

Final report

Creating a 2D Side Scroller game skeleton, with networking, Artificial intelligence, and map generation capabilities



Cardiff university school of computer science

Author: Hossein Ramezani

Supervisor: Dr Frank Langbein

Moderator: Dr Jing Wu

One semester individual project

CM3203

Acknowledgment

First and foremost, I thank god, the almighty, who has granted me countless blessings, opportunities, wealth, health, and knowledge, so that I could finish this project.

“He who does not thank the people is not thankful to Allah.” [Sunan Abi Dawud, graded Sahih by Shaikh al-Albani.], Although I have spent countless hours working on this project, it would never have been completed without the help of many around me. I take this as an opportunity to thank and show gratitude to those who helped me and have been integral in the completion of this project.

I would like to thank my Mother Mrs. Wafa Bayt Meshal and father Mr. Saeed Ramezanian, whom I am forever in debt to and without their aid, guidance and support I would not have had the chance to be where I am today.

I would like to show my greatest appreciation to Dr Frank Langbein, who provide invaluable knowledge and guidance, which without I would not have been able to complete this project. Every meeting I have attended with Dr Frank yielded valuable information and motivation and for that I am forever grateful.

My sincere thanks to my family members who have always supported me through these tough times.

My special thanks to all my friends who gave me company and constructively criticized my work.

I would like to thank the various online communities and forums that helped me find resources and tools to create the project.

Table of contents

1. Introduction	5
2. Background	6
2.1. Game Engine	6
2.1.1 Unity VS Unreal	6
2.2. AI	8
2.2.1. Pathfinding	8
2.2.2. Finite State Machines	9
2.3. Networking	10
2.3.1. Architecture	10
2.3.2. Solutions	11
2.4. Map generation	12
2.4.1 Cellular automata	12
2.5. Definitions	13
2.5.1. Unity components	13
2.5.2. Mirror components	15
3. Design	16
3.1. The problem	16
3.1.1. Environment	16
3.1.2. Playable characters	16
3.1.3. AI/NPC	17
3.1.4. Network	17
3.2. Requirements	18
3.2.1 Environment	18
3.2.2. Playable characters	18
3.2.3. AI/NPC	21
3.2.4. Network	23
3.3. Solution	29
3.3.1. Main menu	30
3.3.2. Demo Scene	31
3.3.3 Map generation tool	37
4. Implementation	41
4.1. Main menu	41
4.1. Demo scene	42
4.1.1. PHGrid.cs	43
4.1.2. Network manager	44

4.1.3. 2D Player prefab:	45
4.1.4. FSM prefab:	47
4.2. Map generation scene	50
4.2.1. CellularGeneration.cs.....	51
4.2.2. Rogue-likeGeneration.cs.....	53
5. Evaluation.....	54
5.1. Frames per second.....	55
5.2. Round trip times.....	56
5.2.1. Locally	56
5.2.2. Over the tunnel.....	58
5.3. Payable character completeness.....	60
5.4. AI	60
6. Future works	61
7. Conclusion	62
8. Reflection on learning.....	63
Table of figures.....	64
References	66

1. Introduction

Since the introduction of computers, game development and design have been a field of interest for many mathematicians and computer scientists [1]. Like software development, game development is a multilayered process that involves many static parts [2] ranging from game design and planning to mathematical implementations and calculations [3].

This project aims to implement the technical aspect of a 2D-Sidecroller game. This includes creating a controllable gravity simulated character, a networked server that allows players to connect over the network and an artificially intelligent agents that play against the players. It also tries to venture into random game map generation algorithms.

In figure (1) the game's main scene is showcased, where two players have connected to a server and are playing against each other. It is important to note, that the project focuses on the technical aspect of implementation, therefore no effort to facilitate game related features, such as a story, combat and progression have been made.

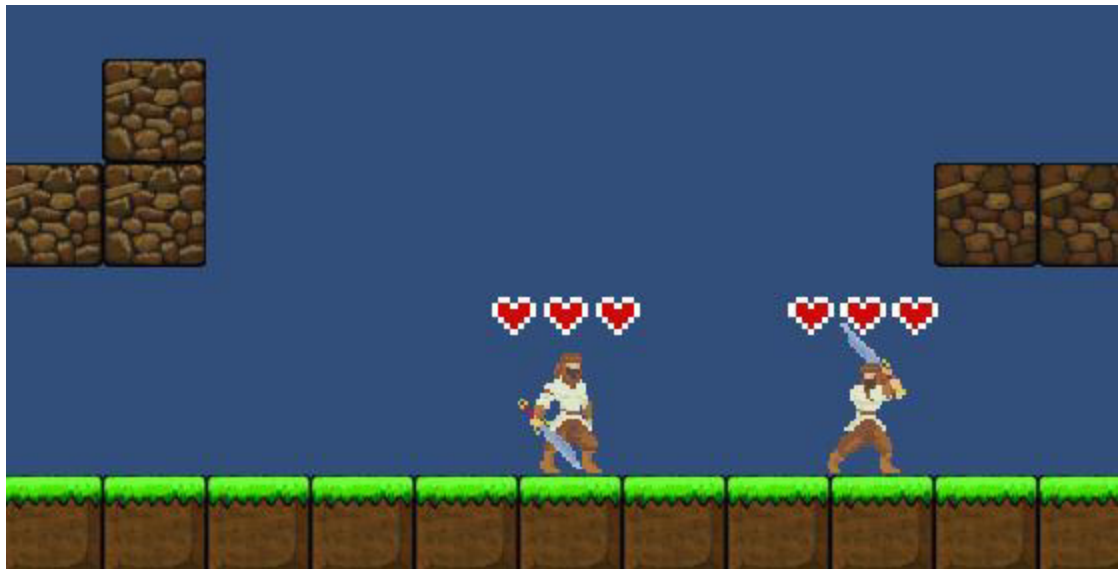


Figure 1, A scene from the game

2. Background

This section will explore all the building blocks of the project. Providing a brief explanation of each part and its components with a justification of some design choices, these range from explaining the unity engine and how it works to the different algorithms that have been implemented.

2.1. Game Engine

The first challenge encountered upon starting development was choosing the right framework/tool for developing the game. Since game development industry is a multimillion-dollar business [4] there was many tools and engines to choose from, however, the main features that were needed to be in a framework were:

- Rendering system.
- Physics system.
- 2D platformer specific tools.

Since these features are considered one of the toughest features to implement from scratch (Rendering system, Physics system) [5], the framework was required to have these systems ready for implementation, so no time would be spent on re-creating those systems.

Prior to starting development, I was comparing Unity game engine and Unreal. Both had pre-built tools for physics and rendering.

2.1.1 Unity VS Unreal

Launched in 2005 [6] the unity game engine is a cross-platform game development engine that offers developers a range of important features such as rendering, code-control and animation creation. The engine could be looked at as a sandbox tool that allows for the creation of code, physics simulation and games [7], many industries use Unity for its various application ranging from Automotive companies to construction and civil engineers [8].

Like Unity, Unreal is a cross-platform game development engine, first showcased in 1998 by Epic games studio [9]. It provides developers with a variety of features that help in creating games, 3D models and movies [10].

Both game engines provide a fully Integrated development environment with many features that allow for changing the environment to meet specific tasks, this made it a lot harder to choose which engine to stick with, and for that a comparison was made against the following criteria:

- Ease of Use
- Performance
- 2D support
- Community and documentation

Ease of use:

Both engines are made with a focus on ease of use [11], however, Unreal is based on C++ as the programming language while Unity uses C# to code the game's logic. Although I had previous experience in Unreal, I had minimal experience with C++ and since C# closely resembles Java which is a language, I am more comfortable with, Unity was favored in this criteria.

Performance:

Since unreal is compiled in C++ it has an edge over Unity and wins the performance criteria by a mile [12].

2D support:

Though unreal supports the development of 2D games, it is more oriented and focused on 3D games, Unity on the other hand has a more streamlined system that is arguable more intuitive and has higher capabilities for rapid creation of 2D games [13].

Community and documentation:

Both engines are heavily documented and offer an asset store to help develop games more easily and rapidly, however, the Unity asset store has many free assets [14] to choose from, it is particularly useful in animating and using art assets to demonstrate the project more coherently.

This comparison favored Unity and **Ultimately unity 2019.4 was chosen to develop the project.**

2.2. AI

Artificially intelligent agents are a big part of the project, it is a way to populate the game when no players are present, it also adds depth and replay ability to games. Many argue that adding AI to a game elevates the player-experience [15].

In platforming games, pathfinding is integral for the AI. This sub-section discusses and provides background information about some tools and algorithms that were used in creating the game's AI.

2.2.1. Pathfinding

One of the biggest challenges in designing an artificially intelligent game Agent, is a realistic pathfinding solution [16]. Pathfinding is responsible for the agent's navigation. there are many algorithms that serve pathfinding in the context of games, mainly graph/grid search algorithms such as Dijkstra and A*.

In this project, A* search algorithm is utilized to set the navigation system for the bots. A* is a graph search/traversal algorithm that relies on informed search methods to create paths in the form of weighted graphs [17]. it aims to search a path from a starting coordinate/node to a goal node while having the smallest cost (the cost is an arbitrary variable that is calculated to serve the goal of the search [18]). At each iteration of the algorithm, A* sets the cost of neighboring nodes according to a predefined formula. Upon completing the search, a list or an array holds all the nodes to be traversed, figure (2).

There are other solutions for pathfinding, such as Dijkstra's algorithm which has a similar approach to A* but instead of relying on a heuristic, it looks for all paths in the graph. A* was primarily favored due to its completeness, optimality, and optimal efficiency [19] and although it has some drawbacks, $O(B^d)$ Space complexity [20], the game would not require many bots at a given moment which makes this issue not very prevalent.

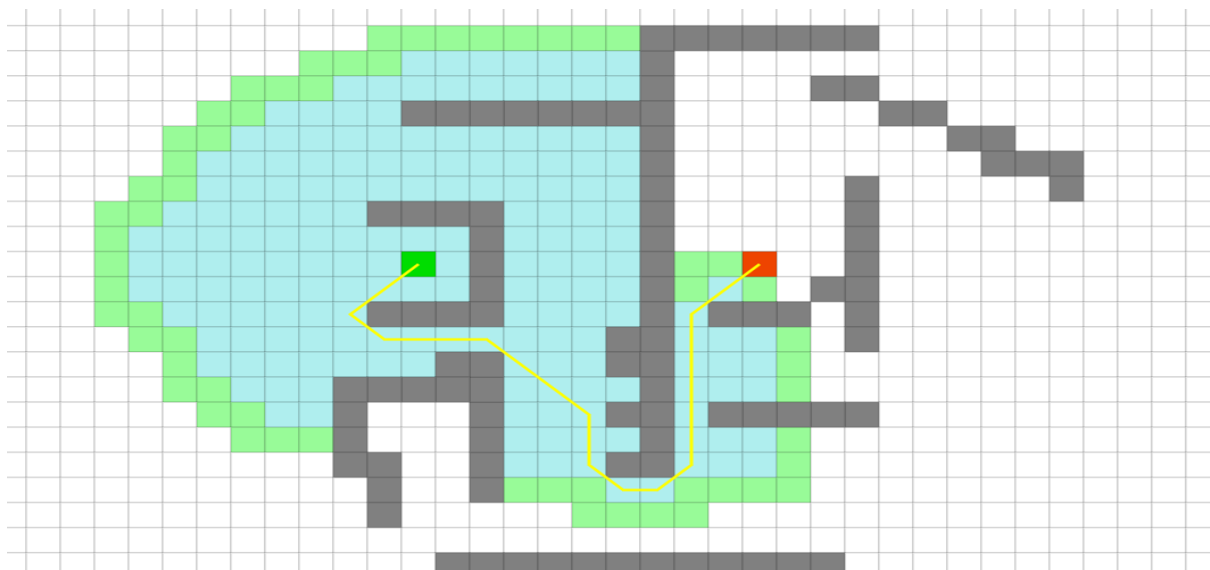


Figure 2, A* search, blue tile represents explored nodes while the yellow line represents the path.

2.2.2. Finite State Machines

Finite state machines are a module of computation that represents a structure for executing code based on a pre-defined set of states [21]. In figure (3) we can see a top-down view of how a finite state machine would be structured, as seen there are four states, find aid, wander, and attack each has its own logic and states switch between each other based on certain conditions and events.

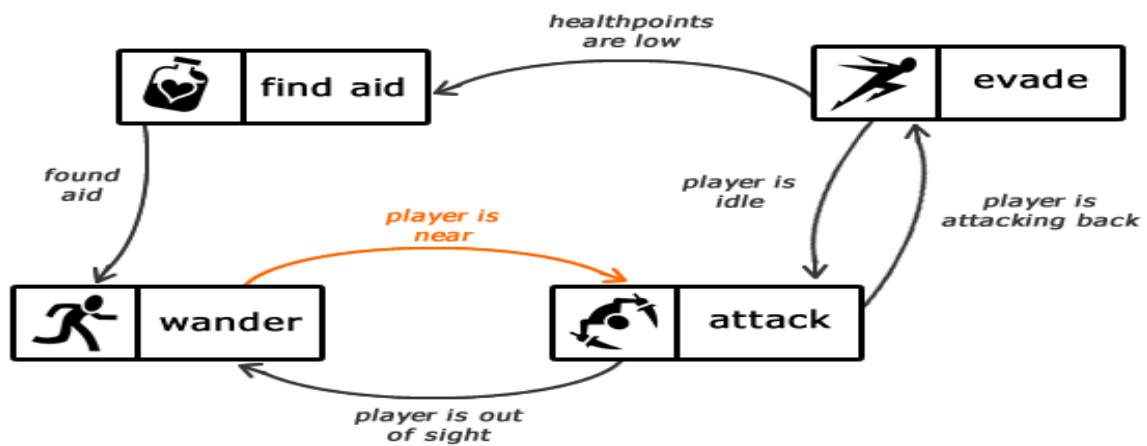


Figure 3, A simple state machine

Ultimately it serves as a skeleton for the game's AI agents and allow for integrating multiple behaviours in an easy-to-follow type of code.

2.3. Networking

Finding the correct networking solution was without a doubt the hardest challenge in creating the game as I had little to no experience with networking. Another problem I faced was finding the right tools, even if tools were found they were hard to use as they required a lot of prerequisites to use. Finally, I had to choose between two network architectures and two networking solutions.

2.3.1. Architecture

After researching which architecture to choose, I was left with either using a client-server model or a peer-to-peer solution.

The client-server model is a networking approach that works by having a host/server maintain the state of the game [22]. Each player's computer receives information from the server and updates its local game state, respectively.

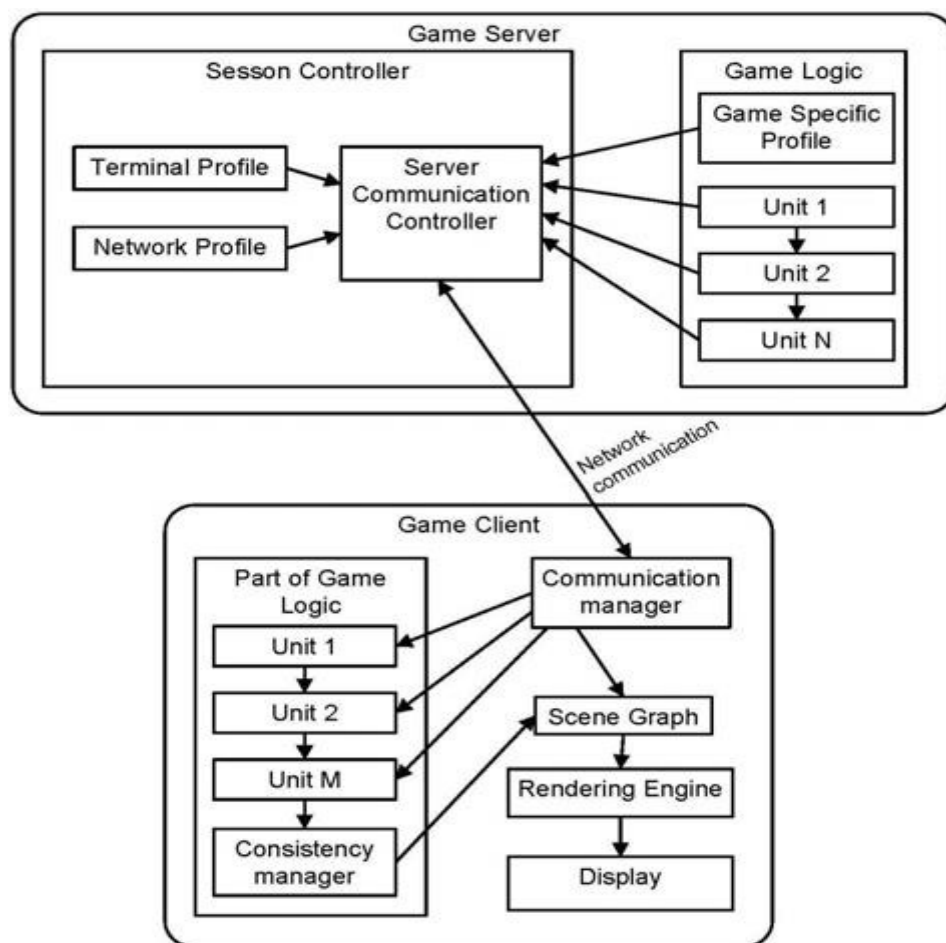


Figure 4, Client-Server model

While the client-server model uses a centralized system, peer-to-peer architecture could be considered as a decentralized system and works by having each computer run and maintain the game world [23], other computers validate any changes in the world and when all the peers agreed on a change the world is then updated.

Although the peer-to-peer model is used in many industries a lot of work is needed to develop technologies that support peer-to-peer architectures in games [24]. On the other hand, the client-server model has been favoured heavily in the gaming community [25], this is due to:

- Reliable data sharing.
 - Since the server maintains the world, we could be sure what data is being sent.
- Easy maintenance.
 - The world is maintained in known place.
- Security.

While also being established and research in game related use-cases [26] This led to choosing the client-Server model as the game's network architecture.

2.3.2. Solutions

After choosing the client-server model as the networking architecture, it was important to find a framework that allows for the implementation of such architecture. The framework needed to work with unity 2019.4 and support a client-server model.

Initially a C# server was developed, however, it added a high level of complexity to the project since it required a lot of prerequisites and a good understanding of networking technicalities. This led to choosing Mirror library as the networking solution.

Mirror is a Unity library that provides a networking solution [27] in client-server model. it has been optimized for ease of use and portability while giving users leverage over the code to adjust it, as necessary [28].

It works by providing an authoritative Host/Client framework, which allows for having the host and client code contained in the same script and within one unity project [29].

Note: Further details about Mirror components are discussed in section 2.5.

2.4. Map generation

Procedural map generation is the processes of creating game maps algorithmically [30]. The game aims to generate 2D side scroller type of maps, but it has been done on a surface level, where a simulation of the maps is represented.

In this project, two algorithms have been utilized to produce their respective demos, Cellular automata, and Rouge-Like generation. The latter is **discussed in the design section** as it is an algorithm I forged from bits of information about the game “Spelunky” [31]. The main reason choosing these algorithms over other available solutions is the amount of interest I found in both algorithms, especially in the cellular automata, by applying simple rules or changing a couple of variables in the algorithm immensely interesting patterns emerge.

2.4.1 Cellular automata

In the context of game design, cellular automata can be used to generate world maps. It works by applying a pre-defined set of rules to a randomized grid of cells [32].

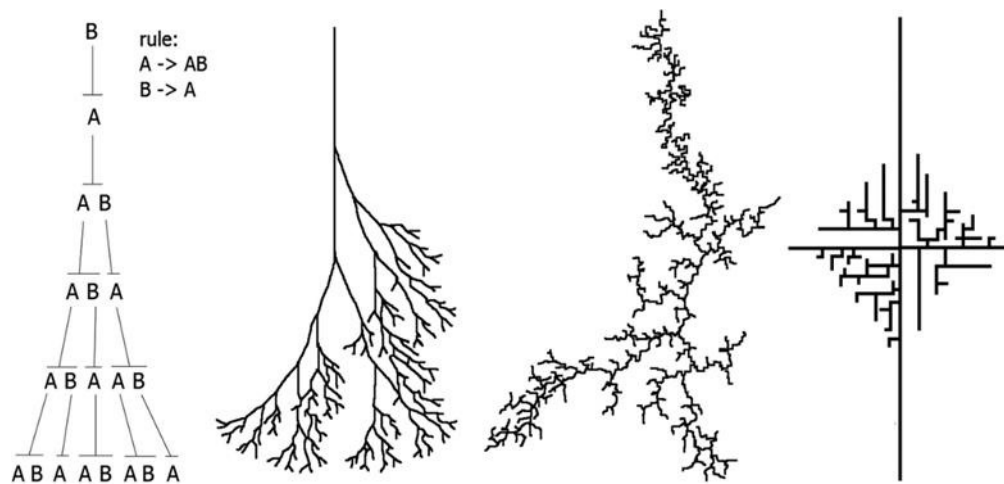


Figure 5, Cellular automata generation

As seen in figure (5), a graph is subjected to a set of rules if “A” then “AB”, else if “B” then “A” simply applying these rules generated a very interesting diagram.

2.5. Definitions

This section provides a brief explanation of components used in making the project, most are specific to unity. It provides important details that are later used in the design and implementation section and are provided to understand the context of which each component is being used.

For each component a top-level explanation is provide with a link to its corelating documentation.

2.5.1. Unity components

Game objects:

The Base class for all entities in Unity and is used to represent an object in the game. Unity sets the object with important functionalities that could be controlled via a script that is attached to them as a child component. These functionalities range from activating/deactivating an object in the scene to controlling the transform of the object.

<https://docs.unity3d.com/ScriptReference/GameObject.html>

prefab:

Unity's prefab systems allow the creation and storing of game objects as a template that could be used in many use-cases.

<https://docs.unity3d.com/Manual/Prefabs.html>

components:

components are unity's way to add functionalities to game objects. For example, a C# script which has the logic of character could be added to the game object which stores the sprites of character.

<https://gram.gs/gramlog/components-in-unity/#:~:text=Unity%20is%20a%20component%20based,and%20attach%20it%20to%20GameObjects.>

Monobehaviour:

By default, the Monobehaviour class is the main class unity scripts inherit from. It provides two main features that make it integral for Unity's implementation.

Co-routines management is the first of many Monobehaviour features, the Monobehaviour allows for starting, stopping, and managing a co-routine. Which is a way to manipulate code over periods of time frames.

More importantly, is the event system that is created within the class. It's a wide range of event messages that allow for the exaction of code relative to the state that the programme is in. the most important of these events is "Start", "Update" and "Fixed Update".

Start: when a game object is first instantiated.

Update: is called at each frame of the programme.

Fixed update: is called at each physics update.

<https://docs.unity3d.com/Manual/class-MonoBehaviour.html>

Event system:

Unity's event system is a way to send information to game objects based on their state relative to input in the game. This could be a click on a button in the game, or a slider change in the UI. Its primary role is:

1. Manage which game object is selected.
2. Manage the type of input system that is being used.
3. Manage object's ray casts.
4. Updating and managing the input.

<https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/EventSystem.html>

Canvas and UI system:

the canvas is a unity component preset, that has certain functionality which serve the creation and manipulation of UI and interface. It offers a simple drag and drop method for creating buttons, HUD's, overlays, menus and much more. It's also highly versatile since all its elements can be utilized by Unity's event system which gives the developer the power to code and add logic manually.

<https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/UITCanvas.html>

Tile maps:

Tile maps are components that store and handle tiles in 2D-space. it has important functionalities to access and change the state of tiles.

<https://docs.unity3d.com/Manual/class-Tilemap.html>

Physics system:

The following components are responsible for enabling physics simulation in Unity:

- Rigid body 2D.
- Capsule collider 2D.
- Tile map collider 2D.

<https://docs.unity3d.com/Manual/class-Physics2DManager.html>

Rendering:

The tile map renderer is responsible for rendering the tiles while the sprites renderer renders individual sprites.

2.5.2. Mirror components

Mirror interacts with the programme in two ways, either via inheriting from its classes or using its pre-set unity components.

The main class to inherit from is NetworkBehaviour which plays an integral role in adding logic to an entity over the network. It extends from the MonoBehaviour class and adds the following tags, which can be used to add functionalities to methods in a script:

- **[Server]** / **[Client]** tags can be used for the server-only and client-only parts.
- **[Command]**s is used for Client->Server communication.
- **[ClientRpc]** / **[TargetRpc]** for Server->Client communication.
- **[SyncVar]**s and SyncLists are used to automatically synchronize state.

Mirror also provides the following Unity components:

- Network rigid body
- Network animator
- Network capsule collider
- Network transform
- Network identity

<https://mirror-networking.com/>

3. Design

A design plan is crucial to the success of a project. It sets the aims and objectives while giving a clear and concise methodology to execute for achieving desired results.

This section details the design ideology behind the project, it is divided into three sections, “The problem” where a discussion will be made explaining the problem at hand, “The requirements” where the requirements that the game components need to meet are illustrated. “The solution” where the proposed game design is discussed, showing a high-level explanation of the various elements that meets the requirements and finally “Other solutions” where a comparison and a validation of the design choices and other design potentials will be made.

3.1. The problem

As discussed in the introduction, the game is a 2D sides scroller that completely focuses on the technical aspects of a game, with no regard to game design approach. This means, that ultimately the project should end up as sand box kind of tool where most of the features required to develop a 2D side scroller are present. The project aims to implement the following features:

- physics simulated environment.
- Playable characters.
- AI- agents with pathfinding capabilities.
- Game server, client-server model.
- Map generation

Each feature adds complexity to the project and in their own are challenges to be overcome.

3.1.1. Environment

The environment enables the player to experience the game, and in the context of 2D side scrolling it is an integral part for the game’s functionalities. The challenge here, is to develop the environment in a way that could allow for gravity or collision detection. Where a player could for example move on a platform from point A to B, or jump and experience falling from B to A. Furthermore, the environment should be manipulatable, for example if an AI agent wants to construct a path, the environment should feed its algorithms to allow for such logic to run.

3.1.2. Playable characters

There should be a way for players to navigate the game, a playable character solves this issue. Ultimately a player should be able to navigate the scene and since the game is networked this should be done over the network. The challenge here, would be taking input from the user, processing the input over the network/locally, sending output back to the user and simulate game features and environment related aspects locally or over the network (Gravity, physics, and collision detection).

3.1.3. AI/NPC

Artificially intelligent agents could add depth and re-playability to a game, it also serves as a player replacement if no other players are available to play with. The challenge here would be implementing and designing an agent that adheres to the game logic and plays as humane as possible.

There is also the challenge of allowing pathfinding capabilities in a 2D game, since most pathfinding algorithms are essentially graph search algorithms that output a graph traversal map there is a problem that arises, when traversing these paths, since the algorithms do not care if there is physics involved, this means a line for the shortest would be drawn from point A to B, which is not very useful for traversing in a real world scenario, so some measures should be taken to make the pathfinding conform to the laws of the game.

3.1.4. Network

Since the game aims to be a networked game, there should be a system that supports players joining each other. The challenge here, would be designing a network architecture that allows for players to join games, process their input, and give them back an output. This also raises other issues, such as data synchronization, for example how would player A know that player B lost X amount of health. Furthermore, interpolation is a concern in networked games since there is potential data loss, to account for that the architecture should be able to expect and calculate where the user would be at a given moment based on several factors such as previous input.

3.1.5. Map generation

Since the scope of the project has proved too wide, the aspect of game generation is not to be completely implemented in the sense that users would play in a randomly generated map. However, a more subtle approach is to be taken, where players can open a map generation scene and experience an illustration of a map generating algorithm. The challenge would be creating and finding interesting map generation algorithms that serve this purpose.

3.2. Requirements

This sub-section discusses the requirements that each section of the game needs to meet (Environment, Characters, AI, network, and map generation) and it aims to adhere to the problems discussed in section 3.1.

3.2.1 Environment

- ❖ Requirement
 - Physics simulation.
- ❖ Description
 - The environment must allow for physics simulation and collision detection.
- ❖ Example
 - Tiles in the environment can detect collision.
 - Some tiles can have physical properties such as friction.

- ❖ Requirement
 - Tile.
- ❖ Description
 - The environment must have a system that stores tiles and enable the manipulation of tiles.
- ❖ Example
 - A data structure that stores all the tiles in a map. It may have a specific class that store information about each cell.

- ❖ Requirement
 - Rendering.
- ❖ Description
 - The environment must have a rendering system the allows for the rendering of tiles and objects in the scene.
- ❖ Example
 - A sprite renderer to render the characters.

- ❖ Requirement
 - Scriptability.
- ❖ Description
 - The environment must allow a high degree of intractability and scriptability, users should have access to the environment information.
- ❖ Example
 - An AI-agent could access the environment to perform calculation via a script.

3.2.2. Playable characters

- ❖ Requirement

- Send input.
- ❖ Description
 - A character must have the ability to send input either locally or over the network.
- ❖ Example
 - A user moves the analog stick to change the position of their character.

- ❖ Requirement
 - Receive output
- ❖ Description
 - A character must be able to receive output either locally or over the network.
- ❖ Example
 - The game changes the health variable of a player after performing a specific action.
 - The server sends the health variable of other players in the game.

- ❖ Requirement
 - Rendering
- ❖ Description
 - A character must be render able.
- ❖ Example
 - Players can see their character in the scene.

- ❖ Requirement
 - Game logic
- ❖ Description
 - A character must have the ability to hold game logic.
- ❖ Example
 - A character may have a health variable that indicates its health in the game.

- ❖ Requirement
 - Physics
- ❖ Description
 - A character must experience physics.
- ❖ Example
 - A character can fall.
 - A character experiences drag.

- ❖ Requirement
 - Processes input
- ❖ Description

- The character can process and manage user's input
- ❖ Example
 - After a player inputs a movement sequence, the character can send information about its physics calculations to the environment.
- ❖ Requirement
 - Interact with environment
- ❖ Description
 - The character must have the ability to interact with the environment and its game objects.
- ❖ Example
 - A character attacks another character in the scene.

3.2.3. AI/NPC

- ❖ Requirement
 - Send input
- ❖ Description
 - The AI must have the ability to send input to the server/locally.
- ❖ Example
 - The server requested a change in a player's health variable.

- ❖ Requirement
 - Receive output
- ❖ Description
 - The AI must be able to receive input from the server/locally.
- ❖ Example
 - The server updated the AI's world position.
 - The local game changed the health variable of the agent.

- ❖ Requirement
 - Generate output
- ❖ Description
 - The AI must have the ability to generate output.
- ❖ Example
 - The AI computed a movement action from point A to B.

- ❖ Requirement
 - Find path
- ❖ Description
 - The agent must be able to calculate paths and generate graphs accordingly
- ❖ Example
 - The agent calculated a path from point A to B using a pathfinding algorithm.

- ❖ Requirement
 - State management
- ❖ Description
 - The AI must have a way to manage its state.
- ❖ Example
 - The AI changes from an attacking state to following state using a finite state machine.

- ❖ Requirement
 - Perform actions
- ❖ Description
 - The agent must have the ability to execute calculated actions.
- ❖ Example
 - An agent traverses a path generated by a pathfinding algorithm.

- ❖ Requirement
 - Interact with environment
- ❖ Description
 - The agent must have the ability to access environment variables
- ❖ Example
 - Casting a ray in the environment.

3.2.4. Network

Note: The diagrams illustrated in this section are meant to be component diagrams and not use-case diagrams, the stick-man figure refers to components.

Server/ Host:

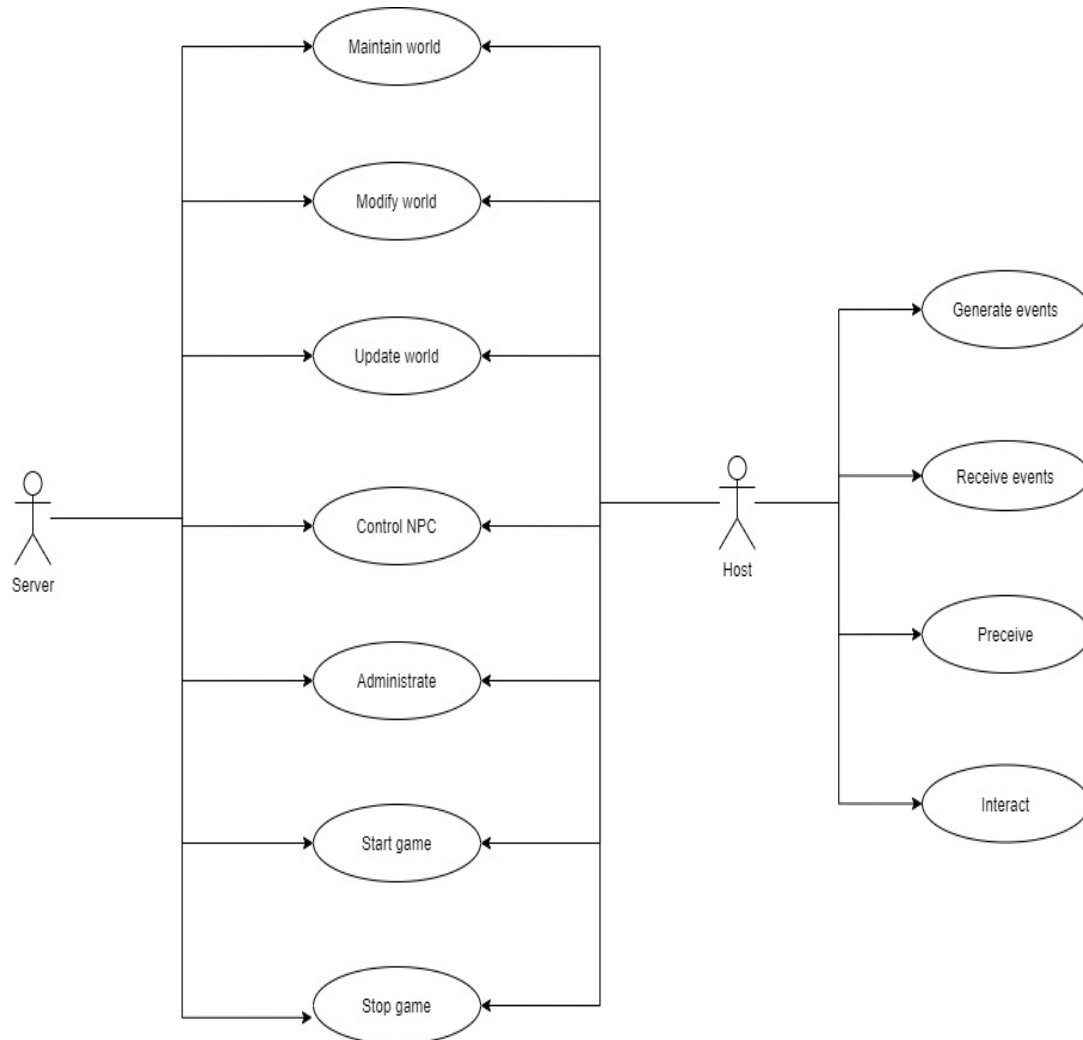


Figure 6, Server/Host component diagram

Requirement	Maintain world
Description	The actors must maintain and manage the virtual game world.
Component	Server, Host.
Example	<ul style="list-style-type: none">The server holds all the information about the environment and its objects.

Requirement	Modify world
Description	The actors must have the ability to access and modify the world.
Component	Server, Host.
Example	<ul style="list-style-type: none"> • The server changes the position of a player.

Requirement	Update world
Description	The actors must have the ability to update the game world and send input/output to clients.
Component	Server, Host.
Example	<ul style="list-style-type: none"> • The server updates an NPC location after a request from the NPC object.

Requirement	Control NPC
Description	The actors must have the ability to manipulate and control the AI in the game.
Component	Server, Host.
Example	<ul style="list-style-type: none"> • The server runs the AI algorithms.

Requirement	Administrate
Description	The actor's administration the game world and it is objects.
Component	Server, Host
Example	<ul style="list-style-type: none"> • A character asks for movement validation.

Requirement	Start game
Description	The actors must have the ability to start a game.
Component	Server, Host.
Example	<ul style="list-style-type: none"> • The server starts the demo map.

Requirement	Stop game
Description	The actors must have the ability to stop a game.
Component	Server, Host
Example	<ul style="list-style-type: none"> • The server stops a map.

Requirement	Generate events
Description	The actors must have the ability to generate an event.
Component	Host, Client.
Example	<ul style="list-style-type: none"> • A host execute a movement function.

Requirement	Receive events
Description	The actors receive an event from the server.
Component	Host, client.
Example	<ul style="list-style-type: none"> • The server tells a client move 5 steps in their game instance.

Requirement	Perceive events
Description	The actors can perceive event in the world.
Component	Host, Client.
Example	<ul style="list-style-type: none"> • Client A sees client B's character move.

Requirement	Interact
Description	The actors can interact with the game world.
Component	Host, Client.
Example	<ul style="list-style-type: none"> • Client A's character attacks a game object.

Client:

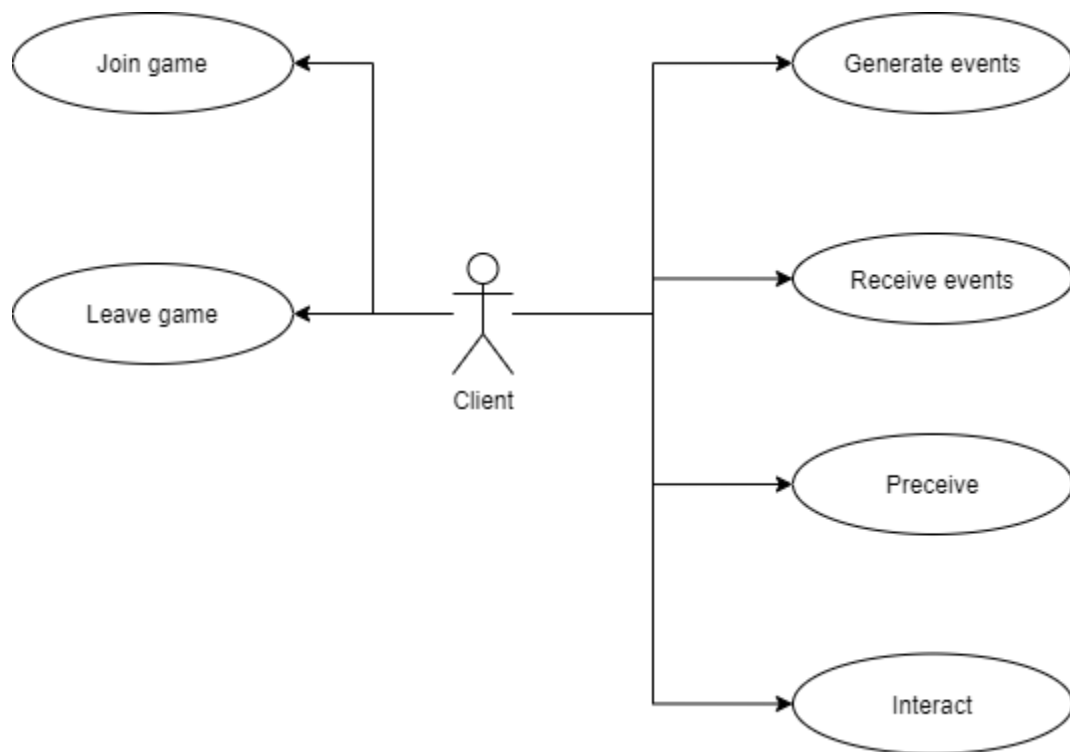


Figure 7, Client component diagram

Some client requirements are shared with Host actors and have already been discussed in the previous sub-section.

Requirement	Join game
Description	The player must have the ability to join a severed game.
Component	Client
Example	<ul style="list-style-type: none">• Player A joins a game Hosted @Locall: Host

Requirement	Leave game
Description	The player must have the ability leave a game.
Component	Client
Example	<ul style="list-style-type: none">• Player A leaves game.

AI-Agent:

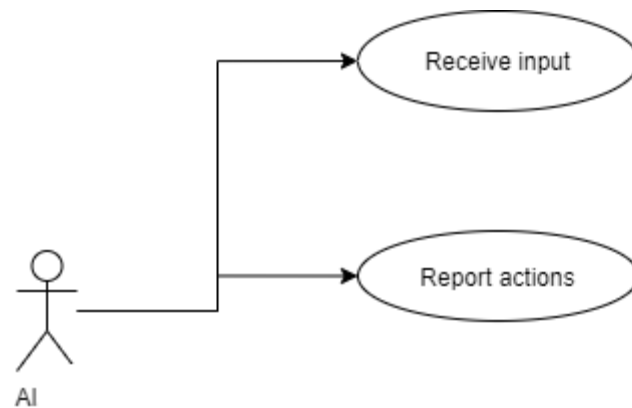


Figure 8, Networked AI component diagram

Requirement	Receive input
Description	The agent must be able to receive input locally or over the network.
Component	AI-agent
Example	<ul style="list-style-type: none">• AI “1” received information to change it’s position.

Requirement	Report actions
Description	The AI must be able to report actions over the network.
Component	AI-agent
Example	<ul style="list-style-type: none">• AI “1” performed an authorized action to move from point A to B.

3.3. Solution

This sub-section aims to discuss the proposed solution to solve the problems discussed in section 3.1.

The proposed solution to the game's framework is Unity's game engine. The engine meets all the game requirements ranging from physics simulation to network architecture. In figure (9) the game's architecture is illustrated.

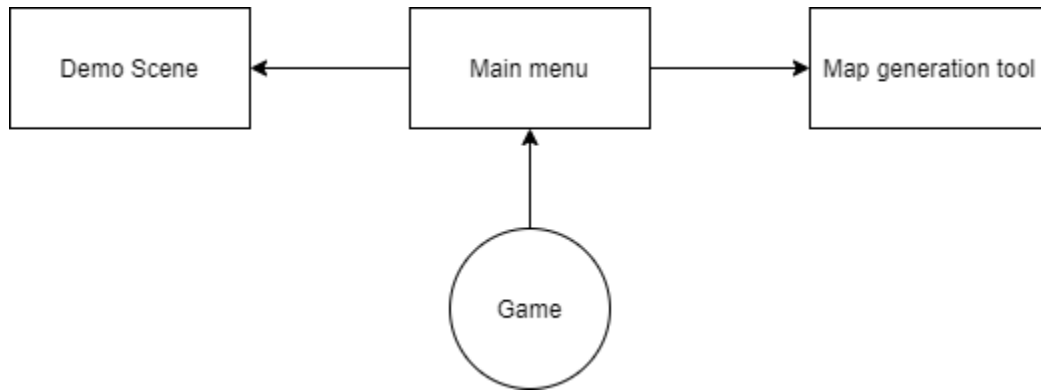


Figure 9, Game architecture

3.3.1. Main menu

The main menu is the face of almost all games, and it is the first thing that a player sees, it also serves as an integral part of connecting the game and avoiding running different fragments of code to experience each section. In the case of this project, the menu needs to connect two scenes (Demo scene and the map generation scene), it also needs to close the game if the users wish to, Figure (10).



Figure 10, Main menu design

Users must be able to quit a scene or navigate back to another scene. thus, a button overlay is added to each scene to allow users to leave their current scene, Figure (11).



Figure 11, demo scene UI design

3.3.2. Demo Scene

The demo scene houses most of the game's functionalities. It is a unity scene that has the environment of the game, playable characters, and AI/NPC-agents. It also is networked, and host/server can start the map while clients can join the server.

Environment:

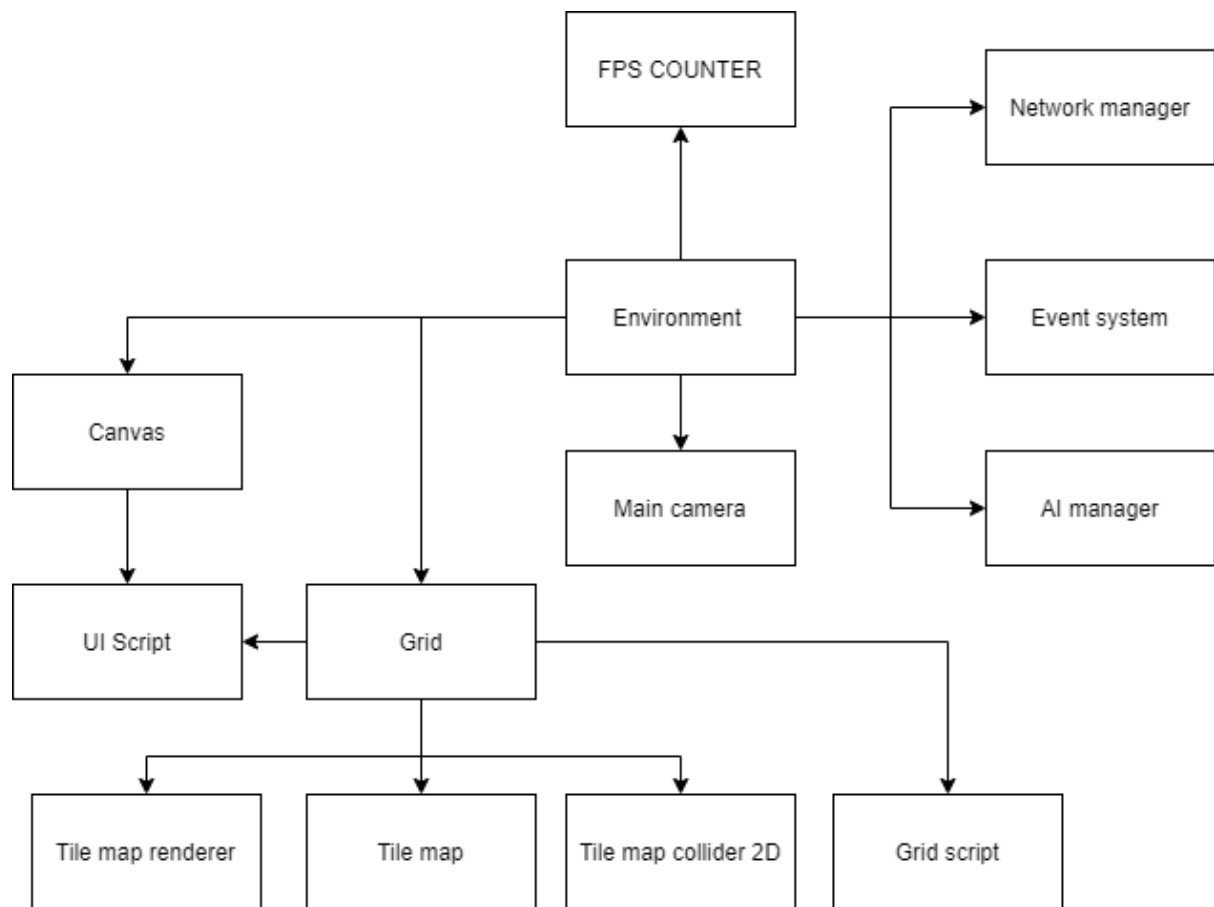


Figure 12 environment architecture

In figure (12), the environment design, relative to unity, is illustrated. each box in the diagram refers to a unity component.

The grid component is a parent game object that houses the following components:

- ❖ Tile map
 - Responsible for storing the tile grid.
- ❖ Tile map collider 2D
 - Adds physics simulation to the grid.
- ❖ Tile map renderer
 - Renders the tile map within the camera in the scene.
- ❖ Grid script

- Allows for adding game logic and custom behaviors.
- **Note:** this component is further discussed in the AI/NPC solution.
- ❖ UI Script/Canvas
 - Allows for adding logic to the UI.

The main camera is responsible for illustrating the scene and the event system allows for unity's event system functionalities.

The AI manager is a parent game object that stores all the AI agents in the scene as children, while the network manager is further discussed in the networking section.

Playable characters:

The playable character is to be stored as a prefab, which has two children, "Ground check" and "Camera". The ground check is an empty game object that indicates the transform position at the prefab's feet while the camera houses the logic of each character's camera.

The character logic is to be stored as a C# script component, it works by utilizing unity's input system and physics system and allows for the following functionalities over the network:

Movement:

Moving the character along the X-axis works by changing the rigid body's velocity, this process is validated and processed by the server. Let us assume Client A wants to move their character to the right, (Positive in the X axis), the user inputs the key "D" a command is sent to the server which holds the integer representation of the input key, in this case it is positive one. The server then changes the velocity of the character using the various Mirror components (explained below in figure (13)) the updated velocity is set and synced for other clients.

Jumping:

Jumping works by first checking if the character is on the ground, that is done by casting a circle from the "Ground check" child game object, if the circle intersects with the ground, then character is indeed on the ground, if that is the case jumping follows the same processes as moving. Simply adjusting the Y velocity via the server.

Attacking/Receiving damage:

When a character wants to attack a ray is casted from the center of the character towards the facing direction, if the ray intersects with another character the server calls a receive damage function stored on the receiving player.

Animations:

Animations are set according to the character's state and velocity. The animator component is further discussed in the implementation section.

Camera:

The camera child component has a C# script that allows the camera follow functionality. The script works by changing the transform position of the camera to the character's position.

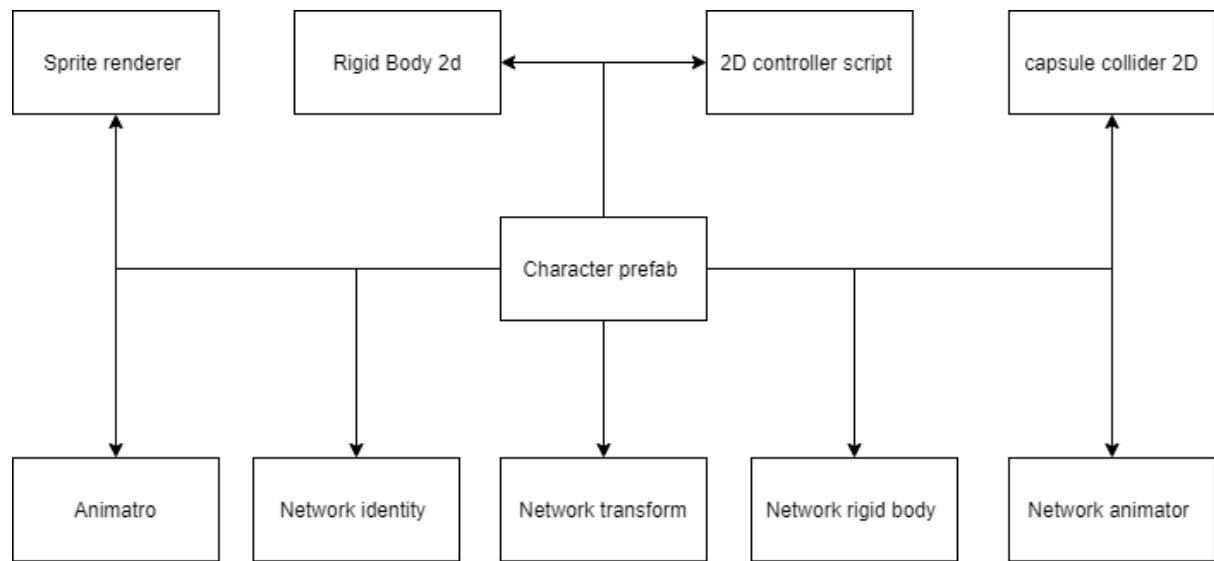


Figure 13, playable character architecture

Figure (13) represents all the components of the prefab. These components are defined and explained in section 2, background.

AI/NPC:

The AI Design is a finite state machine agent that uses A* search to navigate the game. This is to be done via a C# script attached on the agent.

The state machine has been designed to idle, follow, attack, jump and fall, Figure (14).

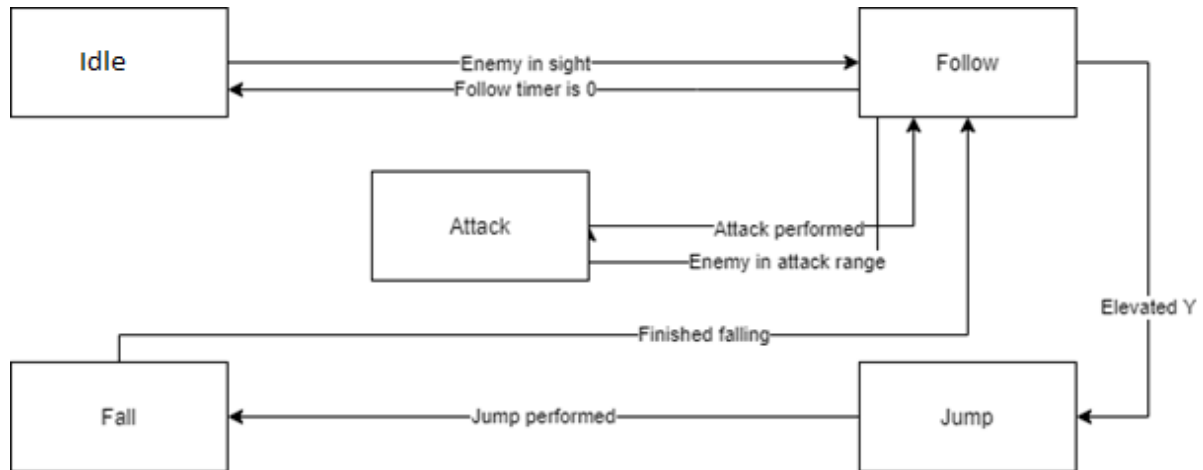


Figure 14, state machine

Idle:

The idle state is the AI's initial state, upon starting the state the AI shoots a ray to look for a player. If a player was found the state is switched to following, however, if no players were found idling will start, the AI would walk towards an initial direction, till the timer reaches zero if so, the AI would change its direction to the opposing direction and would keep patrolling until a player has been found.

Follow:

The follow states implement the A* search algorithm and is responsible for finding and traversing the path. Upon running the state, a ray is shoot towards the facing direction to look for players, if a player is found within attacking range the state is changed to the attacking state, however, if no players were found, following would commence. The AI Uses a graph generated by the Grid script to perform its calculations and it's done as follows:

The nodal generation works by first populating a node list of all tiles that are not on the ground (one cell above a tiled cell and in the case that it is not tiled, to avoid creating unnecessary nodes) (Since the pathfinding is targeted toward ground-moving agents, creating a node for every cell in the tile map is unnecessary and adds complexity to the pathfinding algorithm).

After populating the refined nodal list, a function creates connection between the nodes by comparing the Euclidian distance between the current node and the next node (Using their world positions) If the distance is within a pre-set threshold a parabolic vertex is created and line is casted from Node A and Node B to the parabolic vertex, if the line does not intersect with a tile, a connection between Node A and B is established and their connection list is

updated. (This allows for only connecting nodes that are traversable and reachable). This approach solves the added complexity of having a node for each tile and solves the problem discussed in section 3.1.3 it also allows for simpler physics simulation when agents are traversing a path.

After the graph has been generated in the grid component. the AI accesses the graph and would run the A* search [34] on the graph.

Jump and falling:

The jumping and falling states are intertwined and switched to from the following state based on changes on the Y scale.

Attack:

Based on a preset timer the AI performs attacks, upon finish the timer the state is transitioned to the following state.

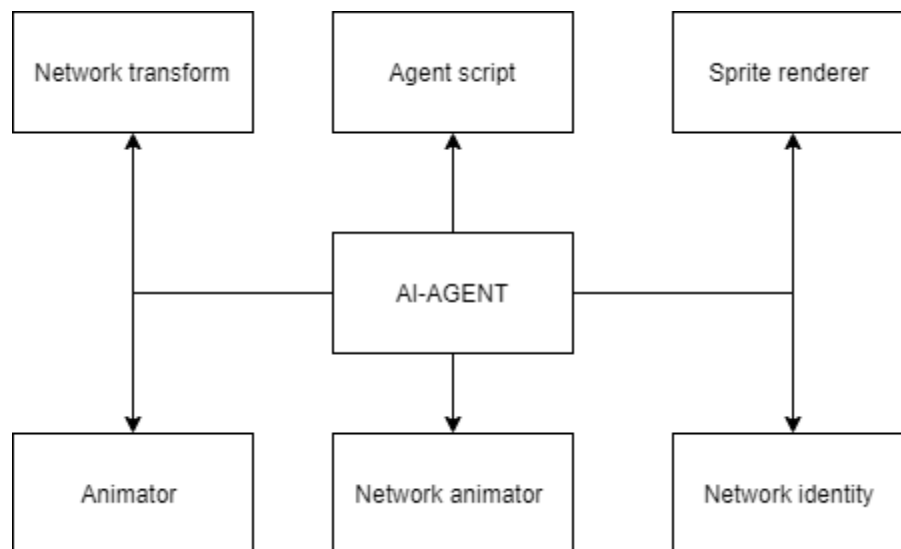


Figure 15, AI Components diagram

Figure (15) represents all the components of the prefab. These components are defined and explained in section 2, background.

Network:

The network manager game object should be responsible for managing the various Mirror components it allows for hosting and joining games. In figure (16) its components are illustrated. The transport type component has been set to KCP as the transport layer of the server. Interpolation and synchronization are managed via other Mirror components that would be attached to objects which need these features.

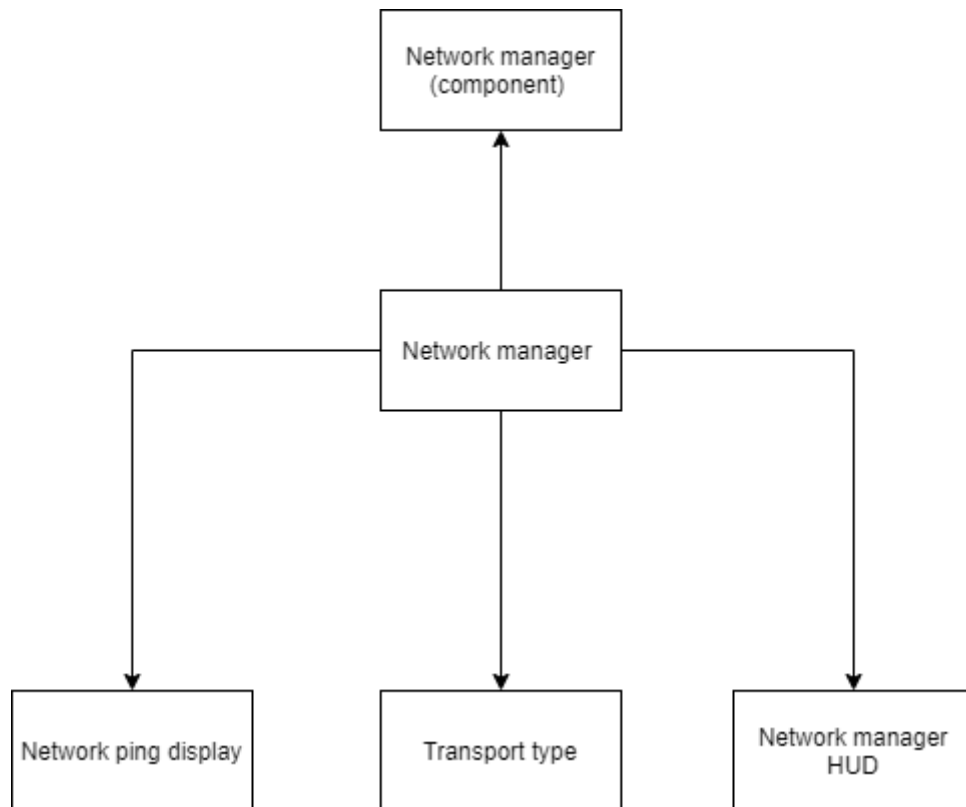


Figure 16, Network manager component diagram

3.3.3 Map generation tool

The map generation tool is a unity scene that illustrates two algorithms, cellular generation and Splunky generation. It should have a UI that switches between the scenes in the following structure:

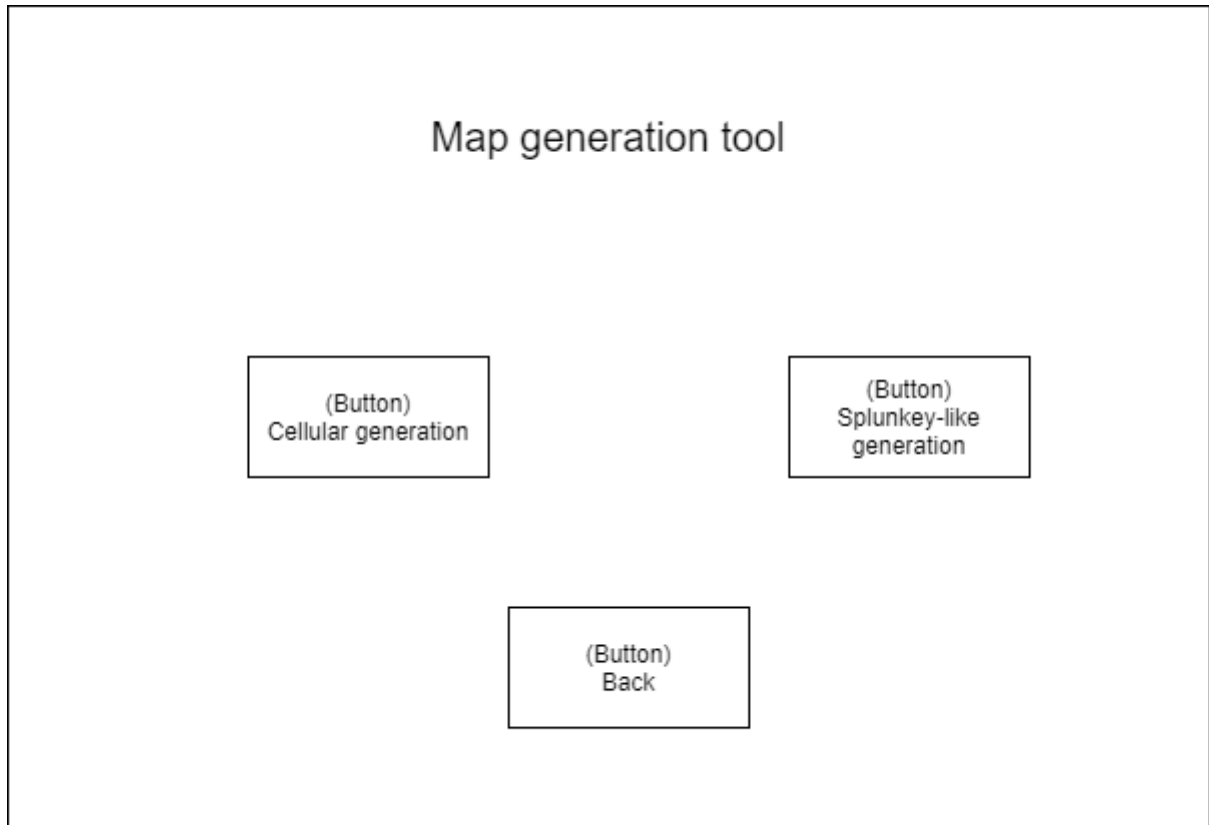


Figure 17, map generation scene UI

Each scene should also allow for a back button, to switch back to the main menu.

The structure of the scene has been designed to fit the requirements of the tool and is illustrated in Figure (18).

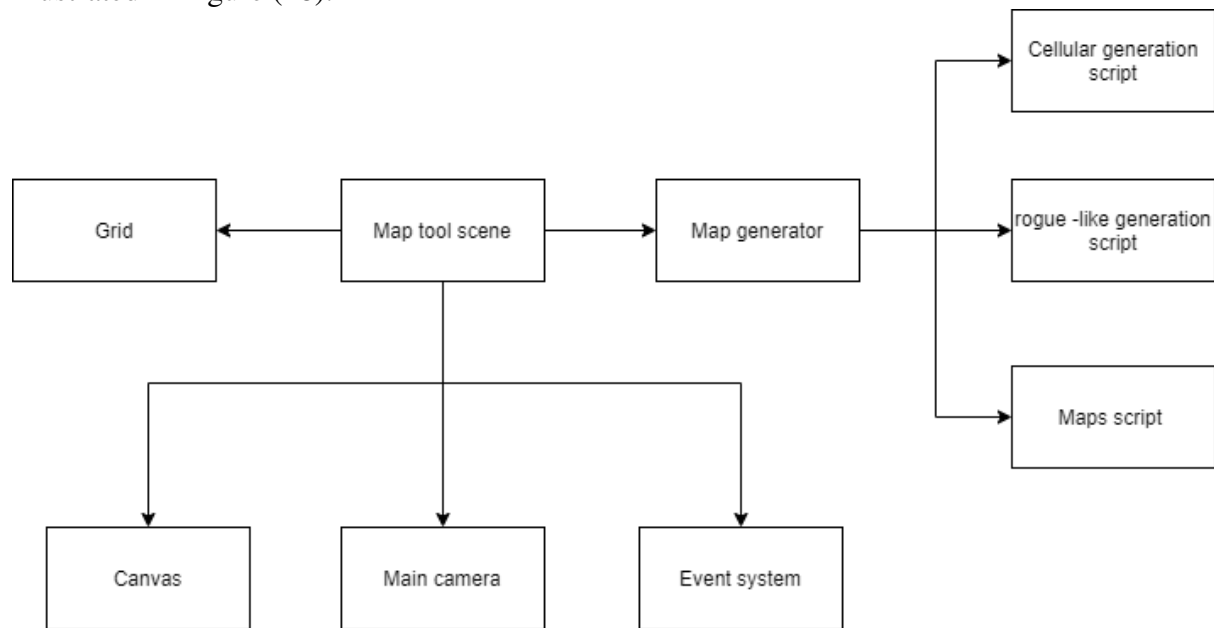


Figure 18, Map generation Component diagram

Cellular automata generation script:

The script should simulate cellular generation by applying pre-defined rules to a randomized grid of cell.

Note: Cellular generation is further discussed in the implementation section.

Spelunky generation script:

The Spelunky generation script would be a C# component that implements an algorithm I have designed based on the game Spelunky [35]. The algorithm aims to generate a 2D side-scroller type of map, which always has an open path for players to travers from the starting position to the end position.

Prior to running the algorithm, a 2D grid/array and a pre-set of room templates is required.

Rooms are a 2D array which have a 4X4 size, Figure (19), that each cell represents a tile, if the value at a cell is “1” then the cell is tiled, else it is empty.

```

roomOpenSidesAndTop = new int[,]
{
    {1,0,0,1},
    {1,0,0,0},
    {1,0,0,0},
    {1,0,0,1}
};
  
```

Figure 19, Room template array

Four templates are required for the algorithm to run correctly.

1. A room with four entrances.
2. A room with two entrances (right and left).
3. A room with three entrances (right, top and left).
4. A room with three entrances (right, left, bottom).

Once this information is available, the algorithm would pick a starting position within ($X < 5$ & $X > 0$) and ($Y < 5$ & $Y > 0$) a random room is then constructed at that coordinate. The algorithm then chooses a random direction (South, west, north, east) and the generating cursor is moved in that direction. A random room pre-set is chosen and if the room is compatible with the previously generated room the generation would keep iterating in the same way, another random direction is chosen and the same processes is repeated, however, if the room is not compatible with the previous room, the generator would keep randomizing until a compatible room has been found, Figure (20).

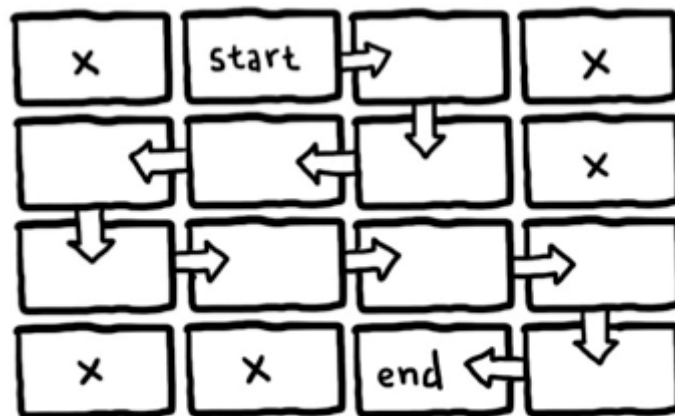


Figure 20, Spelunky type generation

3.4 Alternatives and justification

This sub-section aims to discuss some design choices which have not been justified in the background section.

The playable character aims to implement unity's physics system, that relies on the Rigid body component, another solution is to manipulate the character's transform while checking for collision by looking at tiled cells in the grid, however, the rigid body system has been favoured due to the velocity calculations provided by unity which makes moving the character more fluid and smoother while simulating human-looking jumps.

Another design direction was using the KCP transport protocol over the TCP protocol. However, "TCP is designed for traffic (the amount of kilobits per second of data that can be transmitted), which focuses on the full use of bandwidth. While KCP is designed for the flow rate (the amount of time it takes to send a single packet from one end to the other), with 10%-20% bandwidth waste in exchange for transmission speed 30%-40% faster than TCP. TCP channel is a grand canal with very slow flow rate, but very large flow per second, while KCP is a small torrent with the rapid flow. KCP has both normal and fast modes, achieving the result of flow rate increase". (KCP developer Sky Wind, <https://github.com/skywind3000/kcp>).

4. Implementation

This section discusses the implementation processes of the game and provides a brief explanation for reproducing the game in Unity, it is divided into three sections (Main menu, Demo scene, Map generation scene) each section is further divided into its core components.

Setting up the project first requires Unity 2019 (and above) and visual studio. After creating an empty project, the following steps can be replicated.

4.1. Main menu

The main menu is the simplest part of the implementation, it is created using Unity's canvas and the scene's hierarchy hosts a main camera, canvas, menu manager and an event system, Figure (22).



Figure 21, Main menu

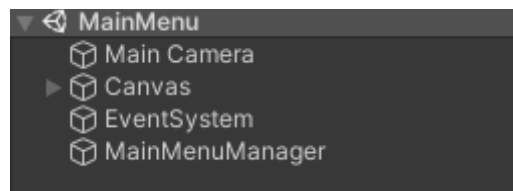


Figure 22, Main menu scene hierarchy

The buttons in Figure (21) use the on click event system to use functions located in the Main Menu Manager game object, specifically in the "MenuManager.cs" component.

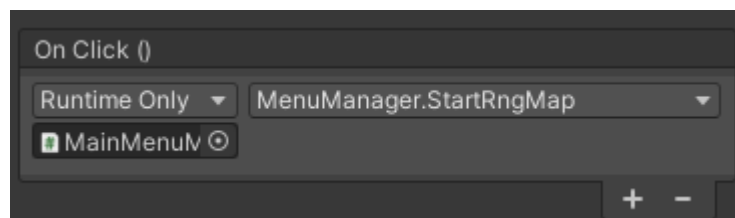


Figure 23, on click event setup.

```

0 references
public void StartDemo()
{
    SceneManager.LoadScene("Demo");
}

```

Figure 24, Code fragment for loading a scene.

And once a button is clicked, the on click function is triggered executing it's assigned function in the case of the "Start" button, it would load the "Demo" scene.

4.1. Demo scene

the demo scene is the game's main scene, where the players can play over the network and face the AI-agents. This scene has been implemented according to the proposed design in section 3, and for it to be functional, the hierarchy is populated with a grid, network manager, AI-manager, canvas, and an event system.

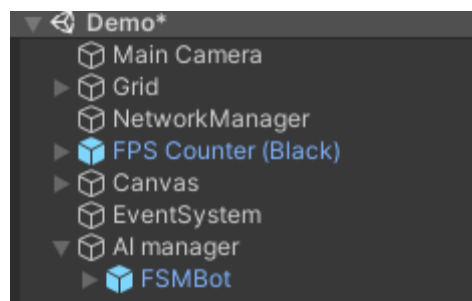


Figure 25, Demo scene hierarchy

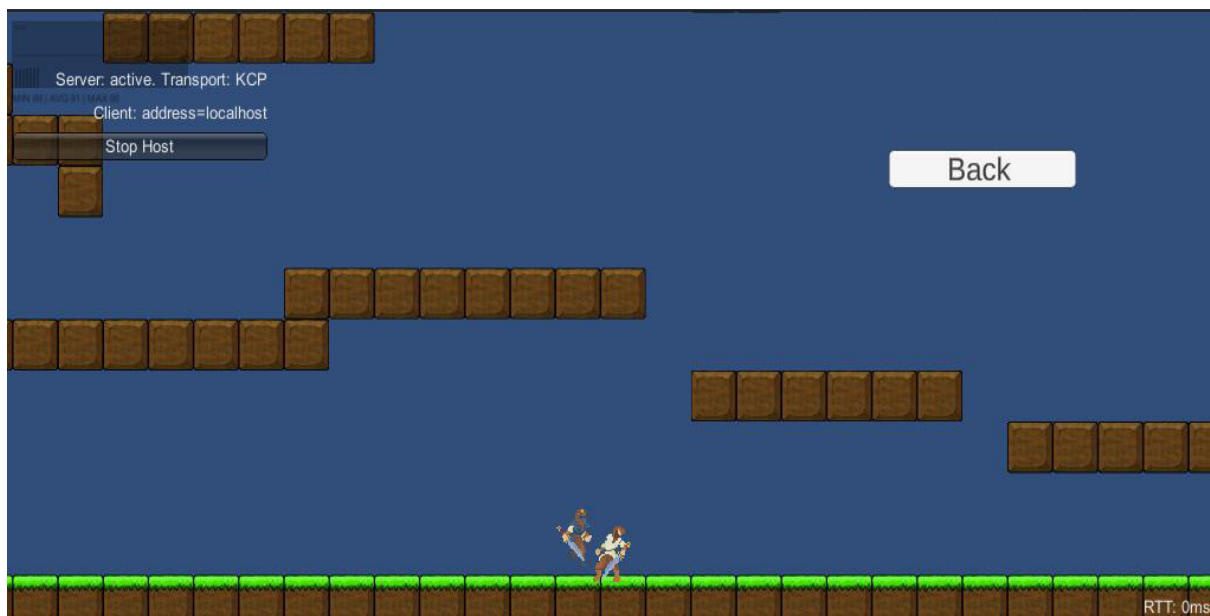


Figure 26, Demo scene sample

4.1.1. PHGrid.cs

The grid game object stores and manages the “**PHGrid.cs**” C# script, that generates the nodal grid of the map (which is latter used by the AI-Agents). The PF Grid component allows for 2-D side scrolling path generation. It is a class that ultimately creates nodes for the agents to run different algorithms on (statically called by other game objects). It uses nodes to represents the way points in the generated graph, a node is a class that hold information about its state (tiled or not), parent node, a list of adjacent nodes, heuristic values (can be set when running PF grid functions) and its world position. It could be considered a tile in the tile map, with added functionality.

Upon calling the Start () Function, all the variables are initialized and allocated, then the node list is generated by looping over the tile map and setting a node corresponding to each tile at the map. CreateConnections () is called to further refine the nodal map by looping over the generated node list, if the looped over nodes are within a pre-defined threshold distance according to their world Euclidean distance, Figure (27).

```
public float FindDistance(Node startCoord, Node endCoord)
{
    float dstX = Mathf.Abs(startCoord.worldPos.x - endCoord.worldPos.x);
    float dstY = Mathf.Abs(startCoord.worldPos.y - endCoord.worldPos.y);

    if (dstX > dstY)
        return 14 * dstY + 10 * (dstX - dstY);
    return 14 * dstX + 10 * (dstY - dstX);
}
```

Figure 27, Find distance function

If that is the case a parabolic vertex is created and a line is casted from Node A and Node B to the parabolic vertex if the line doesn't intersect with a tile a connection between Node A and B is established and their connection list is updated. (this allows for only connecting nodes that are traversable and reachable).

4.1.2. Network manager

The network manager is the essence of networking the scene, it is a component that implements Mirror's assets to facilitate and instantiate the network.

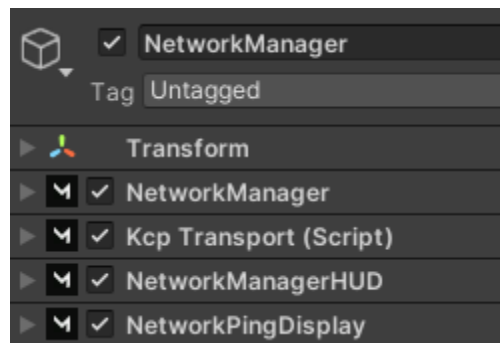


Figure 28, network manager component

It has four components, networkManager, Kcp Transport, Network manager HUD and network ping display. **It is essential to setup the network manager with the correct transport protocol and the correct spawn able player prefab for its functionality.**

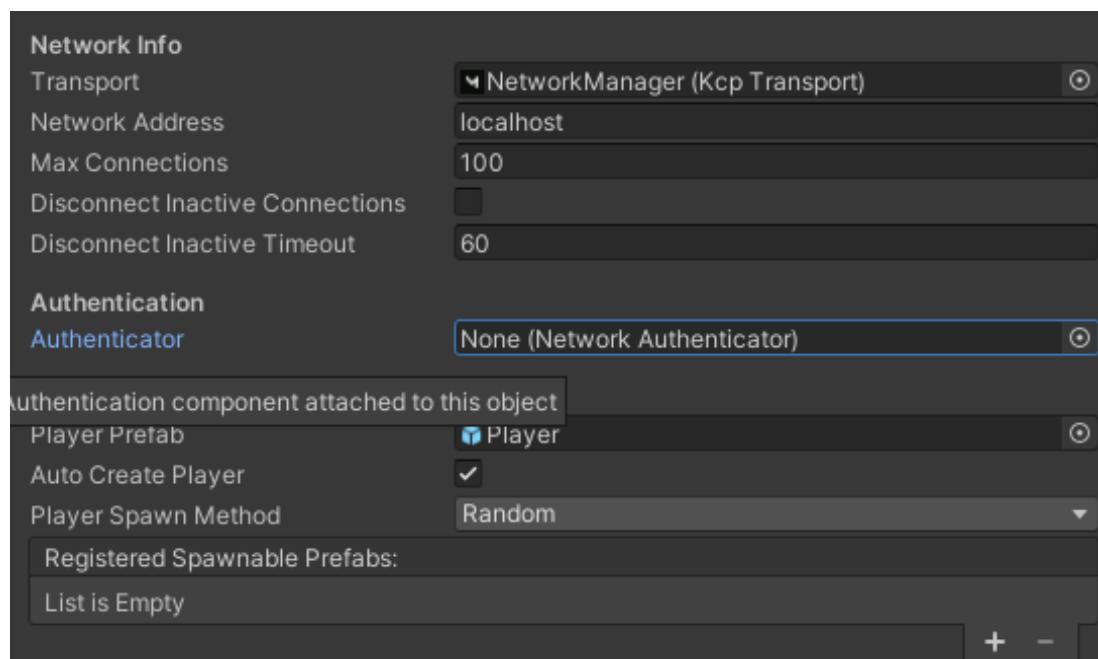


Figure 29, setting up the network manager component

The KCP transport component allows for setting up the protocol and its ports, for this project the port 77777 was used as the standard. Note: the protocol could be swapped to TCP and the programme would run normally.

4.1.3. 2D Player prefab:

The prefab is used by all players over the network to play the game, it is also utilized by the network manager to spawn clients into the game. It has a sprite renderer, rigid body, capsule collider, animator, network identity, network transform, network rigid body, network animator and a player controller.

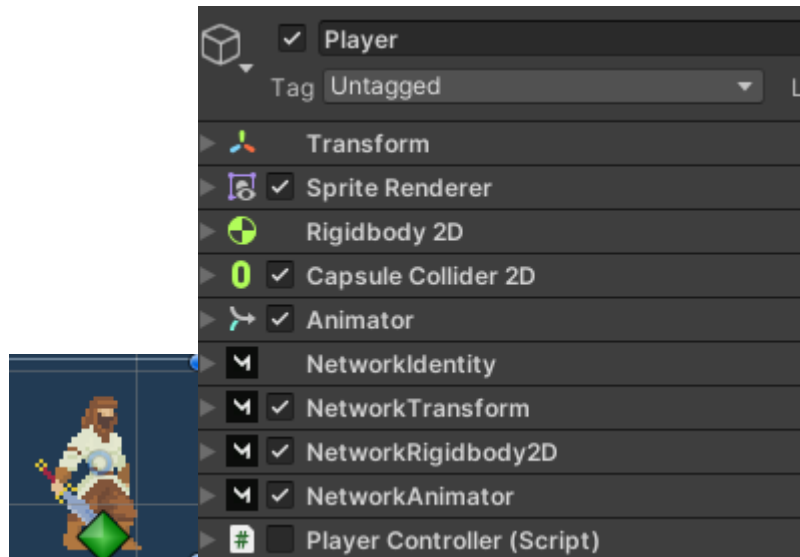


Figure 30, player prefab components

The animator runs and host the sprite's animation:

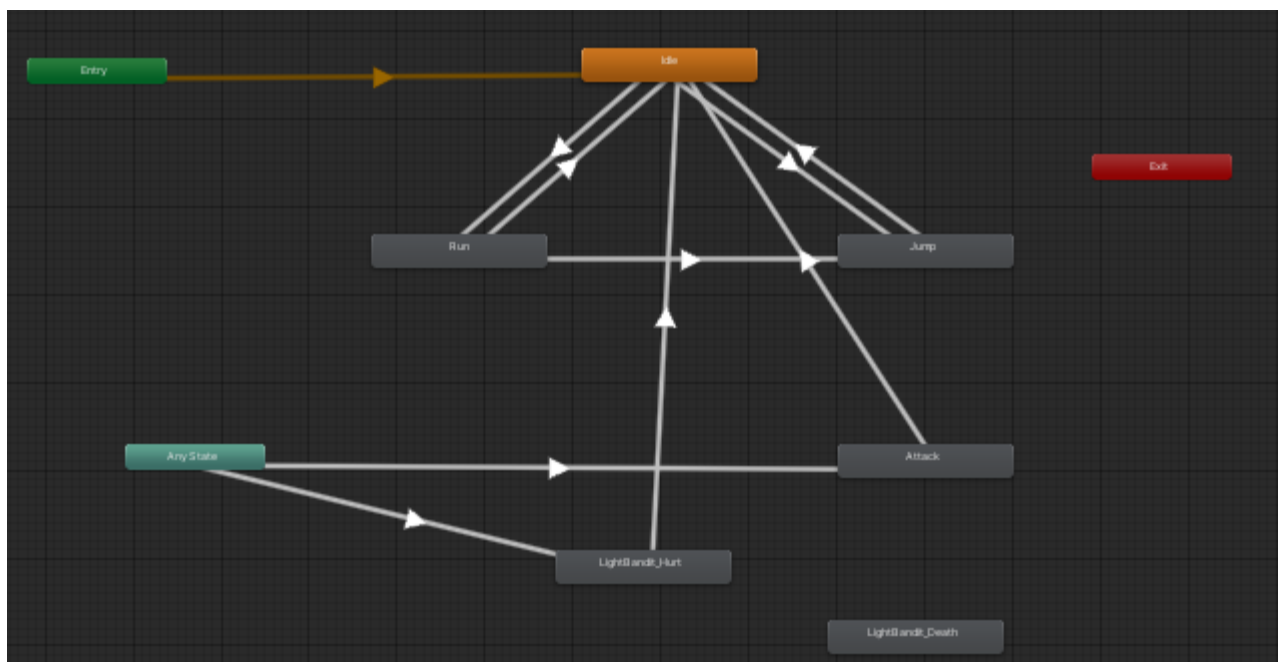


Figure 31, Animator diagram for the player's prefab

The main controller script is the “PlayerController.cs” which in essence allows the control of the character over the network and allows multiple users to control different instances of the prefab. The component is inactive by design and is only activated in the On Start Local player function which limits the input from client to their assigned prefab.

```

public override void OnStartLocalPlayer()
{
    this.enabled = true;
    transform.GetChild(1).gameObject.active = true;
}

```

Figure 32

The prefab uses unity's rigid body system to detect physics and simulate it based on the user's input,

```

1 reference
private void HandleMove()
{
    float XDir = Input.GetAxis("Horizontal");
    if (XDir > 0) lastPlayerDir = 1;
    else if (XDir < 0) lastPlayerDir = -1;
    anim.SetFloat("speedval", Mathf.Abs(XDir * speedval));

    CmdMove(XDir, onGround);

    if (onGround)
    {
        anim.SetBool("jump", false);
    }
}

```

Figure 33, Handle move function

A command function is then sent to the server for validation.

```

[Command]
1 reference
void CmdMove(float XDir, bool onGround)
{
    rb.velocity = new Vector2(XDir * speedval, rb.velocity.y);
}

```

Figure 34, Server command request

If the user has authority over the object (a client using the prefab) the command is validated and the change to the object's velocity is applied, the different Mirror components update and sync the state of the object over the network for other client to see the change. The player can also deal damage to other players by shooting a ray towards the facing direction, if the ray intersects with an object on the player-layer a command is sent to deduct health points from the opposing player, if validated the Sync VAR health, is updated and all other clients know the current updated health of the player.

Note: the animations are set in the same script and are update accordingly while changing the states of the object.

4.1.4. FSM prefab:

The main opponent in the game is the FSM agent, which uses a combination of finite state machines with A* search to function. Similarly, to the player prefab, some Mirror components are being used to sync the agent's state over the network, however, there is an absence of a network rigid body 2D since the agent doesn't use unity's physics system and it's movement is manipulated using it's transform.

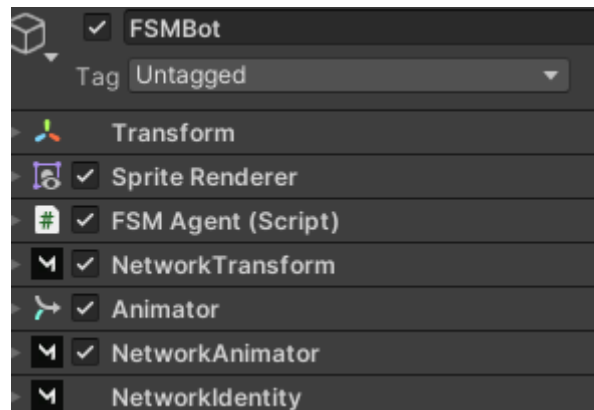


Figure 35, FSM bot components

The FSM agent C# component is the core of the AI and give's it all its properties. In essence, the code state is dependent on the agent's state:

```
Unity Message | 0 references
private void FixedUpdate()
{
    CheckGroundState();

    switch (currentState)
    {
        case State.Ideling:
            IdelState();
            break;
        case State.Following:
            FollowState();
            break;
        case State.Attacking:
            AttackState();
            break;
        case State.Jumping:
            JumpState();
            break;
        case State.Falling:
            FallingState();
            break;
    }
}
```

Figure 36, the main state machine loop

the idle state calls its corresponding function. At each iteration the agent shoots a ray at its facing direction which is initialized to “one” (the positive direction on the x-axis), if the ray intersects with another player the timer is reset and the state is changed to the follow state, else; the agent would keep slowly walking in the positive x-axis and when the timer reaches “zero” the flip function is called, flipping the agent and moving him towards the negative x-axis.

The follow state depends on A* search to find and chase the player. Initially, ray is shoot and if the player is too close the state would be switched to attacking.

```
RaycastHit2D hit = RayShoot(transform.position, 1f, facingDir, playerLayerMask);
if (hit.collider != null)
{
    hit.collider.GetComponent<PlayerController>().ReciveDamage();
    attackTimer = 100f;

    SwitchState(State.Attacking);
}
```

Figure 37

If the ray doesn't intersect with a player and the follow timer is bigger than “zero”. The find path function will be called passing the current position of the bot and the destination position as parameters. This function is an implementation of A* pathfinding and it differs from the normal A* in the neighboring nodes search, where instead of looking into and 2-D array of cells we here look in list of nodes.


```

//iterates through openSet and finds lowest FCost
Node node = openSet[0];
for (int i = 1; i < openSet.Count; i++)
{
    if (openSet[i].FCost <= node.FCost)
    {
        if (openSet[i].hCost < node.hCost)
            node = openSet[i];
    }
}

openSet.Remove(node);
closedSet.Add(node);

//If target found, retrace path
if (node == targetNode)
{
    RetracePath(seekerNode, targetNode);
    return;
}

//adds neighbor nodes to openSet
foreach (Node neighbour in node.connections)
{
    if (closedSet.Contains(neighbour))
    {
        continue;
    }
}

```

Figure 38, A* search

It is important to call the flip function according to the current agent's location for the code to run correctly, in the scenario; a comparison between the current node and the next node is made and depending on the x-axis difference the flip function is called.

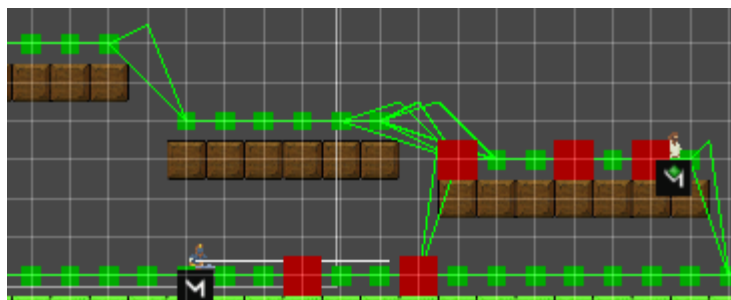


Figure 39, AI traversing a path

An integral part of this implantation is simulating jumps to look as human as possible. This is achieved by detecting a difference between the current y position and the next node y position and if there is a difference, a vertex is set between the two points (based on average x and highest y) the agent would then move towards this point and then switch to the falling state to reach the desired node.

Note: the state of animations is updated according to the state that the agent is in.

4.2. Map generation scene

The map generation scene is the demo scene for the Random map generation tool. It consists of a main menu overlay, Figure (40), and two algorithms discussed in the design section.

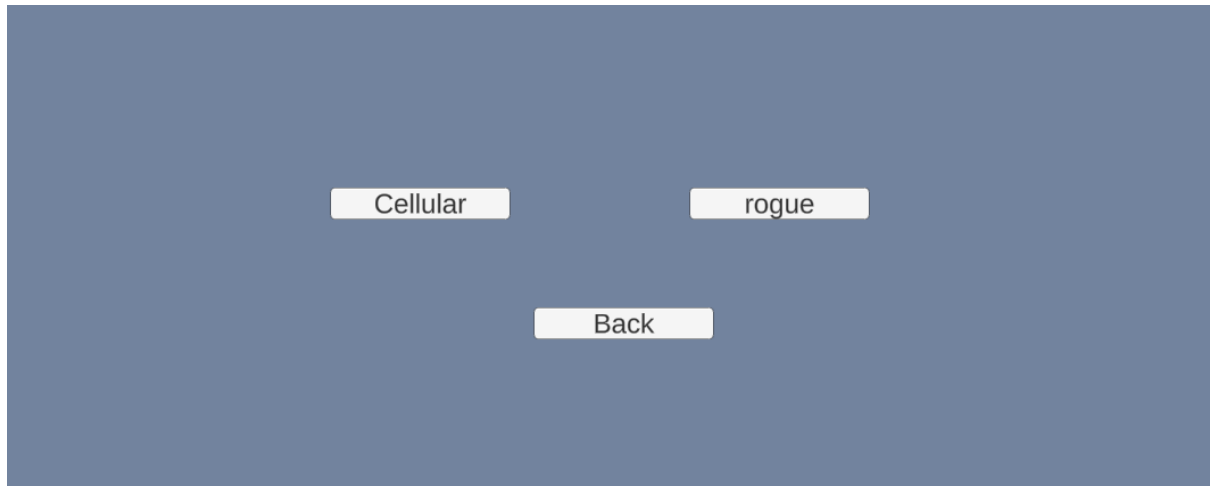


Figure 40, map generation main UI

The scene's hierarchy is populated with the following game objects:

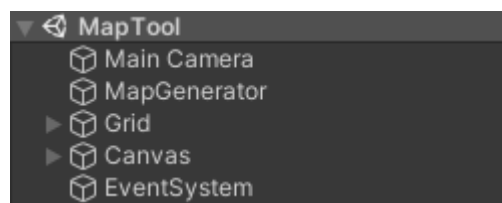


Figure 41, map generation scene hierarchy

The map generator game object houses the two C# scripts responsible for the generation process. The rest of the components are defined in section 2.5.

4.2.1. CellularGeneration.cs

Upon clicking on the “Cellular” button in figure x, the canvas shuts off the menu UI and starts running this script with its corresponding UI. Which is a demo representation of randomly generated cave like structures using the cellular automata algorithm.



Figure 42, Cellular generation scene

the script uses the Unity's update function to generate, and render maps each frame.

```
void Update()
{
    map = new int[width, height];
    density = (int)desnitySlider.value;

    birthLimit = (int)birthLimitSlider.value;
    deathLimit = (int)deathLimitSlider.value;
    iterations = (int)iterationsSlider.value;

    map = GnerateNoise(map, density);
    int count = 0;
    while (count < iterations)
    {
        map = doSimulationStep(map);
        count++;
    }

    RenderMap(map, tileMap, tileBase);
}
```

Figure 43, Cellular generation main loop

After initializing the appropriate variable, either manually or using the sliders in the UI. A noise map is generated via the Generate Noise function which takes in the grid array of cells and the density of the generation as parameters. The function simply populates the grid with either “1” or “0” which signals if the cell should be populated or not, this done using the Random () function provided by Unity’s math library.

After the grid has been populated with noise values, a while loop runs the cellular simulation on the map based on the given number of iterations. It does so by applying the inputted values via the slider (birth limit, death limit) to each cell. For example, if the birth limit was 3 and the death limit is 4, when the doSimulationStep function runs the simulation, it would look at each cell, if the cell is empty and has 3 or more alive neighbours, it would be tiled else if the cell is alive and has 4 or more alive neighbours it would die.

After this map has been generated, render map function would take it as input with a reference to the tile map and the tile base and would render the map each frame using the following command:

```
tilemap.SetTile(new Vector3Int(x, y, 0), tile);
```

Figure 44

4.2.2. Rogue-likeGeneration.cs

Like the cellular generation script this script creates, and renders maps each frame via the Update function.

The generation starts by first choosing a random starting position, it then populates this coordinate with a random room (rooms are pre-defined 2D arrays of a 4x4 dimension, refer to section x) a random moving direction is set, and the generation cursor is moved towards that direction if this newly allocated position is within the bounds of the map, a random room is chosen and if that room is compatible with the previously generated room that room would be populated in the map grid, else the algorithm would keep looking for a room that's compatible until one has been found. This processes would keep iterating until a pre-defined value for generation is met.

Upon finishing the generation the rendermap function is the same function discussed in 4.2.1.

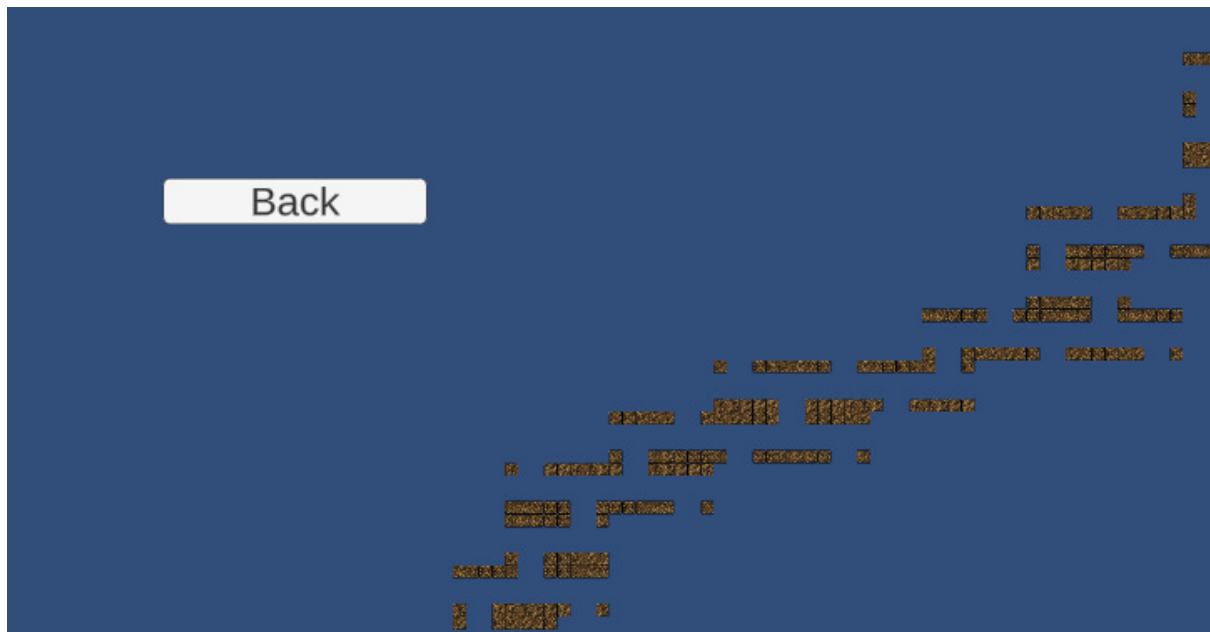


Figure 45, rogue like generation scene

5. Evaluation

This section discusses the evaluations done on the game. The following is the criteria that the evaluation aims to illustrate:

- Frames per second performance.
- Round trip time for clients.
- Playable character completeness.
- AI behaviour.

This evaluation criteria have been tested on a local machine with the following specs:

CPU	Ryzen 3700x
GPU	RTX 2060 super
RAM	32 GB DDR4

The other machine mentioned later in this section had the following specs:

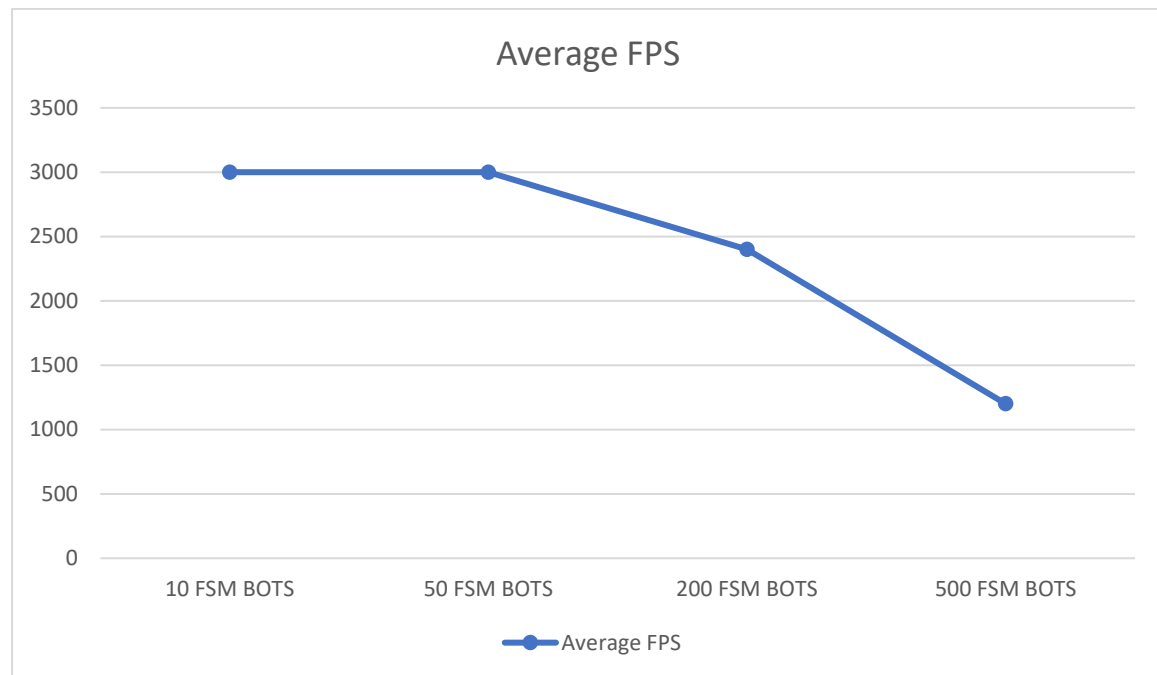
CPU	Intel I9 9 th gen
GPU	RTX 3080
RAM	32 GB DDR4

5.1. Frames per second

This sub-section aims to evaluate the performance of the game by comparing the FPS (Frames per-second) on different loads. The testing loads are:

- ❖ Four clients with 10 FSM bots (control load).
- ❖ Four clients with 50 FSM bots.
- ❖ Four clients with 200 FSM bots.
- ❖ Four clients with 500 FSM bots.

After testing the game on the mentioned grounds, these results have been optioned:



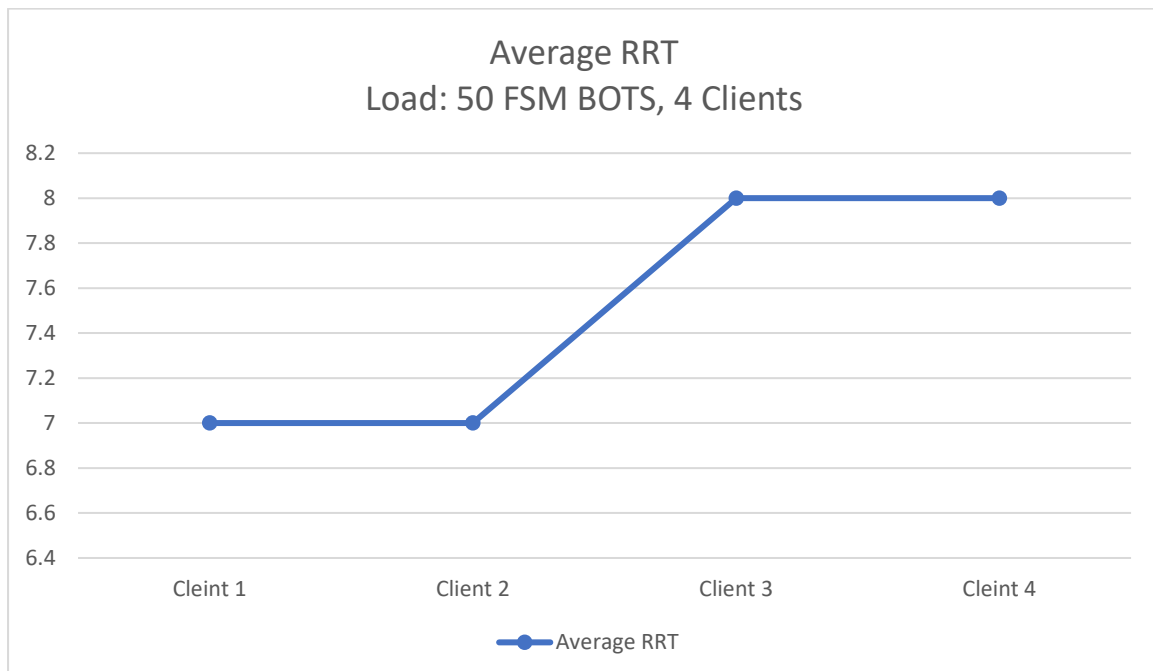
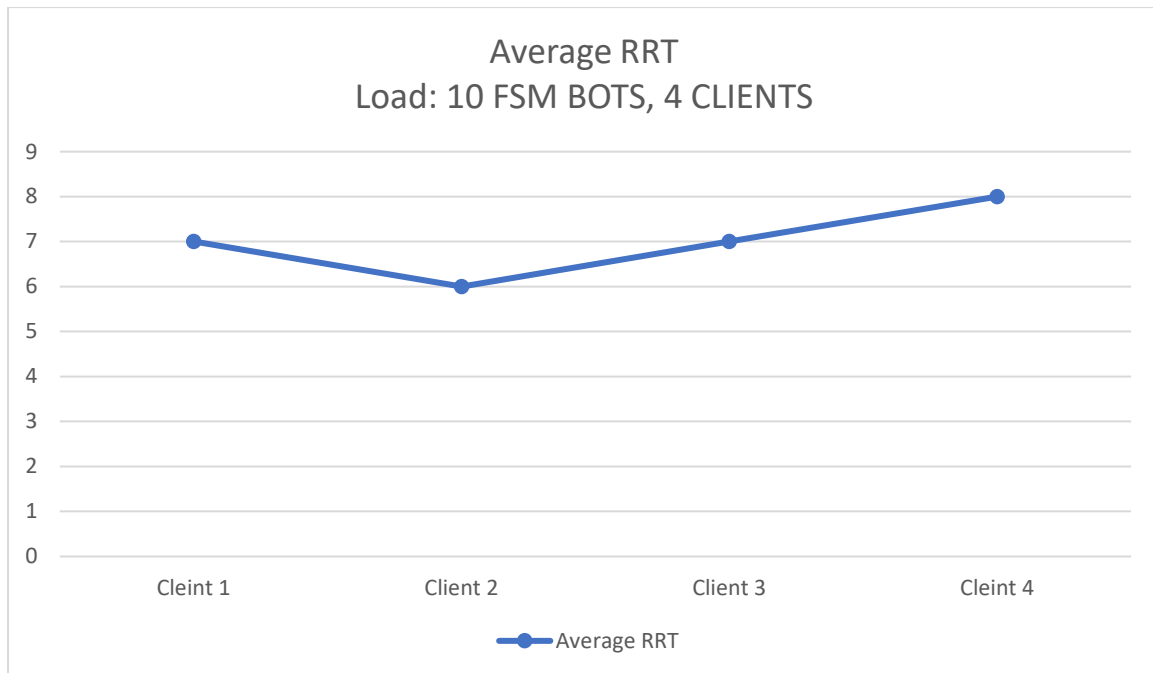
Note: the average FPS represents the average FPS of the four clients.

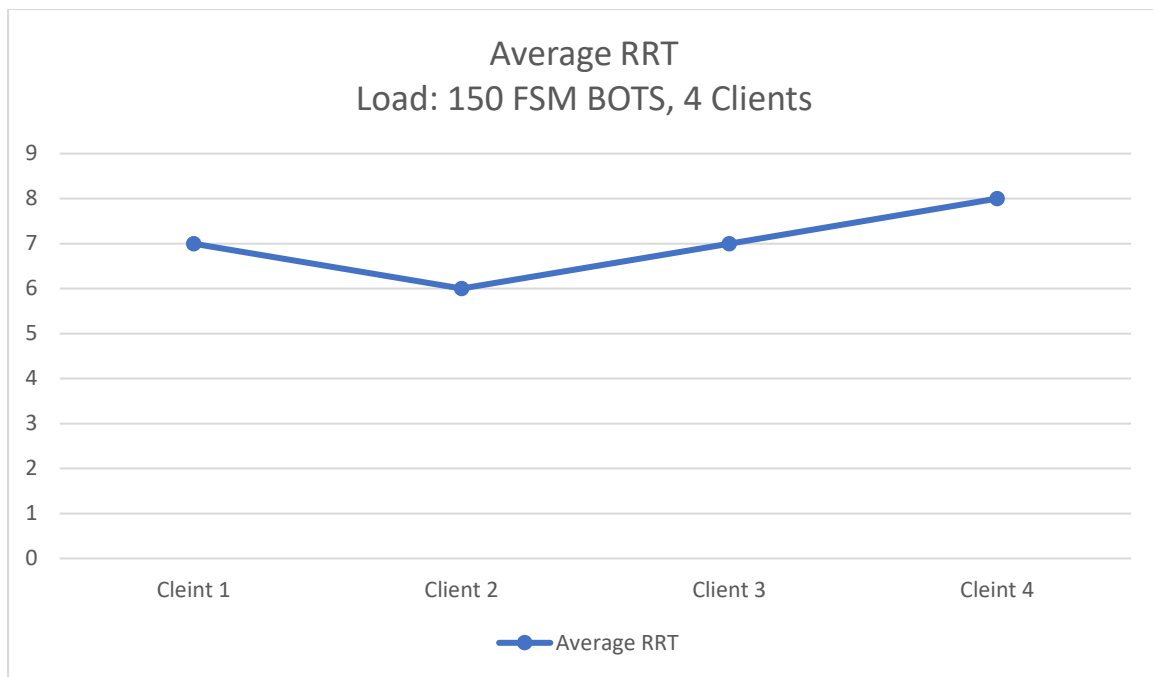
According to the evaluation it appears that there is a coloration between the amount of FSM bots in the game and the average frames per second, however, this does not affect the playability of the programme since games run smoothly on most modern monitors at 60 frames per-second [36].

5.2. Round trip times

This sub-section aims to evaluate the RTT (round trip time) of the client's connection to the game both on a local server and through a VPN tunnel to Kuwait where friends helped me setup the server there.

5.2.1. Locally

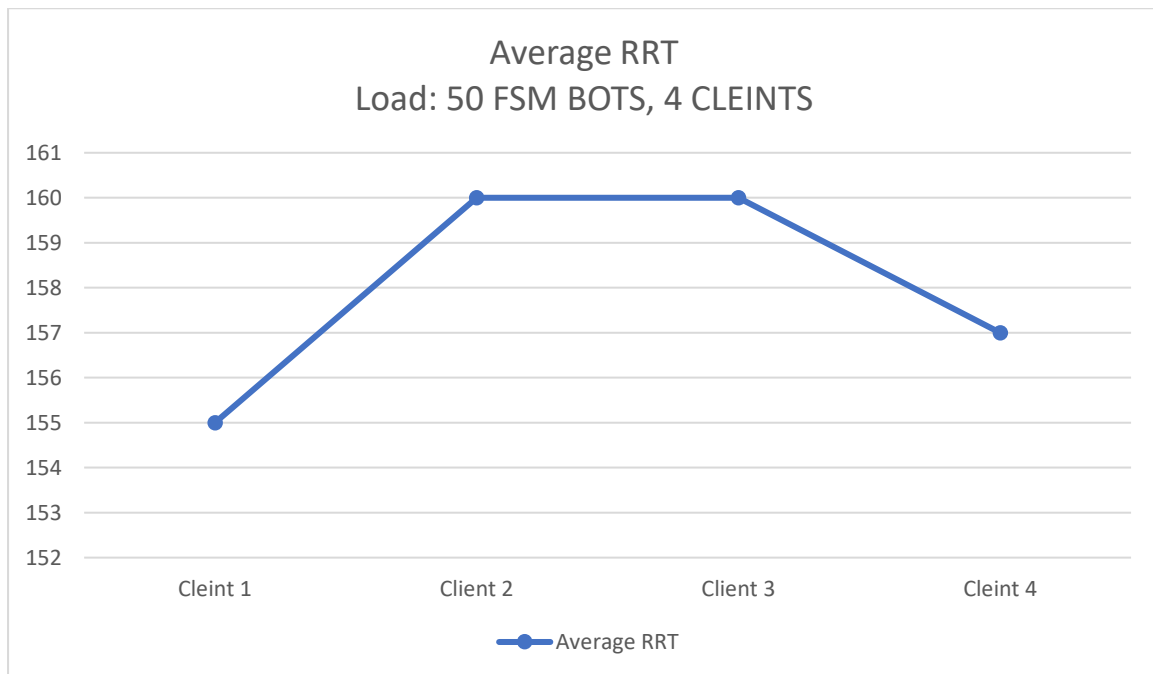
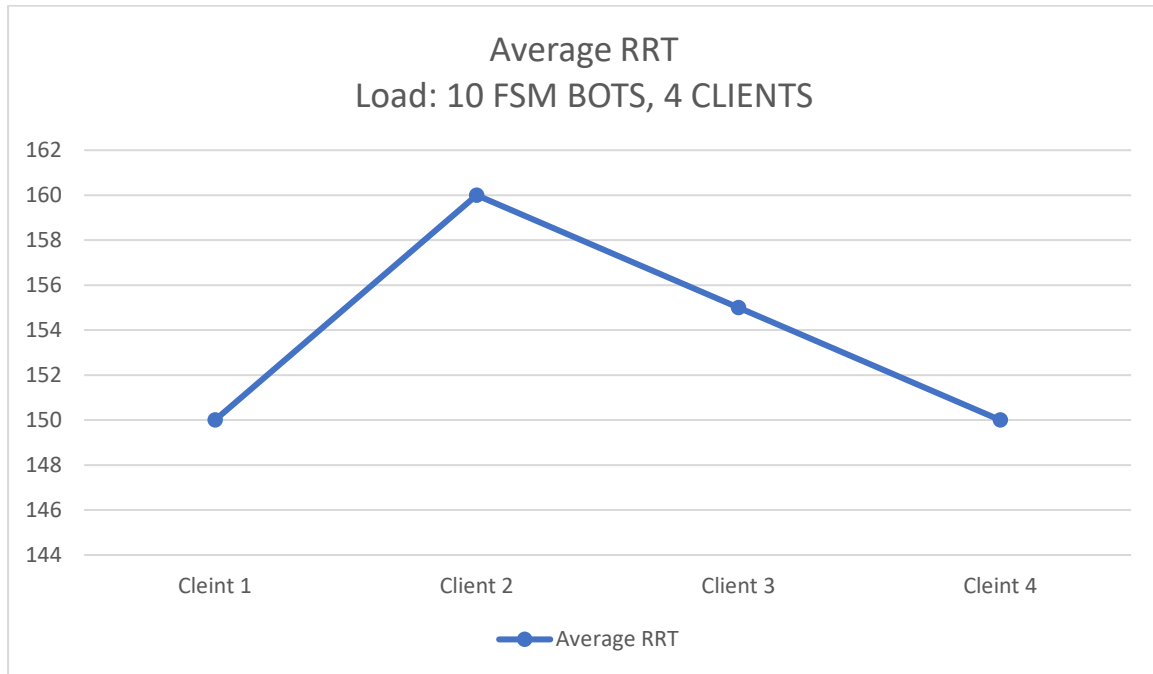


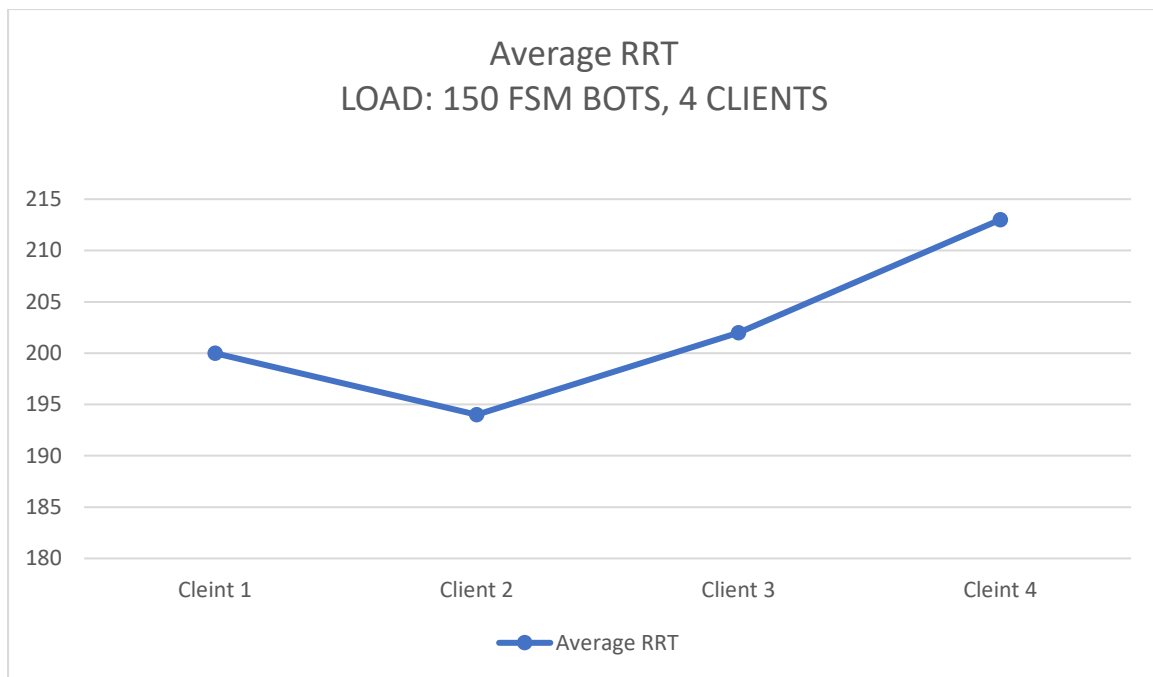


The average RRT times remained stable and did not exceed 8 ms, there was no coloration between different loads on the RRT locally.

5.2.2. Over the tunnel

A VPN tunnel has been set at Kuwait, where a clients connected from the UK to the server setup there.





It appeared that there is a correlation between the different loads and the amount of FSM bots in the game and the RRT times over the tunnel.

5.3. Payable character completeness

The playable characters achieved their proposed design successfully and implemented all the features mentioned in the requirement.

5.4. AI

The AI successfully implemented the finite state machines with a complete integration of the A* search algorithm.

6. Future works

There have been numerous avenues to expand upon for future works. Firstly, performance wise, there are many axes that performance could have improved on, most notably, is evolving the pathfinding to a system that implements heaps and unit requests on a multithreaded way, where the pathfinding script would run on a separate thread on the server. Furthermore, the playable characters have used the gravity system provided by unity, which includes many functionalities that the characters might not need this led to an abundance of features which run on the server with no real use case. A specialized gravity system for the playable character could in theory be a better solution.

Secondly, networking wise, it would have been a great addition to the project if there was a dedicated lobby system, where players would join a lobby and the host would start the game once all players are ready. Furthermore, the Mirror library provided a great networking solution, but it also removed many of the low-level networking technicalities that ultimately sacrifices a lot of improvement grounds for the server, this is like the case of using Unity's gravity system where a lot of features would run unnecessarily.

Thirdly, map generation has been an aspect of the project which has not been expanded as much as other avenues. in the future I aim to further develop the generation system, where each time the server is started a world is randomly generated that works around the physics of the game and has a high playability factor.

Lastly, although the project was centred around the technical and computational aspects of game development, I hope to add game design features to the project, this includes a story, progression, and character creation system.

7. Conclusion

In conclusion, the project aimed to implement systems found in 2D games such as networking, AI, and map generation. The game successfully implemented the design proposed and the project accomplished its goal, however, random map generation was not fully implemented in the sense that maps generated were not playable by the players and only a demo was implemented.

Many evaluations have been conducted on the game and results showed that although the game/programme was always playable and running on high loads a correlation appeared between the number of bots in a game and the average frames per second, the same is applied to round trip times RRT.

8. Reflection on learning

The game was a very interesting project to work on, I have learned many theoretical and technical skills while developing the game. Firstly, I have gained the ability to organize and structure my work to fit and meet specific deadlines, a skill that this project showed me how much I was lacking in. Secondly, I have thoroughly understood many artificial intelligence algorithms and techniques that prior to starting development, I had only known theoretically. Thirdly I have come to understand networking and its technicalities in a more coherent way and learned how much knowledge is required to build systems from scratch. Lastly, I understood what and how map generation works, and learned how tough it is to create a map generation system which resembles a hand-crafted map.

Overall working on this project was a valuable experience, that expanded my horizons drastically on more than one axis.

Table of figures

Figure 1, A scene from the game	5
Figure 2, A* search, blue tile represents explored nodes while the yellow line represents the path.....	8
Figure 3, A simple state machine.....	9
Figure 4, Client-Server model.....	10
Figure 5, Cellular automata generation.....	12
Figure 6, Server/Host component diagram	23
Figure 7, Client component diagram	27
Figure 8, Networked AI component diagram	28
Figure 9, Game architecture.....	29
Figure 10, Main menu design	30
Figure 11, demo scene UI design.....	30
Figure 12 environment architecture	31
Figure 13, playable character architecture	33
Figure 14, state machine	34
Figure 15, AI Components diagram.....	35
Figure 16, Network manager component diagram	36
Figure 17, map generation scene UI	37
Figure 18, Map generation Component diagram	38
Figure 19, Room template array	38
Figure 20, Spelunky type generation	39
Figure 21, Main menu.....	41
Figure 22, Main menu scene hierarchy	41
Figure 23, on click event setup.	41
Figure 24, Code fragment for loading a scene.	42
Figure 25, Demo scene hierarchy	42
Figure 26, Demo scene sample	42
Figure 27, Find distance function	43
Figure 28, network manager component	44
Figure 29, setting up the network manager component.....	44
Figure 30, player prefab components.....	45
Figure 31, Animator diagram for the player's prefab.....	45
Figure 32	46
Figure 33, Handle move function	46
Figure 34, Server command request	46
Figure 35, FSM bot components.....	47
Figure 36, the main state machine loop	47
Figure 37	48
Figure 38, A* search	49
Figure 39, AI traversing a path	49
Figure 40, map generation main UI.....	50
Figure 41, map generation scene hierarchy	50
Figure 42, Cellular generation scene	51
Figure 43, Cellular generation main loop	51
Figure 44	52

Figure 45, rogue like generation scene	53
--	----

References

- [1] - Plato.stanford.edu. 2001. Logic and Games (Stanford Encyclopedia of Philosophy). [online] Available at: <<https://plato.stanford.edu/entries/logic-games/>> [Accessed 20 May 2021].
- [2] - Bethke, E., 2003. Game development and production. Word Ware publishing.
- [3] - R. Ramadan and Y. Widyani, "Game development life cycle guidelines," 2013 International Conference on Advanced Computer Science and Information Systems (ICACSYS), 2013, pp. 95-100, doi: 10.1109/ICACSYS.2013.6761558.
- [4] - 27.group. n.d. The Multi- *Million Dollar 'Video Game' Industry* – 27 Advisory. [online] Available at: <<https://27.group/the-multi-million-dollar-video-game-industry/>> [Accessed 8 May 2021].
- [5] – Private conversations with game developers.
- [6] - Techcrunch.com. n.d. TechCrunch is now a part of Verizon Media. [online] Available at: <https://techcrunch.com/2019/10/17/how-unity-built-the-worlds-most-popular-game-engine/?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlMmNvbS8&guce_referrer_sig=AQAAAKFJc699sHC5QPUVV26Az2xlwATZXKmJAbBcmynWKGICWR-9OObU6rYcjMJFwvr_12ZfDG6MhUiiSuxOReS0CXeM9kWQiaPspmMm8aaKoCvFKoj4olhLAAZfQfJJkwlmIOGNz6eFX_zhbPOgyLxyxowSWAZwnG7CEGgvLbpJkxMB> [Accessed 21 May 2021].
- [7] - Technologies, U., n.d. Unity - Manual: Unity User Manual 2020.3 (LTS). [online] Docs.unity3d.com. Available at: <<https://docs.unity3d.com/Manual/index.html>> [Accessed 20 May 2021].
- [8] - Technologies, U., n.d. Unity case studies | Unity. [online] Unity. Available at: <<https://unity.com/case-study>> [Accessed 20 May 2021].
- [9] - IGN. n.d. History of the Unreal Engine - IGN. [online] Available at: <<https://www.ign.com/articles/2010/02/23/history-of-the-unreal-engine>> [Accessed 20 May 2021].
- [10] - Unreal Engine. n.d. Real-time for everyone - Unreal Engine. [online] Available at: <<https://www.unrealengine.com/en-US/industry/more-uses>> [Accessed 22 May 2021].
- [11] - Technologies, U., n.d. Unity case studies | Unity. [online] Unity. Available at: <<https://unity.com/case-study>> [Accessed 20 May 2021].
- Unreal Engine. n.d. Real-time for everyone - Unreal Engine. [online] Available at: <<https://www.unrealengine.com/en-US/industry/more-uses>> [Accessed 22 May 2021].
- [12] - Medium. n.d. CryEngine vs Unreal vs Unity: Select the Best Game Engine. [online] Available at: <<https://medium.com/@thinkwik/cryengine-vs-unreal-vs-unity-select-the-best-game-engine-eaca64c60e3e#:~:text=While%20matching%20the%20Unity%20vs,difference%20between%20Unreal%20and%20Unity.>>> [Accessed 22 May 2021].

- [13] - Technologies, U., 2021. Unity for 2D | Unity. [online] Unity. Available at: <<https://unity.com/solutions/2d>> [Accessed 21 May 2021].
- [14] - The Ultimate Resource for Video Game Design. n.d. Unity vs Unreal: Ultimate Game Engine Showdown. [online] Available at: <<https://www.gamedesigning.org/engines/unity-vs-unreal/#:~:text=The%20asset%20store%20in%20Unity,is%20a%20bit%20more%20polished.>>> [Accessed 22 May 2021].
- [15] - Nareyek, A., 2004. AI in Computer Games. Queue, 1(10), pp.58-65.
- [16] - Barnouti, N. , Al-Dabbagh, S. and Sahib Naser, M. (2016) Pathfinding in Strategy Games and Maze Solving Using A* Search Algorithm. Journal of Computer and Communications, 4, 15-25. doi: [10.4236/jcc.2016.411002](https://doi.org/10.4236/jcc.2016.411002).
- [17, 18, 19, 20] - Cochran, J.J., Cox, L.A., Jr., Keskinocak, P., Kharoufeh, J.P., Smith, J.C. and Gal, S. (2011). Search Games. In Wiley Encyclopedia of Operations Research and Management Science (eds J.J. Cochran, L.A. Cox, P. Keskinocak, J.P. Kharoufeh and J.C. Smith). <https://doi.org/10.1002/9780470400531.eorms0912>
- [21] - Wang, J. and Tepfenhart, W., n.d. Formal methods in computer science. 978-1-4987-7532-8.
- [22] - Encyclopedia Britannica. n.d. client-server architecture | Definition, Characteristics, & Advantages. [online] Available at: <<https://www.britannica.com/technology/client-server-architecture>> [Accessed 22 May 2021].
- [23] - Ghosh D., Rajan P., Pandey M. (2014) P2P-VoD Streaming:. In: Kumar Kundu M., Mohapatra D., Konar A., Chakraborty A. (eds) Advanced Computing, Networking and Informatics- Volume 2. Smart Innovation, Systems and Technologies, vol 28. Springer, Cham. https://doi.org/10.1007/978-3-319-07350-7_19
- [24] - Ccr.sigcomm.org. n.d. [online] Available at: <http://ccr.sigcomm.org/online/files/p2p_gaming.pdf> [Accessed 22 May 2021].
- [25] - Axne, J. and Axne, J., n.d. Peer-to-Peer vs. Client-Server Networks - Veterinary IT Support. [online] Veterinary IT Support. Available at: <<https://www.veterinaryitsupport.com/peer-to-peer-vs-client-server-networks/#:~:text=Peer%2Dto%2Dpeer%20networks%20connect,the%20data%20and%20manage%20resources.>>> [Accessed 24 May 2021].
- [26] - Gabrielgambetta.com. n.d. Client-Server Game Architecture - Gabriel Gambetta. [online] Available at: <<https://gabrielgambetta.com/client-server-game-architecture.html>> [Accessed 23 May 2021].
- [27, 28, 29] - Mirror-networking.com. n.d. Mirror Networking – Open Source Networking for Unity. [online] Available at: <<https://mirror-networking.com/>> [Accessed 24 May 2021].
- [30] - freeCodeCamp.org. n.d. How to code your own procedural dungeon map generator using the Random Walk Algorithm. [online] Available at: <<https://www.freecodecamp.org/news/how-to-make-your-own-procedural-dungeon-map-generator-using-the-random-walk-algorithm-e0085c8aa9a/>> [Accessed 23 May 2021].

- [31] - Tinysubversions.com. n.d. Spelunky Generator Lessons. [online] Available at: <<http://tinysubversions.com/spelunkyGen/>> [Accessed 24 May 2021].
- [32] - Sciencedirect.com. n.d. Cellular Automata - an overview | ScienceDirect Topics. [online] Available at: <<https://www.sciencedirect.com/topics/computer-science/cellular-automata>> [Accessed 23 May 2021].
- [34] - Edureka. 2021. A* Algorithm | Introduction to the A* Search Algorithm | Edureka. [online] Available at: <<https://www.edureka.co/blog/a-search-algorithm/>> [Accessed 24 May 2021].
- [35] - Spelunky Classic Mods. n.d. Spelunky Level Generation. [online] Available at: <<https://takenapeveryday.wordpress.com/2016/04/14/spelunky-level-generation/>> [Accessed 24 May 2021].
- [36] - Is 1080p Enough? - Benefits, D. and Streaming?, H., n.d. Is 60Hz Enough For Gaming? - Valorvortech. [online] Valorvortech. Available at: <<https://valorvortech.com/is-60hz-enough-for-gaming/#:~:text=60Hz%20is%20enough%20for%20gaming.%2060Hz%20is%20smooth,who%20play%20competitively%2C%20where%20every%20slight%20advantage%20counts.>>> [Accessed 25 May 2021].