Challenge Based Code Training Platform Using Unity Games

Abstract

Content

Introduction

1.Technical Background

1.1 Networking and Communication

1.1.1 Types of Computer Networks

1.1.1 Addresses

1.1.2 Client and Server

1.2.3 TCP & UDP Sockets

1.2.5

1.2 Unity game development

1.2.1 Why Unity?

1.2.2 Unity Editor Window

1.2.3 Unity Game Objects

1.2.4 Unity Build in Components

1.2.5

1.3

2.Software and Program Architecture

2.1 Software used

2.2 Architecture

2.3 Program Flow

2.4 Security

2.5 Python Server

2.6 Python Client

2.7 Setting up the Game

3.Results and discussions

Conclusion and future work

Bibliography

Introduction (needs improvement)

This thesis is about an approach to improve and challenge the coding skills by using a gaming environment. We will study the process of developing of such system and its feasibility.

In the recent years, the education improved a lot. The humanity switched the lever of learning from books in favor of learning from sources like: internet, audio books and online courses. Furthermore, people tend to opt more for the rewarded based systems, they are tired of learning something if they do not find it interesting enough or if they do not get an immediate satisfaction while doing it. There are plenty examples of students that simply rush their studies in order to get rid of them because the courses are not interesting enough to catch their attention. So are the programmers, if they do not like what they are coding, the will never bother coding it.

This thesis aim is to combine the utility with the pleasure, learning to code while playing games. This is achieved through a Unity application that allows the user to play a game by coding the strategy. The code written by the player gets executed and the response is converted into visual output.

The game we have created to demonstrate the capabilities of this system is a bomber man style game simulation. The purpose is to collect items and to eliminate other enemies, both things being done by selecting actions from a list of possible moves.

The technologies that have been used are the following: Python, for implementing the Client and Server side, Unity C# for implementing and designing the game and the environment, Sockets for establishing the connection between the server and the game. Moreover, the game also uses an external asset for the 3D game models.

The thesis is structured in ... chapters.

Chapter 1

Chapter 2

Chapter 3

Chapter 1

Technical Background

* 1. Networking and Data Communication

Two or more computing devices that are connected form a network. A network is used in order to share the resources and the information, as shown in Figure 1

Computer 2

Computer 1

Laptop

Wireless

Figure 1 (Network example)

The most basic computer network starts with at least two connected computers. This network can expand, other computer may join and add their resources to those being already shared.

The notion of “data communication” refers to the share of data between computers that can either be inside or outside of a specific network.

Data communications refers to the transmission of digital data between computers that can be inside or outside of a specific network. The connection between computers is realized using the communication cable or the Wi-Fi. The Internet is the best known computer network.

1.1.1 Types of Computer Networks

     In computer networks, the data is send in packets. A data packet does not always contain only the data to be sent, it may also have headers that carry certain types of metadata. For example, IP data packets have a header containing an IP address of origin and destination IP address.

The devices that transmit or receive data are called “nodes”. There are three main types of networks:

1. Local Area Network or LAN. This represents a small network. A computer network available only to the class of students can be called a LAN.

2. Wide Area Network or WAN. These networks cover a larger area. For the users inside LAN to be able to communicate with computer in other regions, a WAN is required. The most common WAN is the internet.

3. Metropolitan Area Network or MAN. This size of this particular network is somewhere between LAN and WAN.

1.1.1 Addresses

A network address is used to uniquely distinguish a network node or device over a network. It is a combination of numbers or letters that is assigned to any device that request access to or is part of any network.

A network address is very important because it works like a key that facilitates the identification and the reach of a network node or device over a network.

Each computer inside a network uses a network address to identify, locate and address other computers.

1.1.2 Client and Server

Client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs, which share their resources with clients. A client does not share any of its resources, but it requests content or service from a server. Clients, therefore, initiate communication sessions with servers, which await incoming requests. Examples of computer applications that use the client-server model are Email, network printing, and the World Wide Web.

The client-server characteristic describes the relationship of cooperating programs in an application. The server component provides a function or service to one or many clients, which initiate requests for such services. Servers are classified by the services they provide. For example, a web server serves web pages and a file server serves computer files. A shared resource may be any of the server computer's software and electronic components, from programs and data to processors and storage devices. The sharing of resources of a server constitutes a service.

Web Server Web Browser Client

Page request

www.google.com/images

Data

Returned page

Figure 2 (Server request in a web browser)

Whether a computer is a client, a server, or both, is determined by the nature of the application that requires the service functions. For example, a single computer can run a web servers and file server software at the same time to serve different data to clients making different kinds of requests. Client software can also communicate with server software within the same computer.

In general, a service is an abstraction of computer resources and a client does not have to be concerned with how the server performs while fulfilling the request and delivering the response. The client only has to understand the response based on the well-known application protocol, i.e. the content and the formatting of the data for the requested service. Clients and servers exchange messages in a request–response messaging pattern. The client sends a request, and the server returns a response.

A server may receive requests from many distinct clients in a short period of time. A computer can only perform a limited number of tasks at any moment, and relies on a scheduling system to prioritize incoming requests from clients to accommodate them.

1.2 Unity Game Development

1.2.1 Why Unity ?

What is Unity ?



Figure (Unity logo)

Unity3D is a very powerful development framework for building the exciting 2D/3D game. The games developed on this game engine are developed once with a single code but the games can be easily deployed on different platforms.

Unity3D is a game engine that offers revolutionary solutions with features of interactive media installation as well as a stunning architectural visualization that is suitable for cross platform environment. Unity 3D game development engine offers extensive ecosystem for building and deploying.

It ensures rare gaming experience with the enthralling 3D game flow along with a wide array of game authoring tools and intuitive resources. The popularity of the game engine has encompassed for its user-friendly features and other simple and pragmatic reasons.

Unity3D is reaching the new heights in developing games and it has been ruling the world of game development. Unity3D has become the first choice for the independent developers. Game development studios even have highly admired utilization of Unity3D for its versatility, flexibility, and ease-to-access feature.

There are really very few options when it comes to Game development. The three main choices if you want to build games are Unreal, Unity or GameMaker.

Unity gives you a very raw product out of the box, but is highly flexible, well-documented, and highly extensible to build pretty much any genre of game you can think of.

There are plenty of highly successful games such as Escape from Tarkov (FPS), Monument Valley (Puzzler), and This War of Mine (Strategy / Survival) all built in Unity.

In reality the engine you build your first game on is probably not critical, so my advice is just to pick one and go with it.

Benefits:

Unity3D is easy to operate.

The structure of animation and rendering is highly flexible that turns several characters in the game lively.

Unity3D renders natural and fluid movement.

It supports cross platforms as well as various devices.

JavaScript and C are the most popular coding languages which can be used with Unity for scripting.

Unity3D is the preferred game engine for developing multiplayer games.

The user-friendly interface of Unity3D makes it easy-to-handle.

Unity3D is compatible with Windows as well as Mac OS X which is one of the reasons why the developers go for Unity3D.

With Unity3D, it is easier to handle and execute different elements by using the technique of “Drag, Drop, and Animate”.

With Unity3D, games can be developed for around twelve different platforms.

1.2.2 Unity Editor Window

The editor window is split up into a couple of sections. We will cover this very briefly as we will refer to it constantly throughout the article. If your familiar with this already just skip past!

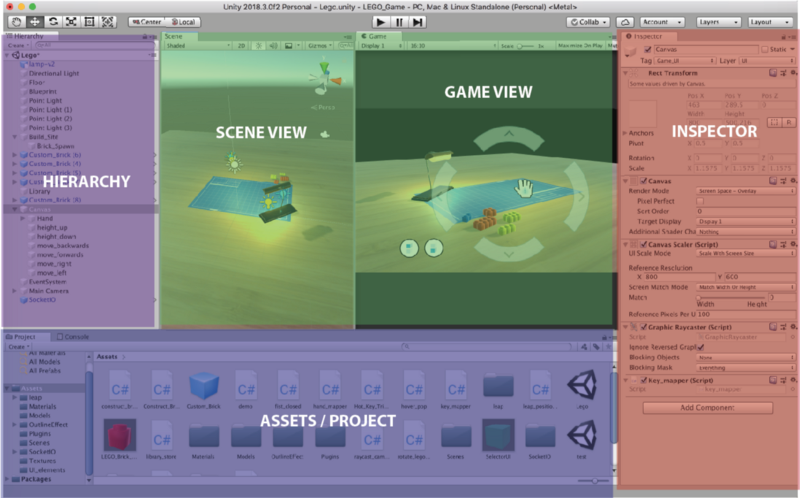


Figure (Unity Editor Window)

Scene View: Allows placement and movement of GameObjects in the Scene

Game View: Previews how the player will see the scene from the camera

Inspector: Provide details on the selected GameObject in the scene.

Assets / Project: All prefabs, textures, models, scripts etc are stored here

Hierarchy: Enables nesting and structuring of GameObjects within the scene

1.2.3 Unity Game Objects

GameObjects are the core building block of everything in the Unity games engine. The name almost gives it away:

Anything you place within a scene in Unity must be wrapped in a ‘game object.’

If you’ve got a web design background, you can think of GameObjects as being a lot like <div> elements! Extremely boring containers, but are highly extensible to create complex functionality or visuals. Below is a list of “primitive” GameObjects that Unity offers.

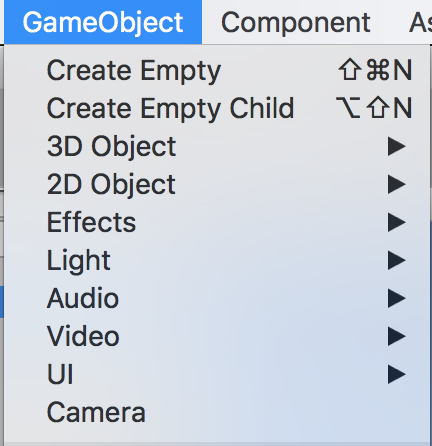


Figure (Unity Standard GameObjects)

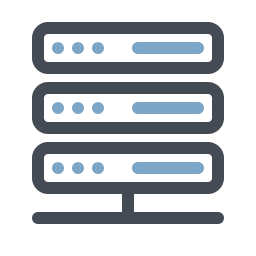
2. Software and Implementation

2.1 Software used

This thesis was made using the following technologies: Python 2.7, for writing the client and the server, Unity C#, to develop and render the game, TCP Sockets for sending and receiving information.

2.2 Architecture

The architecture of the system is the following:



Python Client 1

Unity Client 1

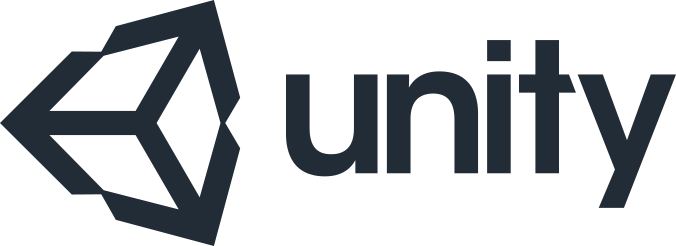
Python Client 2

Unity Client 2

Python Client 3

Unity Client 3





Each user will have a corresponding Unity Client which represents the visual representation of the data, this, being the game and a Python Client which is the code part that will be executed in order to provide data for the game, this may be seen also as the game strategy.

We are going create a python server that will manage the new connections and using TCP communication protocol, the server will be able to transfer data fast and safe between Unity game and Python Client.

The server acts like a lobby and in order to connect, a user has to open a Unity Game instance and provide the server address and port. If all Unity Clients that are connected are ready to play, the server will automatically switch to the next state where it waits for each user to connect with the Python Client. At this point, everything is ready and the game can start. The server will continuously send and receive data over the TCP sockets until the game is over.

2.3 Program Flow

The user starts by executing the Unity Client. In the Unity Client the user chooses one either to play single player (against a hardcoded AI) or multiplayer (against real players in LAN or Internet).

If single player is chosen, Unity Client will create a local unity server which will handle the input from the Python Client. The game on the single player is customizable, meaning that the user is free to choose the number of AIs and the map generation parameters.

If the multiplayer is chosen, Unity Client will create a new process of Python Server which will manage the clients. Now, the Unity Clients will be able to connect to the server using the IP and PORT of the server. After the server is filled up with Unity Clients or all the Unity Clients press the ready button, each Unity Client will get a token which is going to be used by the Python Client to connect to the Python Server. The server will then wait for all the Python Clients then start generating a random map on which users are going to play. The Python Server will then ensure the flow of the data unity client -> python server -> python client and backwards, each round. In case of any problem with the player or the data received, the player will be disconnected from the game and the game will continue.

Render

Render

Render

Render

Process Data

Process Data

Process Data

Process Data

Unity Client 1

Unity Client 2

Unity Client 3

Unity Client 4

Handle New Connections

Thread

Handle Disconnected Players

Server

Process Data

Manage Game

Python Client 4

Python Client 3

Python Client 2

Python Client 1

Process Input

Process Input

Process Input

Process Input

2.4 Security

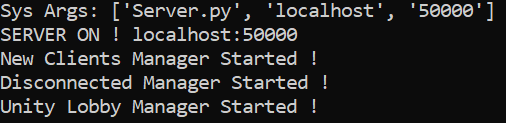
Taking into consideration that there are three modules that share data and work together, there might be a concern about the security of the entire system but, that is not the case.

All three components are designed in such a way that they have no weak points or any exposed data. No component relies on any external files, the unity client and the server have executables, they also have an internal token verification that is protected hence there is no source code, only the executable. The only source code a user is able to see is the connection part inside the Python Client module. This part also cannot be cracked, due to the fact that it requires a specific unique token to send data to the server, token that is verified on the server, not locally. At any small modification of the code, the client will not be recognized by the server and it will be rejected immediately.

Python server

The multi-player sessions can be played on LAN (Local Area Network) or on WAN (Wide Area Network) because there are no limitations. A server may be host by any user and there might be a number of maximum 4 players. Considering this, the only thing that may affect the server from working properly is the internet speed of the host.

The Python Server is the main brain of the game. It creates the sockets available for the players in order to allow them to connect to the server, handles the possible bad scenarios, ensures the normal flow of the data through the sockets, checks if the Unity Clients have all connected and are ready to play, works as a lobby manager, checks if all the sockets are still connected, otherwise safely removes them, logs user friendly messages, generates the game map, combines all the input received from the Python Clients in order to send it to each Unity Client, splits the received from the Unity Clients to send it to the Python Clients in order for the code to decide what to do next. Most of the server operations are performed on threads and are thread safe.



**New Clients Manager**

This part of the server is running on a separate thread. The server has three main states: unity clients connecting, python clients connecting, game running. Players are allowed to connect to the server only on the first two states. For a unity client to connect, it is enough to just enter the server details and hit the connect button, the server will do the rest. In case of a python client, the things change a little bit because, this type of client requires a token generated by the server when connecting with unity client that is unique and needs to be set in the python client connection details area. If the server is in the third state, meaning that the game is running, the player will be automatically disconnected by the server, receiving a corresponding message announcing them that they cannot join a game that is currently running.



**Client Handler**

This is the place where each client gets in order to be verified by the server. A client handler waits a specific amount of time for the client to send the verification token. The server receives the specific token, verifies it and if it corresponds, the client is added to the players list and is officially connected to the server. If the provided token is wrong or it is not provided in due time, the server closes the connection with that specific player.

**Client Class**

The client class is responsible for packing up all the data about a specific user. This class contains a reference to a unity client socket, a python client socket, a user identification token and some information related to the player connection status and states. The class also contains two methods that are running on separate threads, methods used to read data and to write data.

**Game Manager**

This part of the server is running on a separate thread. When the server reaches this point it means that all the necessary unity and python clients are connected to the server and they are ready to play the game. The server creates the initial game data, meaning that it generates the environment data and sends it to each unity client. After this, the game manager waits a period of time and iterates through the list of players to check the response coming from each unity client. If there is no response from a specific player, the server closes the connection with that player. After receiving all the unity clients data, the server forwards it to each corresponding python client to be processed. After the data is processed, each python client sends it back to the server. The server packs all actions received from the python clients and sends it to each unity client to be processed and rendered. This process repeats until the game is over. In case of any timeout from a client, the unity client and the python client are removed from the server.

**Disconnect Manager**

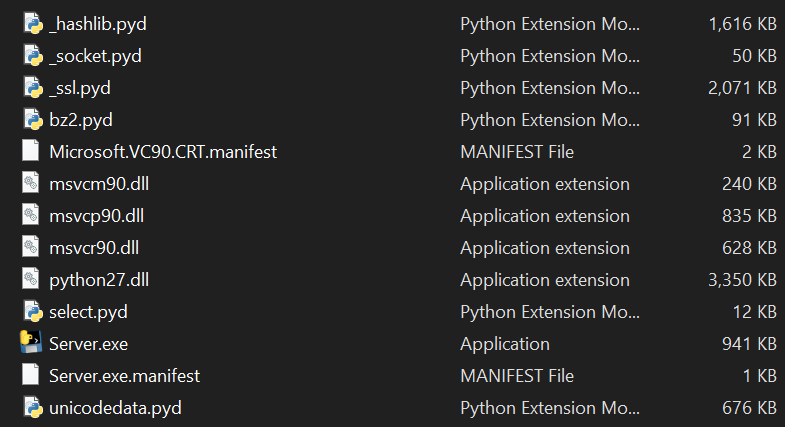
As the name suggests, this manager handles the disconnected players. The code is running on a separate thread and it periodically iterates the list of players and checks if a player has disconnected status. If so, it safely closes the connection between the server and that player, removing the user from the list of the players that are on the server.

**Game Class**

This class is like a module, representing all the information about the game. It is specific for the current game, having unique values and functions. This class includes server settings like: the maximum number of players, read buffer size, waiting times for different kind of game states and specific game parameters. Besides this, the game class also packs up the initial data of the game, meaning that it generates the map and formats the data about each player, in order to create the desired playable game environment.

**Server Build**

Using pyinstaller we made a server build. This bundles a Python application and all its dependencies into a single package The reason for this is to raise the server security by preventing any code modification.

****

Python client

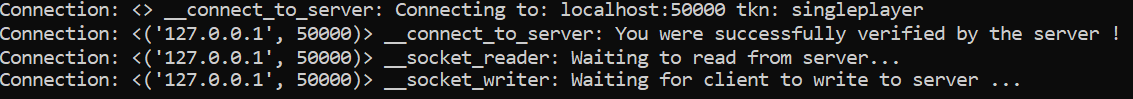
**Connecting to the server**

To communicate with the server, the Python Client uses an TCP socket. To establish a connection, the user is required to provide the server address, the server port and a client token. The last one represents a generated string that is used as an identification code for security reasons when connecting to the server. It is formed of letters and numbers and it is randomly generated. If the user opts for a single player game session, the value of this variable will be “singleplayer”.

A TCP socket is created and the connection is established using the previous server data. If the connection to the specified server address and port fails, the user will get a message.

**Client validation by the server**

After a successful connection to the server, the client token needs to be verified. As mentioned before, this needs to be done for security purposes. Right after establishing a connection with the server, the client automatically sends the token to be verified. The token reaches the server side and after it gets verified, the player gets a feedback with the server response. If everything went well, the python client remains on the server, otherwise, it gets disconnected.

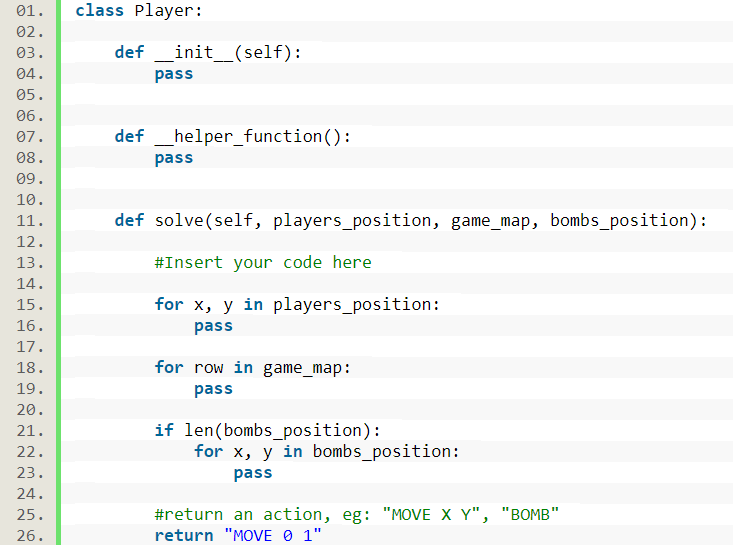


**Reading and writing to the server**

Because of the fact that the game is played using a turn based system, data from the server gets on the client side, it gets processed and then it is sent back to the server. The reading and the writing are separated into two threads. The reading thread waits for the server to send information to the client then it reads and stores this data, on the other side, the writing thread waits for the client to send any data to the server.

**Player Class**

This class represents the editable area of the Python Client. Any user is allowed to modify this class, as it represents the strategy of the game. The most important part of this class is the solve function. This function contains a list of parameters representing the information about the game at a specific moment. The user has to use this data and to code a strategy in order to get points and win the game. The function is characterized with a return type of string which represents an action that the user would like to perform in the next turn of the game, in order to be closer to win the game. The user may select to perform only one action each round and have to be careful because if the action is not possible it will not be executed by the game.



Creating the game

For demonstration purposes we tried to choose a simple game that is already turn based or can be easily played on turns. We came to the conclusion that a Bomber Man like game would be great for the representation.

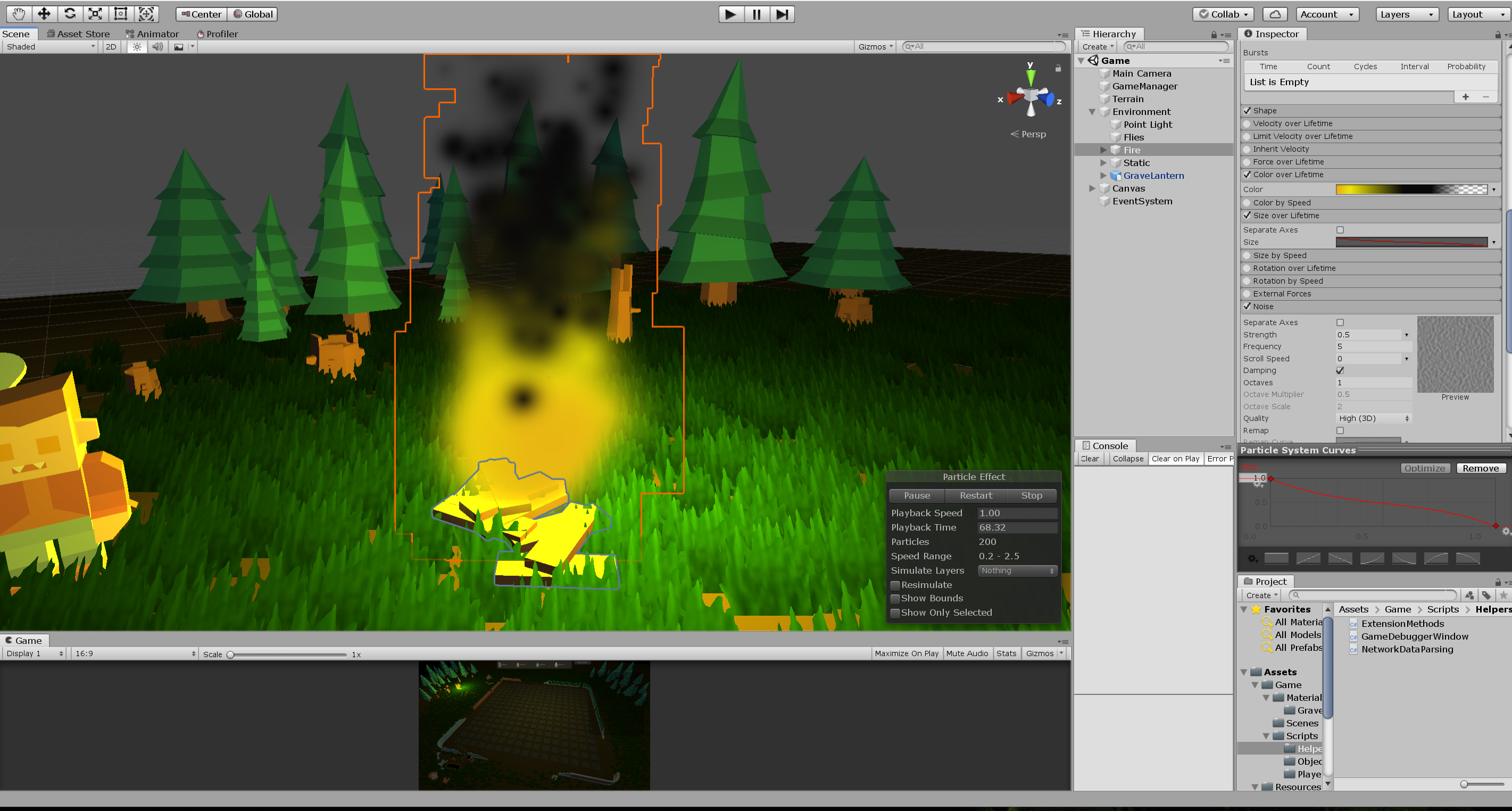
To make the game more interesting, we also added some collectable items that increase the player score. The goal of the game is to collect the most items and to eliminate all other opponents.

The game is called Graveyard Friends because of the 3D asset it features.



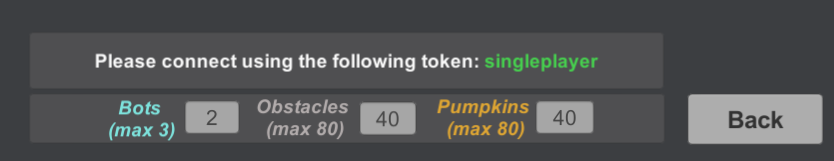
**Level design**

The level design was made using a 3D asset for the scene environment, unity terrain for generating the grass and unity particles for some effects. The asset is from Kenney.nl site and is called “Graveyard Kit”, being a free to use pack with 3d models.



**Single Player Mode**

Because of the fact that this is a challenge based coding platform, there is no need to always have an internet connection hence the user may opt for single player sessions. The benefits of this mode are the following: there is no wait time for the other players, the strategy is easier to test because the user is allowed to change the map parameters and last but not the least, there are AIs to play with.



**AI Player**

AI (Artificial Intelligence) Player represents a non-human type of player that already has a predefined strategy. The AI has a standard strategy, just for demonstration purposes.

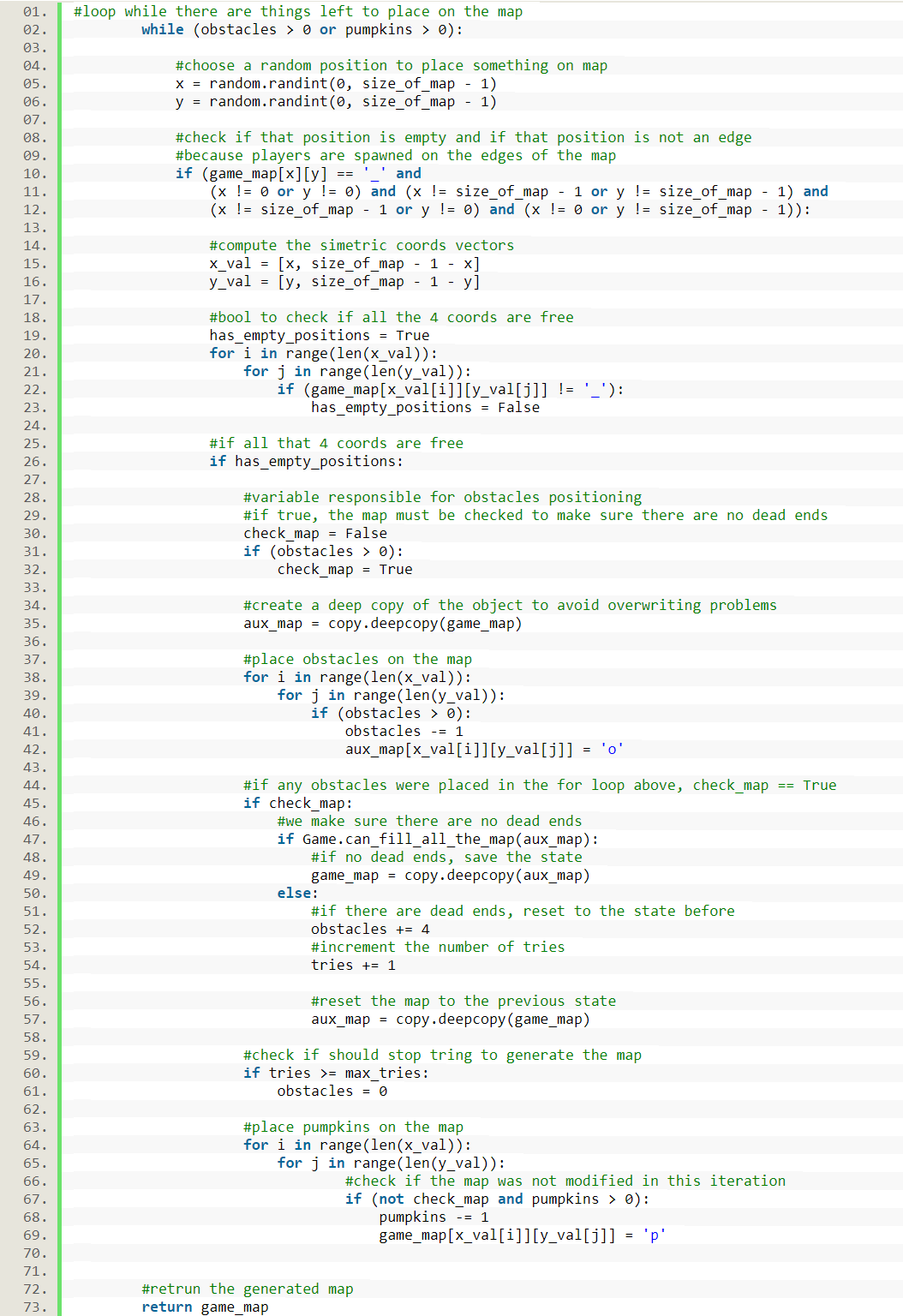
**Map generation**

The map generation is commonly used when single player because the user is allowed to choose the way the map will look like. Each map contains players, obstacles and collectables. Based on some parameters, the map is generated in the following way:

The obstacles are the first to be positioned because the difference between collectables and obstacles is the fact that a player can get through a collectable item but, cannot go through an obstacle. For this reason, it is greatly recommended to start positioning an obstacle that will be surrounded by collectable items than the other way, because the possibility of not placing all the obstacles and creating dead ends is less. When generating the map, because of the fact that the map is a 16x16 square, an object has to be mirrored in all 4 sides of an xoy coordinate system, considering the center of the map the origin of the system because all the players must have the same difficulty. The players are always placed in the corner of the map.

Besides this, every time an object of type obstacle is placed, a function that uses an iterative form of Depth First Search algorithm check for dead ends. A dead end is formed if an obstacle is placed in such a way that a player cannot move along all the free map spaces.

**Multi Player Mode**



The multi-player mode is designed for challenging the coding skills with other players around the world. This type of game can be played on Local Area Network (LAN) or on Wide Area Network (WAN).

**Connecting to Server**

To connect to a server, a user has to switch to the Multiplayer scene. There, the user is required to type in the server address followed by the server port and hit the connect button. If the connection to the server succeeds, a button for “ready” response will pop up followed by a corresponding message, otherwise, a connection status message announcing the player that the connection to the server failed is displayed.

**Network Manager**

The Network Manager is layer, basically a class that either creates a Unity server that will be used for LAN single player or establishes a connection with the python server in case of a multi-player session. It is considered a layer due to the fact that it has “DontDestroyOnLoad()” property enabled. This thing ensures that when switching from a scene to the other, this object will not be destroyed.

The reading and the writing operations are asynchronous due to the fact that we need the rest of the game to be accessible while sending and receiving data. Each received command is sent to a handler that either modifies the connection state or it modifies the data inside the game manager, this being the game.

An interesting situation occurs when the first game data is sent to the game manager. The current scene does not have a game manager, so we need to load the main game scene in order to access that manager. The process of loading the game scene must be done asynchronous due to the fact that the program must respond meanwhile to other requests. Furthermore, we have to ensure that the scene is loaded before trying to access the game manager, otherwise this will lead to errors.

If any connection is established, Unity will periodically ping the other end of the connection to see if the it is still available, if no response in due time, Unity will close the existing connection, redirecting the user to the main menu.

**Single Player**

It might be weird that a single player session requires a network manager but, here is the reason for that: as mentioned before, to ensure a connectivity between a Unity Client and a Python Client, we need TCP sockets that require a server that handles the data.

When going for a single player session, the Unity automatically creates a localhost server and waits for a player to connect in order to start the game.

**Multi-Player**

If the player chooses a multi-player session, after filling up the corresponding fields for the server address and server port, the server will check if a connection might be established on the corresponding information, if so, the network manager will pop up a “ready” button for the player. Pressing that button will mark that the current player is ready to enter the game. The system was designed this way due to the fact that a lobby cannot rely only on a countdown timer to start the game because players connect and reconnect and also, we cannot start the game only by considering the condition that the lobby has the minimum required players, there has to be the possibility to send any number of players to the game whenever they want, basically, whenever they are ready.

**Data Parsing and Initial Game Data**

Right after the moment when all the players are connected to the server and the game is ready to start, there is no playable map. The idea is that, for security reasons the map has to be generated on the server and sent to each Unity Client. This is called the initial game data, the block of data that is responsible for creating the same environment for each user, containing the info about the generated map and the players that will play in that session. Each packet of information received from the server needs to be decoded and interpreted.

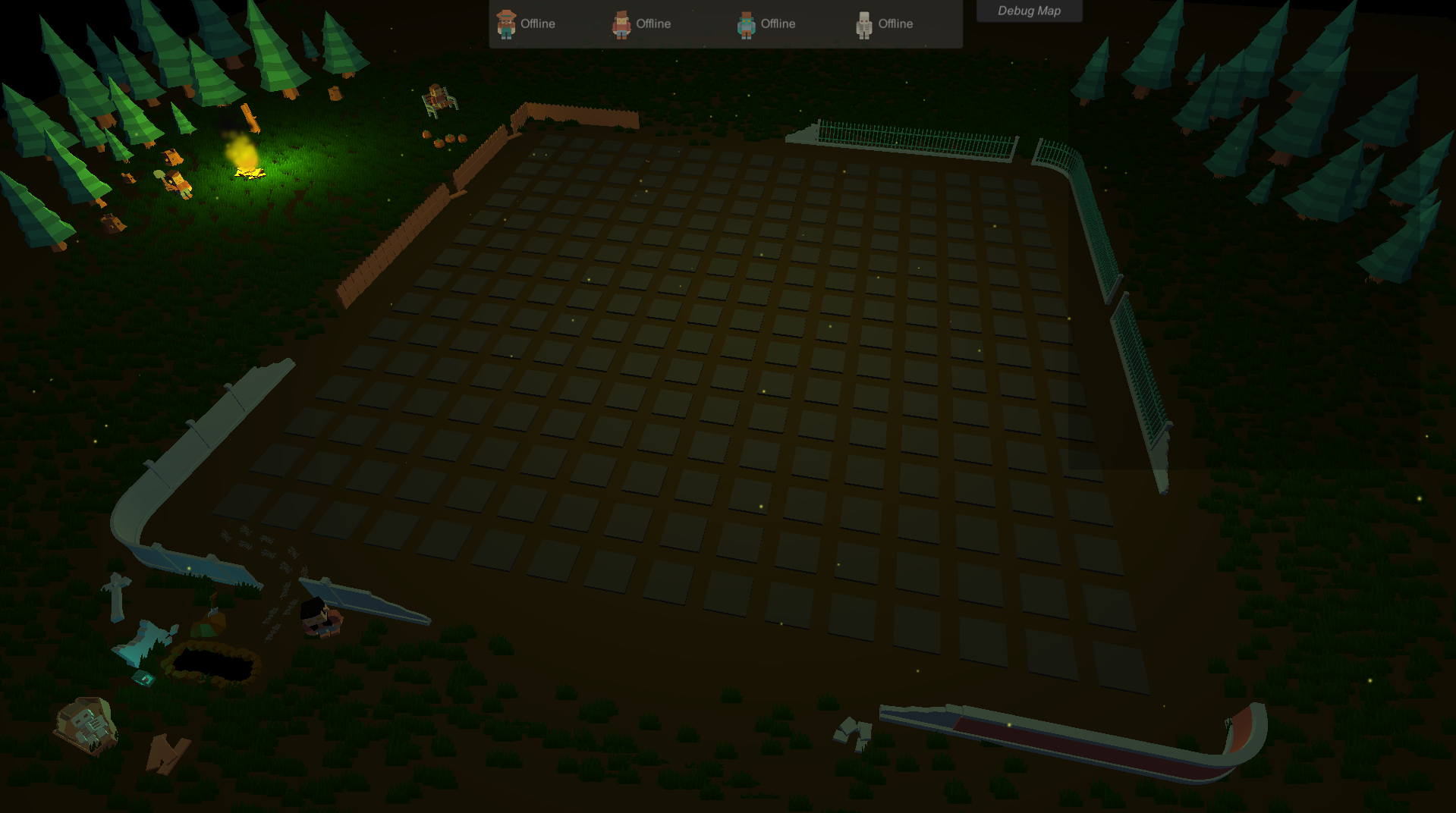
Besides the initial game data that is sent only once, a packet of data contains commands that are separated by new line. A command represents a mapping of form: key: <token> and the value: <action>, which will further be interpreted by the Game Manager of the Unity Client as: The player with <token> is willing to perform <action>.

**Game Manager**

The Game Manager is the brain of the game. All the information received from the server gets processed inside the game manager which keeps track of any object of the scene. This manager is specific designed for the current game.

**Prepare the Game**

This is the first part of the Game Manager that will be executed in order to run a game session. For both single and multi-player, there are some things that need to be prepared before a game can start. Considering the single player, the game manager will generate the corresponding map and will fill the player slots with AIs. If the game is a multi-player one, the game manager will fill the map considering the data received from the server and will position the players accord

****



**Managing the Data**

Each map contains players, obstacles, collectable items and bombs. Considering all of these, a representation of the map might be represented as char matrix is not enough. There are some edge cases: 2 or more players might sit in the same place, a bomb might be placed right under the player and this representation of the map is not enough to recognize these cases. The solution would be to use lists for objects that may overlap.

**Process One Round**

On each turn, the Unity Client gets data from the server about each player. This data represents commands that are going to be executed. To process one round, we need to handle different kind of things in a specific order. At the beginning, the Game Manager will check if there are any bombs active on the map and will update their timer or detonate them because, otherwise, a player may exit the bomb area and this would have no effect, even though it should, leading to a bug. Next, each players command is sent to a handler that will execute it. The handler checks if the player is able to perform that specific action and if so, it gets executed, otherwise, nothing happens. The available actions are: MOVE X Y, BOMB.

At the end of the round, the players that are sitting on a position containing a collectable item, will receive that item.

**Get Session Data**

Each round the server sends data to the Unity Client that is processed locally by each game. Each Unity session needs to send updated game info back to the server. In order to do this, all the data needs to be packed in a specific manner. We will pack up player data. If the current player is not able to play anymore due to the game rules, Unity will send back the following data “SPECTATE” announcing the server and the python client that the player is not able to perform any further actions. If the player is able to play, we first pack up all the player positions starting with the current player then pack up all the map data, followed by the position of the active bombs on the map, if any.

**Map Debugger**

Map debugger is a tool that helps a user to see what is going on the map. This is a separate game panel that might be turned on and off and it consist of a matrix of chars that represents the map. This is very helpful for the users when debugging the code and also a good way to immediately see what is going on the map.



Switching to a new game

Further ideas

Conclusion

Bibliography

**Bibliographic resources:**

**Articles:**

Socket Programming in Python (Guide) - Nathan Jennings, Source : <https://realpython.com/python-sockets/>

The Ultimate Beginners Guide To Game Development In Unity -

<https://www.freecodecamp.org/news/the-ultimate-beginners-guide-to-game-development-in-unity-f9bfe972c2b5/>

<https://medium.com/@vivekshah.P/reasons-why-unity3d-is-so-much-popular-in-the-gaming-industry-705898a2a04>

<https://en.wikipedia.org/wiki/Client–server_model>

<https://www.techopedia.com/definition/20969/network-address>

# COMPUTER NETWORKS AND DATA COMMUNICATION

<https://peda.net/kenya/ass/subjects2/computer-studies/form-4/itcn>

<https://www.techopedia.com/definition/6751/data-packet>