Using Supervised Machine Learning to Map In-Game Interactions to a Player Behaviour Model based on Game Dynamics

Project Report

No Alexander G. Edvars
Mathias N. G. Faldt
Christian B. Johnsen
Kasper Sørensen

Aalborg University
Department of Architecture, Design and Media Technology

Media Technology
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Using Supervised Machine Learning to Map In-Game Interactions to a Player Behaviour Model based on Game Dynamics

Theme:
Medialogy Masters Thesis

Project Period:
Spring Semester 2022

Project Group:
MTA10_GR2

Participant(s):
No Alexander G. Edvars
Mathias N. G. Faldt
Christian Bank Johnsen
Kasper Sørensen

Supervisor(s):
Claus Madsen

Copies: 1

Page Numbers: 61

Date of Completion:
May 20, 2022

Abstract:

The goal of this project was to develop a supervised machine learning algorithm, which could enable video games to adapt to the preferences of the individual player. This had been done before using unsupervised techniques to develop the player behaviour models. However, these models were being difficult to use in the design process since the mappings were hidden. In this research we used the Core Game Dynamics (CGD) model and questionnaire to model player behaviour. We developed a game based on the CGD model, that would appeal to both the Assault, Manage, and Journey player types. We then gathered data from 102 participants, and used this data to train a supervised learning classifier offline. We then evaluated this classifier and found it had a general accuracy of 48%. Furthermore, we tried splitting the data into different segments with different characteristics. Here we found that segmenting players by the mean difference between player types had a positive impact on performance, achieving an accuracy of 71%.

Contents

Preface

We would like to thank our supervisor, Claus Madsen, for advising us through this project. We would also like to thank the participants who participated in the evaluation.

Since this is our Master's Thesis, we would also like to thank our previous supervisors, test participants, and the staff at Aalborg University for helping us throughout the years.

Finally, we would like to give a shout out to all the group members we have worked with over the last 5 years.

Chapter 1

Motivation & Introduction

Imagine you buy the latest hit video game and that the game tailors itself to your specific play style. As you play, the game will be able to identify your favorite aspects and enhance your experience, and all you have to do is to sit back and enjoy the game. An example of this could be if you do not enjoy action-packed gameplay and want to explore instead. The game detects this and gives you a new mission focused on exploration instead of action.

Games are interactive media and by design they need to grab the player's attention and motivate them to keep playing. However, player's are not the same and have different preferences for their games. Because of this, a game designer either has to focus on a specific player type, which would limit market share, or they could make a game for everyone, which would risk compromising the game's quality.

We want to make it possible for games to adapt to specific players playing a game. To accomplish this we need to fit players into categories which describe their behaviour in the game. The first step is to define these categories from existing models on player behaviour. A machine learning classifier can then automatically fit players into these categories. This is a process which is done outside of the actual game but it requires data on the player's in game behaviour. Therefore, we made a game to gather this data. With this data we will be able to train the machine learning classifier. This classifier can then be used within the game to identify which categories a player fit into as they are playing the game. The game can then tailor itself to this category. However, this tailoring is outside the scope of this project.

1.1 Project Summary

We wanted to make a system that could detect player behaviour and use this knowledge to adapt commercial video games. To do this, we used the Core Game Dynamics (CGD) model [1]. We chose three categories from the CGD model to work with; Assault, Manage and Journey. We used these categories to design and implement a game. Within the game we gathered data on the player's behaviour. Furthermore, we determined the player's preferred category using the CGD questionnaire, see section 1.1.1 on page 3 for an example of this. Using the players' preferred category as our classes, we trained an offline supervised machine learning classifier. This classifier was trained using the player behaviour data from the game as the features. We then evaluated this classifier and found it had an accuracy of 48%. However, we found that players with larger mean differences were easier to classify, achieving an accuracy of 71%. This classifier would give possibilities for video games to adapt their content towards the specific player's preferences.

A flowchart visualising this general outline of the project can be seen in figure 1.1.

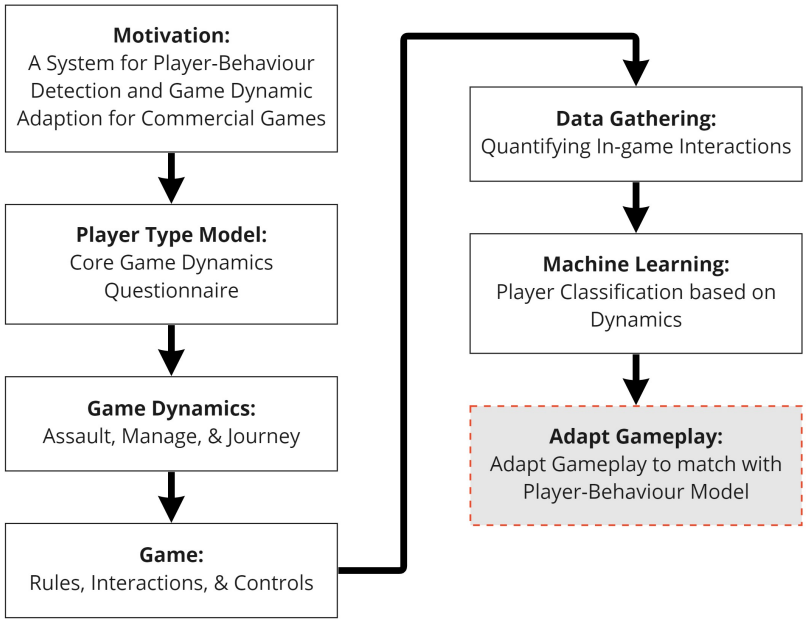

Figure 1.1: *A flowchart which shows a step by step outline of the project. The red outline on the last box signifies that "Adapt Gameplay" is outside the scope of this project.*

1.1.1 Walkthrough of CGD Questionnaire

To give the reader some more context on how the player classes were determined we have created a walkthrough using a short version of the questionnaire and two example players. The questions for Manage is marked in blue , Assault is marked in red and Journey is marked in green .

The Short Questionnaire

In general, how pleasant do you find the following descriptions of game mechanics on a scale from 1 to 7 where 1 is very unpleasant and 7 is very pleasant.

1. **Assault:** Hiding, fleeing, and running for your life.
2. **Manage:** Building, expanding, and enhancing a city, a village, or a base.
3. **Journey:** Collecting rare items and treasures hidden in the game.
4. **Assault:** Shooting multiple enemies and evading enemy fire with fast speed.
5. **Manage:** Acquiring food, equipment, energy, or money through farming, mining, or working.
6. **Journey:** Acting as the main character, immersing in the role and making meaningful decisions.

Example Players: Bob and Hannah

We created two example users; Bob and Hannah, to show two different player types. Bob and Hannah had answered the questionnaire above, their answers can be seen in table 1.1 on page 3.

	1	2	3	4	5	6
Bob answers	7	4	1	7	4	1
Hannah answers	6	6	5	5	6	5

Table 1.1: A table of Bob's and Hannah's answers to the short questionnaire.

Calculating Means

The mean of each category was calculated by summing the questionnaire scores for each category and dividing by the number of questions.

As an example lets calculate Hannah's Assault Mean: ***(6+5)/2 = 5.5***

Calculating Mean difference
The mean difference of each participant was calculated by subtracting the second highest mean from the highest mean.

As an example lets calculate Bob's Mean difference: ***7-4 = 3***

Bob
Bob's means in the different game dynamic categories were:

- 7 Assault
- 4 Manage
- 1 Journey

Bob was an archetype who was clearly an Assault type and he had a ***large mean difference*** of 3 between his preferred game dynamic his second most preferred dynamic.

Hannah
Hannah's means in the different game dynamic categories were:

- 5.5 Assault
- 6 Manage
- 5 Journey

Hannah was the exact opposite to Bob, she was not an archetype and had a ***small mean difference*** of 0.5 between her most preferred game dynamic and second most preferred dynamic.

Chapter 2

Background Research and Related Work

This chapter will describe affective games and how they adapt to the players. The chapter will also describe different player type models and how they have been used before. Finally, the chapter will describe different machine learning implementations in the field of automatic player modeling.

2.1 Affective Games

This section will outline the theory behind affective games. We use the definition from Lara-Cabrera and Camacho. They define affective games as games which react to the player's emotions [2]. This means affective games can adapt the content of the game depending on the player's experience of the game. Because of the interactive nature of games and how the player and the game continually affect each other, the player's experience will also continually change. This means the game should be able to gather data on the player's experience. This data comes from the player, either directly from the player through sensors, or indirectly through the player's behaviour in the game. These types of feedback are broken down to two categories, being "Direct Feedback" and "Indirect Feedback" [2]. This project will focus on the use of indirect feedback to gather data.

2.1.1 Indirect Feedback

Indirect feedback is obtained through the player's behaviour in the game. In other words, you track what is happening in the game and try to understand the player through this data. The idea is that players behave differently in a game based on their emotions. This can be an effective way of collecting data, since this does not require any additional hardware. Hardware which is rarely used for games, and might

be annoying to wear while playing. However, it is more difficult to have reliable methods to understand what a player feels through their gameplay. This is because the data gathered through indirect feedback has been curated by the game designer and therefore might be biased [2].

An example of indirect feedback used for game adaption was Ries et al. [3]. They created a maze game that would adapt the difficulty based on the win rate of previously completed mazes. The adaption worked by either removing obstacles for the player to make the game easier or putting up new obstacles to make it harder. They were able to train a reinforcement learning algorithm to adapt the difficulty of a game in real time to achieve a 50% win-rate.

2.2 Player Types in Games

Different people play games in different ways. Some people enjoy action sequences in games, while others enjoy social or management aspects of games. Some want to spend a lot of time exploring the game world and some want to quickly progress through the game. Several studies have tried to categorise people into different player type models based on these differing preferences and behaviours.

The most well known player type model was the Bartle player types [4], however this model had been widely discredited since it could not be validated for use in research [5, 6, 7]. Furthermore, there were other models like the DGD1 player type model by Batemann and Boon [8], this framework was based on the MBTI personality type model by Myers and McCaulley [9]. The MBTI has, however, also been criticised because it assumed that personality types were mutually exclusive. This criticism might extended to the DGD1 model, making it a non-viable option [5]. The HEXAD model by Marczewski et al. was another way of categorising players, but the model was created specifically for gamification and serious games [10].

The model we chose to use was the Core Game Dynamics model, as it focused on game dynamics, which were easier to design around.

2.2.1 Core Game Dynamics (CGD)

Vahlo et al. categorised players into five different core game dynamic preferences. These were: Assault, Manage, Journey, Care, and Coordination [1]. They clustered these categories in to seven different player types.

The Core Game Dynamics model was based on 700 commercial video game reviews. These reviews were codified, which then resulted in the chosen dynamics for the

questionnaire. From these dynamics and the codification a questionnaire was developed. This questionnaire consisted of 33 questions rated on a likert scale from 1 to 7. These were questions about different aspects of video games and game elements. Depending on how a player answered this questionnaire, the player would get a result stating which of the five categories they preferred. They then used a factor analysis on the dynamic preferences to arrive at the five CGD categories. Finally, they used unsupervised machine learning to cluster their data and then created player types based on these clusters [1].

However, in this report we will focus on the CGD categories instead of the player types. This is because it is simpler and easier to work with and allowed us to focus on fewer categories. We also think that knowing the dynamics that are relevant to each player type will make it easier to decide which data and interactions to log for the machine learning algorithm. In this report we will refer to these categories as player types.

2.3 Using Machine Learning to Classify Player Types

This section will outline the state of research on using machine learning (ML) to classify player types. With this in mind we will start by outlining the concept of *model-based* and *model-free* approaches to data-driven player modelling.

Yannakakis et al. defines the *Model-free* approach as a hidden mapping from the player's game input, to the player's internal state and motivation. An example of this could be unsupervised clustering of data gathered from gameplay. These clusters would represent the player's current state of mind. These clusters could then be analysed to determine the features that are important to the player's state [11]. However, this analysis is often difficult since the mapping is often considered a "black-box", i.e. there is no clear connection between the classifier result and the input data. *Model-based* means that the model of player behaviour and player states are based in a theoretical model. This model would typically be based in social science or Human Computer Interface theory. Using a model-based approach enables the designer to relate the data on the player's internal state back to known theories on player behaviour. The model-based approach is analogous to using supervised learning in ML, since the data is classified to known outcomes. The disadvantage of this approach is that some of the gathered data and behaviour might be irrelevant or misleading to the classification task. This could result in lower accuracy of the final player model [12].

2.3.1 Related Work

This section will outline current research that utilizes ML to classify player behaviour in games.

A large body of model-free approaches that try to model player behaviour exist. These approaches range from clustering of player data [13, 14, 15, 16], to state transition modeling [17, 18]. In terms of research using model-based approaches, we have only been able to find a small amount of research [19, 20].

Odierna et al. used a model-free approach to determine if data mining in the online MMORPG "World of Warcraft" could be used to cluster different player types [15]. They gathered information on the amount of time each player spent in different areas of the game world. They then clustered this data. This meant that they could use their knowledge of player types and the areas within the game to relate the clusters back to Bartle's four player types. Anagnostou and Maragoudakis [14] did something similar, where they clustered player data and later assigned player types to these clusters. The game they developed was a clone of space invaders, and they tracked data like accuracy, kills, and lives used.

Drachen et al. used unsupervised learning to classify different play styles in "Tomb Raider: Underworld" using data from 1365 players [16]. They found four distinct player types through this analysis. They highlighted that using unsupervised learning could be used to find patterns in games of different genres. However, they conclude that the data features would need to be game specific, e.g. shooting accuracy is only relevant to shooting games and not a racing game. Furthermore, they argue that in order for data analysis to be useful, the ML classifier needs to be translatable from data to player behaviour and game design terminology. This last point is especially relevant to our research since this it is what we want to do.

Cowley et al. researched real time classification of player types using a model-based approach in a Pacman-like game [20]. They used the DGD1 player types by Bateman and Boon [8]. They used supervised learning to classify the "conqueror" player type against the rest of the player types in the DGD1 model. To design the ML classifier they analysed the traits of the conqueror type and identified the data needed for classification. Their solution achieved an accuracy of 70%. However, they argued that they could have achieved an even higher accuracy with a model-free approach. The main goal was for the classifier to be translatable into player types and player behaviour. Their research had a similar goal to our research. However, they had a binary approach to classification, meaning that they only identified whether players were a specific player type or not. We wanted to take a more nuanced approach to the classification, since we believed this would work better for game adaption.

Fortin-Côté et al. used direct feedback and a model-based approach to adapt the game "Assassin's Creed: Odyssey" to fit different play styles [19]. They gathered biometric data while players played missions containing three different play styles. The players were then asked to inform about the amount of fun they had during the play session. They did this while watching a video of their playthrough. This data was then used along with biometric data to train a ML classifier. They found that the classifier could predict how well the players liked the mission with an accuracy of 52%. They cite the fact that players might have liked one style of mission the most, but this did not mean that they disliked the other mission styles. This made classification difficult, as the data was not binary and therefore it could not easily be matched with ML classes. Our project could face a similar challenge since the player types were evaluated separately and it was therefore possible for a participant to fit into multiple categories making the match with singular ML classes less accurate.

Model-based vs. Model-free Approach

From the research we saw that the model-free approach was the most effective at classifying player behaviour in games. However, it needed data from a pre-existing game to classify player types. This meant that it did not know the player type behaviour when designing the game. This made designing adaption that is based on the same metrics as the player type classification almost impossible. Furthermore, the model-free approach had an unclear relation between the classifier and the implemented game design terminology. Meaning that game design could not be directly translated to the classifier. Using the model-based approach allowed one to design the game and its adaption before creating the classifier. Furthermore, it allowed for designers to better understand the relationship between player behaviour and the classifier. However, the accuracy of the model-based approach was much lower, since a model on player behaviour was only an approximation of the real player behaviour patterns. We will use a model-based approach using the CGD model.

2.4 Problem Formulation

We chose to focus on indirect feedback as it is more relevant to commercial games. This was because the additional hardware necessary for direct feedback would be a barrier of entry for potential players.

We would use the CGD model to categorise the players and design the game. The CGD model had the potential to be a better tool for player classification because it was based on game dynamics which were easy to relate to game design. This also meant it was easier for us to determine the data features relevant to the ML classifier. Furthermore, the designer could then relate the classifier back to player behaviour.

The model-based approach was better for creating an understandable ML classifier. Because of this it could be used to inform the adaption in a game. Therefore, we decided to use it to create a ML classifier using the player types from the CGD model as the classes for a supervised classifier.

Given this, our problem formulation became:
We wanted to investigate how to automatically classify a player's player type from the CGD model. We wanted to do this by using a model-based supervised Machine Learning on data gathered through indirect feedback.

2.5 Project Design

This section will outline the process and requirements used to go from problem formulation to final product.

In order to train a supervised ML classifier we need classes to classify players into. These classes are based on the CGD questionnaire. To gather data for a ML classifier we need to implement a game. Since the game also defines the features available to the classification, we need to outline requirements to guide the game design. The requirements are as follows:

a ***Requirement A:*** The game needs to implement the "Assault", "Manage" and "Journey" dynamics from the CGD.

b ***Requirement B:*** The game must not bias the user to interact more with one game dynamic over the other.

c ***Requirement C:*** There must be aspects of the game that are quantifiable, meaning the in-game interactions lend themselves to data gathering.

d ***Requirement D:*** The gameplay must be segmentable, such that the game has discrete points for player type measuring and gameplay adaption.

When a game that fulfills these requirements has been implemented, we can gather data for the offline training of the ML classifier. Then we will train the classifiers and test its performance. The rest of the goal is outside the scope of this project, meaning we will not use this classifier in real time, to adapt the in-game content.

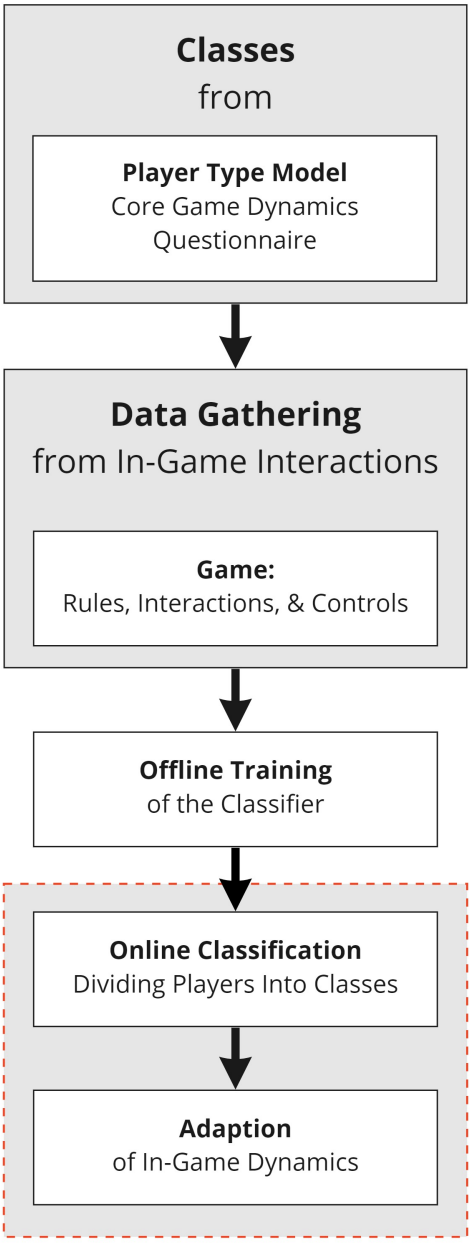

Figure 2.1: *Flow chart representing the project design. The red outline around the last two boxes represent what is outside the scope of this project.*

Chapter 3

Game Design

This chapter will describe the process of the game's development. Section 3.1.1 will briefly describe the game, its genre, and its gameplay to create the foundation from which the following sections will be based on. Then, section 3.1.2 will describe how the gameplay mechanics were designed in accordance with the Assault, Manage, and Journey categories of the Core Game Dynamics questionnaire (CGD). Following that, section 3.1.3 will describe mechanics of the game. Section 3.2, will then describe how the game was designed around the gathering of specific data, to be used to train our classifiers. The last section 3.3, will then explain how everything was implemented over multiple iterations with playtesting between each of these.

3.1 The Game

This section will pitch the final version of the game and the overall game concept. It will then outline how we designed for each of the game dynamic categories. Finally it will describe how the rest of the game's mechanics and features were used to enhance the overall experience.

3.1.1 Pitch

We decided to create a 2D action-adventure game inspired by games such as "Don't Starve" by Klei Entertainment (see an example of gameplay in figure 3.1 on page 14), and the original "The Legend of Zelda" by Nintendo.

Creating a 2D game would reduce the complexity of the controls, camera, characters and more, which would make the tracking implementation easier. We chose to make a top-down 2D game over a side-scrolling 2D game. This was because it enabled players to more easily explore, which accommodated the Journey player type better. This meant that the game was decided to be of the following genre:

Figure 3.1: *Gameplay screenshot from the game "Don't Starve" by Klei Entertainment.*

Single-player, top-down, 2D, action-adventure game. For convenience of testing and gathering of data, the game was made for the PC platform.

For the narrative and game background, we used a gothic setting mildly inspired by Konami's "Castlevania"-universe, and popular media versions of Bram Stoker's "Dracula" and Mary Shelley's "Frankenstein". This was illustrated by creating a world with undead monsters, dark rituals, villagers in need of help, and monster-slaying heroes. These things would be apparent in the visuals and setting of the game. For these visuals, a pixel-art art-style was chosen based on the availability of free assets. Pixel-art was also easier to make compared to 3D assets.

In this world the player would control a nameless hero helping a village in a gloomy and cursed land. They had five days to save the village from impending doom. Either by killing monsters, gathering resources, or unravelling the secrets of the world. The level in which the gameplay took place was a large square map with a border of trees and some decorative objects such as mushrooms, tree stumps, and water found within. Enemies, collectible resources, and lore objects would be spawned randomly within this map each in-game day. The spawn-rates of different types of objects were fixed, such that the amount of each type of object (resources, lore objects, and enemies) would be the same each time, and only the positions and objects within each category would change. The player would start in the center of the level, where a house would represent the village, and a guard-character would present the player with contextual points of interest and narrative motivation. You can see what the start of the game looks like and the guard's dialogue in figure 3.2 on page 15.

The final game was the result of an exhaustive iterative process. This will be further detailed in section 3.3 on page 25. Over the course of these iterations; the world, narrative, and presentation thereof changed as feedback was received from play testing. Furthermore, the game should be able to accommodate the three different CGD player types chosen from the CGD questionnaire: Assault, Manage, and Journey.

Figure 3.2: *An image showing the start of the game and the guard's dialogue.*

3.1.2 Core Game Dynamics Implementation

The designed and implemented mechanics and dynamics needed to represent the descriptions from the CGD questionnaire. This was done to ensure that the game supported each of the three player types (Assault, Manage, Journey). The assumption was that tracking how the player interacted with the game dynamics would make it easier to identify a player's player type through machine learning.

Due to the scope of this project we could not implement all of the dynamics described by the CDG model. Instead, we decided to select a couple of dynamics related to each player type. We chose the dynamics we thought were easiest to implement and gather data from. We also prioritized the ones Vahlo et al. [1] had outlined in the CGD model as the most significant contributors. We then identified the experience, rewards, and gameplay loops which matched the dynamics. We used this to design a game which afforded the chosen player types.

Before translating from the dynamics described in the CGD questionnaire, it was important to consider how dynamics and mechanics relate to each other. Hunicke et al. describes this relationship in their MDA framework [21]. The MDA framework describes dynamics as: *"the run-time behaviour of the mechanics acting on player inputs and each others' outputs over time"*. It describes mechanics as: *"the particular components of the game, at the level of data representation and algorithms"*. In simple terms, dynamics are the things the player experiences in the game like shooting and running. Mechanics are the ways in which these dynamics are implemented through rules and controls [21].

Assault

The following Assault dynamics from the CGD questionnaire were used to design the game (sorted by importance in respect to the dynamics):

- Shooting multiple enemies and evading enemy fire with fast speed.
- Killing, murdering and assassinating by shooting, stabbing or by other violent means.
- Hiding, fleeing and running for your life.

In the game, the player was able to aim and shoot fireballs. This could be used to kill enemy monsters that would chase and try to attack you when you got into close enough proximity to them. Alternatively, the player could choose to run away from the enemies who would stop chasing the player at a certain distance.
We also included some unique enemies in the game which we called major enemies. These were enemies that would be visible on the map and they were both larger and had more health than regular enemies. The purpose of these major enemies was to allow players to more consciously choose what part of the game they wanted to engage with. See figure 3.3 on page 16 for an example of Assault gameplay.

Figure 3.3: *An image showing Assault gameplay, where the player is shooting at enemies while evading enemy fire*

Manage

The following Manage dynamics from the CGD questionnaire were used to design the game (sorted by importance in respect to the dynamics):

- Acquiring food, equipment, energy or money through farming, mining or working.
- Building, expanding and enhancing a city, a village or a base.
- Upgrading and improving objects, vehicles and weapons.

In the game, the player could find and gather resources like wood, berries, and stone for the village. You can see what that looks like in the game on figure 3.4 on page 17.

When enough resources had been gathered, the player would receive a prompt and a notification sound. This would let them know that they could upgrade the village that doesn't matter. The player would be able to do this by returning to the center of the map and interacting with the village guard. This would change the village sprite into a different and larger one representing the player's progress. This upgrade could be performed four times, with an increase in the resource cost to perform each upgrade.

We also included some unique resources in the game which we called major resources. These resources would be visible on the map and they were gatherings of multiple resources. The purpose of these major resources was to allow players to more consciously choose what part of the game they wanted to engage with.

Figure 3.4: *An image showing manage gameplay from the game, where the player is about to collect berries, and can proceed to collect the rock right after.*

Journey

The following Journey dynamics from the CGD questionnaire were used to design the game (sorted by importance in respect to the dynamics):

- Exploring the game world and uncovering the game's secrets, mysteries and story.
- Collecting rare items and treasures hidden in the game.
- Acting as the main character, immersing in the role and making meaningful decisions.

In the game, the player would be able to navigate around the level and interact with lore objects with distinct visuals when in close enough proximity to them. When interacted with, a text box would appear with some object-specific lore text. The player would then be able read through 2 to 4 sentences of lore text. This text was

meant to provide the player with some narrative background to the world. An example of this from the game can be seen on figure 3.5 on page 18

The game also featured unique lore objects, which we called major lore objects. These objects never changed positions in the level. They were designed to be more significant in respect to their lore and narrative implications. They were also visually distinct from their randomized counterparts. This would make them larger discoveries and allowed the player to consciously choose which aspect of the game they wanted to interact with.

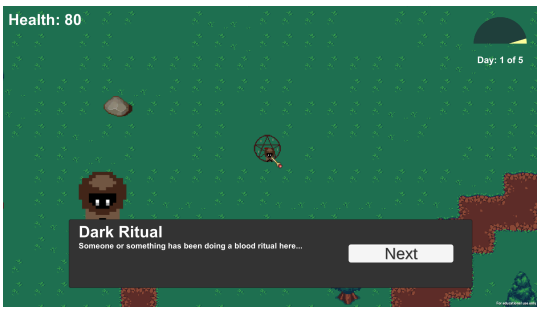

Figure 3.5: *An image showing the player interacting with a lore object, and the character is telling themselves what they are seeing, which is shown by the dialogue box.*

Balancing Across Player Types

In playtesting players who identified with one player type, noted that they interacted more with elements of a different player type. This was because they felt the game rewarded that behaviour. Because of this the dynamics had to be balanced to avoid all of the players playing like the same player type. For this the dynamics needed to be presented as equally important in respect to gameplay and narrative. Furthermore, they should also be equally rewarding.

To balance out the game's dynamics, and ensure that they would be equally engaging, the design was changed over multiple iterations as described in section 3.3 on page 25. One example of how the Manage dynamic was made more engaging, was by adding visual effects and sound effects to increase the satisfaction of interacting with resources. Furthermore, for all player types, the guard character would specifically note, how each of the three dynamics would help the village. This introduction by the guard character also helped as a text-based presentation of possible in-game actions alongside the tutorial displayed right beforehand.

3.1.3 Mechanics, Rules, & Controls

This section of the report will describe the basic functionality and mechanics in the game.

The player moved around in the game world using the "WASD" keys on the keyboard. They would use the mouse to aim their weapon, and the left mouse button to shoot. The "E" button was used to interact with the objects in the game world. The players were told about this functionality through the tutorial screen seen in figure 3.6 on page 19. The players were able to revisit this screen by using the "ESC" button.

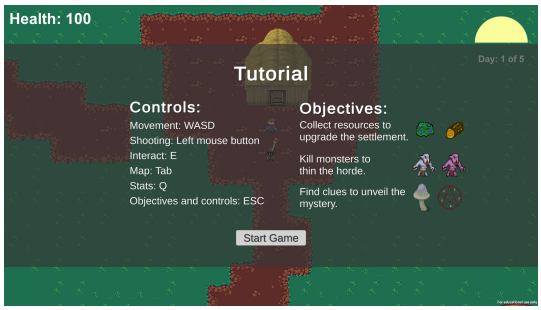

Figure 3.6: *An image showing the tutorial screen in the beginning of the game.*

Using the "TAB" button the players could open the game map, see figure 3.7 on page 19. The game map was implemented to make sure the players could make a deliberate decision on where to go in the game world. The map would start out completely black, and reveal areas in a radius around the player as they moved across the map. See figure 3.8 on page 20 for an image of the game map without fog of war.

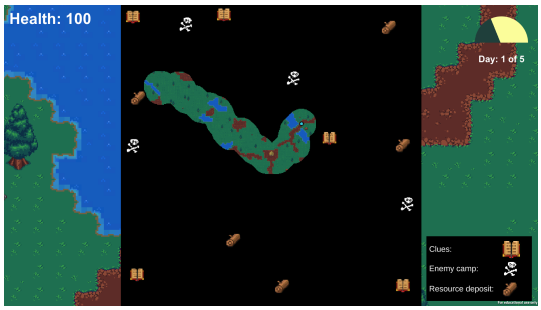

Figure 3.7: *An image showing the map screen within the game. Only a part of the map has been revealed.*

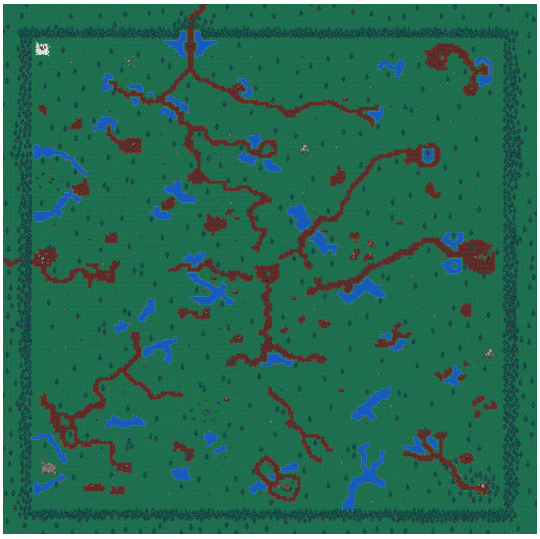

Figure 3.8: *Figure of the entire game map without fog of war.*

3.2 Data Gathering

Gathering data through the player's in-game behaviour was an integral part of the design of the game. Without the right data, the machine learning would not be able to train an accurate classifier. Consequently, data gathering became a focus point of the game's design from early on in the project.

When we gathered data for the machine learning classifier, we had to consider the time interval we gathered the data within. Data from a single frame is too little, and an infinite amount of data is impossible to process. Therefore we needed to choose a time interval. This interval needed to be at most 15 minutes long to keep the experiment comfortable for the participants. To solve this, we chose to implement a day system, segmenting gameplay into five days (or five data intervals) of two minutes each. This would total a ten minute experience. This strategy was further supported by Valls-Vargas et al., who noted that predictive performance of machine learning algorithms is best between 1 to 2 minutes [22]. Furthermore, research suggested that player behaviour changes over the time spent playing [20, 12], meaning that we could detect changes from day to day.

3.2.1 Features

We gathered data on the players' interactions and behavior in the game. The full list of the features we tracked can be seen in table 3.1 on page 23.

In regards to the CDG player types it was particularly important that we tracked all related interactions. This meant that for enemies, lore objects, and resources we tracked; interactions, proximity, and if they had appeared on screen. These features were hypothesized to be directly relatable to their player dynamic preference category. An Assault player type would be assumed to have killed and interacted more with enemies when compared to others. Furthermore, data such as time standing still, reading time, and distance from the starting position was also tracked. These features were hypothesized to contain patterns useful to the machine learning classifier. Many decisions on what data to track came from brainstorms in which the implemented mechanics described in section 3.1.2 on page 15 were considered regarding which data could be logged for each interaction.

Feature:	**Explanation:**
Shots fired	The amount of fireballs the player fired.
Kills	The amount of normal enemies the player killed.
Distance travelled	The distance the player travelled.
Deaths	The number of times the player died.
Distance from spawn	The maximum distance the player moved away from the village.
Time standing still	The amount of time the player was standing still.
Unique tiles	The number of unique 'tiles' the player moved across in the game.
Hits taken	The number of hits the player took from enemies.
Enemies seen	The number of enemies that appeared on the screen.
Enemies close	The number of enemies the player moved close to.
Major kills	The number of major enemies the player killed.
Major enemies seen	The number of major enemies that appeared on the screen.
Major enemies close	The number of major enemies the player moved close to.
Resources	The number of resources the player gathered.

Feature:	**Explanation:**
Resources seen	The number of resources that appeared on the screen.
Resources close	The number of resources the player moved close to.
Major resources	The number of major resources the player gathered.
Major resources seen	The number of major resources that appeared on the screen.
Major resources close	The number of major resources the player moved close to.
Lore interactions	Number of times the player interacted with a lore object.
Lore seen	The number of lore objects that appeared on screen.
Lore close	The number of lore objects the player moved close to.
Lore reading time	The amount of time the player spent reading lore.
Major lore interactions	Number of times the player interacted with a major lore object.
Major lore seen	The number of major lore objects that appeared on screen.
Major lore close	The number of major lore objects the player moved close to.
Major lore reading time	The amount of time the player spent reading major lore.
Opened map	The number of times the player opened the map.
Opened stats	The number of times the player opened the stats page.
Times paused	The number of times the player opened the pause screen.
Map time	The amount of time the player spent looking at the map.
Mouse time	The amount of time the player spent moving the mouse.
Mouse movement	The distance the player moved the mouse.
Time close	The amount of time the player spent close to the village.
Time medium	The amount of time the player spent at a medium distance from the village.

Feature:	Explanation:
Time far	The amount of time the player spent far from the village.

Table 3.1: Feature dictionary. A list of all of the features tracked within the game.

3.2.2 Navigation Behaviour

We also wanted to see if there were any patterns in the players' navigation within the game. With the purpose of classifying their behaviour as either focusing on exploring or points of interest. A system was implemented to track movement through an overlaid 60x60 grid. When the player moved across the grid, cells would become gradually whiter each time the player entered them. At the end of each in-game day, the grid would be saved as a 60x60 image containing the path as a white path over a black background. An example of these player paths can be seen in figure 3.9 on page 23.

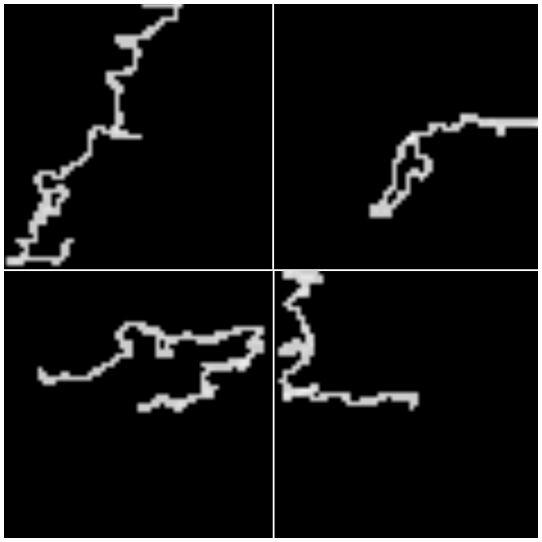

Figure 3.9: *An image showing four saved player paths, from the players in-game navigation.*

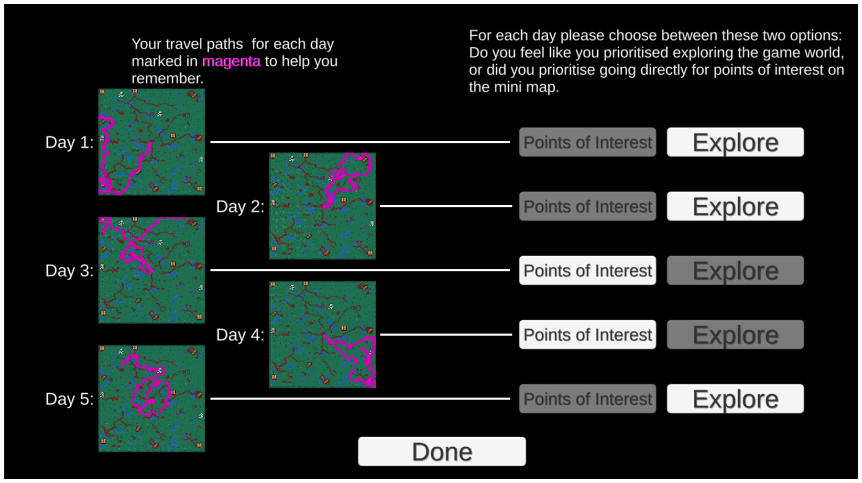

Figure 3.10: *An image showing the end screen of the game, asking players to categorise their navigation behaviour.*

In the end of each play session, each player was presented with their own 60x60 picture for each day, and asked to evaluate whether they were focused on points of interest or exploration. A focus on points of interest meant that the player had a goal in mind that they specifically moved towards. While an exploratory focus meant that the player moved around exploring based on what was visible within their viewport. See an image of the end screen in figure 3.10 on page 24.

3.3 Development Process

The development process of the game went through four iterations with playtests between each iteration, see figure 3.11 on page 25 for an illustration of this. The versions of the game were named as follows: Version 0.1 (MVP), Version 1, Version 2, and Version 3 (Final).

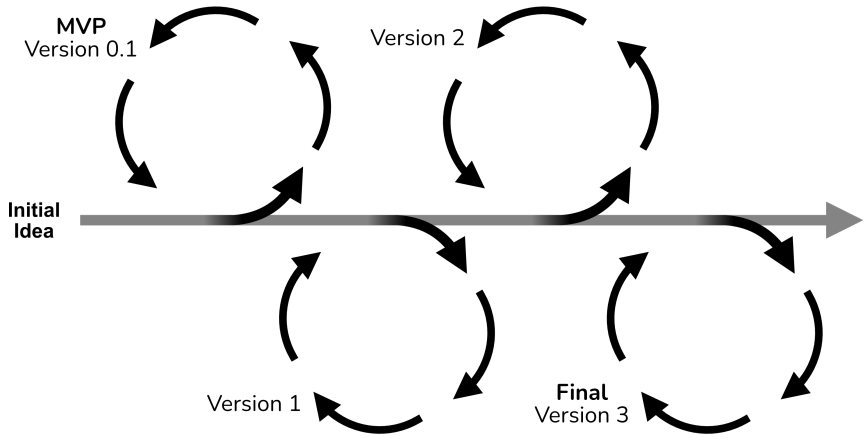

Figure 3.11: *Figure of the iterative process used to develop the final version of the game.*

3.3.1 Version 0.1 (MVP)

The MVP version was kept very simple, and included mechanics mostly related to the Assault dynamic. This meant that you could move, aim, shoot, and kill enemies that would chase you. The amount of enemies would ramp up as the game went on, overwhelming the player in the end. Our goals for the MVP version were the following:

- Build a very simple version of the game, focusing mostly on the Assault dynamic.
- Learn the pipeline to ensure that the data collection worked.
- Train a machine learning model with the gathered data.
- Start evaluating which data to track from the gameplay.

The MVP was intended to test the pipeline of gathering data through the game. This ensured that the pipeline worked from game to data to machine learning. The main takeaway from the MVP version was:

- The pipeline worked in its most basic form.

After the pipeline had been tested, work on implementing the other dynamics and a more detailed level began.

3.3.2 Version 1

Version 1 became more complex and added elements from all 3 game dynamics. This meant that players could kill enemies, gather resources to upgrade the village, and read lore objects about the game world. Version 1 implemented most of the core aspects of the game. Our goals for version 1 were the following:

- Implement mechanics/dynamics that support the Assault, Manage, and Journey types from the CGD model.
  - **Manage:** Collecting resources to upgrade the village.
  - **Journey:** Finding and interacting with different lore objects to expand the backstory of the world.
  - **Assault:** Killing enemies was kept from the MVP version, however the enemies were placed in the level, and the amount did not ramp up.
- Implement a day system that would segment the game into separate days of about 2 minutes each.
- Implement the guard-character to provide context and objectives to the player.
- Implement the CGD questionnaire into the game.
- Add tracking support for the added game functionality.

Version 1 had the core functionality of the full game. Therefore, we needed to test if the game worked as intended. To do this we tested the game using real players, from this we learned:

- The players focused too much on the shooting aspect from the assault dynamic.
- The players interacted with whatever appeared on screen and not their preferred game dynamic
- The data from the participants was not very varied.

The main takeaways from these tests were that the Manage and Journey dynamics were not engaging enough, compared to the assault dynamic.

3.3.3 Version 2

Version 2 mainly iterated on the main functionality and on making sure all dynamics seemed equally interesting. To make sure that the players knew about all aspects of the game a map was implemented. This showed all the major enemies, resources and lore objects in the game. Furthermore, we added visual effects (VFX) and audio effects (SFX) as feedback to the game. The game was also extended to the full length of five days. We also changed the village so it no longer upgraded automatically between days. Instead the player would need to manually interact with the village to upgrade it. This was implemented to provide the Manage dynamic some agency and motivation in the narrative usefulness of gathering resources. Our goals for version 2 were the following:

- Implement the full length game, five days of two minutes each.
- Make it obvious to the player that all the game dynamics would help the village.
- Implement SFX and VFX to increase general gameplay satisfaction.
- Implement an in game map to show the players they could interact with all three dynamics
- Make upgrading the village a more conscious action, by notifying the player when an upgrade was available and letting them do it themselves.
- Gather data on the player's navigation behaviour.

Version 2 solved most of the issues found during the playtesting of version 1. Playtesters told us that the interactions they had with the game were much more intentional. However, players still felt that their actions in the game were not acknowledged in the game's narrative. Players found the resources hard to spot and experienced other minor gameplay issues. From version 2 we learned:

- The narrative did not acknowledge the player helping the villagers.
- The resources still lacked feedback and were sometimes hard to see on screen.
- It was not obvious when a dialog box was the player character's.

Version 2 was mostly fine. It was a big improvement over version 1 and most of the changes needed for version 3 were quality of life improvements.

3.3.4 Version 3 (Final)

The biggest change to Version 3 was adding the navigation behaviour end screen at the end of the game, see section 3.2.2 on page 23. The gameplay changes for Version 3 were minor and therefore our goals were very specific:

- Add a screen after the game had ended which let the player classify their navigation behaviour.
- Add a response to the guard dialogue based on the players previous interactions.
- Add more visual feedback to the resources to make them easier to see.
- Add the player avatar to the dialog box, to symbolize that the player is speaking.

Chapter 4

Machine Learning Design

This chapter will focus on the design of the machine learning used to identify player types and navigation behaviour. It will outline the data, methods, and structure of the process. It will also present how certain aspects were iterated on. For a breakdown of the structure of the machine learning classifiers, see figure 4.1 on page 29.

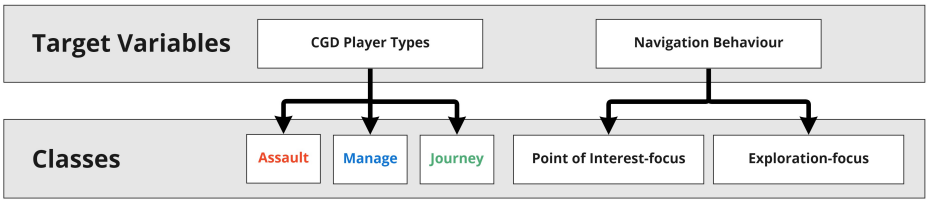

Figure 4.1: *Figure showing how the target variables relate to the different classes explored through the Machine Learning Design chapter.*

4.1 The Data

The data was saved in a csv-file which had 47 columns of data. Of these columns: 37 were features gathered for machine learning, four were demographic data, two columns were the labels we used as classes, three were the means calculated from the Core Game Dynamics (CGD) questionnaire, and the last was a unique ID for each participant. Each row of the data then represented a sample. Each participant provided multiple samples of data. For the primary experiment they provided five samples. One for each in-game day. The per-second data would have approximately 600 data samples per participant. One for each second of gameplay. An example of the data structure can be seen in table 4.1 on page 30.

Dynamic	Means	Features	Demographics	Participant
Journey	m1, m2, m3	f1 ... f37	d1, d2, d3	1
...	...	...	...	...
Journey	m1, m2, m3	f1 ... f37	d1, d2, d3	1
Assault	m1, m2, m3	f1 ... f37	d1, d2, d3	2
...	...	...	...	...
Assault	m1, m2, m3	f1 ... f37	d1, d2, d3	2
Manage	m1, m2, m3	f1 ... f37	d1, d2, d3	3
...	...	...	...	...
Manage	m1, m2, m3	f1 ... f37	d1, d2, d3	3

Table 4.1: *A condensed example of how the csv-files used to store our ML data looked.*

4.2 Preprocessing

Before the gathered data could be fed to the machine learning algorithms it was necessary to first preprocess the data. This both ensured that the data would match the algorithms' expected inputs and maximized their chances of finding meaningful patterns in the data.

4.2.1 Splitting Data

The first step of processing the data was to consider how to prune the data to test for different conjectures. This meant we needed to identify variables which might suggest different distributions in the data. We could then split the data into different files depending on these variables. An example of this was the in-game days. We found it likely that players would behave differently on different days in the game. Therefore, we split each in-game day into a separate file to be classified on its own. A more in-depth explanation of how this was done for each conjecture can be found in chapter 5.

4.2.2 Cleaning Data

The second step was to simply clean the data. This meant the data needed to be separated into features and meta data. The meta data being the data used to classify against, and which could not directly be derived from gameplay. This included player types, age, gender, navigation behaviour, and playtime. It would also always include the feature that was being classified against. An example of this happening would be if we were classifying the in-game day. Lastly, it would remove constant features i.e. features which had zero variance.

For the per-second data a running average was calculated for windows between 10 and 120 samples.

4.2.3 Deriving Features

The third step was to derive new features from existing features. These derived features were primarily ratios and aggregates of other features like the ratio between enemies seen and enemies killed, or the total number of enemies killed instead of normal enemies and major enemies separately. A complete list of the derived features can be seen below:

- Time Per Major Lore Object
- Time Per Lore Object
- Lore Seen vs. Interacted With
- Lore Close vs. Interacted With
- Resources Seen vs. Interacted With
- Resources Close vs. Interacted With
- Enemies Seen vs. Enemies Killed
- Enemies Close vs. Killed
- Time Per Map-Interaction
- Total Kills
- Total Interactions
- Total Lore Seen
- Total Lore Close
- Total Lore Read
- Total Enemies Seen
- Total Enemies Close
- Total Resources
- Total Resources Seen
- Total Resources Close

4.2.4 Standardising

The fourth step was to standardise the data. This was a requirement for certain algorithms and served to equalize the importance of features.

4.2.5 Dimensionality Reduction

The fifth step was dimensionality reduction. This was done to avoid what is known as the curse of dimensionality. Increasing the dimensions of the feature space reduces the accuracy of the classifier. To avoid this while retaining as much information from the data as possible, feature selection and feature extraction were used.
Feature selection simply selects the "best" features in accordance with a statistic like chi-square or the mutual information classifier we used.
Feature extraction instead attempts to extract new features from the original features by identifying the commonalities in features. For this we used principal component analysis (PCA).

4.2.6 Test / Train Splits

The sixth step was to split the data into training and testing sets. For our data it was important that different data points from the same participant did not appear in both the test and training sets.
We also used cross validation to better validate our classifiers. We generally only used two folds due to a lack of data to fill more folds.

4.3 Classification

The seventh step was the actual classification of our data. For this project we made use of the following classifers: K Nearest Neighbour (KNN), Multi layer perceptron or Neural Network (NN), One versus Rest using k nearest neighbour (OvR), and Support Vector Machine (SVM). We chose these classifiers in attempt to use classifiers with varying approaches to establishing their decision boundaries.

We chose the hyper parameters of the different classifiers by trying different ones and settling on the ones with best performance across different conjectures. The final parameters were as follows:

- KNN used the five nearest neighbors.
- NN used three hidden layers with 20, 10, and 5 neurons.
- OvR used KNN, also with five nearest neighbors.
- SVM used a radial basis function as its kernel.

4.4 Feature Dimensionality

To identify the best dimensionality of features for our classifiers we used feature selection. This sorted all of the features and then added them from best to worst while training the classifiers after each feature was added. We then compared the performance of each classifier across all feature dimensions to find the best dimensionality, see section 5.2.3 on page 51.

Chapter 5

Evaluation

This chapter will go through the procedure for testing the game. Furthermore, this chapter will also go through the conjectures and results, and further analyse the data gathered.

5.1 Experiment Design

This section will describe preliminary testing, pilot, data gathering, and how all of this is connected to the machine learning.

5.1.1 Preliminary Testing and Data Gathering

To get data, we needed participants to play the game. These play sessions were tested to ensure a solid procedure from the point of the participant sitting down until the data reached the classifier.

Pilot

To test the procedure, the game, and the data gathering we ran a pilot study on each version of the game. Through pilot testing, we made sure the pipeline was working as intended. Furthermore, this pilot study ensured that the game had the content and balance that we wanted. About 30 participants took part in this part of the project. This ensured that the choices in game design and data gathering were informed.

5.1.2 Data Gathering

Along with the gameplay behaviour data, demographic and miscellaneous data were also saved. This was data such as participant ID, gender, time spent playing video games weekly, and whether they had played earlier versions of the game.

A condensed version of the Core Game Dynamics (CGD) questionnaire was implemented into the game application for players to fill before starting the game. The condensed version only contained questions pertaining to the Assault, Manage, and Journey categories. The result of the CGD questionnaire was used as the classes for the machine learning classifier and was matched with the player's in-game behaviour data.

As described in the design chapter we had an end screen in the game where the participants had to categorize if their navigation behaviour was focused on exploration or points of interest. A screenshot of this end screen can be seen in figure 3.10 on page 24. For the final data gathering we had 102 participants play the game. Since we utilized machine learning, it was desired to have as many data points as possible. However, due to the limited resources available we were satisfied with 102 participants. Which gave us 510 data points in total.

5.1.3 Participant Demographics

We recruited 102 (82 male, 18 female, 2 Other, age range 20-31 years, average age 24.12 (sd = 2.2)) participants as volunteers. They all came from the AAU Create building. The range of weekly playtime was 0-70, with an average of 13.3 (sd = 13.6).

5.1.4 Procedure

This section will describe the procedure we followed during the data gathering process. For the data gathering we recruited participants around the university. The participation itself could be both online and offline, since most of the experiment could be completed autonomously.

When recruiting participants, we asked students at Create if they had 15 minutes to play a game and answer some questions. When the participant arrived at the play session, we would disinfect the PC (see section 5.1.5), other surfaces they might touch and we would offer hand sanitizer. The participants were then asked to read the consent form with information on how we treated and collected data.

The participants would then start the game. Here they would have to input their demographic information; age, average weekly playtime and gender. The application would then show the "Assault", "Manage", and "Journey" questions from the "Core Game Dynamics" questionnaire [1]. After completing the questionnaire, the game would automatically present the participant with the tutorial screen, with information on controls and objectives. Before the game started participants would be informed that we were testing the game, and not their performance. Furthermore, we would inform them of the fact that the game had sounds, and that the participants

would be allowed to ask questions. Then the participants would play the game, and answer the questions at the end screen. Then the data gathering ended, and we offered them a cookie for their participation.

5.1.5 Technical Setup

For the in person evaluation the following technical setup was used:

- **Computer:** Acer Aspire A715-71G
- **Headset:** Hyper X Cloud II
- **Mouse:** HP Office Mouse
- **Game recording:** Windows 10 built-in screen recorder

5.2 Conjectures & Results

We focused on two target variables for our classifiers: Player types and navigation behaviour. We had multiple conjectures for each of these and this section will present each of these conjectures together with their results. The conjectures will be presented in the following format: *"Conjecture X: Target variable | Data Split"*.

Accuracy and Balanced Accuracy

To evaluate the performance of the classifiers we used both accuracy and balanced accuracy. Accuracy is a simple measure of the ratio between correctly predicted samples and total samples. Balanced accuracy is similarly a ratio between samples and correct predictions but each sample is weighted in accordance with the inverse prevalence of its true class.

The requirement for this project was for the classifiers to perform better than random selection. Considering the low risk environment of games a wrong classification would not be a major issue and a larger tendency towards preferred gameplay would be good.

5.2.1 Player Type Classification

This section will present the "conjectures" and results we found in regards to the machine learning classifier performance when identifying player types. During the training of the classifier we ended up testing different conjectures about the player behaviour and the data. The following sections will show the results from these conjectures, and section 5.2.3 on page 51 will outline if our conjectures were correct.

Conjecture A: Player Types | General

The participant's player type can be accurately classified based on the data gathered from in-game interactions during each in-game day.

Results:
The results of this conjecture can be seen in table 5.1 on page 39. A graph showing the effects of the feature dimensionality can be seen in figure 5.1 on page 39. Finally, the confusion matrices for each of the four classifiers can be seen in table 5.2 on page 40.

This conjecture was confirmed with an accuracy of 46% and a balanced accuracy of 39%. Making this classifier able to predict a player's type better than random chance at 33%. It can also be seen in the confusion matrices that the classifier has a

preference for Journey, which is expected since 50% of the data is from the Journey class. Furthermore, it seems that a lower dimensionality performs slightly better than higher dimensionalities.

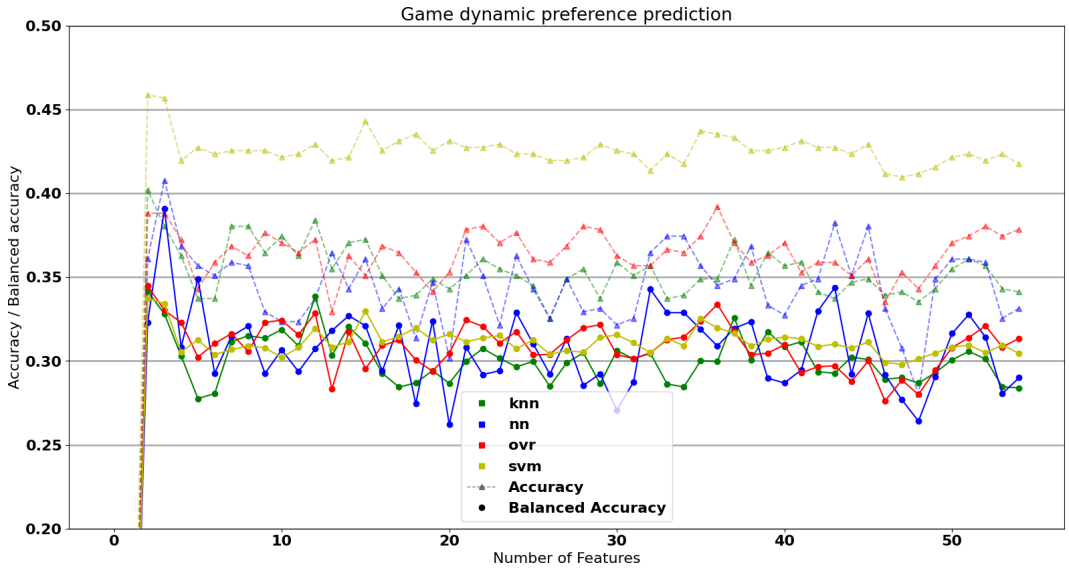

Figure 5.1: *Result of Conjecture A: Graph of the dimensionality test, showing how the accuracy and balanced accuracy of each classifier is affected by the number of features used.*

Classifier	Accuracy	Balanced Accuracy	Dimensionality
KNN	40%	34%	2
NN	41%	39%	3
OvR	39%	34%	2
SVM	46%	34%	2

Table 5.1: *Result of Conjecture A: The best performing dimensionality of each classifier. Chosen based on their balanced accuracy.*

KNN	Actual Assault	Actual Journey	Actual Manage
Predicted Assault	0.24	0.24	0.21
Predicted Journey	0.64	0.64	0.66
Predicted Manage	0.12	0.12	0.13
NN			
Predicted Assault	0.24	0.25	0.27
Predicted Journey	0.62	0.59	0.61
Predicted Manage	0.14	0.16	0.13
OvR			
Predicted Assault	0.13	0.15	0.13
Predicted Journey	0.56	0.55	0.56
Predicted Manage	0.31	0.30	0.33
SVM			
Predicted Assault	0.09	0.11	0.11
Predicted Journey	0.91	0.89	0.88
Predicted Manage	0.00	0.00	0.00

Table 5.2: *Result of Conjecture A: Confusion matrices for the classifiers shown in table 5.1, on page 39.*

Conjecture B: Player Types | Days

The participants might play differently across different in-game days making it easier to identify player types on certain days.

To test this conjecture we trained the classifier using two different data splits. In one of the splits the data was split into five different files. One for each day. Each file would then contain one sample from each participant with the data from all other days removed. An alternative splitting method was also used. This one gathered the days accumulatively. So the data for the last day would be the sum of the data from all five days.

Results:
The results of the 'per day' data split can be seen in figure 5.3 on page 41. The results of the accumulative data split can be seen in table 5.4 on page 42.

The results confirmed the conjecture with an accuracy of 49% and a balanced accuracy of 48% for both splits. Making this classifier able to predict a player's preferred game dynamic on specific in-game days better than random chance at 33%. The overall accuracy was higher when the data was spilt into specific days compared with the non-split data. With an increase of 3% accuracy and 9% balanced accuracy. The accumulative training approach led to a slight increase in accuracy of 2%, and no change in balanced accuracy.

Day	Classifier	Accuracy	Balanced Accuracy	Dimensionality
1	NN	48%	48%	3
2	KNN	42%	41%	5
3	NN	49%	47%	17
4	NN	46%	44%	31
5	NN	46%	48%	49

Table 5.3: *Result of Conjecture B: The best performing classifier for each separate day. Chosen based on their balanced accuracy.*

Accumulative Day	Classifier	Accuracy	Balanced Accuracy	Dimensionality
1	NN	48%	48%	3
2	KNN	49%	45%	4
3	NN	45%	44%	19
4	OvR	51%	46%	4
5	NN	45%	42%	7

Table 5.4: *Result of Conjecture B: The best performing classifier on accumulative days. Chosen based on their balanced accuracy.*

Conjecture C: Player Types | Mean Differences

Participants with larger mean difference between their primary player type and secondary player type play more in accordance with their preferred game dynamic.

To test this conjecture we had to identify the participants with a high mean difference, see section 1.1 on page 3 for examples of this. A number of mean difference thresholds between 0.1 and 1 were used for this. Participants with a separation smaller than the threshold were removed from the data.

Results

The results for this conjecture can be found in table 5.5 on page 43.

This conjecture was confirmed with an accuracy of 71% and a balanced accuracy of 48%. Making this classifier able to predict a player's preferred game dynamic when thresholding by mean difference better than random chance at 33%. The higher the threshold the more accurate the classifier became. The classifier performed better than the general classifier with an increase of 30% accuracy and 9% balanced accuracy.

Mean Dif	Classifier	Accuracy	Balanced Accuracy	Dimensionality
0	NN	41%	39%	3
0.1	NN	46%	39%	3
0.25	NN	44%	41%	9
0.5	NN	55%	46%	17
0.75	NN	53%	41%	21
1	OvR	71%	48%	3

Table 5.5: *Result of Conjecture C: The best performing classifier for each mean difference threshold. Chosen based on their balanced accuracy.*

Conjecture D: Player Types | Playtime

The participants have different behaviour depending on playtime, therefore the classifier may perform differently based on weekly playtime.

For playtime the data was split into two groups depending on a threshold. A group of participants with more playtime than the threshold and a group with less. This was done for multiple thresholds between 0 and 9 hours of play time.

Results
The results for this conjecture can be seen in table 5.6 on page 44.

From the data we gathered it seemed as if playtime had no real effect on the performance of the classifier. The differences we saw, seemed random.

Playtime	Classifier	Accuracy	Balanced Accuracy	Dimensionality
> 0	NN	45%	40%	13
$<= 0$	KNN	54%	44%	19
> 3	NN	47%	46%	8
$<= 3$	NN	40%	40%	13
> 6	OvR	51%	40%	2
$<= 6$	KNN	48%	48%	5
> 9	NN	43%	41%	19
$<= 9$	NN	41%	42%	29

Table 5.6: *Result of Conjecture D: The best performing classifier for different amounts of playtime. Chosen based on their balanced accuracy.*

Conjecture E: Player Types | Data Window size

The size of the data window used to calculate a running average had an effect on the classifier performance.

In addition to gathering data on in-game behaviour for each of the in-game days we also gathered the second to second data.

Results

The results of this conjecture can be seen in table 5.7 on page 45.

This setup allowed us to look at the moment to moment probability that a player belonged in a certain player type, a graph of these predictions from the best performing classifier and window can be seen in figure 5.2 on page 46.

Classifier	Window Size in Seconds	Accuracy	Balanced Accuracy	Dimentionality	Participant
OVR	10	33%	33%	3	59
OVR	20	39%	39%	3	59
OVR	30	28%	28%	3	59
OVR	40	32%	32%	3	59
OVR	50	32%	32%	3	59
OVR	60	36%	36%	3	59
OVR	70	29%	29%	3	59
OVR	80	28%	28%	3	59
Neigh	90	37%	37%	3	59
OVR	90	37%	37%	3	59
OVR	100	41%	41%	3	59
OVR	110	43%	43%	3	59
Neigh	119	66%	66%	3	59
OVR	119	66%	66%	3	59

Table 5.7: *Result of Conjecture E: The best performing classifier for each window size. Chosen based on their accuracy.*

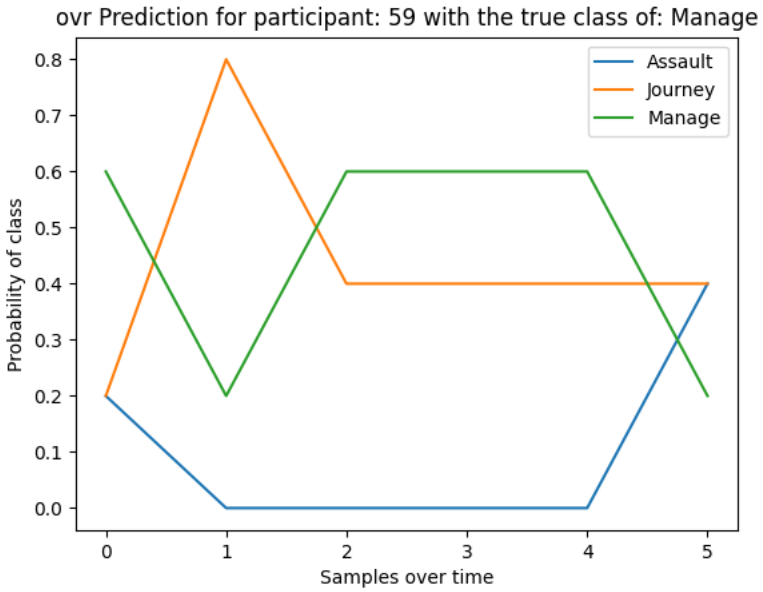

Figure 5.2: *A graph of the predictions from the best performing seconds classifier from table 5.7, on page 45.*

This assumption was correct with the highest accuracy of 66% and a balanced accuracy of 66%. Making this classifier able to predict a player's preferred game dynamic when using second to second data better than random chance at 33% in some cases. The larger the window size the more accurate the classifier becomes. The classifier performed better than the general classifier with an increase of 20% accuracy and 27% balanced accuracy.

However, it is important to note that is has only been run for a single participant, and running it for a larger test set might change the accuracy results. We only used the data from one participant, since the intention of this classifier was to track how one participant might change over a play session.

5.2.2 Navigation Behaviour

This section will present the "conjectures" and results we found in regards to the machine learning classifier performance when identifying player navigation behavior. During the training of the classifier we ended up testing different conjectures about the player behaviour and the data. The following sections will show the results from these conjectures, and section 5.2.3 on page 51 will outline if our conjectures were correct.

Conjecture F: Navigation Behaviour | General

It is possible to identify the participant navigation behavior on any given in-game day.

Results

The results of this conjecture can be seen in table 5.8 on page 48. A graph of the effects of the feature dimensionality can be seen in figure 5.3 on page 47. Finally, the confusion matrices for each of the four classifiers can be seen in table 5.9 on page 48.

This conjecture was confirmed with an accuracy of 74% and a balanced accuracy of 72%. Making this classifier able to predict a players current navigation behaviour better than random chance at 50%.

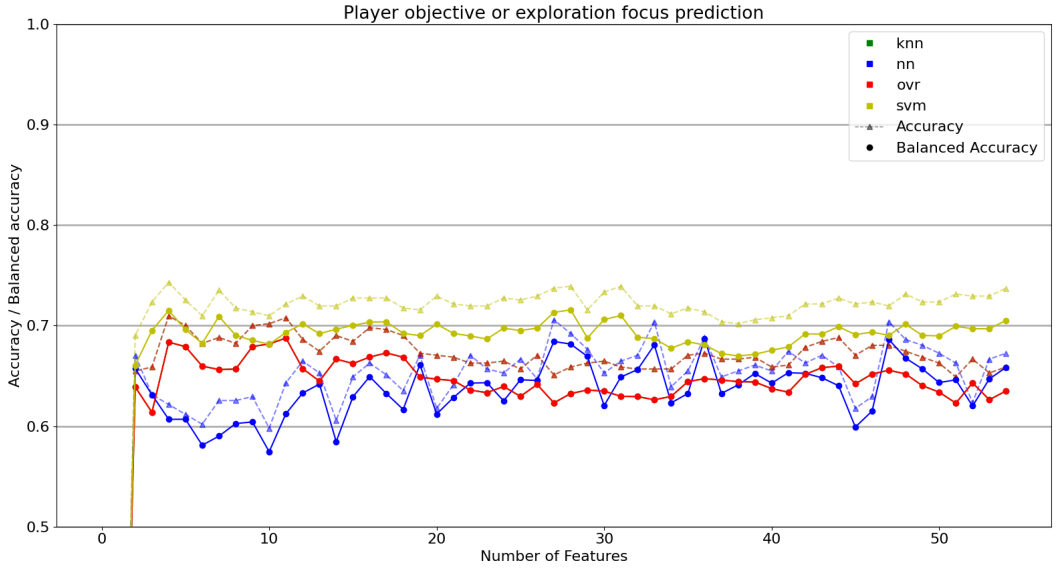

Figure 5.3: *Result of Conjecture F: Graph of the dimensionality test, showing how the accuracy and balanced accuracy of each classifier is affected by the number of features used.*

Classifier	Accuracy	Balanced Accuracy	Dimensionality
KNN	71%	67%	11
NN	69%	69%	36
OvR	71%	69%	11
SVM	74%	72%	28

Table 5.8: *Result of Conjecture F: The best performing dimensionality of each classifier. Chosen based on their balanced accuracy.*

KNN	Actual Objective	Actual Exploration
Predicted Objective	0.78	0.41
Predicted Exploration	0.22	0.59
NN		
Predicted Objective	0.70	0.33
Predicted Exploration	0.30	0.67
OvR		
Predicted Objective	0.78	0.41
Predicted Exploration	0.22	0.59
SVM		
Predicted Objective	0.82	0.39
Predicted Exploration	0.18	0.61

Table 5.9: *Result of Conjecture A: Confusion matrices for the classifiers shown in table 5.8, on page 48*

Conjecture G: Navigation Behaviour | Days

The participants' navigation behaviour differ across different in-game days affecting the performance of the classifier.

The data split for this conjecture was accomplished in the same manner as the split for Conjecture B in section 5.2.1 on page 41.

Results
See results in table 5.10 on page 49.

This conjecture was not correct with an accuracy of 78% and a balanced accuracy of 80%. Making this classifier able to predict a player's navigation behaviour better than random chance at 50%. However, splitting the data into days have not had any meaningful impact on the overall performance of the classifier.

Day	Classifier	Accuracy	Balanced Accuracy	Dimensionality
1	SVM	68%	68%	6
2	OvR	70%	73%	6
3	NN	77%	74%	8
4	NN	78%	80%	2
5	OvR	73%	68%	19

Table 5.10: *Result of Conjecture G: The best performing classifier for each separate day. Chosen based on their balanced accuracy.*

Conjecture H: Navigation Behaviour | Playtime

The participants would behave differently across days causing the performance of the classifier to fall. Separating the participants by playtime could improve the performance.

The data split for this conjecture was accomplished in the same manner as the split for Conjecture D: Player Types | Playtime in section 5.2.1.

Results
The results for this conjecture can be seen in table 5.11 on page 50.

This assumption was not correct with an accuracy of 76% and a balanced accuracy of 74%. Making this classifier able to predict a player's current interaction style between exploration and objective focus better than random chance at 50%. However, splitting the data by playtime did not result in better performance of the classifier.

Playtime	Classifier	Accuracy	Balanced Accuracy	Dimensionality
>0	SVM	76%	73%	4
<= 0	NN	59%	66%	50
>3	SVM	76%	73%	6
<= 3	NN	69%	70%	48
>6	NN	76%	73%	20
<= 6	SVM	70%	68%	35
>9	SVM	76%	74%	10
<= 9	OvR	71%	71%	7

Table 5.11: *Result of Conjecture H: The best performing classifier for different amounts of playtime. Chosen based on their balanced accuracy.*

5.2.3 Result & Conjecture Summary

This section will summarise some of the most interesting results of the evaluation.

Conjecture B through D concerned splitting the data in different ways to achieve different results. From these results it can be seen that splitting the data has generally resulted in an improvement of classifier performances. Especially splitting the data using mean differences had a big impact on classifier performance.

Conjecture E was that the size of the window used for classification on the second to second data would have an effect on the performance of the classifier. This turned out to be true. Furthermore, we can see from the data that the larger the window the higher accuracy we achieve.

As expected the accuracy of our results is generally higher than the balanced accuracy since our data is unbalanced. This can be seen more clearly in Conjecture A where the SVM classifier predicted Manage 0 times, resulting in a relatively high accuracy compared to the other classifiers, but with a similar balanced accuracy.

As a general comment on our data we found that our dimensionality checks showed a general trend towards lower dimensions with 24 out of 45 trained classifiers being most performant with dimensions at or below 10, see figure 5.4 on page 51 for a frequency diagram of the dimensionality.

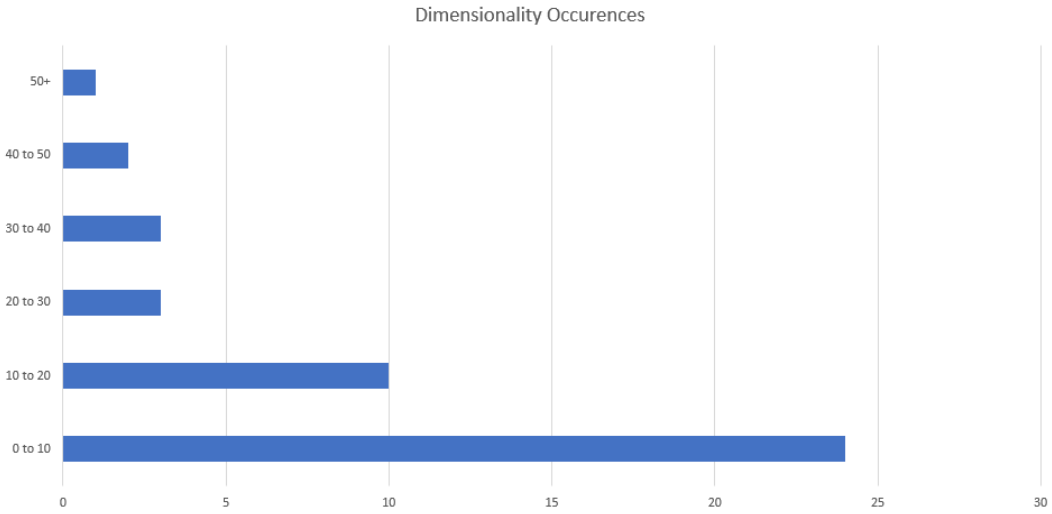

Figure 5.4: *Frequency of different dimensionalities across the different conjectures. Conjecture E has been excluded as we did not test for the best dimensionality for it.*

For each of the conjectures we tested we would record how our feature selection rated all of the features. This data was then used to evaluate which features were "better". This was done separately for the two target variables.

The top three features for classifying CGD player types were:

1 Resources seen vs. interacted with

2 Enemies seen vs. enemies killed

3 Time standing still

The top three features for classifying player navigation behaviour:

1 Opened map

2 Map time

3 Major lore reading time

Chapter 6

Discussion

The goal of this project was to use machine learning to identify the player type of players using indirect feedback. This was evaluated in Conjecture A of the results. We found that the classifier could predict player types with a 46% accuracy and 39% balanced accuracy. This meant we had an accuracy better than random for the three classes. The most accurate class was Journey, but this class was also the one with the most false positives. This makes sense since about half of the data was of the Journey type.

In Conjecture F we attempted to predict the players navigation behaviour during each of the in-game days. We were able to achieve a 74% accuracy with a 72% balanced accuracy. With two classes that was better than random. The results showed a higher accuracy for objectives. This was expected since more samples were focused on objectives than exploration with 314 instances of objectives and 196 instances of exploration.

When evaluating Conjecture E our results show that using a larger window to calculate a running average of features achieved a higher accuracy. This is in line with what Valls-Vargas et al. found in their research [22]. This might be because smaller windows were able to see less of a player's behaviour pattern. This is because all player behaviour patterns look identical in small enough windows. The data seems to support our design decision to segment the game into days since these were equivalent to larger windows.

We had a number of conjectures in regards to how splitting the data based on different measures could improve the performance of the classifiers. It seemed there was a difference in player behaviour across: days, playtime, and mean differences between player types since we achieved better performance by segmenting them. This could point towards the need for a game to have multiple classifiers for different groups of

players. Such as player demographics or players who have more or less playtime in your game.

It would also have been valuable to investigate how different combinations of these splits would have affected the classifiers' performance. Unfortunately, creating these splits involved removing parts of the data and doing so across multiple measures would have required more data than we had gathered. Otherwise we would have been left with too little data to train our classifiers.

Conjecture C, that participants would play more in line with their player type if their mean difference was higher, seems to hold true. At a mean difference of one it increased accuracy from 41% to 71% and balanced accuracy from 39% to 48%. Or proportionally by 70% and 20% respectively. This indicated a difference between participants whose highest scoring player type categories have more separation from the other categories when compared to the participants who do not. This could also mean that there might be a section of the player base that are fine with the game as is. In other words, players with smaller mean differences might simply enjoy the different categories of dynamics equally. They would not need any adaption to the game mechanics as opposed to those with high mean differences. This could mean that players with high mean differences are more likely to enjoy a game that is adapted to better satisfy their player type.

A confounding variable could be that rather than the actual difference between means. The improved accuracy could be caused by each participant simply having a higher mean in their respective player type. This could indicate that they responded positively to more of the dynamics related to this player type. This would increase the chance that one of their preferred dynamics within the player type was implemented in our game. Meaning they were not easier to classify because they preferred the player type more, but because they preferred the specific dynamics implemented in our game. In this case the improvement in performance could also be achieved by implementing more of the dynamics for each of the player types to cover a wider span of player preferences. If this is the case it could be that a high mean difference was not important and that a good coverage of dynamics is.

Our game had a rather poor coverage of the dynamics within each player type. Therefore, players who preferred e.g. the Assault dynamics, might not have liked the specific aspects of Assault that we implemented. Meaning they preferred Assault dynamics in general, however they did not enjoy the Assault mechanics from our game. This could result in our classifier getting data from an Assault type that did not enjoy our particular Assault implementation, and therefore may have played more like one of the other player types.

One of our desires for this project was to develop a more generalisable method for identifying player types in games. Both in regards to classifying player types and designing for the classifier. It is important to note that our implementation only considers the Assault, Manage, and Journey player types. This means that for this type of system to be fully generalisable to other games, one will have to consider the other player types found in the Core Game Dynamics (CGD) model.

In regards to our process we believed that it was generalisable enough to be used in the development of other games. This was because we focused on the game dynamics and how these related to player type classification. If desired, this approach could also be adapted to be used in existing games. However, while we believed the approach to be generalisable, our classifier was not. Furthermore, we did not believe it was possible or at least reasonable to develop a general classifier which worked across all or a large proportion of games. This was because player behaviour was heavily affected by the game's presentation and it was therefore not possible to train the classifier on one game and then extrapolate that to another. Similarly, Drachen et al. argued that games as a media might be too varied for this to be possible. An example of this could be that using the same data features for a shooting game and a racing game is not feasible [16].

In regards to the generalisability of the features we found that two of the three best features ("Resources Seen vs. Interacted With" and "Enemies Seen vs. Enemies Killed") for classifying player types were only applicable to our game, further supporting the argument above. In contrast to classifying player types two out of the three best features ("Opened Map" and "Map Time") for classifying navigation behaviour were applicable to other games. This indicates that it is possible to identify features which are generalisable across games. However, doing so might require a simpler task like classifying navigation behaviour.

Considering the difficulty of developing a generalisable classifier it might be more useful to specialize the classifier more instead. As an example, the questionnaire used to identify player types, contained questions about dynamics which were not implemented in the game. The performance of the classifier might improve if the questionnaire was more tailored to the actual game implementation.

It could be relevant to use our approach to create a classifier, while also adapting a game based on the CGD player types. Furthermore, we also think it would be worthwhile to test this approach in the context of a larger commercial game. This would also allow testing the approach while using a wider spectrum of dynamics for each player type.

Chapter 7

Conclusion

This report evaluated the feasibility of using a machine learning classifier to predict player types using the Core Game Dynamics model. We developed a 2D adventure game based on the "Assault", "Manage", and "Journey" player types. We used a model-based approach and indirect feedback to develop the machine learning algorithm. The game dynamics, machine learning, and data collection were iteratively developed through playtests. After the data was collected, we started to train a number of different classifiers using supervised machine learning. The overall accuracy of the classifier was 48% when predicting a player's player type. However, when investigating different methods of splitting the data we achieved an accuracy of 71%. The best of these splits were those based on mean differences between player types. Furthermore, we think that creating a generalised machine learning classifier will not be feasible, since player behaviour is so heavily influenced by game design. However, we think it is very feasible to create a classifier that is translatable into player types.

Bibliography

[1] Jukka Vahlo, Johanna K Kaakinen, Suvi K. Holm, and Aki Koponen. Digital Game Dynamics Preferences and Player Types. *Journal of Computer-Mediated Communication*, 22(2):88–103, 03 2017.

[2] Raúl Lara-Cabrera and David Camacho. A taxonomy and state of the art revision on affective games. *Future Generation Computer Systems*, 92:516–525, 2019.

[3] Reis Simão, Luís P. Reis, and Nuno Lau. Game adaptation by using reinforcement learning over meta games. *Group Decision and Negotiation*, 30(2):321–340, 04 2021. Copyright - © Springer Nature B.V. 2020; Last updated - 2021-07-16.

[4] Richard Bartle. Hearts, clubs, diamonds, spades: Players who suit muds. `https://mud.co.uk/richard/hcds.htm`, 06 1996.

[5] Rita Orji, Julita Vassileva, and Regan L Mandryk. Modeling the efficacy of persuasive strategies for different gamer types in serious games for health. *User Modeling and User-Adapted Interaction*, 24(5):453–498, 2014.

[6] Sebastian Deterding, Miguel Sicart, Lennart Nacke, Kenton O'Hara, and Dan Dixon. Gamification: Using game design elements in non-gaming contexts. In *Extended Abstracts on Human Factors in Computing Systems*, volume 66, pages 2425–2428, 01 2011.

[7] Yee Nick. Motivations of play in mmorpgs. In *DiGRA '05 - Proceedings of the 2005 DiGRA International Conference: Changing Views: Worlds in Play*, 2005.

[8] Chris Bateman and Richard Boon. *21st Century Game Design (Game Development Series)*. Charles River Media, Inc., USA, 2005.

[9] Isabel Briggs Myers and Mary H. McCaulley. *MBTI Manual: A Guide to the Development and Use of the Myers-Briggs Type Indicator*. Consulting Psychologists Pres, Palo Alto, California, 1985.

[10] A. Marczewski. *Even Ninja Monkeys Like to Play: Gamification, Game Thinking and Motivational Design.* CreateSpace Independent Publishing Platform, 2015.

[11] Georgios N. Yannakakis, Pieter Spronck, Daniele Loiacono, and Elisabeth André. Player modeling. In *Artificial and Computational Intelligence in Games*, 2013.

[12] Danial Hooshyar, Moslem Yousefi, and Heuiseok Lim. Data-driven approaches to game player modeling: A systematic literature review. *ACM Comput. Surv.*, 50(6), jan 2018.

[13] Marlon Etheredge, R. Lopes, and Rafael Bidarra. A generic method for classification of player behavior. In *Association for the Advancement of Artificial Intelligence*, pages 2–8, 10 2013.

[14] Kostas Anagnostou and Manolis Maragoudakis. Data mining for player modeling in videogames. In *2009 13th Panhellenic Conference on Informatics*, pages 30–34, 2009.

[15] Bruno Almeida Odierna and Ismar Frango Silveira. Mmorpg player classification using game data mining and k-means. In *Advances in Information and Communication*, pages 560–579. Springer International Publishing, 2020.

[16] Anders Drachen, Alessandro Canossa, and Georgios N. Yannakakis. Player modeling using self-organization in tomb raider: Underworld. In *2009 IEEE Symposium on Computational Intelligence and Games*, pages 1–8, 2009.

[17] Brent Harrison and David Roberts. Using sequential observations to model and predict player behavior. *Proceedings of the 6th International Conference on the Foundations of Digital Games*, 06 2011.

[18] Seong Jae Lee, Yun-En Liu, and Zoran Popovic. Learning individual behavior in an educational game: A data-driven approach. In *EDM*, 2014.

[19] Alexis Fortin-Cote, Nicolas Beaudoin-Gagnon, Cindy Chamberland, Frédéric Desbiens, Ludovic Lefebvre, Jérémy Bergeron, Alexandre Campeau-Lecours, Sébastien Tremblay, and Philip Jackson. *FUNii: The Physio-Behavioural Adaptive Video Game*, pages 14–28. Springer International Publishing, 06 2019.

[20] Benjamin Cowley, Darryl Charles, Michaela Black, and Ray Hickey. Real-time rule-based classification of player types in computer games. *User Modeling and User-Adapted Interaction*, 23, 08 2012.

[21] Robin Hunicke, Marc Leblanc, and Robert Zubek. Mda: A formal approach to game design and game research. *AAAI Workshop - Technical Report*, 1, 2004.

[22] Josep Valls-Vargas, Santiago Ontañón, and Jichen Zhu. Exploring player trace segmentation for dynamic play style prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 11, pages 93–99. AAAI Press, 2015.