TÉCNICO LISBOA

Adaptive Single Player Challenge Generation for World War Online

Torstein Lundervold Nesheim

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Carlos António Roque Martinho

Examination Committee

Chairperson: Prof. João António Madeiras Pereira
Supervisor: Prof. Carlos António Roque Martinho
Member of the Committee: Prof. Pedro Alexandre Simões dos Santos

November 2021

Acknowledgments

I would like to thank my supervisor, advisor, technical support, guide and library, Professor Dr. Carlos António Roque Martinho for his teachings, support, great advice, encouragement, initiative and engagement throughout this Masters. I would not have gotten this far without him.

Furthermore I would like to thank my parents, my sister and the rest of my family for all their support, guidance, warm gestures and caring throughout both challenging and good times in my life.

I would also like to formally thank my partner, fiancée and best friend, Matilde Barrote Silva, for all her love and support, for all the moments shared and conversations spoken, there is no greater motivation to go through the day than you.

A huge thank you to my bork boys Ole Kristian Halstensen & Magnus Hagesæther Jørs for all the great times we share and for lifelong friendships.

To all the friends and colleagues I have met and spent time with during my Bachelor's degree at INSA Toulouse and these last 2 years at Técnico Lisboa: without you, I can't see myself completing this tough education. The quotidian is truly one of most important factors to a good quality of life, and that has consisted largely of you - Thank you for sharing it with me.

Finally, to everyone else in my life who have supported or inspired me - Thank You.

Abstract

In this work we seek to improve the game experience for new players in free-to-play games by using evolutionary computing and procedural content generation. The hypothesis is based on the positive psychology principle of Flow, in that it focuses on generating challenges that are adequate for the player at their experience level within the game in order to improve their player experience. This was implemented and tested in collaboration with Chilltime, through the “Battle Arena” gamemode within their free-to-play browser-based game “World War Online” (WWO).

The testing was done outside the actual web-based game, directly in the unity application of Battle Arena, with offline generation of challenges due to long generation times. We use the repeated measures design and the Intrinsic Motivation Inventory (IMI) to evaluate our model, but could unfortunately only do testing with 13 participants due to changes in Chilltime and their servers midway in the testing.

The preliminary results we could reach with this sample size indicate that the current model does not increase player enjoyment or improve the experience for new players of WWO, but these results might be a matter of adjusting the difficulty parameter of the evolutionary component of our model. The difference in results between the two versions tested also was not very large, and besides showing that this initial model was close to fulfilling its purpose, it also proves its capabilities of creating interesting challenges, as the main issue found was that the challenges generated were too difficult.

We believe this work can be expanded upon and that interesting results can be obtained through parameter tweaking and more rounds of testing.

Keywords

Procedural Content Generation - Genetic Algorithms - Video Games - Flow

Contents

List of Figures

List of Algorithms

1

Introduction

Contents

In this work we aim to see if we can improve player experience in a free-to-play browser based game such as WWO, by adapting challenges to player skill through the use of Search-based PCG and Genetic Algorithms, a hypothesis that is based on the psychological principle of flow. Despite basing our work on WWO, we hope that the results we found are transferable to other similar games.

1.1 Motivation

Video games are undeniably extremely popular right now, and they are likely to continue to be so in the future. The reasons for their popularity are many; from social aspects such as providing meeting points[1], fun group activities[2] and encouraging strategic cooperation and communication[3] to health aspects such as mitigating PTSD [1], physical rehabilitation through VR [2] and improving mental health by simply providing a fun and easy way to take a break from the stressful reality we live in today, in which more than one in three people experience a lot of worry or stress [3].

In relation to the ongoing COVID-19 pandemic, research shows that during march 2020, 45% of video gamers said that they spent more time playing video games during quarantine than before [4], which emphasizes the importance of video games as a means of entertainment in times of great social distress.

Improving video games is thus undoubtedly an important area, and one way to describe a good player experience is through the psychological principle of "Flow" [5]. Flow is a mental state achieved through an "optimal experience", and it depends on two main components: Skill and Challenge in relation to each other (see 2.1). It presents a direct correlation between player enjoyment and the matching of difficulty with skill, in which a challenge adapted to the skill of the player results in a better player experience. This is the area of video game improvement that we will discuss in this dissertation - Striving to create the optimal experience in a video game by adapting challenges to the players skill.

Video games are all very different which makes it near impossible to do a work that is completely generic within them. In this work we have looked at the case of *World War Online*[4] (WWO), but we hope and believe that the work can transfer to other games as well.

1.2 Problem

According to Marc Robinson's 2013 GDC talk, "On average, less than 40% of players return to a free-to-play game after just one session." [6] From an economic perspective, this statistic could be simplified

[1] Club Penguin, Disney Interactive Studios, 2005; VR Chat, VRChat Inc., 2014
[2] The Jackbox Party Pack series, Jackbox Games, 2014; Super Mario Party, Nintendo, 2018
[3] League of Legends, Riot Games, 2009; Counter-Strike: Global Offensive, Valve, 2012
[4] World War Online, Chilltime, 2010

to say that Game Publishers of free-to-play games are “losing more than 60% of their potential income” from the very first interaction between a player and their game.

So how do we provide challenges that are adequate for the player at any time in their player lifetime, including at the very beginning - in order to mitigate this huge drop in retention rate at the start of a gaming experience? How do we keep the player progression going long enough for the player to understand the game?

One reason for the retention losses at the beginning of a game might be that it is either too difficult - leaving the player anxious, or it might be too easy - leaving the player feeling bored. Either way causing the player to have a sub-optimal experience which might dissuade them from playing any more.

In WWO, a large part of this problem is directly related to the transition from tutorial/campaign to multiplayer battles. The current campaign consists of static, sequential challenges, which presents all players with the same progression curve, even though each player will have a different skill curve. This will, for most players, lead to inconsistent matching between skill and difficulty, which may cause the player to quit playing due to lack of enjoyment.

It is this problem of continuously matching player skill with game difficulty in Battle Arena that we seek to solve in this work. We want that when a new player chooses to leave the current campaign in WWO, and starts to play the new game mode Battle Arena, they will immediately be met with a challenge of suitable difficulty, leading to greater player enjoyment.

1.3 Hypothesis

We hypothesize, in line with the concept of Flow, that constantly providing the players with challenges that are adapted to their skill level will improve their game experience and thus lead to an improved game experience for new players in the case of WWO.

We test this by attempting to use search-based procedural content generation (see 2.2.1) and Genetic algorithms (see 2.3) to help us create meaningful challenges for the players by using choices they make in the tutorial to inform a genetic algorithm of what skill level they are at.

Components of this hypothesis are supported by previous works : when exploring the effects of experience-driven challenge generation in Holiday Knight, Pardal & Martinho [7] concluded that it led to an improved player enjoyment, and similar results where reported by Mestre & Martinho when exploring experience-driven PCG combined with a single player Elo system in “Go Go Hexahedron” [8]. Our search-based PCG is similar to the experience-driven PCG used in these works in that they all aim to adapt challenge difficulty to the player skill, and as a consequence we believe that we will achieve similar results related to the player enjoyment.

We evaluate our hypothesis by having players report on various aspects of their experience after

having played through challenges generated for them. For comparison we also ask participants to report on the same aspects after playing through a control set of challenges, static for all participants. We compare the two versions by how appropriate the player rates the challenges in each and how their experience was while playing them. We expected to see that players report meeting a more appropriate difficulty, and having a better experience when playing through the challenges generated for them.

1.4 Contributions

In this work we contributed with the following:

- A literary review within Procedural Content Generation and Evolutionary Algorithms.
- A computational model combining PCG with genetic algorithms.
- The implementation of said computational model within the commercial game “WWO”.
- An evaluation of the computational model with human players.

1.5 Organization of the Document

This thesis is organized as follows: Chapter 1 introduces the theme, problem, and hypothesis for our work. In chapter 2 we move on to discuss the previous works that we are building our work on top of - what they concluded and how it is relevant to us. We will discuss the psychological principle of flow, procedural content generation, genetic algorithms and the Intrinsic Motivation Inventory (IMI) questionnaire. In chapter 3 we will give an overview of WWO, as well as describe the game mode we developed and used in relation to this work “Battle Arena”. We then proceed to describe how we implemented the genetic algorithm and integrated it with World War Online in chapter 4, before we detail the evaluation methods and experimental procedure in chapter 5 and finally discuss the results leading from the evaluation in chapter 6 and attempt to draw some conclusions based on these in chapter 7.

2

Related Work

Contents

The main focus of our work is to generate, and evaluate the difficulty of, challenges (Defending Unit Setups) in the Battle Arena gamemode of WWO. In order to best achieve this goal, we explore techniques such as procedural content generation, evolutionary algorithms and evaluation techniques such as the Intrinsic Motivation Inventory (**IMI**) and A/B testing. We will also briefly go over the psychological principle of flow which builds the foundation of our hypothesis.

2.1 Flow

In Positive Psychology, Flow is the state of being completely and fully immersed in an activity. It is termed as an "optimal experience" by Csikszentmihalyi, which is highly individual as it *"...depends on the ability to control what happens in consciousness moment by moment"*. [5]

However, in Csikszentmihalyi's studies, he found that what makes an experience enjoyable for people is fairly invariant across profession, social class, culture, age and gender - it was usually described in the same way by all, which led to the list of the 8 major components in flow.

1. We confront tasks we have a chance of completing;
2. We must be able to concentrate on what we are doing;
3. The task has clear goals;
4. The task provides immediate feedback;
5. One acts with deep, but effortless involvement, that removes from awareness the worries and frustrations of everyday life;
6. One exercises a sense of control over their actions;
7. Concern for the self disappears, yet, paradoxically the sense of self emerges stronger after the flow experience is over;
8. The sense of duration of time is altered.

One of the key points of flow is that it is a state of inner equilibrium between "anxiety" and "boredom", where your skill is matched by a suitable challenge. This is visualized in Figure 2.1, which illustrates the flow state in relation to skill, difficulty (challenge), anxiety and boredom. This figure serves to illustrate that there must be a continuous match between skill and difficulty for someone to remain in the state of flow. This becomes especially apparent in the case of video games.

When a player in a video game faces a challenge that is too hard for them, they will probably feel frustration or anger - feelings that are closer to anxiety - whereas if the player easily succeeds in completing all the challenges presented to them, they will likely feel bored and lose interest in the game. In

order to keep the player in an optimal state of play it is therefore key to constantly provide them with challenges that match their skill level.

Flow is therefore an important concept in our work because our end goal is to deliver an enjoyable user experience to the players of “Battle Arena” in WWO. We will address the first component of entering flow through our adaptive challenge system, where we will seek to constantly match the players’ skill with appropriate challenges, in order to improve their player experience.

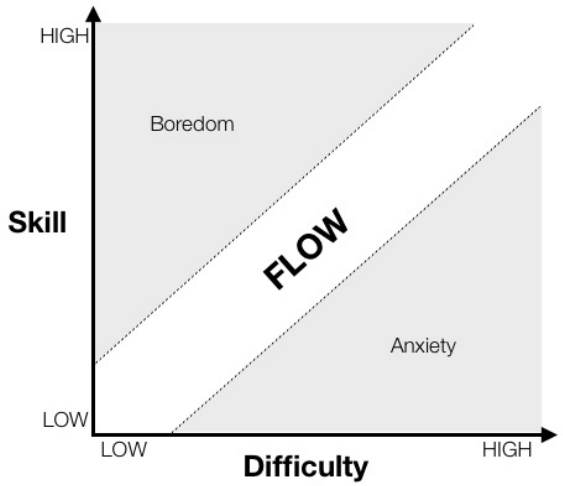


Figure 2.1: The mental state of Flow visualized - Adapted from Csikszentmihalyi, 1990.

2.2 Procedural Content Generation

Procedural Content Generation (PCG) refers to the algorithmic, automatic generation of game content as opposed to the traditional method of manual creation. In this section, game content will refer to all aspects within a game that impact gameplay, with the exceptions of Non-Player Character (NPC) behavior and the game engine itself.

Traditionally, large AAA games have had huge teams working over long periods of time in order to fill their large game worlds with content for the players to explore; for example Rockstar’s “Grand Theft Auto IV” is the work of 1000 people over a period of three years. This “bottleneck” of content creation in video games is one of the reasons why Procedural Content Generation is so popular in the area of Video Game Development. In fact, some researchers estimate content creation to be around 30-40% of the budget in the production of AAA games [9]. It is therefore safe to assume that through the use of Procedural Content Generation alleviating some of the effort previously put in through man-hours, both the costs and the risks associated with video game development can go down substantially.

Within the context of the Battle Arena in WWO, this content generation translates to generating

challenges for the players. By using procedural content generation, we can generate large amounts of challenges without having to spend time designing and testing them individually. This also allows for each challenge to be tailored for a much more specific skill level than if we had to design all the challenges manually, which will make it easier to create an optimal state of play for the players according to the principle of Flow.

In order to properly categorize our work within the space of PCG, we will need some terminology. Here we are referencing the terminology devised by Togelius et al [10].

- **Online versus offline**

  The first distinction we can make between PCG algorithms concerns the time at which the generation takes place. *Online* generation happens during gameplay, while offline generation happens before a game is loaded / played. An example of online PCG is the world around the player in *Minecraft*[1], and an example of offline PCG is when a level designer uses PCG to aid in the level design process before releasing the game.

- **Necessary versus optional**

  Another important difference comes when we discuss the importance of the content generated by a certain algorithm. Necessary PCG creates content that is essential for the game to function as intended, and for the player to be able to progress - like the platforms in Super Mario[2] would be if they were procedurally generated. Optional PCG creates additional content that can add to the player experience but is not essential, like weapons that the player can choose to use or not. Necessary content has to be "correct" while optional content has more leeway for errors.

- **Stochastic versus Deterministic**

  Another dimension to the qualities of PCG concerns the predictability of the algorithm. This distinction lies within the variety of outcomes the algorithm can generate from a single set of parameters (not counting the seed for random number generation). If the algorithm can generate several variants of an output from a single set of parameters, it is named stochastic, whereas if it will always generate the same output from a set of parameters, it is deterministic. Deterministic algorithms can in this way be seen as a form of compression technique.

- **Constructive versus generate-and-test**

  A final distinction between these algorithms can be made in the methodology of their content production. An algorithm that only runs once and delivers the output without verifying it is called *constructive*. These algorithms follow strict set of rules that guarantee their output to always be reliable. On the other hand, we have algorithms which will iteratively converge on an acceptable

[1] Minecraft, Windows PC version, Mojang Studios, 2011
[2] Super Mario Bros, NES version, Nintendo, 1985

output by going through cycles of generation and testing. The tests are what guarantees the reliability of the output, and the content will not be delivered until the test is successful. These algorithms can be described as *generate-and-test* algorithms, and one example of how these can be used is in the dungeon generation of rogue-like games, where part of the test can be to check how many reachable rooms there are in the generated dungeon.

One of the use cases for Procedural Content Generation in video games is level generation. The most common examples of procedural generated levels are typically found in rogue-like games like *Dead Cells*[3], *Hades*[4] and *Slay the Spire*[5]. The premise of rogue-like games is that the player "starts over" every time they die ("permadeath"), thus, procedural level generation is used to provide a different experience each time the player goes through the game and is one of the main components to the replay value of rogue-like games.

There are, however, several other advantages to using PCG in a game, and one such is to directly improve the player experience by taking the player themselves into account as a parameter to the generation algorithm. Using player skill as an input to the procedural generation of levels has been shown to improve the player experience [11]. This is in accordance with Cziksentmihalyi's work mentioned in the previous section (2.1).

2.2.1 Search-based PCG

In our approach we employ the use of a specific case of PCG called Search-based PCG in order to generate challenges for Battle Arena. We will therefore briefly go over what search-based PCG entails, in this subsection. Search-based PCG is a special case of the **generate-and-test** approach to PCG for which Togelius et al. [12] defined the following qualifications:

- The test function does not simply accept or reject the candidate content, but grades it using one *or a vector of* real numbers. Such a test function is often referred to as a fitness function or utility function. In this work we will refer to this as the *fitness function* as in the terminology from evolutionary algorithms (see 2.3), and we will refer to its result as the *fitness* of the content.
- The generation of new content depends on the fitness of the previously generated content. In this way, we will gradually produce new content that has a higher value than the previous ones.

In many of the cases of search-based PCG, the search algorithm is of an evolutionary nature in line with the criteria listed above. This is because evolution through natural selection of "randomly" chosen individuals can be thought of as a search through the space of possible chromosome values [13] and

[3] Dead Cells. Windows PC version, Motion Twin, 2018
[4] Hades. Windows PC Version, Supergiant Games, 2020
[5] Slay the Spire. Windows PC Version, Mega Crit Games, 2020

this will also be the case for our work. Another aspect of having an evolutionary search algorithm is that because of its inherent randomness combined with its converging properties, a very large portion of the search-space is covered naturally, and interesting, unforeseen solutions will often be found.

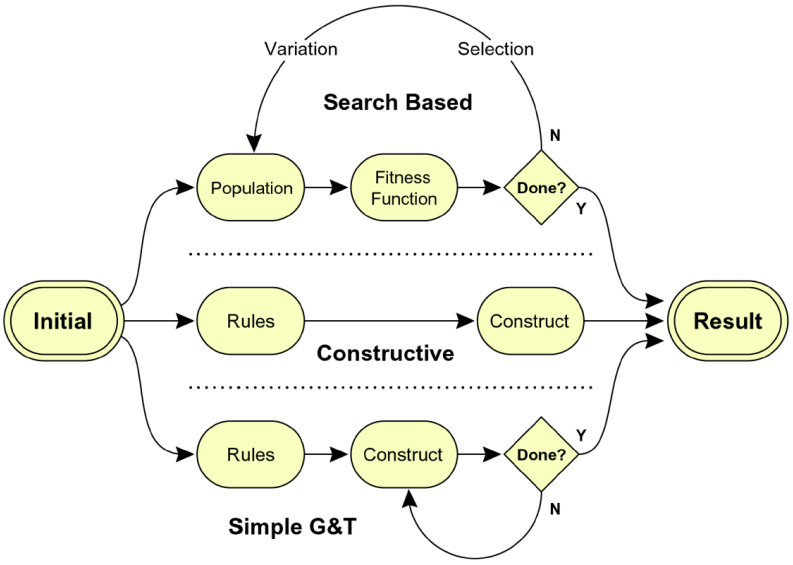


Figure 2.2: The differences between 3 approaches to PCG: *Simple Generate & Test, Constructive* and *Search-Based*, Adapted from Togelius et al., 2011

2.2.2 Classification of Our Approach

Having established a taxonomy of PCG, we can classify our approach within this space as ideally being **online** because the generation should in a permanent solution take place as the player enters Battle Arena, while in this work it is actually **offline**, as it happens between to sessions of play which will be expanded on in chapter 5. Furthermore, our approach is **necessary** because it is the main content the player interacts with and any errors in it will be detrimental to the players experience. Because we plan to use search-based PCG, we can also classify it as **stochastic**, a property imposed by the evolutionary search algorithms we will discuss in the next section.

2.3 Evolutionary Algorithms: Genetic Algorithms

Evolutionary algorithms (EA's) is a branch of algorithms within the field of studies called "Artificial Life" in which scientists explore natural systems to draw inspiration for artificial "copies" of natural phenomena to further our technological advancement. These algorithms are based on the theory of evolution by Charles Darwin, in which species adapt to their surrounding slowly through mutations and the famously

coined term “survival of the fittest”. This coincides well with many of the goals of training AI, in that this process, although “random” in its nature, converges globally towards a solution to a problem over time.

Evolutionary algorithms use a *population* of individuals, where an individual is referred to as a *chromosome*. A chromosome defines the parameters/characteristics of individuals in the population. Each characteristic is referred to as a *gene*. The value of each gene is called an *allele*. In each generation, each individual competes against each other to reproduce themselves. The individuals with the best survival capabilities have the best chance to reproduce and pass on their genes to the next generation. The next generation is generated by combining parts of the parents; this process is referred to as a *crossover*. Each individual in the population can also undergo *mutation* which slightly changes some of the allele in the chromosome. The survival strength of an individual is measured using a *fitness function* which reflects the objectives and constraints of the problem to be solved. After each generation, the strongest part of the population will reproduce and survive to the next generation [13].

Within evolutionary algorithms, Genetic Algorithms are some of the most commonly used in relation to PCG and the training of AI in games [14]. Several variations of GA’s have been developed, and we will discuss several of the different types of selection operators, mutation operators and crossover operators.

The process of a genetic algorithm is similar to the generic EA’s and can be listed as the following steps:

1 Initialize the population

2 Measure the fitness of each individual by the fitness function

3 **Selection**

4 **Crossover**

5 **Mutation**

Steps 2-5 are then repeated until a satisfactory condition is met, typically that the highest fitness of the population is above a certain threshold, a maximum iteration count has been reached or until the population is uniform.

2.3.1 Selection

The selection operator is one of the main operators of genetic algorithms. It is essentially what ensures the “progression” of the population in the sense that the strong reproduce and the weak change or die, directly analogous to the Darwinian principle of survival of the fittest as mentioned earlier. This operator comes into play in two steps of a GA; the selection of the parents/origins for the next generation, and in the reproduction step, in which the selection operator chooses which of the parents reproduce by means of crossover and which of them undergo mutation.

A selection operator is mainly characterized by its *Selective Pressure*. The selective pressure is defined as the speed at which the best solution will occupy the entire population by repeated application of the selection operator alone [15]. A high selective pressure indicates a faster decrease in diversity compared to that of a low selective pressure, and as such, a high selective pressure may lead to the algorithm converging prematurely on suboptimal solutions.

As discussed by Goldberg and Deb [16], the 4 more commonly used selection schemes are:

- Proportionate Reproduction (or "Roulette Wheel") - The probability for selection of a given individual is equal to its normalized fitness value.
- Ranking Selection - Each individual is given a rank based on its fitness value, where the highest fitness value gives the highest rank. The probability for selection is then the normalized rank of the individual.
- Tournament Selection - Several groups of n randomly selected individuals are put to "compete" against each other, meaning that the individual with the highest fitness value of each group is selected.
- Steady State Selection - More a design choice than an operator in itself, steady state selection is when a part of the generation survives unchanged to the next generation. An example of this would be when the strongest half of a population survives, while their offspring replaces the weaker half.

Proportionate reproduction is shown to be significantly slower than the other three types of selection schemes, while ranking selection and tournament selection have similar performances, except for tournament selection being faster. Finally, steady state selection has the highest base selective pressure of all these types of selection schemes, which leads to the risks of premature convergence as discussed earlier, while on the other hand also leading to a faster convergence [16].

2.3.2 Reproduction

The reproduction step in the GA is where the new offspring is generated from the selected individuals of the previous generation, which can be achieved through crossover and/or through mutation. This is an important step, because it decides how the population will "evolve" to the next generation. In order for this step to be as effective as possible in improving the population, "strong" individuals should be favored for crossover while "weak" individuals should be favored for mutation. This will drive the population towards a stronger average individual.

The crossover operator for reproduction can be separated into three main categories of operators that differ in their arity (the number of parents used to generate an offspring) [13]

- **Asexual**: Each offspring is generated from a single parent.

- **Sexual**: A pair of parents generate one or two offspring.
- **Multi-recombination**: More than two parents generate one or more offspring.

Based on the arity, the crossover operator can vary in how the genes from the parents are combined, and it is often very specific to the chromosome structure and to the problem that we attempt to solve.

2.3.3 Mutation

The mutation operators are used to decide which of the genes in a chromosome are modified after crossover, how they are modified and by how much. The mutation operator is therefore responsible for "disrupting" the evolutionary process, introducing randomness to create diversity. The magnitude of this diversity is called the *mutation rate*, which can either be static or adapt over time based on the population.

An important condition for mutation operators is *reachability*. The entire solution space should be reachable from any arbitrary point in solution space. This will often be difficult to achieve if there are constraints that limit the solution space, and some operators cannot guarantee this condition because of it [17].

An example of a mutation operator is the **Inorder mutation**, in which two mutation points in the genetic string are chosen randomly, and all genes between these points undergo random mutation, modifying their alleles.

2.3.4 Reinsertion

Another, optional, operator in genetic algorithms is the reinsertion operator. This operator comes into play at the end of a cycle, and can be used to guarantee or "force" the top fitness to increase over the generations by picking the best individuals from a generation and guaranteeing that they are included in the next one, independently from the results from the selection and reproduction operators.

2.3.5 Conclusion

By representing the unit setups of WWO as a chromosome where each gene is a unit on the battlefield with an associated unit type, position, strategy and amount, we could evolve the unit setup and test its performance against other unit setups to measure its fitness. This is an important step in generating unit setups/challenges with a difficulty adapted to a given players skill level.

2.4 Evaluation Methods

When evaluating our model, we want to gain insight in each player's experience, and how it differs between playing generic challenges versus playing challenges generated specifically for them.

The Intrinsic Motivation Inventory (IMI) is a set of questions related to a scale of 1-7, designed to assess the subjective experiences of participants in experiments. The participants are asked to state how much they agree with each of a series of statements that can be divided in different "dimensions" of experience, four of which are notable in this work; "Interest/Enjoyment", "Perceived Competence", "Effort/Importance" and "Pressure/Tension". The results in each of the dimensions of the IMI are assembled by adding the scores from each statement in a dimension, inverting it in the case of reversed statements, and then averaging the final sum.

A/B testing is a user experience research methodology where the users of a certain service are subjected to two different versions of the same service. This provides an easy way to compare design and implementation decisions related to user experience simply by comparing the results from the two groups A and B, according to pre-defined metrics [18].

Repeated measures design is an experimental design where each participant takes part in both groups A and B, as opposed to A/B testing, where each participant is only part of one group. Repeated measures design is advantageous over A/B testing in that it naturally requires fewer participants, but in return it comes with some disadvantages. There is a larger risk of fatiguing the participants in repeated measures, and there is also a bias introduced when a participant takes part in both groups. This bias is often mitigating by alternating the order in which the participants take part in the groups.

2.5 Summary

In this chapter we have explored how the psychological principle of Flow motivates us to provide players with challenges of a difficulty that matches their skill in order to improve their player experience. We then discussed how search-based PCG combined with genetic algorithms can serve as an effective way to generate challenges of an adequate difficulty for any player. Finally, we referred to combining A/B testing or repeated measures design with an IMI as a good way to evaluate our model. In the next chapter we will describe the game World War Online, before we detail how all of these components were implemented within the context of Battle Arena and WWO in chapter 4.

3

World War Online

Contents

This work revolves around the video game *World War Online*[1] (WWO) , and more specifically their new 3D game mode “Battle Arena”, which we have contributed to the development of. This work is a collaboration with Chilltime, and as such we have full access to the code of WWO, which allows us to fully implement and try our hypothesis within a commercial game.

3.1 Game Overview

World War Online is a browser-based, free-to-play, real-time strategy game with players from over 50 countries, in which players build armies and bases to conquer and defend areas for their in-game country. The players army can consist of a variety of different unit types, which are unlocked as the player increases his/her *military rank*. They can then use this army by assigning units to defend their bases, by attacking other players bases in the multiplayer mode of WWO, or by going through the pre-determined challenges in a classic level-based progression system in the campaign, which doubles as the tutorial. Our work introduces a new game mode to WWO, *Battle Arena*, which is described in a later subsection.

3.1.1 Unit Types

The units in WWO are divided into 4 different **base unit types** (see figure 3.1), and within each base unit type, there are at least 4 different unit types, to counter each base unit type (see figure 3.2 for example). These are then split into **Normal** and **Supreme** units, where Supreme units are empowered versions of the Normal units, dealing much more damage and having a higher health, but therefore being more costly for the player to add to their army.

Figure 3.1: The 4 base unit types in WWO

Each unit has its own set of stats that the player needs to consider when choosing between the units. These are *Movement*, *Attack Range*, and their overall strength against other base units types within the same *class* (Normal / Supreme). An example of the 4 core units in the “infantry” base type with their stats can be seen in figure 3.2.

[1] World War Online, Chilltime, 2010

Figure 3.2: A screen capture from the game depicting the “Soldier”, “Stinger”, “Bazooka” and “LG1” units and their stats. These are the 4 core infantry units in WWO, each countering a specific base unit type.

Due to the real-time multiplayer functionality of WWO, the battles in the game are split into two parts: the unit setup phase, “Unit Setup”, and the battle replay phase, “Battle Play”. This division was also implemented in the 3D Unity battle system, which is currently only being used for Battle Arena, in order to facilitate migrating the entirety of WWO battles to a 3D version in the future. The division led to some extra preparation work for us, described in section 3.2. In between these two “front-end” parts, there is a processing part “Battle Engine” which is responsible for determining the outcome and process of a battle, and which takes place in another unity application. This system can be seen in 3.3.

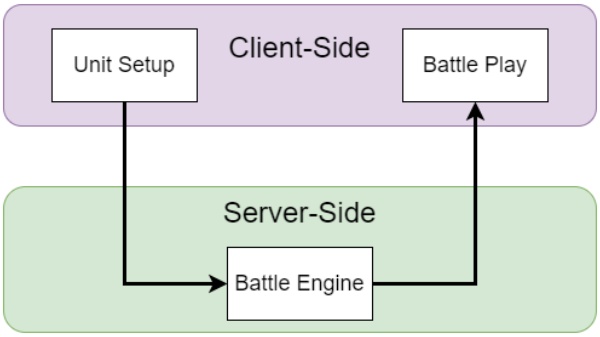


Figure 3.3: The 3 parts of the 3D unity battle system. The player creates their attacking unit setup in the “unit setup” phase, then the “battle engine” processes each action in the battle, before sending the full battle log to the “battle play” part, where the player sees the battle replayed.

3.1.2 Battle Arena

In this work we have assisted the creation of a new game mode in WWO: the *Battle Arena*. WWO is largely built in webGL, and it is a 2D game. Battle Arena therefore distinguishes itself graphically from the rest of WWO by being 3D and built in Unity. Battle Arena is a single player game mode, and in the current version, the players play against challenges that are randomly generated based on their current rating in Battle Arena. The rating system is season-based, and currently resets each week,

distributing rewards to the players with the highest ranking at the end. As players defeat challenges, their rating increases based on how strong their victory was, which is measured by how many units they lost percentage-wise, divided in supreme and normal units. When a player loses, their rating decreases based on the same principle from the opponents perspective. The gameplay loop for a player in the Battle Arena can be seen in figure 3.4. Within the context of "Battle Arena", both **Challenge** and **Level** refers to the **Enemy Unit Setup**. We will in this work always use the term "Challenge".

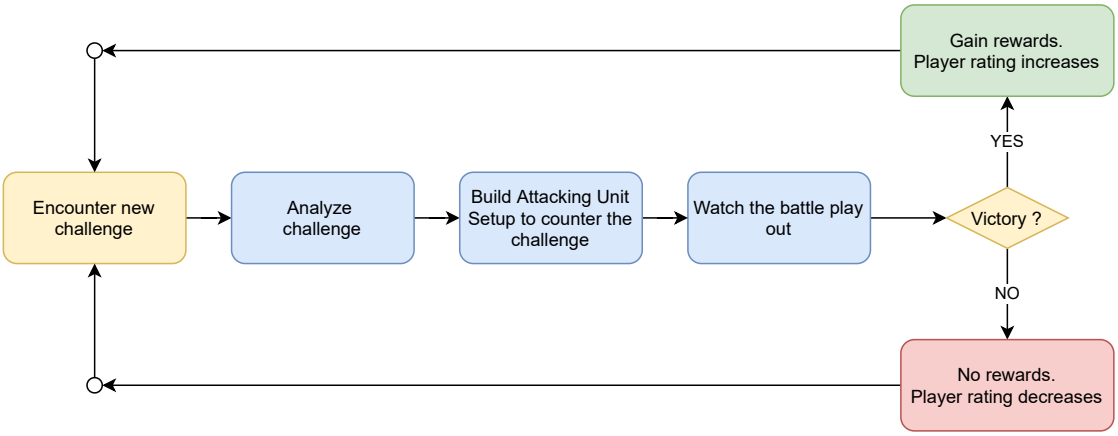


Figure 3.4: The gameplay loop a player goes through while playing in Battle Arena

3.1.3 Player Unit Setup

When a player encounters a challenge in Battle Arena, they have to create an attacking unit setup, consisting of a maximum of 5 different units, to try to defeat the enemy. In this process they have to choose the units, position them on the battlefield, and determine the amount of each unit type they wish to send into battle. The amount of units determines strength and health of that unit type, but for each challenge there is a total budget that the player has to distribute between the units, and each unit lost in battle is also permanent lost to the player. The amount of units to invest must therefore be carefully chosen.

In addition to the type, position and amount, there are also 3 different strategies that each unit can be set to follow. The *Default* strategy causes the unit to spawn at the start of the game and attack its most favorable target until it dies. The *Ambush* strategy allows the player to place the unit on the enemy side as well as their own, and the unit will only appear later in the battle, not at the start. The *Guard* strategy causes a unit to play defensively, and limits its movement to a maximum of 1 tile away from its original position. An example of a player building an attacking unit setup in the Battle Arena can be seen in 3.5 and both a demonstrational video of the setup as well as the resulting battle can be seen on YouTube [19] [20].

Figure 3.5: Battle Arena - The new game mode in WWO. Here the human player "GodVenn" is in the process of choosing his attacking unit setup against the defending computer opponent, whose setup will always be determined beforehand.

3.2 Contributions to WWO Development

When we started the collaboration with Chilltime, "Battle Arena" was still an unfinished game mode. We therefore contributed substantially to its development before being able to use it in our work. This section will briefly detail our contributions in this development.

Due to an error, the 3 parts of the 3D battle system were originally split into 3 different Unity applications, although the "Unit Setup" and "Battle Play" should really just be 2 "scenes" within the same unity application. In order to greatly reduce the load times and memory needed to run Battle Arena, we elected to merge these two Unity applications, which needed to be done manually and with great care so as to not compromise any of the two parts. This merging implied reverse and re-engineering of the existing code base, which proved quite time consuming.

Within the unity application, on the front-end, we finished the User Interface (UI), giving players information about their expected rewards through API calls, as well as their actual rewards at the end of a battle. We added all the sounds to the game, finishing the audio manager system that was already in development. We also finished adding Visual Effects (VFX) to the units 3D models, as well as fixing the animations of all models that consisted of several sub-models, using Unity's animation layers with avatar masks.

In the processing part of the application, "Battle Engine", we contributed with tweaking to the units' decision-making process, as well as optimizations to reduce processing times, as this translates to pure

waiting time for players in Battle Arena. This part of the new battle system needed a lot of follow-up, as it was the key part of the entire system, managing all the units behaviors and interactions. We could unfortunately not use too much of the logic from the original 2D battle system, as that system is turn-based, and along with a shift to 3D, Chilltime also introduced real-time combat, which radically changes the constraints to the units' behavioral logic.

To make "Battle Arena" an attractive option for the players of WWO, we added a leaderboard which tracked all the players rating, reseting weekly and distributing rewards to players based on their placement. To create the leaderboard, we used PHP with HTML and CSS, and Chilltime's internal tool "Auto-UI", which facilitates the creation of UI's within WWO through PHP. The Unix operating system's scheduler "Cronjob" allowed us to schedule the weekly resets through an API written in PHP through Chilltime's Unix-based server.

In addition to gameplay-related contributions, we also made the game fully functional as a part of the rest of WWO, using javascript to run and manage the unity applications, a SQL database to manage all the player and battle data, and finally PhP to manage the webpage in which the javascript is run. PhP is also what we used to write our API's, transferring data between unity and the SQL database.

Additionally, we contributed with maintenance, bug fixes and consistent weekly patching of the 3D Battle System throughout the majority of the duration of the Masters, until the system was stable, and ready for us to use it in our experiment.

3.3 Summary

In this chapter we described the components and mechanics of "Battle Arena", which is a new game mode in World War Online. In "Battle Arena", players are faced with challenges in the form of "defending unit setups", which they have to counter by building their own unit setup. The player can choose up to 5 different unit type, position them on the battlefield and select a strategy and an amount of units. The amount indicates the strength of the unit type in the coming battle, and the player must distribute amounts between units based on a total budget. The battle is then processed and the units move and attack autonomously, with the player as a spectator to the battle itself.

We contributed to the creation of Battle Arena both by working on the actual game in Unity, but also by integrating it into the web-application of WWO, following this new game mode from unfinished prototype to fully implemented feature, as the main responsible during this entire process.

In the following chapter, we will discuss how we created challenges for "Battle Arena" using a Genetic Algorithm, and how we used the processing part "Battle Engine" to simulate battles in this generation process.

4

Challenge Generation Algorithm

Contents

The main part of implementing our hypothesis, after finishing the 3D Battle System, was to create a “challenge generator”, which generates challenges specifically for a player to match their skill level. What we did was to use a genetic algorithm as the search algorithm in search-based PCG to generate suitable challenges for players in Battle Arena (see chapter 2).

4.1 Overview

The goal of the challenge generator is to be able to provide any given player with a challenge that matches their skill level. In order to achieve this, we chose the player’s recent match history as the representation of their skill. This was also done to encourage players to explore different play styles, as the challenge generator would naturally create some sort of counter to their match history, hopefully leading the player to adapt a new play style to defeat the new challenge. An architectural overview of our final implementation can be seen in Fig. 4.1.

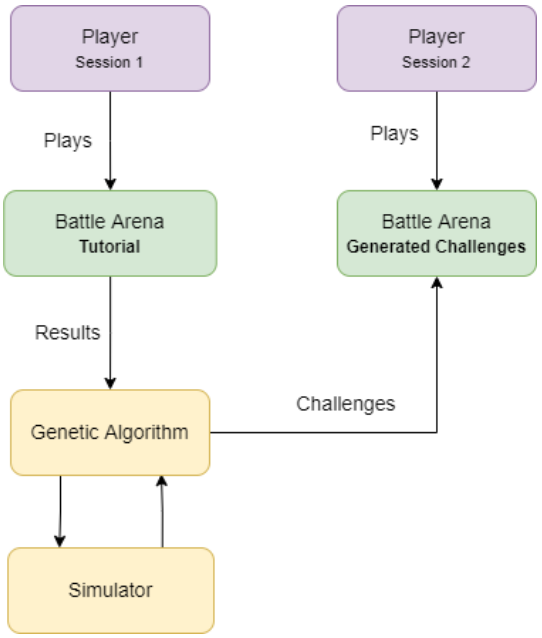


Figure 4.1

For this generation we used offline, search-based PCG (2.2), where the search-algorithm is a genetic algorithm (2.3), and as a consequence the PCG can be classified as stochastic, generating different outputs each time if run several times for a single input. To create the genetic algorithm, we used C# with the *Genetic Sharp* [21] library for genetic algorithms, because it is open source, has been used successfully in several other academic works already, and is also available for commercial use, an important point in our case.

Because this chapter will be relatively specific to WWO we will use the following terms: The *Defend-*

ing Unit Setup refers to the challenge presented to the player, while the *Attacking Unit Setup* refers to the player's own Unit Setup.

4.2 Genetic Algorithm

The genetic algorithm is responsible for producing a defending unit setup that is of an appropriate difficulty based on the player's skill and experience in WWO. In order to achieve this, two important factors are the initial population and the fitness function. We create an initial population which includes all the different unit types available, guaranteeing that each unit is represented at least once in the collection of chromosomes. This is done in order to guarantee covering the solution-space as efficiently as possible in the dimension of unit variety.

The GeneticSharp library provides us with a skeleton for our algorithm, alleviating us from setting up the cycling and the transfer between steps in the cycle, and allowing us instead to focus on creating the operators themselves and defining WWO in terms of variables and components in the genetic algorithm. It also provided us with some basic operators to use, although we ended up only making use of the elitist reinsertion operator provided by the library.

In our work we use two different fitness functions, one fast and approximative which allows us to explore desired characteristics during development by computing the distance to an optimal, hypothetical solution, and another which uses actual combat simulations. The latter fitness function uses the player's recent history of attacking unit setups to evaluate the difficulty of a challenge for said player, by simulating battles between the chromosome's unit setup and the players recent setups, which we call the *Fitness Group*. These will be described in the subsection on Fitness Function, 4.2.3.

4.2.1 Constraints

Before we detail the various components in the genetic algorithm, we will briefly go over the constraints related to generating a unit setup, as this was an important factor in the design process of creating the algorithm.

- **Unique Unit Count**: The total count of unique units for a player on the battlefield is restricted between a minimum and maximum amount defined by the level designer. In WWO, this is typically 1-5 units, but to guide the challenge generation we set this to 3-5 units, which is the typical range. This means that when modifying a chromosome (during crossover and mutation), we must verify that it still has a unit count within these constraints.
- **Budget**: The total *amount* of all units in a setup may not surpass the budget, which is defined by the level designer. There are separate budgets for supreme and normal units.

- **Navy Units & Navy Tiles**: Navy units can only be placed on navy tiles. Navy tiles are found on the two extreme rows of the battlefield. Reciprocally, other units may not be placed on navy tiles. This therefore creates a constraint on a unit's *position* based on its *unit ID*.
- **Supreme Units & Budget**: For a given setup, there are two different budgets to respect, one for supreme units and another one for normal units. This creates a constraint on a unit's *amount* based on its *unit ID*.
- **Certain Units & Strategies** All navy units and certain other units have a movement speed of zero. This means that they can not use the "Ambush" strategy, which implies a constraint on a unit's *strategy* based on its *unit ID*.

4.2.2 Chromosome Structure

The chromosome in our genetic algorithm is meant to represent a unit setup, and as such it needs to contain information on a set of 3-5 different units and their positions. A unit is described by 3 parameters:

- **Unit ID**: The ID which represents the unit type itself.
- **Amount**: The amount of the given unit type. This determines its strength, and is limited by the total budget.
- **Strategy**: A unit can be set to follow one of three different strategies; Default, Ambush or Guard. (See chapter 3)

Our original idea for the structure of the chromosome was to translate the whole battlefield to an array of positions in which units could be "placed", similar to how players in the game create their unit setups, allowing us to perform a sexual crossover which could simply switch parts of the battlefield from both parents to create two new children (see figure 4.2). However, after starting to implement this, we realized that this would work poorly in our case, due to all the constraints of a given challenge/Unit Setup. For example, this structure and crossover could lead to one of the children being empty, which implies that the other child is also outside the constraints as it will have too many units present on the battlefield. Since this crossover was not deemed beneficial, it did not make sense to keep the entire battlefield as a chromosome either.

We therefore switched to a simpler, and perhaps more intuitive, structure, in which the chromosome is simply an array with size equal to the maximum unique units, defined by the user. Each slot in the array can either be empty, or filled with a unit, this time including its position as one of the defining properties of the unit itself. The final structure of the chromosome can be seen in figure 4.3. By allowing for empty cells in the chromosome, we simplify the crossover operator, since we know that all chromosomes will have the same length, despite varying in their unit count.

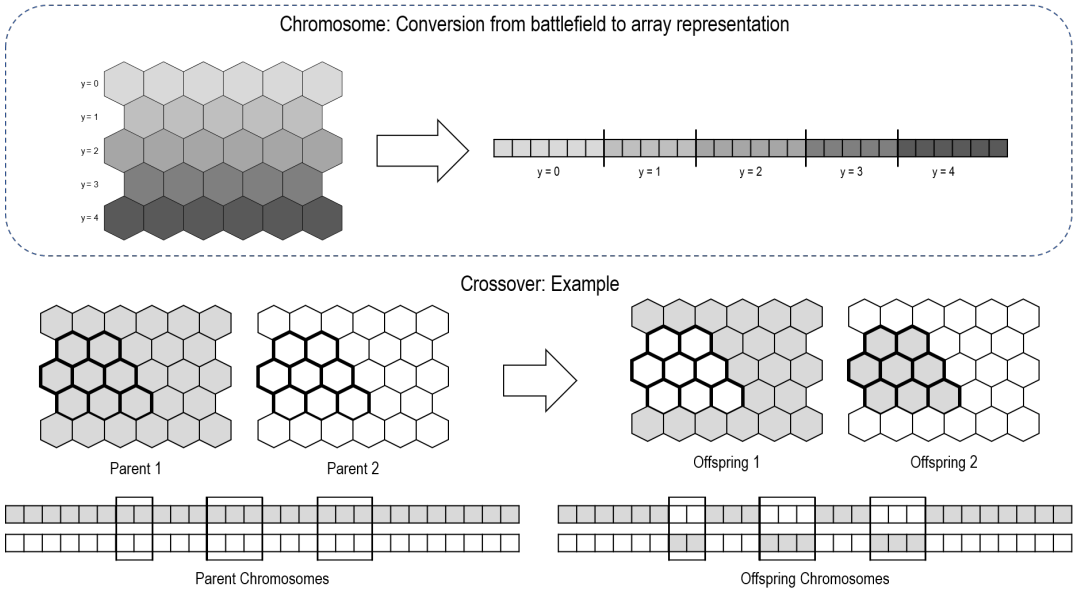


Figure 4.2: The original plan for the chromosome structure and crossover. Top: Mapping a 2D battlefield to a 1D chromosome. Bottom: The related crossover operator.

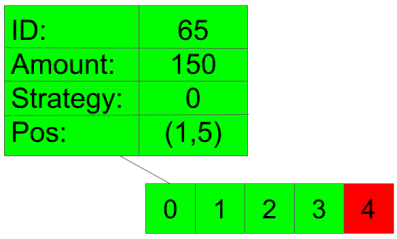


Figure 4.3: An example of a chromosome with the final structure. Here the max unique unit count is 5, as seen by the array size. The array contains 4 unit types, and an example unit is shown for the array index 0.

In terms of data structure within C#, a unit is defined as a *struct*, consisting of 2 integers representing the *Unit ID* and *Amount*, an *enum* representing the *Strategy* and an array of 2 integers to represent the *Position* in x and y coordinates. The *struct* in C# is a value-type variable, which allowed us to easily copy and modify child chromosomes without worrying about the parents from which they came. A chromosome is thus defined as an array of the “Unit Struct” type, and an empty cell is identified by having a *Unit ID* = -1, since all real unit ID’s are unsigned integers.

A chromosome is thus divided in 4 dimensions which are subject to modifications in the crossover and mutation phases of each generation. It can be classified within GA terminology as follows:

- **Chromosome** - Unit Setup
- **Gene** - Unit
- **Alleles** - Unit ID, Amount, Strategy and Position

4.2.3 Main Fitness Function - Simulation

In order to evaluate the adequacy of each proposed solution (unit setup) in the GA, we searched to estimate how each setup would perform against the player for whom the challenge is being generated. To achieve this, we use the processing part of Battle Arena, "Battle Engine", to simulate the battles.

The result of a battle is interpreted into a floating point number between -1.0 and $+1.0$. A negative result signifies that the proposed solution lost, while a positive result means that it won against the "player" (setup in fitness group). The degree of victory/defeat then varies from 0 to 1, where a 0 means that it was a draw, while a 1 means that the winning side did not lose any units, implying a strong victory for that side. To summarize, the percentage of units lost determine the strength of a victory for the winning side. We then compare the average result from these simulations to the desired result, which is a win ratio of 50%, given as input to the algorithm, and normalize the deviation between average result and desired result to a fitness value between 0 and 1 for the unit setup proposed.

The fitness function is divided in 2 classes in our project, the *SimulationFitness* class, which represents the fitness function, and the *BattleSimulator* class, which represents an instance of the "Battle Engine" and handles running it as well as input and output. The input to a simulation is a JSON file containing the two setups that are battling each other. The output is a JSON file containing the final state of the battlefield after the battle, giving us information on the units lost and the winning side.

The *BattleSimulator* generates a hash associated to the instance, and uses this to name the input and output folders for the files. After writing the input file, it then uses C#'s *Processes* to start the simulation, providing the input and output folders as arguments. While the process is running, its output is being redirected to the *BattleSimulator*, which parses the output in a callback function. This callback function registers if the output is an error, in which case the process is aborted and the error message is printed to console, and also checks for the keyword "SIMULATION_COMPLETE", which is how the Battle Engine communicates that it finished processing the battle and writing the output. When the Battle Engine completes, the *BattleSimulator* parses the output and returns the aforementioned value representing the battle result.

The fitness function calls this process sequentially for each of the setups in the "Fitness Group", which is the player's 5 recent setups in our case. It then averages the result and normalizes it to the final fitness value for this proposed solution.

A simplified flowchart of this process is illustrated in figure 4.4.

Because our fitness function involves an external simulation, it is relatively time consuming: the average duration of a full simulation is 3.5 seconds, implying that the duration of a fitness evaluation is equal to this duration times the amount of setups in the fitness group. We can estimate this to be 17.5 seconds, but it does vary based on the complexity of the battles. Naturally, this implies that evaluating the fitness of each setup sequentially would be excessively time consuming. To avoid this, we use

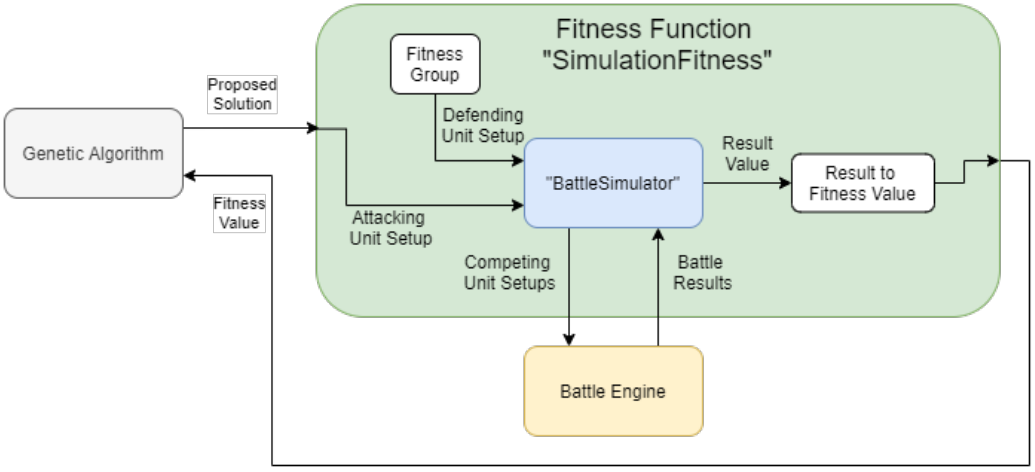


Figure 4.4: A simplified process schematic of the fitness function. **Note:** This illustration is lacking the loop over each setup in the fitness group. The BattleSimulator is run several times, one for each setup in the fitness group, and the result is averaged.

C#'s parallelisation library, which is already integrated with the *Genetic Sharp* library, to evaluate all individuals in a generation simultaneously on different threads of the CPU. We tried to also implement this for the simulations of each unit setup in the fitness group, but it exceeded the capabilities of the CPU used for this project, and proved to worsen the simulation times instead of improving them in our case.

4.2.4 Fast, Approximate Fitness Function

To avoid an excessive iteration time during development, we implemented a second fitness function for the purposes of gathering statistics and optimizing the algorithm. Because the simulation fitness function in theory searches for the optimal unit setup for a given player, we chose to approximate it with a fitness function that searches for a specific, randomly generated, setup. To calculate the fitness value, it evaluates how close a given setup is to the target setup, by attributing a value to each component of each unit in the proposed setup. This function does not care about the order of the units in the proposed setup, as this has no practical importance in WWO. We call this fitness function the *SpecificSetupFitness*, and it is used to illustrate the design choices and optimizations done to the genetic algorithm in this project.

4.2.5 Selection

The selection operator in our genetic algorithm is a standard Tournament Selection Operator (see 2.3.1). We compared tournament selection to roulette wheel selection and found that tournament selection yielded an improved conversion rate for our search. In figure 4.5, we can see how the two operators differ; For runs of 80 generations, the roulette wheel selection seems to stagnate at an average fitness

(in blue) of around 0.8, while the Tournament Selection steadily approaches the maximum fitness value of 1.0. The Tournament selection also has an overall higher conversion rate. For the purposes of these charts, the fitness function used is the approximative, fast version.

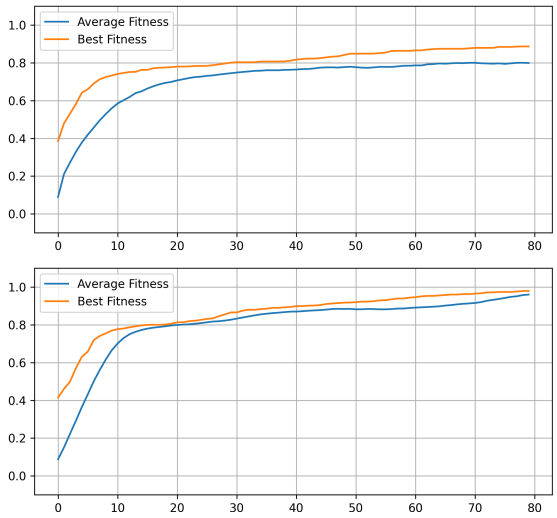


Figure 4.5: A comparison between the fitness evolution per generation of **Roulette Wheel Selection (Top)** and **Tournament Selection (Bottom)**. This data is the average result from 20 runs of the genetic algorithm, each ran for 80 generations, with a population size of 50.

4.2.6 Crossover

An important part of our genetic algorithm, is the crossover operator. It is responsible for mixing potential solutions to create hopefully better solutions, like reproduction in a species. In the case of WWO, we envisioned from the beginning that our crossover operator would somehow switch units between two parents in order to create two children. Originally, we attempted to use a standard 1-point crossover in the array of units that represent the chromosome. This, however, caused a lot of inconsistency in how many individuals in a generation that were actually valid according to the constraints mentioned in 4.2.1.

To illustrate: If we randomly switch two units between the two unit setups (chromosomes), there is no guarantee that the budget is respected, nor is there a guarantee that one of the unit switched is in a unique position (units cannot overlap) or that it has a unique Unit ID (there can only be one of each unit type in a unit setup). We therefore created a new crossover operator, which we called “GeneSwitching”.

This new operator aims to randomly choose a random number of units from each parent and switch them, but verifies in the process that each unit switch is “valid” according to the aforementioned constraints, before making the eventual switch. The budget is not verified in the switching process, as it is not a game breaking constraint, and it is instead corrected for each unit setup at the end of the crossover

process. The probability of a crossover taking place for two selected chromosomes is set to 80%.

Algorithm 4.1: Crossover Algorithm - "GeneSwitching"

inputs: Parent1, Parent2
output: Child1, Child2
$Child1 \leftarrow Parent1$;
$Child2 \leftarrow Parent2$;
$ShuffledGenes1 \leftarrow Parent1.Genes.Shuffle()$;
$ShuffledGenes2 \leftarrow Parent2.Genes.Shuffle()$;
$N \leftarrow RandomInt(1, minChromosomeLength)$; `// The number of switches to perform`
$n \leftarrow 0$; `// The current switch count`
foreach $Gene1$ in $ShuffledGenes1$ **do**
  foreach $Gene2$ in $ShuffledGenes2$ **do**
    if Switching Gene1 and Gene2 is valid **then**
      Replace Gene1 with Gene2 in Child1.;
      Replace Gene2 with Gene1 in Child2.;
      $n \leftarrow n + 1$;
      break;
    end
  end
  if $n \geq N$ **then**
    break;
  end
end

4.2.7 Mutation

The dependencies related to *Unit ID*, mentioned in 4.2.1, made single-dimensional mutation originally impossible without adding a lot of limitations to the mutation itself, which would impede the *reachability* (see 2.3.3) of our mutation operator.

We therefore started off with a mutation operator which could change all the 4 dimensions of a gene (Unit ID, position, amount & strategy) simultaneously. This operator had a different probability for mutating each of the 4 dimensions, and could potentially modify all 4 in a single operation. This caused the search for solutions to become too wide and random, leading us to search for an operator which could do single dimensional mutation, but without impeding its reachability.

One way to achieve such an operator, is to never mutate the *Unit ID* of a gene, as all of the dependencies lead back to this. Instead we would guarantee an initial population which contains at least one gene of each unit ID, spread across the chromosomes, in order to assure that they were all given a chance at being selected for the solution. By doing this, we achieved a single dimensional mutation operator without having to worry about dependencies, while also ensuring full reachability. The difference in performance between the original multi-dimensional operator and the single-dimensional operator with "static" unit ID's can be seen in figure 4.6, where we can clearly see the much improved conversion rate and more stable search of the single-dimensional mutation. The probability of mutation occurring in a

chromosome is set to 15% in our project.

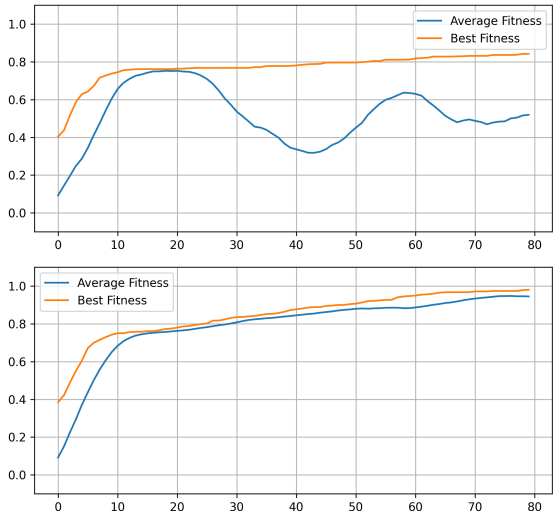


Figure 4.6: A comparison between the fitness evolution per generation of **Multi-Dimensional Mutation (Top)** and **Single Dimensional Mutation (Bottom)**. This data is the average result from 20 runs of the genetic algorithm, each ran for 80 generations, with a population size of 50.

4.2.8 Reinsertion

After the selection and mutation operators, we also used the “Elitist” reinsertion operator. This serves as a way to guarantee that the best solution from each generation is guaranteed to be carried over to the next generation, and thus guarantees that the “best fitness” is monotonically increasing per generation.

The elitist reinsertion is necessary because we use a selection operator that is probabilistic in that a solution with a high fitness value has a higher chance to get selected for breeding, but there are no guarantees that the best solutions will be selected in any given generation, leading to the risk of losing the best solution from one generation to the next.

4.3 Summary

In this chapter we described our method for generating challenges: the genetic algorithm. We are initializing the algorithm with a population size of 50 different chromosomes/unit setups, where each unit ID (unit type, “Soldier”, for example), is guaranteed to be present in at least one of the chromosomes/unit setups to compensate for not mutating the unit ID. The “Tournament Selection” is then being used to select chromosomes for the next generation. In our crossover operator, we switch a random number of random units between to unit setups in order to create two new ones, while abiding to the constraints of the game. The mutation operator then modifies one of three parameters; *Position, Amount or Strategy*,

of a single unit in a chromosome, also abiding to the constraints of the game. We use an elitist reinsertion operator to guarantee that the best chromosome of a generation always makes it to the next generation. Finally, we used two fitness functions, one for development and statistics, and another for the actual challenge generation. The latter one uses the “Battle Engine” Unity application to run simulations for its fitness evaluation, and it is this one we used when conducting our user testing, which we will discuss in the following chapter.

5

Evalutation

Contents

In order to evaluate our hypothesis, we asked players for feedback regarding their experience while playing through the challenges generated by our genetic algorithm. For this purpose we used a repeated measures design, where players played through two sets of challenges, one generated by us, *Version B*, and another "random" set, *Version A*. We used repeated measures design over A/B testing to reduce the amount of participants needed for the experiment. The challenges in version A are the same for each participant, and were generated through the current solution in Battle Arena, for an entry-level rating (see chapter 3). The feedback was given through a questionnaire (see appendix A) which contains an IMI (see chapter 2) section to give us data on different aspects of their experience. In this chapter we will describe this process of user testing and the tools used to gather data.

5.1 Experimental Procedure

The experimental procedure can be divided in two: the playtesting of the game, and the questionnaire to provide feedback about their experience. In the playtesting part, the participants went through a tutorial of 10 levels, before being presented with the two different sets of 5 challenges each. Because our challenge generation is dependent on already having the player's recent match history available, we used the last 5 tutorial levels as this input - the fitness group in our genetic algorithm. A consequence of not having this information before the user testing process, was that we had to split the process into two sessions, scheduled at different times due to the long generation time of our algorithm in between.

Thus, the participants play through the tutorial in the first session, after which we can proceed to generate their challenges, before they play the two sets of challenges in the second session. Both sessions were done via Zoom [22]. We communicated through the voice channel, and shared the link to the questionnaire at the beginning of the meeting. Because Battle Arena is played solely through mouse control, the participants played through the screen sharing function of Zoom, in which we gave them mouse control of our computer. This was a choice we made because having the playtesting directly in the Unity editor allowed for easy switching between game modes and challenge versions, with the added benefit of having everything run locally, facilitating the file management we needed to do; feeding the files containing the participant's result to our algorithm, and later moving the output from the algorithm back into the game as challenges for the player. This process could, of course, be automatized through API's, but we did not have time to implement this.

In the first session, the participants fill out the first part of the questionnaire, which contains an introduction to the game for those that do not already know it, and a demographic section. They then proceed to the first playtesting session, in which they play through 10 tutorial levels of an increasing level of difficulty. The 5 first tutorial levels are copied from the campaign in WWO while the last 5 are actually generated for low-rated players by the current solution in battle arena. These last 5 had to be more

complex than the first because we needed to gather as much information as possible for the generation of challenges described in the previous section. Throughout this part, we were offering help and tips to the participants when they asked for it, to support the tutorial. After the first playtesting session, we scheduled a time for the next session, and extracted the files containing the participant's last 5 unit setups to use these in the challenge generation.

Between the two sessions, our genetic algorithm runs five times to create five different challenges for each participant. In each of the five executions, the participant's last 5 unit setups from the tutorial are used as the fitness group. Our algorithm is able to generate a different challenge each time due to the stochastic nature of genetic algorithms, and the result is a set of challenges in which each one is as individually different as they are in the set of challenges that are randomly generated (version A).

In the second session, the participants play through the two sets of challenges and provide feedback on each of them, following a repeated measures experimental design. The participants are not informed about how the challenges for each version were created, only that challenges have been generated for them since the last session. To mitigate the bias introduced by a participant playing one version before the other, each participant has a 50% chance of starting with either version. After the participant has played through one version, they are presented with the IMI questions to report their experience during that playthrough. They then repeat the same process for the second version, and finally they answer a couple of comparative summary questions in the end.

It is worth noting that because the user testing process is done live through zoom, the participants might have experienced a certain (unintentional) pressure to perform well in the playtesting of the challenges, which we will explore more in the next chapter which discusses the results we gathered.

5.2 Google Forms Questionnaire

We created our questionnaire in *Google Forms*[1], making use of their user-friendly design, accessibility and connection to *Google Sheets*[2], where we gathered the data in a spreadsheet for later processing. The questionnaire consists of a demographic section, two IMI sections (one for each version), and a comparative summary section at the end. We used google forms' conditional section navigation to manage the random order of the versions and the fact that the same questionnaire is answered in two different sessions.

The Intrinsic Motivation Inventory was used for the participants' self-reporting of their experience during playtesting. To investigate the aspects of enjoyment, we used 4 dimensions of the IMI: **Interest/Enjoyment, Perceived Competence, Effort/Importance** and **Pressure/Tension**. From each dimension, we used all the items, for a total of 23 statements that the participant had to evaluate in regards to their

[1] "Google Forms", Google, https://www.google.com/forms/about
[2] Google Sheets, Google, https://www.google.com/sheets/about/

experience, rating them on a scale from “1 - not at all true” to “7 - very true”.

They key dimensions to us are the Interest/Enjoyment and Perceived competence, as we search to increase enjoyment by providing the player with challenges that match their competence. Ideal results for us would therefore be if version B had a significantly higher score in the Interest/Enjoyment dimension and to correlate this to a significant difference in perceived competence between the two versions. The Effort/Importance and Pressure/Tension dimensions were included to provide us with additional data on aspects that could impact the other two dimensions in unforeseen ways.

5.3 Pilot User Testing

Before we began a full-scale user testing process, we did a short pilot with 5 people, in which we took a detailed look at the process, and if there were eventual aspects to be changed, removed or added. We asked the participants in the pilot what they thought of the length of the questionnaire, especially considering the 23 IMI statements for each version, and concluded that this was not an overwhelming amount. During these pilot sessions it also became apparent what aspects of the tutorial we needed to elaborate on in order to give each participant a clear image on what the game is and how it is played optimally, which were then improved accordingly, resulting in an efficient tutorial for the rest of the participants. Due to a low number of total participants, which we will discuss in the following section, the results from the pilot are included in the final results.

5.4 Data Analysis

To analyze the data collected through the questionnaire in google sheets, we used the statistical analysis software “IBM SPSS Statistics” [23], which allows for mapping of variables, filtering, queries and contains a large variety of pre-configured analysis functions. SPSS is based on a spreadsheet structure for its data, making it easy to copy the data from google sheets to SPSS, after which we could map and classify the variables (columns in google sheets) to prepare for analysis.

Due to company changes in Chilltime along with a server change for WWO, our user testing process was unfortunately interrupted unexpectedly and prematurely, as the webflow and API’s used in the playtesting stopped working in this period of time. As we had based our user testing process around the existing architecture of WWO, it would have been too time consuming to work around this, and we ended up with only 13 participants completing the user testing. With such a low number of data sources, the results we can extract from this experiment are limited and not presentable as strong conclusions. Nevertheless, we will discuss our preliminary findings and indications that can be extracted from this small sample size in the following chapter.

6

Results

Contents

In this chapter we will discuss the results from our user testing (appendix B). As mentioned earlier, the amount of data was limited by our few participants (13), so we hope to use these results as indicators for what could be explored further related to our hypothesis. The small sample size also implied that we could not assume a normal distribution of the results, and we therefore only used non-parametric statistical tests. All of the results discussed in this section are based on a scale of 1-7.

The demographics section of the questionnaire shows that our participants were largely young adults with 92% being between 22 and 27 years old, with an even division between casual or dedicated gamers (see Figure 6.1), and with all, except for one, never having any experience with WWO. This last information is especially important, as it implies that the results we collected mainly concerns new players of WWO. The average times spent playing during user testing was 27 minutes for the tutorial and 37 min 12 seconds for the combined set of challenges (Version A + Version B), for a total play time of 64 minutes and 12 seconds on average. This means that these results mostly represent the participants' first impressions, which should be taken into account when interpreting them.

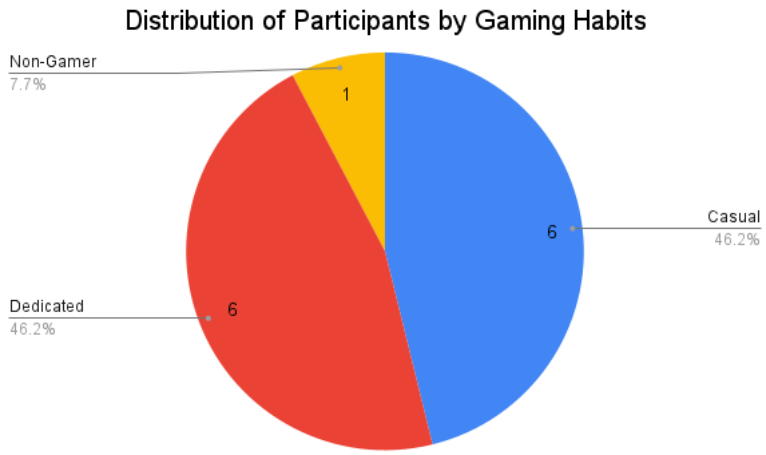


Figure 6.1: Pie chart showing the distribution of participants by gaming habits. There is an even distribution between casual and dedicated gamers, with one participant reporting to generally not play video games.

6.1 Results in IMI dimensions

When performing the Wilcoxon signed-rank test to compare version A and version B in each of the IMI dimensions, there was only a meaningful statistical difference in one of them, the *Perceived Competence*, with $Z = -2.202$ and $p = 0.028$. The mean values of this dimension showed players felt more competent in version A ($mean_A = 4.705$ vs $mean_B = 3.589$) which could indicate that the challenges generated by our challenge generator were more difficult than those in version A, or it could mean that players felt a stronger sense of accomplishment when completing challenges in version A because they

proved more challenging.

The fact that there was no significant statistical difference in the *Interest/Enjoyment* could indicate that the challenges generated did not provide the players with an improved experience compared to those of version A, which would counter our hypothesis, but this would need further exploration with a larger sample size.

6.2 Direct Comparisons

The Wilcoxon signed-rank test also showed significant differences in results for two of the custom questions that focus on a more direct rating of the versions, these being "How would you rate the difficulty of these challenges, compared to your skill level?", hereafter referred to as the **Adaptiveness To Skill**, and "How would you rate the overall experience you had playing through these challenges?", referred to as the **Overall Experience**.

The adaptiveness to skill showed a difference with $Z = 2.295$ and $p = 0.028$, with mean values of $mean_A = 3.770$ and $mean_B = 5.230$. The scale of this question goes from **1 - Too Easy** to **7 - Too Hard**, with 4 being the ideal score of a perfect match between skill and difficulty. These results therefore indicate that the challenges in version A were slightly easier than adequate and that the challenges in version B were noticeably harder than what would be an adequate difficulty, with the challenges in version B being further from the best score on this scale.

The overall experience differed between version A and B with $Z = -2.511$ and $p = 0.012$ and mean values ($mean_A = 6.000$ vs $mean_B = 5.000$) showing that players rated version A as a slightly better experience overall, which counters our hypothesis in which we expected version B to yield a better experience overall.

6.3 Discussion

When combining the results of the perceived competence showing that participants felt more competent in version A, with the results from the adaptiveness to skill showing that version A seems to better match the skill of each participant, they seem to indicate that version B was slightly too difficult for the average participant in this test process. This could be due to the parameter that controls the desired difficulty of the challenges generated, the "DesiredWinRatio" of our Genetic Algorithm's fitness function. For the challenge generation in our experiments, we had this parameter set to 0.50, which means that the challenges generated would ideally break even with the player's match history. Based on these results, we can say that this parameter might be too high, leading to harder challenges, and it could be interesting to experiment further with different values for this parameter.

During the user testing process, we also observed that players would typically rarely lose to any challenge in version A, while often losing 1-2 battles against challenges from version B, which supports the claim that version B is harder than version A.

Regarding the impact of difficulty on experience, it would seem from the results on overall experience that the high difficulty of version B did indeed diminish the experience of playing through those challenges, which matches the theory of Flow.

To summarize, version B was slightly too difficult for our participants, while version A better matched their skill, leading to participants rating their experience in version A as superior. It is worth noting that people tend to rate games as better than others when they perform better at them, so their self-reported experience is not an absolute measurement.

In terms of our challenge generator, these results are not all bad. They confirm that our challenge generator is capable of creating interesting challenges, which is, generally speaking, harder than creating challenges that are too easy. The fact that nearly all of our participants were new to WWO might mean that the challenge generator would have had better results with more experienced WWO players, as these will generally be better players and might appreciate complex challenges in a different way than new players might so that the increased difficulty of version B would be a positive contributor to their experience. The results might also indicate that the player's 5 recent matches do not provide enough information on a player for the algorithm to be able to produce an adequate challenge for them.

In the following chapter we will attempt to conclude on what these results might indicate while also discussing how this work can be continued and expanded upon in order to reach more substantial conclusions in the future.

7

Conclusions

Contents

In this chapter we summarize the development of our challenge generator and the results it yielded concerning our hypothesis, as well as discussing how this work could be expanded on in the future in order to yield more detailed results and possibly conclude on the hypothesis.

7.1 Summary

In this work, we explored how a genetic algorithm could be used as the search-algorithm in a search-driven procedural content generation of challenges in order to improve player experience in the commercial game “World War Online”, in the single-player game mode “Battle Arena”. This work was done in collaboration with Chilltime.

We hypothesized that we could improve a player’s experience in Battle Arena by using their recent match history to create a challenge of a difficulty which matched their skill. We implemented this by creating a genetic algorithm to act as challenge generator which measures the fitness of each challenge candidate by simulating battles between the candidate and the player’s match history using the processing unit of the actual game, called “Battle Engine”, aiming for a 50% win ratio.

To test the hypothesis, we had 13 participants play through two sets of challenges, one static set that was equal for all players - Version A, and one set that was generated specifically for that player based on their match history in the tutorial - Version B. We asked each participant to respond to a questionnaire, in which they compared the two versions indirectly by responding to an Intrinsic Motivation Inventory about each playthrough, and directly by responding to a couple of comparative questions. The low amount of participants was due to changes in Chilltime and to their game servers, which lead to a premature ending of the user testing process.

The final results showed no statistically significant difference in the IMI dimensions of Interest/Enjoyment, Effort/Importance or Pressure/Tension, but showed a difference in Perceived Competence, which showed that participants felt more competent while playing through version A. There was also a significant difference in the Overall Experience and the Skill/Difficulty ratio reported by the participants, where they rated version A as a slightly better overall experience as well as being slightly better adjusted to their skill levels.

Due to the low sample size, we can not reach any strong conclusions from these results, but we have some ideas as to what they could indicate. Taking into account that these results are from players who are new to the game, we can say that it seems the challenges generated by our genetic algorithm were too difficult for these players, which led to a worsened experience when compared to the static challenges. This implies that our current model did not succeed in creating challenges that were adequate for newer players, and did not provide them with an improved experience compared to the static challenges.

We believe that repeating this experiment with players who are experienced in WWO might yield different results, as the challenge generator shows potential to create interesting challenges which newer players might not appreciate the same way. We also believe that different parameterization of the genetic algorithm could yield different results, especially concerning the aforementioned “DesiredWinRatio” which basically sets the desired difficulty for the generated challenge.

7.2 Future work

To conclude this document, we would like to provide some ideas for further exploration of this work. Firstly, we would naturally like to encourage a continuation of testing with more participants in order to gain substantial data for the basis of conclusions. A good way of doing this could be by integrating our model fully with the current game, so that the daily players of WWO would naturally be contributing to the data gathering through in-game logging and measurements, in a continued collaboration with Chilltime.

Regarding the model itself, we believe it would be interesting to explore various sets of parameters for the generation, especially the aforementioned “DesiredWinRatio”, which we believe has a lot of potential to change the results of this experiment, but also regarding the selection of units available to the generator, which could be more dynamically adapted to the players based on their in-game levels.

Finally, our original plan for this work was to combine the challenge generator with a single-player Elo rating system and a challenge database to evaluate challenges over time, and eventually assemble a database of a large variety of challenges for different skill levels, that could simply be searched for matches when a player is going to face a challenge (see Figure 7.1). We believe this combination of techniques could yield even better results due to the self-correcting nature of the Elo rating system over time, and it would also improve wait times for players, as the current generation time is unacceptable in a live game. As a side note to this generation time, a neural network or similar technique might be able to replace the simulation module of our genetic algorithm, which could greatly reduce generation time for challenges.

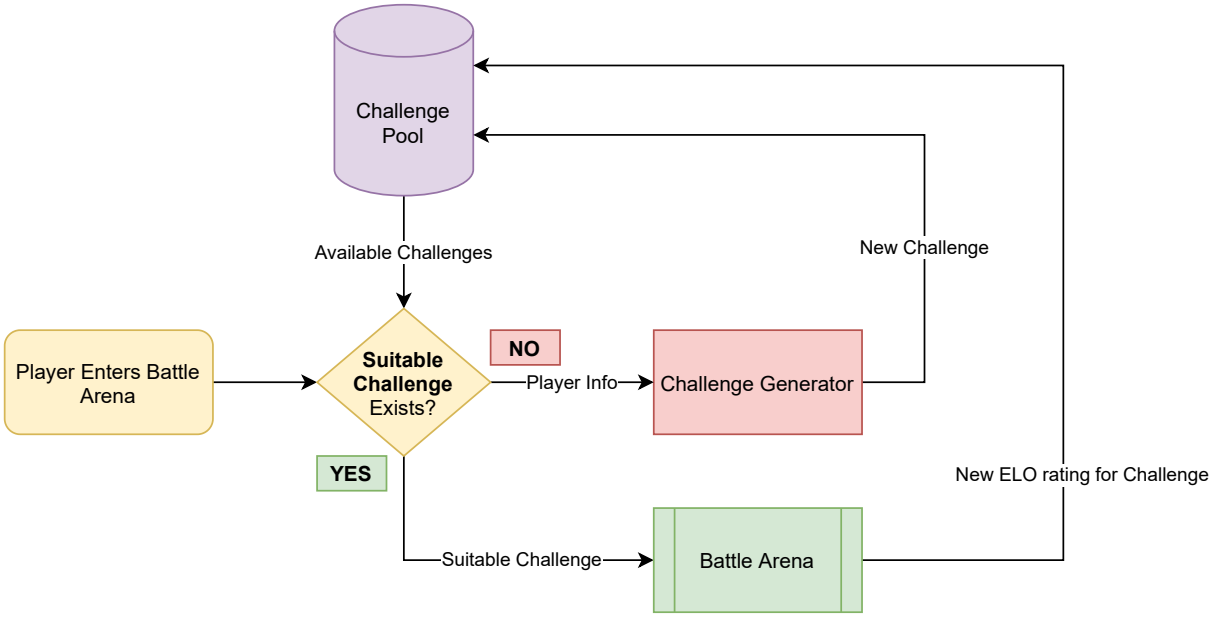


Figure 7.1: An alternative model including an Elo rating system and a challenge database. As the pool of challenges grow, the "Yes path" of the flow will be taken more often than not, leading to the Challenge Generator decreasing in importance over time. (Suitable Challenge = Challenge with an Elo rating that matches the player's.)

Bibliography

[1] L. Iyadurai, S. E. Blackwell, R. Meiser-Stedman, P. C. Watson, M. B. Bonsall, J. R. Geddes, A. C. Nobre, and E. A. Holmes, "Preventing intrusive memories after trauma via a brief intervention involving tetris computer game play in the emergency department: a proof-of-concept randomized controlled trial," *Mol Psychiatry*, 2018.

[2] K. E. Laver, B. Lange, S. George, J. E. Deutsch, G. Saposnik, and M. Crotty, "Virtual reality for stroke rehabilitation," *The Cochrane database of systematic reviews*, 2017.

[3] Gallup, "Gallup global emotions," Tech. Rep., 2020, available at: https://www.gallup.com/analytics/324191/gallup-global-emotions-report-2020.aspx (Accessed: 05 January 2021).

[4] M. Intelligence, "Gaming market - growth, trends, forecasts (2020 - 2025)," Tech. Rep., 2020, available at: https://www.mordorintelligence.com/industry-reports/global-games-market (Accessed: 05 January 2021).

[5] M. Csikszentmihalyi, *Flow: The Psychology of Optimal Experience*. Harper & Row, 1990.

[6] M. Robinson, "Why players are leaving your game," 2013, game Developers Conference (GDC). Available at: https://www.gdcvault.com/play/1020698/Why-Players-are-Leaving-Your (Accessed: 05/01/2020).

[7] J. F. L. Pardal, "Holiday knight: a videogame with skill-based challenge generation," Master's thesis, Instituto Superior Técnico, May 2019.

[8] F. S. Mestre, "Multi-dimensional elo-based challenge progression for single-player games," Master's thesis, Instituto Superior Técnico, October 2020.

[9] M. Hendrikx, S. Meijer, J. Van Der Velden and A. Iosup, "Procedural content generation for games: A survey", *ACM Transactions on Multimedia Computing, Communications, and Applications* (TOMM) 9(1) (2013) p. 1.

[10] J. Togelius, N. Shaker, and M. J. Nelson, *Procedural Content Generation in Games*. Springer International Publishing, 2016.

[11] F. F. N. P. Bicho, “Multi-dimensional player skill progression modeling for procedural content generation,” Master’s thesis, Instituto Superior Técnico, June 2018.

[12] J. Togelius, G. Yannakakis, K. Stanley, and C. Browne, “Search-based procedural content generation: A taxonomy and survey,” *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 3, pp. 172 – 186, October 2011.

[13] A. P. Engelbrecht, *Computational Intelligence: An Introduction*, 2nd ed. John Wiley & Sons, Ltd., 2007.

[14] S. Mascarenhas and C. Martinho, “Darwin’s adventure,” in *Proceedings of the 7th International Conference of Videogame Sciences and Art (VJ’15)*, 2015.

[15] T. Bäck, “Selective pressure in evolutionary algorithms: A characterization of selection mechanisms,” in *International Conference on Evolutionary Computation*, 1994.

[16] D. E. Goldberg and K. Deb, “A comparative analysis of selection schemes used in genetic algorithms,” ser. Foundations of Genetic Algorithms, G. J. RAWLINS, Ed. Elsevier, 1991, vol. 1, pp. 69 – 93. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780080506845500082

[17] O. Kramer, “Genetic algorithm essentials,” ser. Studies in Computational Intelligence. Springer, 2017, vol. 679.

[18] S. W. H. Young, “Improving library user experience with a/b testing: Principles and process,” *Weave - Journal of Library User Experience*, vol. 1, 2014.

[19] T. L. Nesheim, “Thesis - unit setup example,” 2021. [Online]. Available: https://www.youtube.com/watch?v=55uWLoXxU24

[20] ——, “Thesis - battle replay example,” 2021. [Online]. Available: https://www.youtube.com/watch?v=1kBCzThbrpc

[21] D. Giacomelli, “Geneticsharp,” https://github.com/giacomelli/GeneticSharp, 2019.

[22] Zoom Video Communications, “Zoom (version 5.6.4.799),” 2021. [Online]. Available: https://zoom.us/

[23] IBM Corp., “Spss statistics 26,” 2019. [Online]. Available: https://www.ibm.com/products/spss-statistics

Google Forms Questionnaire

Game Experience Questionnaire

Thank you for your participation in this research. This study is part of a Master's dissertation at Instituto Superior Técnico that explores the generation of challenges adapted to the player. This work is implemented on the browser-based free to play strategy game "World War Online".

First, in Part 1, you will be asked some demographic and game-habits related questions. Then, you will play 10 short tutorial levels to learn the basic game mechanics.
After the tutorial, we will need some time to generate the next challenges, and we will ask you to continue the testing with Part 2 after a break of at least 120 minutes, often set on a different date.
Please suggest a time that fits your schedule.

After the break, we will begin Part 2 of the experiment and we will ask you to play against some generated challenges. Finally we will ask you for some feedback on your experience after going through 2 sets of 5 challenges each. These two different sets will be referred to as "Version A" and "Version B", but which one you play first will be randomly decided.

Answering this form and playtesting, not counting the break between sessions, will take, approximately, 70-90 minutes.

We would also like to remind you that:
- participation is voluntary and you can withdraw at any time;
- you can ask any question related to the experiment at any given time;
- you will not be identified at any stage of the study and individual results will not be shared;
- your participation does not involve physical or psychological risks.
- you can change any answer at any time. If you want to go back to a previous section to change an answer, please use the "Back" button present in this Google Form (please refrain from using the back button on the browser, since it can delete your answers).

By proceeding to the questionnaire you are giving your consent.
We thank you for taking the time to answer this questionnaire.

* Required

1. Did you already complete part 1 of this trial and you are ready to proceed to part 2? *

Mark only one oval.

- ◯ Yes, I have completed part 1. *Skip to section 6 (Playtesting Part 2)*
- ◯ No, I will start part 1 now. *Skip to question 2*

Skip to question 2

Demographics

This section aims to provide us with some basic information about you and your gaming habits.

2. What is your gender? *

Mark only one oval.

- ◯ Female
- ◯ Male
- ◯ Prefer not to say
- ◯ Other: _____

3. What is your age? *

_____

4. How often do you play video games? *

Mark only one oval.

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

5. Are you familiar with the "Auto Battler" game genre? (e.g. Hearthstone "Battlegrounds", Dota 2 "Auto Chess", League of Legends "Teamfight Tactics") *

Mark only one oval.

- ◯ I don't play video games / I don't know this game genre.
- ◯ I know this game genre, but I don't have a formed opinion on them.
- ◯ I have played / watched others play this game genre enough to understand that I DON'T enjoy playing them
- ◯ I have played / watched others play this game genre enough to understand that I DO enjoy playing them

6. Are you familiar with the game "World War Online"? *

Mark only one oval.

- ◯ Yes *Skip to question 7*
- ◯ No *Skip to section 4 (World War Online - Short Introduction)*

World War Online Experienced

7. How would you rate your skill level within World War Online? *

Mark only one oval.

	1	2	3	4	5	6	7	
Poor	◯	◯	◯	◯	◯	◯	◯	Very Good

Skip to section 5 (Playtesting Part 1)

World War Online - Short Introduction

World war online is a free-to-play online strategy game. In it, players compete in country-based teams to conquer the world through strategic battles.

In this experiment we will only focus on the strategic battles themselves, effectively ignoring the rest of the game.
In a strategic battle you are faced with a defense consisting of several different units placed on a tile-based map. Your objective is to set up an attack with your units, in order to best defeat the enemy defense.
Unit types counter different unit types, for example: A Bazooka (infantry) is very strong against armored vehicles.

The higher the amount you place of a single unit type, the more damage it can deal, and the less damage it will receive. The unit dies when its amount reaches zero. Distribute your budget between different unit types to establish a tactical setup in order to defeat the enemy without losing too many units.

Navy units can only be placed on navy tiles.

The Units Explained

Units

World War Online has 4 basic **unit types**: Infantry, Armored, Air and Navy. Inside each one, you will find 1 type of unit that has an advantage over another unit type. These are then split into Normal and Supreme Units, where Supreme Units are empowered versions of Normal Units. Given their strength, fewer of them are allowed on the battlefield (max. 25), where a player must make a choice on whether they wish to use a Supreme Unit or a Normal unit, given that the budget is fair and shared. You can know more about it in **Unit Stats** section.

The UI Explained

World War Online - Short Game Trailer

http://youtube.com/watch?v=NPbQ7ug5nwg

Skip to section 5 (Playtesting Part 1)

Playtesting Part 1

Now it's time for you to test the game, please LET ME KNOW if you are reading this. The game needs to be started manually by me. We will finish part 1 through zoom.

First we will ask you to go through some tutorial levels, then we will take a break, before we present you with some more challenging battles.

Please click submit and then "Edit Response" to get a link for later use, before we start the tutorial.
(IMPORTANT: DO NOT CLOSE THIS WINDOW AFTER SUBMITTING - CLICK EDIT RESPONSE).

Playtesting Part 2

Now we would like you to play through two different sets of challenges (5 each), and rate each set after playing.
Thank you for your patience and participation!

Experience Feedback

In this section we will ask you some questions to try to gauge how you experienced playing through the challenges we presented you with in the version you just played.

8. Which version of the challenges are you currently giving feedback on? If you are uncertain, please ask me! *

Mark only one oval.

- ◯ Version A *Skip to question 9*
- ◯ Version B *Skip to question 13*

Version A Feedback

9. How would you rate the overall experience you had playing through these challenges? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very bad	◯	◯	◯	◯	◯	◯	◯	Optimal

10. How would you rate the difficulty of these challenges, compared to your skill level? *

Mark only one oval.

	1	2	3	4	5	6	7	
Too Easy	◯	◯	◯	◯	◯	◯	◯	Too Hard

11. Please rate each of the following statements based on how much you agree with each of them. (1 = Not at all true, 4 = Somewhat true & 7 = Very true) *

Mark only one oval per row.

	1	2	3	4	5	6	7
I enjoyed playing this game very much.	◯	◯	◯	◯	◯	◯	◯
This game was fun to play.	◯	◯	◯	◯	◯	◯	◯
I thought this was a boring game.	◯	◯	◯	◯	◯	◯	◯
This game did not hold my attention at all.	◯	◯	◯	◯	◯	◯	◯
I would describe this game as very interesting.	◯	◯	◯	◯	◯	◯	◯
I thought this game was quite enjoyable.	◯	◯	◯	◯	◯	◯	◯
While I was playing this game, I was thinking about how much I enjoyed it.	◯	◯	◯	◯	◯	◯	◯
I think I am pretty good at this game.	◯	◯	◯	◯	◯	◯	◯
I think I did pretty well at this game, compared to other participants.	◯	◯	◯	◯	◯	◯	◯
After playing this game for	◯	◯	◯	◯	◯	◯	◯

awhile, I felt pretty competent.							
I am satisfied with my performance in this game.							
I was pretty skilled at this game.							
This was a game that I could not play very well.							
I put a lot of effort into this.							
I did not try very hard to do well at this game.							
I tried very hard in this game.							
It was important to me to do well at these challenges.							
I did not put much energy into this.							
I did not feel nervous at all while playing the game.							
I felt very tense while playing the game.							
I was very relaxed while playing the game.							

I was anxious while playing the game.							
I felt pressured while playing the game.							

12. Did you already play through and give feedback on the other version (B)? *

Mark only one oval.

- Yes, I have played both versions. *Skip to question 17*
- No, I will play version B now. *Skip to section 11 (Version B Playtesting)*

Version B Feedback

13. How would you rate the overall experience you had playing through these challenges? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very bad								Optimal

14. How would you rate the difficulty of these challenges, compared to your skill level? *

Mark only one oval.

	1	2	3	4	5	6	7	
Too Easy								Too Hard

15. Please rate each of the following statements based on how much you agree with each of them. (1 = Not at all true, 4 = Somewhat true & 7 = Very true) *

Mark only one oval per row.

	1	2	3	4	5	6	7
I enjoyed playing this game very much.	◯	◯	◯	◯	◯	◯	◯
This game was fun to play.	◯	◯	◯	◯	◯	◯	◯
I thought this was a boring game.	◯	◯	◯	◯	◯	◯	◯
This game did not hold my attention at all.	◯	◯	◯	◯	◯	◯	◯
I would describe this game as very interesting.	◯	◯	◯	◯	◯	◯	◯
I thought this game was quite enjoyable.	◯	◯	◯	◯	◯	◯	◯
While I was playing this game, I was thinking about how much I enjoyed it.	◯	◯	◯	◯	◯	◯	◯
I think I am pretty good at this game.	◯	◯	◯	◯	◯	◯	◯
I think I did pretty well at this game, compared to other participants.	◯	◯	◯	◯	◯	◯	◯
After playing this game for	◯	◯	◯	◯	◯	◯	◯

awhile, I felt pretty competent.							
I am satisfied with my performance in this game.							
I was pretty skilled at this game.							
This was a game that I could not play very well.							
I put a lot of effort into this.							
I did not try very hard to do well at this game.							
I tried very hard in this game.							
It was important to me to do well at these challenges.							
I did not put much energy into this.							
I did not feel nervous at all while playing the game.							
I felt very tense while playing the game.							
I was very relaxed while playing the game.							

I was anxious while playing the game.							
I felt pressured while playing the game.							

16. Did you already play through and give feedback on the other version (A)? *

Mark only one oval.

- Yes, I have played both versions *Skip to question 17*
- No, I will play version A now. *Skip to section 10 (Version A Playtesting)*

Version A Playtesting

Please play through version A, and then come back here and click "next" once you finish.

Skip to question 9

Version B Playtesting

Please play through version B, and then come back here and click "next" once you finish.

Skip to question 13

Summary

In this section we would like to ask you to briefly compare the two sets of challenges you played through.

17. How well did you feel the challenges in version A were adapted to you? *

Mark only one oval.

	1	2	3	4	5	6	7	
Not well adapted at all								Perfect for you

18. How well did you feel the challenges in version B were adapted to you? *

Mark only one oval.

	1	2	3	4	5	6	7	
Not well adapted at all	◯	◯	◯	◯	◯	◯	◯	Perfect for you

19. What was, in your opinion, the main reason for the different adaptiveness (If any) of the two versions?

B

SPSS statistical analysis results

Nonparametric Tests

Hypothesis Test Summary

	Null Hypothesis	Test	Sig.
1	The median of differences between InterestEnjoyment_Total_A and InterestEnjoyment_Total_B equals 0.	Related-Samples Wilcoxon Signed Rank Test	.141

Hypothesis Test Summary

	Decision
1	Retain the null hypothesis.

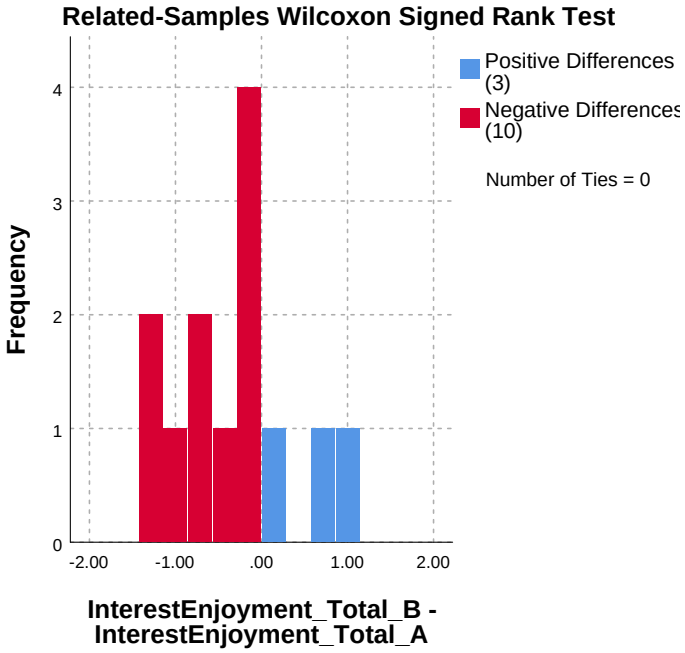
Asymptotic significances are displayed. The significance level is ,050.

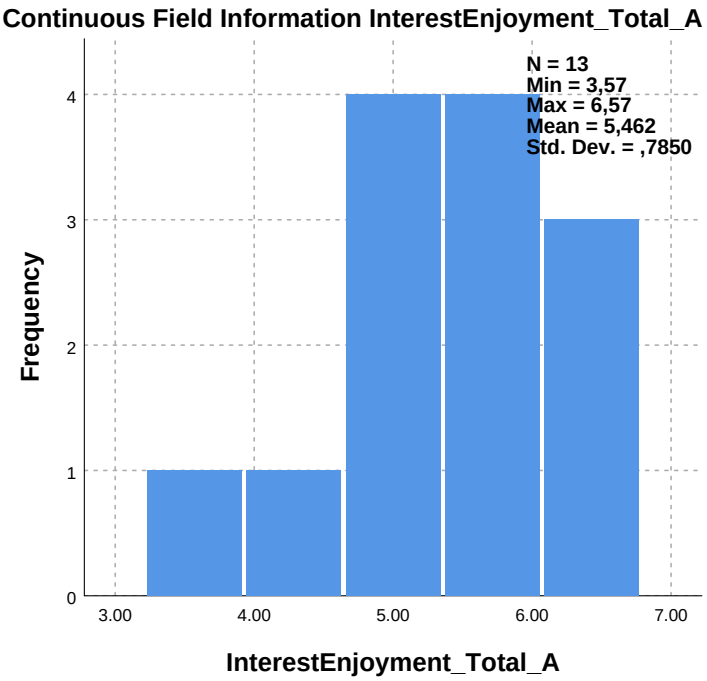
Related-Samples Wilcoxon Signed Rank Test

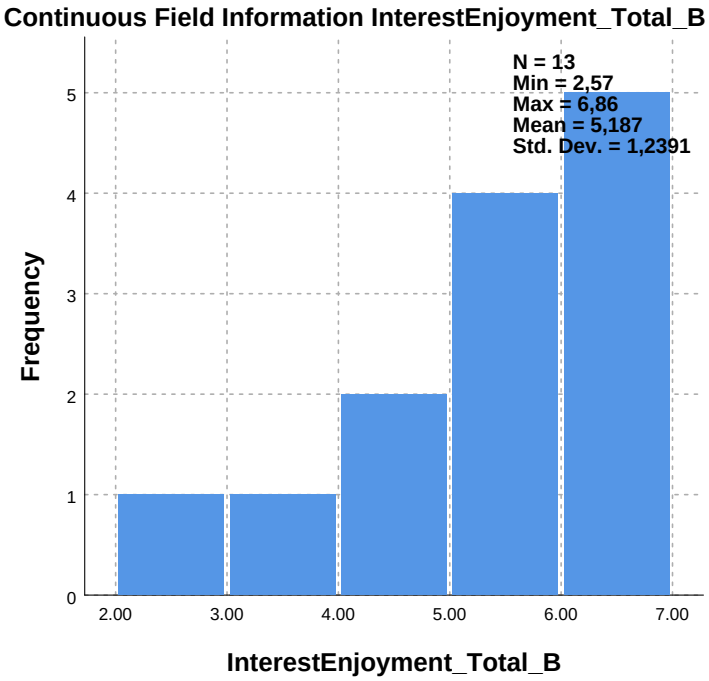
InterestEnjoyment_Total_A, InterestEnjoyment_Total_B

Related-Samples Wilcoxon Signed Rank Test Summary

Total N	13
Test Statistic	24.500
Standard Error	14.257
Standardized Test Statistic	-1.473
Asymptotic Sig.(2-sided test)	.141


Related-Samples Wilcoxon Signed Rank Test
Positive Differences (3)
Negative Differences (10)
Number of Ties = 0
Frequency
0
1
2
3
4
-2.00
-1.00
.00
1.00
2.00
InterestEnjoyment_Total_B - InterestEnjoyment_Total_A


Continuous Field Information InterestEnjoyment_Total_A
N = 13
Min = 3,57
Max = 6,57
Mean = 5,462
Std. Dev. = ,7850
Frequency
0
1
2
3
4
3.00
4.00
5.00
6.00
7.00
InterestEnjoyment_Total_A



Nonparametric Tests

Hypothesis Test Summary

	Null Hypothesis	Test	Sig.
1	The median of differences between PerceivedCompetence_Total_A and PerceivedCompetence_Total_B equals 0.	Related-Samples Wilcoxon Signed Rank Test	.028

Hypothesis Test Summary

	Decision
1	Reject the null hypothesis.

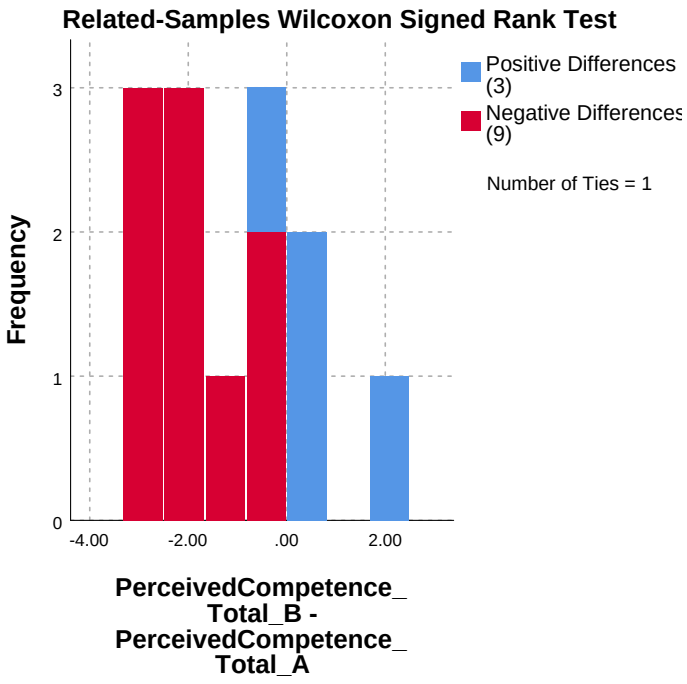
Asymptotic significances are displayed. The significance level is ,050.

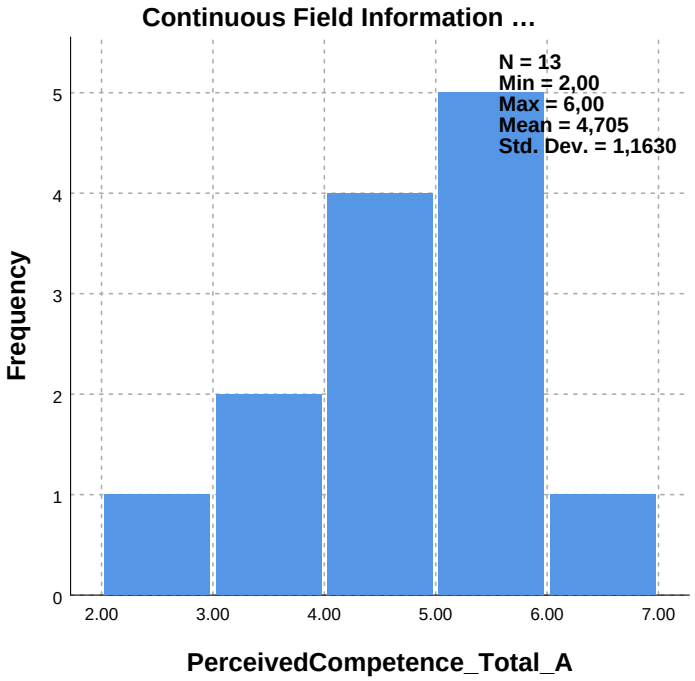
Related-Samples Wilcoxon Signed Rank Test

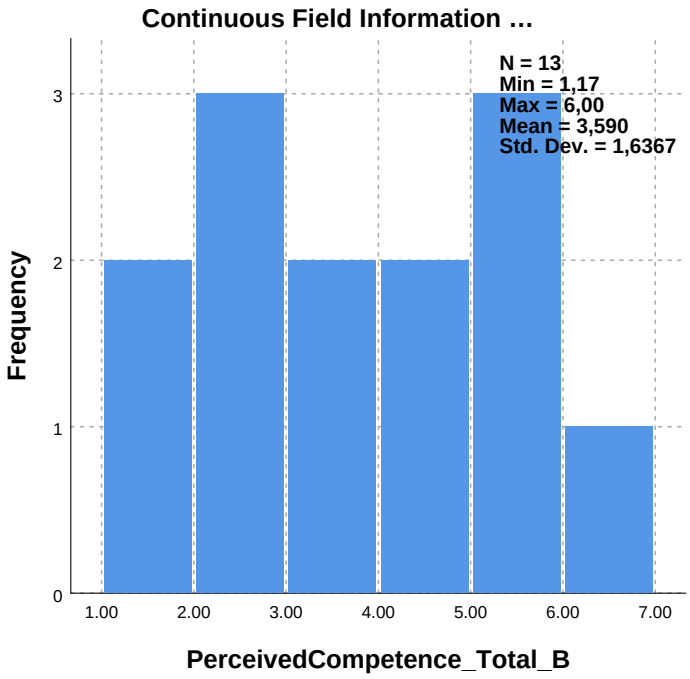
PerceivedCompetence_Total_A, PerceivedCompetence_Total_B

Related-Samples Wilcoxon Signed Rank Test Summary

Total N	13
Test Statistic	11.000
Standard Error	12.718
Standardized Test Statistic	-2.202
Asymptotic Sig.(2-sided test)	.028




Continuous Field Information …
N = 13
Min = 2,00
Max = 6,00
Mean = 4,705
Std. Dev. = 1,1630
Frequency
0
1
2
3
4
5
2.00
3.00
4.00
5.00
6.00
7.00
PerceivedCompetence_Total_A



Nonparametric Tests

Hypothesis Test Summary

	Null Hypothesis	Test	Sig.
1	The median of differences between EffortImportance_Total_A and EffortImportance_Total_B equals 0.	Related-Samples Wilcoxon Signed Rank Test	.592

Hypothesis Test Summary

	Decision
1	Retain the null hypothesis.

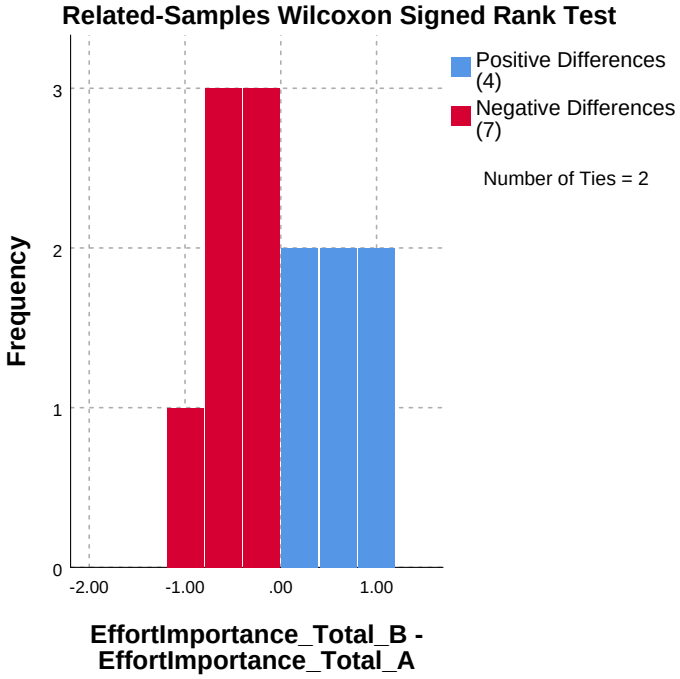
Asymptotic significances are displayed. The significance level is ,050.

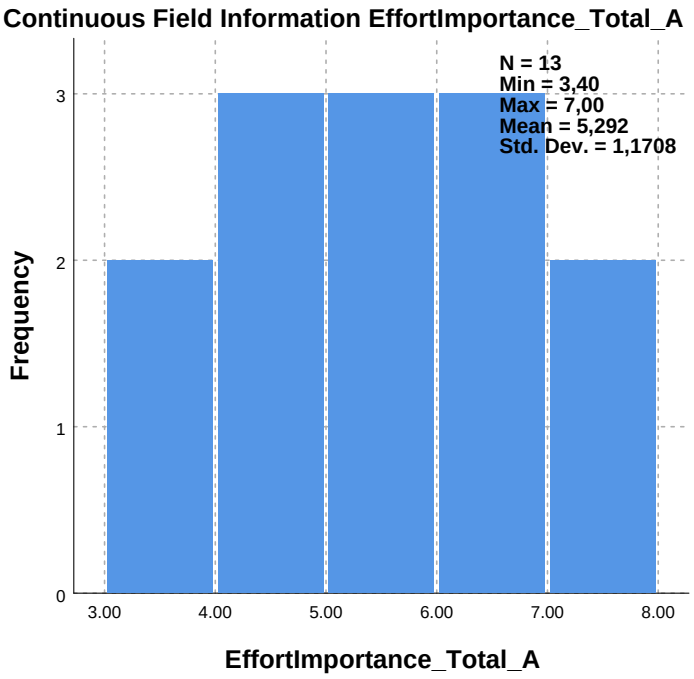
Related-Samples Wilcoxon Signed Rank Test

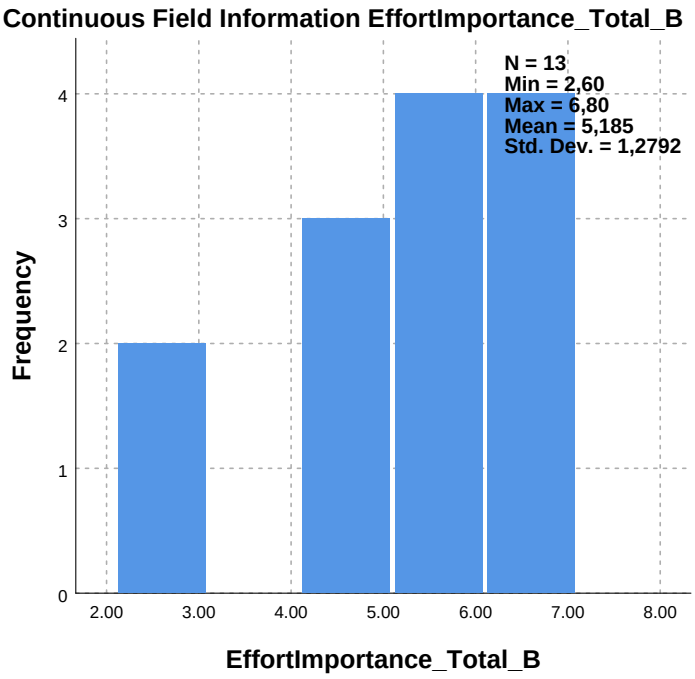
EffortImportance_Total_A, EffortImportance_Total_B

Related-Samples Wilcoxon Signed Rank Test Summary

Total N	13
Test Statistic	27.000
Standard Error	11.208
Standardized Test Statistic	-.535
Asymptotic Sig.(2-sided test)	.592




Continuous Field Information EffortImportance_Total_A
N = 13
Min = 3,40
Max = 7,00
Mean = 5,292
Std. Dev. = 1,1708
Frequency
0
1
2
3
3.00
4.00
5.00
6.00
7.00
8.00
EffortImportance_Total_A



Nonparametric Tests

Hypothesis Test Summary

	Null Hypothesis	Test	Sig.
1	The median of differences between PressureTension_Total_A and PressureTension_Total_B equals 0.	Related-Samples Wilcoxon Signed Rank Test	.265

Hypothesis Test Summary

	Decision
1	Retain the null hypothesis.

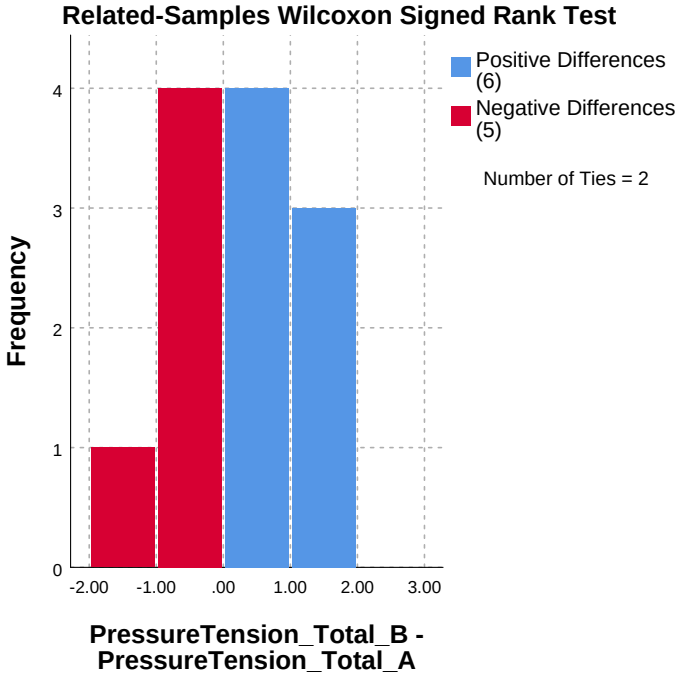
Asymptotic significances are displayed. The significance level is ,050.

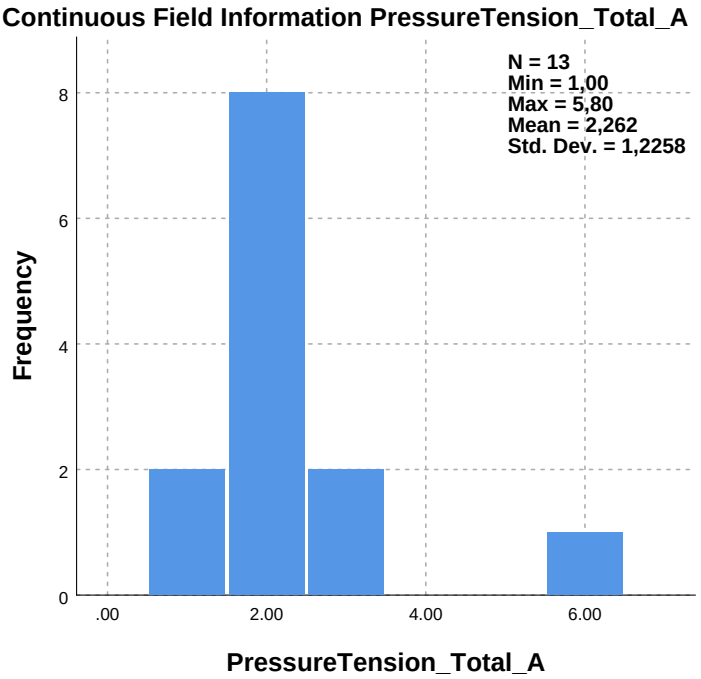
Related-Samples Wilcoxon Signed Rank Test

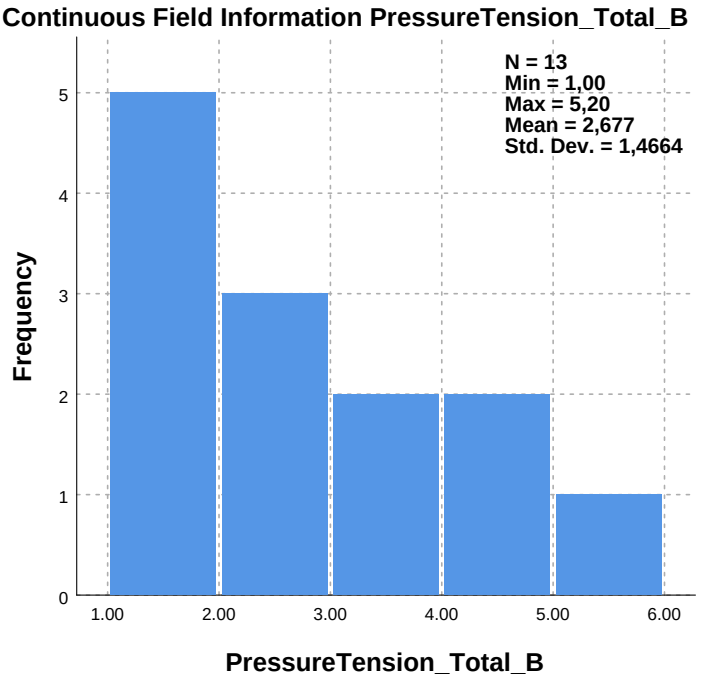
PressureTension_Total_A, PressureTension_Total_B

Related-Samples Wilcoxon Signed Rank Test Summary

Total N	13
Test Statistic	45.500
Standard Error	11.225
Standardized Test Statistic	1.114
Asymptotic Sig.(2-sided test)	.265




Continuous Field Information PressureTension_Total_A
N = 13
Min = 1,00
Max = 5,80
Mean = 2,262
Std. Dev. = 1,2258
Frequency
0
2
4
6
8
.00
2.00
4.00
6.00
PressureTension_Total_A



Nonparametric Tests

Hypothesis Test Summary

	Null Hypothesis	Test	Sig.
1	The median of differences between OverallAdaptiveness_VersionA and OverallAdaptiveness_VersionB equals 0.	Related-Samples Wilcoxon Signed Rank Test	.153

Hypothesis Test Summary

	Decision
1	Retain the null hypothesis.

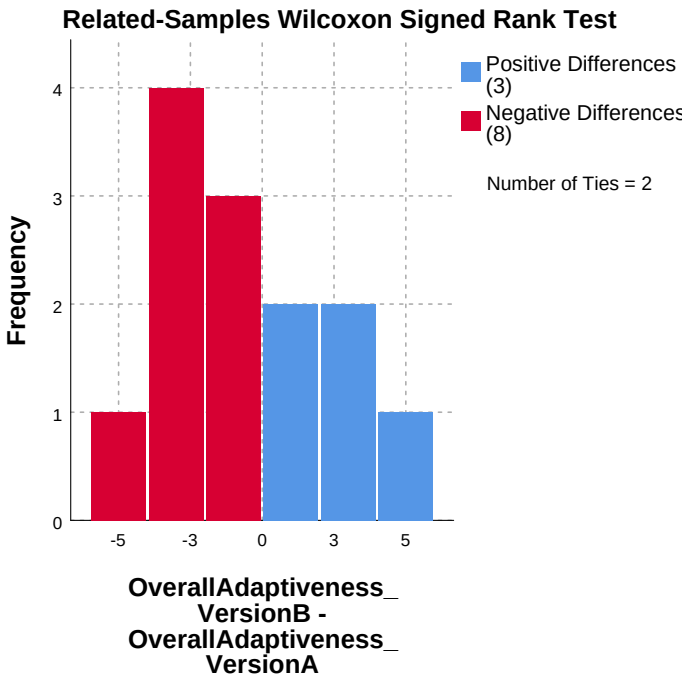
Asymptotic significances are displayed. The significance level is ,050.

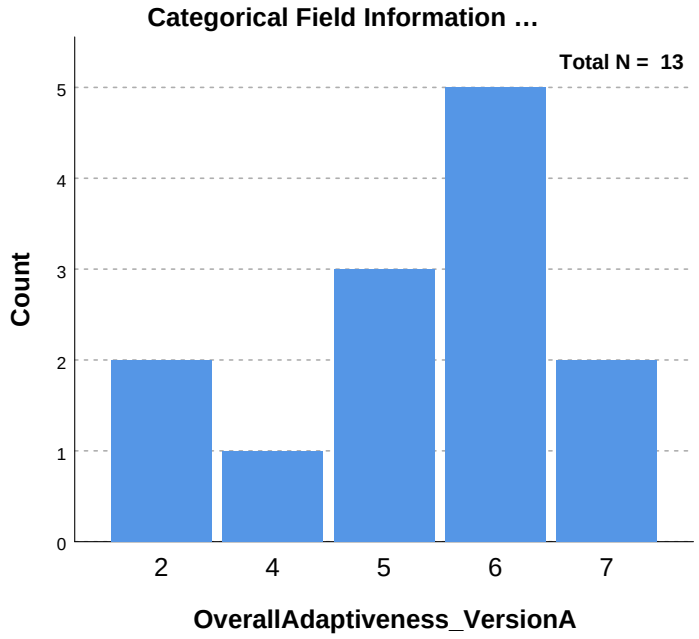
Related-Samples Wilcoxon Signed Rank Test

OverallAdaptiveness_VersionA, OverallAdaptiveness_VersionB

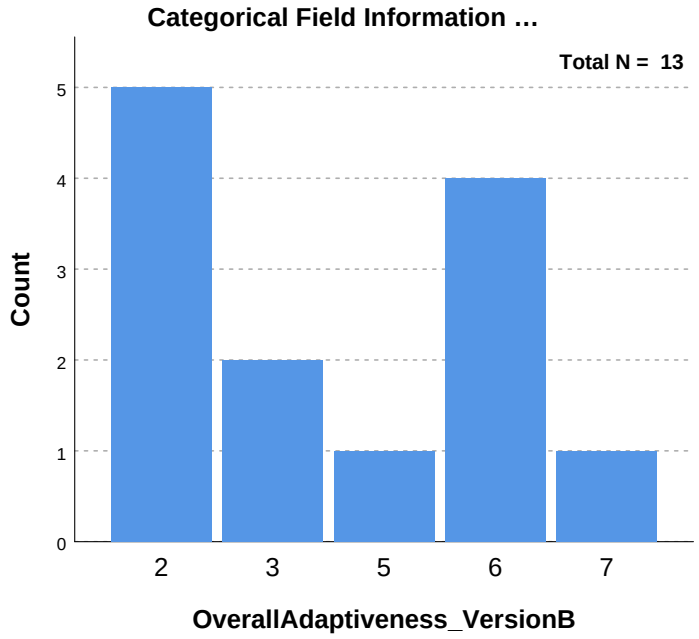
Related-Samples Wilcoxon Signed Rank Test Summary

Total N	13
Test Statistic	17.000
Standard Error	11.192
Standardized Test Statistic	-1.430
Asymptotic Sig.(2-sided test)	.153





OverallAdaptiveness_VersionA field is ordinal but is treated as continuou…



OverallAdaptiveness_VersionB field is ordinal but is treated as continuou…

Nonparametric Tests

Hypothesis Test Summary

	Null Hypothesis	Test	Sig.
1	The median of differences between How would you rate the difficulty of these challenges, compared to your skill level? and How would you rate the difficulty of these challenges, compared to your skill level? equals 0.	Related-Samples Wilcoxon Signed Rank Test	.028

Hypothesis Test Summary

	Decision
1	Reject the null hypothesis.

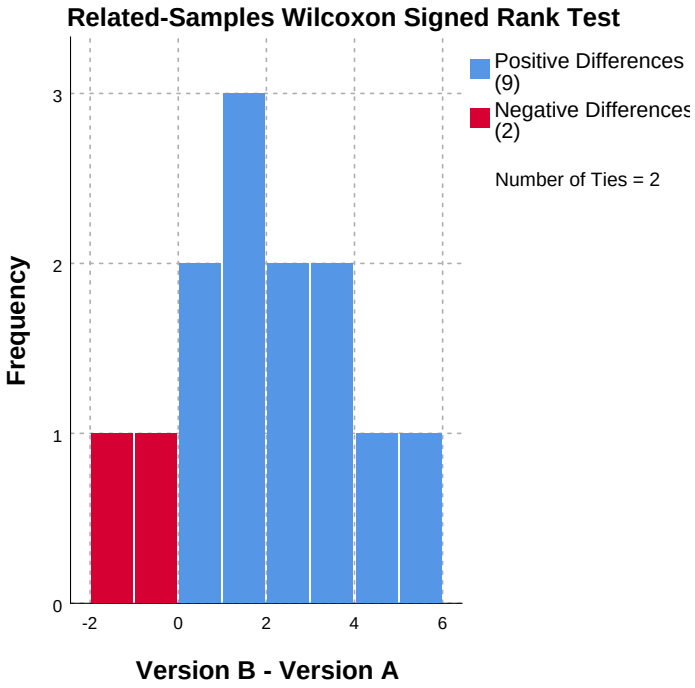
Asymptotic significances are displayed. The significance level is ,050.

Related-Samples Wilcoxon Signed Rank Test

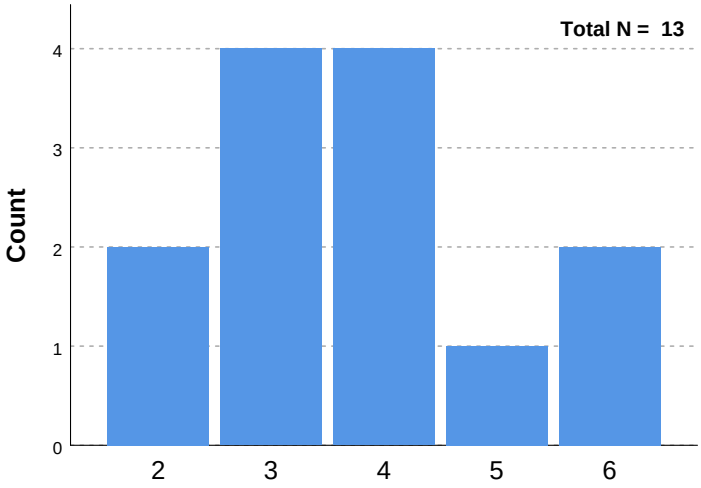
[Version A vs Version B]
How would you rate the difficulty of these challenges, compared to your skill level?

Related-Samples Wilcoxon Signed Rank Test Summary

Total N	13
Test Statistic	57.500
Standard Error	11.164
Standardized Test Statistic	2.195
Asymptotic Sig.(2-sided test)	.028

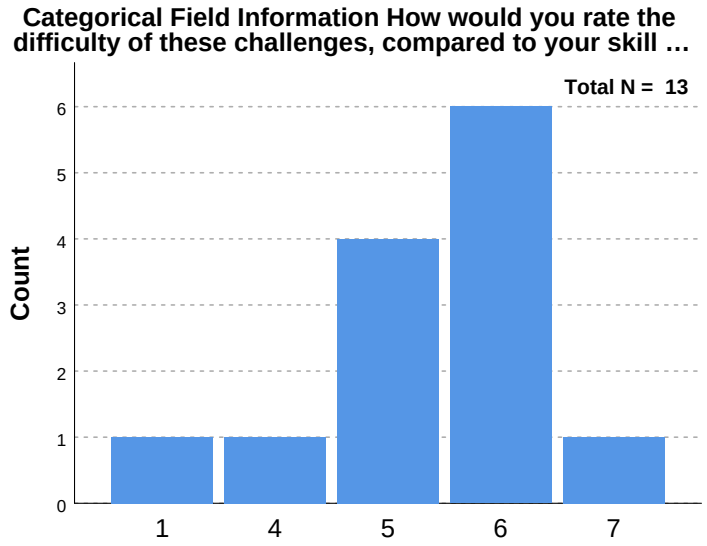

Related-Samples Wilcoxon Signed Rank Test
Positive Differences (9)
Negative Differences (2)
Number of Ties = 2
Frequency
0
1
2
3
-2
0
2
4
6
Version B - Version A

Categorical Field Information How would you rate the difficulty of these challenges, compared to your skill …



[Version A] - How would you rate the difficulty of these challenges, compared to your skill level?

How would you rate the difficulty of these challenges, compared to your skill level? field is ordinal but is treated as continuous in the test.



How would you rate the difficulty of these challenges, compared to your skill level? field is ordinal but is treated as continuous in the test.

Nonparametric Tests

Hypothesis Test Summary

	Null Hypothesis	Test	Sig.
1	The median of differences between How would you rate the overall experience you had playing through these challenges? and How would you rate the overall experience you had playing through these challenges? equals 0.	Related-Samples Wilcoxon Signed Rank Test	.012

Hypothesis Test Summary

	Decision
1	Reject the null hypothesis.

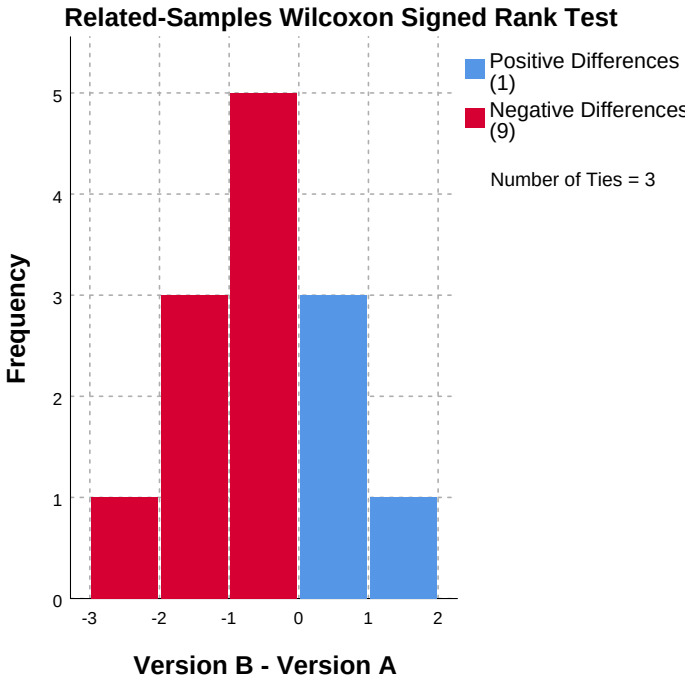
Asymptotic significances are displayed. The significance level is ,050.

Related-Samples Wilcoxon Signed Rank Test

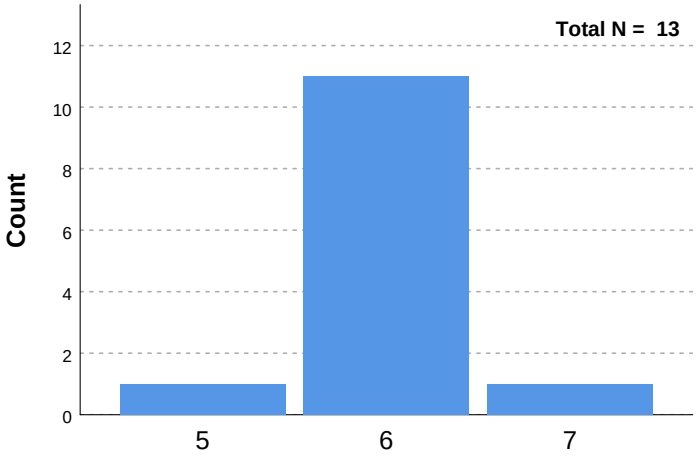
[Version A vs Version B]
How would you rate the overall experience you had playing through these challenges?

Related-Samples Wilcoxon Signed Rank Test Summary

Total N	13
Test Statistic	3.500
Standard Error	9.559
Standardized Test Statistic	-2.511
Asymptotic Sig.(2-sided test)	.012

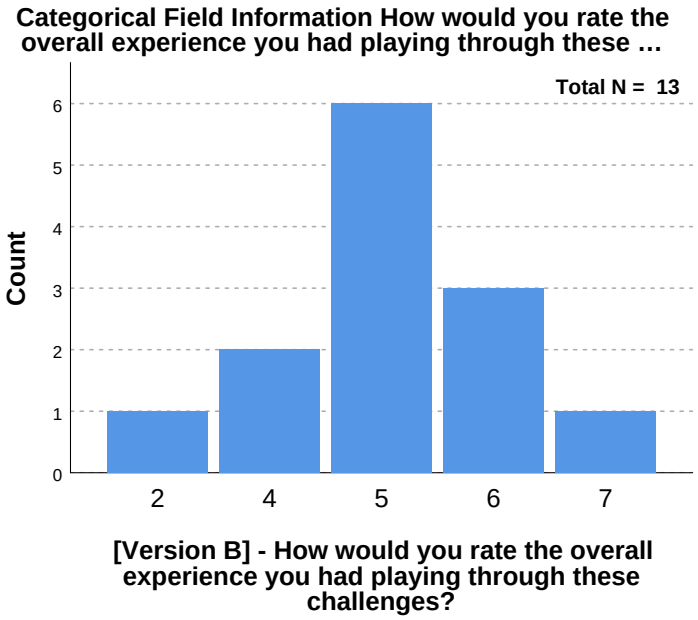

Related-Samples Wilcoxon Signed Rank Test
Positive Differences (1)
Negative Differences (9)
Number of Ties = 3
Frequency
0
1
2
3
4
5
-3
-2
-1
0
1
2
Version B - Version A

Categorical Field Information How would you rate the overall experience you had playing through these …



[Version A] - How would you rate the overall experience you had playing through these challenges?

How would you rate the overall experience you had playing through these challenges? field is ordinal but is treated as continuous in the test.



How would you rate the overall experience you had playing through these challenges? field is ordinal but is treated as continuous in the test.