Politechnika Wrocławska

Faculty of Computer Science and Management
Field of study: **COMPUTER SCIENCE**

Bachelor Thesis

FIRST PERSON SHOOTER WITH PROCEDURAL GENERATED LEVEL MADE IN UNREAL ENGINE

Leszek Szymczak

keywords:
Procedural generation Multiplayer
Computer game Unreal Engine 4
First Person Shooter

short summary:
In this thesis I have presented a project and an implementation of a video game using Unreal Engine 4. The game is a 3D multiplayer First Person Shooter than utilizes procedural generation for level creation.

Supervisor	Marek Kopel PhD ……..		
	Title/ degree/ name and surname		
The final evaluation of the thesis			
Chairman of the Diploma Examination Committee	.....	.....	…….
	Title/ degree/ name and surname	*grade*	*signature*

For the purposes of archival thesis qualified to: *
a) Category A (perpetual files)
b) Category BE 50 (subject to expertise after 50 years)
* Delete as appropriate

stamp of the faculty

Wrocław, 2020

Abstract

Goal of the thesis was creation of a First Person Shooter with procedural generated level. When approaching said project an idea arose to make it even more unique by adding multiplayer to said game.

Thesis consists of introduction that aims to make the reader more familiar with the presented problem, overview of existing solutions, project and implementation descriptions, ending with achieved results and a short summary. Thesis describes in depth the reasoning for certain approach to most important game elements as well as the process of their implementation within Unreal Engine. Alongside results are presented plans for possible future development.

The end product is a game that can be played from two up to six players and has clear way for future improvements.

Streszczenie

Celem pracy było stworzenie strzelanki pierwszoosobowej z proceduralnie generowanym poziomem. Podczas początków prac nad projektem pojawił się pomysł żeby uczynić tę grę jeszcze bardziej unikalną poprzez dodanie do niej rozgrywki wieloosobowej.

Praca składa się z wstępu mającego za zadanie zaznajomić czytającego z przedstawionym zagadnieniem, omówienia istniejących rozwiązań, opisu projektu oraz implementacji, kończąc się osiągniętymi wynikami i krótkim podsumowaniem. Praca dogłębnie opisuje powody stojące za konkretnym podejściem do najważniejszych części składowych gry oraz proces ich implementacje w silniku Unreal Engine. Razem z wynikami zaprezentowane są plany na dalszy rozwój aplikacji.

Końcowym produktem jest gra która może być grana przez dwóch do sześciu graczy i ma jasno wytyczoną ścieżkę przyszłego rozwoju.

Contents

1. Introduction

Video game industry is relatively new and extremely quickly developing branch of the entertainment industry. With origins in early 1970s thorough fifty years of it's existence it has changed drastically. According to researches current value of the global video game market is worth around 150 billion dollars and if the trends remain it should continue to increase[1]. In this ever-changing environment one of the ways game developers choose to follow in order for their game to succeed is creating a game combining characteristics of more than one of the well established genres. Such approach can be observed usually in Indie games that are games developed and published independently by developers without the help of major publishers[2]. However before moving forward I would like to introduce a couple of key terms that will be used throughout this thesis.

1.1. Terminology

1.1.1. First Person Shooter

First-person shooter (FPS) - is a video game genre focused on gun (and other weapons) based combat that is perceived by the player from first-person perspective, that is player sees the world inside the game from protagonist's eyes. Released in May 1992, Wolfenstein 3D is believed to be the first modern FPS with DOOM released in 1993 these two games gave birth to and popularised said genre[3]. Nowadays FPS genre is one of the most popular video game types. According to statistics in 2018 shooters (that term usually refers to both FPS and TPS games out of which first category is much more popular) were 20% of all games sold in USA, ranking as the second most popular genre(Figure 1.1,[4]). Present-day First-person shooters are a very complex genre that consist of many different subgenres such as tactical shooters like Tom Clancy's Rainbow Six Siege that put high emphasis on tactics and are played on a small maps by two teams competing to achieve their contradicting objective or Battle royales that are played on a huge map usually filled with random weapons for players to find putting an emphasis on players adaptability and agility where the goal is to survive.

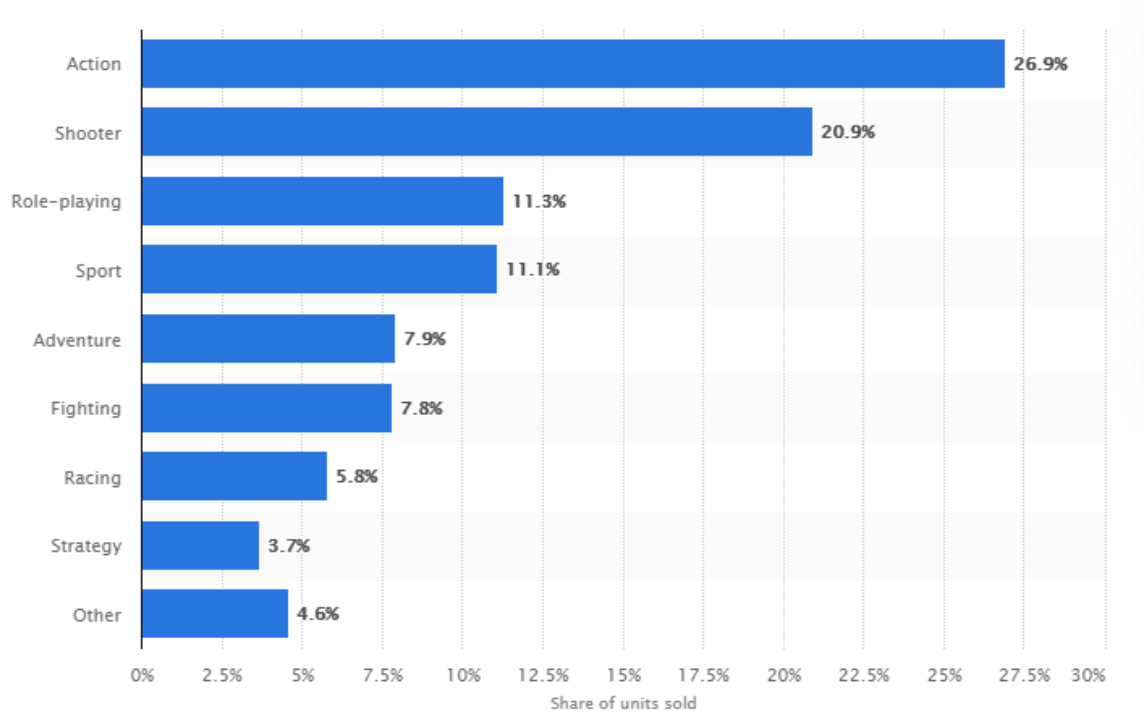

Fig. 1.1. Genre breakdown of video game sales in the United States in 2018, access 2020.12.1 [4].

1.1.2. Multiplayer

Multiplayer - refers to games that can be played by multiple people in the same game environment at the same time. Said players may compete against one another or cooperate to overcome challenges presented by the game. Multiplayer games can be divided in to types, ones with non-networked/local multiplayer and the ones with networked multiplayer. Representatives of the first group are some of the first games ever created, for example Pong was originally played by two players using the same console. However there are still modern video games that include local multiplayer with players sharing one keyboard or having multiple controllers plugged in to the same device the second group is far more popular. First networked multiplayer games were played on LAN networks which allowed players to play on different devices however still required them to be within the range of same network. With the development of the Internet followed development in networked multiplayer. Even though playing offline games' is still more popular than playing online ones, online games popularity is constantly growing and according to researchers it will continue to increase.(Figure 1.2,[1]). Online multiplayer has been inseparable with previously mentioned FPS games ever since first DOOM and even though there are few exceptions multiplayer has become one of this genre's main features nowadays[5].

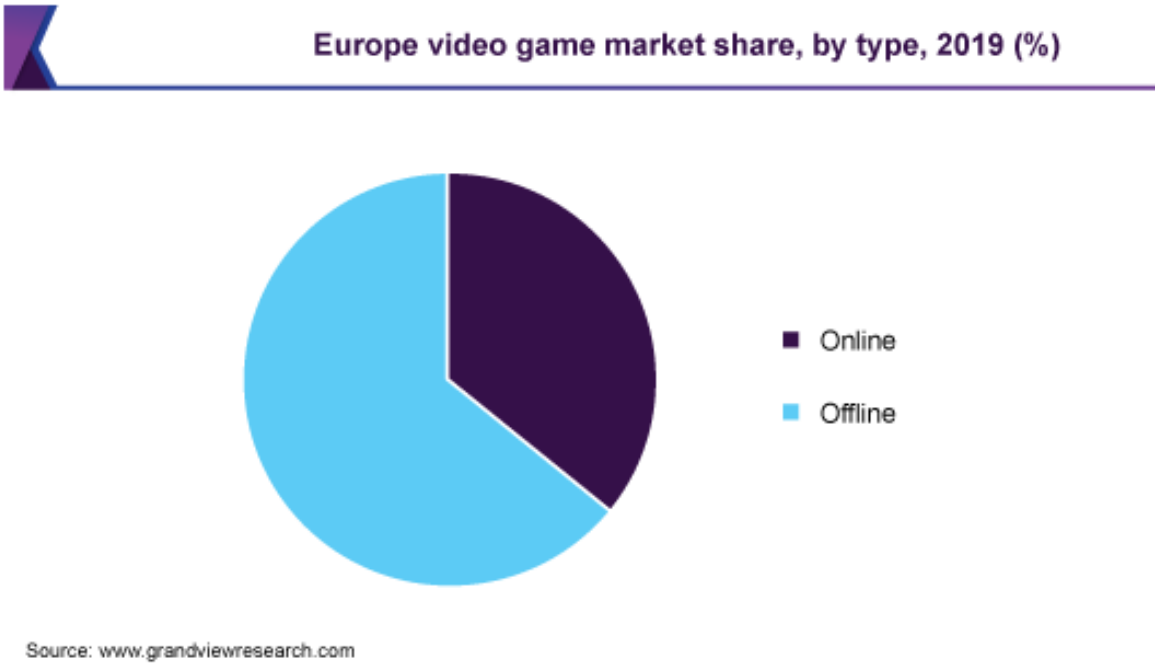

Fig. 1.2. Europe video game market share, by type, 2019(%), access 2020.12.1, [1]

1.1.3. Procedural generation

Procedural content generation - refers to usage of algorithms for content creation. It is commonly used across the whole video game industry due to it's potential to decrease developers' workload. Even though it offers great help to the developers, procedural generation requires to be done carefully and in a way that will be appropriate for the game it will be used in. It due to the fact that procedurally generated content needs to satisfy both players and the creator which is harder than with manually created content[6]. That is one of the reasons why even though it is possible to generate most of the game's contents like 3D models, music, maps, textures, in game items or even story; procedural generation is usually used only for some of these[7]. The usage of procedural generation varies between genres, rogue-like and survival games often use this method to generate endless levels whereas some MMO games use algorithms to randomize rewards for players.

1.2. Goal of the thesis

Goal of the thesis is creation of a project and implementation of a 3D First person multi-player shooter played on a procedurally generated level that will be a fixed size maze. The game will be played over LAN or Online. Project implementation will be carried out using Unreal Engine 4.

1.3. Scope of the thesis

Scope of the thesis includes development of the game's project and it's implementation. It consists of analysis of the existing solutions, formulating both functional and nonfunctional requirements, designing and developing adequate gameplay, adaptation of appropriate procedural generation algorithm, designing necessary user interface, implementation of designed game and proper multiplayer system. During and after the implementation the game should be tested so that it was free of errors that could negatively influence players experience and would offer gameplay adequate to the game's genre.

2. Existing solutions and tools analysis

2.1. Analysis of existing solutions

2.1.1. Valorant

Released on June 2, 2020 Valorant is a perfect example of a multiplayer FPS game. It is a free-to-play team-based tactical shooter developed by Riot Games, in which the players choose between several playable characters and many different weapons(Figure 2.1). Each of the characters offers unique abilities. Therefore Valorant is a combination of a hero shooter a type of game where it is the playable characters and their unique abilities that shape the player experience the most and tactical shooter mentioned before where cooperation between teammates and tactical approach to the task is the most important[8]. When it was available in so called limited access(access was randomly granted among users that asked for it) on April 7, 2020 it was the most popular game when it comes to viewership on all streaming platforms.

Fig. 2.1. Gameplay in Valorant, access 2020.12.2 [8]

2.1.2. Spelunky

Spelunky is a free-to-play 2D indie platformer game, created by Derek Yu released in December 2008 on Micresoft Windows and remade in 2012 for it's console release(Figure 2.2).

In order to complete the game the player must survive exploration of caves filled with deadly traps and monsters whilst gathering as much of the treasures abandoned in the caverns as possible. While moving through the procedurally generated tunnels the player will find items that will help them in their endeavor. The game has a unique approach to it's level generation as it consists of both carefully hand crafted sub-rooms that the level will be composed of and algorithm that will generate placement of these parts within the cave[9]. Spelunky was well received both by critics and by the public and had an immense influence on many games that wanted to follow it's example(mostly from rougelike genre). Many critics and publicists believe it to be on of the best games of all times.

Fig. 2.2. Screen shot of gameplay in Spelunky, access 2020.12.2, source : [10]

2.1.3. Brawhalla

Developed by Blue Mammoth Games Brawhalla is a 2D free-to-play multiplayer fighting game(Figure 2.3). Released in October 2017 it is still developed and updated now offering player 51 unique playable characters. The game features many different game modes but it's most iconic ones are centered around players fighting one another in order to remain on the map. The players are supposed to push their opponents off the platforms they start the game at. Even though it's offers online matchmaking what means playing with available players from all over the world, many of it's players value it's local multiplayer options and approach to gameplay and multiplayer in general that makes for a good game to play with friends. Another important element of the game is so called cross-play that allows for players enjoying the game

on different platforms (PC, Xbox, PS4, Nintendo Switch, iOS, Android) to play the game together.

Fig. 2.3. Screen shot of gameplay in Brawhalla, access 2020.12.2, source : [11]

2.2. Tools

When creating a video game the most important tool is the video game engine. Game engine is a software-development environment that can be used as a foundation for making games[12]. The extent in which the tool supports developers may differ from one engine to another. What also will be different is the compatibility of the engine with different game genres, for example an engine perfect for making FPS games may be not optimal or unsuitable at all for making strategy games. Most popular modern game engines are however suitable for development of games from all sorts of different genres and support developers with almost all tools they may need. Core features that most game engines provide are physics engine, rendering graphics(2D and/or 3D), collision detection and responses, animations, scripting, artificial intelligence, sounds, networking, memory management, user interface creation.

Game development companies have different approaches when it comes to engines some create their own, some use other ones created by other developers and some use their own engine for some games and external engines for other games.

Because of that there are many game engines, out of which two most popular are Unity engine and Unreal Engine.

2.2.1. Unreal Engine

Unreal Engine is a video game engine developed by Epic Games. It made it's first appearance with the FPS game called Unreal in 1998. Despite the fact it was originally developed for first person shooters, the engine quickly escaped genre's boundaries and became widely

used across all different types of games. There have been already 4 versions of the engine with it's current 4th version released in 2014. Unreal engine 5 is announced to appear in the end of the 2021. Current version of the engine supports 2D and 3D graphics, augmented reality, virtual reality and has been even used outside video games industry for film making. The engine allows for development of products for many different platforms, most notably Microsoft Windows, macOS, Linux, Android, iOS, current and previous consoles generation and several VR headsets.

Until it's 4th version Unreal Engine's main scripting language was UnrealScript. It was removed however in favour of C++ and Blueprints Visual Scripting system.

The engine is popular both among big development studios that consist of hundreds of people and independent developers. It has great publicly available documentation, lively forum and tutorials made both by Epic Games and engine's enthusiasts. The engine also offers Unreal Engine Marketplace where developers may purchase or sell all sorts of different assets (models, animations, code snippets etc.).

2.2.2. Unity

Unity is a video game engine developed by Unity Technologies. First released in 2005 for macOS now supports over 20 different platforms. It supports 2D and 3D graphics, augmented and virtual reality and has been used outside gaming industry not only in film making but also in training artificial intelligence.

The engine has been created with the idea to make video games development available to more developers. Now the engine is the most popular one among indie developers and has extraordinary amount of related tutorials, articles and forums.

Current version of Unity uses C# as it's main scripting language. It also supports different various visual scripting systems that can acquired in the Unity Asset Store.

Unity Asset Store allows users to sell and purchase all sorts assets needed in video game development. Due to engine's great popularity Unity Asset Store offers incredible variety and quantity of user-generated assets.

3. Project

The general concept of the game is for it to be a fast paced FPS that is supposed to be played by a couple of people. The replayability is supposed to be achieved by level not repeating itself as it will be procedurally generated and therefore can take many shapes and sizes. Such a mix is an uncommon one as most FPS games tend have carefully handcrafted levels. It is therefore fundamental to remember that the map must suite the type of game it is in. Lets take a situation in which map generates to an enormous size then the players could have problems finding one another. The game would very quickly become boring. That is why it is vital to keep in mind how all elements of the game would fit together.

3.1. Functional and nonfunctional requirements

3.1.1. Functional requirements

— Hosting game for a couple of other players.
— Joining other peoples games.
— Procedural level generation.
— Possibility to move across the map.
— Shooting and killing other players' characters.
— Respawning(reappearing on the map) after being killed.
— Ability to leave the match at any point in the game.

3.1.2. Nonfunctional requirements

— Application is available for personal computers with Microsoft Windows and most Linux distributions.
— Application is available in English.
— Possibility to play without Internet connection on the local network.
— Mouse and keyboard support.
— Easy to learn intuitive gameplay.
— Readable and simple UI.
— Proper synchronisation of players' actions between machines.
— Gameplay free of game-breaking errors.

3.2. Gameplay design

Gameplay can be defined in many different ways, in general it is the specific way in which players interact with the game[13]. It is the core of every game and it's most important element that all developers try to embed within their games as well as they can[14].

Because of that fact it is customary in game development to define the general structure of game's gameplay as a way of planing the project. After launching the program, players will have two options of starting the game in the main menu either hosting a game or finding an already existing session. After creating or joining the lobby, hosts starts the game and the players appear on the map. The map is generated by an algorithm and is always a bit different. Players' starting location is random and after they spawn they are free to explore the level. While on the map players can shoot other players. Each player has a healthbar, the visualisation of how much longer they can withstand enemy shots. After getting shot the healthbar decreases and when it reaches 0, player dies and is respawned after a couple of seconds to fight again. There are many ways to resolve the match, it can be after some time elapses or when one of the players gets enough points for killing others to win it can be also simply after all players decide that they want to finish playing and leave the match. In this particular game choosing match ending scenario at the very beginning of development is rather irrelevant.

3.3. Multiplayer

Across video game development industry there are many approaches to multiplayer. Some multiplayer games are oriented at cooperation between players other want players to fight one another. Games also differ in amount of players that can play a match together and in technologies used to connect players.

Most games nowadays offer networked multiplayer as opposed to the local one which is still an option in some games like for example Brawhalla but is far less popular. Most online multiplayer games use dedicated servers to which players connect in order to play together. Some games , especially those with multiplayer for small groups of people, instead of using dedicated servers use one of the players as the host and their machine acts as the server(Figure 3.1). Such solution wouldn't be a good one for a game played by tens of players at once or games that are supposed to be played for many hours because the host would need to stay in the game all this time (unless the game had some sort of host migration system). It is however a perfectly sufficient solution for a game that is supposed to be played by a small group of people for short periods of time which is the exact idea for the game developed in this project.

The game will allow the players to connect to sessions hosted online and to those hosted on their local network so that if the players share the same network they can still play together even without the Internet.

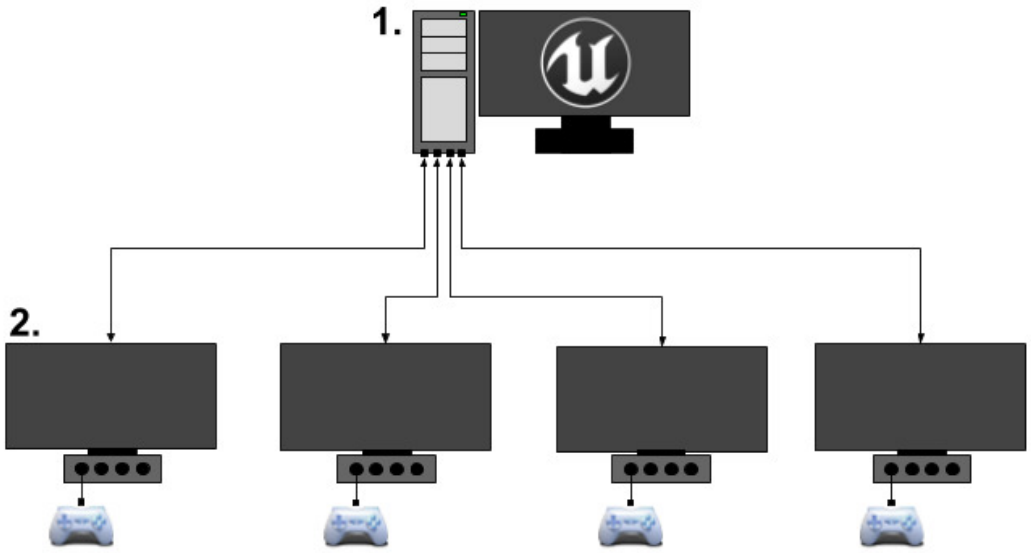

Fig. 3.1. Illustration of Client machines(2) connecting to Host(1), source: [15]

3.4. Procedural Generation Algorithm

There is plenty of different ways in which games generate their levels. Some generate them endlessly like Minecraft and some use procedural generation to randomize their layout like Spelunky. It is always detrimental to decide what kind of algorithm will suite certain game the most.

It is already decided that the game created in this project will have a small maze generated at the start of the match. Small size will make players meet each other more often and make the game more dynamic and filled with combat.

Another important feature of the map should be the ability of the player to reach every part of the map. If the players will be placed around the map at random, existence of some closed off spaces could not only unnecessarily limit the map that is already small just enough but also create situations when some players are closed off from the others and can't fight and enjoy the game.

That eliminates a full random algorithm which due to it's unpredictability could create unwanted elements in the level like closed off parts or big chunks of walls taking up a lot of level's space. Map however is supposed to be generated randomly as it should differ between matches. Taking that and the fact that all parts of the map should be connected into consideration I decided to use a randomized version of depth-first-search algorithm. Such algorithm would not only be easy to implement but would provide a random maze with all of it's parts connected.

After generating the level algorithm should select several locations around the map that

are unoccupied(by walls) as places for players to appear at when the match starts or when they die. These locations can be chosen completely at random and the only regulations they require is for them to be within the maze, not inside the walls and to be of appropriate number. For example only one such place would lead into a situation in which all players would start the match on top of one another and also would always reappear in the same place after being killed. After one of the players would win the initial skirmish they could just wait for their enemies to reappear and kill them instantly knowing exactly where they will spawn.

3.5. User Interface

User interface in video games should be adequate to the platform they are intended for and to the game's genre[16]. In FPS genre the UI should be intuitive and simple.

3.5.1. In-game view

In FPS games in-game UI should be as minimalistic as possible in order not to obscure players' vision. Therefore in-game UI will consist only of player's healthbar that signalizes how much longer player will survive enemy attacks, full red bar means player's character is in perfect condition, when the bar gets empty player character dies, and of crosshair that will indicate where exactly the player is aiming helping them shoot more accurately. Image of the in game UI can be seen later in the implementation chapter(Figure 4.7).

3.5.2. Main Menu

When player starts the game first what they will see will be Main Menu. In main menu(Figure 3.2) players are presented with 4 different buttons out of which first three will take the player to another menu and Quit button will close the game. Host Game and Find Game will take the player to menus from which they can later get to lobby and after that start playing and Options will let player customize their game settings. Menu if as simple as possible and should be intuitive to any user.

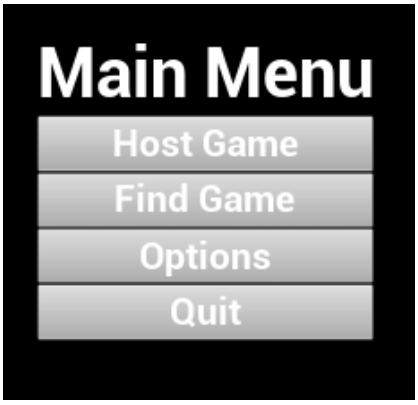

Fig. 3.2. Main Menu screen, own sources

3.5.3. Host Menu

After clicking Host Game button the player will be taken to Host Game menu(Figure 3.3). In here player must name their server, define whether they want to host the game on local network or via the Internet and they can also specify the number of players(minimum of 2 and maximum of 6). After all that session can be created. Player can also click 'Cancel' button and get back to main menu.

Fig. 3.3. Host Menu screen, own sources

3.5.4. Find Match Menu

After clicking Find Game button the player will be taken to Find Match menu(Figure 3.4). In this menu player can choose to look for either Online game or LAN game. If the game is not found players will be notified about this and if it will be they will be presented with an option to join found game(Figure 3.5). Player can also click 'Cancel' button and get back to main menu.

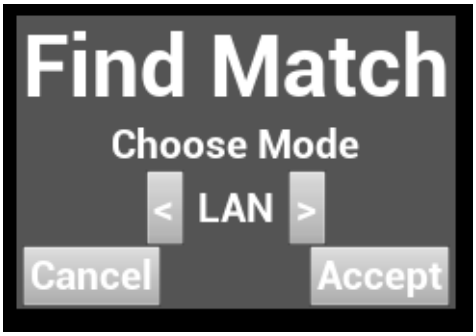

Fig. 3.4. Find Match Menu screen, own sources

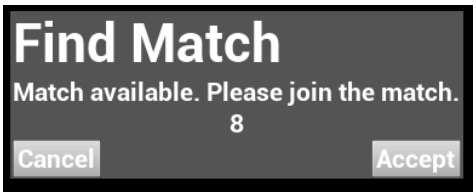

Fig. 3.5. Game waiting for the player to either join found match or change their mind, own sources

3.5.5. Player Options Menu

Currently the only option available in the options menu is to set player's name.

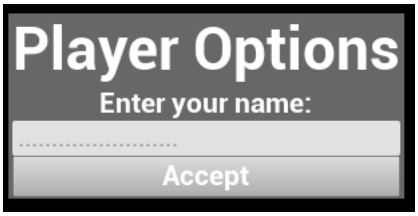

Fig. 3.6. Player Options screen screen, own sources

3.5.6. Lobby Menu

After either creating session or joining an already exiting one player joins the lobby(Figure 3.7). Lobby screen consists of current players number and maximum players number as well as players list on the right side of the screen and server name and two buttons on the left side of the screen. The first button changes depending on whether it is viewed by the host or by the guests. Guests can signal the host that they are ready to begin the match. Host on the other hand can sees this button as 'Start Session' button and can use it to start the match. All players can leave the lobby and be moved to main menu at any time with 'Leave Lobby' button.

Fig. 3.7. Player Lobby screen screen, own sources

4. Implementation

This chapter of the thesis will be dedicated to approaches to different parts of the game, their implementation and related problems that were encountered during said implementation.

4.1. Unreal Engine 4

In the section dedicated to tools, two game engines were compared. Both offering vast documentations, huge amount of online articles, forum posts and tutorials and market with free and paid ready made assets. What is more both are free to use for small developers. The main difference between Unreal and Unity is the fact that Unity offers better performance at weaker machines and Unreal helps developers achieve better visuals more easily[17]. That is why Unity is more popular with small indie developer teams and Unreal Engine is more popular with large teams producing high-end games. It does not mean however that Unity is not used by big companies or that Unreal Engine is unused by the small ones. In many cases, especially in one person teams it boils down to personal preference of the developer.

In my case having basic knowledge of the Unreal Engine and some experience with C++ before starting the project and having very little experience with both Unity and C# the more likely choice was Unreal. What is more since the decision to make a multiplayer FPS Unreal Engine was even more obvious as it offers much help with networking and provides very good ready made assets for starting of first person shooter development.

Version of Unreal Engine used for project implementation was 4.25.4 and it was the only tool used for project implementation.

4.1.1. Used Assets

Assets used it the project come from three places:

— Starter Content pack - available with the engine it is a pack of simple meshes, textures and materials.
— First Person template - available with the engine it is a perfect base for first person shooter as it has all the needed assets like player character model (represented by two arms), gun, projectile, firing sound and several static meshes. What is more this template has several most basic functionalities implemented. The character can move and look around, jump and shoot the gun without from the very begining.
— Animations Starter Pack - available in the Unreal Engine marketplace, developed and

published by Epic Games(Unreal Engine's developer and owner) it is a pack containing 62 simple animations designed to be used with classic Unreal Engine's Mannequin.

Several more notable assets are:

— First Person character script from First Person Template
— Gun mesh from First Person template(Figure 4.2)(Figure 4.3)
— Mannequin from Animations Starter Pack(Figure 4.1)
— Mannequin arms from First Person template(Figure 4.2)
— Cube mesh from Starter Content
— First Person animations from First Person template(Figure 4.3)
— Animation for standing walking from Animations Starter Pack(Figure 4.1)

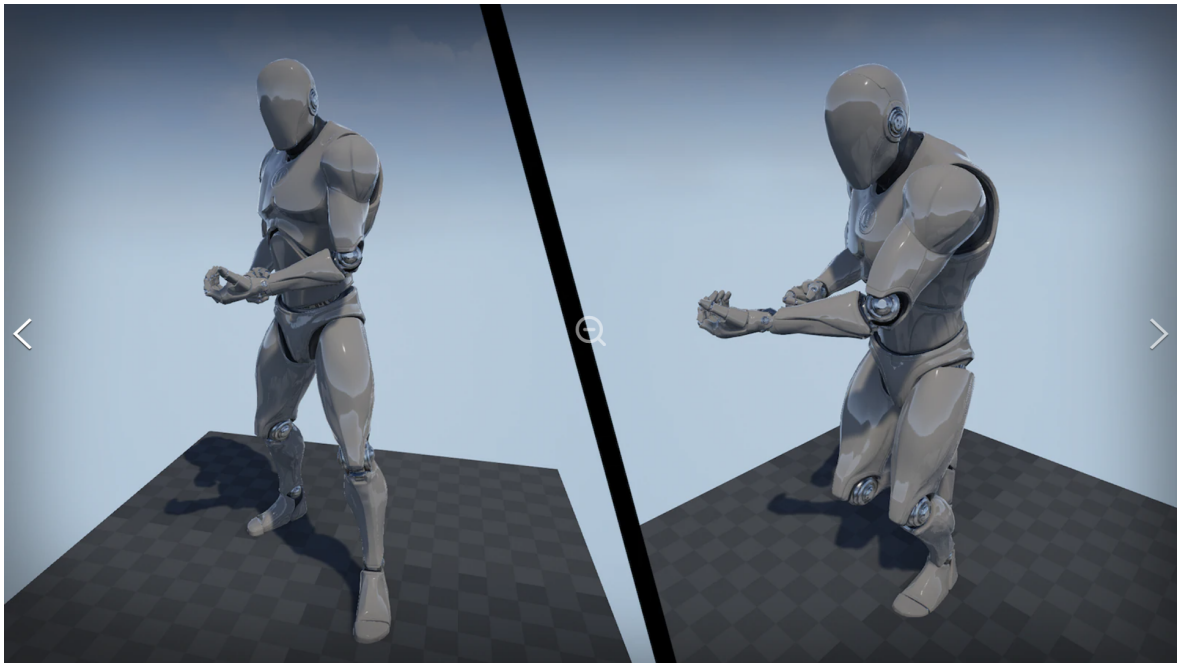

Fig. 4.1. Standing and walking animations from Animations Starter Pack, access 2020.12.9, source : https://www.unrealengine.com/marketplace/en-US/product/animation-starter-pack

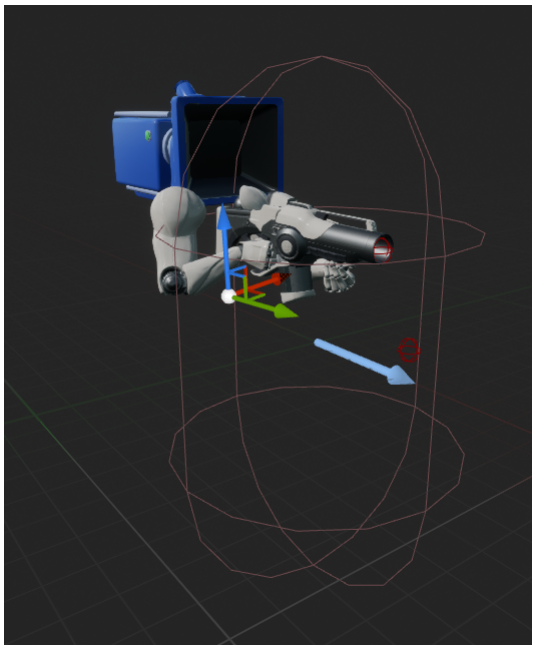

Fig. 4.2. First Person Character from First Person template viewed in the engine, own sources

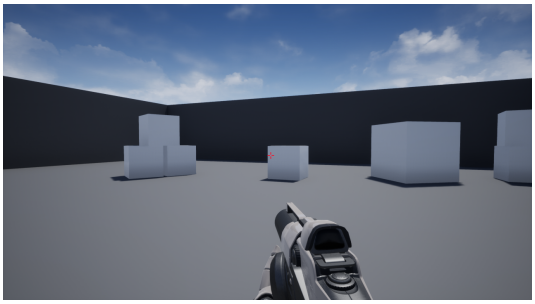

Fig. 4.3. In game view from First Person template, own sources

4.1.2. Blueprints Visual Scripting

Blueprints Visual Scripting is a visual scripting system used in Unreal Engine[18]. It uses node-base interface in which instead of writing code developer adds nodes, connects them with lines and manipulates their attributes.

As mentioned before Unreal Engine supports both Blueprints Visual Scripting and C++ for scripting. Both have full documentation, plenty of tutorials and other useful materials. Usually the choice between the two is based on personal preference. In case of this project the choice two use Blueprints was made because I wanted to learn something new while developing this game. Unreal Engine allows for combination of C++ and Blueprints within one project. Therefore in situation in which Visual Scripting system would occur to be uncomfortable to use there always exited two options of either abandoning existing project and starting a new one in C++ or stopping using Blueprints and switching to C++ within existing project.

Blueprints used in this project will be mostly Blueprint Class often shortened as Blueprint. They are assets that allow developers to add functionalities to existing gameplay classes and can be placed as instances within the game[19]. When creating a new blueprint developer must choose a Parent Class as the base(Figure 4.4). One important blueprint used in the project and unmentioned in the table below is Game Instance Blueprint Class which object is spawned at the start of the game and is not destroyed until the game is shot down.

Class Type	Description
Actor	An Actor is an object that can be placed or spawned in the world.
Pawn	A Pawn is an Actor that can be "possessed" and receive input from a Controller.
Character	A Character is a Pawn that includes the ability to walk, run, jump, and more.
PlayerController	A Player Controller is an Actor responsible for controlling a Pawn used by the player.
Game Mode	A Game Mode defines the game being played, its rules, scoring, and other faces of the game type.

Fig. 4.4. Most common Parent Classes, source: [19]

Another important type of Blueprint (not Blueprint Class) is Blueprint Interfaces. They are used to share and send data between different Blueprints[20].

4.2. Game structure

4.2.1. Game Instance

Whenever the game is started it creates a game instance. In this project it will be responsible for showing menus, creating and joining sessions as well as destroying them and displaying errors if any networking problems occur(Figure 4.5). Unlike game modes and player controllers it is always present and is not tied to levels or servers.

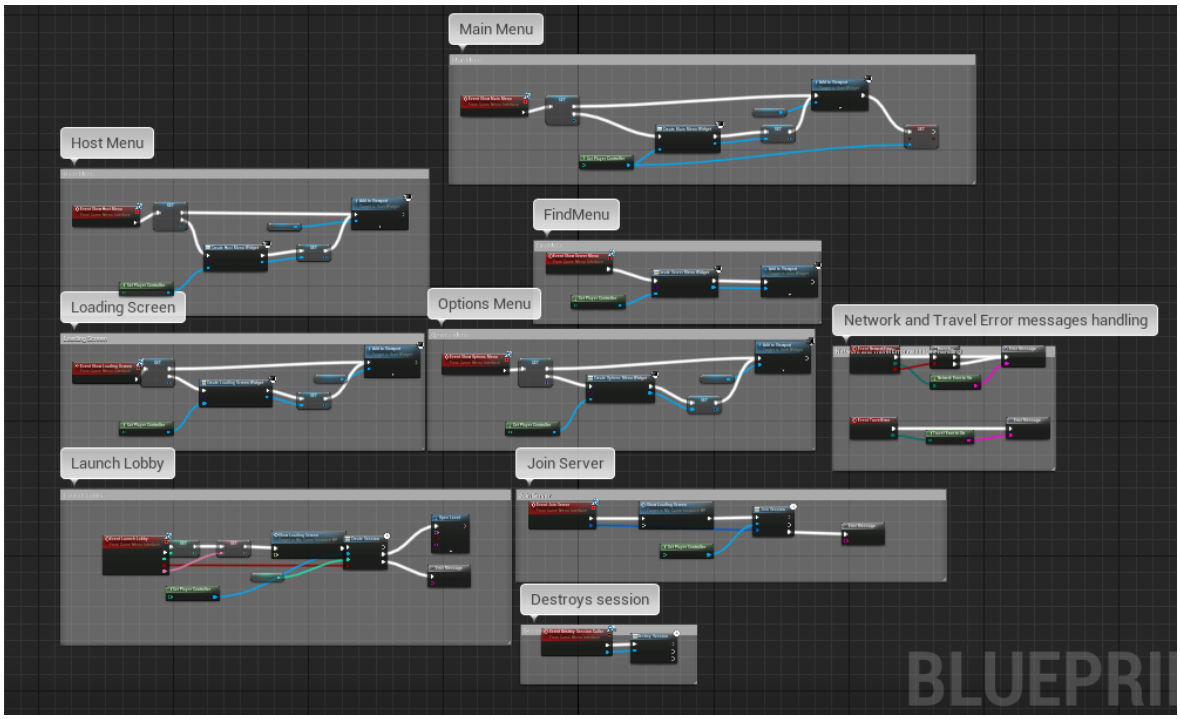

Fig. 4.5. Game Instance event graph, own sources

4.2.2. Levels

There are three levels between which the player will be moved. Each of them has it's own game mode and player controller. The levels are be Main Menu, Lobby and Main Game Level.

4.2.2.1. Main Menu

Main Menu is a level that consists of several Widget Blueprints which are used in Unreal Engine to create UI. Even though the level is called Main Menu it consists of several menus that were all presented in Project section dedicated to the UI. The level consists only of menus and is very simple so it uses a default game mode and default player controller as it only requires the player to have mouse cursor visible and be able to click the buttons. When the Level is opened it always show Main Menu widget at first. Player is moved to other menus when they click adequate buttons.

When player hosts game, game instance will run event for player controller to create a session with the information acquired from Host Menu widget blueprint and open Lobby level moving current player and any players that join the session there.

When player wants to find a game the situation is similar but first the Find Game Widget Blueprint will try to find the game for player controller using provided settings and if it finds one and the player accepts it then game instance will run event for the player controller to join found session.

4.2.2.2. Lobby

Lobby is a level to which all players are connected it governs the functionality of gathering the players and counting the number of currently connected players. When the host starts the match it will move all the players to the Game Level. Main Menu does not have special player controller or game mode as all it's functionalities can be performed by on_click button events. Lobby however needs these two for preforming previously mentioned functions. Currently it has no more functionalities but it can be easily extended by functionalities like chat or configuring map generation parameters like seed from which the map will be generated or map's size.

4.2.2.3. Main Game Level

This level just like Lobby level has it's own FPS Player Controller and First Person Game Mode. This is the level on which all the in-game events will happen. Unlike the previous levels this one is gameplay oriented and has only one menu that the player may open in order to disconnect from the server and be moved to Main Menu. Most of this levels functionalities are discussed in the upcoming sections.

4.3. FPS Gameplay implementation

Even though many of the features and elements of the game that will be talked of in either multiplayer or procedural generation sections, can and should be qualified as gameplay in general, this section exists mainly to put an emphasis on the fact that some of them are perfectly valid in a singleplayer FPS and are things to be thought of when developing an FPS game in general.

Despite the fact that some basic features like moving or shooting are already implemented in First Person Blueprint. There are still some basic functionalities missing and some require changes.

4.3.1. Shooting

There are two main approaches to shooting implementation in FPS games.

First one works by casting a ray(tracing a line between) from player character's gun to the location the player is aiming at, whenever the player shoots their gun. If the ray hits something the appropriate event happens being it damaging the surface or injuring the enemy. The hit happens in an instant and as there is no simulated bullet it is not affected by any physics like gravity and does not ricochet. It is however possible to check through how many and what kind of objects did the line pass and for example allowing players to shot opponents while standing behind a door(while not allowing for it if the bullet would have to go through a wall) or killing multiple enemies with one shot. It is the most popular way of implementing shooting in shooters in general and can be found in previously mentioned Valorant.

Second one is implemented in the Unreal Engine's FP template from the beginning and can be found in for example Battlefield 3. It works by creating a physical projectile that is an object on it's own. The projectile is shot from the gun to the location the player is pointing. It can be achieved by spawning projectile object just before the gun and giving it a certain velocity in the appropriate direction. This allows for bullets to be impacted by for example gravity which is useful in creating games that are supposed to simulate real combat conditions.

Because bullets ricocheting seemed an interesting and definitively uncommon feature the second approach was chosen for the game. The fact that it's implementation was already started did helped a bit however it required modifications. First all original bullet from first player blueprint is greatly affected by the gravity and has extremely slow speed. This was easily fixing by changing projectile's gravity to 0 as it is rather unnecessary unnecessary features on small closed-off maps and increasing projectiles speed 10 times(Figure 4.6). Despite the fact that now the projectile is much faster the player may still try to avoid enemy bullets which can be considered an interesting feature. When gravity was turned off and speed was increased it could have been easily observed that on long distances the projectile does not travel to location indicated by the crosshair at all. It was not a problem before as the projectile was very quickly going down due to gravity. It was necessary to change projectile spawn parameters until it would travel to the desired location.

After these two modifications the shooting system seemed appropriate for a fast paced game and what is more offered seemingly unique and interesting feature of bullets ricocheting from walls allowing players to shot enemies trying to hide behind covers and possibility for skillful players to try avoiding incoming bullets.

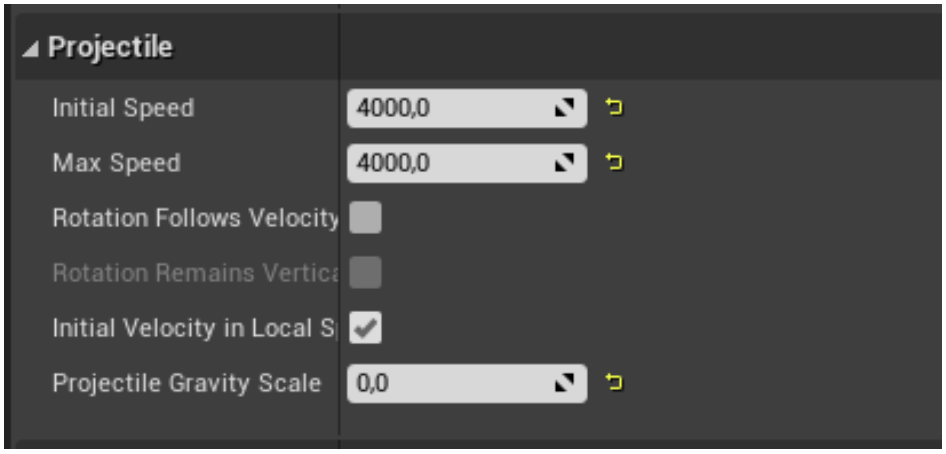

Fig. 4.6. Projectile parameters in Unreal Engine, own sources

4.3.2. Health system

Vast majority of FPS uses health bar to show in what condition player character is in. It is common approach in almost all genres in which player controls some character. This approach is used in order to allow players to take several hits/shots before dying. Some games don't use

health system and players die whenever something damages them, however in this project the decision was made to use health system.

All players start with 100% health when they spawn, be it at the start of the match or when they respawn after death. Whenever the player is hit with a projectile they lose 20% of the maximum health. This change is visible on the health bar. Health bar is situated at the down right corner of the screen and it is big enough for players to be able to monitor it without losing their focus on the match they are playing(Figure 4.7).

Fig. 4.7. In game view of a slightly damaged player character, own sources

4.3.3. Killing and respawning player character

As mentioned in the project chapter of the thesis, whenever player character health drops to zero the player will die and then respawn after a while.

Whenever the bullet hits player character it sends a message to it through a Blueprint Interface. When Player character receives this message it subtracts it's health and if the health gets below certain threshold(Figure 4.8) it calls "Die" event which calls "Kill Character" function and later "Respawn Player" function.

All this events and functions belong to First Person Character object, except for Respawn Player function which is is a part of First Person Game Mode class. It is because when it is the time for the player to respawn they will get a new character and therefore the previous one is no longer in use so it can't handle placing a new body. Originally the recently deceased player character would have it's physics enabled and lie on the floor, however for performance

reasons the decision has been made to destroy previous player character when giving the player a new one.

When respawning the gamemode will look for actors of Spawn Point class and place newly created player character at their location(Spawn points distribution throughout the level is described in the Procedural generation implementation and modification section).

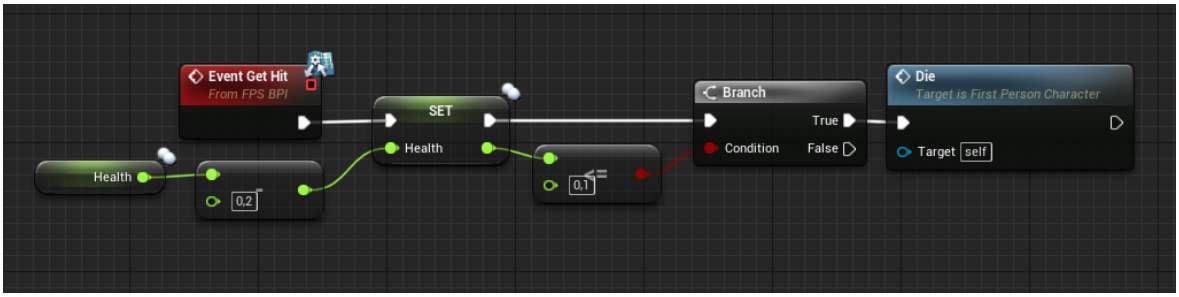

Fig. 4.8. Function responsible for for managing player character's health and in case it reaches zero - character's death, own sources

4.4. Multiplayer implementation

In this section main focus will be on the implementation of multiplayer within the game. Meaning different players seeing each others actions and being influenced by them.

4.4.1. Seeing other players

In games that have first person perspectives and multiplayer there are two ways in which player character is viewed. One is from it's own eyes and the other is from a third person view. It is very important as in many FPS games(FP template included) player character model is just hand and the gun, that is why in many games when player looks down they often can't see their character legs. However in multiplayer game character needs a full body for other players to see. There are two ways of achieving such a state. Developer may either implement a full body as if the game was played in third person perspective and place player's camera at the character's eyes location therefore achieving a first person perspective or create two bodies for the player character. In the second approach one of the bodies is just hands with the gun and is seen only by the player that owns the character and the other body is a full body model that is not visible to it's owner but everybody else can see it(Figure 4.9).

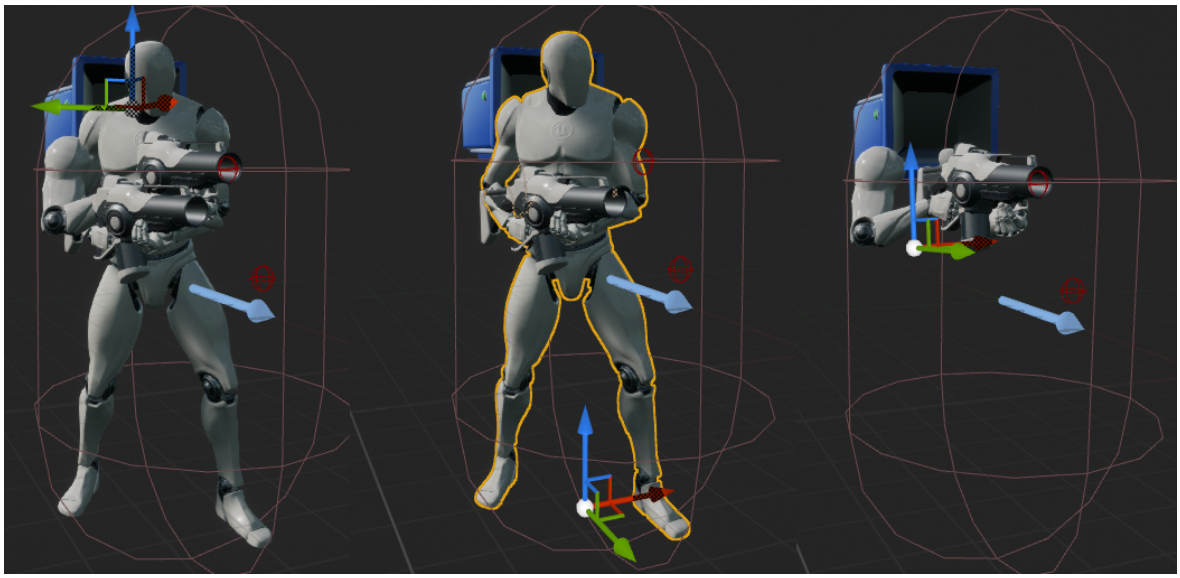

Fig. 4.9. Screen from the engine of first person character with different meshes visible(from the right full body with gun and first person model with gun together, only full body with gun and only fp model with gun), own sources

4.4.2. Replicating already implemented events

In the networked game each of the players has they own copy of the game's world and objects within it[15]. Some should be performed just on the local machine like showing healthbar on the players screen or displaying in game menu. However the ones affecting other players need to be performed by the server. The process of relaying information to other machines is called replication.

There were several changes needed to already functioning game elements so that they would also function in a multiplayer environment. Projectiles needed to be created by an event that runs on server. Otherwise they would exist only within the game run locally by the player shooting them. What is more the vertical direction in which the player is aiming is not replicated by default so it needed to be transmitted to other machines. Finally Killing and respawning the player required synchronization between all the connected machines.

4.5. Procedural generation implementation and modification

This subsection will be dedicated to implementation of level procedural generation algorithm and it's usage within the existing game.

4.5.1. Algorithm implementation

As stated in the Project section the game will use a randomised version of first depth search algorithm in order to creates it's level. Very similar algorithm and it's implementation in older version of Unreal engine can be found here[21].

The level will be a square shaped grid consisting of tiles in which the algorithm will place either a wall or a path. First find the algorithm will take in will be size of the maze. Size is the length of the square's side. After that the algorithm will generate walls at the borders of the square so that players can't leave the map. Next it will choose a random tile within the square and start from there.

How algorithm creates the paths within the maze can be illustrated with the following pseudocode(Listing 4.1). In which next_tile has always one tile away from the current_tile. It is to leave space between the paths that walls will populate. Next_tile is valid if it or the tile between it and current_tile is not already a path or bridge tile or is not a border.

Listing 4.1. Backtrack algorithm (source: own study)

```
void backtrack(current_tile)
{
  next_tile = pick_tile_from_random_direction
  while(next_tile is not valid and
        current_tile has other unchecked directions)
  {
    next_tile = pick_tile_from_other_direction
  }
  if(next_tile is valid)
  {
    add tile between next_tile and current_tile to bridge_tiles
    add next_tile to path_tiles
    backtrack(next_tile)
  }
  current_tile add to checked_tiles
  for(i = path_tiles_size to i=0)
  {
    if(path_tile[i] is not in checked_tiles)
    {
      current_tile = path_tile[i]
      break
    }
  }
  add last_bridge_tile to checked_tiles
  remove last_bridge_tile from bridge_tiles
  if(there is no more bridge_tiles)
    return
  else
    backtrack(current_tile)
}
```

Now all tiles that aren't already used become walls and the maze is finished.

In Unreal Engine the maze is a Blueprint Class with Actor Blueprint as it's Parent. It is composed of Instanced Static Mesh Component it is a component type used in Unreal Engine for procedural generation as it allows for rendering multiple instances of one static mesh

component. Therefore the maze is one object that is composed of many instances of one mesh. The static mesh that the maze is composed of is Shape_Cube mesh from Starter Content.

Algorithm placed tiles of different types on the grid, however on the level they need to be placed according to the world coordinates. After the grid is changed to coordinates construction script of the maze spawns cube mesh instances with height appropriate for their type. Border walls need to be high enough so that the player can't get past them. Path tiles are very thin as they act as a floor.

When all the tiles are occupied with instances of static mesh the algorithm is finished. In order to generate the Maze within the game it is enough to use Unreal Engine's Spawn Actor function and select to spawn an actor of class BPMazeGenerator. When the actor is spawned it starts it's construction script with the generation algorithm(Figure 4.10), therefore creating a maze(Figure 4.11) within the level.

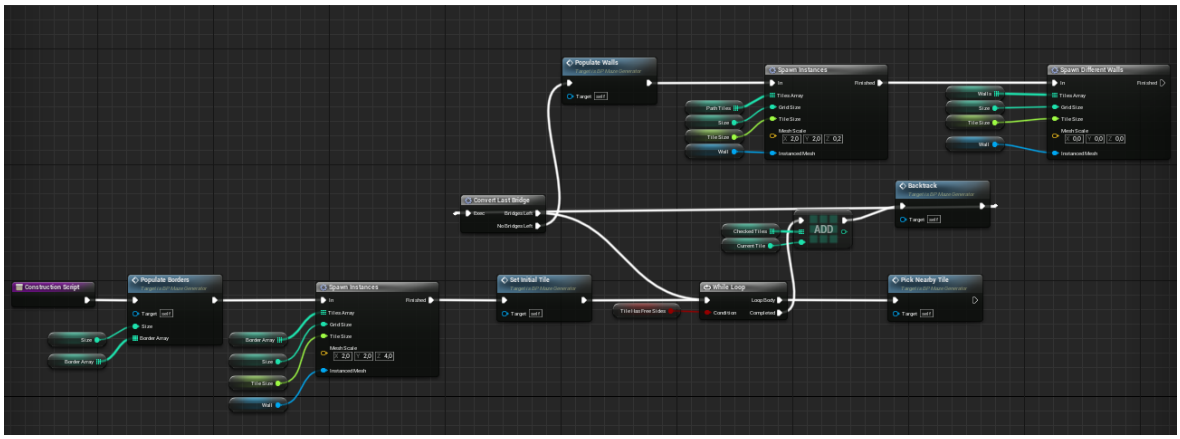

Fig. 4.10. Maze's Construction Script, own sources

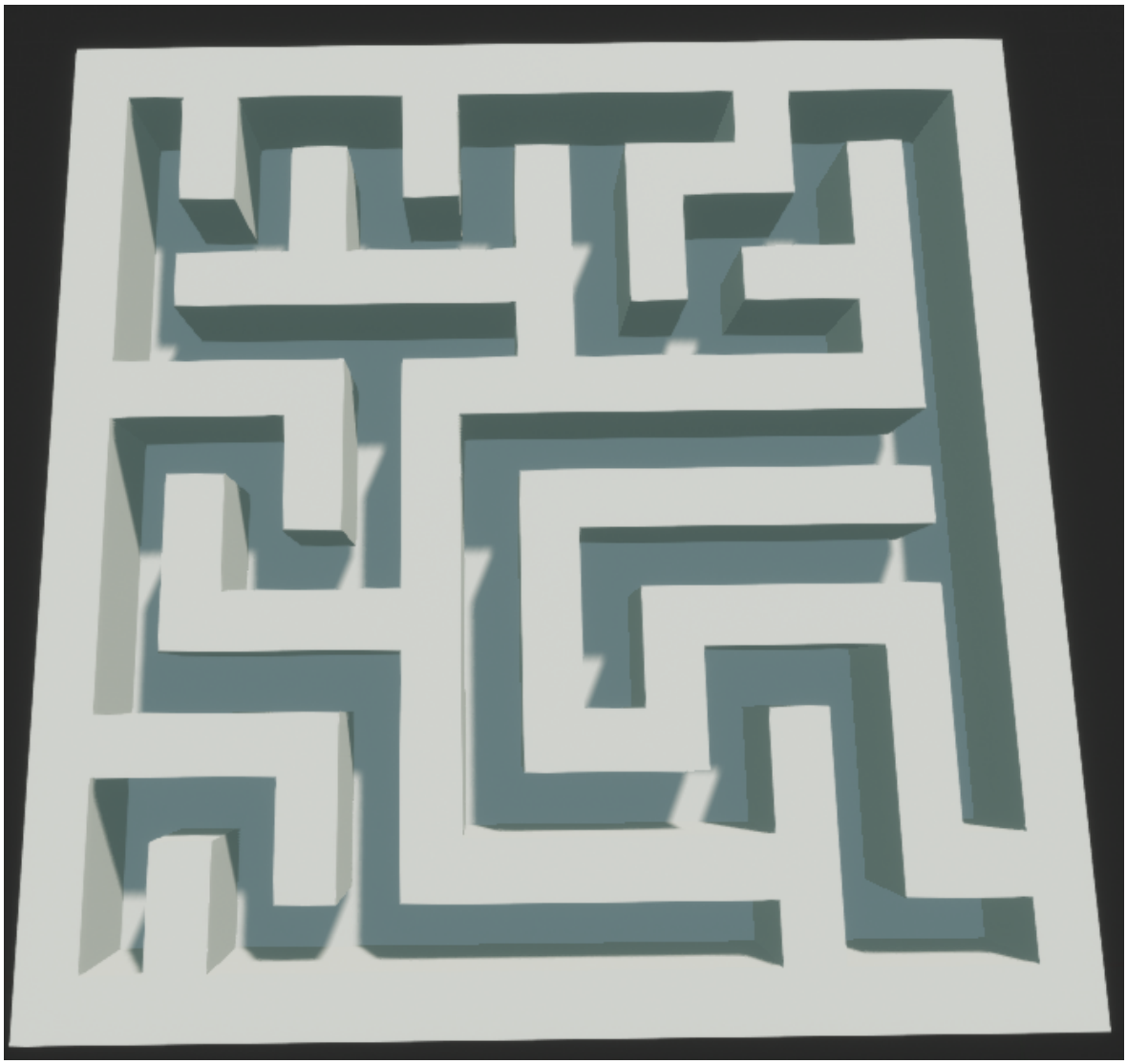

Fig. 4.11. Example of Generated Maze, own sources

4.5.2. Algorithm modification

There are several issues with currently generated maze. It is a simple maze with corridors that do not allow for much movement, meaning that dodging enemy bullets is almost impossible and hitting the opponent is incredibly easy. Therefore the algorithm needed modifications. Current modified version of the algorithm when creating walls assigns their height at 4 different heights at random. One of the heights is the same as floors so that the levels have a bit more open space. Second height is just the same as the border walls so remains unchanged. Last two heights are supposed to act as cover for the player but also allow for a bit of vertical movement. Lower of the two is just enough for the player to jump on and the other can be jumped on from the previous one(Figure 4.12).

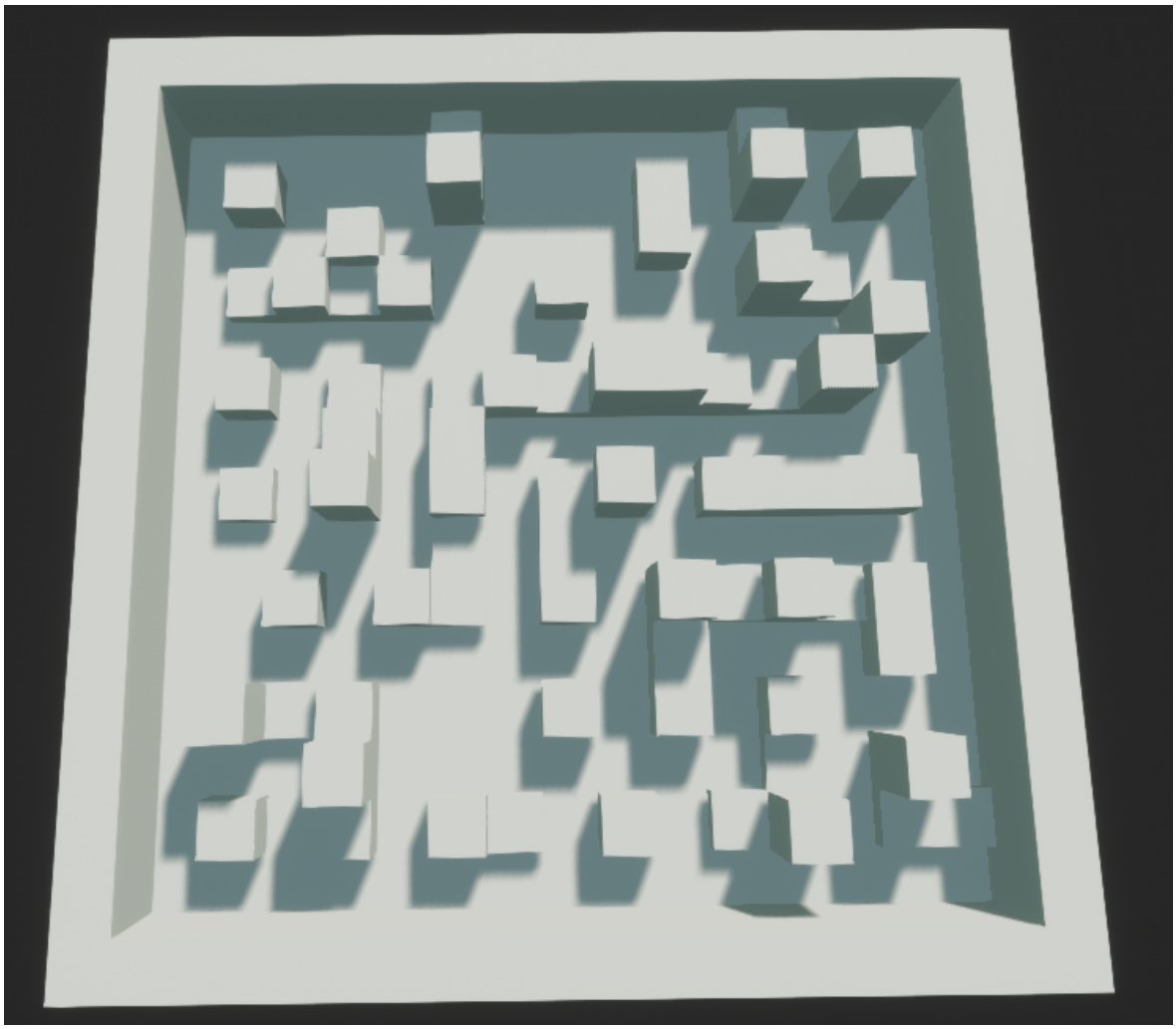

Fig. 4.12. Example of Generated Maze with modified walls, own sources

The player still can't get on top of the border walls and escape the level but they can explore the map more freely. The level is hardly a maze now but is far better suited for FPS game and even, walls height is random so it makes the level differ from one match to the other but is free from the problems of a fully randomly generated level.

4.5.3. Finishing the algorithm

After the maze is finished it should be filled with places for the player to spawn. This so called spawn points will be placed at random on the tiles that were previously classified as path tiles. So when player appears on the map they will be standing on a random path tile within the level. Spawn points should not be on top of every path tiles to minimize the chance of players appearing right next to each other, although there should be a lot of them to avoid situations in which players know where their enemy will respawn.

4.6. Steam Online Subsystem

Steam is a digital distribution system developed and owned by Valve Corporation[22]. Steam allows any developer to publish their game within the system for only 100 US dollars. Steam allows developers to add achievements to their game which players may later collect adding to the experience. It also has a community page for each game where players share mods, screenshots and tutorials for games. What is more it provides developers with it's online services that help in managing multiplayer games. Steam is currently the most popular distribution platform both among developers and players. It is both a shop and a platform which people use to communicate and play together.

The Steam Online Subsystem can be easily implemented within a game developed with Unreal engine. All it requires is adding a appropriate plug in and editing a couple of files. For the purpose of developing and testing a game in Unreal Engine a developer can use App ID that is shared by all other developers.

After Steam Online Subsystem has been added the game allows all players that posses a Steam account to play with each other online.

4.7. Implementation problems

This section will be dedicated to problems encountered during the implementation of the project and how they were resolved.

4.7.1. Seeing other players

First of the problems encountered when implementing the project was connected to the original approach to player character's body. Originally for the players to see one another there was an attempt to base the project on third person template and move camera to character's eyes. If the game was a singleplayer one such approach would work. Gun was added alongside with proper animations. Shooting and health systems were implemented. The problem occurred in multiplayer and is tied to the way networking in games works.

On the host there was no issue. However on the connected clients if the client moved, their walking animation was all shaky. Character's arms holding the gun that should move a bit while walking were shaking intensively(Figure 4.13). That is because the client will always play it's walking animation faster than it receives information from the server about it's body current state. So the information form the client and from the server will be displayed at one so character's mesh will be jumping between those states.

The best solution to this problem was implementing two bodies different bodies. One for the server and the other for the client. It was also an opportunity to start the project implementation anew and this time implement it better.

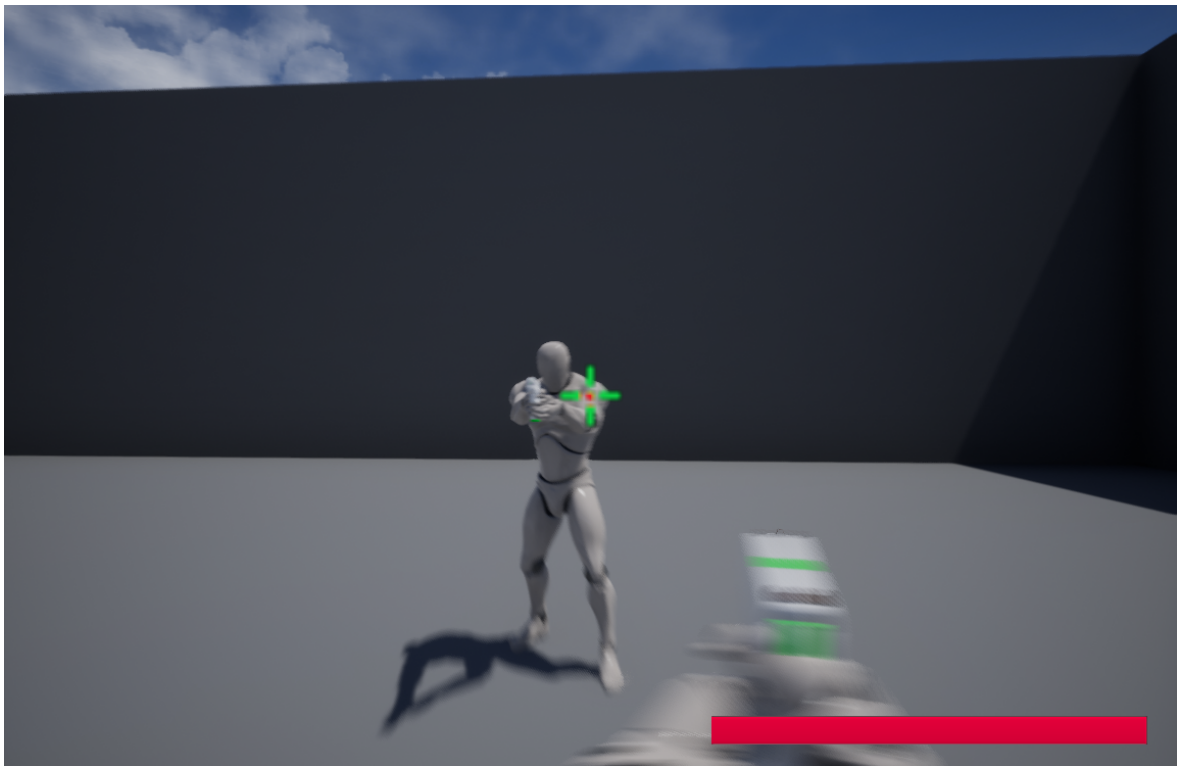

Fig. 4.13. Picture of early version of Project implementation(basing on Third Person template) with visible animation stutter while walking, own sources

4.7.2. Shooting while being dead

When player character loses all it's health it becomes invisible and the player loses the possibility to move around until they get a new character. However player can still move around and use all character's functions that are not tied to moving. Because of the way the shooting is implemented it means that the player could die, wait to be respawned and in that time kill their opponent. It was possibly the simplest Problem to fix as all it required was checking if the player character has more than 0 health before firing a projectile.

4.7.3. Players disconnecting from the server on match start

Originally there was an EventEndPlay implemented in the Lobby Player Controller blueprint that would manage a situation in which Host would leave the Lobby. The event would display error message and return clients to main menu. While in place it produced an error. When players would travel to the game level and leave Lobby level thus changing their player controller to a new one, this event would run and disconnect clients from the server. The event occurred to be unnecessary when the game was tested as an standalone application instead of testing it in editor. So it was removed.

4.7.4. Clients generating their own mazes

The biggest implementation problem and the hardest to fix is the one connected to maze generation in a multiplayer game. The problem is with replicating the maze between the server and the clients. If both clients and host generate the maze with the same parameters they will end up with mazes of same size but unfortunately different layout. As Unreal Engine will generate different numbers for each of them even despite the fact that they have the same seed. If the event happens only on the server and the maze is replicated to the clients, it works fine as long as the parameters responsible for constructing the maze within the game are the same as the ones specified in MazeGenerator class. Such solution in perfectly fine in the current game it however would need to be replaced with a different one if the game should allow for custom seed or map size. In that case due to replication specifics of unreal engine and how instanced meshes work it would be necessary to rewrite to generation part so that it would take place within FirstPerson Gamemode. It would need to spawn separate actor with static mesh for every tile within the level instead of creating whole maze at once.

5. Results

The final result is a game that has all the specified requirements. It is fully functional and allows players to play together over their local network or over the internet. It generates a maze, fills and modifies it into a level that is adequate for a FPS game. The game allows players to move around the level and fight each other. It is still however a playable prototype rather than a fully finished game. Animations are as basic as possible. So are the meshes within the level and many other elements of the game. It state however is developed enough to allow for the decision is such a design of a video game has any place on the market. Such playable prototypes are often developed by companies to ascertain if a specific game is worth making and some studios prepare many prototypes before releasing a single game[23].

5.1. Testers' opinions

Playtesting is a method of testing the game by letting people play it and gather feedback from them afterwards[24]. It is a method of testing essential to game development and is used not only for video games but for tabletop games as well.

After the game reached a playable state several people were asked to play the game a bit and share their opinion. The group was rather small one but all the people were experienced players that play games on everyday basis so were able to provide valuable information.

Most commonly reported problems with the game was the fact that there is only one weapon , game lacks a goal, animations and meshes are extremely simple. For some bullets were to big and to slow as these testers would prefer more classical approach to FPS game. What was also pointed out was the fact that the game feels quite 'clunky' meaning movement would require some adjustments.

5.2. Possible improvements

From the things that should be added for the game to be more of a completed product ready for market rather than a prototype the most crucial and simplest to implement are:

— Adding more unique weapons and either scattering them across the map for the players to collect them or giving them to the players from the beginning.
— Adding a score counting system that would allow the game to elect a winner either after some time elapses or after some player gathers enough points. That would provide a goal and a match over scenario different than host leaving the game.
— Adding textures and finding more interesting models and animations.

— Developing more functions in Lobby such as allowing the Host to choose map size or put in map seed that would allow the players to play on the same map they did before if they liked it.
— Adding chat to the game.
— Adding and utilizing a AdvancedSteamSessions plug in that could allow for inviting players from one's Steam friends list into a Lobby to play together and other Steam specific options.

Summary

To conclude, the goal of the thesis was fulfilled as through project and implementation stages a fully functional FPS game was developed. It uses procedural generation for creating it's level and can be played by multiple people at once. Both on local network and through the Internet.

During project implementation there was sometimes a need of changing previously chosen approach be due to problems with implementation or to deliver the best project possible. Such need for flexibility is something needed in game development as sometimes seemingly good ideas occur to be negatively impacting player experience. This alongside with that fact that every portion of the game needed adjusting in order for the multiplayer to function made developing this game quite challenging but informative experience.

As mentioned in the Results chapter the game is currently a functional prototype, in order to became a game that is good enough to be published it still requires polishing. After the addition presented possible improvements it can become a very interesting and unique product. That however would require not only programming skills but also that of animating and creating models and textures.

Bibliography

[1] *Video Game Market Size, Share & Trends Analysis Report By Device (Console, Mobile, Computer), By Type (Online, Offline), By Region, And Segment Forecasts, 2020 - 2027*. https://www.grandviewresearch.com/. May 2020. URL: https://www.grandviewresearch.com/industry-analysis/video-game-market (Last access 12/01/2020).

[2] Fred Dutton. *What is Indie?* Eurogamer. Apr. 18, 2012. URL: https://www.eurogamer.net/articles/2012-04-16-what-is-indie (Last access 12/01/2020).

[3] Steven Schneider. *The 5 Best 'Doom' Clones Ever Released*. Tech Time. May 4, 2016. URL: http://www.techtimes.com/articles/154622/20160504/the-five-best-doom-clones-ever-released.htm (Last access 12/01/2020).

[4] *Genre breakdown of video game sales in the United States in 2018*. Statista. July 24, 2020. URL: https://www.statista.com/statistics/189592/breakdown-of-us-video-game-sales-2009-by-genre/ (Last access 12/01/2020).

[5] Gerald A Voorhees, Joshua Call, and Katie Whitlock. *Guns, grenades, and grunts: First-person shooter games*. Bloomsbury Publishing USA, 2012.

[6] Mark Hendrikx et al. "Procedural Content Generation for Games: A Survey". In: *ACM Trans. Multimedia Comput. Commun. Appl.* 9.1 (Feb. 2013). ISSN: 1551-6857. DOI: 10.1145/2422956.2422957. URL: https://doi.org/10.1145/2422956.2422957.

[7] R. van der Linden, R. Lopes, and R. Bidarra. "Procedural Generation of Dungeons". In: *IEEE Transactions on Computational Intelligence and AI in Games* 6.1 (2014), pp. 78–89. DOI: 10.1109/TCIAIG.2013.2290371.

[8] Austen Goslin. *Riot's Valorant mashes up Rainbow Six with CS:GO for a speedy new tactical shooter*. Polygon. Mar. 2, 2020. URL: https://www.polygon.com/2020/3/2/21156352/riot-valorant-project-a-shooter-csgo-overwatch-rainbow-six (Last access 12/02/2020).

[9] Derek Yu. *Spelunky: Boss Fight Books# 11*. Vol. 11. Boss Fight Books, 2016.

[10] *Spelunky Steam*. Valve Corporation. URL: https://store.steampowered.com/app/239350/Spelunky/ (Last access 12/10/2020).

[11] *Brawlhalla Is A Free-To-Play Super Smash Bros. Alternative For PC*. PCG Pedia. URL: https://www.pcgpedia.com/brawlhalla-is-a-free-to-play-super-smash-bros-alternative-for-pc/ (Last access 12/10/2020).

[12] Jason Gregory. *Game engine architecture*. crc Press, 2018.

[13] Katie Salen, Katie Salen Tekinbaş, and Eric Zimmerman. *Rules of play: Game design fundamentals*. MIT press, 2004.

[14] Kevin Oxland. *Gameplay and design*. Pearson Education, 2004.

[15] *Networking Overview*. Epic Games. URL: https://docs.unrealengine.com/en-US/InteractiveExperiences/Networking/Overview/index.html (Last access 12/11/2020).

[16] Daniel Johnson and Janet Wiles. "Effective affective user interface design in games". In: *Ergonomics* 46.13-14 (2003), pp. 1332–1345.

[17] *Unity vs Unreal Engine: which game engine is for you?* Creative Bloq. Feb. 4, 2019. URL: https://www.creativebloq.com/advice/unity-vs-unreal-engine-which-game-engine-is-for-you (Last access 12/09/2020).

[18] *Blueprints Visual Scripting*. Epic Games. URL: https://docs.unrealengine.com/en-US/ProgrammingAndScripting/Blueprints/index.html (Last access 12/09/2020).

[19] *Blueprint Class*. Epic Games. URL: https://docs.unrealengine.com/en-US/ProgrammingAndScripting/Blueprints/UserGuide/Types/ClassBlueprint/index.html (Last access 12/10/2020).

[20] *Blueprint Interface*. Epic Games. URL: https://docs.unrealengine.com/en-US/ProgrammingAndScripting/Blueprints/UserGuide/Types/Interface/index.html (Last access 12/10/2020).

[21] Pattison Michael. *UE4 Tutorial - Random Maze Generator*. URL: http://voxagon.blogspot.com/2015/02/ue4-tutorial-random-maze-generator.html (Last access 12/12/2020).

[22] Christopher L Moore. "Digital games distribution: the presence of the past and the future of obsolescence". In: (2009).

[23] Jason Schreier. *Blood, sweat, and pixels: The triumphant, turbulent stories behind how video games are made*. Harper, 2017.

[24] Tracy Fullerton, Chris Swain, and Steven Hoffman. *Game design workshop: Designing, prototyping, & playtesting games*. CRC Press, 2004.

List of Figures

Listings