indicator for how good the player is at evading obstacles. The obstacles consist of mines and plant turrets. They are described in Section 4.1.3.

A player is defined to be 'good' at evading obstacles if little damage is taken because of interactions with obstacles. If a player takes a lot of damage because of interactions with obstacles, the player is considered to be 'bad' at evading obstacles.

The obstacle evasion parameter is a floating point value in the range [1,5]. A player who is very bad at evading obstacles is represented by a value of '1', while a player who is very good is represented by a value of '5'. The obstacle evasion value and the actual obstacle evasion skill of the player are (theoretically) directly proportional.

Wall evasion Indicator for how good the player is at avoiding the top and bottom walls of the cave. A player is viewed as 'good' at evading cave walls when the player takes little damage from colliding with the cave

Event type	Description	Attribute	Description
Collision with bullet	Player collides with a bullet that has been shot by a turret plant.	Invulnerability	Indicator for whether the player submarine was invulnerable at the time of the collision. This may be relevant for deciding if a collision was harmful to the player or not.
Evasion of bullet	Player evades a bullet that is shot by a turret plant.	*No specific attributes*	
Collection of a collectable	Player collects a collectable, usually by colliding with it. Collectables include coins and oxygen bubbles.	Type of collectable	Kind of collectable that has been collected. This can be either a bronze, silver or gold coin and an oxygen bubble.
		Difficulty of placement	Indicator for how difficult the location of the collected collectable is to reach. Collection of a collectable at a location of a higher difficulty may indicate a higher skill level of the player than collecting a collectable at a location of lower difficulty.
Missing a collectable	Player does not collect a collectable. This typically happens when the collectable goes off screen. Collectables include coins and oxygen bubbles.	Type of collectable	Kind of collectable that has been missed. This can be either a bronze, silver or gold coin and an oxygen bubble.
		Difficulty of placement	Indicator for how difficult the location of the collected collectable is to reach. Missing of a collectable at a location of a lower difficulty may indicate a lower skill level of the player than missing a collectable placed at a location of higher difficulty.
Oxygen supply exhausted	Player has run out of oxygen and receives a penalty.	*No specific attributes*	
Collision with mine	Player collides with a mine.	Invulnerability	Indicator for whether the player submarine was invulnerable at the time of the collision. This may be relevant for deciding if a collision was harmful to the player or not.
Evasion of mine	Player evades a mine. This typically happens when the mine goes of screen without the player having hit it.	*No specific attributes*	
Collision with cave wall	Player collides with the top or bottom wall of the cave.	Speed of collision	Speed with which the player submarine collides with the cave wall. Collisions at a faster speed may indicate a lower skill level.
		Top or bottom wall	Indicator for whether the collision was with the top or bottom wall of the cave.
		Invulnerability	Indicator for whether the player submarine was invulnerable at the time of the collision. This may be relevant for deciding if a collision was harmful to the player or not.
		Wall section difficulty	Indicator for how difficult the wall section that the player collided with was to navigate.

Table 4.2: All event types in the system, along with their specific attributes.

walls. A player is said to be 'bad' at evading cave wall when taking a lot of damage because of collisions with the cave walls.

Similar to the obstacle evasion parameter, the wall evasion parameter is a floating point value in the range [1,5]. The meaning of the value is analogue to the meaning of the value of the obstacle evasion parameter.

Collectable collecting Indicator for how good the player is at collecting collectables. These collectables include bronze, silver and gold coins and oxygen bubbles. A player is considered to be 'good' at collecting collectables if he collects a large part of the collectables in the game, while a player is seen as 'bad' at collecting collectables if the player only collects a small part of the collectables in the game.

Similar to the obstacle evasion parameter, the collectable collecting parameter is a floating point value in the range [1,5]. The meaning of the value is analogue to the meaning of the value of the obstacle evasion parameter.

The player parameters can be used for several different purposes. They are primarily used to generate level generation parameters in order to adjust the difficulty of the game to fit one particular person.

If one would include none-performance parameters in the player parameters, like anxiety, one could also use the player parameters for purposes outside the game, like informing the dentist about the anxiety level of the patient.

The player parameters can be set manually by the developers of the game, by the user of the system, or they can be dynamically generated. The player parameters are determined solely by the previous state of the player parameters and the set of logged events. The influence of the logged events on the three player parameters is shown in Table 4.3. See Section 4.2.1 for details about event logging.

Event type	Obstacle evasion	Collectable collecting	Wall evasion
Bullet collision	↓		
Bullet evasion	↑		
Collectable collection		↑	
Collectable miss		↓	
Oxygen exhausted			
Mine collision	↓		
Mine evasion	↑		
Wall collision			↓

Table 4.3: Influence of event types on player parameter values. An upwards arrow is shown if the occurrence of the event of the specified type has an increasing effect on the player parameter, while a downward arrow is shown if it has a decreasing effect.

The logic that is used to determine the values of the player parameters is shown in Algorithms 1 to 3 for obstacle evasion, wall evasion and collectable collecting respectively. The logic will be clarified for each parameter. Note that the algorithms shown here do not exactly mirror the implementation but only serve to show the conceptual logic behind the parameter value computation.

Obstacle evasion The algorithm for computing the obstacle evasion parameter value is shown in Algorithm 1. This algorithm is executed each time an obstacle is encountered.

A sliding window of fixed size is used that slides over the encountered obstacles. The evasion rate, or the fraction of encountered obstacles that has been evaded, for the window is compared with the total evasion rate for the whole game session except the current window. If the evasion rate is sufficiently high, the obstacle evasion parameter value is increased, otherwise it is decreased. Note that a rate modifier is used to determine how high the window's evasion rate should be compared to the total evasion rate in order for the obstacle evasion parameter value to increase.

Algorithm 1 Computing a new obstacle evasion parameter value

```
1: if (obstaclesEncountered mod windowSize) = 0 then
2:     if windowEvasionRate() ≥ rateModifier ∗ totalEvasionRate() then
3:         offsetBase := increasePerObstacle
4:     else
5:         offsetBase := decreasePerObstacle
6:     end if
7:     newOffset := offsetBase ∗ windowStepSize
8:     obstacleEvasion := obstacleEvasion + newOffset
9: end if
```

Wall evasion The algorithm for computing the wall evasion parameter value is shown in Algorithm 2. This algorithm is executed each game time step. This algorithm maintains the average number of wall chunks that are successfully navigated between two wall collisions. A wall chunk is successfully navigated when the player does not collide with it. The algorithm also maintains the average difficulty of the chunks with which the player collides.

When a collision occurs, these two values are compared with the number of successfully navigates chunks and the difficulty of the chunk that the player has collided with, respectively. If either of the current values is smaller than the averages, the wall evasion parameter value is decreased. If the player does not collide with a wall for a specified period of time, the wall evasion parameter value is increased.

Algorithm 2 Computing a new wall evasion parameter value

```
1: if collisionOccurred then
2:     bufferInterval := 0
3:     if lastChunkInterval < averageChunkInterval OR
4:         lastCollidedDifficulty < avgCollidedDifficulty then
5:         wallEvasion := wallEvasion − decreaseValue
6:     end if
7: else if bufferInterval ≥ intervalThreshold then
8:     wallEvasion := wallEvasion + increaseValue
9:     bufferInterval := 0
10: end if
```

Collectable collection The algorithm for computing the collectable collecting parameter value is shown in Algorithm 3. This algorithm is executed each time a collectable is collected or missed (i.e. not collected).

If a collectable is collected, the difficulty of placement is compared with a threshold difficulty. If the difficulty is higher than the threshold, the collectable collecting parameter value is increased. A similar method is used for missing a collectable. In that case, it is checked whether the difficulty of placement is lower than some threshold value.

If collecting or missing happens a certain threshold amount of times in a row, the collectable collecting parameter value is increased or decreased regardless of the difficulty value of the collectables that have been missed or collected.

Algorithm 3 Computing a new collectable collecting parameter value

```
1:  if collectableCollected then
2:      collectedCounter := collectedCounter + 1
3:      missedCounter := 0
4:      if difficulty ≥ difficultyUpValue OR collectedCounter ≥ counterThreshold then
5:          collectableCollecting := collectableCollecting + increaseValue
6:          collectedCounter := 0
7:      end if
8:  else if collectableMissed then
9:      missedCounter := missedCounter + 1
10:     collectedCounter := 0
11:     if difficulty ≤ difficultyDownValue OR missedCounter ≥ counterThreshold then
12:         collectableCollecting := collectableCollecting – decreaseValue
13:         missedCounter := 0
14:     end if
15: end if
```

4.3. PROCEDURAL CONTENT GENERATION DESIGN

We will discuss the PCG design by focusing on the chunk design in Section 4.3.1 and level parameters generation in Section 4.3.2.

4.3.1. CHUNK DESIGN

The procedural game level generator uses 'building blocks' or 'chunks' to construct the level. The generated level is a sequence of these chunks, which vary, for example, in shape and in location of collectables and obstacles. The variation among the chunks ensures that the level resulting from the generation will be varied enough. In each chunk objects can be placed at the location of markers in several places inside the chunk. The usage of these markers is controllable and is influenced by the player model. Sometimes, only a specific group of chunks may be considered for placement. For example, to influence the width of the cave, placing a chunk of only one of the following three types may be desirable.

Narrowing chunk The cave becomes smaller when this type of chunk is placed, because of the significant height difference between the left and right end of the chunk.

Widening chunks The cave becomes wider when this type of chunk is placed, because of the significant height difference between the left and right end of the chunk.

Straight chunks The cave width stays approximately the same, because the height difference between the left and right end of the chunk is small.

In Figures 4.2 to 4.7 several chunks used for varying the cave width are shown.

Chunks are also categorized by difficulty in order to control the difficulty of the game. The game uses an easy, a medium and a hard category. The placement of a chunk in one of the categories is based on the roughness of the chunk and the amount of protrusive cave wall parts. The chunk in Figure 4.2 contains stalactites which make the chunk more dangerous than a flat chunk like Figure 4.3. Chunks with large 'holes' as in Figure 4.7 are more spacious and give the player more room to move. Therefore, these types of chunks are of much lower 'difficulty'.

As part of the aesthetic, chunks also contain 'contours' which form a moving background to creating a sense of depth by placing them at varying distances from the cave wall.

4.3.2. LEVEL PARAMETERS GENERATION

The level generator uses level parameters to steer the level generation. These level parameters are influenced by the player parameters. All the level parameters are real numbers ranging from 1.0 to 5.0. The level parameters that the game uses are now listed, categorized by the game element they influence.

- Cave: width, chunk difficulty
- Coins: placement

Figure 4.2: Top chunk that makes the cave smaller.

Figure 4.3: Top chunk that does not affect the cave width.

Figure 4.4: Top chunk that makes the cave wider.

Figure 4.5: Bottom chunk that makes the cave smaller.

Figure 4.6: Bottom chunk that does not affect the cave width.

Figure 4.7: Bottom chunk that makes the cave wider.

- Oxygen: frequency, placement
- Turret: frequency, accuracy
- Mines: frequency, placement
- Camera: speed

The **cave width** level parameter is computed as follows and depends on all the player parameters:

$$\begin{aligned} &maxWidth - (maxWidth - minWidth) * (\frac{1}{3} * \frac{1}{4} * (WallEvasion - 1) + \\ &\frac{1}{3} * \frac{1}{4} * (CollectablesCollecting - 1) + \frac{1}{3} * \frac{1}{4} * (ObstacleEvasion - 1)) \end{aligned} \tag{4.1}$$

The $maxWidth$ and $minWidth$ are the maximum and the minimum width of the cave, respectively.

The **chunk difficulty** is determined by the WallEvasion and CollectablesCollecting player parameters, as follows:

$$\frac{1}{2} * WallEvasion + \frac{1}{2} * CollectablesCollecting \tag{4.2}$$

Each chunk has a difficulty rating from 1, indicating the chunk is easy to navigate, to 3, indicating the chunk is difficult to navigate. Chunks are selected for placement based on the combined difficulty of the top and bottom chunks. Here the sum equals the difficulty rating of the top chunk plus the bottom difficulty rating of the bottom chunk.

- 1: sum = 2.
- 2: sum <= 3
- 3: 3 <= sum <= 4
- 4: 4 <= sum <=5
- 5: sum >= 5

A chunk difficulty parameter value of 1 means that only chunks with a combined difficulty rating of 2 are used for the generation. For a chunk difficulty parameter value of 5, only chunks with a combined rating of 5 or higher are used. The lowest possible combined rating is 2 (1 + 1) and the highest possible combined rating is 6 (3 + 3).

The **collectable placement** parameter is equal to the CollectableCollecting player parameter, rounded to the nearest integer. The difficulty of the locations in which collectables are placed is as follows.

- 1: placement difficulty 90% easy, 10% medium
- 2: placement difficulty 70% easy, 30% medium
- 3: placement difficulty 50% easy, 50% medium
- 4: placement difficulty 20% easy, 70% medium, 10% hard
- 5: placement difficulty 10% easy, 70% medium, 20% hard

Suppose the collectable placement difficulty parameter value is 4. Then, for each collectable that is placed in the level, the chance of it being placed in an easy, medium or hard location is 20%, 70% and 10%, respectively.

The **obstacle frequency** level parameter determines the frequency with which all kinds of obstacles, like mines and plant turrets, are placed. It is equal to the ObstacleEvasion player parameter, rounded to the nearest integer.

- 1: 1 per 5 chunks
- 2: 1 per 4 chunks
- 3: 1 per 3 chunks
- 4: 1 per 2 chunks
- 5: 2 per 2 chunks

For example, when the obstacle frequency level parameter has a value of 3, an obstacle will be placed once for every 3 chunks.

The **obstacle difficulty** affects the difficulty of the locations in which turrets are placed and the rate at which turrets fire bullets. It is determined by the ObstacleEvasion and CollectablesCollecting player parameters and is computed as follows:

$$\left\lfloor \frac{1}{2} * ObstacleEvasion + \frac{1}{2} * CollectablesCollecting \right\rfloor \tag{4.3}$$

The fire rate and placement location difficulty of the obstacles is determined by the parameter as follows.

- 1: turret accuracy 30%, placement difficulty 100% easy
- 2: turret accuracy 45%, placement difficulty 75% easy, 25% difficult
- 3: turret accuracy 60%, placement difficulty 50% easy, 50% difficult
- 4: turret accuracy 75%, placement difficulty 25% easy, 75% difficult
- 5: turret accuracy 90%, placement difficulty 100% difficult

The **oxygen frequency** is computed based on the WallEvasion and ObstacleEvasion and the current oxygen level. The general idea is to provide the player with more oxygen when the player is currently low on oxygen and if the player is bad at evading.

$$\frac{1}{4} * (6 - WallEvasion) + \frac{1}{4} * (6 - ObstacleEvasion) + \frac{1}{2} * \frac{100 - OxygenLevel}{20} \tag{4.4}$$

- 1: 1 bubble per 4 chunks
- 2: 1 bubble per 3 chunks
- 3: 1 bubble per 2 chunks
- 4: 1 bubble per chunk

- 5: 2 bubble per chunk

For example, when the oxygen frequency level parameter has a value of 3, one oxygen bubble will be placed every other chunk.

The **camera speed** is based on the CollectablesCollecting parameter and on the part of the total game time that has already elapsed. The camera speed slowly increases during the game, but is scaled by the CollectablesCollecting player parameter. This means that if the player is good at collecting collectables, the camera speed will be higher on average. The camera speed will be at least the minimum camera speed and at most the maximum camera speed.

$$minSpeed + \frac{1}{5} * CollectablesCollecting * minMaxElapsedTimeScale(0,1) * (maxSpeed - minSpeed) \tag{4.5}$$

5

FINAL PRODUCT

In this chapter the final product is described. The final product consists of a fully controllable procedural game level generator, the player model modules and the addition of various gameplay elements. In Section 5.1, the extensions and changes of the overall gameplay are outlined. In Section 5.2, the finalized player model structure is discussed. Similarly in Section 5.3, we review the finalized procedural game level generator. Finally, in Section 5.4, we evaluate if the initial requirements (section 2.3) were met in the final product and what game features (appendix E) were eventually included.

5.1. GAMEPLAY EXTENSION

The gameplay has changed by having a different concept of a 'level'. Compared to the prototype game, the game no longer has separate levels. It operates on an 'endless' level, which is dynamically constructed by the level generator.

The concept of the player's 'life gauge' is changed as well. The player now has an 'oxygen meter' which counts from 100% to 0%. The player uses 1% oxygen per second to breathe underwater, which results in a decrease of 1% oxygen per second visible in the life gauge. Three color ranges are used to indicate low (red), medium (yellow) and high (green) oxygen levels.

In order to account for the new player life concept, oxygen bubbles are introduced to help the player regain their oxygen. These bubbles can be found during the whole level.

The left edge of the screen now pushes the player forward, instead of letting them go off the screen and placing them back in the middle of the screen in exchange for some health.

Furthermore, the overall difficulty scales to the player's abilities. For example, the player will encounter a different amount of obstacles, which may harm the player, depending on their in game performance.

Additionally, death now causes a score penalty instead of causing the level to end.

Highscores have also been added to the game, which allow the player to save his score at the end of the game and compare the score to the his own previous scores or to scores of other players. The player can identify his self using three letters.

Plants are animated in the background scenery while in game.

Additional sounds for providing feedback when the player receives damage and collects coins have been added.

Finally, the ability to pause the game has also been added.

5.2. PLAYER MODEL EVALUATION

The final product uses a player model to steer the level generation based on a player's skills. Three skills were identified that formed a profile for a player, sufficiently describing the player's characteristics in terms of gameplay performance. The player's performance is measured by the types of events that occur (see Table 4.2). These events are also logged in a separate file. The player skills are based on the performance in navigation, collecting collectables and evading harmful elements.

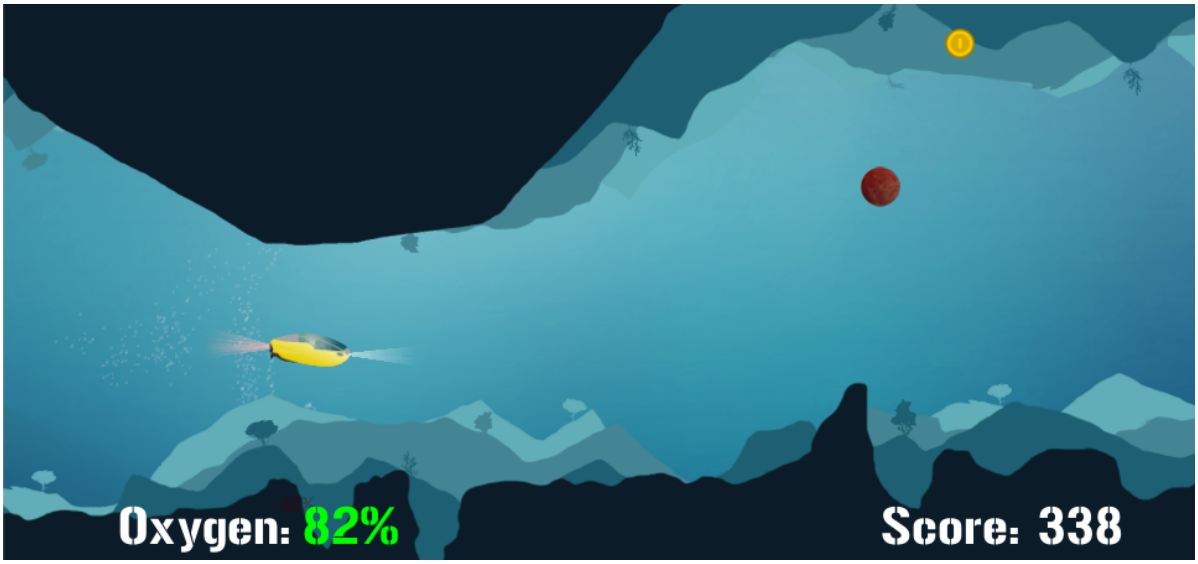


Figure 5.1: The player encounters a mine and a coin.

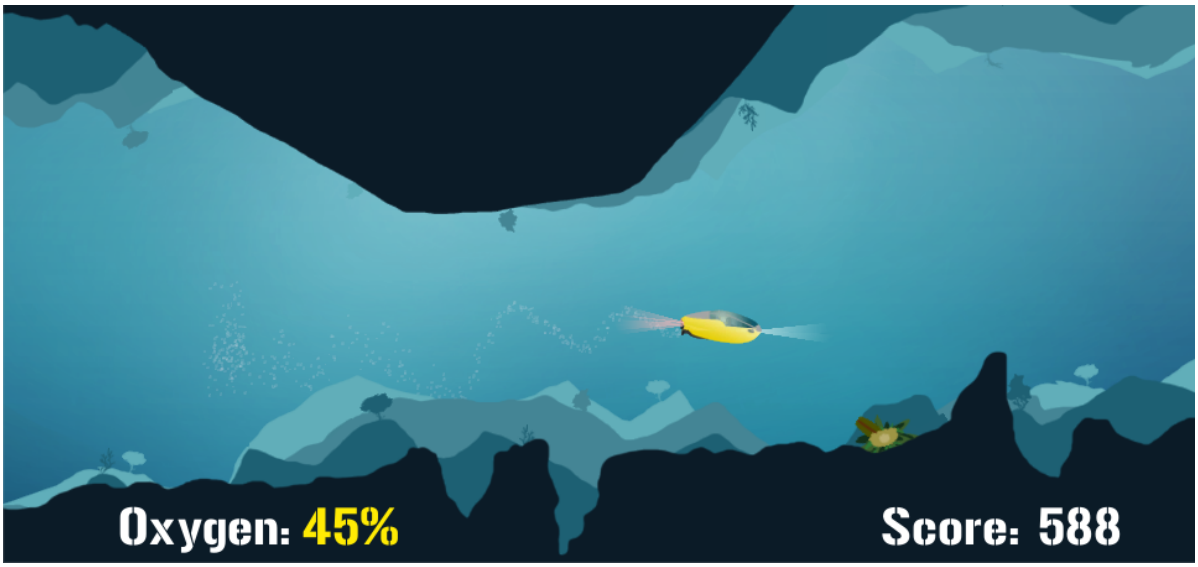


Figure 5.2: The player encounters a turret.

5.3. PROCEDURAL GAME LEVEL GENERATOR

The entire level is dynamically constructed during gameplay by placing 'building' blocks or chunks (see Figure 5.3). There are 30 unique chunks in total (top and bottom combined) excluding mirrored versions which brings the total above 45 chunks. Not all chunks have a mirrored version, hence the total is not higher. The large selection of chunks allows for a lot of variation in the generated levels. These chunks are designed beforehand and contain object placement markers that allow for controllability of object placement inside these chunks (see Figure 5.4). As a result, the player may encounter different types of game objects based on the player model steering the level generator. This is illustrated in Figures 5.1 and 5.2. The player encounters a coin and a mine in Figure 5.1, while a turret is encountered in the same chunk in Figure 5.2. Because the level 'moves' from left to right, the chunks will also be placed in a similar fashion. Every new chunk will be placed to fit the characteristics of the adjacent chunk. To ensure optimal use of computer working memory space, chunks are dynamically placed and removed. Only the chunks in and close to the camera view are kept in memory. This construction is shown in Figures 5.5 and 5.6. The size of the width window in which chunks are kept in memory can be adjusted.

The level generation is configurable in many ways, such as controlling the object placement by means of

difficulty, odds of placements, amount and their type (e.g. collectables, hazards or turrets). This is done using the aforementioned object placement markers (see fig. 5.4). These markers have a variable chance of objects being placed at their location determined by the skill of the player from the player model.

The level generator uses all level parameters from Section 4.3.2 to steer the level generation. The level generator also attempts to smoothly transition to newly generated content (i.e. chunks with matching characteristics to what the player model determined). The player will experience smooth gameplay, because the effects of the parameters gradually build-up towards the set value. This means that when the level parameters are adjusted, the effect is slowly applied across several chunks.

As an example, when the cave width adjusts to a newly determined width by evaluating the player model, the level generator will smoothly transition the cave to this new width. This means that it may take several chunks to be placed to achieve the new width.

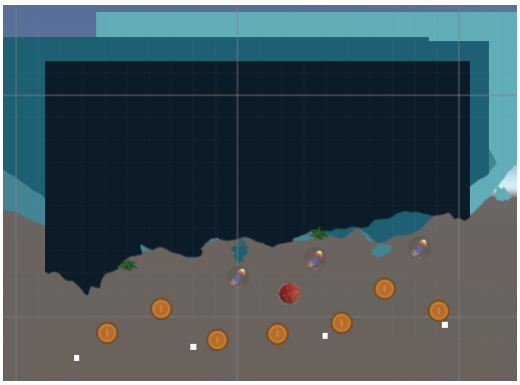


Figure 5.3: A single chunk containing markers for objects.
These markers do not represent actual objects, but act as a possible location for them.

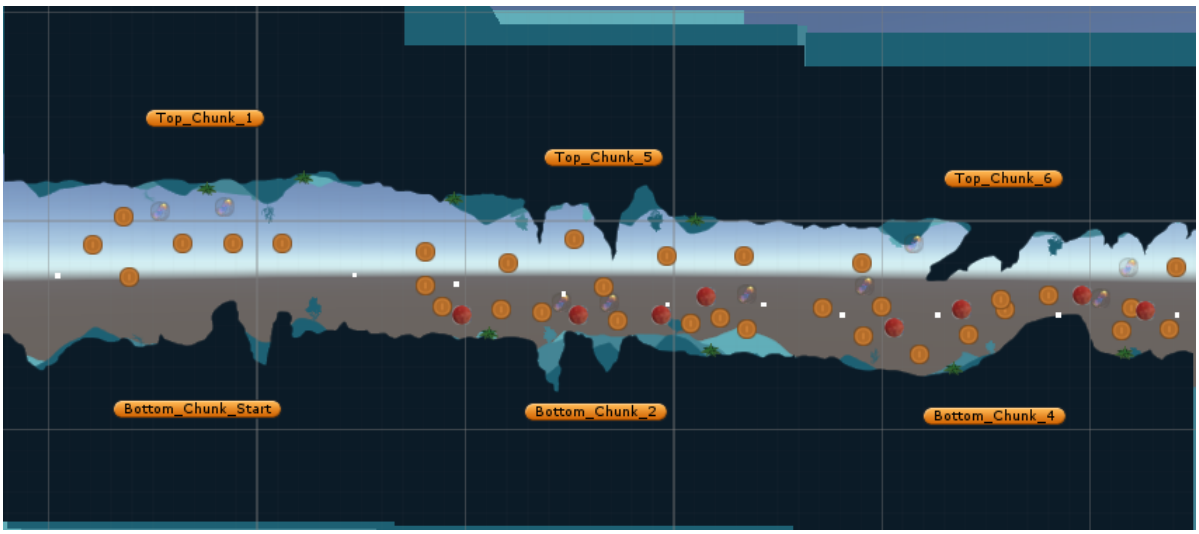


Figure 5.4: Several chunks used for the level generation.

5.4. REQUIREMENTS EVALUATION

We reflect on the initial requirements (section 2.3) in Section 5.4.1 and the game design feature list (appendix E) in Section 5.4.2. We outline what requirements are met in the final product and which are not.

5.4.1. FUNCTIONAL & NON-FUNCTIONAL REQUIREMENTS EVALUATION

We establish, following the functional requirements in MoSCoW prioritisation in Section 2.3, that all **must have**, almost all **should have**, and several **could have** requirements have been fulfilled.

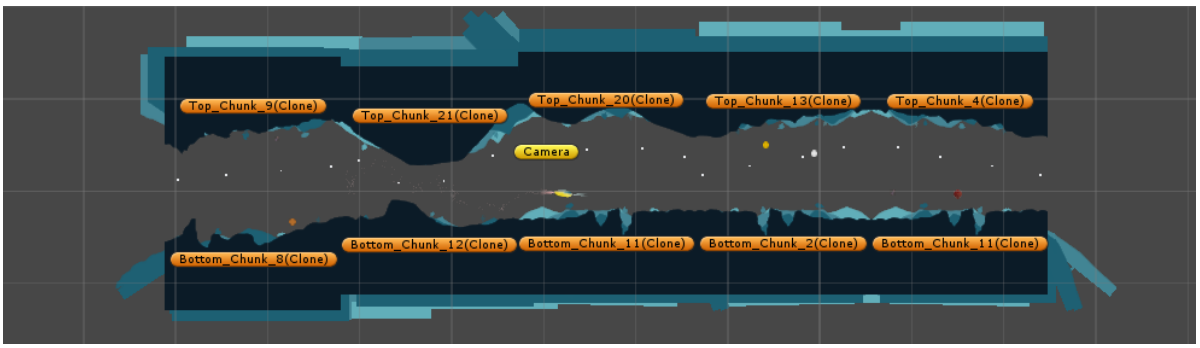


Figure 5.5: The chunks that are in memory at any point in time are always around the camera view. The player does not notice this, because the chunks in view are always in memory.

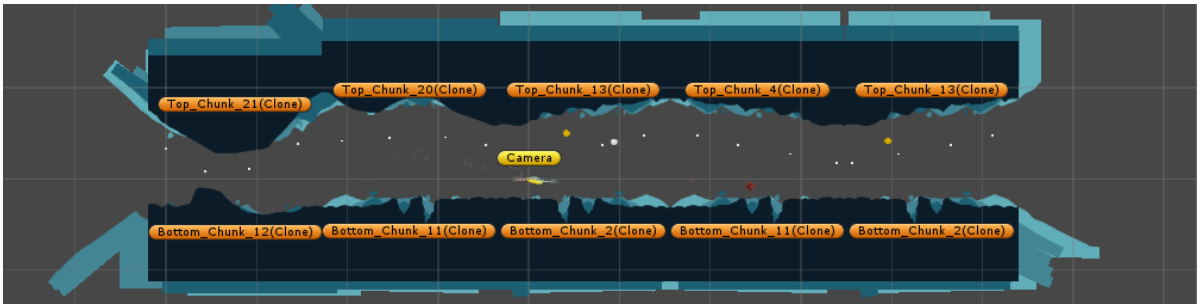


Figure 5.6: Most left chunks are deleted and new chunks are placed at the right.

The **must haves** included implementing a player model that captures relevant player data, and the level generator using the player model to construct the levels. Additionally, the time-context is also taken into account by the level generator by adjusting the duration of the game.

Following the **should haves**, the game has a clear ending by letting the player out of the caves into an open space. Furthermore, the game is able to be paused by pressing the 'ESC' button. The game is also intuitive with the goal of collecting as many coins as possible, because it is made clear that these coins reward the player. Players also take damage when crashing into the cave walls, which makes it clear that they should steer away from the walls. However, the goal of 'escaping' the cave may not be clear because this is not explicitly explained or shown until the very end. Last, the background elements are clearly distinguished from the gameplay elements. The difference in color and depth makes it clear that the parts of the background will not influence the player.

Several of the **could haves** have also been included in the final product. The leaderboard is shown after the player inputs his name at the end. The player will see his placement in the rankings compared to previously played games. The continuous feedback loop was not included because of the new level design. The levels were no longer separate levels that the player could 'complete'. Therefore, it was not necessary to provide a continuous loop of feedback. The game also does not account for different styles of play. This was originally meant as a way for players to completely focus on one part of the game they particularly liked and isolate other elements by omitting them. At last, the player is indeed rewarded for playing well by achieving a higher score.

The non-functional requirements have all been achieved. During the development special attention was given to several of the requirements such as intuitivity of the controls, quick starting of the game and no scary game elements. The game has been tested using the Vuzix 1200 VR goggles and there were few issues except from sometimes having a low quality of the view. This was solved by using the various sliders on the goggles to adjust the view.

5.4.2. GAME FEATURE EVALUATION

In Appendix E, the concrete game features following from the game design have also been categorized using a MoSCoW prioritization. The final product includes all the **must have** features and all **should haves** except for the 'tranquil zones'. This feature was initially implemented, but did not survive a reorganization of the

player model module structure. This feature was not reimplemented in the end due to time constraints. Furthermore, the **could have** features did not make the final product because of the time constraints of the project.

6

IMPLEMENTATION

We begin by discussing the implementation of the event logging in Section 6.1. Followed by the player parameters and level parameters in Sections 6.2 and 6.3. Finally we discuss how chunks and markers are placed in Sections 6.4 and 6.5.

6.1. EVENT LOGGING

Event logging is implemented using `AbstractEvents`, which are records of events that are triggered during the game. The responsibility of an `AbstractEvent` is to hold information about an event that has occurred. It does not need to detect and record the events that happen. This is the responsibility of an `AbstractEventDetector`, which is described later.

The events being logged are described in Section 4.2.1 and include, for example, wall collisions and collection of coins. Each type of event is represented by a subclass of `AbstractEvent`. A UML class diagram showing the structure of the event classes is included in Figure 6.1. A UML class diagram of the whole system is shown in Appendix D. `AbstractEvent` defines attributes that are recorded for all events, while the subclasses of `AbstractEvent` define the attributes that are specific for each event type. For a detailed description of the events that the subclasses of `AbstractEvent` represent, refer to Section 4.2.1.

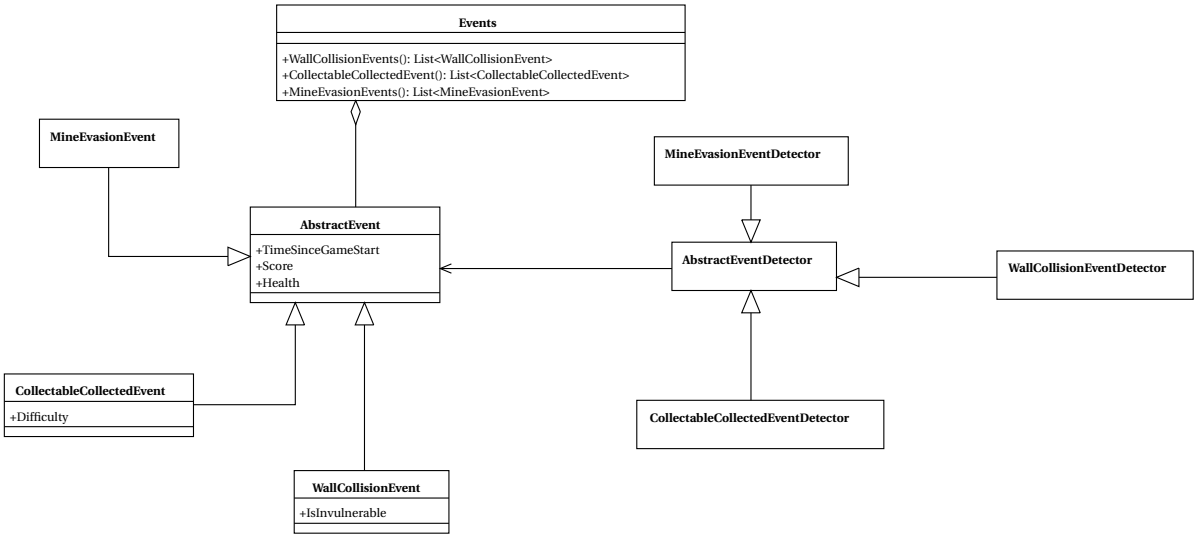


Figure 6.1: UML class diagram showing the structure of the event classes

The `Events` class manages a collection of events. `Events` provides methods for adding new `AbstractEvents` and for retrieving `AbstractEvents` that have been added previously. `AbstractEvent`s that are added to an `Events` instance are never deleted while the `Events` instance exists.

A central `Events` instance is used by the game for recording events. This object is attached to a Unity

GameObject, which is only destroyed, along with the attached Events instance, when the Unity scene is closed. Other instances of Events are used in some cases, for example for loading an event log file that has been created after a play session.

Note that all components that have been discussed until now in this section have almost no knowledge of the Unity engine. The task of interfacing with the Unity engine for detecting events is fulfilled by AbstractEventDetectors. Each AbstractEventDetector is attached to some Unity GameObject for detecting events that are related to those GameObjects. All subclasses of AbstractEvent have a corresponding subclass of AbstractEventDetector for detecting the event. The WallCollisionEventDetector, for example, listens for collisions of the player submarine with a cave wall, and logs the event in a WallCollisionEvent instance if such a collision occurs.

6.2. PLAYER PARAMETERS

The player parameters, for which the design has been described in Section 4.2.2, are represented in the PlayerParameters class. The actual generation of the parameters based on the occurred events is handled by classes implementing the IPlayerParameterGenerator interface. The PlayerParameters class, the IPlayerParameterGenerator class, other relevant system elements and their relations are shown in the UML class diagram in Figure 6.2.

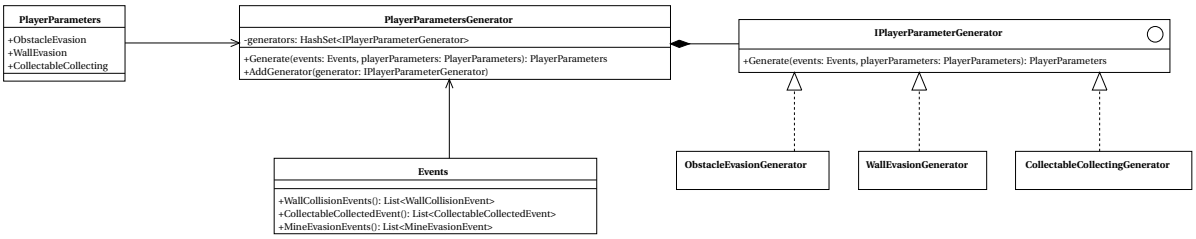


Figure 6.2: UML class diagram showing the structure of the player parameter classes

In the PlayerParameters class, each parameter is stored as a C# property that can be accessed by other system elements. The PlayerParameters class is only responsible for storing the parameters and ensuring that they are in an allowed state.

A PlayerParametersGenerator is responsible for determining what the value of each of the player parameters should be. It takes in consideration the set of all events that have occurred, represented by an Events instance, and the current set of player parameters represented by a PlayerParameters instance.

The game maintains one PlayerParameters instance and one PlayerParametersGenerator instance that are used for managing the player parameters during a game session. The PlayerParameters instance that is used during a game session can be accessed via a Unity GameObject, to which the PlayerParametersGenerator is also attached for updating the player parameter values.

A PlayerParametersGenerator generates new values for the parameters with the help of one or more IPlayerParameterGenerators. Each IPlayerParameterGenerator is responsible for determining the value of one player parameter using the current state of the central Events and PlayerParameters instances. There is one IPlayerParamerGenerator for each player parameter in PlayerParameters. That is, the system includes a ObstacleEvasionGenerator, a WallEvasionGenerator and a CollectableCollectingGenerator. Each of these classes is responsible for generating the value of the player parameter corresponding to its name.

Note that one could create more classes for generating the value for one parameter to create different behaviors. For example, one may develop different game types that require different logic for adjusting the player parameters or one may want to adjust the logic while the player is progressing during one game session at runtime. This modular design would allow easily changing the way in which parameter values are determined in such cases.

Also, because the Events instance that is passed to each IPlayerParameterGenerator contains all events that have occurred throughout the game, the player parameter generation can be made as complex or as simple as one desires. This, along with the modular design, allows one to create simpler designs to start off with and create more complex and sophisticated designs when the project matures, without changing existing code.

6.3. LEVEL PARAMETERS

Game levels are constructed by `LevelBuilder`s. These `LevelBuilder`s take level generation parameters as arguments for building levels. The design of the level generation parameters is described in Section 4.3.

The parameters are passed to a `LevelBuilder` in the form of a `LevelParameters` instance, which maintains the state of all level generation parameters. Level parameters include: width of the cave, difficulty of cave sections, number of obstacles and difficulty of collectable placement. This structure and the rest of the structures described in this section is shown in the UML class diagram in Figure 6.3. The UML class diagram of the whole system can be found in Appendix D.

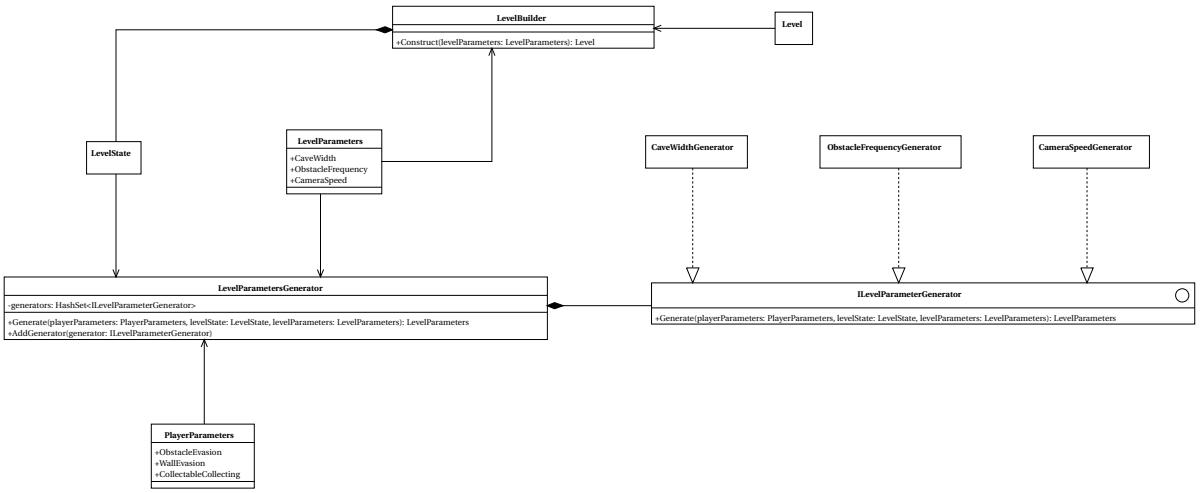


Figure 6.3: UML class diagram showing the structure of the level parameter classes

The level generation parameters that will be used can be specified manually, but they may also be generated dynamically by some other module. They also need not be the same during the whole game session and may be changed at each point during gameplay. When testing the game during development, static parameters may be desired, but when deploying the game, one or more parameters may need to be dynamic in order to create an adaptive experience for the player.

In the system's current state, level generation parameters are generated based primarily on the state of a `PlayerParameters` instance, which has been described in Section 6.2. The level generation is handled by a `LevelParametersGenerator`, which generates level generation parameters in the form of a `LevelParameters`, based on an existing `LevelParameters` instance, a `PlayerParameters` instance and a `LevelState` instance. An instance of `LevelState` maintains several parameters that store the state of a level. Level state parameters may include the time until the next tranquil zone (see Section 4.1.3) and the total time played during the game session.

The `LevelParametersGenerator` generates the level generation parameters using one or more `ILevelParameterGenerator`s. This structure is about the same as the structure described in Section 6.2 for generating player parameters. Examples of `ILevelParameterGenerator`s that are included in the system are `CaveWidthGenerator`, `ObstacleFrequencyGenerator` and `CameraSpeedGenerator`.

Each `ILevelParameterGenerator` is responsible for generating the value for one level generation parameter based on the existing `LevelParameters`, the `LevelState` and the `PlayerParameters`.

In the state of the system, there is one `ILevelParameterGenerator` for each level generation parameter specified in `LevelParameters`. Like with the `IPlayerParameterGenerator`s in the player parameters generation case, the amount of `ILevelParameterGenerator`s included in the system for each level generation parameter is not fixed.

6.4. CHUNK PLACEMENT

Chunks are placed in the level depending on the camera's viewport (i.e. the screen's view as seen by the player). The right bound of the camera is used as an indicator of how far the player can 'see'. Depending on the defined amount of pre-computed placement the chunks may be placed earlier in advance. This results in more chunks being loaded in memory. Deletion of the chunks is also affected by the same amount of pre-computed placement. For instance, changing the value of the pre-computed placement to 1.5 would result

in 1.5 extra chunks being placed in advance relative to what the player would see in his view.

Every chunk has a different point in height at the starting point and ending point (i.e. the height at the left and right edge of the chunk). By retrieving the most left edge of the chunk, done by measuring the x-coördinates, we can place them at the correct height compared to the previously placed chunk. Repeating this process combined with the defined characteristics (see Section 4.3.1) the placement of chunks can be controlled in terms of cave width. The game engine also handles collision detection by means of constructing a polygon collider around the chunk.

6.5. MARKER PLACEMENT

Every chunk contains markers used for the generation of the real objects. These markers function as a potential location of the object. Placement of these objects is controllable by means of configurable odds. In each chunk, a collection of all different types of markers is being tracked. Every collection has a configurable field that determines the amount of maximum markers being used. On top of this, a placement chance is defined for each type of marker. In a chunk with, e.g. 5 hazard markers, setting the amount of hazard markers used to 2 and setting the chance of the marker placement to 50% will cause the generator to randomly pick 2 out of the 5 markers and place an actual hazard at the locations of those markers with a chance of 50% for each marker.

7

FIELD TESTING

To test a game using unit tests will not give a lot of information about the quality of the gameplay. Therefore, at the start of the project, we had planned to do a couple of field tests. One of them was at a dental hygienist to test if the game is sufficiently distracting for the patient but not for the dentist, and to test if the other requirements (see Section 2.3) are met. The other test was at the IFoT to determine if the adaptability of the game works and if players are enjoying the game.

In the next sections we will describe what we wanted to test, the setting of the two tests, how we tested this, what the results were, and what the implications for our project were.

7.1. TESTS AT THE DENTAL HYGIENIST

There were a couple of goals we wanted to achieve with the game that we wanted to test at the dental hygienist:

- The game is sufficiently distracting, so that it may distract patients during a treatment to reduce fear and anxiety.
- It is important that the dentist is not impaired by the patient playing the game, and the dentist has to be able to communicate with the patient at all times during the treatment.
- The game is not too exciting so that the patient does not lose his calm during the treatment by playing the game.
- Setting up and starting the game should not take too much time.

These tests took place at the Mondhygiënisten Sweelinckplein practice in The Hague. Here we let three adult patients of Sander Roeleveld play the game during a dental treatment (see Figure 7.1), while one of us was present to observe, and asked them to complete a short survey afterwards (see Appendix F), additionally we asked for feedback from Sander on how it was to treat a patient while the patient was playing the game.

The results that we found:

- We observed that setting up the game and getting the patients started with the game did not take a lot of time, no more than a few minutes, which was acceptable for Sander. Correctly adjusting the Vuzix goggles took the most time.
- The patients indicated that during the treatment they were thinking less about the treatment than normal, this indicates that the game provides distraction from the treatment. One of the patients was distracted enough not to notice she had started bleeding during the treatment.
- While one of the patients made some noises when hitting objects in the game, overall they remained calm and did not move more than normal.
- Sander was able to communicate with the patients at all times, although he indicated that he missed the ability to have eye contact with the patient while the patient was playing the game.

Figure 7.1: Gaming at the dentist.

What we took away from the tests was that our goals were accomplished, that it could be a good idea to give the player the possibility to pause the game, and that there should be a way to warn the dentist that the patient is feeling/becoming unwell.

7.2. TESTS AT THE IFOT

The goal of the user tests at the IFoT was to test:

- The player model in combination with the DDA, does the game correctly adjusts its difficulty to the skills of the player?
- The gameplay, do the players think the game is fun?

The IFoT is a festival in Delft where innovative technology and art are displayed and music is played. We were invited to display our project here and we used this opportunity to test our game. Around 40 people varying in age from 7 to 53 played our game here. The game was played while sitting on an old dentist chair and with the goggles on to resemble how the game would be played during a visit to the dentist (see Figures 7.2 and 7.3). The duration of the game was set to three minutes so that people did not have to wait too long. After participants played the game we asked them to complete a short survey (see Appendix G). We used the survey to ask about the perceived difficulty of the game, the fairness of the game, and the enjoyment the participant experienced. In addition, we also asked them to identify the element they enjoyed most about the game and the one thing about the game they would change.

The results that we found:

- Most of the players enjoyed the game (see Figure 7.4).
- The participants were divided on how challenging the game was, some saying the game was relatively easy and others rating it as challenging (see Figure 7.5).

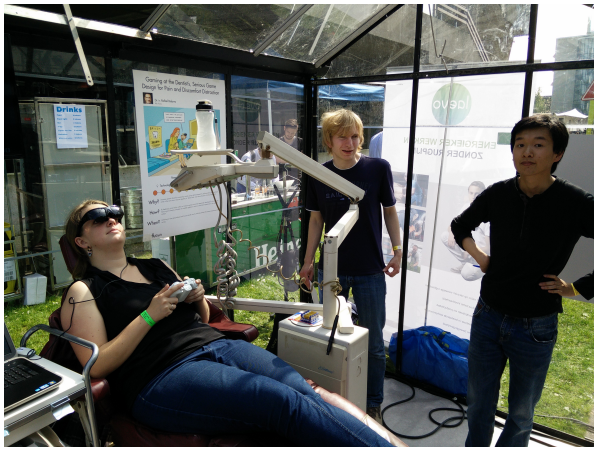


Figure 7.2: At the IFoT in the afternoon.

Figure 7.3: At the IFoT in the evening.

- Most of the participants thought the game was fair (see Figure 7.6).
- The most common complaint was that the oxygen bubbles were not visible enough.
- Scoring points by collecting coins was the most common most enjoyable thing.

We found that most of the players thought that the game was fun to play, which was a good sign. Although, because the duration of the game was only three minutes, we cannot be sure that the game would still have been fun for everyone over a longer duration. The oxygen bubbles were difficult to see, especially when playing with the goggles on, so they should be made more visible. Almost everyone thought the game was neither overly challenging nor easy, while the player model was still rather simple this indicated that the player model and DDA worked decently but not great yet and would have to be revisited.

Figure 7.4: The enjoyment score with which the participants rated the game. The scale is from 1, which means 'Not at all', to 5, which means 'Very much'.

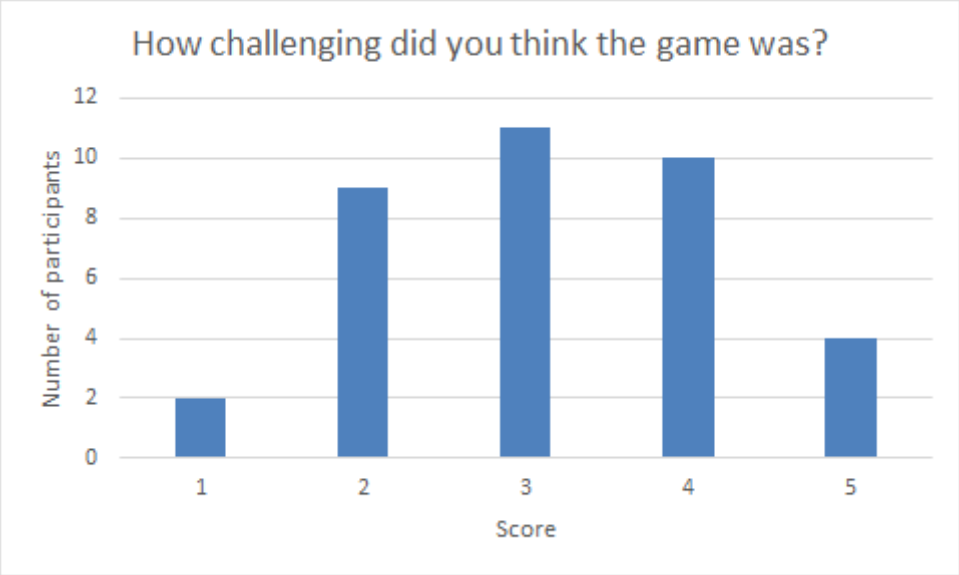


Figure 7.5: The challenge score with which the participants rated the game. The scale is from 1, which means 'Far too easy', to 5, which means 'Very challenging'.

Figure 7.6: The fairness score with which the participants rated the game. The scale is from 1, which means 'Not fair at all', to 5, which means 'Very fair'.

8

PROCESS EVALUATION

In this chapter we will describe the process of our project and we will evaluate this process. We will tell about the planning, the software development methodology, testing, version control, communication, and the feedback from the Software Improvement Group (SIG).

8.1. COMMUNICATION

The most important parties in communication in the project were the team (i.e. Björn, Rolf, and Owen), the Coach (Rafael), and the Client (Rob). Other important parties that we have been in contact with during our project were a game designer (Dylan) to discuss game design and receive some feedback on the game during our process, the dental hygienist (Sander) who helped us test the game in a dental practice, and one of the members of the organization of the IFoT (Marloes) to coördinate our participation in the festival.

8.1.1. INTERNAL COMMUNICATION

For the communication within the team we made use of Slack. A Trello [1] board (see Figure 8.1) was created to keep track of the tasks we had created. Most of the days we worked together on a central location (e.g. TU Delft Library or the EEMCS faculty). Working on a central location was a good idea since it made it easier to assure everyone was making progress, helped to quickly solve emerging problems, and it increased communication about important decisions.

8.1.2. EXTERNAL COMMUNICATION

For our external communication we appointed one member (Rolf) as the contact person. At the start of the project we planned that we would have weekly meetings with Rafael and Rob (if he was available) on Fridays at 14:30. Via email we kept Rafael and Rob up to date with weekly updates with the progress of that week and our planning for the next week. We also used Slack to communicate with Rafael for short questions. Rafael brought Rolf into contact with Dylan, Sander, and Marloes.

8.2. TEAMROLES

We had the following roles and division of the roles within our team:

Björn: Lead developer, lead tester.
Owen: Lead artist, Scrum master.
Rolf: Lead communication, producer, lead game designer.

Since we mostly worked at the same location a lot of our decisions have been made as a team.

8.3. EXPERIENCED PROBLEMS

There were a couple of things that took a lot more time than we expected. We will shortly describe the problems in that area.

Unity uses 'Prefabs' as a template from which to create new game object instances in the game. However,

[1] www.trello.com

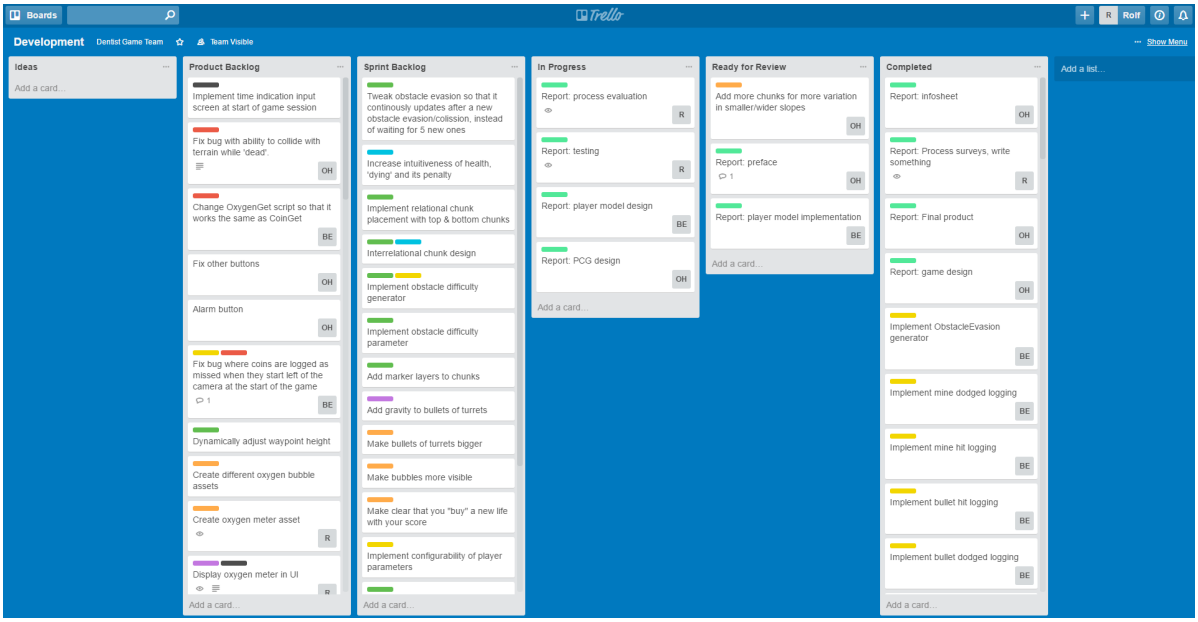


Figure 8.1: Our Trello board.

if you store a Prefab within another Prefab and subsequently alter it, it will not be updated in the other Prefab. So, if you want to change all the instances of a Prefab that have been placed in other Prefabs you have to do this manually. Since our chunks were prefabs that contained other prefabs (e.g. coins, mines) this happened a lot. Later on we used a way to work around this by placing a 'marker' Prefab we created that instantiated other Prefabs.

While having the code from the prototype was useful, to facilitate our level generator we had to create a lot of the assets (mainly chunks) for the game ourselves. Creating new chunks complete with carefully placed contours took a lot of time, especially since several times changes had to be made due to new design choices. One of those was the decision to place the contours further back to create more depth in the game, this had been decided during the mid-term meeting.

In retrospect it would have been wise to not have the tests at the dental hygienist in the same week as the IFoT, since this would have given us more breathing room and a chance to react to unexpected events.

We also had some trouble with a part of our player model implementation, because it took a while to find out that we had different visions across-the-board, both within the team and between the team and the Client and the Coach. This took a while, because in week 6 we missed a meeting with the Client and the Coach due to some miscommunication, and in week 7 we were very busy with preparations for and standing on the IFoT and the tests at the dental hygienist. This meant a lot of time of time had to be spent revising the player model we already implemented.

8.4. SOFTWARE DEVELOPMENT METHODOLOGY

In this section we will describe the method we used for our software development, which tools we used to develop our software, and the feedback from the SIG and how we used this feedback to improve our code.

8.4.1. SCRUM

At the start of the project we decided to use the Scrum method. We decided to use Scrum because we were familiar with it and because it felt appropriate for this project, since in our previous experiences it worked well with similar projects. Our sprints lasted one week and the planning and evaluation of the sprints were done on Thursday. We used Trello to maintain our product backlog, sprint backlog, and the tasks that are in progress, ready for review, or completed (see Figure 8.1). Overall the Scrum method worked well for our project. However, because of a number of holidays (e.g. Kings Day) and some test weeks, a lot of the sprints were really short. Because of this we think it would have been better if we had planned sprints of two weeks instead of one week.

8.4.2. VERSION CONTROL

Version control was done with Git, using a GitHub repository as our central repository. We implemented a model such as in Figure 8.2. Each member created feature branches to implement new features which were reviewed by at least one other member before being merged into the develop branch. New release branches were created before important moments such as the tests at the dentist, the test at the IFoT, and before delivering to the SIG. This model helped us create stable releases before important deadlines and to keep a good overview of the code.

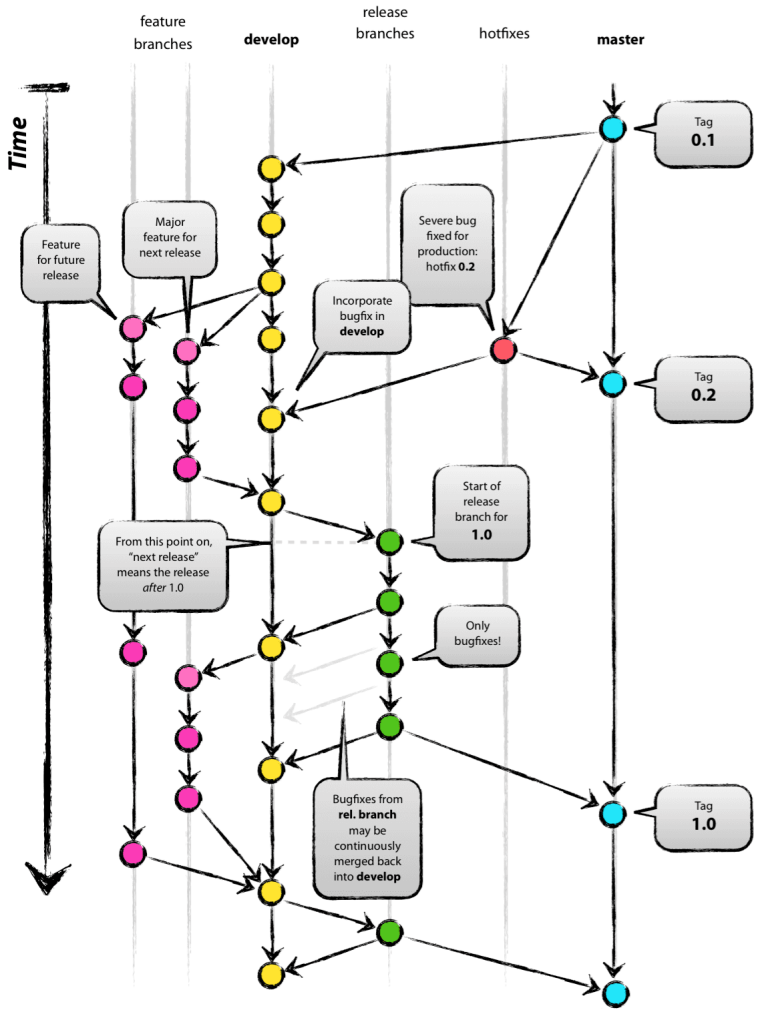


Figure 8.2: Git version control model [2].

8.4.3. CODING TOOLS

Coding was done using C# and Unity was used as the game engine. This was done since these were requirements from the Coach and the Client. Since the dentist game prototype was also developed using Unity this was very practical. Visual Studio was used as integrated development environment because of the integration with Visual Studio that Unity offers.

8.4.4. SIG FEEDBACK

The analysis of the SIG showed that we scored four out of five stars in their maintainability-model. What held us back from achieving five stars was a lower score on Duplication and Unit Complexity. This means that some of our code appeared on multiple places and that some of the methods we used were too complex (the examples in our code that they pointed out were methods that were too long). The duplication happened

[2] http://nvie.com/posts/a-successful-git-branching-model

because C# does not allow for type co-variance and our work around caused this code duplication. After the feedback we have revised our complex methods to create easier code that is easier to maintain and understand. SIG also mentioned that they did not find any (unit) test-code. We have since added some unit tests in order to test some of the key aspects of our game. However, since the interaction between our code and Unity was hard to test using (unit) test-code, we have also done a lot of playtesting ourselves.

9

Conclusion

The last few months we have worked on creating a game that offers a personalized gaming experience and that can be played while being treated by a dentist. To achieve this we have used the previous work that was available and adapted and expanded on this with the help of our Coach, Client and game designer Dylan, who had also helped with the game design of the previous version of the game.

The test at the dental hygienist helped us to test our non-functional requirements in a practical setting and here we found that our project satisfied the requirements that were initially set in Section 2.3. Some additional feedback has been incorporated in Chapter 10.

We were able to implement all the 'Must Haves' and most of the 'Should Haves' that we had.

We had a great opportunity to test our game at the IFoT, where we found that the game was fun to play with the assets and elements we had added. We have used the feedback from the IFoT and the feedback of the Coach and the Client to improve on our player model and made it easily tweakable to make the level generation more flexible.

In spite of some troubles and unexpected time sinks during the project we have been able to successfully create a procedural level generator and a player model structure that together help create a gaming experienced tailored to every individual player's skills. The resulting game can be played while being under treatment at a dentist without disturbing said treatment, while still distracting the patient from the treatment.

10

FUTURE RECOMMENDATIONS

More work could be done to improve the current version of the game. In this chapter we give some recommendations for future work.

10.1. ALARM BUTTON

Feedback we received from the dental hygienist was that due to the loss of eye contact and the loss of view of a large part of the patient's face because of the Vuzix goggles it is now a lot harder to see if a patient is becoming unwell. While this does not happen often, it is still important to be able to react as quickly as possible when something does occur. The hygienist's suggestion was to implement a sort of 'alarm button' that the patient can press when something does go wrong. This would help in some situations but in others the patient might not feel it coming and would be unable to press the button in time, or just not remember to press the button. Nevertheless this is an important issue that should be addressed when further developing the product.

10.2. PLAYER MODEL

We have not had the time to thoroughly test the player model and the parameters thereof. Therefore, future work should incorporate testing of the player model and tweaking of the parameters if this appears to be necessary to improve the game.

10.3. GAMEPLAY ELEMENTS

Some of the feedback we received during our tests was about the style of the game and about adding additional features that could improve the game. In addition to this there were also several ideas of our own that we have not been able to implement (see Appendix E). This feedback and these ideas could be incorporated in future versions of the game.

10.3.1. TRANQUIL ZONES

One of the prominent ideas was to add some periods of rest for the player and create a less linear experience was to add 'tranquil zones'. In these tranquil zones the cave would be wider and the camera would slow down compared to the normal level, and the player would be able to collect a lot of coins. The level would become more difficult while approaching a tranquil zone and after leaving the tranquil zone the difficulty would be a little easier than it was just before entering the tranquil zone.

10.3.2. ART

Adding more color to the game could be a good way to increase the visual attraction of the game and has been suggested by a few of the participants of our tests. In addition, the game could be made more lively by having sea-life such as whales or small exotic fish swim around in the background.

10.3.3. PUZZLE ELEMENT AND INCREASED VARIETY IN ENEMIES

Adding letters of a word in the game that the player can collect to form a word relating to the underwater theme could pique the player's interest and thereby increase immersion in the game. Adding a larger variety

of enemies could also help improve the gameplay. This would be true especially when the game is played during longer sessions, because this would mean it would take longer to become familiar with each individual enemy, since less would appear within the same time frame.

10.3.4. PHYSIOLOGIAL DATA

Using physiological data of the player such as the heart rate or perspiration rate for the player model could be great if there is a way to adjust the gameplay in such a way that it can help decrease anxiety or stress of a patient when heart rate and perspiration rate became higher than wanted. Additionally, this could help to give the dentist an indication of the state of the patient.

BIBLIOGRAPHY

[1] Rafael Bidarra, Dien Gambon, Rob Kooij, Dylan Nagel, Maaike Schutjes, and Ioanna Tziouvara. Gaming at the dentist's–serious game design for pain and discomfort distraction. In *Games for Health*, pages 207–215. Springer, 2013.

[2] Darryl Charles and Michaela Black. Dynamic player modeling: A framework for player-centered digital games. In *Proc. of the International Conference on Computer Games: Artificial Intelligence, Design and Education*, pages 29–35, 2004.

[3] Jenova Chen. Flow in games (and everything else). *Communications of the ACM*, 50(4):31–34, 2007.

[4] Heather Desurvire, Martin Caplan, and Jozsef A Toth. Using heuristics to evaluate the playability of games. In *CHI'04 extended abstracts on Human factors in computing systems*, pages 1509–1512. ACM, 2004.

[5] Joris Dormans. Adventures in level design: generating missions and spaces for action adventure games. In *Proceedings of the 2010 workshop on procedural content generation in games*, page 1. ACM, 2010.

[6] Melissa A Federoff. *Heuristics and usability guidelines for the creation and evaluation of fun in video games*. PhD thesis, Citeseer, 2002.

[7] Robin Hunicke. The case for dynamic difficulty adjustment in games. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 429–433. ACM, 2005.

[8] Bill Kapralos, Faizal Haji, and Adam Dubrowski. A crash course on serious games design and assessment: A case study. In *Games Innovation Conference (IGIC), 2013 IEEE International*, pages 105–109. IEEE, 2013.

[9] Hannu Korhonen and Elina MI Koivisto. Playability heuristics for mobile games. In *Proceedings of the 8th conference on Human-computer interaction with mobile devices and services*, pages 9–16. ACM, 2006.

[10] Emily F Law, Lynnda M Dahlquist, Soumitri Sil, Karen E Weiss, Linda Jones Herbert, Karen Wohlheiter, and Susan Berrin Horn. Videogame distraction using virtual reality technology for children experiencing cold pressor pain: The role of cognitive processing. *Journal of pediatric psychology*, page jsq063, 2010.

[11] Ricardo Lopes, Ken Hilf, Luke Jayapalan, and Rafael Bidarra. Mobile adaptive procedural content generation. In *Proceedings of the fourth workshop on Procedural Content Generation in Games (PCG 2013), Chania, Crete, Greece*, 2013.

[12] Thomas W Malone. What makes things fun to learn? heuristics for designing instructional computer games. In *Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems*, pages 162–169. ACM, 1980.

[13] LE Nacke, Anders Drachen, and Stefan Göbel. Methods for evaluating gameplay experience in a serious gaming context. *International Journal of Computer Science in Sport*, 9(2):1–12, 2010.

[14] Jeanne Nakamura and Mihaly Csikszentmihalyi. The concept of flow. In *Flow and the Foundations of Positive Psychology*, pages 239–263. Springer, 2014.

[15] Christopher Pedersen, Julian Togelius, and Georgios N Yannakakis. Modeling player experience for content creation. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(1):54–67, 2010.

[16] David Plans and Davide Morelli. Experience-driven procedural music generation for games. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(3):192–198, 2012.

[17] Noor Shaker, Georgios N Yannakakis, and Julian Togelius. Towards player-driven procedural content generation. In *Proceedings of the 9th conference on Computing Frontiers*, pages 237–240. ACM, 2012.

[18] Adam M Smith, Chris Lewis, Kenneth Hullet, and Anne Sullivan. An inclusive view of player modeling. In *Proceedings of the 6th International Conference on Foundations of Digital Games*, pages 301–303. ACM, 2011.

[19] Tarja Susi, Mikael Johannesson, and Per Backlund. Serious games: An overview. 2007.

[20] Julian Togelius, Renzo De Nardi, and Simon M Lucas. Making racing fun through player modeling and track evolution. 2006.

[21] Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. Search-based procedural content generation. In *Applications of Evolutionary Computation*, pages 141–150. Springer, 2010.

[22] Julian Togelius, Noor Shaker, and Mark J. Nelson. The experience-driven perspective. In Noor Shaker, Julian Togelius, and Mark J. Nelson, editors, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2015.

[23] Julian Togelius, Noor Shaker, and Mark J. Nelson. Grammars and l-systems with applications to vegetation and levels. In Noor Shaker, Julian Togelius, and Mark J. Nelson, editors, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2015.

[24] Julian Togelius, Noor Shaker, and Mark J. Nelson. Introduction. In Noor Shaker, Julian Togelius, and Mark J. Nelson, editors, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2015.

[25] Brenda K Wiederhold, Kenneth Gao, Camelia Sulea, and Mark D Wiederhold. Virtual reality as a distraction technique in chronic pain patients. *Cyberpsychology, Behavior, and Social Networking*, 17(6): 346–352, 2014.

[26] Georgios N Yannakakis and Julian Togelius. Experience-driven procedural content generation. *Affective Computing, IEEE Transactions on*, 2(3):147–161, 2011.

[27] Alexander Zook and Mark O Riedl. A temporal data-driven player model for dynamic difficulty adjustment. In *AIIDE*. Citeseer, 2012.

Acronyms

AI Artificial Intelligence.

ANN Artificial Neural Network.

CDPCG Context-Driven Procedural Content Generation.

DDA Dynamic Difficulty Adjustment.

EDPCG Experience-Driven Procedural Content Generation.

IFoT International Festival of Technology.

PCG Procedural Content Generation.

PEM Player Experience Modelling.

SBPCG Search-based Procedural Content Generation.

SIG Software Improvement Group.

A

Software Improvement Group evaluation

The feedback received from the SIG is listed in the following two sections. The feedback reflects on the software quality of the product and is taken into account in the project.

First evaluation

“De code van het systeem scoort 4 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Duplication en Unit Complexity.

Voor Duplicatie wordt er gekeken naar het percentage van de code welke redundant is, oftewel de code die meerdere keren in het systeem voorkomt en in principe verwijderd zou kunnen worden. Vanuit het oogpunt van onderhoudbaarheid is het wenselijk om een laag percentage redundantie te hebben omdat aanpassingen aan deze stukken code doorgaans op meerdere plaatsen moet gebeuren. Er zijn in dit project meerdere duplicaties te vinden: bijvoorbeeld een duplicatie van 15 lines of code is te vinden tussen de bestanden CollectableCollectedEvent.cs en CollectableMissedEvent.cs. Het is aan te raden om dit soort duplicaten op te sporen en te verwijderen.

Voor Unit Complexity wordt er gekeken naar het percentage code dat bovengemiddeld complex is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, makkelijker te testen is en daardoor eenvoudiger te onderhouden wordt. Een voorbeeld van een complexe methode is Health.FixedUpdate of CameraScript.determineCameraDirection. Opsplitsen van een methode in kleinere methodes of herschrijven van een methode met een meer generieke implementatie helpt om de code complexity te verbeteren.

Als laatste nog de opmerking dat er geen (unit)test-code is gevonden in de code-upload. Het is sterk aan te raden om in ieder geval voor de belangrijkste delen van de functionaliteit automatische tests gedefinieerd te hebben om ervoor te zorgen dat eventuele aanpassingen niet voor ongewenst gedrag zorgen.”

Second evaluation

“In de tweede upload zien we dat zowel de omvang van het systeem is gestegen terwijl de score voor onderhoudbaarheid ongeveer gelijk is gebleven.

Verbetering is te zien bij de score voor Unit size en Unit complexiteit. De score voor Duplication daarentegen is verlaagd en er zijn nieuwe duplicaties geïntroduceerd.

Wat betreft de testcode is het goed om te zien dat jullie, in tegenstelling tot bij de eerste upload, nu wel testcode hebben geschreven. Vergeleken met de omvang van de productiecode is dit echter nog steeds erg weinig.

Uit deze observaties kunnen we concluderen dat een deel van de aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject.”

B

PROJECT DESCRIPTION

The following project description is taken from the original project description from BepSys.

B.1. PROJECT DESCRIPTION BEPSYS

Gaming is increasingly deployed for a variety of purposes, other than pure entertainment. Among them, a novel application of games in the health domain aims at distracting a patient from the pain and anxiety usually experienced during uncomfortable treatments. A dentist game prototype, developed at TU Delft with this purpose [1], has identified a number of challenges specific for this domain (see also the short video on TU Delft TV: https://http://www.youtube.com/watch?v=NoRi6YM6jIU).

This project will approach and solve one of these challenges: the fact that different patients, with disparate skill and preferences, undergoing treatments with very different durations should be offered a personalized game experience, that takes all those factors into account. This will involve developing a) a complete, fully controllable procedural game level generator, b) a player model manager that monitors and captures all relevant player data, and c) a time context-dependent gameplay manager able to (up/down-) scale the game experience according to a prescribed treatment's duration [2].

Together, these modules will steer the in-game level generation in order to dynamically adjust to each particular player and treatment. In the background, all player profiles and scores are maintained and reassessed, in order to guarantee the continuity and improvement of the game experience for each patient throughout a series of treatments.

All features will be implemented in C# and tested standalone, before being integrated into a new version of the dentist game prototype, developed in Unity3D. In addition, both the level generator and the player model will have to be validated in practice, including field tests to be performed with real patients in a real dentist clinic setting. Fortunately, we have various very enthusiast and specialized dentists in our development team, who will be more than glad to facilitate that phase.

[1] Bidarra R, Gambon D, Kooij R, Nagel D, Schutjes M, Tziouvara I (2013) Gaming at the dentist's serious game design for pain and discomfort distraction. Proceedings of Games for Health Europe, 27-28 October, Amsterdam, The Netherlands

[2] Lopes R, Hilf K, Jayapalan L and Bidarra R (2013) Mobile adaptive procedural content generation. Proceedings of PCG 2013 - Workshop on Procedural Content Generation for Games, co-located with the Eighth International Conference on the Foundations of Digital Games, 1417 May, Chania, Crete, Greece

B.2. COMPANY DESCRIPTION

TU Delft - Computer Graphics and Visualization Group

C

INFOSHEET

GENERAL INFORMATION

Project title: Personalized Adaptive Dentist Game
Client organisation: Delft University of Technology
Presentation date: July 1, 2016

DESCRIPTION

The Delft University of Technology had developed a prototype dentist game to distract patients during dental treatment, specifically designed for uncomfortable treatments. However, this prototype was not suitable yet to be played for different lengths of treatments. Additionally, the fact that different patients have different skills and preferences needed to be addressed by personalizing the game experience. The Delft University of Technology wanted the prototype game to be extended by designing a procedural game level generator, generating content depending on the time-context and a player model which captures relevant player data. During the research phase it became apparent that the most challenging aspect of the design would be retaining a natural feeling of the level while content is generated based on the player model. The project was carried out using the Scrum methodology. The end product is a fully controllable procedural game level generator driven by a player model. The time duration of the game is set prior to starting and is then used alongside the player model to steer the level generator. The game was tested during the project at a dental hygienist with several patients and a second time at the International Festival of Technology. There is still great potential to expand upon the game. More gameplay elements such as additional hazards and enemies can be added. Extending the player model by means of capturing physiological data such as heartbeat, heat signatures and other factors may greatly improve the profile of the players.

TEAM MEMBERS

All team members have cooperated in the division of labour, planning, research report, final report and presentation. Major design decisions were always made together. Several special roles were performed by a single person, while others were performed by every team member. **Rolf** had taken care of all communication outside the team. **Owen** was responsible for all the stylizing of the game and **Björn** maintained the overview of the code structure.

CLIENT
R.E. Kooij
Delft University of Technology, Department of Intelligent Systems, Network Architectures and Services group

COACH
R. Bidarra
Delft University of Technology, Department of Intelligent Systems, Computer Graphics and Visualization group

CONTACT
Rolf Starre, e-mail: *rolf.starre@gmail.com*

The final report for this project can be found at: `http://repository.tudelft.nl`

D

Full UML

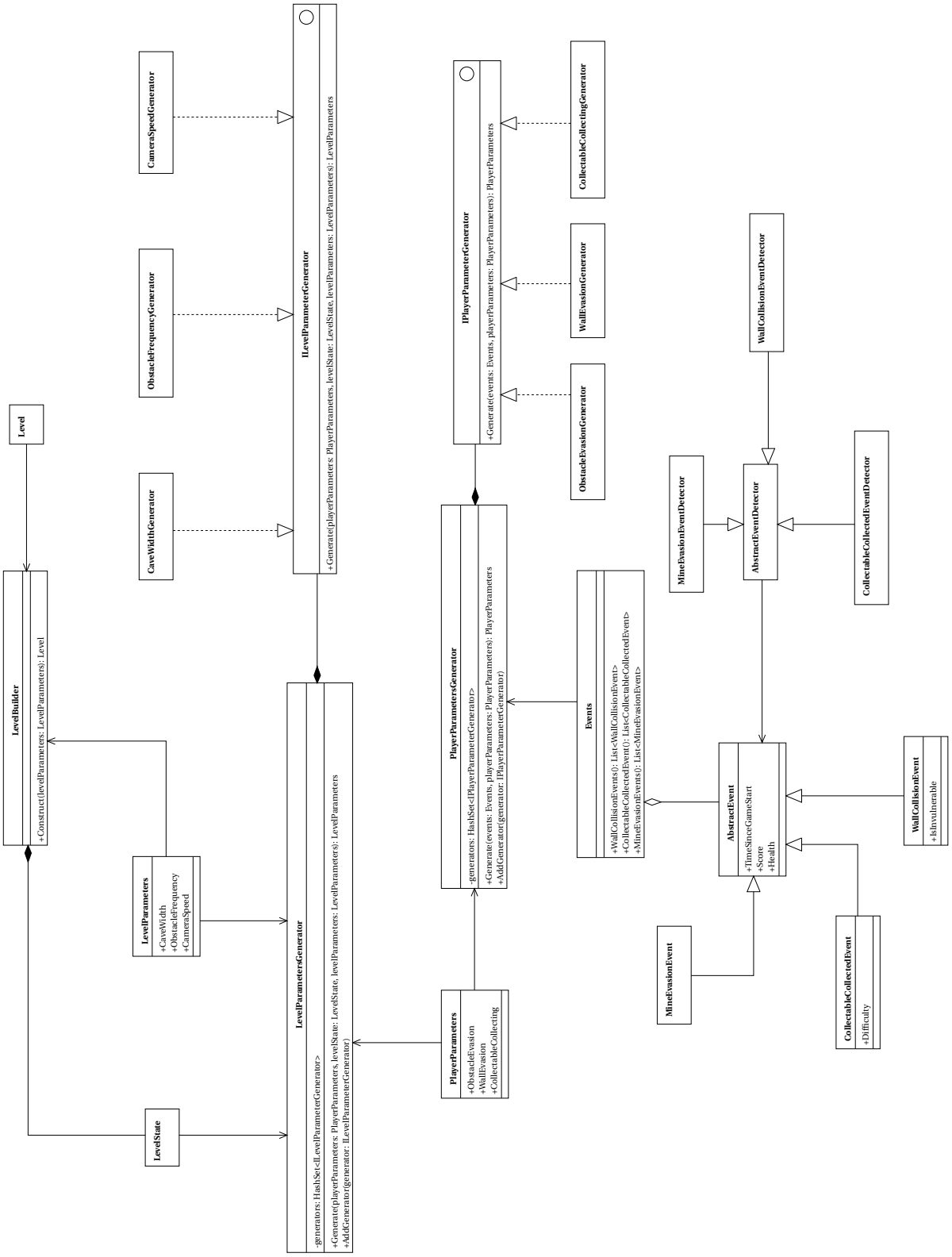


Figure D.1: UML class diagram showing the structure of the whole system

E

GAMEPLAY FEATURE LIST

Features in terms of gameplay are categorized in order of importance using the MoSCoW model, similar to the requirements in Section 2.3. Must haves are definitely included in the game. Should haves will most likely be implemented and could haves are merely a nice addition to the game to add extra depth to the gameplay. The motivation behind the game features is also described.

E.1. MUST HAVE

- Endless right-scrolling level design
  - Motivation: This allows the game to be easily scaled to different game lengths (time). It also fits well with the idea of surfacing at the end of the game.
- User is pushed forward when at the left-most side of the screen instead of being damaged.
  - Motivation: Replaces the old aspect of the game in the prototype, where the player gets damaged. This feels more intuitive and makes the game easier in the beginning.
- The game becomes more difficult when the player performs well and becomes easier when the player does poorly.
  - Motivation: The game dynamically adapts to the skill set of the player based on the constructed player model to find a balance between challenge and enjoyment.
- Gravity on submarine: the submarine is slightly pulled down at all times by gravitational force.
  - Motivation: Provides incentive for the player to keep moving.
- Collectables: coins, pearls, and diamonds that give points.
  - Motivation: Gives short term goals to the player.
- Game over: The game resumes at the same point where the player dies. Upon death, a score deduction of 5000 will be applied to the current score.
  - Motivation: To give a transition between going game over and continuing in a calm way. Also actively punish bad performance to incentivize learning.

E.2. SHOULD HAVE

- Tranquil zones: Between parts of the level there will be zones where the player will have a moment of rest. Flags are placed showing the distance left to the ocean (end) inside the tranquil zones.
  - Motivation: Passing these zones will give the player a sense of progress and give a goal to work towards. Furthermore, gives a rhythm to the game of completing 'levels'. Also, gives a moment to relax.
- Oxygen tank meter: The meter decreases very slowly over time (every second a deduction of 1%). The meter starts at 100% and is depleted at 0%. Upon depletion of the meter the 'death' animation is shown.
  - Motivation: Gives the player a short term goal.

- Collision with impassable terrain or hazards deduct oxygen from the oxygen meter.
  - Motivation: Has an effect on oxygen meter to avoid cluttering the screen with additional meters. Additionally, gives an incentive to avoid hitting walls/objects.
- Collectables: Oxygen bubbles, these bubbles fill the oxygen meter. Different kinds ranging from 5%/25%/75%.
  - Motivation: Gives the player a way to replenish their oxygen to prolong life.
- Hazards: Mines, these cause the player to lose oxygen (15%) on collision.
  - Motivation: Increases difficulty of the game, providing more challenging aspects to navigation and dodging of objects.
- Enemies: Seed plant, shoots seeds in fixed intervals aimed at the player. Collision with a seed deducts a small amount of oxygen from the oxygen meter.
  - Motivation: Deepens gameplay by expanding variety in dangerous elements of the game.
- Pause game: The game can be paused by pressing the 'ESC' key. An overlay with the word (paused) is shown.
  - Motivation: Whenever needed, the dentist is able to pause the game.

E.3. COULD HAVE

- Ghost example submarine: The ghost submarine shows the player what to do in the very beginning of the game. It shows possible movements the player can make and what the controls are.
  - Motivation: Functions as an 'introduction' and shows what controls are necessary for the game.
- Collectables: Letters forming a word. If the player misses a letter, the same letter is repeated again later on.
  - Motivation: Provide mission based objectives throughout the game. This will give the player an additional goal.
- When player runs out of oxygen, another NPC submarine approaches the player and gives oxygen (this is an animation).
  - Motivation: Part of the 'game over', this is to provide a smooth transition between dying and 'respawning'.
- Combo modifier: The combo modifier will be displayed on screen. It starts at 1.0 and is capped at 5.0. This means that treasures are worth 5 times more than usual at a combo modifier of 5.0. The combo modifier influences the amount of points gained by picking up treasures and other positive objectives by using the formula: score increase = combo modifier * object value. For every treasure the combo meter will increase. The combo modifier resets to 1.0 when the player misses a treasure.
  - Motivation: Creates incentive to collect more treasures and generally perform well (dodging hazards, not hitting walls etc.). Capping the combo meter is to prevent an unlimited combo modifier to limit frustration when a player loses a high combo modifier.
- Vicious plants: these are harmful entities that try to hurt the player by shooting projectiles or other harmful substances.
  - Types of enemies: Venomous plant: occasionally releases a poison cloud in a fixed radius around the plant that reverses player controls. Water cannon that occasionally releases streams of water that pushes the player in the direction depending on the angle of the water cannon.
  - Motivation: To provide additional challenge to players depending on their strengths in various game aspects, such as dodging.
- Hazards
  - Proximity mines: these act similar to regular mines but activate after a delay when the player gets close to them.
  - Water stream: accelerates player movement when entered.
  - Motivation: Proximity mines add more depth to the mines. The water stream adds more overall variation in controlling the submarine.

F

DENTIST SURVEY

Videogames bij de tandarts

Ik ben ….. jaar oud.

Ik ben een: man/vrouw

Cirkel het nummer aan waarmee je het meest eens bent.

1. Ik ben bang voor de tandarts.

1 helemaal niet | 2 | 3 | 4 | 5 heel erg

2. Ik vond het spel leuk!

1 helemaal niet | 2 | 3 | 4 | 5 heel erg

3. Ik zou erg graag naar de tandarts willen gaan als ik dit spel kan spelen.

1 helemaal niet | 2 | 3 | 4 | 5 heel erg

4. De tijd leek voorbij te vliegen!

1 helemaal niet | 2 | 3 | 4 | 5 heel erg

5. Ik dacht minder dan normaal aan de tandarts behandeling.

1 helemaal niet | 2 | 3 | 4 | 5 heel erg

G

IFoT survey

Dentist game

Please circle or fill in the applicable below.

I am: ☐ male ☐ female
My age: ____ years
In a typical week, I play video games for: ____ hours

1. How much did you enjoy the game?

1 2 3 4 5

Not at all *Very much*

2. How challenging did you think the game was?

1 2 3 4 5

Far too easy *Very challenging*

3. How fair did you think the game was?

1 2 3 4 5

Not fair at all *Very fair*

4. What was the most enjoyable thing about the game?

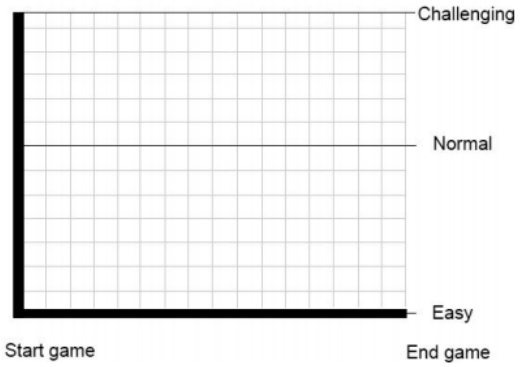
.....

5. If you could change ONE thing about the game, what would it be?

.....

6. Draw the evolution of the difficulty as you experienced it from start to end.

Difficulty



Thank you for playing!