INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Dynamic Difficulty Adjustment in First-Person Shooters

JOHANN WOLFGANG BOTELHO MACEDO KNORR
Junho de 2021

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Dynamic Difficulty Adjustment in First-Person Shooters

Johann Wolfgang Botelho Macedo Knorr

Dissertation for obtaining the Master's Degree in Informatics Engineering, Specialization Area in Graphics Systems and Multimedia

Supervisor: Carlos Vaz de Carvalho

Porto, June 2021

Instituto Superior de Engenharia do Porto

Dynamic Difficulty Adjustment in First-Person Shooters

Johann Wolfgang Botelho Macedo Knorr

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Sistemas Gráficos e Multimédia

Orientador: Carlos Vaz de Carvalho

Porto, junho 2021

Resumo

Os desenvolvedores de videojogos perceberam que jogadores, com vários níveis de competência, necessitam de desafios diferentes para desfrutar totalmente das suas experiências de jogo. Tradicionalmente, os jogos digitais permitem que o jogador selecione o nível de dificuldade antes de começar o jogo, contudo este método tem o problema de ser demasiado rígido e limitado, dado que muitos jogadores tem a sua zona de conforto entre os níveis existentes.

Neste projeto, analisamos como o *Dynamic Difficulty Adjustment* (DDA) pode ser usado para melhorar a experiência dos jogadores durante as suas sessões de jogo e assim aumentar o valor que um jogo lhes oferece. Após um breve resumo do estado da arte e das suas descobertas, iremos propor a nossa própria solução para o problema. Para isso desenvolvemos um sistema que analisa o comportamento do jogador durante o jogo e ajusta o desafio de acordo. Após a implementação do sistema, realizamos um teste piloto para avaliar se o sistema foi capaz de melhorar a experiência dos jogadores e identificar os principais fatores que permitem os jogadores superar os desafios que oferecemos. Os resultados do estudo demonstram que o sistema é capaz de melhorar a experiência dos jogadores, nomeadamente em termos de desafio, experiência positiva e concentração no jogo.

Palavras-chave: First-Person Shooter, Dynamic Difficulty Adjustment, Digital Game, Flow

Abstract

Video game developers have found that players, with diverse skill levels, require different challenges to fully enjoy their gaming experiences. Traditionally video games allow the player to select a difficulty level before starting the game, however, this method has the problem of being too restricting and limited, as many players have their comfort zone between the levels created by the developers.

In this project, we analyse how Dynamic Difficulty Adjustment (DDA) can be used to improve the experience of the players during their play sessions and so increase the value a game offers them. After a summary of the current state of the art and their findings, we propose our solution to the problem. For that purpose, we developed a system that analyses the player's behaviour during the game and adjusts the challenge accordingly. After implementing the system, we conducted a pilot test to assert if the system was able to improve the experience of the players and identify the factors that allowed the players to overcome the challenges we offered. The results of that study show that the system is capable of enhancing a player's experience, especially in terms of challenge, positive experience, and flow.

Keywords: First-Person Shooter, Dynamic Difficulty Adjustment, Digital Game, Flow

Acknowledgements

First and foremost, I would like to thank my family for helping me become who I am today and for all the support they give me.

I thank my friends at ISEP, Flávio Costa and João Magalhães for all the hours we worked together, pooling our knowledge, and mutual support during the course.

Lastly, I want to thank my supervisor, Carlos Vaz de Carvalho, for guiding and assisting me in the development of this project.

Index

1 Introduction 1

1.1 Objectives 2

2 State of the Art 3

2.1 Dynamic Difficulty Adjustment 4

2.2 Examples of DDA in games 5
2.2.1 Resident Evil 4 6
2.2.2 The Left 4 Dead Series 6
2.2.3 FIFA 7
2.2.4 Mario Kart 8
2.2.5 Overview of the games and their DDA techniques 8

2.3 Technology related to Dynamic Difficulty 9
2.3.1 Hamlet 9
2.3.2 DRE-Bot 10
2.3.3 GoldSource 10
2.3.4 Half-Life 11
2.3.5 GameAnalytics 11

2.4 Conclusion 11

3 Value Analysis 13

3.1 New Concept Development Model 13

3.2 Value Proposition 14

3.3 Porter's Value Chain 15

3.4 Business Model Canvas 16

3.5 Analytic Hierarchy Process 19
3.5.1 Establishment of the criteria hierarchy 20
3.5.2 Allocation of weights to the criteria 21
3.5.3 Assignment of numerical values and calculation of the final grade 23
3.5.4 Conclusion 23

4 Analysis and Design 25

4.1 Difficulty in the base game 25

4.2 Difficulty Adjustment 26

4.3 Tracking Player Actions 27

4.4 Architecture 27

4.5 Functional and Non-Functional Requirements 28

5 Implementation 31

5.1 Tracking the player 31

5.2 Difficulty Rules 33
5.2.1 Enemy health rule 36
5.2.2 Enemy and player damage rules 37
5.2.3 Enemy accuracy rule 38
5.2.4 Health kit and battery rules 39
5.2.5 Health charger and suit charger rules 40
5.2.6 Monster Maker Rule 41
5.3 Determining when to take action 42
5.4 Modifying rule weights 44
5.5 Changing item drops 45
5.6 Analytics 46
6 Testing and Evaluation 49
6.1 Hypothesis and Criteria 49
6.2 Assessment of the player's experience 50
6.2.1 System Usability Scale 51
6.2.2 Game Experience Questionnaire 52
6.2.3 Game Analytics 53
6.3 Assessment of the project's development 54
6.4 Results 59
6.4.1 Profile of the testers 59
6.4.2 Responses to the GEQ 61
6.4.3 Responses to the SUS 62
6.4.4 Validating the Hypothesis 63
6.4.5 Quantitative Evaluation Framework Results 64
6.5 Inferences from the game analytics data 65
7 Conclusions 69
7.1 Proposed vs achieved goals 69
7.2 The solution 70
7.3 Future work and limitation 71
8 References **73**
Annex A. Filled Quantitative Evaluation Framework 77
Annex B. Player Profile Responses 79
Annex C. Responses to the DDA session 80
Annex D. Responses to the unaltered session 83
Annex E. Sample Survey 86

List of Figures

Figure 1 - Game Flow State (Hunicke & Chapman, 2004) 5
Figure 2 - Alternative enemy placements in Resident Evil 4 (Game Makers Toolkit, 2015) 6
Figure 3 - AHP Hierarchy 20
Figure 4 - Component Diagram 28
Figure 5 - Deployment Diagram 28
Figure 6 - Structure that holds player statistics 32
Figure 7 - Death tracking additions to the CBasePlayer::Killed function 33
Figure 8 - Dynamic difficulty rules diagram 33
Figure 9 - ApplyRule function of the EnemyHealthRule class 36
Figure 10 - Header of the CalculateMonsterHealth function 36
Figure 11 - Health recalculation in the MonsterInit function 37
Figure 12 - ApplyRule function of the EnemyDamageRule class 38
Figure 13 - ApplyRule function of the PlayerDamageRule class 38
Figure 14 - ApplyRule function of the EnemyAccuracyRule class 38
Figure 15 – Function that determines if a shot should hit based on the current accuracy modifier. 39
Figure 16 - ShootAtEnemy function of the CBaseMonster class 39
Figure 17 - ApplyRule function of the HealthKitRule and BatteryRule 40
Figure 18 - ApplyRule function of the HealthChargerRule and SuitChargerRule classes 40
Figure 19 - Function that determines the charge of health and armour stations. 41
Figure 20 - ApplyRule function of the MonsterMakerRule class 41
Figure 21 - Header of the CalculateMonsterMakerLimit function 42
Figure 22 - Spawn Function of the CMonsterMaker class 42
Figure 23 - Encounter with increased and decreased enemy spawn rates. 42
Figure 24 - Function that determines if a difficulty changing action is necessary 43
Figure 25 - Screenshot of an item drop with low player health. 46
Figure 26 - Screenshot of an item drop with high player health and armour. 46
Figure 27 - Deaths in level c1a3 recorded from 01/06/2021 to 14/06/2021 47
Figure 28 - In-game console with the player's current statistics 48
Figure 29 - Usability scores graded on a curve (Sauro, 2011) 52
Figure 30 - Heatmap of player deaths in the Early Access version of Divinity Original Sin 2 (Vincke, et al., 2020) 54
Figure 31 - Ages of the participants 59
Figure 32 - Participants previous exposure to half-life 60
Figure 33 - Mean hours spent on video games by the participants 61
Figure 34 - Participants difficulty preferences 61
Figure 35 - Number of weapon selections 65
Figure 36 - Player death causes 66
Figure 37 - Player death heatmap for section c1a3a 67
Figure 38 - Most lethal encounters in section c1a3a 67

List of Tables

Table 1 - DDA Techniques 4
Table 2 - Comparison of games and their DDDA techniques 8
Table 3 - Business Model Canvas 18
Table 4 - Value of AHP criteria (Vargas, 1990) 19
Table 5 - AHP Criteria comparison matrix 21
Table 6 - Normalised AHP criteria comparison matrix 21
Table 7 - Relative priority between AHP criteria 21
Table 8 – Random Consistency Index Table (Saaty, 1980) 22
Table 9 - AHP Qualitative score to numerical value conversion 23
Table 10 - AHP Qualitative score of each alternative solution 23
Table 11 - AHP final score of each alternative solution 23
Table 12 - Functional Requirements 28
Table 13 - Non-functional requirements 29
Table 14 - Rules and their modifiers 34
Table 15 - Quantitative Evaluation Framework 56
Table 16 - Average responses to the GEQ 62
Table 17 - SUS Scores for the original version and the DDA implementation 63
Table 18 - Test Statistics and Critical value for the 12 hypotheses 63

Acronyms and Symbols

List of acronyms

AHP	Analytic Hierarchy Process
AI	Artificial Intelligence
AID	The AI Director
DDA	Dynamic Difficulty Adjustment
FEI	Front End of Innovation
FPS	First-Person Shooter
GEQ	Game Experience Questionnaire
HUD	Heads-up Display
KPI	Key Performance Indicator
NCD	New Concept Development Model
NPC	Non-Player Character
NPPD	New Product and Process Development
QEF	Quantitative Evaluation Framework
SDK	Software Developer Kit
RPG	Role-playing Game
SUS	System Usability Scale

1 Introduction

Different kinds of players are motivated by different factors when playing video games. Among these factors is one known as the “Hard Fun” factor. This factor is represented by the challenges the players face and the enjoyment that ensues from their successful resolution. To maximise the “Hard Fun” factor, it is thus important to create experiences that match the player's skills as, when players face a challenge that matches their skill, they enter a state of mind that stimulates their creativity and enjoyment.

This project aims to answer the question: How can the difficulty of a game be adjusted dynamically to improve a player’s gaming experience? Most video games still apply static difficulty levels, which can be altered manually by the player at any time. However, this manual change still instils in the player the notion that they are not performing as well as they should. By not informing the player of changes in the difficulty we lower the risk of the player becoming disappointed. However, if the Dynamic Difficulty Adjustment (DDA) algorithm takes too many actions which result in a lower difficulty, then the players might notice and feel like the game is handholding them too much, which in many cases leads to a lesser enjoyment of the experience. If the algorithm takes too few actions or the wrong actions for a given situation, then players might not benefit from the algorithm, or in the worst case, be hindered by it. However, should the algorithm perform well, then it is expected that the players feel more immersed in the game, which in turn leads to a better experience in the end.

In our proposed solution, we alter an already existing videogame, Half-Life, so that it applies changes to the game’s difficulty while the player is playing, providing an improved experience.

The algorithm will increase the difficulty gradually as the player improves, but it will also lower the challenge when the player is struggling and offer them the tools required to succeed.

1.1 Objectives

The objective of this work is to assess if using DDA in a First-Person Shooter (FPS) game can enhance a player's experience. For this purpose, we will be designing, developing, testing, and validating a modification, that uses DDA, to an already existing FPS game, Half-Life. We will also conduct research on current DDA implementations used in the videogame industry as well as solutions found by other researchers and the factors that lead players to enjoy their gaming experiences.

To guide our development and deliver maximum value to its users we will also conduct a value analysis describing how this project delivers value by creating experiences that adjust themselves to the user's skills and which implementation will best suit the project's needs.

Finally, the project will be assessed in two stages. In the first stage, we will use the Quantitative Evaluation Framework (QEF) to measure the completion of the project. In the second stage, we will be applying a pilot test to our target audience to measure the quality of the system. At the end of the test sessions, the volunteers of the target audience will answer two questionnaires to assess if the implementation has the positive effect for which it was designed.

2 State of the Art

Nicole Lazzaro identifies four main factors that each kind of player uses to justify their enjoyment of video games (Lazzaro, 2004). The first factor, “Hard Fun”, is all about players overcoming challenges in their game, creating emotions of frustration when the player loses and personal triumph when the player gets rewarded. Players want to test their mettle, reach objectives, and rely more on their skill than on random luck. The second factor, “Easy Fun”, revolves about immersing the player in an alien environment, using their curiosity to inspire sensations of wonder and awe. Players who focus on this factor enjoy exploration, adventure, and prefer to progress through the story, even at the cost of having to use a walkthrough or guide. “Altered States”, the third factor, is preferred by players who enjoy feeling different emotions when playing games. These players prefer experiences that create a sensation of excitement or relief during and after play. Finally, “The People Factor”, is referred to by players whose enjoyment stems from interacting with other players. Players who refer to this factor might even play games they dislike so they can spend time with friends.

To maximise the effect of the “Hard Fun” factor, digital games, traditionally, ask the player to select their preferred difficulty settings from a few predetermined selections (e.g., Easy, Normal, Hard). After the difficulty selection was done, the games would apply different methods for increasing or decreasing the difficulty permanently. Depending on the type of game, this could mean, for example, that the player’s enemies would be stronger or that the player would have to make decisions more quickly. This original method has some fundamental flaws, however: the amount of different difficulty levels is limited, the difference between the levels can be too steep for some players and the players are more aware of the

changes. While some games allow their players to re-select the difficulty throughout the game, this comes at cost of the player's game experience. (Sepulveda, et al., 2019).

It became apparent, as gaming technologies improved, that manually selected static levels of difficulty would no longer be sufficient to maintain the player's attention as they failed to offer an adequate challenge to their skills. Players would often feel frustrated if the difficulty became too high, or bored if they did not feel challenged (Zohaib, 2018). The solution would then be to create an environment where the difficulty of the game was just right for the player.

2.1 Dynamic Difficulty Adjustment

DDA describes techniques that allow a game to modify itself in response to player actions, to better fit the current player's skills. One well-known example of DDA is the practice of "rubber banding". In this practice, used primarily in racing games, the vehicles which are placed behind the player receive speed boosts, while the vehicles placed ahead are slowed down (Rietveld, et al., 2014). Other practices, like dynamic scripting, maintain several sets of rules which are used to create new scripts that control the behaviour of the player's opponent, each time a new opponent is generated. The rules are selected based on a weight value that is adjusted based on the success or failure of the rules in the script. After an encounter with the player's opponents, rules that lead to success on the player's part are rewarded with an increase in weight, while rules that potentially lead the player to fail to see their weight reduced (Spronck, et al., 2004). Table 1 offers an overview of techniques that can be used for difficulty adjustment.

Table 1 - DDA Techniques

Technique	Description
Rubber-Banding	Accelerating vehicles behind the player and slowing down vehicles ahead of the player
Dynamic Scripting	Generating behaviour patterns based on rules that lead to positive outcomes
Modifying Enemy layout	Placing or removing enemies from the scene
Modifying Enemy types	Replacing enemy non-player characters (NPC) with others the player has more, or less, success against
Modifying Item layout	Replacing or removing items depending on the player's needs
Reaction Time Modification	Balancing the time between when the enemy artificial intelligence (AI) notices the player and their actions
Modifying Enemy Stats	Changing enemy NPC hit-points, speed, accuracy, and damage
Modifying Player Stats	Changing player speed, hit-points, accuracy, and damage

Competitive Matchmaking	Grouping players in a multiplayer game based on their previous performances
Machine Learning	Using machine learning agents to control NPC

The goal of these practices is to create experiences that match the skill of the players. When players are presented with experiences that are just challenging enough, meaning that they are adjusted to the player's ability, they enter a state of mind of increased creativity and enjoyment, colloquially known as "the zone", but officially named Flow. Flow, as described by Csikszentmihalyi, is a state in which a person voluntarily invests their attention to achieve an optimal experience, becoming absorbed in the experience and ignoring external stimuli. Optimal states result from experiences in which one is focused on goals that can realistically be achieved and one's skills are sufficient to partake in the available opportunities. The appeal in achieving an optimal experience results from its intrinsic rewarding nature. The activities which result in optimal experiences are thus undertaken for their own sake and no other (Csikszentmihalyi, 1990).

As a person partakes in each experience, their skill level increases, which in turn means that the necessary challenge must increase at a similar rate, avoiding the creating situation of distress or boredom, to guarantee that the flow state is maintained.

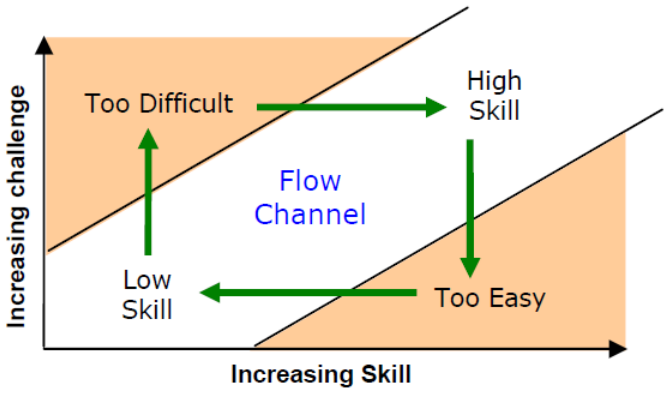

Figure 1 - Game Flow State (Hunicke & Chapman, 2004)

2.2 Examples of DDA in games

Some game development studios have already adopted DDA mechanisms in their video games. In this section, we will demonstrate the solutions they used on some successful commercial videogames.

2.2.1 Resident Evil 4

The sixth major instalment in the Resident Evil series, Resident Evil 4 was developed by Capcom in 2005 as a survival horror, third-person shooter. The player selects a difficulty at the start of the game, as usual, but that difficulty can then increase or decrease within certain thresholds, according to the player's actions. The game alters the reaction time of the enemies, their health and strength, as well as the items the player receives. If the player is performing well, the enemies will do more damage, have increased health and decreased reaction time. The player will receive fewer items, and, in some cases, more enemies will be present. Should the player be performing poorly, then the enemies will do less damage, have fewer hit points, and give the player more time to react to their presence. The player will also be awarded more items and more enemies will be present. (Capcom, 2005)

Figure 2 represents one of those moments where the game can alter the number of enemies the player encounters. On the left side of the image, one can count nine enemies, seven on the same floor as the player and two standing on balconies. This represents the enemy arrangement in case the player is performing well. Should the player be performing more poorly, or fail repeatedly to overcome this challenge, then the game will automatically decrease the number of enemies and present the player with the situation on the right, where the two enemies on the balconies are removed and only the seven on the same floor as the player are present.

Figure 2 - Alternative enemy placements in Resident Evil 4 (Game Makers Toolkit, 2015)

2.2.2 The Left 4 Dead Series

Left 4 Dead is a series of first-person shooter survival horror games, developed by Valve. The first instalment released in 2008, while the second, and last, released exactly one year later, in 2009. In this game, four players are tasked with escaping from a horde of zombies. The

players must cooperate and coordinate their movements to achieve their goals. To guarantee that the player team receives a challenge that matches their skill, the games implement one DDA system known as “The AI Director” (AID). The AID is responsible for changing the pacing and difficulty of the game. To achieve this, the AID can choose between different positions to place the enemies, as well as change the number they appear in. This decision is done based on the situation each player is in, their current health and equipment status, skill, and location within the map. Should the players be moving through the map too slow or too quickly, the AID will spawn a zombie horde to either force the players to change location or to further hinder their advances. The AID also forces the players to cooperate, as players who wander off from the rest of the group will be preferred targets of the AID’s enemy placement. While some items are fixed within each map, the AID can decide to spawn more items at some pre-determined locations or replace items with others that would be more advantageous to the players. Finally, the AID can also decide to give the players a few moments to rest and recover from previous engagements (Left 4 Dead Wiki, 2021) (Booth, 2009).

2.2.3 FIFA

FIFA is a football (soccer) game developed by Electronic Arts. The series has a yearly release schedule, meaning that each year a new version comes to the market. The players can select a team and are tasked with controlling the football players on the field, score goals and thus win the match. The games offer several game modes; however, the core concept of the game does not differ much. The developers at Electronic Arts found that to increase player engagement, it was beneficial to implement a system that was able to predict when the player faced repeated difficulties and was at risk of abandoning a game. Taking into consideration the player’s track record in previous matches/attempts, giving more importance to more recent tries they adjust the difficulty of the game to improve the likelihood of the player remaining in the game (Xue, et al., 2017). The system that was created and is used in many of Electronic Arts recent games can adjust the speed of in-game characters as well as their accuracy and even the responsiveness of controls, noting that in some cases the character’s characteristics would be lessened rather than improved, to create an optimal experience (Aghdaie, et al., 2017). FIFA is also a multiplayer game and offers players the option to play against other human players. Players are then selected based on their performance history in previous games, so that more experienced players play against other more experienced players instead of those who are just learning how to play the game.

2.2.4 Mario Kart

Mario Kart is a racing video game developed and published by Nintendo where players use go-karts to race against various characters from the Super Mario universe. Players are tasked with steering the vehicle and avoiding obstacles and be the first to reach the finish line. Along the track, players can pick up several different items that give them abilities that can be helpful during the race. To balance the difficulty of the race, the game uses rubber-banding, a practice where vehicles that are in lower positions in the race receive speed boosts so that it becomes easier for them to catch up to the leader, and also put some pressure on the players that are leading the race. Alongside this practice, Mario Kart also dispenses different items in accordance with the player's position, where the last players are given more powerful items than the ones who are leading. Both of these practices have come under criticism as they are easily exploitable (Myers, 2017) (Karvande, 2020).

2.2.5 Overview of the games and their DDA techniques

Having analysed some games that natively use DDA we can now proceed to compare them and see which techniques are most prominent. Table 2 offers an overview of all the games we presented and the prevalence of each of the previously listed techniques. Although the list is not extensive, it still allows us to compare the games and see which techniques are most appropriate for each type of game.

Table 2 - Comparison of games and their DDDA techniques

Game	Technique				
	Rubber-Banding	Dynamic Scripting	Enemy Layout	Enemy Types	Item Layout
Resident Evil 4			X		X
Left 4 Dead			X	X	X
FIFA 2017					
Mario Kart	X				X
Game	Technique				
	Reaction Time	Enemy Stats	Player Stats	Matchmaking	Machine Learning
Resident Evil 4	X	X	X		
Left 4 Dead					
FIFA 2017	X	X	X	X	
Mario Kart				X	

2.3 Technology related to Dynamic Difficulty

DDA implementations have been the target of several studies over the last years. These implementations can be divided into two categories. First, we have the implementations that use algorithms to monitor the player's performance and adjust the game using pre-determined actions, as we exemplify with The Hamlet System. The other category of DDA implementations we identified make use of machine learning to enhance non-player enemies as a way of adjusting their capabilities to the player's skill, as is exemplified by DRE-Bot. This chapter is dedicated to illustrating the benefits and shortcomings of both types of implementations and describe some of the tools used in the development of our solution as well.

2.3.1 Hamlet

The Hamlet (Hunicke & Chapman, 2004) system, developed by Robin Hunicke and Vernell Chapman, for *Half-Life* is a set of libraries embedded in the game's engine that monitors several metrics to estimate how well the player will adjust to a future situation in the game. If the system determines that the player is on a path that will lead them to fail repeatedly, it will adjust the game accordingly.

The assessment of the player's situation is made by observing the state of their inventory, tracing how the player has used their items so far, watching for possible shortfalls, and by the damage, the player has been taking in previous encounters. If an unpleasant situation is detected, Hamlet may either take proactive or reactive actions. Reactive actions will adjust elements that are currently present, like enemies that are aware of the player's presence, by modifying the accuracy, or the damage of the enemy's attacks or their health. Proactive actions modify elements that are not yet visible to the player, which might entail changing the type and number of enemies that appear as well as their accuracy and damage (before they appear).

Each of the modifying actions is given a cost, which helps determine if taking said action is necessary for the current state, avoiding turning the game experience too trivial and boring. Should the system make too many changes, which are perceived by the player, then the player might lose their immersion in the experience.

The authors concluded that dynamically adjusting the difficulty comes with some trade-offs regarding the type, number, and frequency of adjustments, as well as in performance. Furthermore, they conclude that making a perfect simulation of the player's actions would "*be less useful due to the time constraints of real-time gameplay and the limitations of commercial gaming hardware*." Instead, they opted to create an abstract simulation as that would give them an array of possible situations to steer away from.

2.3.2 DRE-Bot

Developed by Frank G. Gavin and Michael G. Madden, DRE-Bot is an architecture for controlling non-player characters (NPC) in the game *Unreal Tournament 2004*, a first-person shooter. DRE-Bot uses three reinforcement learners based on the tabular Sarsa (λ) algorithm.

The proposed architecture consists of three main modes, namely *Danger*, *Replenish* and *Explore* mode. The architecture activates *Danger* mode whenever an opponent is detected, or the bot is actively being damaged. *Replenish* mode occurs when the bot's ammunition or health threshold is below a low (40%) or critical (20%) level. Finally, *Explore* mode is activated when none of the other two modes is active. Each of the modes corresponds to a different reinforcement learning agent. The idea behind the separation into three different agents is that having different learning parameters, actions, states, and rewards would lead to an advantageous learning process.

In their experimentation, the authors tested the capabilities of the DRE-bot in a deathmatch server against two regular bots, until the DRE-Bot had been defeated 200 times. Their results showed that while the bot did perform well and learning did occur, the performance of said learning appeared to be insensitive to parameter changes in the algorithm, which was explained by the implicit randomness intrinsic to the game and the small and simple map which was chosen for the experiment. On average, the DRE-Bot carried out 100 more kills than the regular bots.

2.3.3 GoldSource

The GoldSource engine, also known as GoldSrc, is a 3-D videogame engine, running on C++ based on ID Software's Quake Engine. The engine was created in 1996 by the Valve Corporation and was used to create some of the better-known titles of the time, such as Half-

Life and Counter-Strike. Despite its age, it still widely used for the creation of modifications for the titles released on it, most notably Half-Life (Valve Developer Community, 2020). Many of the titles released on the engine were initially created as modifications to Half-Life: Counter-Strike, Team Fortress, and Day of Defeat.

2.3.4 Half-Life

Half-Life is an FPS game developed by the Valve Corporation in 1998 (Valve Corporation, 2020). This game features both a single-player campaign as well as a multiplayer mode. In the multiplayer mode, players can engage in deathmatches and team deathmatches, where the goal is to eliminate the enemy players and players respawn into the game shortly after they are downed. In the single-player campaign, players take on the role of a scientist named Gordon Freeman, who must escape a research facility invaded by an alien life-form. As the player advances in the campaign, they unlock new weapons and are faced with different, stronger, types of enemies. The game was re-released on Valve's Source engine in 2004, but this version albeit making use of a better engine is considered to be inferior to the original in many ways as it introduces many bugs that were not present in the original version (Valve Developer Community, 2021).

2.3.5 GameAnalytics

GameAnalytics is a tool developed by GameAnalytics that allows developers to visualise their key performance indicators (KPI). GameAnalytics offers a Software Development Kit (SDK) for most modern game engines (Unity, Unreal, Godot, etc.) but also features SDKs for any engine that uses C++ or C#. Users can specify custom KPIs and use them to further their understanding of their game's audience behaviours (GameAnalytics, 2021). This project will make use of the GameAnalytics tool to visualize the conditions under which the players struggle to keep up with the difficulty of the game and adjustments should be made, as well as gather other useful data on player's behaviour during play sessions.

2.4 Conclusion

Over time, game development studios and researchers have experimented with different ways to adjust the difficulty of games. Either through artificial intelligence or pre-determined

algorithms, games have successfully been shown to cater to the skills and needs of the players. Different game genres are also shown to be more suited for different approaches to DDA. While no commercial videogame has yet to officially acknowledge using dynamic scripting, it is more suitable for RPGs, while first- and third-person shooter games are more suited to alterations of the player's environment, through enemy and item placements and modification of the enemy's strength and rubber-banding being only really suitable for racing games.

3 Value Analysis

This chapter will analyse the value this project will bring as well as other factors that lead to this project's creation.

3.1 New Concept Development Model

The innovation process, as described by Koen, is comprised of three sections. The "Front End of Innovation" (FEI), the "New Product and Process Development" (NPPD) and the commercialization phase. The New Concept Development Model (NCD) provides a common language and identifies the primary stages of the FEI.

The NCD is comprised of three components. The "inner area", which defines the five components of the FEI, the "Engine" which acts as the driving force between the FEI elements and is influenced by the organization's culture and leadership, and finally, the "Influencing Factors" which describes the environment that influences the decision-making process in the first two components (Koen, et al., 2001).

The five components of the FEI, identified by Koen are the following:

- **Opportunity Identification** – One of the objectives of videogames is to keep the player's interest for as long a time as possible, however, static difficulty levels in games are a source of player frustration, as they face challenges unfit for their skill level. GILT thus decided that this project, by adjusting the difficulty of a game dynamically, as a player plays the game, would be advantageous.

- **Opportunity Analysis –** After careful consideration, it was decided that the best course of action would be to implement DDA capabilities in an existing game that already enjoys a large audience.

- **Idea Genesis** – In this component alternative ways to implement DDA were discussed, as well as different games on which to work. The first idea involved using machine-learning algorithms to control bots in a multiplayer deathmatch game, the second idea focused on altering aspects of the game's single-player campaign as the player progressed, the last idea was to use a machine learning agent to control the player character in a multiplayer match against human players and assess their reaction.

- **Idea Selection –** In this step we selected the alternative that would best fit the project's requirements, taking into consideration aspects such as performance, market saturation and adaptability. To assess which solution would be best, based on each criterion, the Analytic Hierarchy Process (AHP) was used, and we determined that the best course of action would be the adjustment of the difficulty during the single-player campaign of the game.

- **Concept & Technology Development** – In the final step, we outline the business plan so that the project has the best chances to be successful.

3.2 Value Proposition

A company's value proposition consists of the products or services it delivers to full fill the customer needs of each type of client and describes how the company differs from its competitors, and the reason its customers prefer it to other companies and their products. (Kambil, et al., 1996). Several factors contribute to the products perceived value. These factors can be either quantitative, like the price and performance, or qualitative, like the design or customer experience (Osterwalder & Pigneur, 2010).

This project aims to deliver value to the customer by providing a game experience that tailors itself to each player and relates most strongly with the "Customization" factor identified by Osterwalder, where value is created by offering products and services which are tailored to each customer's needs.

The goal is thus to create a DDA mechanism that analyses the players' skill level and balances the game accordingly. By creating the perfect challenge for the player, it becomes easier for them to maintain their flow state and be immersed in the experience, increasing their enjoyment of the game and thus its value.

3.3 Porter's Value Chain

Any function or task a company engages in can be seen as a process in so far as it transforms certain inputs into outputs. A business process is thus defined as a set of related activities that, from one or more inputs, generate an output with value for the customer (Pinto, et al., 2018). According to Porter, these processes are linked and can be classified in either primary or support activities. How these activities are linked determines costs and affects the company's profits (Porter, 1985). The five primary activities are:

- **Inbound Logistics** – involve the relationships with suppliers and the activities related to receiving, storing, and disseminating inputs. Here we will be depending on Valve's GoldSrc Engine and the available documentation which are directly procured from the Valve Corporation, and the git version control system (VCS).

- **Operations** - are all the activities that transform inputs into outputs (products and services). In the case of this project the activities that fall into the operations category revolve around development.

- **Outbound Logistics** - include all the activities required to collect, store, and distribute the outputs. The project will be available for download on the project's internet page.

- **Marketing and Sales** – activities that make buyers aware of the products being offered and increase the likelihood that the customer purchases them. This activity is fulfilled by publicising the project on video-game forums and direct communication with possible users.

- **Service** - includes all the activities that maintain the product for the buyer after it is sold and delivered. In the case of this project, this activity would be done through direct communication with the users via email or through the project's page.

The four secondary activities are:

- **Procurement** - the acquisition of inputs, or resources, for the business. This process is done internally and consists of the direct acquisition of the inputs from the Valve Corporation. as well as the establishment of the VCS.

- **Human Resource Management** - all activities involved in recruiting, hiring, training, developing, compensating, or dismissing personnel. Given that the project consists of only one person, no hiring or firing will occur. However, training and development will occur in the form of researching algorithms and reading documentation.

- **Technological Development** - equipment, hardware, software, procedures, and technical knowledge created in the firm's transformation of inputs into outputs. The technical development of this project will be an approach to DDA which can be applied to other games in the same or similar genres.

- **Infrastructure** - serves the company's needs and ties its various parts together, it consists of various functions or departments such as accounting, legal, quality assurance and general management.

3.4 Business Model Canvas

The Canvas is a simple chart, representative of a business model, which describes a product's value proposition, finances, customers, and resources, allowing us to focus our attention on the critical aspects of a business model. The canvas can be divided into two sections. The first section, the backend which describes activities, resources, partners, and costs. The second section, the frontend describes the customer segments, distribution channels, customer relationships and revenue channels (Osterwalder & Pigneur, 2010). In its entirety, the canvas is thus composed of 9 segments:

- **Value Propositions:** describes what is offered to each customer segment and how it satisfies the customers need. This section explains why our product is valuable to the customer.

- **Customer Relationships**: explains the business' relationship with its customer base, the expectations the customers have from this relationship and how it relates to the product.

- **Customer Segments**: lists the different types of customers the product is catered to and identifies the main customer segment of the product.

- **Channels**: identifies the ways to reach out to the customers, their preferred communication channels, how the channels are connected and their effectiveness.

- **Revenue Streams**: describes the customer's preferred payment methods, the services/aspects of the product which the customers find most valuable and how it contributes to the total revenue of the business.

- **Key Partners**: Identifies the most important partners to the business and its suppliers, as well as the resources the suppliers are supplying and the activities the partners perform that benefit the business.

- **Key Activities**: Lists all the activities which result from the elements described in the frontend.

- **Key Resources**: Lists the resources needed to fulfil all the elements described in the frontend.

- **Cost Structure**: Identifies the most important and most expensive costs in the business model.

The canvas which describes this project is illustrated in Table 3

Table 3 - Business Model Canvas

<table>
<tr><th>Key Partners</th><th>Key Activities</th><th>Value Propositions</th><th>Customer Relationships</th><th>Customer Segments</th></tr>
<tr><td rowspan="4">• GILT
• Valve Corporation
• GameAnalytics</td><td rowspan="2">• Design the DDA approach
• Implement the DDA approach
• Test and validate the DDA approach</td><td rowspan="4">• Modification of Half-Life to dynamically adjust the difficulty to the player’s skill</td><td>•Personal assistance via e-mail.
•Face-to-face conversation</td><td rowspan="4">• Half-Life Players
• Players looking for a game with DDA
• FPS Players</td></tr>
<tr><td>Channels</td></tr>
<tr><td>Key Resources</td><td>• Surveys
• Direct Communication with the customers
• GitHub project page</td></tr>
<tr><td>• GoldSrc SDK
• Half-Life</td></tr>
<tr><th colspan="3">Cost Structure</th><th colspan="2">Revenue Streams</th></tr>
<tr><td colspan="3">• Development Costs
• Promotion Costs.</td><td colspan="2">• Free Software</td></tr>
</table>

3.5 Analytic Hierarchy Process

The AHP is a decision-making method developed by Thomas Saaty in 1980. It helps in the selection of different alternative solutions to a problem by quantifying relative priorities on a ratio scale based on the decision-maker's judgement (Saaty, 1980).

The AHP evaluates each possibility using criteria selected by the decision-maker in the following steps:

- Establishment of the criteria hierarchy: the highest level of the hierarchy is occupied by the end goal of the decision-making process. The next levels are filled by sub-criteria derived from the end goal; this helps the decision-maker set priorities between the sub-criteria.

- Allocation of weights to the criteria chosen in the previous step, creating more sub-criteria when necessary. The criteria of each level of the hierarchy are compared pairwise. This comparison allows us to focus on only two criteria at a time so that we can better establish which criterion is more important. During the comparison, each criterion is given a value between 1 and 9, a description of the meaning of each value can be consulted in Table 4.

- Assignment of numerical values to each dimension on the evaluation scale. These dimensions are generally classified as poor, fair, good, very good, and excellent and are given a numerical value between 0 and 1. Which dimension corresponds to each exact value is up to the discretion of the decision-maker.

- Calculation of the final grade for each alternative. The total grade of the project is calculated from the sum of all products of weights and numerical values of the project (Palcic & Lalic, 2009).

Table 4 - Value of AHP criteria (Vargas, 1990)

Value	Description
1	Criteria have equal importance
3	Somewhat greater importance of one criterion over another
5	Strong superiority of one criterion over another
7	Very strong superiority of one criterion over another
9	Absolute superiority of one criterion over another

Note: usage of values in between (2,4,6,8) is permitted

3.5.1 Establishment of the criteria hierarchy

In this step, we define the end goal, the criteria, and the possible solutions and organize them in the hierarchy so that it becomes easier to choose the best solution for the project. Three criteria were chosen for this assessment:

- **Performance** – We consider the performance of the solution to be the degree to which it can cater the difficulty of the game to each player's abilities.

- **Market Saturation** – Novel Ideas are more interesting as they bring new insights. Market saturation is thus defined by the number of different projects that use a similar approach.

- **Adaptability** – Adaptability is here defined by how well the solution would perform in different game genres from the one chosen. While some ideas can easily be translated into other games, others would be more difficult to adapt. Thus, it is of greater interest to study solutions that can be applied to a greater number of games.

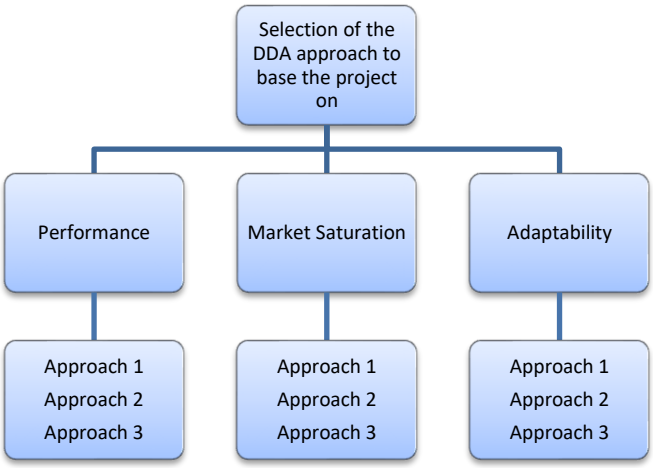

Figure 3 - AHP Hierarchy

The 3 approaches shown in Figure 3 result from the idea genesis section of the NCD and can be surmised as follows:

- **Approach 1** – Creation of a machine learning agent to control several bots in a multiplayer deathmatch game, creating a bot that learned from the player's playstyle.

- **Approach 2** – Modifying aspects of the single-player campaign as the player progressed through the game, increasing difficulty, and limiting resources if the player performed well and lowering the difficulty and offering more items if the player was struggling.

- **Approach 3** – Creation of a machine learning agent to control the player character in a multiplayer deathmatch game with the intent of gathering data on human players' reactions to the confrontation.

3.5.2 Allocation of weights to the criteria

In this step, we determine the relative importance of each criterion in comparison with the remaining criteria using a comparison matrix. The resulting matrix can be consulted in Table 5.

Table 5 - AHP Criteria comparison matrix

Criterion	Performance	Market Saturation	Adaptability
Performance	1	5	5
Market Saturation	1/5	1	2
Adaptability	1/5	1/2	1

Having obtained these values, it becomes necessary to normalize them to identify the relative priority between the criteria.

Table 6 - Normalised AHP criteria comparison matrix

Criterion	Performance	Market Saturation	Adaptability
Performance	5/7	7/9	5/8
Market Saturation	1/7	1/6	1/4
Adaptability	1/7	1/13	1/8

By finally calculating the average of each row of Table 6 we can find the relative priority of each criterion.

Table 7 - Relative priority between AHP criteria

Criterion	Relative Priority
Performance	0.70
Market Saturation	0.18
Adaptability	0.12

We can thus conclude that the most important criterion in the decision-making process is the performance, followed by market saturation and finally the adaptability.

Before moving on to the next step, it is still necessary to verify the consistency of the relative priorities, if the consistency scores above 0.1 then the tables must be re-done.

The consistency of the relative priorities is given by the consistency ratio (RC) using the following formula:

$$RC = \frac{IC}{IR}$$

Where IC is the consistency index and IR is the random consistency index. The IC can be obtained by:

$$IC = \frac{\lambda_{max} - n}{n - 1}$$

Where n corresponds to the number of criteria and λ_{max} is the eigenvalue of the matrix.

$$\begin{bmatrix} 1 & 5 & 5 \\ 1/5 & 1 & 2 \\ 1/5 & 1/2 & 1 \end{bmatrix} \times \begin{bmatrix} 5/7 \\ 1/5 \\ 1/9 \end{bmatrix} = \begin{bmatrix} 2.19 \\ 0.55 \\ 0.35 \end{bmatrix}$$

λ_{max} is then given by the average of the division of each row of the final matrix and the priority vector:

$$\lambda_{max} = \frac{\frac{2.19}{\frac{5}{7}} + \frac{0.55}{\frac{1}{5}} + \frac{0.35}{\frac{1}{9}}}{3} = 3.05$$

Now that λ_{max} has been found, we can proceed to determine the IC using the formula described above.

$$IC = \frac{3.05 - 3}{3 - 1} = 0.025$$

Using Thomas Saaty's index table to determine our IR we can then calculate our RC.

Table 8 - Random Consistency Index Table (Saaty, 1980)

N	1	2	3	4	5	6	7	8	9	10
IR	0.00	0.00	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.49

Considering that our decision is based on 3 criteria, we establish that our $\mathrm{IR} = 0.58$ and thus our RC equals:

$$RC = \frac{0.025}{0.58} \approx 0.04$$

Having determined that our RC is below the threshold of 0.1 we can safely proceed with the decision-making process.

3.5.3 Assignment of numerical values and calculation of the final grade

In this step, we will be assessing how each solution scores on each criterion, for that we will be using qualitative dimensions as these make it easier for the decision-maker to determine each solution fulfils each criterion.

Table 9 - AHP Qualitative score to numerical value conversion

Qualitative Identifier	Poor	Fair	Good	Very good	Excellent
Numerical Value	0.00	0.15	0.35	0.70	1.00

Table 10 - AHP Qualitative score of each alternative solution

	Total	Criterion	Performance	Market Saturation	Adaptability
		Weight	0.7	0.18	0.12
Approach 1		Score	Good	Very Good	Good
Approach 2			Excellent	Fair	Very Good
Approach 3			Fair	Excellent	Good

Having determined the qualitative score for each criterion we can then replace the qualitative dimension with its numerical value and determine the final score for each alternative.

Table 11 - AHP final score of each alternative solution

	Total	Criterion	Performance	Market Saturation	Adaptability
		Weight	0.7	0.18	0.12
Approach 1	0.413	Score	0.35	0.70	0.35
Approach 2	0.811		1.00	0.15	0.70
Approach 3	0.257		0.15	1.00	0.35

The total score for each alternative is calculated by:

$$Total = \sum Criterion\ Weight \times Criterion\ Score$$

3.5.4 Conclusion

Having determined the total grade of each alternative we can now select Approach 2 as being the best fit for the goals of this project. The weight of the performance criterion being the highest made it crucial in the final score. This can be verified by comparing Approach 2 and 3, while both had one “Excellent” and one “Fair” score, the dominance of the performance criterion resulted in a major difference in the end.

4 Analysis and Design

Having acquired some information on techniques used to dynamically adjust the difficulty, we can now make informed decisions on how we will solve our problem. This chapter will outline different approaches to the two main factors we take into consideration when adjusting the difficulty of the game.

4.1 Difficulty in the base game

Half-Life is an FPS, and as such the player takes control of a single character. This character is equipped with weapons and armour which help the player stay alive. The player is hindered in the progression of the game by multiple encounters with enemy NPC who will attack the player on sight.

The regular version of Half-Life offers the player's the ability to choose one of three difficulty levels. According to the selected difficulty, the enemy NPC the player can encounter will have different amounts of hit-points and damage they can do to the player on a successful hit. Items that are dispensed to the player will also have different effects depending on the difficulty, for instance, a "Health Kit" will heal the player for 15 hit-points on Easy difficulty, but only 10 on Hard difficulty. Finally, some NPC also have different abilities, where one of the NPC gains the ability to become invisible for a certain amount of time on higher difficulties.

Considering that the game already possesses the ability to change these aspects, albeit only at the start of the game, we can use the same functions as a basis for some of our difficulty adjusting actions.

4.2 Difficulty Adjustment

To match the challenges to the player we considered two alternatives. The first alternative would be the analysis of the player's behaviour, for instance, if the player has low accuracy, then we can increase the damage they do with each hit or lower the enemy's health, additionally, we can also present them with more ammunition with the weapon that they prefer. Should the player be more accurate, then we can lower the amount of ammunition they receive, increase the enemy's health but also increase the damage the player does on critical hits. Players who see themselves constantly at low health would receive stronger health items and the enemies would do less damage. This alternative has the benefit of providing the players with exactly the modifications that they need; however, it also implies stricter tracking of the player's actions, successes, and failures.

The second alternative to our DDA implementation would be to create a system similar to dynamic scripting where we would allocate a weight and a cost to each difficulty altering action. After defining a base weight and cost for each action we could then select randomly from the action set and analyse the outcome. If the selected actions resulted in a positive outcome, then we would increase their weight, increasing the likelihood that those actions get selected the next time the difficulty gets adjusted. The primary benefit of this alternative is that it is more dynamic than the first, but it might also introduce more erratic behaviour due to the implicit randomness in the action selection.

Additionally, independently of the selected alternative, it is also beneficial to track the duration of encounters to determine when a difficulty increasing or lowering action should be taken, as too long or too short encounters are detrimental to the player's experience.

Taking into consideration both the advantages and disadvantages of each approach we decided to create a system based on both alternatives, where difficulty altering actions related to combat (damage output, enemy hit-points, accuracy, etc.) are regulated by the system based on dynamic scripting, but actions related to item placement are controlled by the first approach. This way we guarantee that the player receives the items that they need, removing the randomness from this process, but we also improve the process by which we regulate the player's enemies while reducing the tracking needed for that adjustment.

4.3 Tracking Player Actions

As the player progresses through the game, we need to keep track of how they solve the challenges that they are faced with, so that we can take the appropriate actions in response. This tracking must be done on both a local level, to affect the difficulty of the player currently playing the game, however, it also needs to be done globally to help us identify possible shortcomings of our solution. Integrating this tracking with the online services provided by GameAnalytics we can perform the tracking in two ways.

One solution would be to submit to the GameAnalytics platform each action as soon as it occurred, meaning that each time the player used a weapon, for instance, we would upload that information. This alternative has the drawback of performing multiple and repeated requests to the GameAnalytics API, however, it is also the alternative that is most accurate in terms of tracking the player's actions, as we would be certain that every action got tracked in each session.

Our second alternative would be to track the statistics locally and only submit them when the player finished the level or failed and retried and when the player closed the game. By using this alternative, we would reduce the number of requests done throughout the session, but we would also incur a performance impact at the time of uploading the data.

Considering the performance impact of the second alternative, we determined that the first option would be the most beneficial.

4.4 Architecture

To solve the problem at hand we must next conceptualize the relationship between the different systems in our solution, to highlight how they will be interacting. Figure 4 highlights the relationship between the base game, our two key components, and the GameAnalytics API. The DDA component is responsible for handling all the difficulty modifying actions, offering all the functions that allow the game to modify its parameters per the current changes. Meanwhile, the Analytics component is responsible for formatting and communicating the gathered information between the DDA component and the external GameAnalytics API.

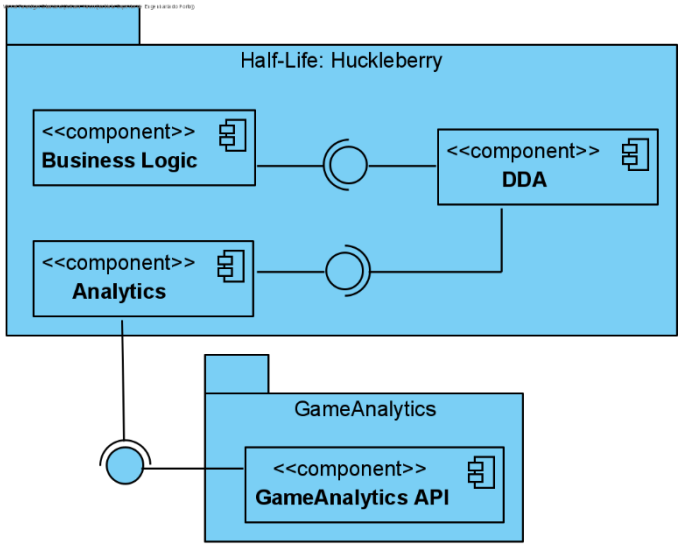

Figure 4 - Component Diagram

Figure 5 highlights the relationship between our game modification and the service offered by GameAnalytics.

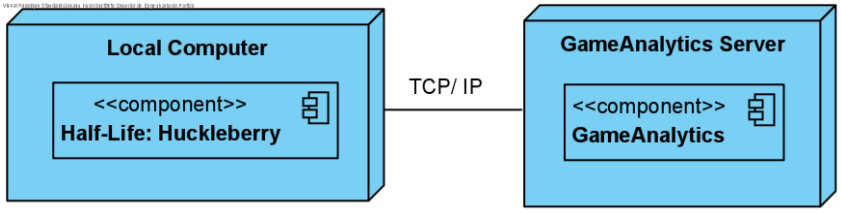

Figure 5 - Deployment Diagram

Considering the nature of the GameAnalytics service it is clear that the data can only be sent if an active internet connection is established, however, in case no such connection can be established during the session, the events are also stored locally and submitted once such a connection is available.

4.5 Functional and Non-Functional Requirements

As our system aims to modify the game's difficulty, we must establish what actions the system should be able to perform to increase or decrease the difficulty and persist said changes over time. For this purpose, we list a set of functional requirements the system must implement, which can be seen in Table 12.

Table 12 - Functional Requirements

Functional Requirement No.	Function Requirement Description
FR1	The system shall increase or decrease NPC health

FR2	The system shall increase or decrease NPC damage
FR3	The system shall increase or decrease NPC spawn rate
FR4	The system shall increase or decrease NPC accuracy
FR5	The system shall increase or decrease NPC reaction times
FR6	The system shall increase or decrease player damage
FR7	The system shall increase or decrease item effectiveness
FR8	The system shall increase or decrease wall charger charge
FR9	The player shall be able to verify the active changes
FR10	The system shall track player accuracy
FR11	The system shall track player kills
FR12	The system shall track player deaths
FR13	The system shall track the current difficulty attempt time
FR14	The system shall save the active changes for the next session
FR15	The system shall import the changes from the last session
FR16	The system shall persist changes across different playthroughs
FR17	The system shall change what items are dropped to the player
FR18	The system shall determine which items are most needed by the player
FR19	The system shall increase and decrease difficulty rule weights
FR20	The system shall determine when to change the active rules
FR21	The system shall determine which rules to activate
FR22	The system shall submit gathered player information to the GameAnalytics server
FR23	The system shall activate at the start of the game

Having determined what the system must do, we must also determine how the system shall do it, keeping in mind factors such as extensibility, privacy, performance, and resilience. For this purpose, we list a set of non-functional requirements in Table 13.

Table 13 - Non-functional requirements

Non-functional Requirement No.	Non-functional Requirement Description
NFR1	The system shall not collect personal player data
NFR2	The system shall not inform the player of changes unless the player explicitly demands it
NFR3	The system shall be receptive to new rule types
NFR4	The system shall stay responsive when changing active rules
NFR5	The system shall be easy to install
NFR6	The system shall function when it receives rules with negative weight
NFR7	The system shall function when it receives multiple active rules of the same type
NFR8	The system shall correct rules with negative weight
NFR9	The system shall function without an internet connection

5 Implementation

To change the difficulty of the game we created two separate systems. The first system, inspired by dynamic scripting, uses a set of rules which modify the characteristics of NPC and other similar entities. The second system analyses the state of the player to determine which items are most useful in the current situation. These two systems work in tandem to enhance the user's overall experience throughout gaming sessions.

5.1 Tracking the player

The most important aspects to determine which actions to take regarding the difficulty are the player's actions and status. Only by knowing how the player has behaved thus far are we capable of determining if the difficulty is adequate for the player.

To keep track of all the player's statistics, we created the PlayerInfo struct, which tracks the player's current inventory state, health and armour levels, accuracy, kills and most importantly, the time between rule changes, the time between player deaths and how often the player has died between rule changes.

```
struct PlayerInfo {

	long lastRuleChangeTime = NAN;
	long lastDeathTime = NAN;
	int deathCount = 0;
	float health = 0;
	float armor = 0;
	int shotsFired = 0;
	int shotsHit = 0;
	std::string lastUsedWeapon = "";
	std::vector<const char*> killedMonsters;
	int healthLost = 0;

	bool has357Gun = false;
	bool has9mmGun = false;
	bool hasShotgun = false;
	bool hasCrossbow = false;
	bool hasEgon = false;
	bool hasGauss = false;
	bool hasMP5 = false;
	bool hasRPG = false;

	//<current ammo, max ammo>
	std::pair< int, int> count357 = {0,36};
	std::pair< int, int> count9mm = { 0,250 };
	std::pair< int, int> countBuckshot = { 0,125 };
	std::pair< int, int> countCrossbow = { 0,50 };
	std::pair< int, int> countUranium= { 0,100 };
	std::pair< int, int> countHandGrenade = { 0,10 };
	std::pair< int, int> countM203Grenade = { 0,10 };
	std::pair< int, int> countRPG = { 0,5 };
	std::pair< int, int> countSatchel = { 0,5 };
	std::pair< int, int> countTripmine = { 0,5 };
};
```

Figure 6 - Structure that holds player statistics

By using the game's CBasePlayer::UpdateClientData function, which exists to update the player's heads-up display (HUD) at each frame, we can also update the information on our struct pertaining to the player's inventory and health state.

Tracking the player's deaths is done in the CBasePlayer::Killed function which, as the name suggests, is already responsible for resetting several parameters after the player's health reaches 0. The RecordDeath function increases the death count in our struct and also pushes other relevant information to our analytics system, namely the level and coordinates where the player died, and the reason that lead to their death. We determined that the system should not act in case the player's death was a result of using the "Kill" console command, as well as when the player's death was a result of fall damage, as those deaths could not be prevented by changing the difficulty of the game and could be easily exploited by the player.

```
	const char* attackerName = STRING(pevAttacker->classname);
	const char* mapName = STRING(gpGlobals->mapname);

	if (strcmp(attackerName, "player")!= 0 && strcmp(attackerName, "worldspawn") != 0
&& strcmp(attackerName, "trigger_hurt") != 0) {
		RecordDeath(attackerName, mapName, pev->origin);
		RecalculateDifficulty();
	}
```

Figure 7 - Death tracking additions to the CBasePlayer::Killed function

Having access to all this information, as well as the time between deaths and time since the last rule change, we can now determine when taking a difficulty changing action is necessary, and which items are most useful to the player.

5.2 Difficulty Rules

The system that has the greatest impact in terms of changing difficulty is the DDA Rule system. This system consists of a list of 155 rules of 9 different types. At any given time, one rule of each type is active, allowing us to change the difficulty in several ways.

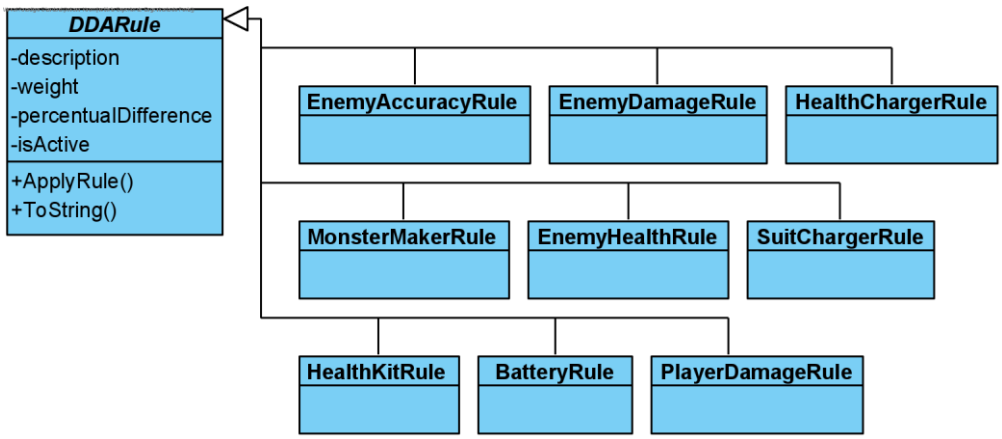

Figure 8 - Dynamic difficulty rules diagram

Each rule type extends from our base DDARule abstract class and is composed of the following attributes and functions:

- **Description**: used to in-game verify the active changes to difficulty.
- **Weight**: used in weighted random rule selection process.
- **PercentualDifference**: the degree to which the rule changes its respective parameter.

- **isActive**: indicates if this rule is currently active.
- **ApplyRule**(): changes the parameters of the game per the rules percentual difference
- **ToString**(): returns a string containing all the information about the rule, used in persisting the currently active rules.

The rules, shown in Table 14, are stored in a text file named huckleberry.dda and can be modified at any time, much like the original game difficulty settings which can be accessed and modified at the skill.cfg file. This allows us to rapidly deploy a different set of modifiers or change the number of rules for each type.

Table 14 - Rules and their modifiers

Rule Type	Modifier	Starts Active
EnemyHealthRule	-20	NO
	-15	NO
	-10	NO
	-5	NO
	0	YES
	10	NO
	20	NO
	30	NO
	40	NO
	50	NO
	60	NO
	70	NO
	80	NO
	90	NO
	100	NO
	110	NO
	120	NO
	130	NO
	140	NO
	150	NO
EnemyDamageRule	-25	NO
	-20	NO
	-15	NO
	-10	NO
	-5	NO
	0	YES
	10	NO
	20	NO
	30	NO
	40	NO
	50	NO
	70	NO
	80	NO
	90	NO
	100	NO
	110	NO
	120	NO
	130	NO
	140	NO
	150	NO
PlayerDamageRule	50	NO
	45	NO
	40	NO
	35	NO
	30	NO
	25	NO
	20	NO
	15	NO
	10	NO
	5	NO
	0	YES
	-5	NO
	-10	NO
	-15	NO
	-20	NO
	-25	NO
	-30	NO
	-35	NO
	-40	NO
	-45	NO

EnemyAccuracyRule	-50	NO
	40	NO
	50	NO
	60	NO
	70	NO
	80	NO
	90	NO
	100	YES

HealthKitRule	30	NO
	25	NO
	20	NO
	15	NO
	10	NO
	0	YES
	-10	NO
	-15	NO
	-20	NO
	-25	NO
	-30	NO
	-40	NO

HealthChargerRule	50	NO
	45	NO
	40	NO
	35	NO
	30	NO
	25	NO
	20	NO
	15	NO
	10	NO
	0	YES
	-10	NO
	-15	NO
	-20	NO
	-25	NO
	-30	NO
	-35	NO
	-40	NO
	-45	NO
	-50	NO

BatteryRule	50	NO
	45	NO
	35	NO
	30	NO
	25	NO
	20	NO
	15	NO
	10	NO
	0	YES
	-10	NO
	-15	NO
	-20	NO
	-25	NO
	-30	NO
	-35	NO
	-40	NO
	-45	NO
	-50	NO

SuitChargerRule	50	NO
	45	NO
	40	NO
	35	NO
	30	NO
	25	NO
	20	NO
	15	NO
	10	NO
	0	YES
	-10	NO
	-15	NO
	-20	NO
	-25	NO
	-30	NO
	-35	NO
	-40	NO
	-45	NO
	-50	NO

MonsterMakerRule	-30	NO
	-25	NO
	-15	NO
	-10	NO
	0	YES
	25	NO
	50	NO
	75	NO
	100	NO
	125	NO
	150	NO
	175	NO
	200	NO
	225	NO
	250	NO

These 155 rules are then imported into a single list in our DDAController, which to change the difficulty simply verifies which rules are active and calls upon their ApplyRule() function.

5.2.1 Enemy health rule

The first rule we will analyse is the rule that modifies enemy NPC hit points. The first step is to calculate the new maximum hit points for each type of enemy NPC. For that purpose, we store the new health in a separate structure from the original health, so that we can perform later calculations on the original value.

```
void EnemyHealthRule::ApplyRule()
{
	float percent = static_cast<float> (percentualDifference) / 100.0f;
	ddaMods.agruntHealth = gSkillData.agruntHealth * (1 + percent);
	ddaMods.apacheHealth = gSkillData.apacheHealth * (1 + percent);
	ddaMods.bullsquidHealth = gSkillData.bullsquidHealth * (1 + percent);

	// These lines repeat for the remaining enemy NPC
}
```

Figure 9 - ApplyRule function of the EnemyHealthRule class

Having determined the new maximum health for the enemy, we must calculate the corresponding current health. The CalculateMonsterHealth function receives the current enemies maximum hit points, current hit points and the type of enemy to calculate the new values based on the percentage of health the enemy had before the rule was applied.

```
float* CalculateMonsterHealth(float maxHP, float currentHP, const char*
className);
```

Figure 10 - Header of the CalculateMonsterHealth function

Now that we have determined the enemy's new health values, we simply replace them when the monster is initialised or when a save-game is restored, as the value is susceptible to change between when the game was saved and when the save game is restored.

```
void CBaseMonster::MonsterInit(void)
{
// Lines unrelated to modifications have been supressed

	pev->max_health = pev->health;

	float* healthParameters = CalculateMonsterHealth(pev->max_health, pev-
>health, STRING(pev->classname));

	if (healthParameters[0] != -1.0f) {
		pev->max_health = healthParameters[0];
		pev->health = healthParameters[1];
	}
```

Figure 11 - Health recalculation in the MonsterInit function

Should the monster that runs this function not be one of the enemies we are changing then the function returns -1 to indicate that the health values should not be changed, as can be seen in Figure 11.

5.2.2 Enemy and player damage rules

Contrary to the previous rule, the EnemyDamageRule and the PlayerDamageRule alter the data in the gSkillData structure. The ddaBaseEnemyDamages struct contains the original values on which we intend to act upon. This difference is because the game uses the data contained in the gSkillData struct when an entity is hit, unlike in the previous case where the changes had to be applied at the start. Altering the data in the struct is thus sufficient to modify the damage at any time.

```
void EnemyDamageRule::ApplyRule()
{
	float percent = static_cast<float> (percentualDifference) / 100.0f;
	gSkillData.headcrabDmgBite = ddaBaseEnemyDamages.headcrabDmgBite * (1 +
percent);
	gSkillData.hgruntDmgKick = ddaBaseEnemyDamages.hgruntDmgKick * (1 +
percent);
	gSkillData.houndeyeDmgBlast = ddaBaseEnemyDamages.houndeyeDmgBlast * (1 +
percent);
	gSkillData.monDmg12MM = ddaBaseEnemyDamages.monDmg12MM * (1 + percent);
	gSkillData.monDmg9MM = ddaBaseEnemyDamages.monDmg9MM * (1 + percent);
	gSkillData.monDmgHornet = ddaBaseEnemyDamages.monDmgHornet * (1 +
percent);
	gSkillData.monDmgMP5 = ddaBaseEnemyDamages.monDmgMP5 * (1 + percent);

//These lines repeat for the remaining enemy NPCs and their different attacks
}
```

Figure 12 - ApplyRule function of the EnemyDamageRule class

```
void PlayerDamageRule::ApplyRule()
{
	float percent = static_cast<float> (percentualDifference) / 100.0f;
	gSkillData.plrDmg357 = ddaBasePlayerDamages.plrDmg357 * (1 + percent);
	gSkillData.plrDmg9MM = ddaBasePlayerDamages.plrDmg9MM * (1 + percent);
	gSkillData.plrDmgBuckshot = ddaBasePlayerDamages.plrDmgBuckshot * (1 +
percent);

	//These lines repeat for the remaining player weapons
}
```

Figure 13 - ApplyRule function of the PlayerDamageRule class

5.2.3 Enemy accuracy rule

This rule implements a feature that was not present in the base game, namely enemy NPC missing shots on purpose. As usual, we store the percentage of shots that should hit in our structure.

```
void EnemyAccuracyRule::ApplyRule()
{
	float percent = static_cast<float> (percentualDifference) / 100.0f;
	ddaMods.accuracyMod = percent;
}
```

Figure 14 - ApplyRule function of the EnemyAccuracyRule class

Having determined the percentage of shots that should hit we use the ShouldHit function to determine when, as the name implies, the enemy should hit their shot.

```
bool ShouldHit()
{
	bool didHit[100] = { false };
	int percentage = ddaMods.accuracyMod * 100;
	for (int i = 0; i < percentage; i++) {
		didHit[i] = true;
	}
	int pos = rand() % 100;
	return didHit[pos];
}
```

Figure 15 - Function that determines if a shot should hit based on the current accuracy modifier.

Lastly, we modified the ShootAtEnemy function to include a call to our ShouldHit function. If the enemy is supposed to hit the player, then the function continues as it normally would, otherwise we modify the vector that is supposed to represent the player's position, making the enemy fail their shot.

```
Vector CBaseMonster::ShootAtEnemy(const Vector &shootOrigin)
{
	Vector lastKnownPosition = m_vecEnemyLKP;
	Vector missVector;
	bool shouldHit = ShouldHit();
	CBaseEntity *pEnemy = m_hEnemy;
	if (pEnemy)
	{
		if (shouldHit) {
			return ((pEnemy->BodyTarget(shootOrigin) - pEnemy->pev-
>origin) + lastKnownPosition - shootOrigin).Normalize();
		}
		else {
			return ((pEnemy->BodyTarget(shootOrigin) - pEnemy->pev-
>origin) + lastKnownPosition - shootOrigin + missVector).Normalize();
		}
	}
	else
		return gpGlobals->v_forward;
}
```

Figure 16 - ShootAtEnemy function of the CBaseMonster class

5.2.4 Health kit and battery rules

Two of the simpler rules, the HealthKitRule and BatteryRule work by simply modifying the information contained in our structure based on the percentage we want the items to be more or less effective. Then we replaced the calls to the gSkillData struct with our own to change the effectiveness of the two items.

```
void HealthKitRule::ApplyRule()
{
	float percent = static_cast<float> (percentualDifference) / 100.0f;
	ddaMods.healthkitCapacity = gSkillData.healthkitCapacity * (1+percent);
}

void BatteryRule::ApplyRule()
{
	float percent = static_cast<float> (percentualDifference) / 100.0f;
	ddaMods.batteryCapacity = gSkillData.batteryCapacity * (1+percent);
}
```

Figure 17 - ApplyRule function of the HealthKitRule and BatteryRule

5.2.5 Health charger and suit charger rules

The suit charger and health charger rules function similarly, as is shown in Figure 18. We calculate what the new maximum value for the chargers should be based on the specified percentual difference and the capacity contained in the base game's `gSkillData` struct, which holds the base values upon which we want our changes to occur.

```
void HealthChargerRule::ApplyRule()
{
	float percent = static_cast<float> (percentualDifference) / 100.0f;
	ddaMods.healthchargerCapacity = gSkillData.healthchargerCapacity *
(1+percent);
}

void SuitChargerRule::ApplyRule()
{
	float percent = static_cast<float> (percentualDifference) / 100.0f;
	ddaMods.suitchargerCapacity = gSkillData.suitchargerCapacity * (1+per-
cent);

}
```

Figure 18 - ApplyRule function of the HealthChargerRule and SuitChargerRule classes

Having determined the new value for the chargers' maximum charge we must then allow the charger to update its maximum charge and adjust its current charge accordingly. For this purpose, we also created a new attribute in the charger classes as they initially only stored the current charge, having no information on the maximum charge.

```
int * CalculateWallChargerJuice(int maxJuice, int currentJuice, const char *
className)
{

	int juiceParameters[] = { 0, 0 };
	if (maxJuice == 0) {
		return juiceParameters;
	}

	if (strcmp(className, "func_healthcharger") == 0) {
		juiceParameters[0] = ddaMods.healthchargerCapacity;
	}
	else {
		juiceParameters[0] = ddaMods.suitchargerCapacity;
	}
	juiceParameters[1] = (juiceParameters[0] * currentJuice) / maxJuice;
	return juiceParameters;
}
```

Figure 19 - Function that determines the charge of health and armour stations.

5.2.6 Monster Maker Rule

The MonsterMakerRule is responsible for changing the number of enemies that spawn at select places in the game's maps. Using an entity named CMonsterMaker we modify the maximum number of enemies that can spawn and also the maximum number of enemies that can be present simultaneously.

```
void MonsterMakerRule::ApplyRule()
{
	float percent = static_cast<float> (percentualDifference) / 100.0f;
	ddaMods.monsterMakerMod = percent;
}
```

Figure 20 - ApplyRule function of the MonsterMakerRule class

After storing the active percentage in our structure, we use the function represented in Figure 21 to calculate the new maximum amount of enemies that can be present and the amount that is allowed to be present simultaneously, based on the modifier and the enemies that are present at the time when the rule is activated.

```
int*    CalculateMonsterMakerLimit(int    maxCount,    int    currentCount,    int
maxAliveCount);
```

Figure 21 - Header of the CalculateMonsterMakerLimit function

Finally, we modified the Spawn function to include a call to our CalculateMonsterMakerLimit function so that our changes are applied, with the condition that if the maximum number of enemies to spawn is equal to 0 then the process can be skipped as nothing would be changed.

```
void CMonsterMaker::Spawn()
{
	huck_MaxNumMonsters = m_cNumMonsters;
	m_cLiveChildren = 0;
	int* limitParameters = CalculateMonsterMakerLimit(huck_MaxNumMonsters,
m_cNumMonsters, m_iMaxLiveChildren);
	if (limitParameters[0] != 0) {

		huck_MaxNumMonsters = limitParameters[0];
		m_cNumMonsters = limitParameters[1];
		m_iMaxLiveChildren = limitParameters[2];
	}

	//Lines unrelated to modifications have been supressed
```

Figure 22 - Spawn Function of the CMonsterMaker class

Figure 23 shows us two instances of the same encounter, on the left side of the image, with increased spawn rates we can count 5 enemy NPC, while on the right, with decreased enemy spawn rates, a lone enemy NPC can be encountered.

Figure 23 - Encounter with increased and decreased enemy spawn rates.

5.3 Determining when to take action

The change of which rules are active, and their weight can be triggered in three different ways: repeated player death, check-point completion or level completion. Depending on the player's

statistics we can then determine if the attempt was successful, which results in an increase of the active rules weight, or unsuccessful, where the active rules are punished while the inactive rules receive a weight increase depending on if the difficulty should be increased or lowered. For this solution, we considered that a checkpoint was reached when the game performed its auto-save function.

```
float ShouldChangeDifficulty()
{
	long AttemptTime = GetTime() - DDAPlayerState->info.lastRuleChangeTime;
	long LastDeathTime = GetTime() - DDAPlayerState->info.lastDeathTime;
	int AttemptDamage = DDAPlayerState->info.healthLost;
	int deathCount = DDAPlayerState->info.deathCount;

	if (deathCount >= 3) {
		if (AttemptTime < 60 * 5) {
			return 0.5;
		}
		else {
			return 0.8;
		}
	}
	else if ((LastDeathTime > 60 * 10 || AttemptDamage >= 250)) {
		if (AttemptDamage <= 200 || LastDeathTime > 60 * 15) {
			return 2;

		}
		else {
			return 1.5;
		}
	}
	else if (deathCount == 1 && LastDeathTime > 60 * 5) {
		return 1;
	}
	else {
		return 0;
	}

}
```

Figure 24 - Function that determines if a difficulty changing action is necessary

Determining the success of the run is done by analysing how often the player has died since the last rule change, how much damage the player has received and the time between player deaths. A player who rarely dies is considered to have an adequate challenge, while a player who dies repeatedly or does not die at all for an extended period, is considered to have an unmatching challenge. Finally, we also have the option to delegate the rule change to the next trigger in case nothing transpired since the last rule change.

While it may appear contradictory to increase the difficulty for players who receive more damage instead of decreasing it, this decision was reached after performing tests that

revealed that it was the players who had a better performance who would accumulate the most damage. Meanwhile, players who were dying more often would often die due to starting a run with already decreased health and not reach an acceptable damage level to act upon.

5.4 Modifying rule weights

Should the system determine that an action must be taken to modify the weights of our difficulty rules, then we can encounter one of three scenarios. In the first scenario, where the rules are considered to be adequate, we increase the weight of the active rule and the two rules that surround it by a flat value of 10.

In the second scenario, where we determine that the selected rules created a challenge the player could not overcome, we use the following two expressions to increase the weight of easier rules and decrease the weight of more difficult rules.

$$W = \frac{W' + 10}{|d| \times c}, \forall\, d \in \mathbb{R}_{<0}$$

$$W = \frac{W' - 10}{d} \times c, \qquad \forall\, d \in \mathbb{R}_{>0}$$

Where W is our new rule weight, W' represents our old weight, d represents the distance between the current and active rules and c represents the value obtained from our `ShouldChangeDifficulty` function, which in this case is a value between 0 and 1.

Lastly, in case the offered difficulty was not sufficient to challenge the player, we use two expressions similar to the previous ones to decrease the weight of easier rules and increase the weight of more difficult ones, based on their distance to the active rule.

$$W = \frac{W' + 10}{|d| \times c}, \forall\, d \in \mathbb{R}_{<0}$$

$$W = \frac{W' - 10}{d} \times c, \forall\, d \in \mathbb{R}_{>0}$$

Where d, again, represents how far the rule is in the ordered rule list and c represents the value obtained from our `ShouldChangeDifficulty` function, which in this case can be any value greater than 1.

Following any weight modifying action, we determine the corresponding percentage the weight represents in the total sum of weights for that rule type and set that value as the rule's weight.

5.5 Changing item drops

Our second system, responsible for determining and modifying item drops, works by analysing the player's health and armour status and inventory to ascertain what items would be most beneficial to the player. Items are dropped to the player on certain NPC death and when the player breaks pre-determined crates. Using the information contained in the PlayerInfo struct, we attribute a weight to each item that we could drop and select the item using a weighted random selection. If the player's health and armour levels are high, we restrict healing items from spawning, likewise, we only select ammunition for weapons the player already possesses and for which they do not already carry the maximum allowed amount. We also determined that we would not include alien items or any weapons in our list of items as giving these items to the player would interfere with the game's narrative and progression.

Figure 25 - Screenshot of an item drop with low player health.

Figure 26 - Screenshot of an item drop with high player health and armour.

When the game was naturally going to drop an item to the player, we compare the items name to the list of items we selected to be possible to drop. If the name matches one of the items which we can replace, we select a new item from our list, otherwise, we drop the item that was going to be dropped without our intervention. We use the item's name as that is how the game's CREATE_NAMED_ENTITY function works to spawn new items into the map.

5.6 Analytics

Our analytics system, created to gather data that allows us to make inferences about the behaviour of the players and how to further enhance the difficulty modifying system, makes use of the service offered by GameAnalytics. As the events that we are tracking occur in-game, we generate a string representation of that event that is then pushed to the GameAnalytics server. The GameAnalytics SDK, responsible for the communication between our local machine and the remote server gathers these events and submits them every 8 seconds.

Should the server be unreachable, then the events are stored locally so they can be pushed again at a later date.

Each event string is composed of two or more fields, separated by colons. The first field represents the type of event that is being recorded, while the later fields record additional information of that event, additionally, events can also have a numerical value associated with them. For example, our event that tracks player deaths is composed of three fields and a numerical value, taking the following form: EVENT_DEATH:LEVEL_NAME:KILLER_NAME, with the numerical value representing the coordinates of the player's death. We transformed the coordinates into a single value due to a limitation of the GameAnalytics system that only allows us to create ten thousand different events and the fact that all the possible permutations would rapidly exceed that limit.

Figure 27 - Deaths in level c1a3 recorded from 01/06/2021 to 14/06/2021

Figure 27 shows us one of the possible charts which allow us to identify the leading cause of death in the first part of the “We’ve got Hostiles” level and verify if any of the causes might be overrepresented considering their dispersion in the level.

We also use the events to show the player how they have been performing throughout each session if they so wish, which also informs the player of the rules that are currently impacting their game, as can be seen in Figure 28.

Figure 28 - In-game console with the player's current statistics

6 Testing and Evaluation

In this chapter, we will outline how the development of this project will be measured and evaluated. Once the project is in a usable state, we will perform a pilot test to ascertain how the project performs in a real-world scenario, that will be measuring the project's usability using the System Usability Scale (SUS) and general player satisfaction and experience using the Game Experience Questionnaire (GEQ). The pilot test will be performed by allowing a group of volunteers to play the game for an hour using the DDA modification and another hour without the modification. Following each hourly session, the volunteers will be asked to fill out the SUS and GEQ forms which allow us to compare the results between each session. The order of using or not using the modification will be alternated between the participants to reduce any kind of bias to their responses. Using game analytics, we will also be able to gather statistics (such as player deaths and item usage) on the player sessions that used DDA, which allows us to further refine what actions should be taken and when to take them. Throughout the development of the system, we will also be applying the Quantitative Evaluation Framework (QEF), a tool that tracks the development and quality of the system, allowing us to control the state of the project. The following sections offer an overview of these tools, as well as the hypothesis which is to be tested and the criteria on which the project will be evaluated.

6.1 Hypothesis and Criteria

To test if the development of a DDA algorithm contributes to higher player satisfaction, the hypothesis must verify if the player's experience is enhanced by the adjustment of the

difficulty. The hypothesis will be verified by measuring two criteria. The first criteria will be the usability of the system, using the SUS, while the second criteria will be the player's experience in terms of challenge and competence, using the GEQ.

Null Hypothesis (H_0) – Using the developed DDA algorithm does not enhance the player's experience.

Where H_0 means that despite using the DDA algorithm there was no meaningful difference in the player's responses on the SUS and GEQ, meaning that the player's experience did not change.

Alternative Hypothesis (H_1) – Using the developed DDA algorithm enhanced the player's experience.

Where H_1 means that after using the DDA algorithm there was a meaningful difference in the player's responses on the SUS and GEQ, meaning that the player's experience changed by adjusting the difficulty of the game.

6.2 Assessment of the player's experience

As stated in the introduction to this chapter, the evaluation of this project's effect on the player's experience will be performed by subjecting a group of players to a pilot test. These players will be selected from volunteers that match our target audience, namely men or women between the ages of 18 and 30 years of age who have prior experience with FPS games. The pilot test consists of subjecting the players to two one-hour play sessions of the game and collecting their responses to the SUS and the GEQ. In one session, the player will be presented with a version of the game that uses the DDA algorithm, which was developed for this project, while during the other session, the player will be playing on an unmodified version of the game. The volunteers will not be told which version of the game they are playing in each session and the order they are presented with each version of the game will be alternated to reduce bias in their perception. By comparing the results from each session, we can then verify the performance of the algorithm in improving the player's experience and the general system's usability. After each session, the players will be asked to fill out two forms. The first form, assessing the usability of the game, will be done by responding to the SUS, the second form will assess the player's experience in playing the game, by responding to the GEQ.

6.2.1 System Usability Scale

The System Usability Scale (SUS) (Brooke, 1995) is a ten-item scale that measures the global usability of a system and is at its essence a specific version of a Likert scale. Using the SUS, the respondent states if he agrees or disagrees with a given statement on a scale from 1 to 5, where 1 indicates that the respondent strongly disagrees, while 5 means that the respondent strongly agrees. The items on the SUS were selected in a way that generally, the respondent would agree strongly with half of the statements and disagree strongly with the other half, this was done so that the respondent would have to put in more effort when deciding if they agreed with each item.

The SUS is one of the more popular usability measuring tools because it is technology independent, meaning that it can be applied to a wide variety of different systems without modification and because it is quick to use for both the participants and the proponents of a study. Finally, the output of the SUS can be easily understood by not only the programmers of the system, but also non-technical people like project managers and other people who have little experience in usability (Bangor, et al., 2008).

One of the stronger drawbacks of using the SUS to measure usability is that it is difficult to apply in countries where English is not a native language. A study from 2006 showed that some respondents who were not native speakers of the English language failed to understand the meaning of some words, like “cumbersome” (Finstad, 2006). Other studies have also attempted to translate the SUS into different languages, like European Portuguese, but have found that the translated version of the SUS had low reliability (Martins, et al., 2015).

6.2.1.1 Scoring the SUS

Scoring the SUS is a complicated process as the alternating tone of the items and the initial score ranging between 0 and 100 might induce errors. The first step needed to calculate the correct usability score using the SUS is to adjust the scores of each item to a range between 0 (poorest rating) and 4 (best rating), taking into consideration the number of each item. Odd-numbered items, which have a positive tone, must have their raw score subtracted by 1, while even-numbered items must subtract their raw score from 5. After calculating the new score from each item, the scores are added together and multiplied by 2.5 to get the final SUS score of the system (Lewis, 2018).

Having calculated the SUS score, it is important to note that it does not function as a percentage. As stated by Jeff Sauro, the best way to interpret the score is to normalize it and convert it to a percentile rank (Sauro, 2011).

As can be seen in Figure 29, the scores of the SUS can be graded from F to A+, this process, which is similar to grading on a curve, is based on the distribution of the scores from 500 studies, which found that the average score to be 68. To achieve a score equal to the top 10%, the system would have to score above 80.3. A score of 70 is close to the average score of 68, meaning that it would be representative of about 50%.

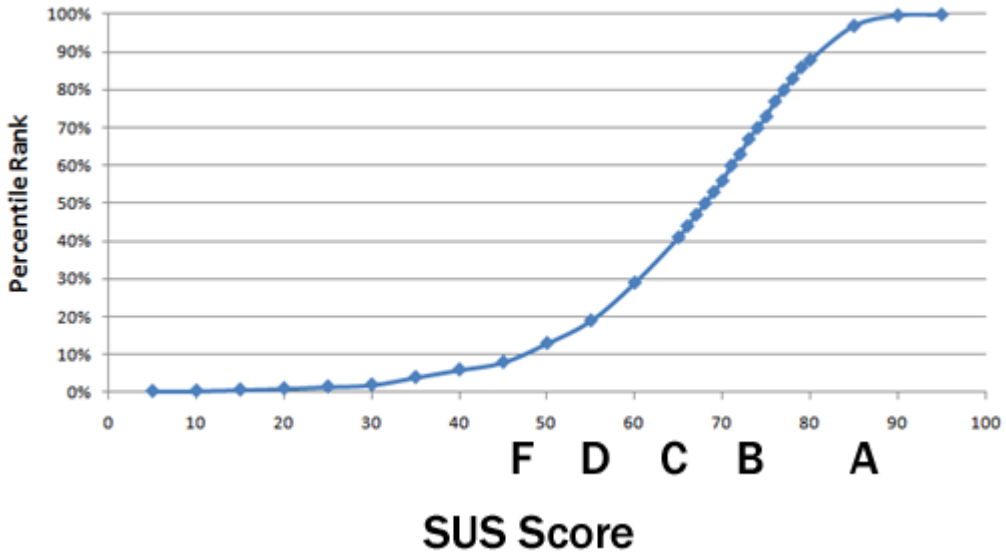

Figure 29 - Usability scores graded on a curve (Sauro, 2011)

6.2.2 Game Experience Questionnaire

The Game Experience Questionnaire (GEQ) (IJsselsteijn, et al., 2013) is a tool for characterising a player's experience. The GEQ is a modular questionnaire and should be answered as soon as a game session finishes. The first module, the "core questionnaire" assesses immersion, flow, competence, tension, challenge, and positive and negative effect. The second module focuses on the social aspect of the game, measuring the psychological and behavioural involvement of the player with the characters present in the game, and other players, should the game have multiplayer capabilities. Given the social aspect of this module, it should only be included when the game includes social entities. The last module, the "post-game module" evaluates the player's sentiments and thoughts once they stopped playing the game.

Knowing the player's experiences, it is then possible to adjust the game to reinforce certain aspects and adjust the methods used for DDA so that an optimal player experience can be achieved.

Despite the GEQ's popularity, some researchers have found that the GEQ's ability to measure the psychological state of the players is still questionable and that it is often used improperly. These authors note that if the tool is flawed, then so will be the conclusions drawn from the tool's usage (Law, et al., 2018) (Kent, 2013). However, the existence of these studies also reinforces the tool's popularity and by using the same tool as other developers it becomes possible to compare results, which would be impossible with a custom questionnaire that would still face some of the same problems as the GEQ. Taking into consideration all these aspects we still decided to use the GEQ as we consider that the benefits of its usage outweigh the potential drawbacks.

6.2.3 Game Analytics

To achieve a better understanding of how the players face each challenge, how they spend their resources and find the areas they struggle with the most, game developers use game analytics to track events within their games. Game Analytics thus allows the game developers to adjust the characteristic of encounters, map layouts or item placements to enhance a player's experience. By examining the data gathered it is also possible to identify problematic areas in the game as well as different approaches players take to overcome the challenges, they are presented with (Hick, et al., 2016).

During the development of Larian Studio's Divinity Original Sin 2, the studio implemented a system that allows the developers to visually represent the areas where each player character died. The heatmap, shown in Figure 30, then helped the developers to understand where the game was presenting the players with challenges that were too difficult to overcome, but also allowed them to see which areas the players were failing to explore and change the level design accordingly to hint the players that those areas were available to them. Game Analytics was also used to track the use of in-game abilities and items by the players so that the developers could understand which should be promoted (Vincke, et al., 2020).

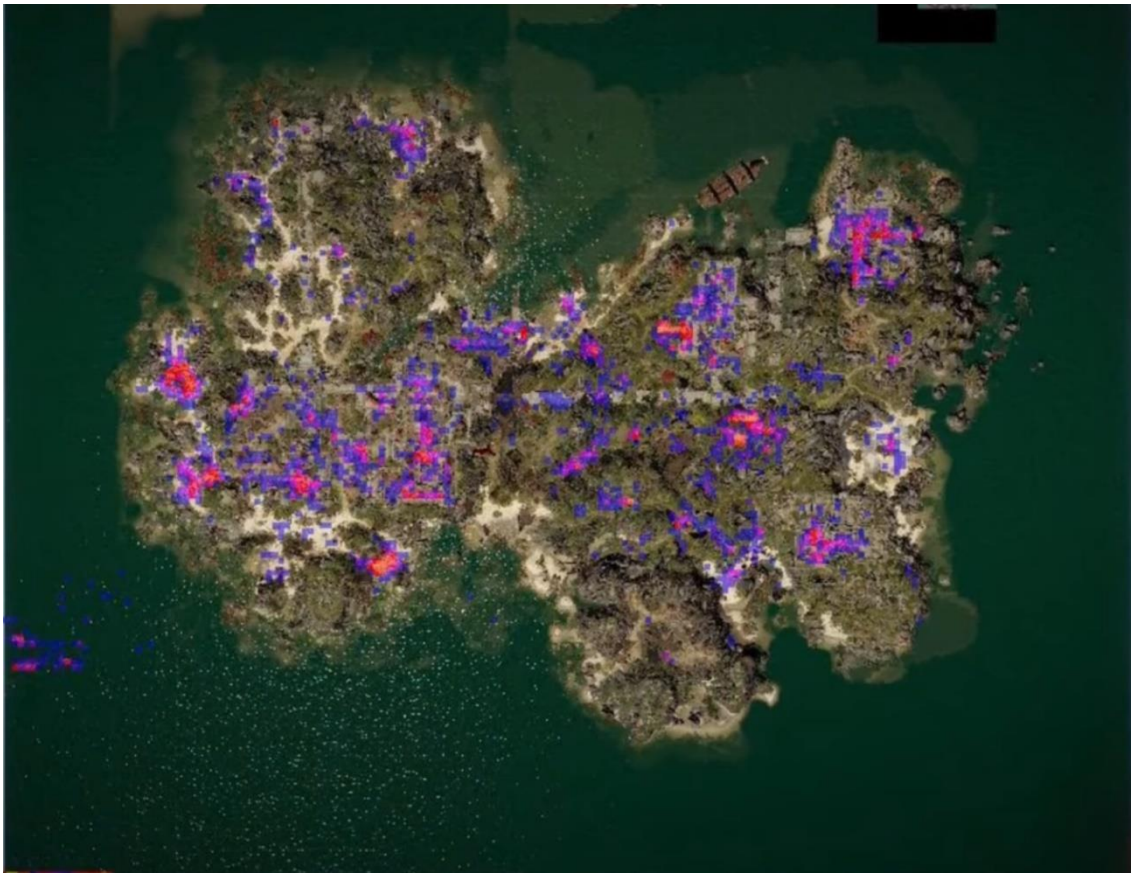

Figure 30 - Heatmap of player deaths in the Early Access version of Divinity Original Sin 2 (Vincke, et al., 2020)

Likewise, by implementing methods that allow us to track player statistics, namely their deaths, accuracy, and item usage, we can then use the data gathered from the experimental group of the pilot test to further our understanding of what actions lead to the player's success and when to take those actions, which then allows us to adjust our DDA algorithm accordingly.

6.3 Assessment of the project's development

To track the development and quality of the system, it is necessary to have a tool that allows us to measure said characteristics. For this purpose, we will be using the QEF (Escudeiro & Bidarra, 2008). This framework can be used at any time during the development process to ascertain if any problems emerge and thus solve them as they appear. In this model, the quality of the system is represented by a three-dimensional space, where each axis e representative of one of three domains. The first domain, the technical domain, measures operational aspects like content management, adaptability, and technical aspects of the

implementation. Secondly, the ergonomic domain measures aspects involving the usability and interactivity of the system. Finally, the pedagogic domain measures cognitive, affective, and psychomotor aspects.

Each domain can have a different weight in the quality of the project, depending on the project's characteristics. The quality of the project can then be measured by comparing the current state of the project to the best-case scenario.

To determine the quality of the project one must follow six steps:

1) **Classification of the criteria** – In this step, we will be determining all the features the project should have.

2) **Classification of the factors in each dimension** – This step attributes each criterion with a weight between 0 and 10, per the criterion's importance. A criterion with a weight of 10 is fundamental, while a criterion with a weight of 0 is irrelevant.

3) **Evaluation of the results** – This step is done throughout the development of the project and involves measuring the completeness of each determined criterion.

4) **Calculation of the achieved performance in each domain** – Taking into consideration only the criteria that affect each domain, one will, in this step, one will calculate the quality of one specific domain.

5) **Calculation of the global deviation** – This step now considering all three domains can now calculate the Euclidean distance between the current state of the system and the ideal scenario with coordinates (1,1,1).

6) **Calculation of the quality of the system** – This final step calculates the percentage to which the system fulfils the needs for which it was initially designed.

This first step in the evaluation phase will allow us to properly determine and classify the functionalities which should be developed, as well as their importance within the system. Table 15 offers an overview of the requirements for this project and their respective importance.

Table 15 - Quantitative Evaluation Framework

q	D	Q_i	Dimension	Qj	Wij (Factor Weight j in Dim i) [0,1]	Factor	rwjk (requirement weight k in Factor j) {2, 4, 6, 8, 10}	Requirement	wfk % requirement fulfilment k) [0,100]
0%	1.73	0	Technical	0	0.53	Gameplay	10	TG01 - Modify Player damage output	
							10	TG02 - Modify Enemy damage output	
							10	TG03 - Modify Enemy health	
							10	TG04 - Modify Enemy accuracy	
							10	TG05 - Modify Enemy speed	
							10	TG06 - Modify Enemy placement	
							10	TG07 - Modify Item placement	
							10	TG08 - Modify Health Item effectiveness	
							10	TG09 - Determine DDA actions to take	
				0	0.47	Analytics	8	TA01 - Track player weapon usage	
							8	TA02 - Track player ammunition usage	
							6	TA03 - Track player accuracy	
							8	TA04 - Track player kills	
							10	TA05 - Track player deaths	
							8	TA06 - Track player health	
							8	TA07 - Track player time spent on level	
							8	TA08 - Show current statistics in the console	
		0	Ergonomic	0	0.60	Usability	10	EU01 - The DDA implementation does not incur a drop in usability	
							10	EU02 - Modifications are seamless	
							10	EU03 - The game does not present untreated error messages to the user	
				0	0.40	Gameplay	10	EG01 - The game is challenging	

							10	EG02 - Game correctly assesses player skill level	
		0	Pedagogic	0	1.00	Affective	10	PA01 - Players show signs of competence in the game	
							10	PA02 - Players show signs of immersion in the game	
							10	PA03 - Players show signs of positive emotion	

6.4 Results

In this section we will discuss and evaluate the results obtained from our pilot study, considering the criteria established in 6.3. In total, fifteen people participated in the testing sessions.

6.4.1 Profile of the testers

The first group of questions serve to identify the characteristics of our testing population. Our respondent population consists of 15 males with ages ranging from 18 to 23.

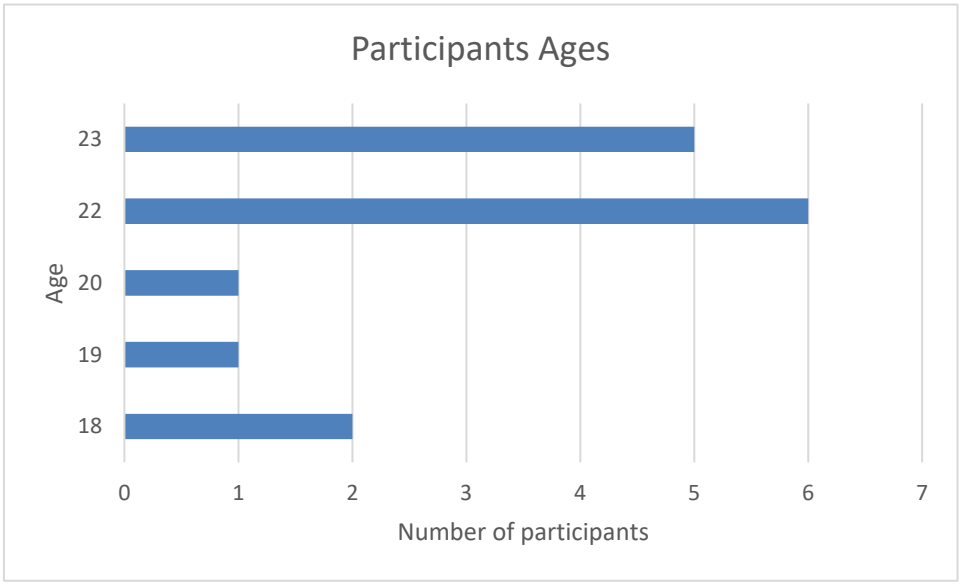

Figure 31 - Ages of the participants

Of these same fifteen, 53% had no prior exposure to the half-life game before the first session, while the remaining 47% had played half-life at least once at some point in their lives.

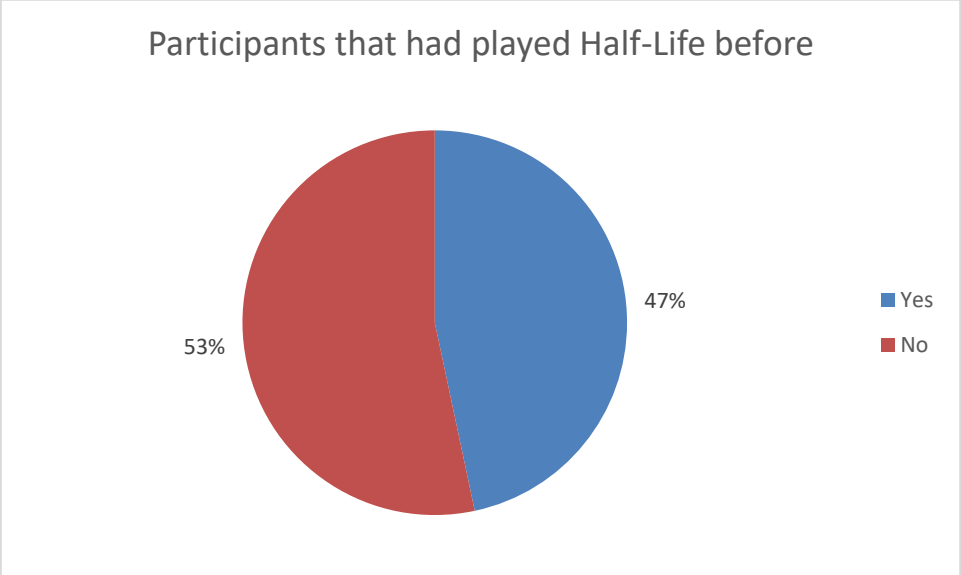

Figure 32 - Participants previous exposure to half-life

Lastly, the testers were inquired about their video-game habits, both in terms of the hours they usually spend per week playing video games and the difficulty setting they choose when they have the opportunity. 53% per cent of the respondents, the largest group, spend over 32 hours per week playing video games, followed by the groups of testers who play between 1 and 8 hours and 9 to 16, where both represent 20% each.

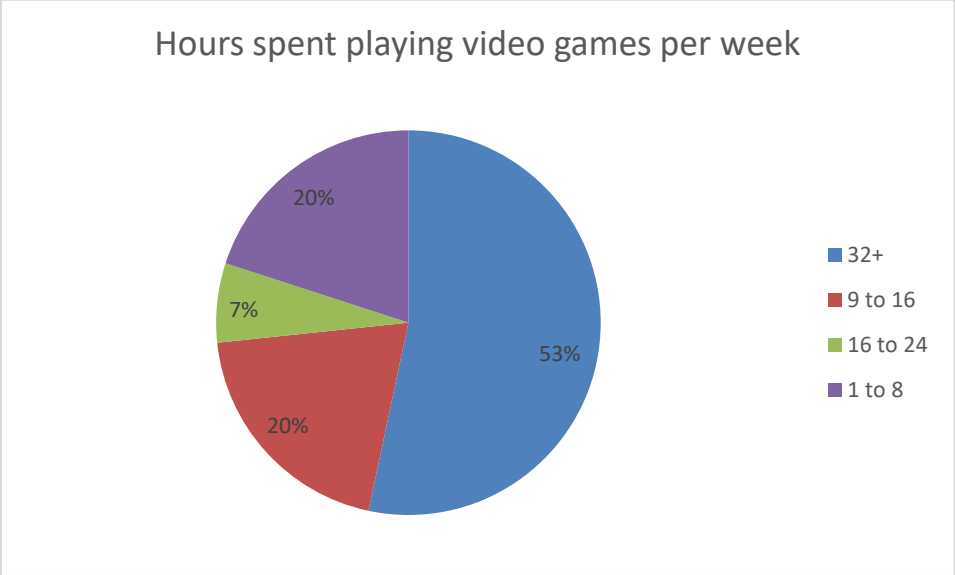

Figure 33 - Mean hours spent on video games by the participants

Regarding the preferred difficulty, the players preferred to play games on the medium setting or settings that are more difficult than the medium setting, with only one respondent preferring easier experiences.

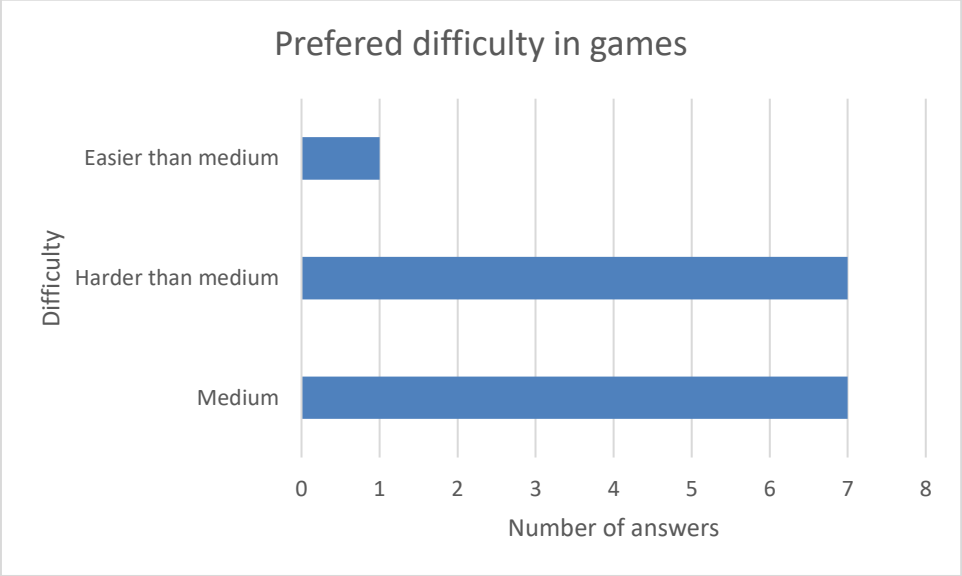

Figure 34 - Participants difficulty preferences

6.4.2 Responses to the GEQ

The second section of the survey focuses on the questions found on the GEQ, these questions are then divided into eleven categories: competence, immersion, flow, annoyance, challenge, negative affect, positive affect, positive experience, negative experience, tiredness and

returning to reality. The score for each category is obtained by calculating the average of the related questions for each person and then calculating the average of the previous averages.

The version of the GEQ that was employed is composed of two modules, for easier identification of the questions they have been named according to the module they belong to, where questions C1 to C33 belong to the core module of the GEQ and questions P1 to P17 belong to the post-game module.

Table 16 - Average responses to the GEQ

Metric	No DDA	DDA	Difference	Questions
Competence	2.95	2.6	-0.35	C2, C10, C15, C17, C21
Immersion	1.90	1.96	0.06	C3, C12, C18, C19, C27, C30
Flow	2.00	2.57	0.57	C5, C13, C25, C28, C31
Annoyance	0.56	0.98	0.42	C22, C24, C29
Challenge	0.96	1.84	0.88	C11, C23, C26, C32, C33
Negative Affect	0.97	0.87	-0.1	C7, C8, C9, C16
Positive Affect	2.71	2.87	0.26	C1, C4, C6, C14, C20
Positive Experience	1.67	2.02	0.35	P1, P5, P7, P8, P12, P16
Negative Experience	0.57	0.50	-0.07	P2, P4, P6, P11, P14, P15
Tiredness	0.47	0.70	0.23	P10, P13
Returning to Reality	0.76	0.93	0.17	P3, P9, P17

As can be verified in Table 16, where positive outcomes are highlighted in green and negative outcomes are highlighted in red, the DDA implementation fared better than the original version in 8 out 11 metrics. Where the most important failure of the implementation corresponds to the competence metric, however, since the metrics that we consider most important: flow, challenge and positive experience are positive, we consider the result to be overall beneficial.

6.4.3 Responses to the SUS

The last section of the survey is composed of the SUS questions and aims to measure the usability of both versions. Table 17 shows us the scores each version achieved for each question as well as the final result, which is calculated as described in 6.2.1.1. The final result shows a decrease in the usability of about 3.67 points, from 84.50 to 80.83, which can still be considered a moderately good result despite the decrease.

Table 17 - SUS Scores for the original version and the DDA implementation

Question	Result with DDA	Result without DDA
I think that I would like to use this system frequently.	2.33	2.53
I found the system unnecessarily complex.	3.33	3.53
I thought the system was easy to use.	3.27	3.47
I think that I would need the support of a technical person to be able to use this system.	3.40	3.93
I found the various functions in this system were well integrated.	2.93	2.73
I thought there was too much inconsistency in this system.	3.47	3.60
I would imagine that most people would learn to use this system very quickly.	3.40	3.33
I found the system very cumbersome to use.	3.53	3.53
I felt very confident using the system.	3.07	3.47
I needed to learn a lot of things before I could get going with this system.	3.60	3.66
Result	80.83	84.50

6.4.4 Validating the Hypothesis

The validity of the previous two sections hinges on the rejection of the initially proposed null hypothesis, which stated that there was no statistically significant difference in the scores of the SUS and GEQ. For this purpose, we conducted twelve paired two-tailed student's t-tests (The Editors of Encyclopaedia Britannica, 2020), to determine if the mean difference is equal to 0. These tests allow us to determine if the mean difference between the two groups is statistically different, allowing us to make inferences about the two groups. We conducted one t-test for each of our metrics and the system usability scale. Since we are studying twelve hypotheses at once we apply a Bonferroni correction (Haynes, 2013) to compensate for the increased likelihood of making erroneous inferences. With an initial significance level of 0.05, for a confidence level of 95%, we thus divide the significance by the number of simultaneous hypotheses to obtain:

$$\alpha = \frac{0.05}{12} \cong 0.004$$

Using this new α value we conduct our t-tests with 14 degrees of freedom to obtain the following test statistics and critical value:

Table 18 - Test Statistics and Critical value for the 12 hypotheses

Metric	Test Statistic	Critical Values	Rejects null hypothesis
Competence	-1.31850	+/- 3.437867	No
Immersion	0.15288		No

Flow	1.94169		No
Annoyance	1.92491		No
Challenge	4.13656		Yes
Negative	-0.47712		No
Positive	0.63973		No
Positive Experience	1.11429		No
Negative Experience	-0.69481		No
Tiredness	1.10092		No
Returning to Reality	2.08555		No
SUS	0.37292		No

Since our test statistics are lower than the critical value for each of our hypotheses, we cannot reject the null hypotheses that the true difference in means is 0 and thus not conclude that the means are significantly different. With the exception of the Challenge factor, where the test statistic is more extreme than the critical value and it is thus safe to reject the null hypothesis.

6.4.5 Quantitative Evaluation Framework Results

Having compiled all the information from the surveys we can now use the QEF to make a final assessment of our project's success and completion. In the metrics where surveys were involved, if the difference in scores was beneficial to the version with DDA we attributed a requirement fulfilment value of 100 and a value of 0 if the score was beneficial to the unaltered version.

In the technical dimension, we achieved a quality of 94.12%, missing only the implementation of requirement TG05. Moving on to the ergonomic dimension we tied the result of requirements EU02, EG01 and EG02 to the result of the SUS and the challenge and flow metrics of the GEQ respectively, obtaining a quality of 80%. Lastly, in the pedagogic dimension, we tied all three requirements to the surveys, where PA01 corresponds to the competence metric of the GEQ, PA02 corresponds to the immersion metric and PA03 results from the positive affect, negative affect, positive experience, and negative experience metrics. Compiling the quality of each dimension according to their weight we get a final quality rating of 85%. A complete version of the QEF can be consulted in Annex A.

6.5 Inferences from the game analytics data

Game analytics help us to determine how to further improve our implementation. By analysing the data gathered in our testing session we can make the following inferences:

- Players prefer using the submachine gun or pistol over the shotgun.
- Human soldiers NPC are overrepresented in the player's death causes.
- Players die too often in the level sections c1a2a[1] and c1a3a[2].

Regarding the first point, data shows that despite the shotgun being available to the player shortly after the pistol and much earlier than the submachine gun, players still refrained from making use of this weapon. Figure 35 shows that the pistol is chosen more than twice as often as the shotgun, despite being available at almost the same time and the submachine gun is also chosen two thirds as often as the shotgun, despite only being available from the third level onward.

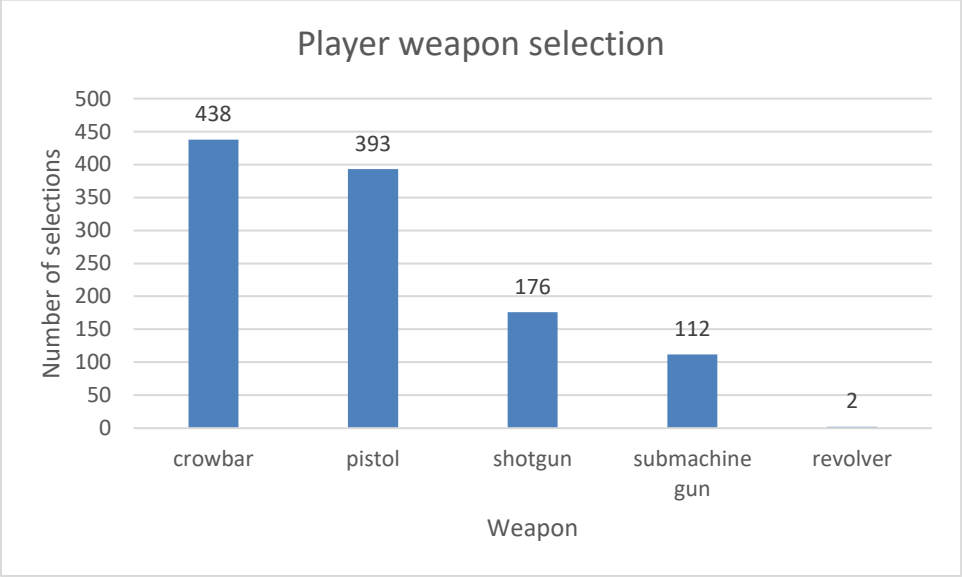

Figure 35 - Number of weapon selections

Knowing now that the shotgun is one of the least favourite weapons, we could change the drop rate of shotgun ammunition to reflect its popularity and increase the drop rate of

[1] C1a2a refers to the second section of the level named "Office Complex".
[2] C1a3a refers to the second section of the level named "We've Got Hostiles".

ammunition for the weapons the player makes use of. Alternatively, we could create a difficulty rule that specifically alters the shotgun's properties to make it more attractive to the players.

The second point is evidenced by Figure 36. Considering that the soldier NPC only appears during the third level and the fifth and that most players did not reach the fifth level, the fact that they are the leading cause of player death is a motive for concern. This problem could be resolved by creating a difficulty rule that targets this type of NPC specifically, making it so that only this NPC is altered while the others retain the characteristics applied by the current rule system.

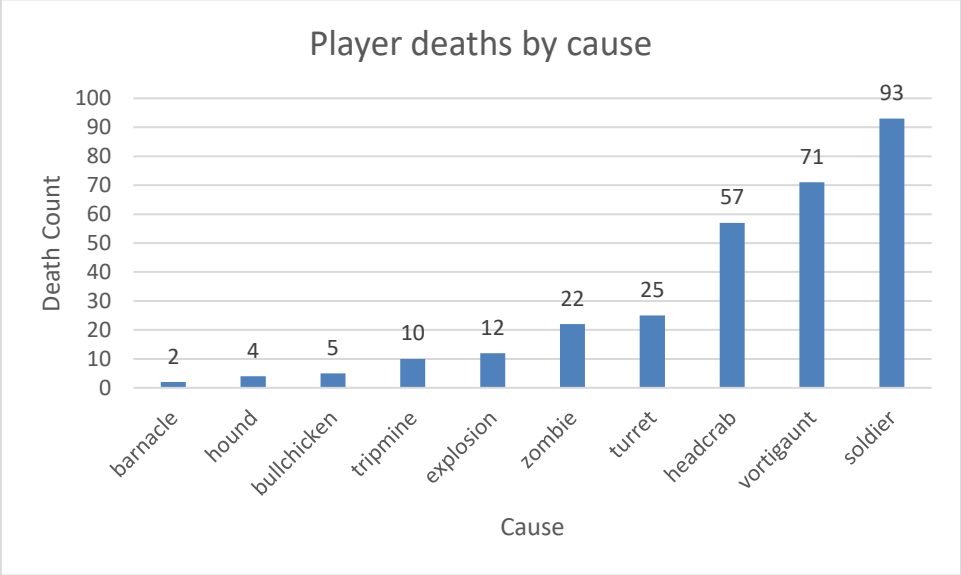

Figure 36 - Player death causes

The last point, which concerns the second sections of the second and third levels, is based on the notion that most player deaths occurred during those two sections.

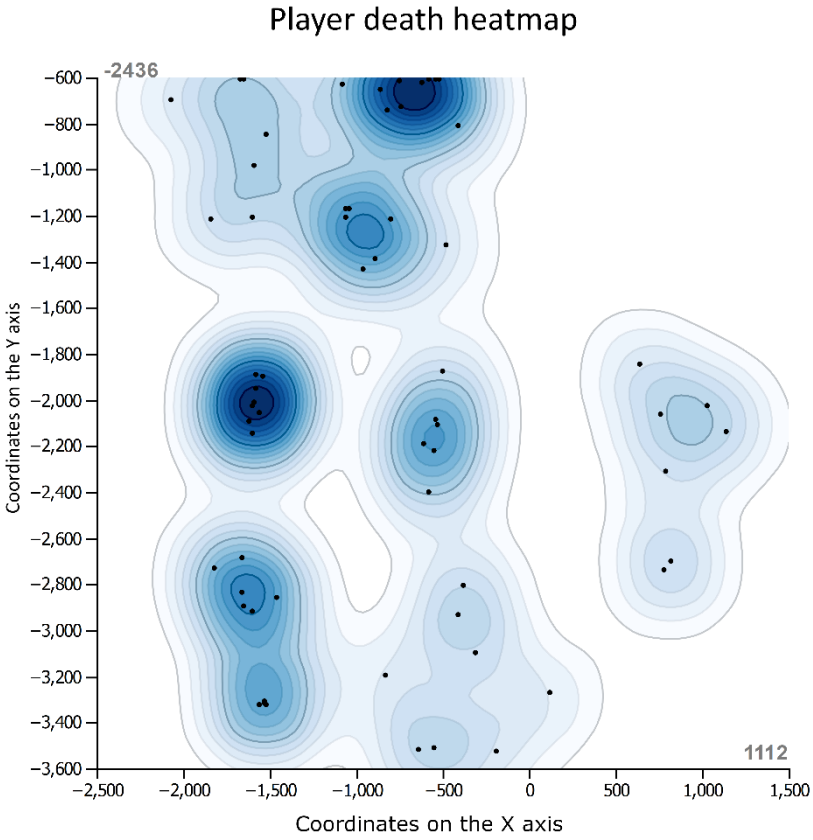

Figure 37 - Player death heatmap for section c1a3a

Figure 37, which represents the locations where the players died in the second section of the third level indicates that there are two instances where players are dying more often. Analysing the coordinates, we can relate them to two specific situations, one which relates to a trap the player must solve, and thus is expected to have a higher death count and one related to an encounter with enemy soldier NPC.

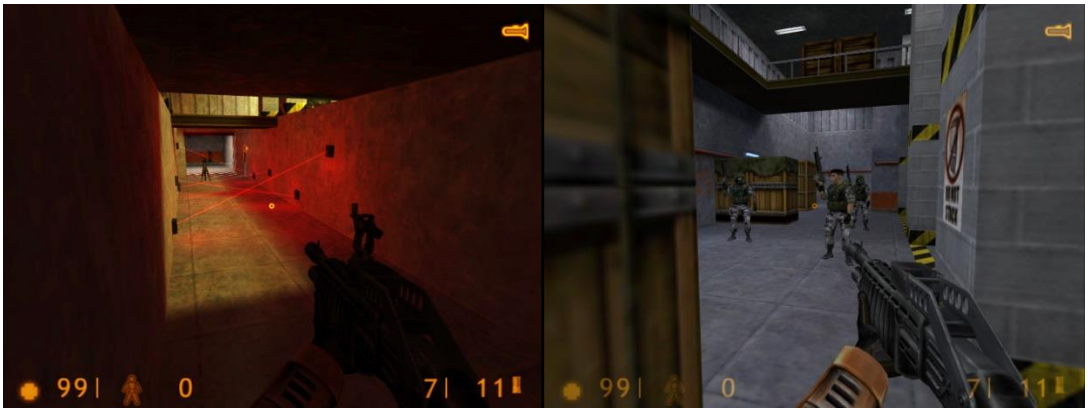

Figure 38 - Most lethal encounters in section c1a3a

By applying the changes to the soldier NPC discussed in the second point this scenario can most likely be averted, however.

Regarding the high death count in section C1a2a, this can be explained by the fact that the players are ambushed multiple times in that section and are thus supposed to have a more difficult time in completing that section. Players are often surprised by a new enemy NPC while already confronting several others which forces the players to adopt a new strategy, until said strategy is adopted most players will perish repeatedly.

7 Conclusions

In this chapter we will look back at all the work done during the development of this project, judging the implemented solution on its merit, and highlighting its faults. First, we will list the objectives that we achieved. Following up we will offer a brief description of the implemented solution and finish with the limitations our system incurs upon due to its design. Once this analysis is done, we will offer some suggestions on how further research could be conducted on this topic and highlight the most limiting factors in the development of this research.

7.1 Proposed vs achieved goals

Here we will list the initial objectives proposed by the project:

- Study implementations of dynamic difficulty
- Design and implement a dynamic difficulty system
- Test the implemented system

Having listed all the proposed objectives, we will now list what was achieved:

- Several different DDA implementations were studied, ranging from models that tried to predict user behaviour, to rule-based systems and even the use of artificial intelligence agents. From the conclusions of those projects, we determined how our solution could change the difficulty to the player's needs.

- A two-part DDA system was designed based on a weighted random rule system and item dispensing to the player according to their state.

- We ran several test sessions of our implementation. Volunteers were subjected to two one-hour play sessions where they would at random play the original version of the game and the modified version. After each session, the volunteers answered a survey composed of two sections, the first section revolving around the players' experience, using the GEQ, and the second section focusing on the usability of the system, using the SUS.

7.2 The solution

For the implementation of our DDA system, two subsystems were created. For the first system, where we select difficulty modifying rules based on their success, rules of nine different types were created, each with multiple modifiers to ensure that multiple permutations were possible and increase the adaptability of the system. Rules that were deemed to be corresponding to the player's skill were rewarded with a weight increase and thus selected more often, while rules that were deemed either too easy or too difficult were punished with a weight decrease, lowering their probability of being selected.

The second subsystem modifies item dispensing to guarantee that the player always has the items that he needs we created a system that changes the items that are given to the player depending on their health and inventory state, making it so that if the item has no use to the player in the current state, then it is not dispensed.

The solution was then tested by surveying a group of 15 volunteers for their overall experience and the usability of the system. The Player's responses indicated that the system had a positive effect in stimulating player flow, immersion and positive emotion and that players felt challenged when playing, however, the results also show a decrease in the player's perception of their competence.

7.3 Future work and limitation

In the conduction of this research, the most limiting factor was the limited and sometimes incorrect documentation of the GoldSrc engine. The existing documentation of the GoldSrc engine often references implementations of the newer source engine, which do not apply to how the GoldSrc engine functions. Other limiting factors include the implementation of the GameAnalytics SDK which only allows one associated value for each event, forcing us to create a workaround for the tracking of the coordinates of player deaths and the old code base of the Half-Life SDK, which due to its age uses several deprecated functions.

Future work should be conducted on better documented, but still modifiable, game engines, like the Source engine. This engine is still similar to the one used in this project but both more recent and better documented. There also exists a large community around modifications for games on this engine which can be an advantage when solving problems.

Regarding the rule system, future work could implement more specific types of rules, which only affect certain kinds of NPC, increasing once again the adaptability of the system.

Work should also be conducted on testing the efficacy of this type of system on players who prefer easier experiences as that group was underrepresented in the testing phase of this project.

This work originated two publications (still awaiting approval) submitted to the GALA 2021 - Games & Learning Alliance Conference and the indexed journal Multimedia Tools and Applications (Digital Games Track).

8 References

Aghdaie, N. et al., 2017. *DYNAMIC DIFFICULTY ADJUSTMENT.* United States of America, Patent No. 20170259177.

Bangor, A., Kortum, P. T. & Miller, J. T., 2008. An Empirical Evaluation of the System Usability Scale. *International Journal of Human-Computer Interaction,* pp. 574-594.

Booth, M., 2009. *The AI Systems of Left 4 Dead.* s.l.:Valve Software.

Brooke, J., 1995. SUS: A quick and dirty usability scale. *Usability Eval. Ind.,* Volume 189.

Capcom, 2005. *Resident Evil 4: The Official Strategy Guide.* s.l.:Future Press Verlag und Marketing GmbH.

Csikszentmihalyi, M., 1990. *Flow: The Psychology of Optimal Experience.* s.l.:Harper.

Escudeiro, P. & Bidarra, J., 2008. Quantitative Evaluation Framework (QEF). *Revista Ibérica de Sistema e Tecnologias de Informação,* Issue 1, pp. 16-27.

Finstad, K., 2006. The System Usability Scale and Non-Native English Speakers. *Journal of Usability Studies,* 1(4), pp. 185-188.

Game Makers Toolkit, 2015. *What Capcom Didn't Tell You About Resident Evil 4 | Game Mechanics Explained - Youtube.* [Online]
Available at: https://www.youtube.com/watch?v=zFv6KAdQ5SE
[Accessed 03 February 2021].

GameAnalytics, 2021. *GameAnalytics | Trusted by 100K Game Developers.* [Online]
Available at: https://gameanalytics.com/
[Accessed 22 February 2021].

Haynes, W., 2013. Bonferroni Correction. In: O. Wolkenhauer, K. Cho & H. Yokota, eds. *Encyclopedia of Systems Biology.* New York, NY: Springer New York, p. 289.

Hick, D. et al., 2016. Using game analytics to evaluate puzzle design and level progression in a serious game. In: *Proceedings of the Sixth International Conference on Learning Analytics & Knowledge.* Edinburgh, United Kingdom: Association for Computing Machinery, p. 440–448.

Hunicke, R. & Chapman, V., 2004. AI for dynamic difficulty adjustment in games. *Challenges in game artificial intelligence AAAI workshop,* Volume 2.

IJsselsteijn, W., de Kort, Y. & Poels, K., 2013. *The Game Experience Questionnaire.* Eindhoven: Technische Universiteit Eindhoven.

Jennings-Teats, M., Smith, G. & Wardrip-Fruin, N., 2010. Polymorph: Dynamic difficulty adjustment through level generation. In: *Proceedings of the 2010 Workshop on Procedural Content Generation in Games.* Monterey, California: Association for Computing Machinery, p. 4.

Kambil, A., Ginsberg, A. & Boch, M., 1996. *Re-Inventing Value Propositions,* s.l.: Stern School of Business, New York University.

Karvande, H., 2020. *Medium.com Mario Kart Tour's Rubber-Banding is at Odds with its Monetisation | by Harshal Karvande | Game Design Post | Medium.* [Online]
Available at: https://medium.com/game-design-post/mario-kart-tours-rubber-banding-is-at-odds-with-its-monetisation-1f5b43909348
[Accessed 27 February 2021].

Kent, N., 2013. GEQ (Game Engagement/Experience Questionnaire): A Review of Two Papers. *Interacting with Computers,* Volume 25, pp. 278-283.

Koen, P. et al., 2001. Providing Clarity and a Common Language to the "Fuzzy Front End". *Research-Technology Management,* Volume 44, pp. 46-55.

Law, E. L.-C., Brühlmann, F. & Mekler, E. D., 2018. Systematic Review and Validation of the Game Experience Questionnaire (GEQ) – Implications for Citation and Reporting Practice. *CHI PLAY '18: Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play,* pp. 257-270.

Lazzaro, N., 2004. Why We Play Games:Four Keys to More Emotion Without Story. *Game Dev Conf.*

Left 4 Dead Wiki, 2021. *The Director | Left 4 Dead Wiki | Fandom.* [Online]
Available at: https://left4dead.fandom.com/wiki/The_Director
[Accessed 04 February 2021].

Lewis, J. R., 2018. The System Usability Scale: Past, Present, and Future. *International Journal of Human–Computer Interaction,* pp. 577-590.

Martins, A. I. et al., 2015. European Portuguese validation of the System Usability Scale (SUS). *Procedia Computer Science,* Volume 67, pp. 293-300.

Myers, M., 2017. *Either I'm Paranoid Or Mario Kart 8 Deluxe's AI Is Cheating.* [Online]
Available at: https://www.kotaku.com.au/2017/06/either-im-paranoid-or-mario-kart-8-deluxes-ai-is-cheating/
[Accessed 27 February 2021].

Osterwalder, A. & Pigneur, Y., 2010. *Business Model Generation: A Handbook for Visionaries, Game Changers, and Challengers.* 1 ed. s.l.:Wiley.

Palcic, I. & Lalic, B., 2009. Analytical Hierarchy Process as a tool for selecting and evaluating projects. *International Journal of Simulation Modelling,* 8(1), pp. 16-26.

Pinto, C. A. M. et al., 2018. *Fundamentos de Gestão.* 7 ed. Lisbon: Editorial Presença.

Porter, M. E., 1985. *Competitive Advantage.* New York: The Free Press.

Rietveld, A., Bakkes, S. & Roijers, D., 2014. Circuit-Adaptive Challenge Balancing in Racing Games. *Conference Proceedings - 2014 IEEE Games, Media, Entertainment Conference, IEEE GEM 2014.*

Saaty, T. L., 1980. *The Analytical Hierarchy Process.* New York: McGraw-Hill.

Sauro, J., 2011. *MeasuringU: Measuring Usability with the System Usability Scale (SUS).* [Online]
Available at: https://measuringu.com/sus/
[Accessed 05 February 2021].

Sepulveda, G. K., Besoain, F. & Barriga, N. A., 2019. *Exploring Dynamic Difficulty Adjustment in Videogames.* Valparaiso, Chile, s.n.

Spronck, P., Sprinkhuizen-Kuyper, I. & Postma, E., 2004. On-line Adaptation of Game Opponent AI with Dynamic Scripting.. *Int. J. Intell. Games & Simulation,* Volume 3, pp. 45-53.

The Editors of Encyclopaedia Britannica, 2020. *Student's t-test. Encyclopedia Britannica.* [Online]
Available at: https://www.britannica.com/science/Students-t-test
[Accessed 24 June 2021].

Valve Corporation, 2020. *Half-Life.* [Online]
Available at: https://www.half-life.com/en/halflife
[Accessed 22 February 2021].

Valve Developer Community, 2020. *Goldsource - Valve Developer Community.* [Online]
Available at: https://developer.valvesoftware.com/wiki/Goldsource
[Accessed 22 February 2021].

Valve Developer Community, 2021. *Half-Life: Source Bugs - Valve Developer Community.* [Online]

Available at: https://developer.valvesoftware.com/wiki/Half-Life:_Source_Bugs
[Accessed 22 February 2021].

Vargas, L. G., 1990. An overview of the analytical hierarchy process and its application. *European Journal of Operations Research,* Volume 48, pp. 2-8.

Vincke, S., Smith, A. & Perkins, C., 2020. *Baldur's Gate 3 Early Access Release Date Announcement - Panel From Hell* [Interview] (18 August 2020).

Xue, S. et al., 2017. Dynamic Difficulty Adjustment for Maximized Engagement in Digital Games. *WWW '17 Companion: Proceedings of the 26th International Conference on World Wide Web Companion,* p. 465–471.

Zohaib, M., 2018. Dynamic Difficulty Adjustment (DDA) in Computer Games: A Review. *Advances in Human-Computer Interaction,* Volume 2018, pp. 1-12.

Annex A. Filled Quantitative Evaluation Framework

q	D	Qi	Dimension	Qj	Wij (Factor Weight j in Dim i) [0,1]	Factor	rwjk (requirement weight k in Factor j) {2, 4, 6, 8, 10}	Requirement	wfk % requirement fulfilment k) [0,100]
85%	0.39	94.12	Technical	88.889	0.53	Gameplay	10	TG01 - Modify Player damage output	100
							10	TG02 - Modify Enemy damage output	100
							10	TG03 - Modify Enemy health	100
							10	TG04 - Modify Enemy accuracy	100
							10	TG05 - Modify Enemy speed	0
							10	TG06 - Modify Enemy placement	100
							10	TG07 - Modify Item placement	100
							10	TG08 - Modify Health Item effectiveness	100
							10	TG09 - Determine DDA actions to take	100
				100	0.47	Analytics	8	TA01 - Track player weapon usage	100
							8	TA02 - Track player ammunition usage	100
							8	TA03 - Track player accuracy	100
							8	TA04 - Track player kills	100
							8	TA05 - Track player deaths	100
							8	TA06 - Track player health	100
							8	TA07 - Track player time spent on level	100
							8	TA08 - Show current statistics in console	100
		80	Ergonomic	66.667	0.60	Usability	10	EU01 - The DDA implementation does not incur a drop in usability	100
							10	EU02 - Modifications are seamless	0
							10	EU03 - The game does not present untreated error messages to the user	100

				100	0.40	Gameplay	10	EG01 - The game is challenging	100
							10	EG02 - Game correctly assesses player skill level	100
		66.67	Pedagogic	66.667	1.00	Affective	10	PA01 - Players show signs of competence in the game	0
							10	PA02 - Players show signs of immersion in the game	100
							10	PA03 - Players show signs of positive emotion	100

Annex B. Player Profile Responses

Age:	Sex:	How many hours, on average, do you spend playing video games per week?	Given the option, which difficulty setting do you normally select?	Have you ever played Half-Life before?
22	Male	32+	Medium	Yes
22	Male	32+	Medium	No
22	Male	9 to 16	Medium	Yes
22	Male	16 to 24	Harder than medium	Yes
20	Male	9 to 16	Medium	No
23	Male	32+	Medium	Yes
22	Male	32+	Harder than medium	No
18	Male	9 to 16	Harder than medium	Yes
22	Male	1 to 8	Harder than medium	No
19	Male	1 to 8	Easier than medium	No
23	Male	32+	Harder than medium	Yes
23	Male	32+	Medium	Yes
23	Male	32+	Harder than medium	No
23	Male	32+	Medium	No
18	Male	1 to 8	Harder than medium	No

Annex C. Responses to the DDA session

I felt content	4	3	3	2	3	4	4	2	3	1	3	4	3	2	3
I felt skilful	4	3	4	3	2	3	4	1	0	1	3	4	2	2	3
I was interested in the game's story	2	1	1	2	3	2	1	3	1	3	1	2	2	1	3
I thought it was fun	4	3	3	4	4	3	4	3	2	1	4	4	4	2	3
I was fully occupied with the game	4	3	4	4	4	4	4	3	2	1	4	4	3	3	4
I felt happy	4	2	4	4	2	3	4	2	0	1	4	3	3	3	2
It gave me a bad mood	0	1	0	0	2	0	0	1	1	1	0	0	0	1	0
I thought about other things	4	1	0	1	0	1	0	0	0	1	1	0	1	0	0
I found it tiresome	0	2	2	1	3	0	0	4	0	4	0	0	0	3	1
I felt competent	4	3	3	3	2	3	4	1	1	0	3	4	2	2	2
I thought it was hard	2	2	1	3	3	2	3	3	3	2	2	2	3	2	0
It was aesthetically pleasing	2	2	2	4	0	2	1	2	0	0	1	2	3	0	2
I forgot everything around me	4	1	2	3	3	3	0	3	1	0	2	3	2	3	3
I felt good	4	3	3	2	1	3	4	2	0	0	3	4	3	3	2
I was good at it	4	3	4	2	2	4	4	0	0	0	4	4	2	2	3
I felt bored	3	2	0	1	1	1	0	0	0	4	0	0	0	2	1
I felt successful	4	4	3	3	1	3	4	3	1	0	4	4	2	3	2
I felt imaginative	4	2	1	2	0	1	1	0	0	0	3	2	1	3	2
I felt that I could explore things	4	2	0	3	4	2	2	3	2	0	4	4	3	3	3
I enjoyed it	4	3	3	4	2	3	4	4	2	0	4	4	3	2	3
I was fast at reaching the game's targets	4	2	3	3	3	3	3	3	1	0	3	4	2	3	3
I felt annoyed	3	2	1	1	1	0	0	3	1	4	0	0	0	2	1
I felt pressured	3	2	1	1	2	0	2	0	0	0	1	0	0	3	2
I felt irritable	0	1	1	2	1	0	0	2	0	0	0	0	0	2	0
I lost track of time	4	3	0	1	4	4	1	3	1	4	3	3	2	4	2

I felt challenged	3	2	2	3	3	3	4	4	3	0	3	3	3	3	1
I found it impressive	3	3	2	3	0	3	3	3	2	0	2	2	2	2	1
I was deeply concentrated in the game	4	2	3	4	2	3	4	4	1	0	3	4	1	4	3
I felt frustrated	0	2	2	2	3	0	0	2	1	0	1	1	0	2	0
It felt like a rich experience	3	3	3	3	1	3	2	1	2	1	2	3	1	2	3
I lost connection with the outside world	4	2	1	3	2	3	0	1	1	0	2	3	1	3	2
I felt time pressure	3	2	1	1	2	0	4	0	0	0	2	0	0	1	1
I had to put a lot of effort into it	3	2	2	2	1	2	4	4	2	0	2	2	2	2	1
I felt revived	2	2	0	1	1	2	4	0	0	0	2	3	0	2	0
I felt bad	0	2	0	1	0	0	0	0	1	0	0	0	0	1	1
I found it hard to get back to reality	1	1	0	0	0	0	0	0	0	0	0	1	0	1	0
I felt guilty	2	1	0	0	0	0	0	0	0	0	0	0	0	1	0
It felt like a victory	4	3	0	3	1	2	4	4	1	0	3	4	2	3	1
I found it a waste of time	0	1	0	0	1	0	0	0	0	2	1	0	0	2	0
I felt energised	3	3	0	2	1	1	3	4	1	0	2	2	0	3	0
I felt satisfied	4	3	2	3	2	3	4	4	1	0	3	3	3	3	1
I felt disoriented	2	1	0	0	3	0	0	1	2	4	0	0	0	1	0
I felt exhausted	0	2	0	1	0	0	0	2	1	4	0	0	0	1	0
I felt that I could have done more useful things	0	1	3	3	0	2	1	3	0	0	1	1	2	3	0
I felt powerful	4	3	2	3	2	3	4	3	0	0	2	4	1	2	1
I felt weary	0	2	1	0	1	0	0	3	1	0	0	0	0	2	0
I felt regret	1	2	0	0	1	0	0	0	0	0	0	0	0	1	0
I felt ashamed	0	2	0	0	0	0	0	0	1	0	0	0	0	0	0
I felt proud	4	3	1	2	1	2	4	3	1	0	2	3	1	3	0
I had a sense that I had returned from a journey	3	2	2	2	2	1	2	2	1	0	1	2	1	3	0

I think that I would like to use this system frequently.	5	4	4	4	2	5	2	4	4	1	3	5	3	1	3
I found the system unnecessarily complex.	1	2	2	3	2	2	1	1	2	1	3	1	1	2	1
I thought the system was easy to use.	5	4	4	5	4	5	5	3	5	3	4	5	4	4	4
I think that I would need the support of a technical person to be able to use this system.	2	2	1	5	3	1	1	1	1	1	1	1	2	1	1
I found the various functions in this system were well integrated.	5	4	5	4	4	4	5	4	4	3	4	4	3	4	2
I thought there was too much inconsistency in this system.	1	1	1	2	1	2	1	1	2	3	2	1	1	2	2
I would imagine that most people would learn to use this system very quickly.	4	4	4	5	5	5	5	5	5	3	4	5	4	4	4
I found the system very cumbersome to use.	1	2	1	2	1	1	1	1	1	3	2	1	1	2	2
I felt very confident using the system.	5	4	4	4	2	4	5	3	5	3	5	5	4	4	4
I needed to learn a lot of things before I could get going with this system.	1	1	2	1	3	1	2	1	1	1	1	1	2	2	1

Annex D. Responses to the unaltered session

I felt content	2	3	3	3	3	2	4	2	3	1	3	2	3	3	2
I felt skilful	4	4	3	4	4	2	3	4	3	0	3	1	2	3	2
I was interested in the game's story	1	1	2	2	3	1	0	2	3	4	1	2	2	1	2
I thought it was fun	3	2	4	3	4	2	3	3	3	2	3	1	3	3	3
I was fully occupied with the game	3	3	4	4	4	1	3	3	3	4	3	1	4	3	3
I felt happy	3	2	3	3	4	2	4	3	3	1	3	2	3	3	1
It gave me a bad mood	3	1	0	0	0	1	0	0	0	1	0	0	0	1	0
I thought about other things	3	1	0	1	1	3	0	0	1	2	0	3	1	1	0
I found it tiresome	3	3	0	1	0	2	0	1	1	4	1	0	1	1	0
I felt competent	4	3	3	3	3	2	3	0	3	2	3	2	3	3	3
I thought it was hard	1	2	2	1	2	0	1	0	0	2	2	0	2	1	0
It was aesthetically pleasing	2	3	3	3	0	2	1	2	1	0	1	2	3	1	2
I forgot everything around me	3	1	2	0	3	1	0	3	0	2	1	1	2	3	1
I felt good	2	3	3	3	4	2	3	3	3	0	3	2	3	3	2
I was good at it	4	2	3	3	3	4	3	4	3	0	3	4	3	3	3
I felt bored	3	1	0	1	1	3	0	1	0	2	0	3	0	1	0
I felt successful	4	3	3	3	4	3	3	4	3	2	3	1	3	3	3
I felt imaginative	3	2	1	1	3	1	1	2	3	0	1	2	1	2	2
I felt that I could explore things	3	3	4	2	4	2	3	2	3	0	3	3	2	3	2
I enjoyed it	3	3	4	3	4	2	3	3	3	1	4	2	3	3	2
I was fast at reaching the game's targets	4	2	3	3	4	4	3	4	3	1	4	4	2	4	4
I felt annoyed	2	1	0	2	0	2	0	0	0	2	1	0	1	1	0
I felt pressured	3	1	0	0	0	0	0	0	0	0	1	0	0	1	0
I felt irritable	2	1	0	1	0	0	0	0	0	0	0	0	0	1	0
I lost track of time	2	2	3	1	3	0	1	1	3	4	1	0	2	3	2

I felt challenged	2	4	3	1	3	1	1	0	2	0	2	1	1	2	0
I found it impressive	2	3	3	2	1	2	1	2	0	0	2	2	1	3	0
I was deeply concentrated in the game	3	2	4	3	3	2	3	1	1	0	3	1	2	3	3
I felt frustrated	1	2	0	1	1	0	0	0	0	0	2	0	0	1	0
It felt like a rich experience	2	3	3	2	2	1	3	2	1	1	3	2	1	2	2
I lost connection with the outside world	3	2	2	0	2	0	1	0	0	0	1	0	2	3	3
I felt time pressure	2	4	0	0	0	0	2	0	0	0	2	0	1	1	0
I had to put a lot of effort into it	1	3	2	1	1	0	1	1	0	1	2	0	1	1	0
I felt revived	2	2	1	0	1	1	0	0	0	0	1	1	0	2	1
I felt bad	1	2	0	0	0	0	0	0	0	2	0	0	0	1	0
I found it hard to get back to reality	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0
I felt guilty	1	2	0	0	0	0	0	0	0	0	0	0	0	0	1
It felt like a victory	4	3	3	1	3	1	1	3	2	0	2	2	1	3	1
I found it a waste of time	1	2	0	1	0	2	0	0	0	0	1	1	0	1	0
I felt energised	2	1	2	0	4	1	0	3	0	0	2	0	0	3	0
I felt satisfied	3	2	3	2	3	2	3	3	2	0	3	1	3	3	1
I felt disoriented	2	1	0	0	0	0	0	0	0	4	0	0	0	1	0
I felt exhausted	2	2	0	0	0	0	0	0	0	2	0	0	0	1	0
I felt that I could have done more useful things	3	3	3	2	0	3	0	2	1	2	0	2	0	3	1
I felt powerful	4	2	3	3	4	1	1	1	3	1	2	1	0	3	1
I felt weary	2	1	0	1	0	0	0	1	0	1	0	0	0	1	0
I felt regret	1	1	0	0	0	0	0	0	0	2	0	0	0	1	0
I felt ashamed	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
I felt proud	4	3	3	1	3	1	1	2	1	0	2	1	0	2	2
I had a sense that I had returned from a journey	2	2	2	2	4	1	0	2	1	1	0	2	1	3	1

I think that I would like to use this system frequently.	3	3	5	3	4	3	4	3	4	5	3	5	3	1	4
I found the system unnecessarily complex.	1	3	2	1	1	1	1	1	1	3	2	1	1	2	1
I thought the system was easy to use.	4	5	5	4	5	5	3	5	5	4	4	5	4	4	5
I think that I would need the support of a technical person to be able to use this system.	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1
I found the various functions in this system were well integrated.	4	4	5	4	4	4	3	4	4	4	3	4	3	3	3
I thought there was too much inconsistency in this system.	1	1	1	1	1	1	2	1	1	1	4	1	2	2	1
I would imagine that most people would learn to use this system very quickly.	4	4	5	4	4	4	5	5	5	5	3	5	4	4	4
I found the system very cumbersome to use.	1	2	1	2	1	1	1	1	1	3	1	1	2	2	2
I felt very confident using the system.	5	4	5	4	4	4	5	5	5	4	4	5	5	4	4
I needed to learn a lot of things before I could get going with this system.	1	2	1	2	1	1	1	1	1	1	2	1	2	2	1

Annex E. Sample Survey

Dynamic Difficulty Adjustment in First-Person Shooters

Video game developers have found that different players, with different skill levels, require different challenges to fully enjoy their gaming experiences.

Traditionally video games allow the player to select a difficulty level before starting the game, however, this method has the problem of being too restricting and limited, as many players have their comfort zone between the levels created by the developers.

In this project, we analyse how Dynamic Difficulty Adjustment (DDA) can be used to improve the experience of the players during their play sessions and so increase the value a game offers them.

*Obrigatório

1. Age: *

2. Sex: *

Marcar apenas uma oval.

- ◯ Male
- ◯ Female

3. How many hours, on average, do you spend playing video-games per week? *

Marcar apenas uma oval.

- ◯ None
- ◯ 1 to 8
- ◯ 9 to 16
- ◯ 16 to 24
- ◯ 24 to 32
- ◯ 32+

4. Given the option, which difficulty setting do you normally select? *

Marcar apenas uma oval.

- ◯ Easier than medium
- ◯ Medium
- ◯ Harder than medium
- ◯ I don't play video games

5. Have you ever played Half-Life before?

Marcar apenas uma oval.

- ◯ Yes
- ◯ No

Game Experience Questionnaire – Core Module

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

0: not at all
1: slightly
2: moderately
3: fairly
4: extremely

6. I felt content *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

7. I felt skilful *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

8. I was interested in the game's story *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

9. I thought it was fun *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

10. I was fully occupied with the game *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

11. I felt happy *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

12. It gave me a bad mood *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

13. I thought about other things *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

14. I found it tiresome *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

15. I felt competent *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

16. I thought it was hard *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

17. It was aesthetically pleasing *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

18. I forgot everything around me *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

19. I felt good *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

20. I was good at it *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

21. I felt bored *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

22. I felt successful *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

23. I felt imaginative *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

24. I felt that I could explore things *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

25. I enjoyed it *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

26. I was fast at reaching the game's targets *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

27. I felt annoyed *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

28. I felt pressured *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

29. I felt irritable *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

30. I lost track of time *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

31. I felt challenged *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

32. I found it impressive *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

33. I was deeply concentrated in the game *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

34. I felt frustrated *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

35. It felt like a rich experience *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

36. I lost connection with the outside world *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

37. I felt time pressure *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

38. I had to put a lot of effort into it *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

GEQ – post-game module

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

0: not at all
1: slightly
2: moderately
3: fairly
4: extremely

39. I felt revived *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

40. I felt bad *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

41. I found it hard to get back to reality *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

42. I felt guilty *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

43. It felt like a victory *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

44. I found it a waste of time *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

45. I felt energised *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

46. I felt satisfied *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

47. I felt disoriented *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

48. I felt exhausted *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

49. I felt that I could have done more useful things *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

50. I felt powerful *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

51. I felt weary *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

52. I felt regret *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

53. I felt ashamed *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

54. I felt proud *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

55. I had a sense that I had returned from a journey *

Marcar apenas uma oval.

0	1	2	3	4
◯	◯	◯	◯	◯

System Usability Scale

56. I think that I would like to use this system frequently. *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

57. I found the system unnecessarily complex. *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree						Strongly agree

58. I thought the system was easy to use. *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree						Strongly agree

59. I think that I would need the support of a technical person to be able to use this system. *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree						Strongly agree

60. I found the various functions in this system were well integrated. *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree						Strongly agree

61. I thought there was too much inconsistency in this system. *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

62. I would imagine that most people would learn to use this system very quickly. *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

63. I found the system very cumbersome to use. *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

64. I felt very confident using the system. *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

65. I needed to learn a lot of things before I could get going with this system. *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree