UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Sercan Altundaş

NPC AI System Based on Gameplay Recordings

Master's Thesis (30 ECTS)

Supervisor(s): Margus Luik, MSc

Tartu 2018

NPC AI System Based on Gameplay Recordings

Abstract:

A well optimized Non-Player Character (NPC) as an opponent or a teammate is a major part of the multiplayer games. Most of the game bots are built upon a rigid system with numbered decisions and animations. Experienced players can distinguish bots from human players and they can predict bot movements and strategies. This reduces the quality of the gameplay experience. Therefore, multiplayer game players favour playing against human players rather than NPCs. VR game market and VR gamers are still a small fraction of the game industry and multiplayer VR games suffer from loss of their player base if the game owners cannot find other players to play with. This study demonstrates the applicability of an Artificial Intelligence (AI) system based on gameplay recordings for a Virtual Reality (VR) First-person Shooter (FPS) game called Vrena. The subject game has an uncommon way of movement, in which the players use grappling hooks to navigate. To imitate VR players' movements and gestures an AI system is developed which uses gameplay recordings as navigation data. The system contains three major functionality. These functionalities are gameplay recording, data refinement, and navigation. The game environment is sliced into cubic sectors to reduce the number of positional states and gameplay is recorded by time intervals and actions. Produced game logs are segmented into log sections and these log sections are used for creating a look-up table. The lookup table is used for navigating the NPC agent and the decision mechanism followed a way similar to the state-action-reward concept. The success of the developed tool is tested via a survey, which provided substantial feedback for improving the system.

Keywords:

Non-player character, game bot, artificial intelligence, game logs, gameplay recordings, virtual reality, HTC Vive, Unity, FPS games

CERCS:

P170 Computer science, numerical analysis, systems, control

P176 Artificial intelligence

Mängu tegevuse lindistamisel põhinev tehisintellekti süsteem mitte-mängija tegelastele

Lühikokkuvõte:

Hästi optimeeritud mitte-mängija tegelased (MMT) on vastaste või meeskonna kaaslastena üheks peamiseks osaks mitme mängija mängudes. Enamus mängu robotid on ehitatud jäikade süsteemide peal, mis võimaldavad vaid loetud arvu otsuseid ja animatsioone. Kogenud mängijad suudavad eristada mängu roboteid inimmängijatest ning ette ennustada nende liigutusi ja strateegiaid. See alandab mängukogemuse kvaliteeti. Seetõttu, eelistavad mitme mängijaga mängude mängijad mängida pigem inimmängijate kui MMTde vastu. Virtuaal reaalsuse (VR) mängud ja VR mängijad on siiani veel väike osa mängutööstusest ja mitme mängija VR mängud kannatavad mängijabaasi kaotusest, kui mänguomanikud ei suuda leida teisi mängijaid kellega mängida. See uurimus demonstreerib mängulindistustel põhineva tehisintellekt (TI) süsteemi rakendatavust VR esimese isiku vaates tulistamismängule Vrena. Teema mäng kasutab ebatavalist liikumisesüsteemi, milles mängijad liiguvad otsiankrute abil. VR mängijate liigutuste imiteerimiseks loodi AI süsteem, mis kasutab mängulindistusi navigeerimisandmetena. Süsteem koosneb kolmest peamisest funktsionaalsusest. Need funktsionaalsused on mängutegevuse lindistamine, andmete töötlemine ja navigeerimine. Mängukeskond on tükeldatud kuubi kujulisteks sektoriteks, et vähendada erinevate asukohal põhinevate olekute arvu ning mängutegevus on lindistatud aja intervallide ja tegevuste põhjal. Loodud mängulogid on segmenteeritud logilõikudeks ning logilõikude abil on loodud otsingutabel. Otsingutabelit kasutatakse MMT agentide navigeerimiseks ning MMTde otsuste langetamise mehanism jäljendab olek-tegevus-tasu kontseptsiooni. Loodud töövahendi kvaliteeti hinnati uuringu põhjal, millest saadi märkimisväärset tagasisidet süsteemi täiustamiseks.

Võtmesõnad:

Mitte mängija tegelane, mängu robot, tehisintellekt, mängu logid, mängu tegevuse lindistused, virtuaalreaalsus, HTC Vive, Unity, esimese isiku vaates tulistamismängud

CERCS:

P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)
P176 Tehisintellekt

Table of Contents

1 Introduction

Video games have a long history of more than 50 years and artificial intelligence (AI) applications in video games nearly two decades [1]. AI applications of video games started with simple decision algorithms, scripted behaviors, and state machines. Today, AI developers focus on bringing better gameplay experience to users and try to create human-like non-player characters (NPC) for building immersive gameplay experiences. Especially for fast paced competitive games, NPCs have different levels of difficulties which make them too hard or too easy to play against. As they become predictable after a while, the game experience becomes monotonous. These approaches are unable to increase the quality of the experience above a certain level as the players easily predicted NPC actions [1]. One approach to overcome this problem is using gameplay recordings of human players to create human-like bots.

The topic of creating human-like NPCs requires new approaches when the subject is a Virtual Reality game. Despite being an old concept, Digital Virtual Reality finally reached the commercial success to be used at home since the advancement in the technology brought thinner displays, faster processors and smaller hardware in general. However, technology is still in development and the number of games to play with Virtual Reality peripherals are not quite as many as other PC games (less than 10% of games on Steam[1]). To make VR games more appealing for gamers, better NPCs with proper AI systems have become a necessity. Controls of an FPS game in VR is different than conventional mouse and keyboard controlled FPS games, and it gives players ability to move their hands freely. This difference also changes the answer to the "what a game bot can do?" question.

With this research, a tool for Unity game engine developed that records player's actions during a game session. This tool helps developers examine the game sessions, prune unnecessary data and sort the best outcomes. Lastly, the tool generates the data that is used by the navigation algorithm for creating human-like NPC agents. For achieving this goal, a lookup table used for storing sections of game logs as training data and an approach that is influenced by a state-action-reward combination of Reinforcement Learning (RL) techniques are used.

1.1 Scope and Limitations of Thesis

The testbed of this project was a Virtual Reality game of First Person Shooter genre. For this purpose, a log AI tool is created for recording, processing game logs and navigating the agents. This tool is limited and depended on this case study game's variables. In addition, HTC Vive VR set and Steam VR tools are used for working on this project.

The development of this research is made possible by the Computer Graphics and Virtual Reality Laboratory[2] at the University of Tartu which has VR equipment for students to work with a wide play area to play and test products.

The log AI tool is developed on Unity game engine using C# programming language since the case study game has been under development using this game engine. The version of the game engine used during the development of the tool was Unity version 2017.3.1.

[1] https://steamdb.info/instantsearch, category type of **Game** resulted in 23992 products and 2277 of these products were tagged **VR.** Last visited [05.19.2018].
[2] http://cgvr.cs.ut.ee

The scope of the research was creating an NPC AI system using game logs to make game bots in the case study game navigate and complete the basic movement. In the game, the player could not move as in conventional FPS games. Movement ability of the player is restricted to a few meters by walk, as much as the tracking area of VR set allows. Players used grappling hooks for pulling themselves to move around.

The log AI tool did not cover decision-making processes for item collection and fighting other opponents, yet possible solutions and implementation ideas are discussed in chapter 9.2, Future Work.

The log AI tool did the work of collecting and processing of the game logs. No third party tool is used for these operations. Collected game logs and lookup tables are stored in JSON format.

1.2 Problem Statement

Playing multiplayer games have many motivators such as progression, discovering the game world and the lore, creating a new persona, role-playing, having many different customizations and socializing. These motivators can be grouped into three sections as achievement component, immersion component and social component [2]. Namely, people choose to play multiplayer games to socialize, to be a part of teamwork, to challenge each other and build relationships.

Compared to all PC gamers[3], the population of VR players[4] is a small part of the whole. This fact creates challenges for providing some of the motivators to the players when the subject game is a multiplayer VR game. VR has yet to be a mainstream concept in video game scene where PCs and video game consoles have a bigger portion of the game market. Fortune Magazine mentioned [3] that a developer survey from Virtual Developers Conference in 2017 reveals that people are having problems adapting this technology because of motion sickness problems and lack of business applications available in VR market. Since the category of game applications is the biggest category by 78% in the VR market, many people have the impression that VR headsets are just toys. This causes slowdown on the adaptation of this technology.

These statements underline that creating a multiplayer game with socializing and team play aspects may not be preferable for developers as there are not many players exist compared to other game platforms. If a new multiplayer VR game cannot reach to a high number of players from launch, it is likely that early adopters will abandon the game and the game as a product will fail. And gamers who are aware that a VR game has a low number of online players may choose not to buy that game.

A solution to overcome this issue is using AI agents, also known as game bots to create competitive gameplay environment. Moreover, if these bots were to play like humans do, instead of acting like predictable robots, then players would get more satisfaction and enjoy the game.

The challenging and novel part of this research is creating an AI system for unconventional controls of the case study game Vrena. Since the game introduces a movement by using grappling hooks, it is not possible to implement regular FPS game bots into the game. To supply the players with an experience surrounded by human-like bots, it is needed to make

[3] https://www.statista.com/statistics/748044/number-video-gamers-world, 2.21 billion gamers in 2017
[4] https://www.statista.com/statistics/426469/active-virtual-reality-users-worldwide, 171 million users in 2018

them act as VR set users. One known way to achieve this ability is to use game logs, which is proven to be effective in some studies [4] [5] [6].

1.3 Research Questions

This research aimed to create a tool for the case study game and to develop a method to use gameplay recordings (logs) in this FPS VR game. The tool has the ability to record, store and examine gameplay sessions. Additionally, the tool used processed game logs to give NPCs movement ability.

In this regard, the research aimed to answer these questions:

RQ 1. *How can we implement a system that uses game logs for NPC movement and decision?*

Previous studies of using game logs for NPC movement and decision making should be examined for finding relevant methods. Which types of information are extracted from game logs and used for generating NPC agents?

RQ 2. *How can we create a configurable log recording and processing system, what measures should be taken into consideration while recording?*

What type of information should be recorded and what should be left out? How often logs should be saved and how should the data be structured? Possible ways of recording game logs should be looked for and the following question should be answered when a game session is to be recorded.

How can we make the recording part of this tool configurable? The developed tool should have the ability to aid its users to configure and debug the system and adapt to the changes in the logs if the logs are modified.

RQ 3. *How can we benefit from modern AI techniques?*

Since applications of artificial intelligence have been in game development scene for a long time, there are many ways to create bots for games. Existing methods should be examined and possible candidates should be selected for use.

After the recent achievements[5] of Machine Learning (ML), its applications have attracted attention to this field. Studies of game bots using ML and other AI techniques should be examined for answering this question.

1.4 Research Methods

To understand the problem better, play sessions were held to familiarize with the case study game. These play sessions useful for understanding its mechanics, controls, states, and environment. Additionally, these play sessions helped developing ideas about how to create similar player movement via an AI system.

[5] https://blog.statsbot.co/deep-learning-achievements-4c563e034257

Two Massive Open Online Courses (MOOC) were completed to learn more about AI and ML fields. One of these MOOC was Andrew Ng's Machine Learning course[6] from Stanford University and the other was Reinforcement Learning course[7] from Google DeepMind's David Silver. These courses were useful for understanding the application areas and techniques of ML and RL in general.

Since the case study game is being developed in the Unity game engine, the author started to learn the game engine in depth to be able to create the necessary tool to solve the problem. This learning task consisted of working on example projects, Unity's own programming tutorials, and custom tool design lectures[8]. In September 2017, Unity 3D announced their Machine Learning Agents[9] as an open source tool to enable users of the game engine to have an opportunity to easily implement RL agents in their games and train them. After this announcement, Unity organized an online programming challenge for their tool. To discover the abilities of this tool, the author worked on a simple Pong game project[10], which used main abilities of the given tool.

To get familiar with the game AI, human-like agent behavior and modern Computational Intelligence (CI) topics, online content is explored, books, websites, and technical documentation are examined.

The content of the research mainly gathered from the scholarly literature libraries such as SpringerLink, Google Scholar, and IEEE Xplore. The most relevant and valuable papers from the search results in all the libraries are selected.

The first focus of the search is to use the most relevant keywords "First Person Shooter" and "Artificial Intelligence", "Virtual Reality", "Game Logs" and "Game Play Recordings" along with abbreviations FPS, VR and AI. Since "HTC Vive" is the equipment for displaying and providing the experience to the player and "Unity" is the game engine in use these keywords had lesser importance in the search. Also, keywords of "Non-Player Characters" and "Game Bots" are used with different combinations since they are widely used synonyms.

In the field of Computer Science and Artificial Intelligence "First Person Shooter" and "Artificial Intelligence" resulted

- 59 research papers in Google Scholar
- 50 research papers in IEEExplore
- 121 research papers in SpringerLink

When "Game Logs" or "Game Play Recordings" applied to these search results, they narrowed the results down to

- 15 research papers in Google Scholar
- 2 research papers in IEEExplore
- 6 research papers in SpringerLink

[6] https://www.coursera.org/learn/machine-learning
[7] http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
[8] https://unity3d.com/learn/tutorials
[9] https://unity3d.com/machine-learning
[10] https://connect.unity.com/p/5a5fd5ae0909150019730a11

Many research papers collected from the results above and from various combinations of the keywords. These papers listed by relevance and their abstracts have been read to exclude the irrelevant papers.

During the development, additional research papers on different topics are also used. These papers were about the comparison of data storing formats and making research surveys which are also acquired from the same scholarly libraries. Even though these research papers were not directly connected to the main topic of the research, they provided additional information to improve the development and results.

Overall, this method resulted in total 20 research papers and books. These documents have helped learn about AI techniques in general, AI applications in game development, information extraction from game logs, behaviour mining from game logs, choosing right formats to save logs and conducting research surveys to collect user feedback.

2 Background

In this section background information about the thesis is provided. As this research aims to solve an AI problem of the case study game that is developed on Unity game engine, the game engine's features and ability to help this research are discussed.

Secondly, the VR device HTC Vive is introduced, which is used in both production and testing of the game. HTC Vive's abilities, technical specifications, and requirements are explained.

Thirdly, the case study game Vrena is explained in detail. The controls, environments, rules and gameplay mechanics of the game is discussed. The challenges of working on an NPC AI system for a multiplayer VR game is discussed.

Lastly, general artificial intelligence and its applications in game industry and especially the evolution of NPC AI in games is discussed.

2.1 Unity 3D Game Engine

Unity is a game engine which is widely used in video game development and movie production. It has reached 770 million[11] gamers with its multiplatform support. With this support, developers are able to develop games from one game engine to many gaming environments such as Windows, Mac or Linux, Android, IOS, Windows Phone, PS4, Xbox One, PS Vita, WebGL, and Facebook[12].

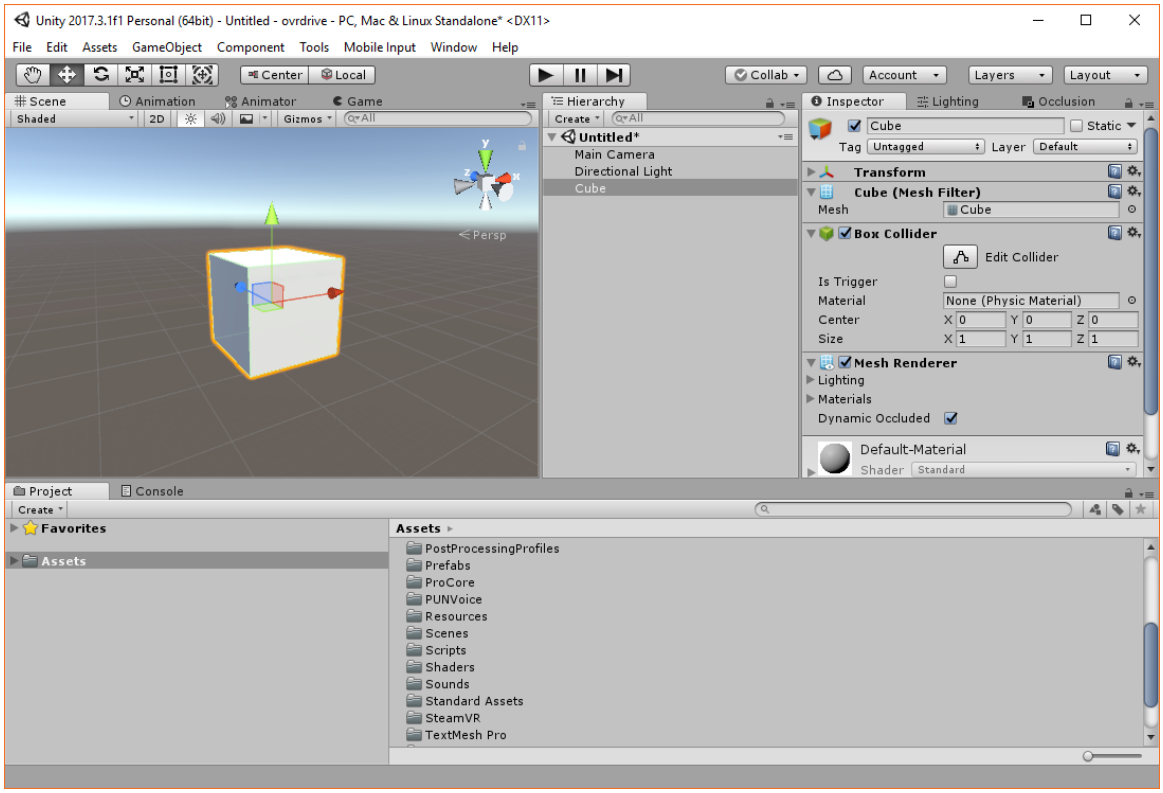

Figure 1: *The User interface of Unity 3D game engine.*

[11] https://unity3d.com/public-relations
[12] https://unity3d.com/unity

The game engine mainly uses C# language for game programming purposes. Previously, it also had support for Boo and UnityScript (a JavaScript-like scripting language) but they decide to remove the support for those and deprecated them due to the low amount of usage amongst developers and new .NET dependent features [7].

As the game engine adapts OOP practices on the programming side, it also uses an Entity-Component architecture. In this concept Game Objects[13] are Entities and Components are behaviour scripts inherited from Unity's Component[14] class that can be attached to the Game Objects. This ability gives the possibility to have the AI tool ported to other similar games and create AI agents with just simple tweaks and changes.

Unity provides 3D view window where the user can drag and drop game assets to a game scene which can be seen in *Figure 1*. Game engine houses its own animator, resource management tools, network system, shader editor and many other tools to simplify the development of a game.

As of September 2017, Unity released their open source Machine Learning tools called Unity ML-Agents. The tool provides a test bed for researchers and game developers to create and use RL agents in their games. The RL code is written in Python and uses Python ML libraries [8].

Unity has full Cross Reality (XR) compatibility which helps developers work with their choice of the VR device without problems. This technical support provides latest driver versions of each VR device, single API for interacting with any of the VR devices, increased performance with low-level engine optimization and ability to switch between multiple devices in projects [9].

One other thing makes the development of VR games easy for Unity developers who work with HTC Vive is the partnership between Unity and Valve that brought SteamVR to Unity platform [10]. This gives developers a better access to the hardware and better performance on their VR projects. Vive and SteamVR are discussed in more detail in the following topic.

2.2 HTC Vive for VR

Vive is a Virtual Reality headset developed by the collaboration of HTC[15] and Valve LLC[16], which was announced in early 2015 and released after approximately one year. This strong collaboration between HTC (an OEM for smartphones) and Valve Corporation (a video game development company and owner of Steam[17], the biggest online store for video games) lead to the creation of Vive.

Valve started VR research more than 3 years before announcing Vive [11] and they worked on both hardware and software and especially on display technology, tracking systems and cross-platform VR API for developers which they called SteamVR. This API helps developers to work with Vive and easily integrate their games into Valve's online game distribution platform Steam.

[13] https://docs.unity3d.com/ScriptReference/GameObject.html
[14] https://docs.unity3d.com/ScriptReference/Component.html
[15] https://www.htc.com/us/about
[16] http://www.valvesoftware.com/company
[17] http://store.steampowered.com/about

On top of that, Valve provided an open source VR API[18] which helps developers to work on VR projects without having certain dependencies on a specific VR vendor's SDK. This project (called OpenVR) provides industry standard API for VR devices, compatibility with latest VR hardware, and optimization for mobile VR.

Vive is a combination of other peripherals like many other VR devices. Communication of these devices provides the Virtual Reality experience as a whole. Parts of the Vive shown in *Figure 2* are explained in the following list.

- **Headset:** Most complex part of the Vive is the headset with two 3.6" AMOLED screens with 1080x1200 pixels resolution, 2160x1200 in total. The refresh rate of a screen is 90Hz and it provides 110 degrees field of view. The front face of the headset contains a camera for safety and the device houses tracking sensors, g-sensor for detecting play area, a gyroscope and proximity sensor. For computer connection, it contains HDMI, USB 2.0 hubs, headphone jack and Bluetooth connector. The headset is made comfortable with adjustable head straps, screen distance adjustment settings and large eye and nose rest which can be used with eyeglasses.
- **Controllers:** Vive includes two hand controllers surrounded by tracking sensors. A controller contains a multi-function trackpad, grip buttons, a trigger button, a menu button and a system button. Controllers are wireless and Micro-USB chargeable. A fully charged controller lasts approximately 6 hours.
- **Base Stations (Lighthouses):** Base stations are sensory boxes to help the system define the room-scale play area. Tracking the headset and the controllers made possible by these devices. Base stations can be used with sturdy tripods or can be attached to walls.
- **Other Accessories:** Vive headset can be enhanced by a head strap that contains integrated headphones. Moreover, additional trackers can be used with Vive, which can be attached to real-life objects and used in the VR games, such as tennis rackets or toy weapons.

Figure 2: *Main components of the HTC Vive.*[19]

[18] https://github.com/ValveSoftware/openvr/wiki/API-Documentation
[19] Figure Source: https://www.vive.com/media/filer_public/b1/5f/b15f1847-5e1a-4b35-8afe-dca0aa08f35a/vive-pdp-ce-ksp-family-2.png

One of the distinct features of the Vive compared to other VR devices currently on the market is its 360 Roomscale[20] feature, shown in *Figure 3*. Previously mentioned Base Stations (also known as Lighthouses) of Vive are amplified with a 3D spatial laser-tracking system. This technology allows defining a play area up to 5 meters diagonally. In the virtual environment defined by this area, a player may move around seamlessly. Vive can be used seated and standing, sensors within headset and controllers are tracked by the system. Tracking accuracy, high resolution, and small latency convince players that they are in the virtual environment and this let them be fully aware of their state of presence [12].

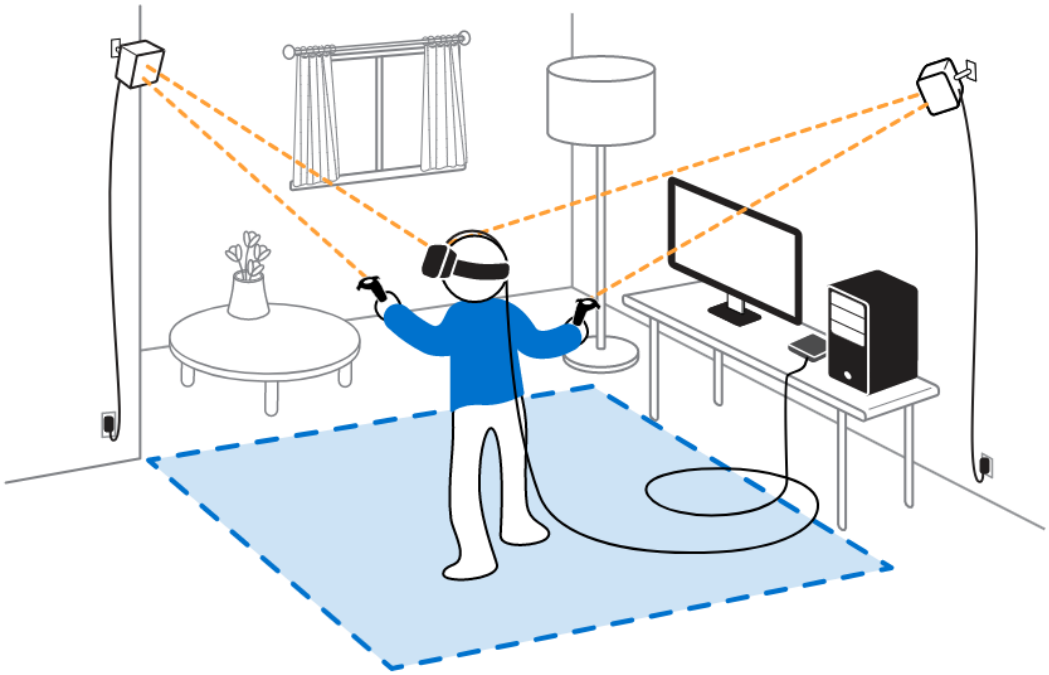

Figure 3: *A representation of player with Vive inside the play area.*[21]

Additionally, to help players be clear of any blockage in the play area and help them to avoid accidents, Vive provides a virtual grid (called Chaperone) that notifies the players if they are leaving the play area. This warning grid is set to define the boundaries of the play area by walls and ground.

2.3 Vrena, the Case Study Game

Vrena is a Multiplayer Arena FPS game on VR platform that is being developed by a local game developer, Jens-Stefan Mikson in Tartu, Estonia. The game has two modes. The first mode is Deathmatch, where all players challenge their rivals free to attack any other player. The second mode is Capture the Flag mode, where the game is played by two teams with the goal of securing the enemy flag.

[20] https://blog.vive.com/us/2017/10/25/roomscale-101

[21] Figure Source: https://www.htc.com/managed-assets/shared/desktop/vive/Vive_PRE_User_Guide.pdf, page 24

In the game, a player uses grappling hooks to move in the game environment instead of walking or using a vehicle. Due to uncommon controls of the game, it is difficult to create bots those interact with the game world identically to the human players. To understand the problem better, the game controls, environment, and rules are examined.

To illustrate the movement mechanics of Vrena, popular examples of games such as The Amazing Spiderman 2 [Treyarch, 2014][22] and the Attack on the Titan [Omega Force, 2016][23] can be mentioned, which have the similar shoot-grab-swing type of player movements.

Game Controls

Virtual Reality games are fundamentally different from non-VR games as their way of connecting the player with the game world is different. VR games are played using a headset that provides visuals of the game world and two hand-controllers to interact with the game environment. Moreover, to be able to track the locations of the controllers and headset many sensors are used. These sensors calibrate the visuals that players receive and they create the feeling of player's presence in the VR environment. Vrena is developed and tested using HTC Vive with the support of SteamVR's development tools for Unity.

The playstyle of VR FPS games has noticeable differences from FPS games played by mouse-keyboard combination or gamepads due to changes in controller characteristics. While playing Vrena a player can use two different weapons independent of each other. In traditional FPS games, the player mostly controls one weapon that is mounted on the player character, which moves relative to the player and the player controls the point of view via a mouse or a game controller.

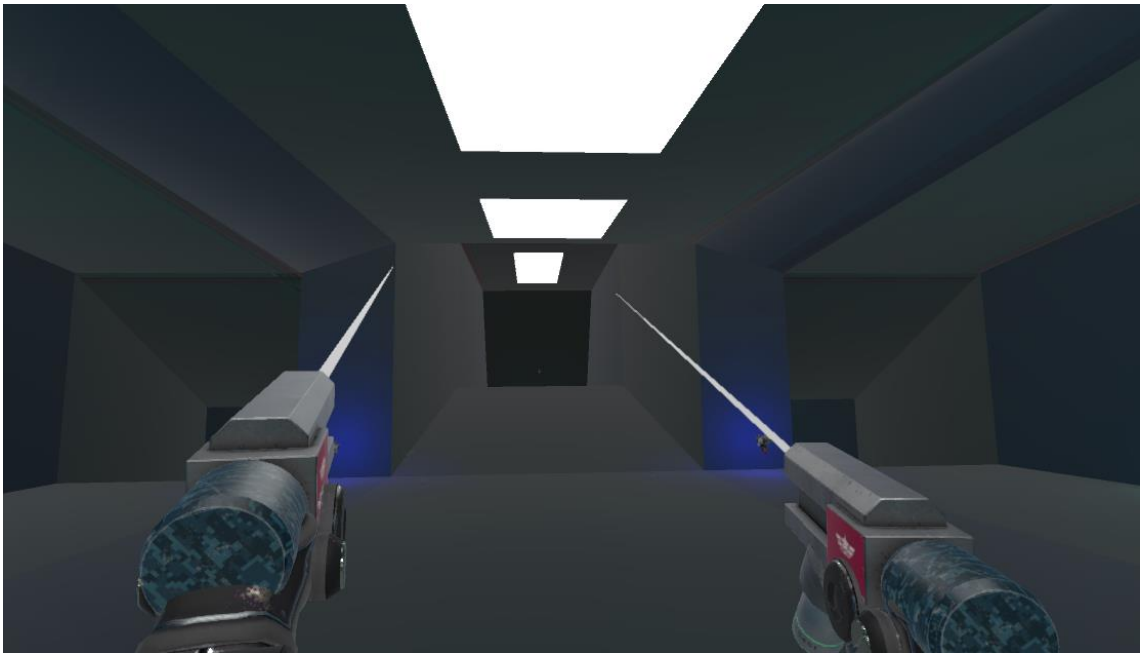

Figure 4: *Screenshot from Vrena, showing the pistols and grappling hooks.*

[22] https://www.activision.com/games/spider-man/the-amazing-spider-man-2

[23] http://www.koeitecmoamerica.com/attackontitan

Commonly there is a static cursor in the middle of the screen and the player can only shoot in that direction. Even in the cases of using multiple weapons in FPS games, the player can only shoot the weapons only to one point. Namely, the player looks and aims at the same and single point in the game world.

Having two separately controlled weapons give the player extra freedom and increases the possibility of different ways they can engage the environment since where the player looks at and where weapons aim will be different. *Figure 4* shows a screenshot of Vrena, where both hands are holding pistols and firing grappling hooks to different directions.

One other difference of Vrena is the movement mechanics. In VR games, players either cannot move with the headset in the real world or have a very limited tracking space of several meters [13]. Moving in long distances is mostly provided by clicking on a point in the game map and teleporting to that location if allowed.

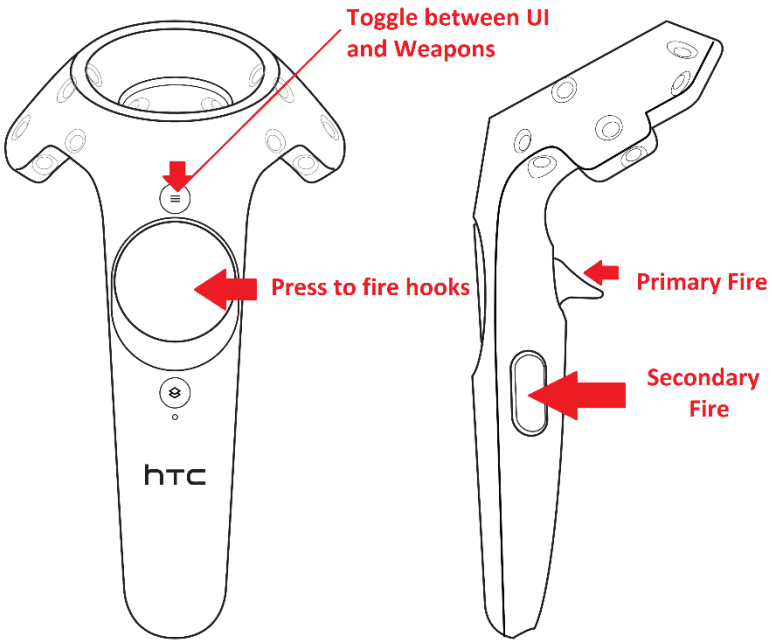

Figure 5: *Image from game menu shows the functions of the Vive controllers.*

In Vrena, players have "grappling hooks" attached to their weapons, which is the only means of transportation. These grappling hooks let the player shoot the hooks from the controllers and when a hook connects to a wall in the game environment the player is pulled towards that point.

The shoot-grab-and-pull mechanism is the only way to move the players from one location to another. Using the grappling hooks efficiently helps the player move to the right locations on the map, avoid enemies and complete objectives. Mapping of the Vive controllers' buttons is presented in *Figure 5*.

Rules and Gameplay

In the game, players have two hands and both hands can wield different weapons. These weapons are a pistol, plasma gun, rocket launcher and railgun. All weapons have infinite ammunition and they do not require reloading. Grappling hooks are always attached to player's hands and are not affected when a player changes weapons. At any time players can use both of their hands to fire grappling hooks for moving around.

Players have pistols in each hand by default when the game starts or they are respawned. Other weapons can be picked up from the game map by placing the player's hand in one of the weapon pickup spots. When the player acquires a new weapon from the pickup spot the old weapon is discarded, it is not dropped or replaced with the gun at that pickup spot.

All the weapon types of the game have different features. Rocket launcher fires rocket projectiles that deal area of effect (AOE) damage in addition to the direct hit. Plasma gun fires plasma bolts, which are also projectiles and deal AOE. Compared to a rocket launcher, plasma gun has higher fire rate and faster traveling projectiles but lower direct hit damage and AOE radius. Pistol and railgun are hitscan weapons, where the point of hit is calculated right after player pulls the trigger and its effect on the environment or the rival player is generated immediately. The pistol has a higher rate of fire but it causes a small amount of damage and the railgun has a very slow rate of fire and causes the most damage amongst all the weapons.

Players have health and they can take damage. At maximum, a player has 100 health points. If a player's health goes down to 0 the player dies and is respawned after 5 seconds with full health. In each map, there are health pickups to increase the health points, which can be picked up by simply colliding with them. Health cannot exceed 100 points and decreases only when the player gets shot. Falling from a height does not cause a health penalty.

The game has a main menu where a player may choose to create a new game session to play or can join another game that already is in play. When creating a game session, players can choose a map, match duration, number of maximum players and a score to reach in order to win the game which is shown in *Figure 6*. This operation also serves as search parameters for the players who want to join already running game sessions.

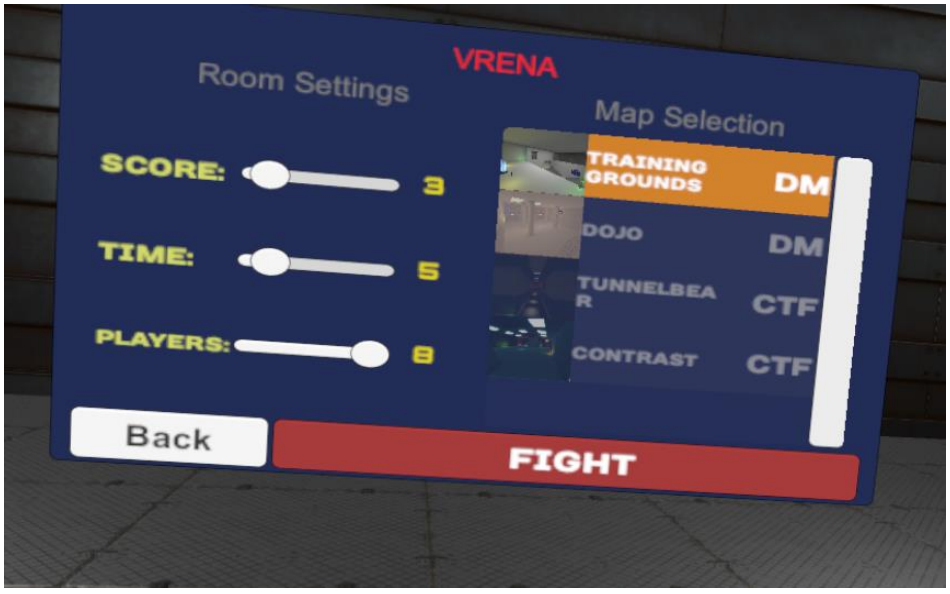

Figure 6: *Screenshot of server creation menu.*

Death-match (DM) mode is where players are responsible for fighting alone and defending themselves. Each kill yields one point to the score. If a player commits suicide, one point is deducted from the player score. Each player keeps an individual score and the first player to reach the max score wins the match. When the match is won, a new round begins in 10 seconds.

In capture the flag (CTF) game mode, there are two teams (red team and blue team) fighting each other. A player who joins the game is automatically moved into the team with fewer players to keep team sizes equal. Both teams have a main base where their members are respawned when they are killed and the team flag is kept in the main base.

The goal of the teams is to capture the enemy flag and take it to their own base. The enemy flag is captured just as in collecting health pickups, by simply colliding with them. Once a player captures the enemy flag, the flag is moved wherever that player moves, and this indicates where the flag currently is. If a player carrying the flag is killed, the flag drops at that location and it can be picked up by both rival players and teammates. If a teammate captures own flag, the flag immediately returns to team's base.

Securing the enemy flag yields one point for the team and the team that reaches the maximum score wins the game. Killing enemy players or capturing the enemy flag also yields personal score as in death-match mode. However, this score does not affect the team score and games result, but it is used for displaying the best players when the game ends.

Environment

At the current development stage, Vrena has nine maps. Five of these maps are death-match (DM) maps and other four maps are capture-the-flag (CTF) maps. However, three of these maps are still under development and six of them are ready for play.

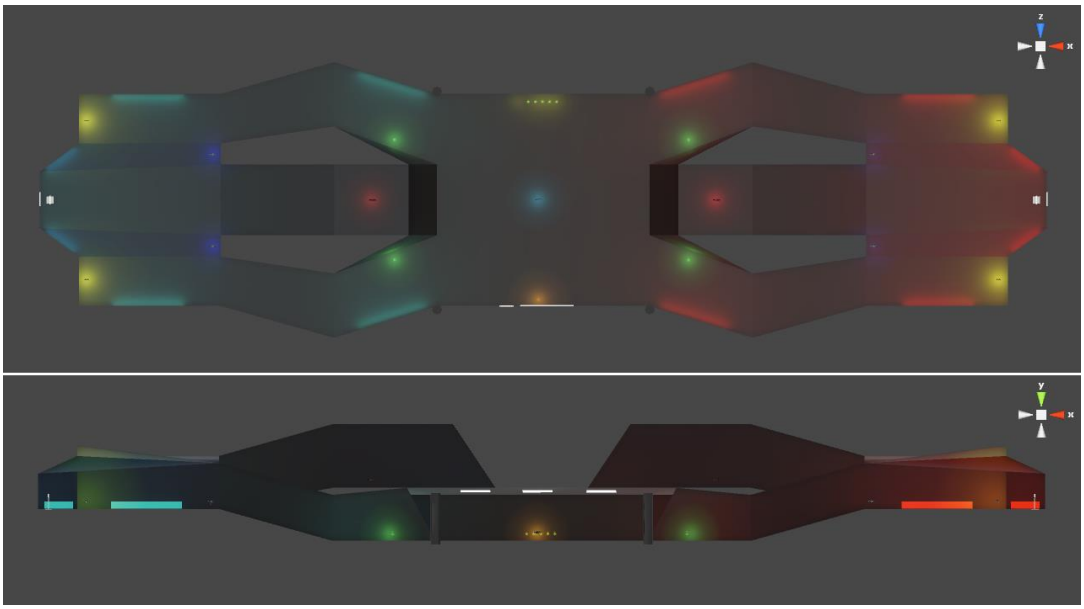

Figure 7: *Top and side views of the game map Contrast.*

For testing the developed tool it is decided to work on a CFG map instead of a DM map where the ability of decision making can easily be observed. Only map candidates that

comply with these criteria are maps called “Contrast” and “Tunnel Bear”. The map Tunnel Bear is relatively bigger and more complex than the map Contrast, which would require more game logs to be collected. To ease the overall process the map Contrast is selected, *Figure 7* shows the top and side view of the map.

Contrast is a CFG map, which has a horizontally symmetric structure. Both ends of the map have spawn points for the players and there are the flags to capture at the far ends of the map (see white points in *Figure 7*).

Design of Contrast is good enough to test the ability of bots to move around, execute climbing pulls and capturing the flags. This map has three corridors from the teams’ spawn points to a bigger central hall. Players can reach the central hall from the side corridors of the map which leads them to go down there. Alternatively, they can reach there from the middle corridor which leads the players to a higher ground level on the map then let them jump down to the central hall.

There are no restrictions in this map when using grappling hooks. To move around, players can shoot their hooks to any wall, floor or ceiling to pull themselves.

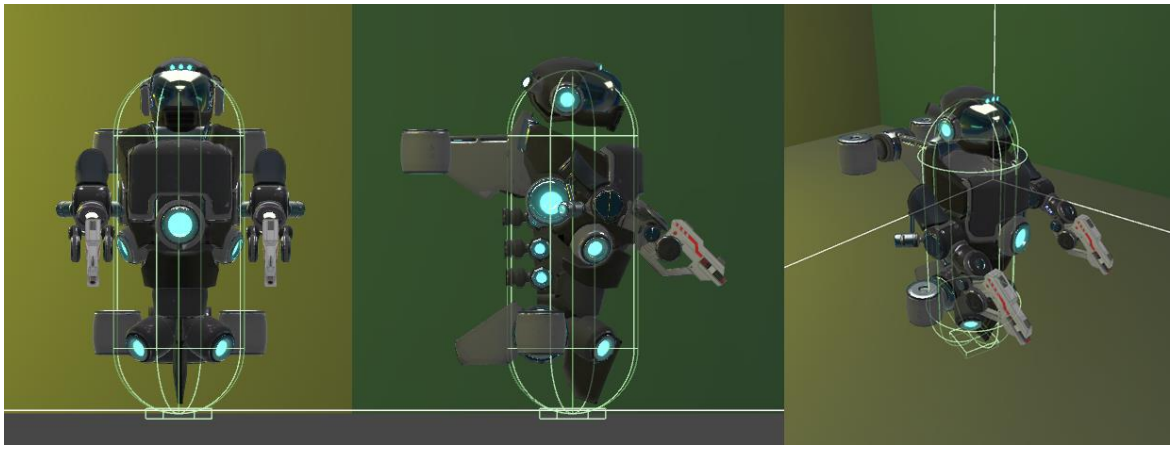

***Figure 8**: Side, front and corner top view of the Player Object in Vrena*

The Player Object which represents the players in Vrena is a game object that contains many different components. The Player Object has the arms and the head as independently moving objects which get their position and rotation information from the VR set. The surrounding green capsule on the Player Object seen in *Figure 8* is the collider of the object, which is used in detecting collisions with the game map and other game objects. The small box-shaped collider at the bottom conveys the information about whether the player is on the ground.

2.4 AI in Video Games

Artificial Intelligence helps people understand and handle the uncertain and/or complex computational real-world problems. To be able to solve human problems, AI can be observed from four different perspectives. Philosophically, AI is to perceive human nature and intelligence to build software that can imitate how thinking process works. Psychologically and biologically, it is understanding circuitry and mechanisms of human brain and its mental process to project the same workflow into decision processes while building software.

Lastly, from engineering perspective AI means building algorithms, those help software and machinery to perform human tasks as humans do [14].

In general, AI methodologies help games with movement, decision-making, strategic thinking (group AI), agent-based AI models and many other systems. Even though AI has been around more than a half-century in academia, its applications in games came in 80's with very primitive forms.

In its early times, one of the remarkable examples of AI in games was Pacman [Midway Games West, 1979] [14]. In the game player had to move in a maze-like map and collect points, meanwhile avoiding randomly routing enemies known as ghosts. If the player is close to a ghost then the ghost starts following the player until it loses the track of the player. This simple technique was a usage of State Machines in this game. This play style made it challenging and interesting as if the enemies understood the game environment and reacted according to it. Pacman did not only become very popular but it inspired many developers to use state machines in their games in more complex forms.

To this day, many commercial games' AI is a combination of a state machine and random decisions made depending on the current state of the agent. For strictly rule-based games such as backgammon and solitaire, these techniques are just enough since the decision possibilities are not many. However, games like chess and go require decision trees because of a vast number of choices the players can make [15].

Later shortest path and pathfinding algorithms had usage in many Real-Time Strategy games. One of the most popular pathfinding algorithms is A* (A Star) algorithm, which has wide usage and regarded as one of the optimal solutions for this task. First examples of commercially successful games which heavily relied on path-finding algorithms were such games as Warcraft [Blizzard Entertainment, 1994] and Command and Conquer [Westwood, 1995] [14]. In these games, players select and move their troops and vehicles to the marked points on the game map, where many obstacles existed. Game units were able to find a way to reach the destination, even though it may not be the closest route.

Around the year 2000, commercial game developers also started to implement neural networks (NN) in games to add different personalities to NPC's. NN applications in games helped developers to create NPCs which can develop their own characteristics depending on effects of the game environment. Such game example is The Sims [Maxis Software, 2000] [14], where players control the life of a simulated human avatar (called a sim). The game used a system called artificial life [16] which is based on NN techniques.

Behavior trees became a popular technique for creating AI agents in games soon after the millennium. One of the strong examples of behavior trees usage in games was Halo 2 [Bungie Software, 2004] [14]. Behavior trees have the same abilities of the finite state machines but its structured design makes it easy to understand and use even the logic gets more complex. Each state of a behavior tree is called a task and tasks can have sub-behavior trees, which makes them have a hierarchical design and depth.

One of the recent topics in AI research is Machine Learning in games. It is still more in the academic scene of the game development rather than commercial. However, AI beating world champions at go and chess has gathered a lot of attention and has proven how powerful ML techniques can be. One notable example of ML techniques is Reinforcement Learning. This usually consists of a state, action and reward/punishment usage for teaching the agent. During the training, NPC explores the game environment with partially random actions and the system rewards or punishes the agent in relation to the positive or negative

outcome of that action. This training cycle goes on until the agent satisfies the predetermined ending conditions.

Even though RL agents can carry out given tasks successfully, overfitting issues may cause the agent to choose always the best route or similar routes to complete the task. Poor tuning of variables can also cause movement jitters and unexpected behaviors, which make the agents distinguishable from human players. ML techniques can produce better results if configured carefully, and sometimes used with other AI techniques.

3 Related Work

Many studies have been done on the topic of creating human-like AI for FPS games using various techniques. Some of these studies use commercial games such as Quake 2 [17] [18], Quake 3 [19] and Counter-Strike: Source [20] for the reason that these games provide a playback mechanism of their game sessions. For general purpose, this data is used for teaching the AI to perform its tasks. Moreover, some researchers use their own testbed games that provide basic features of commercial FPS games such as collecting ammo and health packs [19]. To note, all these games were traditional FPS games for PC, not VR games.

McPartland and Gallagher (2011) [17] used Reinforcement Learning to teach a game bot to navigate, collect items and combat using Tabular Sarsa Algorithm. The bot received state inputs from the game world via six sensors and reacted with necessary action. Then state-action-reward sets saved in a look-up table. One of the drawbacks of this method occurs if the subject game has continuous state domain, such as FPS games, because of the need to use huge look-up tables to store the data. Moreover, training requires a big amount of memory and strong CPUs to be able to cope with calculations. Despite this research has shown that RL provides promising usage for FPS bots.

Patel et al. (2011) [21] uses Q-Learning, a variation of RL, to investigate this algorithm's applicability on FPS agents. To simplify the problem, they scale the 3D game environment down to two dimensions and divide the map into square areas, which are regarded as location state of the game bot. To find the optimal solution, they do tests on a different number of maps with different section sizes. This approach gave the idea of dividing the game map into cubic areas to create a matrix representation of the game map and using this structure as location state of the agent which helped to avoid the continuous state domain problem.

Later McPartland and Gallagher (2012) [19] used another RL related technique to train FPS bots, that they called Interactive Training. During training of bots, they used an interface to let human observer punish and reward the AI agent instead of programming the reward mechanism into the game. Differently, they created a predetermined list of states along with actions to define the changes of the agent states. Some instances of the states were "low health", "high health", "low ammo", "high ammo", "type of weapon", "is enemy in sight", "enemy distance". Depending on these states player took actions such as "melee attack", "ranged attack", "wander", "use health item", "reload weapon", "dodge" and "hide".

Similarly, Wehr and Denzinger (2015) [4] extracted a list of agent states from the game logs they collected. They called one instance of the recorded state set "an observation" which included necessary information for the NPC. Additionally, they worked on the elimination of the irrelevant data from the game logs and after refining the logs, they eliminated too costly action candidates from their data set. After the cleaning, this data saved into a similarity matrix identifying similar action candidates. A parallel design is followed for this research, determining the much smaller state space of Vrena and refine the collected game recording before using them.

Kazmi et al. (2010) [5] mentioned ways capturing player behavior patterns from game recordings in his research. To achieve this, they reorganized the game environment with triggers to change agent states. This method gave them the ability to get state changes not only related to the agent but to the game map too. Recorded data was also saved as strings containing action and states for post-game usage. Additionally, they recorded important events like the death of the player as well.

Capturing the movement patterns from the game logs and using the gathered information for creating NPC agents was another approach. Thurau et al. (2004) [6] worked on Quake

II training samples to gather information about player behaviors using Neural Gas Algorithm to extract player movement on a waypoint map. After the formation of waypoint map, potential movement trajectories are placed into a topological map and this information is used with a Multi-Layer Perceptron to guide bots. One of the difficulties faced in this research was to imitate complex behaviors of the players such as waiting behind an object to ambush an enemy player.

Conducting player surveys to evaluate outcomes of AI was one of the test methods used in the reviewed literature. Hladky et al. (2008) [20] used observational game data only visible to a human player from logs for predicting opponent positions. They used game recordings of Counter-Strike to achieve this goal to make the bots act more human-like. To compare the success of the predictions extracted from game logs they conducted a human subject study to let test players make guesses and compare the outcomes to the developed system's predictions.

Conroy et al. (2011) [22] mentioned the importance of AI with human-like behavior since players consider it unbalanced or cheating when agents are not configured well. Static and rule-based actions become predictable after a while of playing FPS games. They analyzed the gameplay recordings of the players and extracted information about how and when they act in a certain way. To measure the success of the AI and flaws of its behavior they conducted a survey. Depending on these experiences, it is reasonable to conduct player surveys to measure rate the quality of the results of the research.

Sheehan et al. (2008) [23] mentioned the potential use of game logs to replace time-consuming process of supervisory learning of AI agents. In the research, they worked on Quake II gameplay recordings and used external data mining tools to retrieve information from the game logs that would be useful. They noted this method had been highly unsuccessful and deeply flawed due to not automating information extraction process from game logs and poor quality of information representation in the game logs. These issues provided the idea of creating a custom way of recording game logs with the high quality of information representation and automating the information extraction from the data to avoid the pitfalls mentioned in this research.

As there are many different ways to create NPC AI, it is not easy to decide when the main goal of the research is to construct it from gameplay recordings. Kopel and Hajas (2018) [24] made a comparison of different artificial intelligence algorithms to study their learning speed. These algorithms were a combination of Decision Trees and Finite State Machines, Neural Networks, a combination of Q-Learning (QL) and Genetic Algorithms (GA) and lastly Q-Learning algorithm alone. Their study revealed that among these, best option was QL. Yet the hybrid method of GA-QL was as good as QL alone. This gave the idea that different approaches may be combined together to achieve a near optimum result.

4 General Approach

Using Reinforcement Learning's state-action-reward design is a promising and validated idea for establishing the fundamentals of the NPC AI. There are many variations of RL algorithms and most of them use Bellman Equation (see formula (1)) and its modifications. Simply put, the equation is based upon updating a value function that returns the action (a) to take depending on the given state (s) parameter. Before updating the function, the next state (s_{t+1}) multiplied by decay value (γ) and the reward value of the next step (r_{t+1}) is summed for each action. Then the maximum value among these sums is registered to the value function V(s).

$$V(s) = \max_{a}\left(r_{t+1} + \gamma V(s_{t+1})\right) \quad (1)$$

If this approach is discussed on a classical Grid World example shown in *Figure 9*, the agent in the maze moves one tile each action and ends up on a new state. The environment contains a winning state which has the reward of 1 and a losing state which has the reward of -1, both states ends the process. All the other tiles have the reward of 0 if a movement penalty reward for increasing the speed of finding a solution is not introduced.

The agent can move vertically and horizontally, these actions can also be influenced by an exploration rate (ε) which helps the agent to take a random decision instead of the best one. After the agent arrives at a bad or good ending, the value function is updated for that action and state. Later when the agent is to move on this state with an updated value, it will decide to move onto it or avoid it depending on this value. After many tries, the value function will have an optimal solution for the environment.

Use of a decay value helps the system appoint lower values to the states that are away from the ending, this way having equal values for each state is avoided.

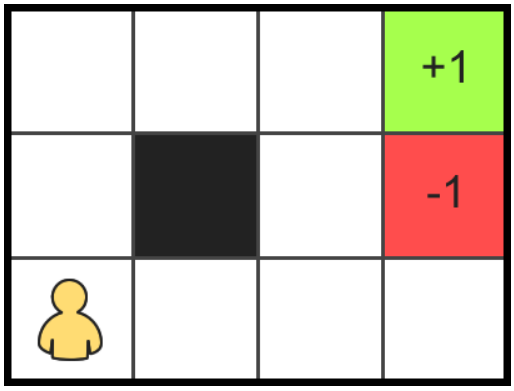

Figure 9: *A classical Grid World maze used in RL problems.*

In Vrena, players can take actions which may result in different step sizes. Namely, a player can jump into a random state on the map. This may break the flow of evaluating states step by step as they get close to the good ending. A player can jump from start to the end, alternatively can also use many steps in between. This results in an average value for that step instead of a low value which needs to be avoided.

Additionally, training an agent taking random actions in a game that is heavily physics based takes a long time and the results might look unnatural. Another problem is implementing an exploration rate to the system to let agent chose random actions sometimes, instead of using the optimal route it finds. The random actions the agent takes may be totally unexpected and noticeable by the human players.

Some of the working mechanisms of the RL techniques can still be adopted by this project. Decision-making functionality of the RL can be used since gameplay recordings are going to be used as the training data. Reinforcement Learning agents explore the game environment semi-randomly during training and receives the reward depending on the next state. Yet there are possible implementations of RL with Interactive Training where a human supervisor can give rewards and punishments from outside of the system. Similarly, the reward values can be connected to necessary sections of the game logs and let the agent make decisions accordingly.

Predetermining the states and actions of the game and then developing a custom gameplay recording tool simplifies the data mining process. This way information representation of the data within the gameplay recordings becomes high and its content becomes human-readable. Considering Vrena, it is reasonable to record states when controller actions are taken and game state changes when entering special areas such as health, weapon and flag pickups. These special actions and states all should be carefully selected and changed during the gameplay recording sessions.

To be able to collect the necessary information about the game environment and the agent, a distance sensor is attached to the agent to change agent's state. Slicing the game environment into cubic areas and then using this information as the positional state can help with the continuous domain problem in the RL methods.

After creating gameplay recordings, the system should collect all the logs and refine this data to create a pool of information. This movement pieces are acquired by slicing the session recordings into subparts which have a similar structure. This structure is defined as a log section, which is a piece of the recording when the player makes a jump and then lands on the ground again. Namely, each part of the log that follows ground-air-ground pattern forms a log section.

Once log sections from all the game recordings are generated, these log sections can be measured in their success. Then this success value is added to each log section automatically as a reward point, depending on the factors of grappling shoot count in the air, distance and time.

Log sections can be stored in a lookup table, which is a custom associative array with a number of elements up to the same size of the cubic sectors the game environment was divided into. Log sections are placed into the array by their cubic sector. This way agent can decide to take one of the log sections from the lookup table, which is the current cubic sector it is in.

Using a small look-up table helps the system use less memory and have less computational burden compared to look-up table usage with continuous state spaces. Additionally, it is reasonable to eliminate the log sections with too low values to normalize the data before saving the log sections to a lookup table.

Since the player movement in the game is heavily influenced by the physics when actions are executed from game logs, they may not create the exact same trajectory of recorded action. But the most reasonable thing is not to use character position from the game logs to move the agent, instead use the hook and pull actions to imitate the same action from the

log. This causes slightly different results each time due to physics but creates a more realistic look since the same log section's result looks different. Recordings of hands, body and head positions and locations are used for cosmetic purposes to create human-like gestures and movements.

To evaluate the success of the log AI tool a survey containing videos of a human player and AI player is conducted for a player opinion test to see whether the players can decide which payer is a human and which one is an AI agent. This survey contains additional questions to ask these players what affected their decisions.

The log AI tool is a hybrid system that uses gameplay recordings as its navigation data and uses a modified state-action-reward mechanism to move the agents. To be able to achieve this system, first, a logger component is implemented into the game to collect the necessary data from the game sessions. Then this data is collected and refined into the form that the decision mechanism can use it. And lastly the decision mechanism chooses the right actions depending on the configurations predetermined and the agent uses this actions to move around.

5 Recording Gameplay

In this section, the gameplay recording ability for the log AI tool is discussed form the design and implementation perspectives. Additionally, the structure of the log files and the decision of data storing format is discussed.

5.1 Approach

A gameplay recording, also known as a game log or demo file, can be defined as a file that contains information captured and stored from a live game session. Structure of this informational data is generally a sequential list of observations recorded by time intervals or by actions taken. The content of the gameplay recording can be either human-readable text or a text that needs to be parsed before being used [23].

Previously it has been suggested interactive computer game logs have the potential to supplement or even replace supervised learning techniques (Sheehan & Watson, 2008). Yet this research was largely unsuccessful. Researchers recommended that the information represented in the game logs must be well defined and the information extraction process should be largely automated.

Traditionally, game logs capture every state change in the game sessions to be able to visualize replays seamlessly. In turn-based games, board games or puzzles a recording can be taken by the turn or the action. The state space and the number of steps to finish a game might not be as big as compared to other game genres. Thus, size of the log files created is not a big problem to deal with.

As for an FPS game, logger should be recording all the related state changes in the game environment, states of the player and the other characters or game's network traffic by periods of timeframes. This technique creates bigger log outputs compared to board games or puzzles.

Initially, the type of the information added to the game logs should be determined. Finding optimum solution is important as it affects the amount of data that is gathered at the end of the recording session and the NPCs' decision making speed. For this reason, it is needed to utilize the log recording system to capture just the necessary information to reach the smallest size.

The fundamental purpose of the research is to make a game agent in the case study game move as a real player would do. Therefore at first, recording sessions were focused on recording one player's movement information. Once enough gameplay recordings are collected, this information is used for navigating the AI agent in the game environment.

In the case study game, maps have verticality in their design. Players do not only move on a plain navigation mesh but they move on disconnected islands, high grounds, and walls. For this reason, the system requires the ability to record player's three-dimensional position on the map, rather than coordinates to move on a plain board.

Capturing the player's position and using this information to create a lookup table is inefficient due to a continuous state problem. One of the approaches to solving this problem is to divide the map into artificial cubic sectors and record the current sector the player is in.

Log File Structure

Actual recordings of the game session are encapsulated by an object called Session Log. The meta-data fields about recording sessions' start date, end date and the name of the map recording are filled when the recording is started. This information is used while game logs are refined for use. The following variables are kept in a Session Log:

- ***Session Start Date (string):*** Date and time when the recording of the gameplay is initiated. This information is used for naming the created log file and determining the length of the session.
- ***Session End Date (string):*** Date and time when the recording of the gameplay is over. This information is used for determining the length of the session.
- ***Map Name (string):*** Name of the game map the recording is made in. This information is used during data is being refined, logs from the same maps are used together.
- ***Log Sections (list of Log Sections):*** A list that contains the sets of recorded information from the game.

Log Section is the structure that stores a set of information about each distinct movement decision a player makes. This set of information is defined as a jump, in which contains the recorded observations between ground-air-ground state changes of the player. This way the NPC agent can continue its movement and make his next decision once it is on the ground. For this reason, Log Section class is designed as follows:

- ***Sector (vector3):*** Representation of unit cube that the player is located in. This information tells where this log section belongs, as a start position of a decision.
- ***Success Value (float):*** The numeric value that keeps the success value of the log section. This value is used for ordering the log sections and this order is used for difficulty settings of the NPC agent.
- ***Log Lines (list of Log Line):*** A list of saved observations. Core information that is collected from the player and the game environment.

Imitating the movement and gestures of a VR player requires recording rotation and positions of the hand controllers and the headset. Additionally, player and action states should be recorded. The observation of the player and the game environment that contains this information is called a Log Line, and the structure of the file is as follows:

- ***Time (double):*** Time of the one recording line or observation. This information is used for determining the time difference between each observation of the recording session.
- ***Player State (enum):*** Player state provides the information whether the player is in the air or on the ground. Since the game logs are sliced into sections and used independently of the log file they are taken from, this information is used for deciding from where to separate these sections.
- ***Action State (enum):*** Whenever the player uses the hand controllers to shoot and hook grappling hooks and whenever he releases corresponding buttons on the controller, state of this data changes and it gets recorded. When the game starts, the state of the action is idle and if player presses or releases right or left buttons on the controller it changes for one observation.

- ***Position of the headset (vector3):*** HTC Vive provides its users a tracking area where the player can move around a couple of meters. Even though this information is not used while NPC agent navigates using log data, it provides a cosmetic value when the player is stationary in the game world.
- ***Rotation of the headset (quaternion):*** When the players look around using the VR headset, their avatar animates the same way in the game world. Horizontal rotation of the headset can be regarded as the rotation of the player's body in the game. This information is used for cosmetic changes.
- ***Positions of the hand controllers (vector3):*** To be able to place AI agents hands in the 3D space, hand positions are recorded separately. This position data is captured as position relative to the headset. This is used in the same way when NPC agent navigates using logs.
- ***Rotation of the hand controller (quaternion):*** Same as previous usage, rotation of the hand controllers is used to create a cosmetic effect for rotation of the weapons the player holds.
- ***Target point of the hand controller (vector3):*** When the player shoots the grappling hook, the point it hits in the 3D environment is recorded. This info helps the agent to aim the same location to hit when NPC agent moves.

For the current state of the study, structure of the log files are designed to achieve the NPC movement. Many other factors are left out, which can be included in future versions of the research. These are factors such as the type of the weapon, the health of the player, amount of the ammo, collected items.

Some of the variables saved in the game logs are used for creating cosmetic visuals. These visuals are not necessary for navigation of the agent, but for giving players the illusion of playing with a human rival. Movement of the player's hands, rotation of the player's body are simulated in real-time to imitate the human player's actions.

The position of the headset from the logs is not used to move the agent on a trajectory that is created during the session recordings. Doing this would create an accurate but rigid movement. This needs to be avoided since the game is heavily influenced by the physics. Suppose that an explosion happened near the NPC agent, it is naturally expected that this impact changes the current momentum and movement direction of the agent. If the agent was mounted on a fixed trajectory and moved, this effect would vanish and create an unreal display. Because of this scenario, change of the action state is the only factor that affects the navigation of the agent.

Another thing to note about using action state for movement is the effects caused by computational latency. In Vrena, players gain velocity as long as they keep the control button pressed and the length of this time determines the point they land on the game map. A CPU latency during NPC movement would cause a slightly longer button press, which would cause a subtle increase in velocity. Consequently, the NPC agent lands on a further point in the map. Even though this effect might look like a drawback, it actually adds some diversity to the movement since it breaks out of the main trajectory on a small scale.

Saving target points in the log helps to recover from the accuracy problems caused due to unexpected physical effects and computational latency. The regular thing to do while simulating NPC actions is to shoot the hooks to the point where the controllers look at. This information can be gathered from rotations of the controllers, which is already stored in the log lines. However, the points controllers aim at can change significantly depending on the size of the sector area and where the agent stands in this area.

Storing Log Files

After a log file is created it is stored should be stored on the disk to be used later. One possible way to store it is to take a snapshot of the game log from memory after gameplay recording is over and then save this data in binary format. However, this method should be avoided since it eliminates human-readability of the log files and makes it difficult for other developers to use it later.

Previous studies which interested in the usage of game logs chose to work with Extensible Markup Language (XML) format [25] [26] for data storage. However, JavaScript Object Notation (JSON) a newer file storage format that can be used for this purpose. *Figure 10* shows a comparison of XML and JSON files for game logs.

Extensible Markup Language is a structured file format to store data, which has a format very similar to HTML. In XML, you can create your own custom elements instead of pre-defined tags and attributes. One of the downsides of XML is additional characters and repetition of elements which are required for expressing the data. However, XML files can follow a strict structure using schemas that define the requirements of that file [15] [27].

On the other hand, JSON is another widely used data storage format, which is designed for the same purpose as XML. Even though its origin is the programming language JavaScript, JSON is an independent data representation format that is used and parsed in many programming languages. It is generally used for transferring data over the internet due to its lightweight design [15] [27].

```json
{
    "sessionStart": "16-03-18-12-33-09",
    "sessionEnd": "16-03-18-12-33-12",
    "mapName": "TestMap",
    "logSections": [{
        "sector": {
            "x": 2.0,
            "y": -1.0,
            "z": 0.0
        },
        "logLines": [{
            "time": 37.84491729736328,
            "state": 0,
            "action": 1,
            "playerPosition": {
                "x": 24.560218811035158,
                "y": -8.940696716308594e-8,
                "z": 3.3498525619506838
            },
            "cameraRotation": {
                "x": 0.24549755454063416,
                "y": 0.017123013734817506,
                "z": 0.031348951160907748,
                "w": -0.9687389135360718
            },
    ...
```

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<root>
    <sessionStart>16-03-18-12-33-09</sessionStart>
    <sessionEnd>16-03-18-12-33-12</sessionEnd>
    <mapName>TestMap</mapName>
    <logSections>
        <sector>
            <x>2</x>
            <y>-1</y>
            <z>0</z>
        </sector>
        <logLines>
            <time>37.84491729736328</time>
            <state>0</state>
            <action>1</action>
            <playerPosition>
                <x>24.560218811035156</x>
                <y>-8.940696716308594e-8</y>
                <z>3.3498525619506836</z>
            </playerPosition>
            <cameraRotation>
                <x>0.24549755454063416</x>
                <y>0.017123013734817505</y>
                <z>0.031348951160907745</z>
                <w>-0.9687389135360718</w>
            </cameraRotation>
    ...
```

Figure 10: *Representation of JSON vs XML formatted log files.*

Both file storage formats have their own unique abilities, yet it is shown [27] that JSON requires fewer resources, is faster than XML and provides good human-readability. C# language has built-in support for XML language, yet Unity game engine provides its own JSON serializer and parser which simplifies the programming process.

5.2 Summary

In the first part of the system, the suggestions made by previous researchers are taken into consideration to avoid the pitfalls of data complexity, low information representation and manual data extraction from the game logs. For this reason, a log recorder component is a necessary part of the developed system which helps to deal with less complex data and acquire clear information.

The log files contain only the agent's states, the agent's body parts and actions taken by the agent. This information is saved by action and by time intervals. Recorded actions and states help the agent to navigate. Rotation and position of the agent's body parts are used for creating human-like movements.

Logs are created in a structured manner. The main file of the Log Session contains the information about the game session and a list of Log Sections. A log section is a set of recorded observations which are called Log Lines. Log Sections also contain the value of the cubic sector it starts in and a success value. Log Lines contain rows of logs for every change considered for recording.

Finally, logs are saved in JSON format due to being human-readable, compact and supported by Unity game engine.

6 Refining and Arranging Data

In this section, refining and arranging the gameplay recordings and preparing them for NPC agent's decision process are discussed.

An FPS game may have a three-dimensional environment. However, the dimensionality of gameplay recordings goes higher due to many other factors such as collectible items, other players, visuals and sounds in the game. To create the most useful game log, this data should be filtered and dimensionality of it should be lowered. After recording the necessary data and refining it, it should be ready for use by the game bot [23].

6.1 Approach

An initial step for arranging the data has already been taken in the previous chapter, which was dividing the log files into sections to extract series of movement decisions from the log files. This operation was done during the gameplay was being recorded, which helped to avoid saving information about which sector the player is in every log line. Following steps complete the data refinery process and leads to the creation of navigation data.

Success Value

A major factor in arranging the collected game logs is the success factor of the log sections. It solely relies on the definition of a log section's success. The idea of the success rate may change depending on states of the game and the agent. For NPC navigation problem, a straightforward approach is followed to make the success value simple and useful.

Supposing that, if a player moves the longest distance *(d_{total})* without touching the ground, makes this movement with the least number of grappling hook shoots *(s)* and in the shortest time possible *($t_{section}$)* the movement is the most successful movement. Here the distance d is considered to be the cumulative distance from start to end. If the player runs in a circular shape and comes back to start, the distance is zero but the perimeter of that shape. The section duration is also calculated by subtracting the time value of the last observation in the section from the first ones. Division of total distance by duration times grappling hook shoots creates the success value *(r),* as it is shown in the *formula (2).*

$$t_{section} = t_n - t_0$$

$$d_{total} = \sum_{n=1}^{n} \sqrt{section_n{}^2 - section_{n-1}{}^2} \tag{2}$$

$$r = \frac{d_{total}}{t_{section} * s}$$

After the game logs are generated, they are collected and the log sections extracted from them create a log section pool. Each item in this pool is run through this function and assigned a success value.

Cleaning the Data

Before sorting the log section pool, unnecessary parts of the data are cleaned. Cleaning the log section pool is done by removing the log sections with zero or too low success values. This operation is done to normalize the data set and create a better decision palette without extraordinary movements. To achieve the desired data set, mean, variation and standard deviation values of the log section pool is calculated. Derived from these values, log sections with success value below second standard deviation (2σ) of the log section pool are removed from this pool.

In the pool of log sections, there are always some items created by bouncing from the ground. These log sections do not contain any action in them and they are created due to high jumps or explosions pushing the player into the air. Since these log sections do not have any actions inside, they are removed during this cleaning process.

Sorting Log Sections

Players enjoy adaptive difficulties in games and want to have set of options to choose from. Video games provide these settings most commonly as easy, medium and hard options. However, a player's skill level may stand between one of these settings and the difficulty choice they make may feel a little harder or easier for them [5]. Having log sections labeled with a success value gives the opportunity to sort them and determine what range of logs to choose from.

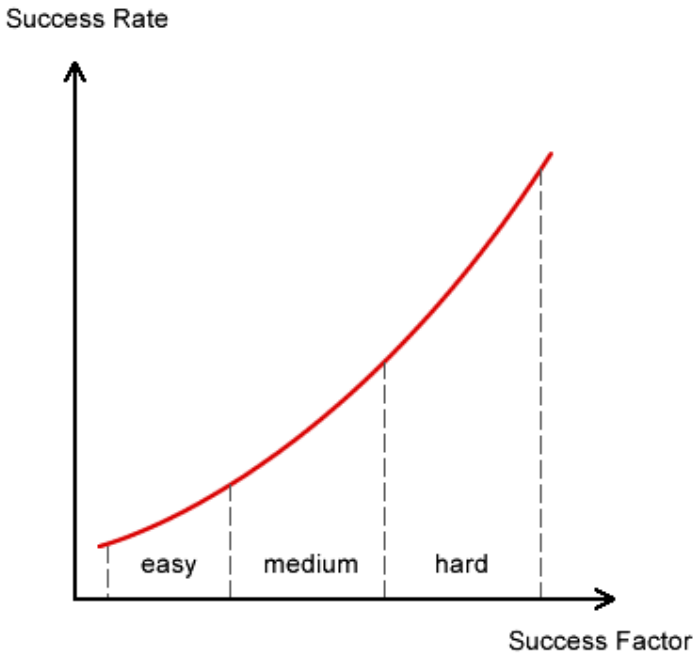

Figure 11: *Success factor and success rate for deciding NPC difficulty.*

These sorted lists are used for determining the difficulty of the NPCs. From all lists, a slice of data is taken depending on its success value. *Figure 11* shows the representation of difficulty ranges changing by success rate and success factor.

By using this method, the system receives the ability to create many different difficulty settings and even difficulty adaptation for the player, if such ability is to be added in the future. Players enjoy the games more when they play against other players who have the same range of skills, which balances the gameplay and creates the possibility of winning and losing equally, thus it has significant importance [28].

A game session may not contain sole successful actions or states. If every single instance of game sessions from the logs were to be examined, it would be obvious that the game sessions have both successful and unsuccessful decision instances. This rate may change during gameplay and any player may make bad choices during the session despite being the winner at the end. That means, rate of the successful decision made in the game session is the deciding factor in the overall success.

To emulate this effect, it has to be ensured that NPC agents have wide varieties of decisions available to them when a movement decision is to be made. This would help the NPCs have a diverse way of movement which varies within a range of decisions rather than having similar steps.

Lookup Table for Navigation

After cleaning the data and ordering the log sections, a look-up table for the decision process is created. The lookup table is an associative array structure that contains a sector as key and a list of log sections as value. Since the log sections are already ordered by their success values, log sections from the log pool are inserted into the lookup table by their sectors. This creates the lookup table with ordered log sections in for every sector. The representations of sectors over the game map is shown in *Figure 12.*

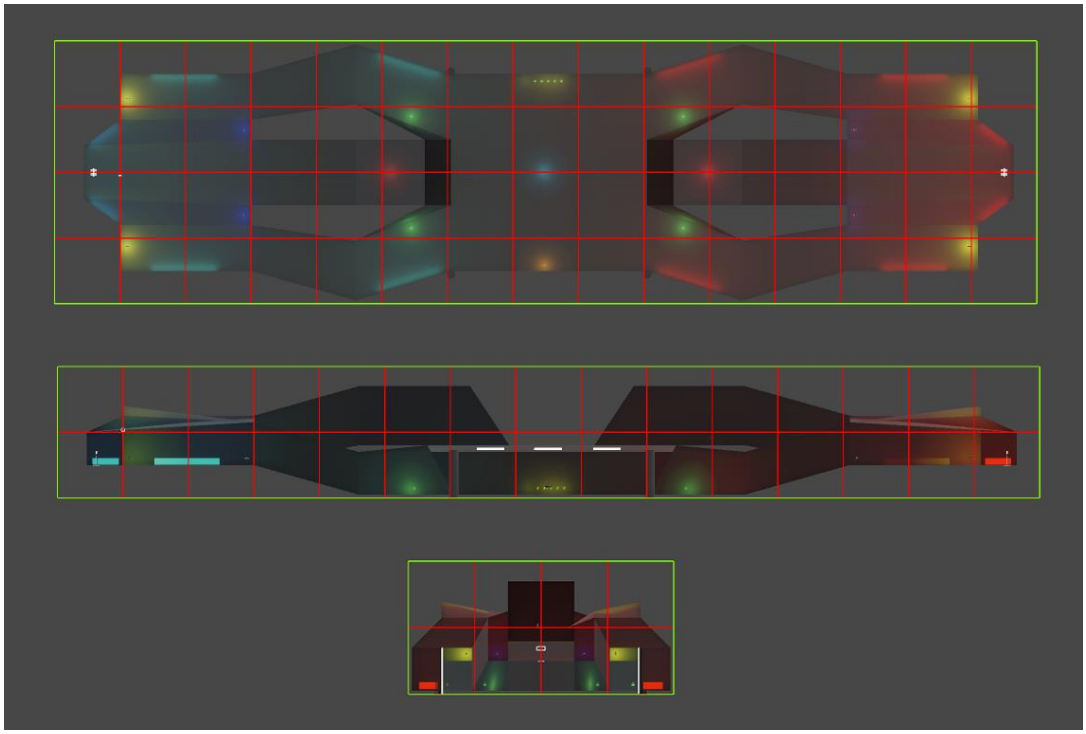

Figure 12: *Division of map into sectors. Top, side and front view total 120 sectors.*

The log pool that is used for creating the lookup table is a combination of the refined data from multiple game logs. Game sessions those are recorded in the same map but in different times or different players can be collected and fused together. This ability gives the developed tool the ability to expand the dataset over time. Once new gameplay recordings are gathered, the log pool goes through the same process and creates a new lookup table.

6.2 Summary

In the Refiner component created log files are collected and processed for generating a single data table for the navigation system.

Initially, all the log sections are given their success value. This value is calculated using the distance traveled, number of actions taken and the lifetime of the log section. Even though this does not represent what the real success of the action is, it helps to sort the actions with the longest distance traveled in shortest time with a minimum number of actions.

The log section pool is cleaned by removing unnecessary log sections. Items with low success values and which fall below the second derivation of all log sections are cleaned out to create a more homogenous data pool.

All the log sections collected from log files are sorted according to their success values. Using sorted log sections enables the AI tool to make navigation decisions by a fluid difficulty selection.

Lastly, sorted log sections are placed into a lookup table according to their sectors which are the key values. As a result, all the log files are fused into one master data table which is used for navigation.

7 Creating a Navigation Algorithm

In this chapter, using the generated lookup table and the influence of state-action-reward on decision process are discussed.

7.1 Approach

Many of the Machine Learning (ML) methods have shown their success in finding optimal solutions to given problems. However, using ML methods has downsides such as long computational time depending on the subject game's complexity. Additionally, as the NPCs try to achieve the most rewarding case in ML applications, some unconventional and unexpected behaviors occur as a result.

When the subject is to create NPC agents which navigate on various difficulty settings, using gameplay recordings gathered from human players can fix these problems. Since the expected result is not an optimal solution but human-like behavior, restricting the agent's actions by collected recordings can solve the problem of the unwanted behaviors. Moreover, using the lookup table as results of training can aid the long computational time problem.

State Action Reward

Reinforcement Learning (RL) is a set of techniques, which adapts learning from past experiences. Every time the agent takes an action, the state of the game environment changes, feedback of difference in the previous state and the current state determine the success of that action and this information is registered in a value function. This mechanism is shown in *Figure 13*.

Later in training, this experience is used for deciding if there is a better action to take in the same situation. In a game with huge state domain, discovering every state takes a very long time and the agent may not even find the optimum solution if the results converge to many other possible solutions.

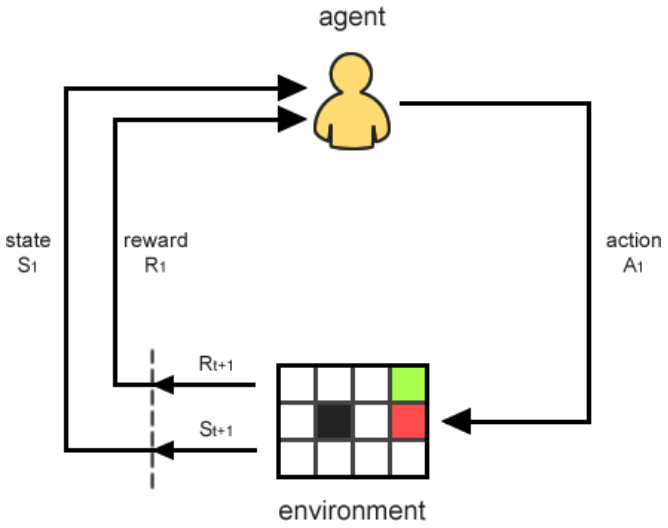

Figure 13: *Representation of state, action and reward mechanism.*

If an agent needs to explore the environment as much as possible In an RL application, the trade-off between exploration and exploitation should be considered. In this concept, an exploration value which is called epsilon value (ε) is used. A high epsilon value should be assigned to the system to increase the possibility of random decisions. Or, to keep going with the best action that is learned so far, the epsilon value should be decreased.

Additionally, a decay value (γ) is used in RL techniques to decrease the reward values. This value is used for encouraging cumulative reward instead of immediate rewards.

The navigation system developed for Vrena is inspired by the working mechanics of this concept. The lookup table created by log sections resemble the reward function of the RL which is a table updated by agents actions and each cell of the table is a state on this table.

Since the AI tool does not need to update the lookup table only with the best actions for the table's states (sectors) it has a list of them to be able to make a variety of choices. In RL this choice is done by a randomly if epsilon value allows it. Yet, in the developed AI tool this movement action is not random but its selection is semi-random. Selection of the log section for navigation is done within a range of difficulty.

Actions taken in the game environment are the selected log sections in this system. When a decision for movement is made the log section is used for movement. This movement as an action should not be confused by the VR controller actions.

Lastly, navigation success value can be compared to the reward value of each state contains after the RL training is done. Yet in the developed AI tool, these values are calculated not depending on next movement but by some of the attributes of that log sections.

The Decision Algorithm

The developed tool has a configuration of fluid difficulty in the range of most successful and least successful log sections in the current state. A pair of values called difficulty range determine the range of the difficulty settings.

This range has a constant minimum value (r_{min}) of 0 and a constant maximum value (r_{max}) of 10. Difficulty selected by the user has a start (d_{min}) and an end (d_{max}) point within the difficulty range. To find the index of a log section that falls between these start and end points, the number of logs (n) is divided by the max difficulty to calculate the value of each step in the difficulty space (p). A rounded random value between multiplication of start and end difficulty values with step value gives the index of the log section from desired difficulty range. Here, success value does not affect choosing the log section, but the order of the log sections by success value matters. This process is shown in the *formula (3)*.

$$p = (n-1)/r_{max}$$
$$index = Round(Random(d_{min} * p,\ d_{max} * p)) \quad (3)$$

When the game is started, an agent with active navigation system initiates a loop of the decision-making process. The current state of the game environment is observed and a random action within the bounds of difficulty range is chosen. In this context, log sections are equivalent to actions in RL. Selected log section is played and when it is done, current step is observed and another action is taken. This process goes on until the game is over.

Algorithm 1 Decision Algorithm

```
1: procedure RunAgent
2:     gameOver ← false
3:     currentAction ← new logSection
4:     dr ← difficulty range [0-10]
5:
6:     while gameOver = false do
7:         state ← getCurrentState()
8:         actionList ← lookUpTable[state]
9:
10:        while actionList = null do
11:            neighbourState ← getNextClosestState(state)
12:            actionList ← lookUpTable[neighbourState]
13:
14:        currentAction ← getRandomAction(actionList, dr)
15:
16:        gameOver ← runAction(currentAction)
```

Figure 14: *Pseudo code of decision algorithm.*

If there is not any action listed in the lookup table for the current state of the game, a method to detect the closest state is run. Found closest state replaces the current state and the decision algorithm is executed with the new state. This way it is ensured the NPC agent does not get stuck in a state without actions. This process is also shown in *Figure 14*.

One important factor is to make the decision-making process happen as fast as possible. If the process of retrieving the best match from game logs is slow, it is possible that the AI agent makes the decisions after the current situation is over and it lags the actions. In fast-paced games, even small amount of latency can cause the quality of gameplay to decrease. This is a big problem as it could be disregarded in non-time-depended or slow games. The lookup table generated from refined game log prevents this issue by reducing the amount of data needed for decisions.

7.2 Summary

In a continuous state space game, it may be difficult to find an optimum solution for navigation problem using RL. It requires a long time for training and may not even converge to a plausible result at the end. Working on a game environment sliced into sections makes this job easier.

It is assumed that the collected logs are the training results and this data is used for making movement decisions. Compared to RL, success values of the log sections cover reward of the action and within the difficulty range, a random action is taken.

Movements upon decision are continuously done in the main loop while the game runs. Depending on the new state the player is in, a new action is pulled out of the lookup table within a difficulty range and it is played to navigate the NPC agent.

8 Evaluation

In this section developed Log AI Tool is discussed and explain its components are explained. Conducted player survey and its results are discussed in detail to conclude what can be done to improve the system with the collected feedback.

8.1 The Product – Log AI Tool

A tool called Log AI Tool is developed as a product of this thesis to test the idea of navigating NPC agents using game logs. The tool is developed in Unity game engine for Vrena. The source code of the log AI tool, Unity project, log files and lookup table outputs can be found in *Appendix I*.

This tool is a game object that consists of 5 different components. As it is displayed in *Figure 15*, these parts are called Agent Manager, Visualizer, Recorder, Refiner, and Navigator.

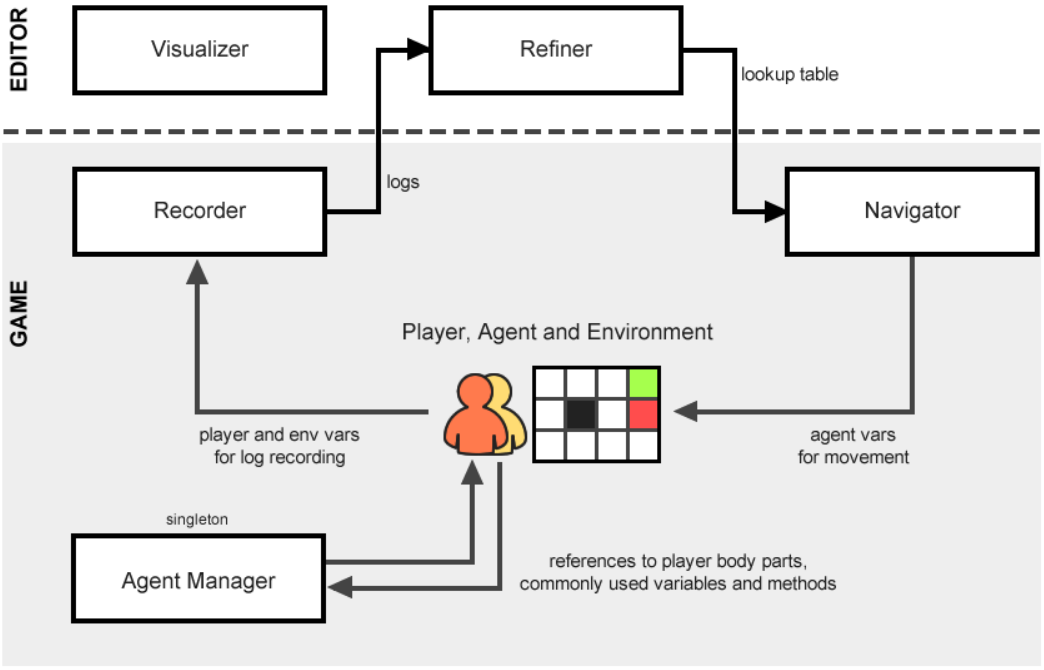

Figure 15: *Representation of the components of the Log AI System.*

The game object containing these components are attached to the tested AI agent. Body parts of the agent moved by VR controllers and other environmental variables are configured on these components.

Agent Manager

Agent manager class is a singleton class that contains the variables that are reached from other components of the tool and used frequently. Namely, it is the component which is used as a single collection of commonly used variables and functions.

The interface of the component provides the user the “Unit Size” value. This value defines the edge length of the sector cubes the game environment is sliced into. This variable is an important attribute of the system and is used in all the other.

A list of object selectors under the “Player Components” label is exposed to the user. These objects are the moving body parts of the player and the trackers of the VR set. Player’s body is mainly made up of four parts and those are two hands, the head and the body of the player object. Hands are controlled by the VR controllers and body and the head is controlled by the VR headset. Pointers are the objects which enable Vive to track the hand controllers. The interface of the Agent Manager component is shown in *Figure 16*.

This class contains functionality for registering the actions to a dictionary by names as keys and functions (Action in Unity) as values. These actions correspond to the left and right controller’s trackpad button which represents firing and releasing the grappling hooks. Names of these actions are “PressRight”, “ReleaseRight”, “PressLeft”, “ReleaseLeft”.

Another function of the Agent Manager class is moving the agent’s body parts from one log line to another. Move function of this component takes the logs lines to play and linearly interpolates the values of each body part one by one. Since this feature is used in both Visualizer and Navigator classes, this feature is moved in Agent Manager to reduce the code repetition.

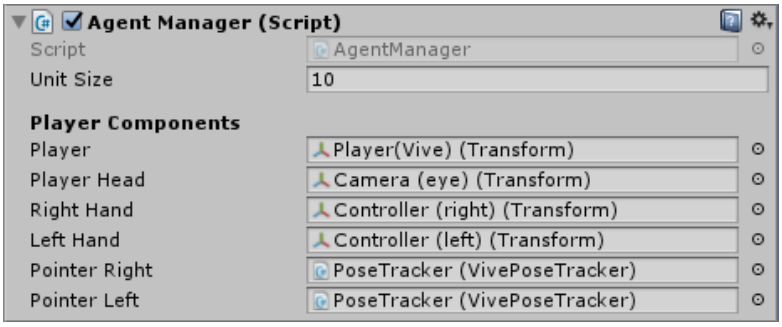

Figure 16: *Agent Manager component interface in Unity inspector.*

Lastly, Agent Manager contains a function for disabling VR controls. This functionality has been necessary since agent receives its position and rotation information from the VR set during the runtime. When the agent is navigating from Navigator class or replaying a log file from Visualizer class its connections to VR camera and controllers are cut, this way the information retrieved from log files or lookup table can be applied without any interruption.

Visualizer

Visualizer class is the only part of the system that is purely for debugging purposes which have no direct impact on the working cycle of the tool. The component helps the user visually control the tool’s area of effect, view and replay log files and see the density of value count in the lookup table. The interface of the component is shown in *Figure 17*.

To display the visualizer changes “Debug Area” toggle should be switched on. The graphical changes created by this component are only active in the editor window but not in game mode. To display this graphical changes game should be running.

The size of the tool's area of effect can be determined using the variables under the "Bounding Box Variables" label. Size of the bounding box is determined here. The pivot point of the bounding box is configured here as well, which is an offset value for adjusting the position of the bounding box to the map object. The bounding box can be colorized and viewed in Unity without running the project.

The sector cube the player currently is in can be displayed by switching the "Highlight Active Cube" toggle. This option helps the user see which sector of the map the player is in while navigating.

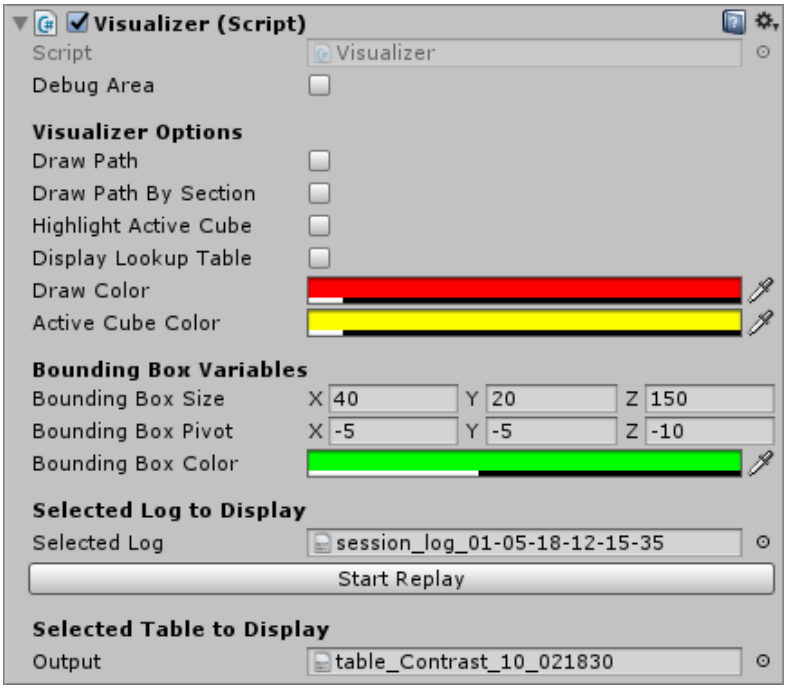

Figure 17: *Visualizer component interface in Unity inspector.*

The user can select one of the logs from the log pool using the selector under "Selected Log to Display" label. Movement trajectory of the selected log can be displayed by switching the "Draw Path" toggle on. And if "Draw Path" and "Draw Path by Section" toggles are switched on together, the movement trajectory is colored differently for each log section of the log file. The actions taken in the game log is also printed by their name at the occurring locations and marked by black spheres. The target points of the grappling hooks are displayed by red spheres on the walls. A scene showing a path drawn by log sections can be seen in *Figure 18*.

In the visualizer component, the user can select a lookup table to display its density and coverage. The table is selected using the selector under "Lookup Table to View" label. Once a lookup table output is selected "Display Lookup Table" toggle can be switched on. This action highlights each sector cube in the bounding box depending on the number of values that that cell has. The cells are highlighted with yellow color and the cells that have fewer log sections are more transparent.

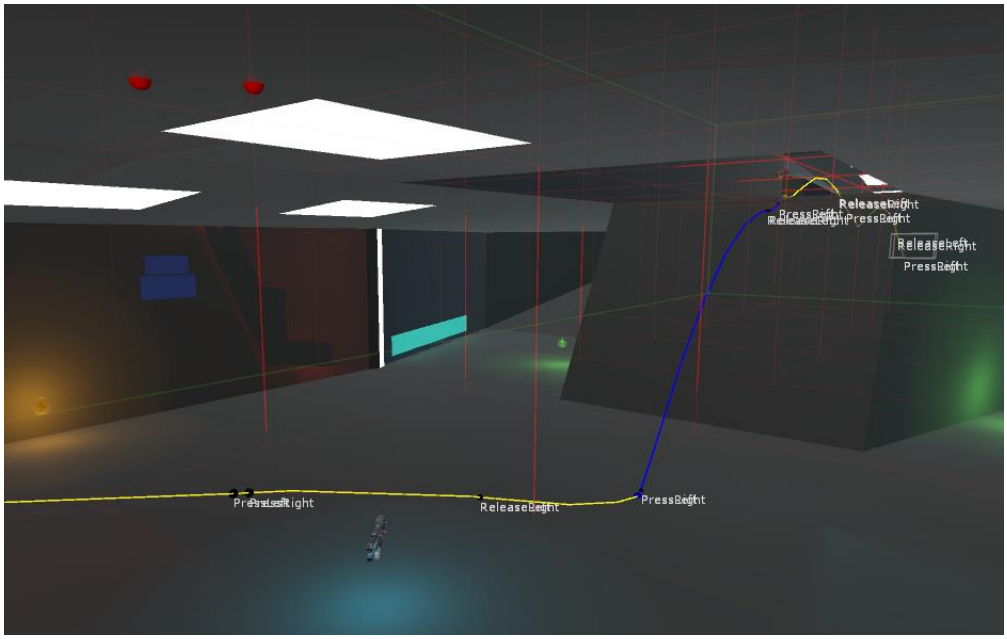

Figure 18: *A log file trajectory is drawn by sections.*

The selected log file can also be replayed from the visualizer component. The interface of the component contains a “Start Replay” button. Once clicked, the agent is relocated to the start point of the selected path and the actions are replayed until the log ends.

Recorder

Recorder class contains the functions for starting and stopping log recording process, checking the player state, and recording the log lines.

The inspector of the Recorder component exposes the “Logger Tick” variable to the user. This variable is the time delay between each log line. When recording starts Logger function is invoked repeatedly every Logger Tick until the recording is stopped. The interface of the Recorder component is presented in *Figure 19.*

Figure 19: *Recorder component interface in Unity inspector.*

The component contains a button for starting and stopping the recording process. This button helps the user start and stop log recording process from the interface. This functionality is also given to the Grip Button of the Vive controller, so the player can start and stop recording process while playing the game.

The major functionality of this component, the logger method, is contained in this class with its two variations. In this function variables of LogLine objects are filled and saved when

the recording ends. The LogLine object contains necessary variables of the player's body parts and game states.

When logs lines are being recorded, they are also checked for state changes in this class. Change in the state determines the creation of log sections, and the saved log files are stored as a set of log sections instead of log lines. Names of the log files follow the naming convention of "session_log_Date-Time.json".

Refiner

Refiner class contains the functions for utilizing recorded log files and preparing them for use of Navigator component.

The interface of the Refiner component contains only one "Collect Logs" button that starts the refinery process and creates a lookup table file as a product. The button calls a series of functions which handles a different step of this process. The interface of the Refiner component is presented in *Figure 20.*

Initially, the log files are collected from the stored directory. Then the log sections from the log files are extracted and added to a list to create a pool of all the log sections.

Next step of the process is to set the success values of the log sections. Each log section in the pool is visited and a success value is assigned to "successValue" variable of that log section. This success value is calculated according to the predetermined criteria.

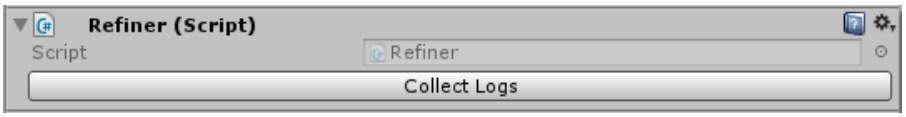

Figure 20: *Refiner component interface in Unity inspector.*

After all log lines receive their success value the log lines with low success value are removed from log pool. First, the log lines with 0 score are removed from the list to homogenize the pool since the number of log lines with 0 score may be many. After this operation, second standard deviation of the log pool is calculated according to success values and the log lines below lower second standard deviation (-2σ) are removed from the pool.

Items of the refined log pool are sorted in the next step according to the success values of the log lines.

Finally, sorted log lines are added to the lookup table. The lookup table is an associative array structure which has the sector of the log sections as keys and a list of log sections as values. Each element in the log section pool is visited and they are added to the lookup table by their sectors. Once all the log sections are added to the lookup table, an output of this data is created and saved. The name of the output file follows the naming convention of "Table_NameOfTheMap_UnitSize_HHmmss.json". The created output is used by the Navigator component to move the AI agent.

Navigator

Navigator class contains the functions for finding a fitting log section and using that log section to navigate the agent. The interface of the Navigator component is presented in *Figure 21*.

The interface of this component provides access to a two-ended minimum-maximum slider that is called “Difficulty Range”. Difficulty range value can be configured by using the ends of the slider or the input areas above the slider. The user can select a difficulty range within 0 and 10 with a minimum range of 1. These values are used when the agent decides on selecting a log section from the log table.

To mobilize the agent a lookup table output should be selected from the selector under the “Selected Lookup Table” label on the inspector. After a table is selected the “Play Agent” button should be pressed to activate the agent. Once the agent is active and moving on the map, the “Agent is Online” toggle is switched on. To stop the agent this toggle should be switched off.

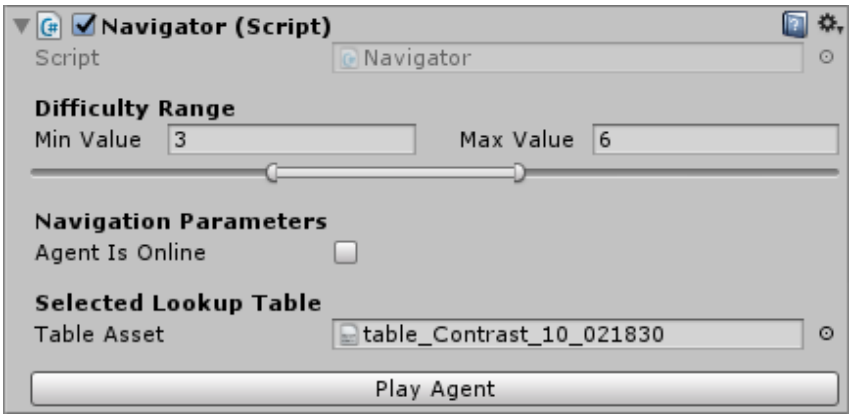

Figure 21: *Navigator component interface in Unity inspector.*

Movement decision for the agent is done in the Update method of unity, which is a basic game loop used in every Behaviour component. If the agent is online and it is not moving at the moment, a decision is made and the agent moves accordingly. Next decision waits until the current movement is over. This process goes on unless the agent is set offline.

When a decision is to be made, the component checks the current cubic sector the agent is in. This info is used as a key on the lookup table and the value list from the corresponding cell is taken. The difficulty range is used here to determine the index of the log section from this list. Within the difficulty range, a random log section is picked as the next movement decision and it is played.

In case of being in a sector that has not any log sections in the corresponding table cell, there is a supporting functionality for searching for the closest neighbor sector. Until a neighbor with table cell with log sections is found similar decision steps are taken there. This operation avoids the situation that the agent is stuck at a point of the map that is not covered by the lookup table.

8.2 Reaction Survey

A survey of 30 people is conducted to measure people's ability to distinguish the bots from human players and evaluate the believability of the developed AI system. To collect the results of the survey in a short time the survey is done online instead of inviting attendees to the VR laboratory and having them to play the game. Video footage of the human and AI players are included in this survey to show the attendees how the game is played. Survey sheet, responses, and video content can be found in *Appendix II*.

Since Vrena is an FPS game, players see their opponents from their personal view. For this reason, a monitoring system is added to the game map that has cameras following the player to record videos.

The monitoring system consisted of 10 box colliders covering all the map and 10 cameras, one for each of collider areas which are shown in *Figure 22*. When the player enters one of the collider boxes the camera of that area gets activated and it follows the player. Similarly, when the player exits one collider box area, the camera of that area gets deactivated and the camera stops following the player. This way it is made sure there is always only one active camera.

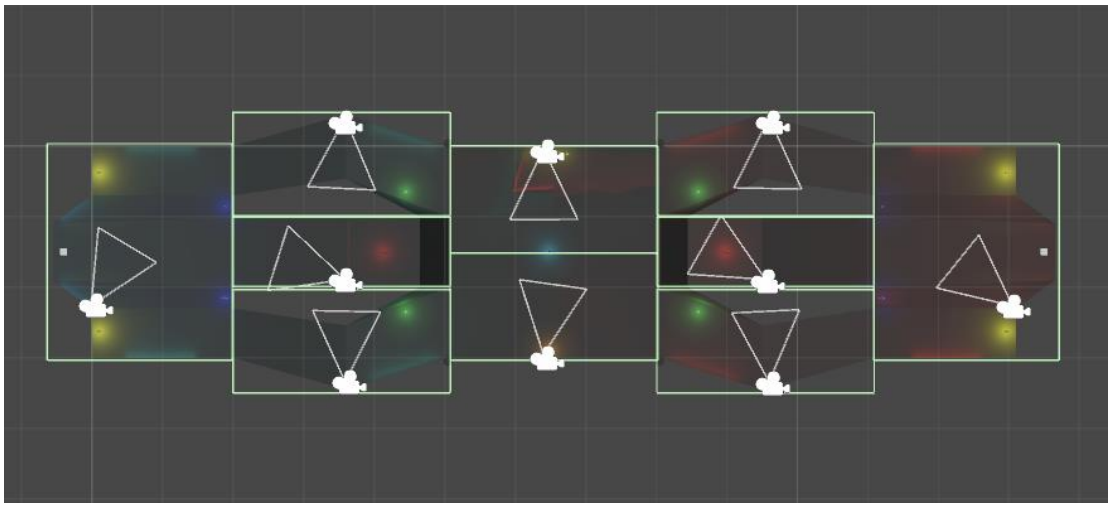

Figure 22: *Collider boxes and cameras for recording player and agent movement.*

Using the created monitoring system six videos were recorded which have a footage of a player moving from the blue base (left side on *Figure 22*) to the red base (right side on *Figure 22*). Two of these videos are replays of gameplay recordings and the other four are bots playing the game using navigation system.

Questions in the survey created according to Likert scale, where answers to the questions have a defined palette of agreeability or opinion. Likert (1932) developed this way of measuring responses of people to be able to collect their statements about specific topics according to how much they agree or how much it affects them.

As it is suggested by Joshi et al. (2015) [29] to give more gradually different options to the attendees and let them choose clearly opposed alternatives, the seven-point scale is used for questions. The scale had two opposing ends and had symmetrical options where attendees

had the chance to give a neutral answer. For all the questions sample answers for Likert scale questions[24] are used.

The survey consisted of three question types and six questions, where last four questions were about video footages. Addition to the questions, instruction texts, images and two videos of bots playing the game provided to the attendees to familiarize them with the concept and give them a point of reference about how the bots are playing the game.

Firstly, questions of how often do they play video games and how familiar are they with VR technology are asked. Later four questions about video footages and if the player in action is a bot or a human is asked. Following these four questions, an explanation for what influenced their decision is asked. Two of these videos were of bots and two were human player replays.

First two questions of the survey aimed to collect information about attendees' video game play frequency, shown in *Table 1*, and familiarity with VR, shown in *Table 2*. When the success of the predictions and the answers to these questions are compared, it is seen that people who play games more had a better ability to identify bot players. On the other hand, attendees' familiarity with VR did not have an impact on these predictions.

Table 1: *Frequency of video game play of survey attendees.*

How often do you play video games?						
Every Day	Usually	Frequently	Sometimes	Occasionally	Rarely	Never
20%	6.7%	33.3%	13.3%	23.3%	3.3%	0%

Table 2: *Survey attendees' familiarity with VR.*

How would you define your familiarity with Virtual Reality technology, equipment and, games? (I use a VR set every week = Extremely, I never used a VR set = Not at all)				
Extremely	Moderately	Somewhat	Slightly	Not at all
16.7%	16.7%	26.7%	36.7%	3.3%

Last four questions showed that a bot and a human player were easy to detect, however other two players were not detected by the survey attendees successfully. The results of all the guesses on human and bot players are shown in *Table 3*.

Results of the questions with video footage shown that attendees were likely to distinguish bots from human players as shown in *Table 4*. About 53% of the attendees successfully identified the players in the videos as human or bot, and about 37% of the attendees were unsuccessful to distinguish bots from the humans and gave wrong answers. Rest of the attendees, 10%, were unable to decide.

[24] http://www.marquette.edu/dsa/assessment/documents/Sample-Likert-Scales.pdf

Text responses of the attendees provided necessary information about extracting problematic patterns. There are two sets of patterns, those indicate a player is, in fact, a bot and those make a player look like a human. And additionally, there is an intersection of these two sets which some people thought they were bot-like actions and some people thought otherwise.

Table 3: *Guesses of survey attendees.*

What is your opinion on this player?							
	A bot	Most likely a bot	Likely a bot	I don't know	Likely a human	Most likely a human	A human
Human 1	20%	13.3%	10%	6.7%	23.3%	20%	6.7%
Bot 1	20%	20%	20%	10%	20%	6.7%	3.3%
Human 2	10%	3.3%	13.3%	10%	26.7%	20%	16.7%
Bot 2	16.7%	10%	13.3%	10%	20%	20%	10%

The single pattern survey attendees thought that could only be done by human players were hand gestures, even though one of the exemplary bot videos had a similar action in it. This shows that players believe that bots would not make such movements but make robot-like rigid movements.

The intersecting part of two sets of patterns contained aiming and hitting good spots with grappling hooks, namely good accuracy while playing the game. Almost equal number of people believed good aim could only be an ability of a human player or a very well programmed bot. Since the exact target points of player hits from the logs are used, both good and bad aimed actions are actually taken from the recorded players.

Even though the patterns of "taking quick movement actions" and "thinking for a moment to decide where to aim" are just the opposite of each other, attendees believed those were both human and bot players' attributes. Some attendees claimed that bots spent some time to calculate where to aim, and human players just knew where to go. On the contrary, some attendees claimed bots made quick decisions to move but human players needed to think before acting.

Table 4: *Distribution of right and wrong guesses.*

	Certain	Most likely	Likely	Total
Right	15.025%	17.500%	20.825%	53.35%
Wrong	10.825%	10.825%	15.825%	37.475%
Neutral				9.175%

The information that is most useful for improving the future versions of the system was the patters that made survey attendees spot out bot players. One of the most mentioned patterns was player's position sways (repositioning) between jumps. This problem is caused by human player's positions in the room-scale area while gameplay is being recorded. If the position of the player is far away in the selected log section position of the agent sways to the new position.

The second issue was clipping through walls, which is caused by Player Object's structure. The body collider of the player is always in the center of the play area and if the player is, again, away from the center of the play area it clips through walls.

Another mentioned issue was aiming while sight is blocked. In the videos, 10 units are used to define each sector, which is rather large and provides average accuracy for bots. Depending on the position of the bot in a sector, it may try to move even if its sight is blocked. This happens due to the selected log section being actually taken from a side of that sector where the human player was not blocked. This issue can be solved by decreasing the size of the sectors.

Last problematic pattern was sudden aim corrections of the player. This is another problem that will be solved after modifying the player object, in the current version of the player object the hands are given 50 degrees of initial angles, and this creates additional rotational calculations for making hands look at target points while moving. To cover this, the instant rotational change applied to the hands. However, if the position of the player in the current sector is away from where that piece of log section was recorded sudden rotation change in the hands become apparent.

8.3 Examining Results

The five component tool developed in Unity engine can help a single agent move like a human player, using a data file that is harvested from gameplay recordings. The tool also has features to help developers and configurations about the system.

Using this tool, gameplay recordings are collected, refined and the output of the system is used for navigating an agent in the game. As a result, the agent is able to make decisions those resemble human player movements.

Video footage of agent movements and human player movements are recorded and used in a survey to evaluate the success of the system. The survey resulted that attendees were able to distinguish two of the players in the subject footage and were not in the other two videos. The success of distinguishing bots were about 53%, and 37% of the attendees failed to spot the bot players.

Depending on this information it can be assumed that the current AI system has an above average success to resemble human-like movement. Additionally, the information the attendees provided addressed several issues that create uncanny behavior and made them notice the bots.

As a conclusion, problematic patterns noticed by the survey attendees can be solved by modifications on the system and a more consistent movement function can be developed for the movement at the current state of the project.

9 Conclusions

In this thesis project, an NPC navigation system based on gameplay recordings is developed. For the development of the AI system, Unity game engine is used and the product of this project will be used for the case study game Vrena. The system handles the whole process step by step starting from recording the logs to using the output data to navigate the AI agents.

The research has been an initial step to create a complete AI system for Vrena. Using this system as a foundation will provide the motivation to develop the system further, modify and expand the capabilities of the AI agents in the game.

9.1 Answers to the Research Questions

Depending on the experience obtained during the research project, the research questions can be answered as follows:

RQ 1. How can we implement a system that uses game logs for NPC movement and decision?

The initial step should be creating a customized log recorder for collecting relevant data from the game environment and the players. Recorded logs should be divided into smaller pieces which can be used as action units. These actions are ordered by their success on specific criteria and aggregated into a data file by their starting points. When this data is to be used for movement, the selected action leads the agent into a new state in the game environment. Using the value of this new state, another action is retrieved from the data file and the process follows the same procedure until it is terminated.

RQ 2. How can we create a configurable log recording and processing system, what measures should be taken into consideration while recording?

The process of "navigating NPC agents using game logs" divided into sub-processes. A component for each of these processes is developed with the help of Unity game engine's component-based architecture. The components have interfaces where developers can expose the desired variables and dynamically configure them.

From the three main components of the system, the Recorder and the Navigator work during runtime of the game sessions. Thus, these two components needed to run as fast as possible. Recording least number of variables and recording those with least frequency provides the desired speed in this process. Recording by the controller action is a must and cannot be eliminated. However, recording by time is an option to the developer which actually determines the quality of cosmetic appearance of player gestures.

For decision making the right unit size for sectors must be determined. Too big sectors create faulty movement decisions and too small sectors make it difficult to reach the maximum coverage, needing more log files per lookup table. Size of the sectors must also comply with the detail of the map design.

Storing log files and the lookup table is done using JSON file format for being a commonly used format and saving the files in a human-readable format in small size.

RQ 3. How can we benefit from modern AI techniques when we create this system for NPC movement and decision.

One of the methods used is slicing the game environment into sectors for covering the states, which was a technique used in some of the previous research. This helped to decrease the number of log files needed to cover every possible decision and ease the decision-making process in the same way by creating a look-up table for all the possible decisions. When the agent is in a state, it makes a decision depending on what is available in that state. This helps to achieve a nonscripted behavior system that can be expanded and changed by the new content.

Another method used is a custom modification of a state-action-reward concept. This method is inspired by the working mechanics of Reinforcement Learning algorithms. When the developed system and this concept is compared, it is assumed that the created log files regarded as the training process, the lookup table as the value function, sectors in the game environment as the states, success values of the log sections as the reward value, log sections as the actions and the difficulty range of decision as the exploration.

The system calculates a movement success value for each log section, this value determines the quality of that decision. Since the system does not aim to choose the optimal path but have diversity in the decisions to resemble a human-like behavior, a difficulty range used for making random decisions from that range. By this feature, a fluid difficulty setting is created and the system can be modified for adaptive gameplay.

9.2 Future Work

Result and the product of this thesis project is the foundation of a system for NPC AI based on game logs. And the project will be worked on further for the case study game Vrena for providing it a better multiplayer gameplay experience. There will be many modifications and new features to add to the system and some of these improvements are listed as:

- ***Adding new player and environment states to the log files****:* So far, only position state of the agent to make decisions is used. However, players may show different behaviors depending on the type of weapons they use, amount of health they. These states will be added to the game logs to expand the abilities of the bots.
- ***Getting success value dynamically:*** Currently, the success value for movement is set into the log sections in the lookup table. After introducing other player and environment states to the system, we will need to have success values depending on those states as well. Rather than using predetermined success values, a behavior for the current state can be selected and best movement action can be found depending on that state with a success calculation.
- ***Creating a better log sectioning algorithm:*** There are many different play styles when VR games are played. In Vrena some people play by jumping and some people by dragging themselves on the ground. Also, the frequency of actions they take may vary as well. Log sectioning algorithm of the system should be modified to use these types of movement and categorize the playstyles.
- ***Uploading logs to the cloud and refine remotely:*** To expand the quality and abilities of the agents, gameplay will continuously be recorded from the active players. To automate the process, the recorder will upload the game logs to the web. All the logs collected in the server will be refined and saved as an output file in the cloud. The game will check for new output files to update the lookup tables.

10 Terminology Table

AI	Artificial Intelligence
NPC	Non-Player Character
ML	Machine Learning
RL	Reinforcement Learning
CI	Computational Intelligence
VR	Virtual Reality
XR	Cross-Reality
FPS	First Person Shooter
AOE	Area of Effect
DM	Death Match
CTF	Capture the Flag
ML	Machine Learning
FSM	Finite State Machine
RTS	Real-Time Strategy
NN	Neural Networks
GA	Genetic Algorithm
QL	Q-Learning
XML	Extensible Markup Language
JSON	JavaScript Object Notation

11 References

[1] G. N. Yannakakis, "Game AI Revisited," in *Proceedings of the 9th conference on Computing Frontiers*, Cagliari, Italy, 2012.

[2] N. Yee, "Motivations for Play in Online Games," *CyberPsychology & Behavior,* vol. 9, no. 6, 2006.

[3] J. Vanian, "Virtual Reality Survey Highlights Lingering VR Issues," Fortune, 22 June 2017. [Online]. Available: http://fortune.com/2017/06/22/virtual-reality-developers-naseau-survey/. [Accessed 18 03 2018].

[4] D. Wehr and J. Denzinger, "Mining game logs to create a playbook for unit AIs," in *IEEE CIG 2015*, Tainan, Taiwan, 2015.

[5] I. J. P. S. Kazmi, "Action Recognition for Support of Adaptive Gameplay: A Case Study of a First Person Shooter," *International Journal of Computer Games Technology,* vol. 2010, p. 14, 2010.

[6] C. Thurau, G. Sagerer and C. Bauckhage, "Imitation Learning at All Levels of Game-AI," in *Proc. Int. Conf. on Computer Games, Artificial Intelligence, Design and Education*, 2004, pp. 402-408.

[7] R. Fine, "Unity Blog," Unity, 11 08 2017. [Online]. Available: https://blogs.unity3d.com/2017/08/11/unityscripts-long-ride-off-into-the-sunset. [Accessed 19 03 2018].

[8] Arthur Juliani, "Unity Blog," Unity, 19 September 2017. [Online]. Available: https://blogs.unity3d.com/2017/09/19/introducing-unity-machine-learning-agents/. [Accessed 20 12 2017].

[9] "Unity Documentation," Unity, 2017. [Online]. Available: https://docs.unity3d.com/Manual/VROverview.html. [Accessed 01 04 2018].

[10] J. P. Hawkins, "Unity Blog," Unity, 10 02 2016. [Online]. Available: https://blogs.unity3d.com/2016/02/10/valve-brings-steamvr-to-the-unity-technologies-platform. [Accessed 13 05 2018].

[11] A. Vlachos, "Advanced VR Rendering," in *Game Developers Conference*, San Francisco, 2015.

[12] M. Usoh, C. Alberto and M. Slater, "Presence: Experiments in the Psychology of Virtual Environments," 1999.

[13] D. Nield, "PopSci," 29 August 2017. [Online]. Available: https://www.popsci.com/choose-right-vr-gear-for-you. [Accessed 26 11 2017].

[14] J. F. Ian Millington, "Chapter 1 - Introduction," in *Artificial Intelligence for Games*, Burlington, Elsevier, 2009, pp. 5-7.

[15] S. Madhav, Game Programming Algorithms and Techniques, A Platform-Agnostic Approach, Indiana, USA: Addison-Wesley, 2014.

[16] J. W. Daniel Johnson, "Computer Games With Intelligence," in *Fuzzy Systems, The 10th IEEE International Conference on 2001*, vol. 3, IEEE, 2001, pp. 1355-1358.

[17] M. Gallagher and M. McPartland, "Reinforcement Learning in First Person," *IEEE Transactions on Computational Intelligence and AI in Games,* pp. 43 - 56, 2011.

[18] B. Gorman, C. Thurau, C. Bauckhage and M. Humphrys, "Bayesian Imitation of Human Behavior in Interactive Computer Games," *Proceedings of the 18th International Conference on Pattern Recognition,* 2006.

[19] M. Gallagher and M. McPartland, “Interactively Training First Person Shooter Bots,” *IEEE Conference on Computational Intelligence and Games,* pp. 132-138, 2012.

[20] S. Hladky and V. Bulitko, “An Evaluation of Models for Predicting Opponent Positions in,” *Symposium on Computational Intelligence and Games,* 2008.

[21] P. G. Patel, N. Carver and S. Rahimi, “Tuning Computer Gaming Agents using Q-Learning,” in *Federated Conference on Computer Science and Information Systems,* Szczecin, 2011.

[22] D. Conroy, P. Wyeth and D. Johnson, “Modeling Player-like Behavior for Game AI Design,” in *ACE '11 Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology,* Lisbon, Portugal, 2011.

[23] M. Sheehan and I. Watson, “On the Usefulness of Interactive Computer Game Logs for Agent Modelling,” in *Pacific Rim International Conference on Artificial Intelligence,* Hanoi, Vietnam, 2008.

[24] M. Kopel and T. Hajas, “Implementing AI for Non-player Characters in 3D Video Games,” in *Springer International Publishing,* 2018.

[25] L. Wang, Y. Wang and Y. Li, “Mining Experiential Patterns from Game-Logs of Board Game,” *International Journal of Computer Games Technology,* vol. 2015, p. 20, 2015.

[26] A. Tveit and G. B. Tveit, “Game Usage Mining: Information Gathering for Knowledge Discovery in Massive Multiplayer Games,” in *Proceedings of the International Conference on Internet Computing,* CSREA Press, 2002, pp. 636-642.

[27] N. Nurseitov, M. Paulson, R. Reynolds and C. Izurieta, “Comparison of JSON and XML Data Interchange Formats: A Case Study,” in *Proceedings of the ISCA 22nd International Conference on Computer Applications in Industry and Engineering,* San Francisco, 2009.

[28] D. Buckley, K. Chen and J. Knowles, “Rapid Skill Capture in a First-Person Shooter,” *IEEE Transactions on Computational Intelligence and AI in Games,* vol. 9, no. 1, 2017.

[29] A. Joshi, S. Kale, S. Chandel and D. K. Pal, “Likert Scale: Explored and Explained,” *British Journal of Applied Science & Technology,* vol. 7, pp. 396-403, 2015.

12 List of Figures

13 List of Tables

Appendices

I. Source Code

The source code of the project can be found in the attached **SourceCode.zip** file.

Contents of the attachment:

- **LogAI Unity Package:** A file that can be opened by Unity game engine and should be imported into an existing project to be used.
- **LogAI Folder:** The files of the unity package, where **.cs** files containing the code can be found.
- **Logs Folder:** Log files used in the research can be found in this folder.
- **Tables Folder:** Lookup table output files can be found in this folder.

URL to the source code: https://github.com/srcnalt/Log-AI-System

II. Survey

Files related to the reaction survey can be found in the attached **Survey.zip** file.

Contents of the attachment:

- **Survey_Form.pdf:** The survey form that contains the questions of the conducted reaction survey.
- **Ammended_Survey_Form.pdf:** The edited survey form. An additional info line added to the second question of the survey.
- **Survey_Answers.xlxs:** The worksheet that contains the answers to the reaction survey.
- **Survey_Videos Folder:** The folder that contains the video footage of the human and AI players, which were used in the reaction survey. It contains two human player and 4 AI player footage, a total of six videos.

III. License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Sercan Altundaş**,

herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis

NPC AI System Based on Gameplay Recordings,

supervised by Margus Luik,

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **23.05.2018**