BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION COMPUTER SCIENCE IN ENGLISH

DIPLOMA THESIS

CHALLENGE BASED CODE TRAINING PLATFORM USING UNITY GAMES

Scientific supervisor: Lect. Ph.D. Arthur Molnar

Author: Sabau Andrei-Mircea

Abstract

Contents

List of figures

Introduction

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

In this chapter we are taking a look at the concepts and technologies we will be referencing throughout the paper. Section 1.1 gives a detailed overview of what networking means and the how the data is transferred between machines that share the same network. The following part, Section 1.2 will be an introduction to the game development industry using Unity3D. This section points out the reasons why this framework became so popular and which are its strong points, followed by a short detailing of Unity3D user interface and functions.

1.1 Networking and Data Communication

A computer network consists of a collection of computers, printers and other equipment that is connected together so that they can communicate with each other [1]. A network is used in order to share the resources and the information. [1.1]

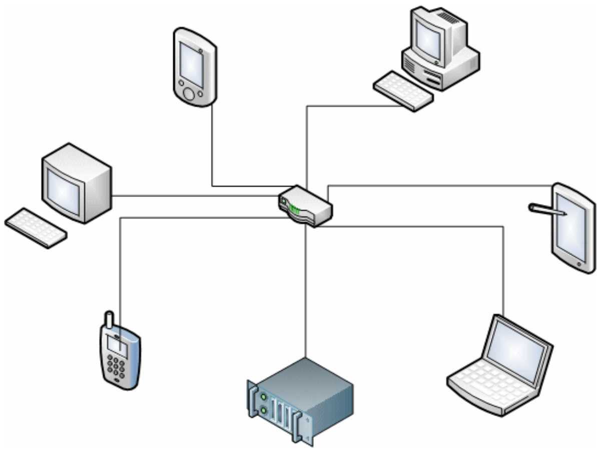


Figure 1.1 Network example.

This network can expand, other computers 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. The connection between computers is realized using the communication cable or the Wi-Fi. The Internet is the best known computer network. 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 [2]. Here are some advantages and disadvantages of using computer networks [3]:

Advantages

- Easily share of data between computers inside the same network.
- The facilities of the network can be accessed only by authorized users.

Disadvantages

- Data and information may be stolen by computer hackers if the security of network is not reliable. Once entered in a network, a hacker can easily get to each computer in the network.
- If any computer in a network gets affected by computer virus, there is high chance of spreading computer viruses on the other computer creating a chain reaction, this leading to a corrupted network.
- Computers depend on the server for resources and these may be limited.

1.1.1 Types of Computer Networks

The devices that transmit or receive data are called “nodes”. There are three main types of networks [4] [1.1.1]:

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.

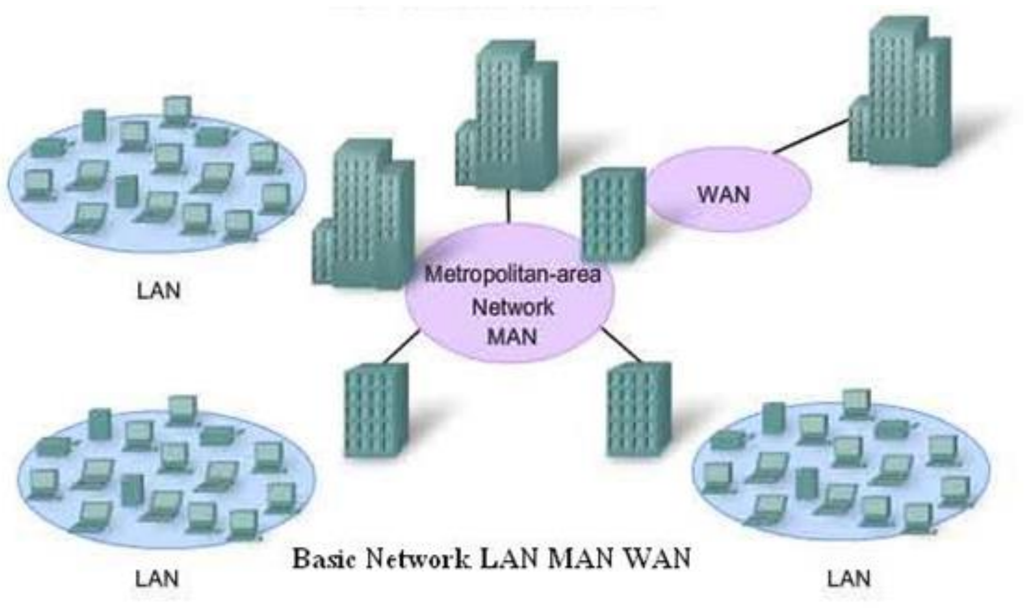


Figure 1.1.1 LAN MAN and WAN

1.1.2 Addresses

A network address is used to uniquely distinguish a network node or device over a specific 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 to other computers [5].

1.1.3 Client and Server (refactoring)

The client-server relationship occurs between two computer programs in which the program labeled as client, makes a service request from another program, labeled as server, which fulfills the request.

The client server idea may be used within the same computer but, it is more effective when used within a network. In a network, the client-server model provides a convenient way to interconnect programs that are distributed efficiently across different locations. The client-server model has become one of the central ideas of network computing.
Web browser is a client program that requests services from a Web server in another computer somewhere on the Internet [1.1.3]. Similarly, the computer with TCP/IP installed allows users to make client requests for files from File Transfer Protocol (FTP) servers in other computers on the Internet.
Other program relationship models included master/slave, with one program being in charge of all other programs, and peer-to-peer, with either of two programs able to initiate a transaction [9].

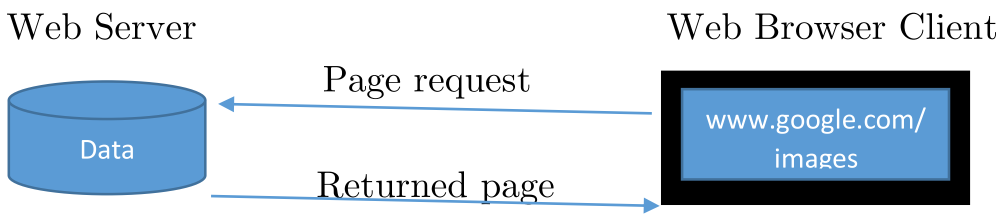


Figure 1.1.3 Server request in a web browser

Figure 1.1.2 Client and Server

Client-Server Applications

The client-server model distinguishes between applications as well as devices. Network clients make requests to a server by sending messages, and servers respond to their clients by acting on each request and returning results. One server generally supports numerous clients, and multiple servers can be networked together in a pool to handle the increased processing load as the number of clients grows.

A client computer and a server computer are usually two separate devices, each customized for their designed purpose. For example, a Web client works best with a large screen display, while a Web server does not need any display at all and can be located anywhere in the world. However, in some cases a given device can function both as a client and a server for the same application. Likewise, a device that is a server for one application can simultaneously act as a client to other servers, for different applications.

[Some of the most popular applications on the Internet follow the client-server model including email, FTP and Web services. Each of these clients features a user interface (either graphic- or text-based) and a client application that allows the user to connect to servers. In the case of email and FTP, users enter a computer name {or sometimes an IP address} into the interface to set up connections to the server.
Local Client-Server Networks
Many home networks utilize client-server systems without even realizing it. Broadband routers, for example, contain DHCP servers that provide IP addresses to the home computers (DHCP clients). Other types of network servers found in home include *print servers* and *backup servers*.

1.1.4 Sockets

Sockets ensure the communication between two different processes on the same or different machines. It represents a way to talk to other computers using standard Unix file descriptors. In Unix, every I/O action is done by writing or reading a file descriptor. A file descriptor is just an integer associated with an open file and it can be a network connection, a text file, a terminal, or something else [7].

The most common type of socket applications are client-server applications, one side acts as the server and waits for connections from clients.

Transmission Control Protocol (TCP)

Since it is based on connections between clients and servers, TCP provides a connection oriented service. This kind of transmission protocol provides reliability. When a TCP client send data to the server, it requires a feedback in return. If that feedback is not received, TCP automatically retransmit the data and waits for a longer period of time [8] [1.1.4.1].

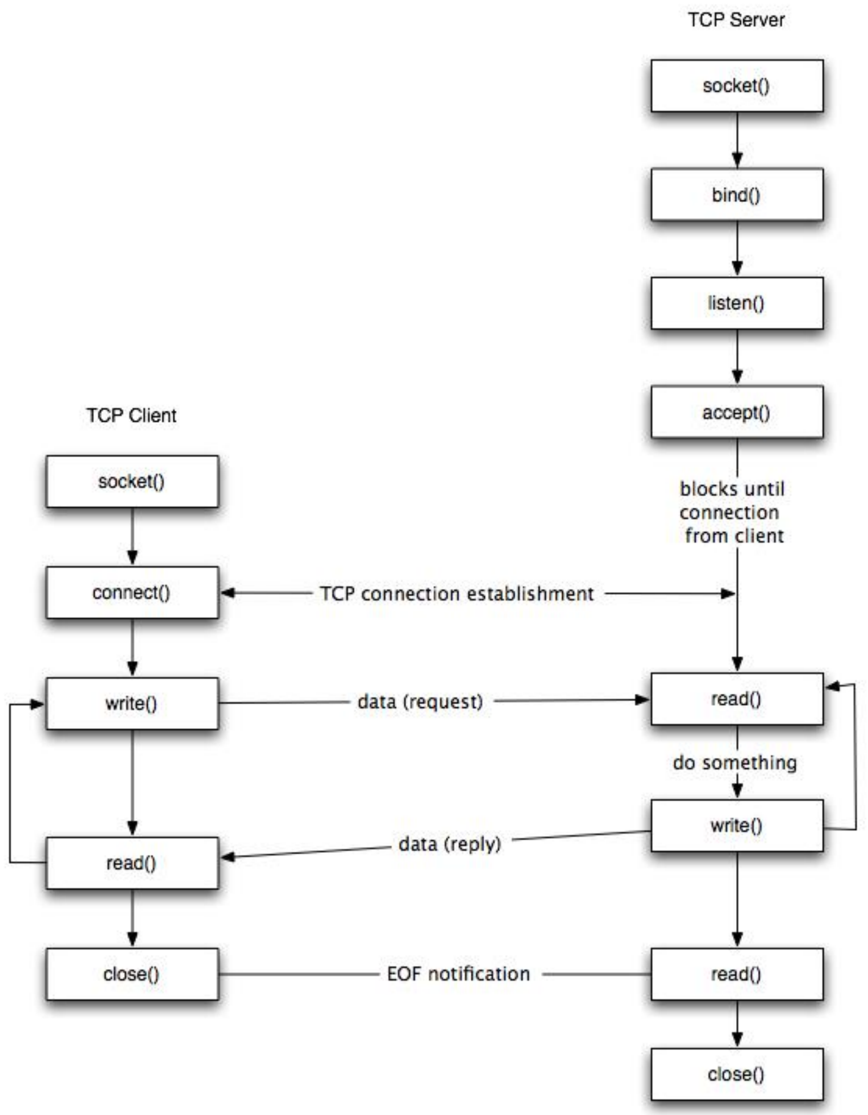


Figure 1.1.4.1 TCP Scheme

User Datagram Protocol (UDP)

UDP is a simple transport-layer protocol. The application writes a message to a UDP socket, which is then encapsulated in a UDP datagram, which is then encapsulated in an IP datagram, which is sent to the destination. The UDP cannot guarantee that the message will reach the destination, that the order of the datagrams will be preserved across the network or that datagrams arrive only once. The problem of UDP is its lack of reliability: if a datagram reaches its final destination but the checksum detects an error, or if the datagram is dropped in the network, it is not automatically retransmitted. Each UDP datagram is characterized by a length. The length of a datagram is passed to the receiving application along with the data.

No connection is established between the client and the server and, for this reason, we say that UDP provides a connection-less service [8].

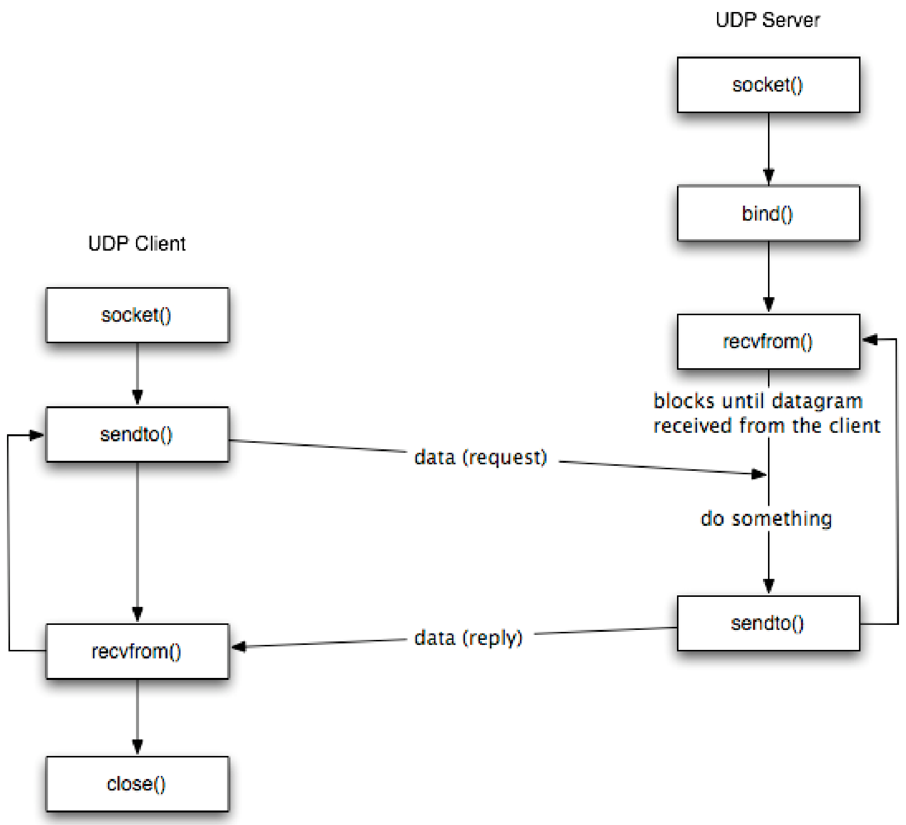


Figure 1.1.4.1 UDP Scheme

1.2 Unity Game Development

1.2.1 What is Unity?

Figure 1.2.1 Unity logo.

Unity 3D [1.2.1] is a game development framework for building 2D and 3D games. The applications are developed once, then, the user have the possibility to deploy its project on different platforms. This framework offers a lot of solutions, features and a stunning architectural visualization. It is an easy to use platform that offers a lot of support for building and deploying. A lot of games have been developed using Unity3D, such as Escape from Tarkov (FPS), Monument Valley (Puzzler), and This War of Mine (Strategy / Survival), the gaming experience being amazing. This engine gathers more and more audience due to its versatility and flexibility, independent developers and gaming studious using it on a daily basis [6].

Why Unity?

Benefits: refactoring

- Easy to operate with.
- 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
- The drag and drop interface saves a lot of time and it is easy to use.
- 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 lot of mini windows [1.2.2].

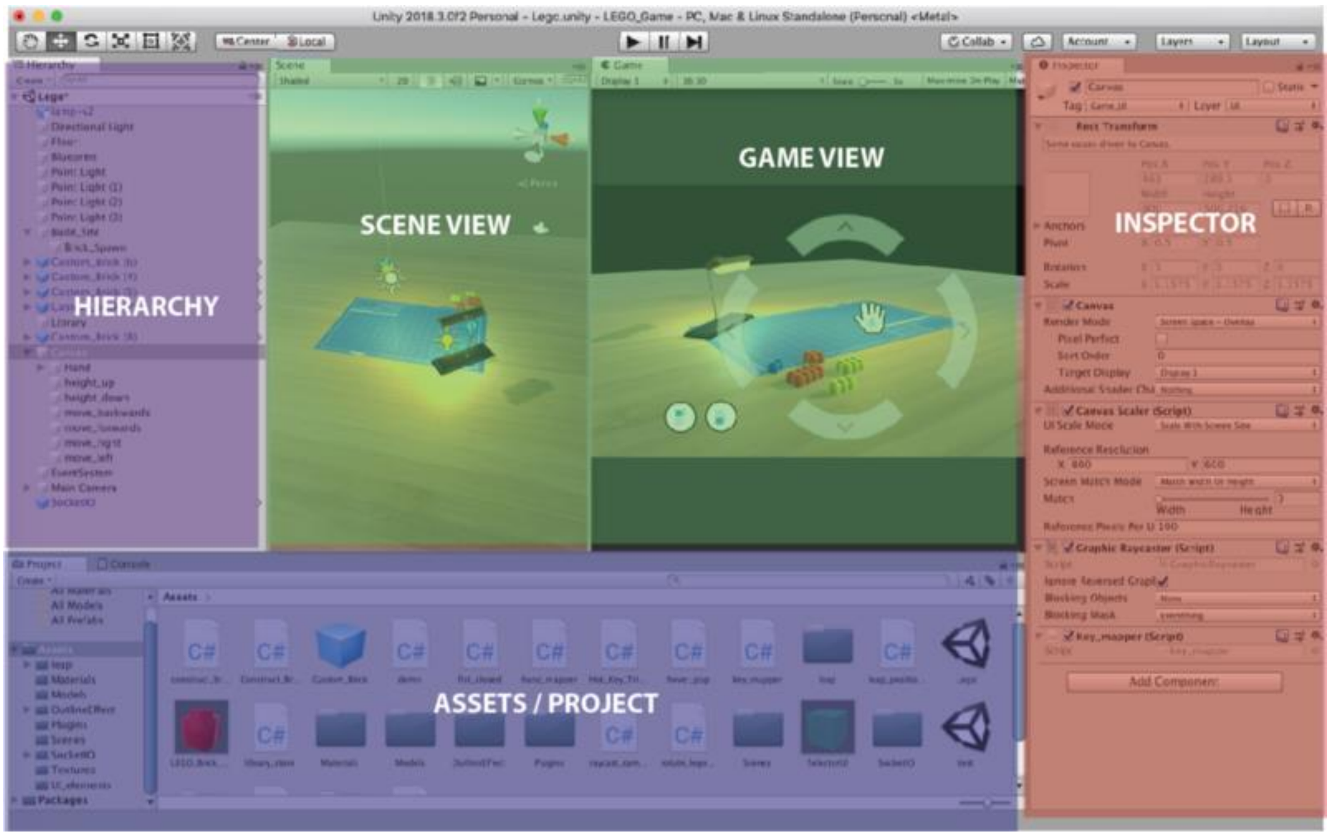

Unity 2018.3.0f2 Personal - Lego.unity - LEGO_Game - PC, Mac & Linux Standalone (Personal) <Metal>
HIERARCHY
SCENE VIEW
GAME VIEW
INSPECTOR
ASSETS / PROJECT

Figure 1.2.2 Unity Editor Window

Hierarchy: The hierarchy contains all the objects that will be used in the current scene. The objects can be easily structured.
Inspector: This windows contains all the information about a selected GameObject. The inspector window allows the user to see each property of the object, to add or remove properties, to modify the current ones and to attach scripts.
Game View: Previews the game point of view, allowing the user to see the final look of the scene.
Scene View: This window represents a scene view, allowing the user to select and modify each object inside the scene. The objects might get their position, rotation or scale modifyed.
Assets / Project: This represents the resources window. It might contain, scripts, materials, objects, images, shaders and many more.

1.2.3 Unity Game Objects

GameObjects are the main type of objects that Unity3D offers. Each thing that is placed inside a scene is wrapped in a game object. This represents a base class that has a lot of properties. The GameObject tab contains a various range of already defined GameObjects [1.2.3] like:

- 3D Objects: Cubes, Capsules, Spheres
- 2D Objects: Sprites, Tilemaps
- Effects: Particle Systems, Lines, Trails
- Light: Point Light, Directional Light, Spot Light

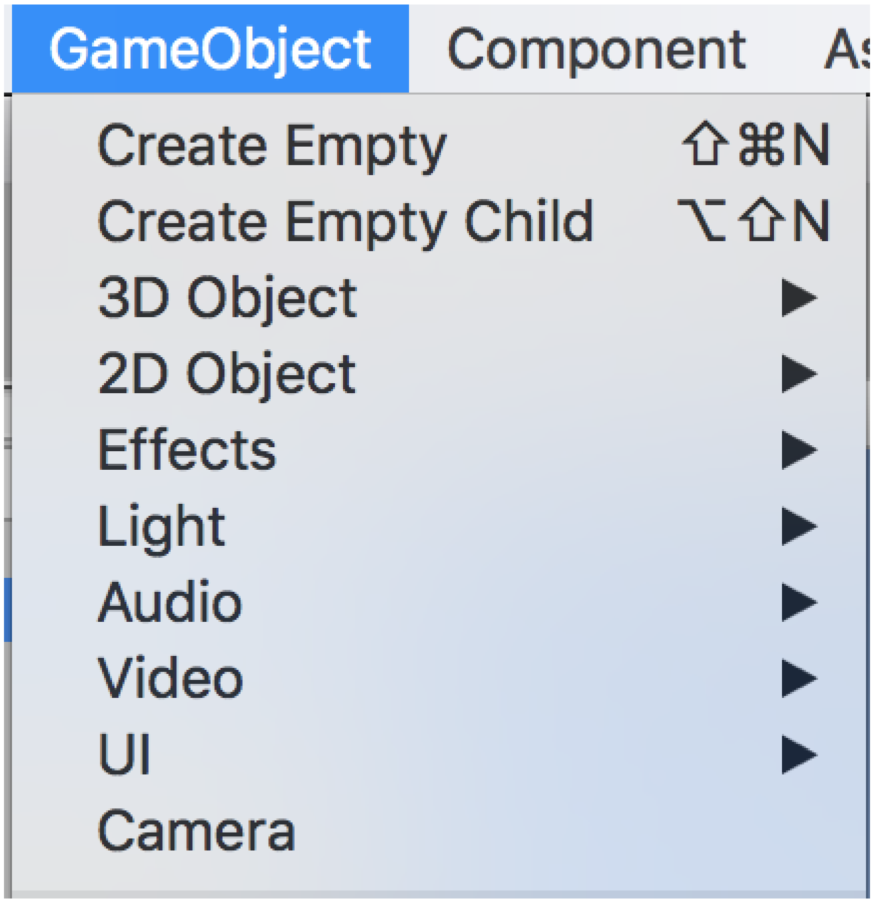


Figure 1.2.3 Unity Standard GameObjects

Chapter 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 [2.2.1]:

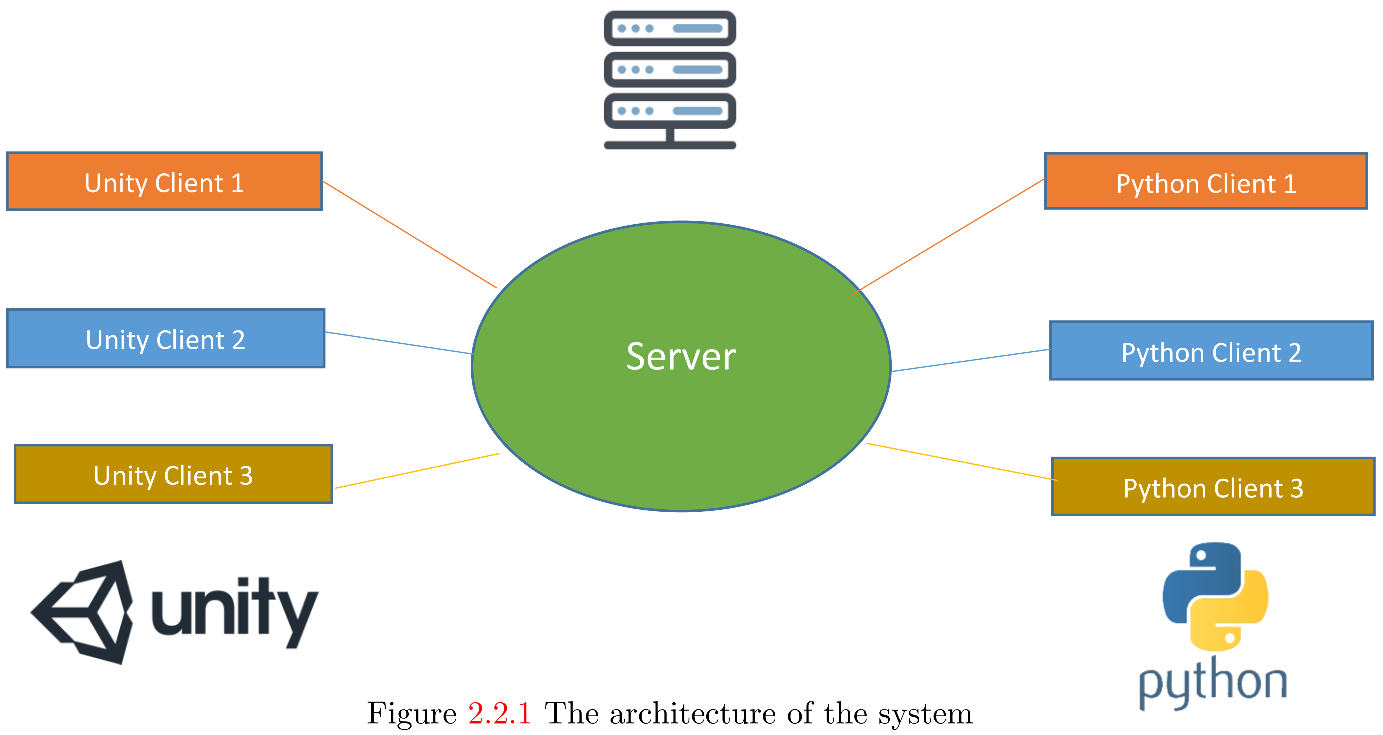


Figure 2.2.1 The architecture of the system

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 [2.3.1] 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.

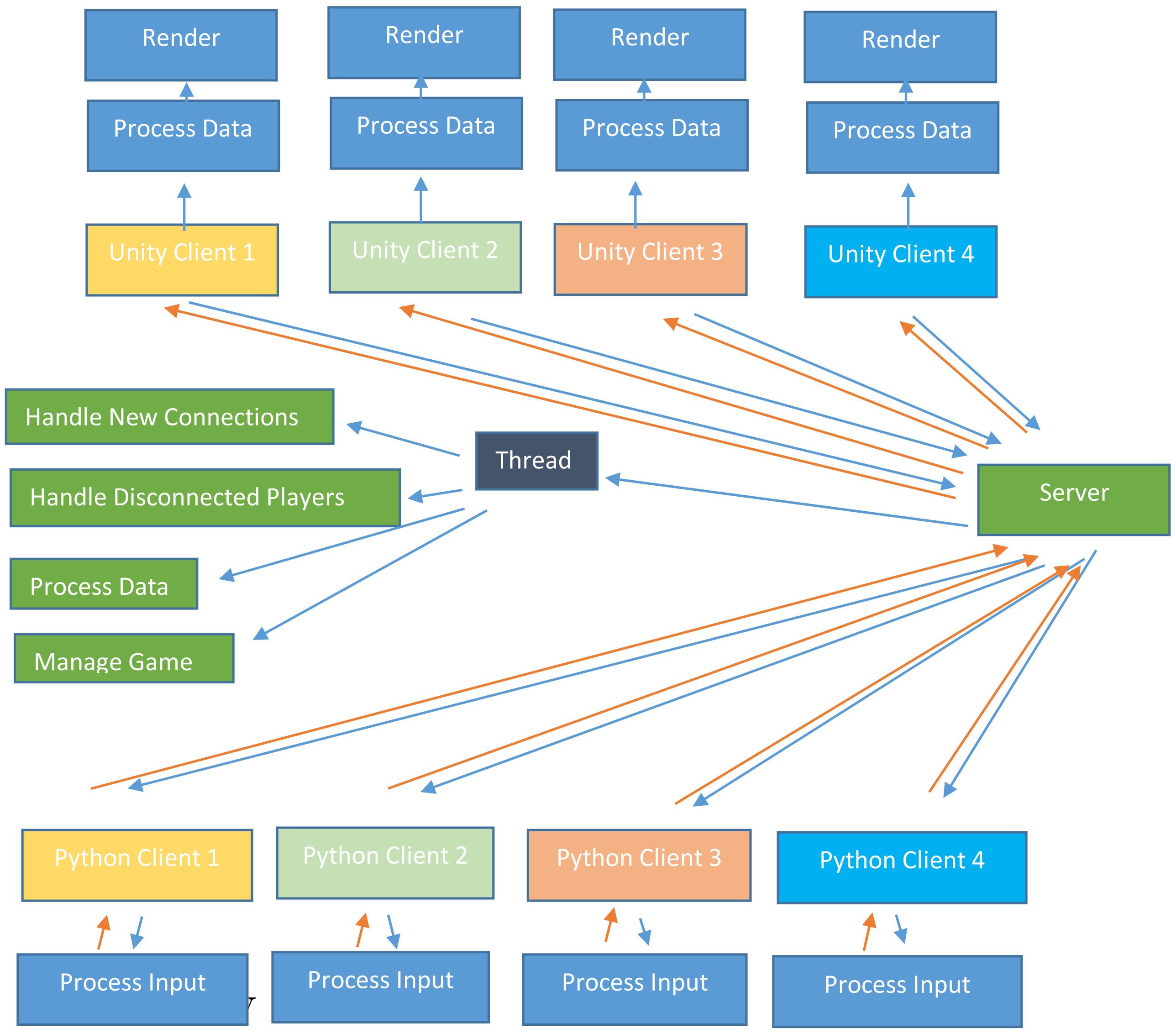


Figure 2.3.1 Program flow

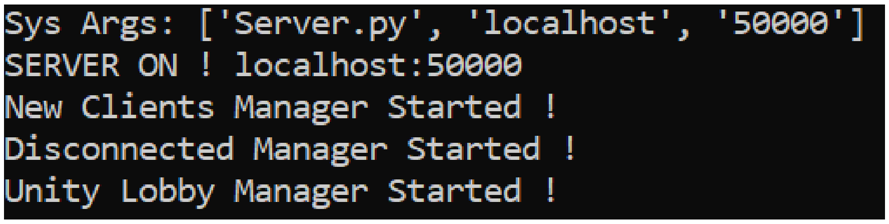
Taking into consideration that there are three modules that share data and work together, there might be a concern about the security of the entire system but, that is not the case.

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.

2.4 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 [2.4]. 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.

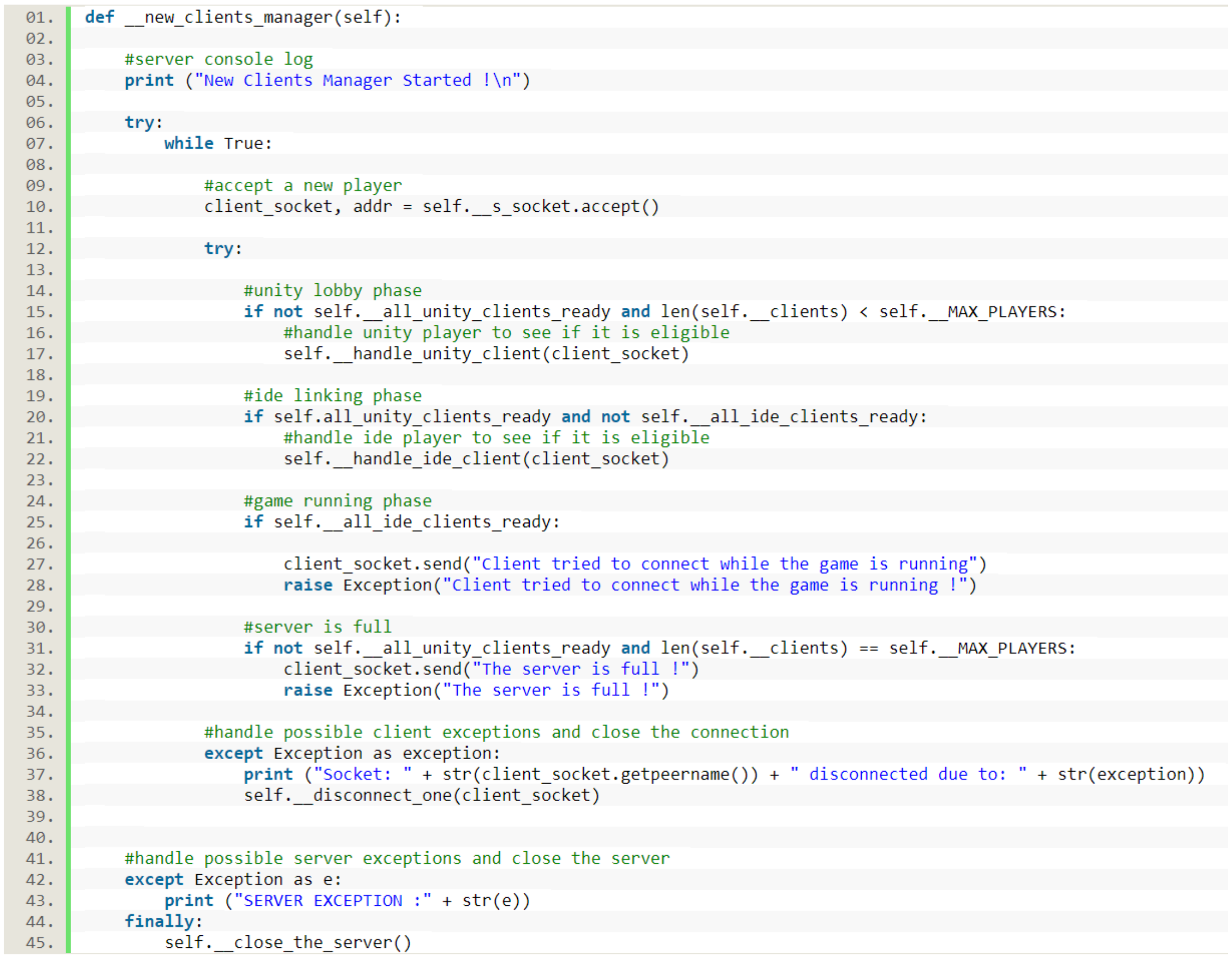


```
Sys Args: ['Server.py', 'localhost', '50000']
SERVER ON ! localhost:50000
New Clients Manager Started !
Disconnected Manager Started !
Unity Lobby Manager Started !
```

Figure 2.4 Server console log

2.4.1 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 [2.4.1]. 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.



```python
def __new_clients_manager(self):

    #server console log
    print ("New Clients Manager Started !\n")

    try:
        while True:

            #accept a new player
            client_socket, addr = self.__s_socket.accept()

            try:

                #unity lobby phase
                if not self.__all_unity_clients_ready and len(self.__clients) < self.__MAX_PLAYERS:
                    #handle unity player to see if it is eligible
                    self.__handle_unity_client(client_socket)

                #ide linking phase
                if self.all_unity_clients_ready and not self.__all_ide_clients_ready:
                    #handle ide player to see if it is eligible
                    self.__handle_ide_client(client_socket)

                #game running phase
                if self.__all_ide_clients_ready:

                    client_socket.send("Client tried to connect while the game is running")
                    raise Exception("Client tried to connect while the game is running !")

                #server is full
                if not self.__all_unity_clients_ready and len(self.__clients) == self.__MAX_PLAYERS:
                    client_socket.send("The server is full !")
                    raise Exception("The server is full !")

            #handle possible client exceptions and close the connection
            except Exception as exception:
                print ("Socket: " + str(client_socket.getpeername()) + " disconnected due to: " + str(exception))
                self.__disconnect_one(client_socket)


    #handle possible server exceptions and close the server
    except Exception as e:
        print ("SERVER EXCEPTION :" + str(e))
    finally:
        self.__close_the_server()
```

Figure 2.4.1 New clients Manager Method.

2.4.2 Connection 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.

2.4.3 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

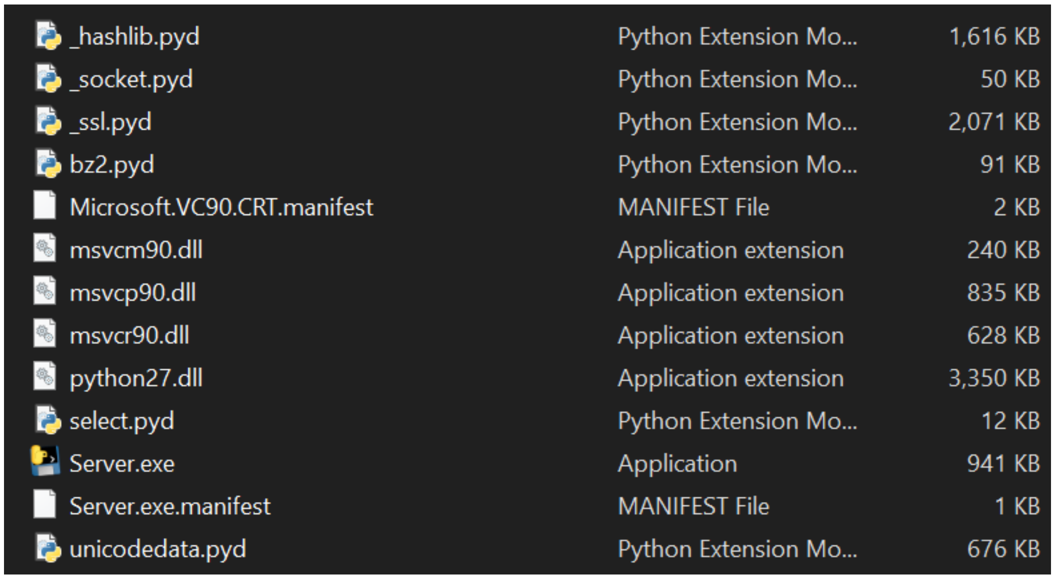
As the name suggests, this manager handles the disconnected players. The code is running on a separate thread and it periodically iterates the list of players and checks if a player has disconnected status. If so, it safely closes the connection between the server and that player, removing the user from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

2.4.4 Server Build

Using pyinstaller we made a server build [2.4.1]. 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.



_hashlib.pyd	Python Extension Mo...	1,616 KB
_socket.pyd	Python Extension Mo...	50 KB
_ssl.pyd	Python Extension Mo...	2,071 KB
bz2.pyd	Python Extension Mo...	91 KB
Microsoft.VC90.CRT.manifest	MANIFEST File	2 KB
msvcm90.dll	Application extension	240 KB
msvcp90.dll	Application extension	835 KB
msvcr90.dll	Application extension	628 KB
python27.dll	Application extension	3,350 KB
select.pyd	Python Extension Mo...	12 KB
Server.exe	Application	941 KB
Server.exe.manifest	MANIFEST File	1 KB
unicodedata.pyd	Python Extension Mo...	676 KB

Figure 2.4.4 Server build files

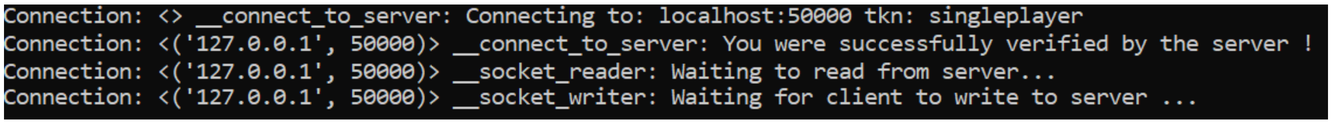
2.5 Python client

2.5.1 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.

2.5.2 Client validation by the server

After a successful connection to the server, the client token needs to be verified [2.5.2]. 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.



```
Connection: <> __connect_to_server: Connecting to: localhost:50000 tkn: singleplayer
Connection: <('127.0.0.1', 50000)> __connect_to_server: You were successfully verified by the server !
Connection: <('127.0.0.1', 50000)> __socket_reader: Waiting to read from server...
Connection: <('127.0.0.1', 50000)> __socket_writer: Waiting for client to write to server ...
```

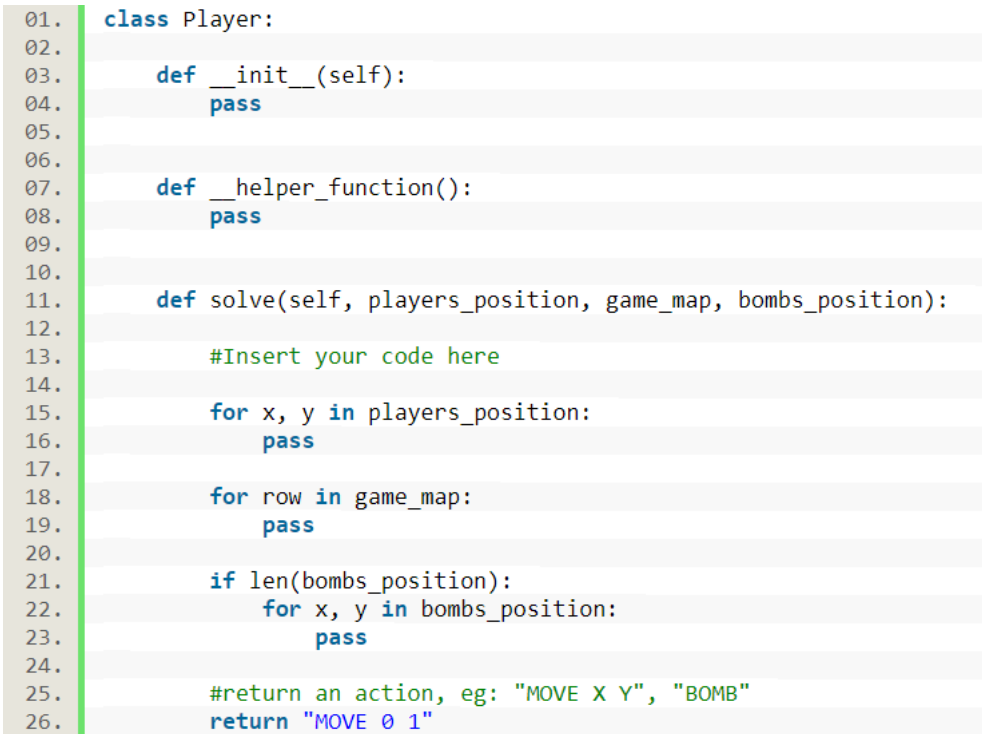
Figure 2.5.2 Client console log

2.5.3 Sending and receiving data

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.

2.5.4 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 [2.5.4]. 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.



```python
class Player:

    def __init__(self):
        pass


    def __helper_function():
        pass


    def solve(self, players_position, game_map, bombs_position):

        #Insert your code here

        for x, y in players_position:
            pass

        for row in game_map:
            pass

        if len(bombs_position):
            for x, y in bombs_position:
                pass

        #return an action, eg: "MOVE X Y", "BOMB"
        return "MOVE 0 1"
```

Figure 2.5.4 Player Class

2.6 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 [2.6] because of the 3D asset it features.

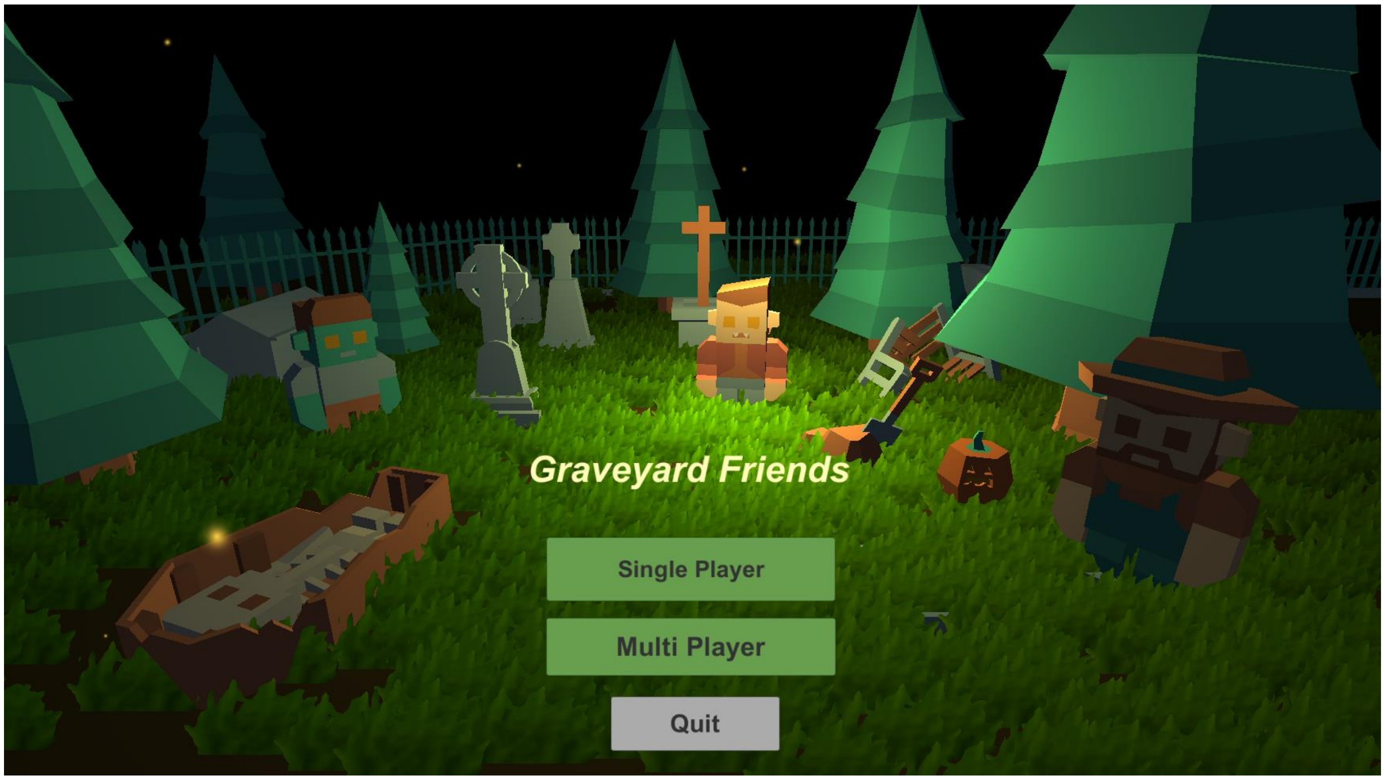


Figure 2.6 Game Main Menu

2.6.1 Level design

The level design was made using a 3D asset [2.6.1.1] for the scene environment, unity terrain for generating the grass and unity particles [2.6.1.2] for some effects. The asset is from Kenney.nl site and is called "Graveyard Kit", being a free to use pack with 3d models.

Figure 2.6.1.1 Asset containing the 3D Models

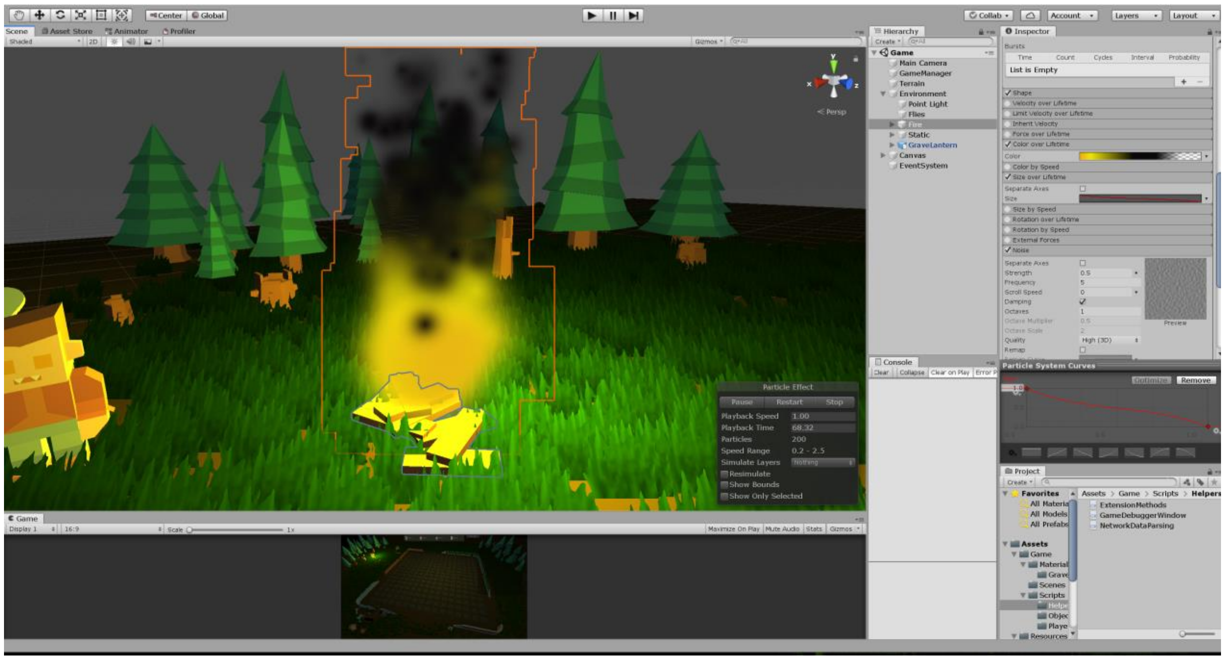


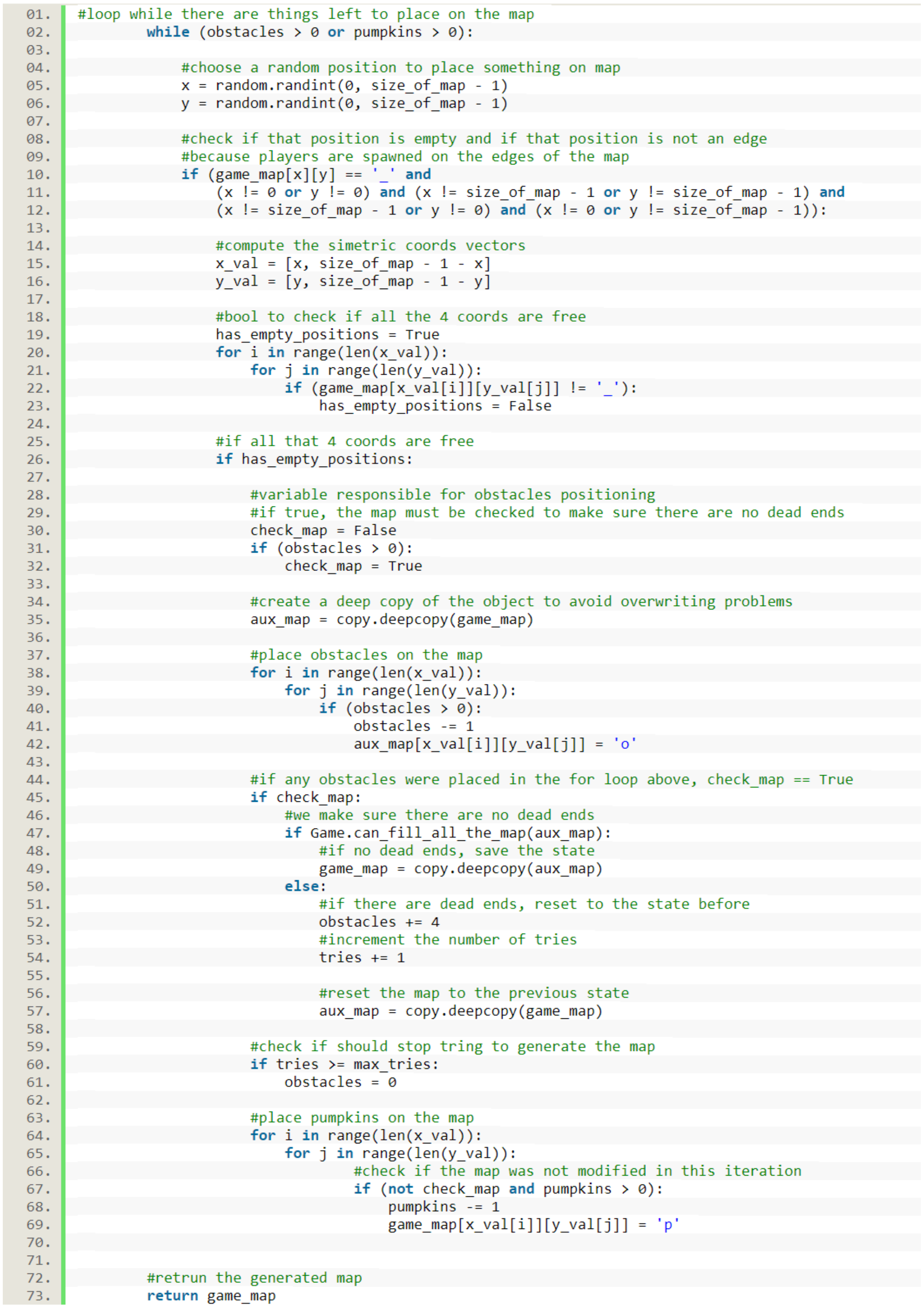
Figure 2.6.1.2 Particles System

2.6.2 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 [2.6.2] 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.



```python
#loop while there are things left to place on the map
        while (obstacles > 0 or pumpkins > 0):

            #choose a random position to place something on map
            x = random.randint(0, size_of_map - 1)
            y = random.randint(0, size_of_map - 1)

            #check if that position is empty and if that position is not an edge
            #because players are spawned on the edges of the map
            if (game_map[x][y] == '_' and
                (x != 0 or y != 0) and (x != size_of_map - 1 or y != size_of_map - 1) and
                (x != size_of_map - 1 or y != 0) and (x != 0 or y != size_of_map - 1)):

                #compute the simetric coords vectors
                x_val = [x, size_of_map - 1 - x]
                y_val = [y, size_of_map - 1 - y]

                #bool to check if all the 4 coords are free
                has_empty_positions = True
                for i in range(len(x_val)):
                    for j in range(len(y_val)):
                        if (game_map[x_val[i]][y_val[j]] != '_'):
                            has_empty_positions = False

                #if all that 4 coords are free
                if has_empty_positions:

                    #variable responsible for obstacles positioning
                    #if true, the map must be checked to make sure there are no dead ends
                    check_map = False
                    if (obstacles > 0):
                        check_map = True

                    #create a deep copy of the object to avoid overwriting problems
                    aux_map = copy.deepcopy(game_map)

                    #place obstacles on the map
                    for i in range(len(x_val)):
                        for j in range(len(y_val)):
                            if (obstacles > 0):
                                obstacles -= 1
                                aux_map[x_val[i]][y_val[j]] = 'o'

                    #if any obstacles were placed in the for loop above, check_map == True
                    if check_map:
                        #we make sure there are no dead ends
                        if Game.can_fill_all_the_map(aux_map):
                            #if no dead ends, save the state
                            game_map = copy.deepcopy(aux_map)
                        else:
                            #if there are dead ends, reset to the state before
                            obstacles += 4
                            #increment the number of tries
                            tries += 1

                            #reset the map to the previous state
                            aux_map = copy.deepcopy(game_map)

                    #check if should stop tring to generate the map
                    if tries >= max_tries:
                        obstacles = 0

                    #place pumpkins on the map
                    for i in range(len(x_val)):
                        for j in range(len(y_val)):
                                #check if the map was not modified in this iteration
                                if (not check_map and pumpkins > 0):
                                    pumpkins -= 1
                                    game_map[x_val[i]][y_val[j]] = 'p'


        #retrun the generated map
        return game_map
```

Figure 2.6.2 Map Generation Code
]

2.6.3 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.

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.

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

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).

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.

2.6.4 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>.

2.6.5 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.

Game preparation

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 [2.6.5.1]. 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 accordingly [2.6.5.2].

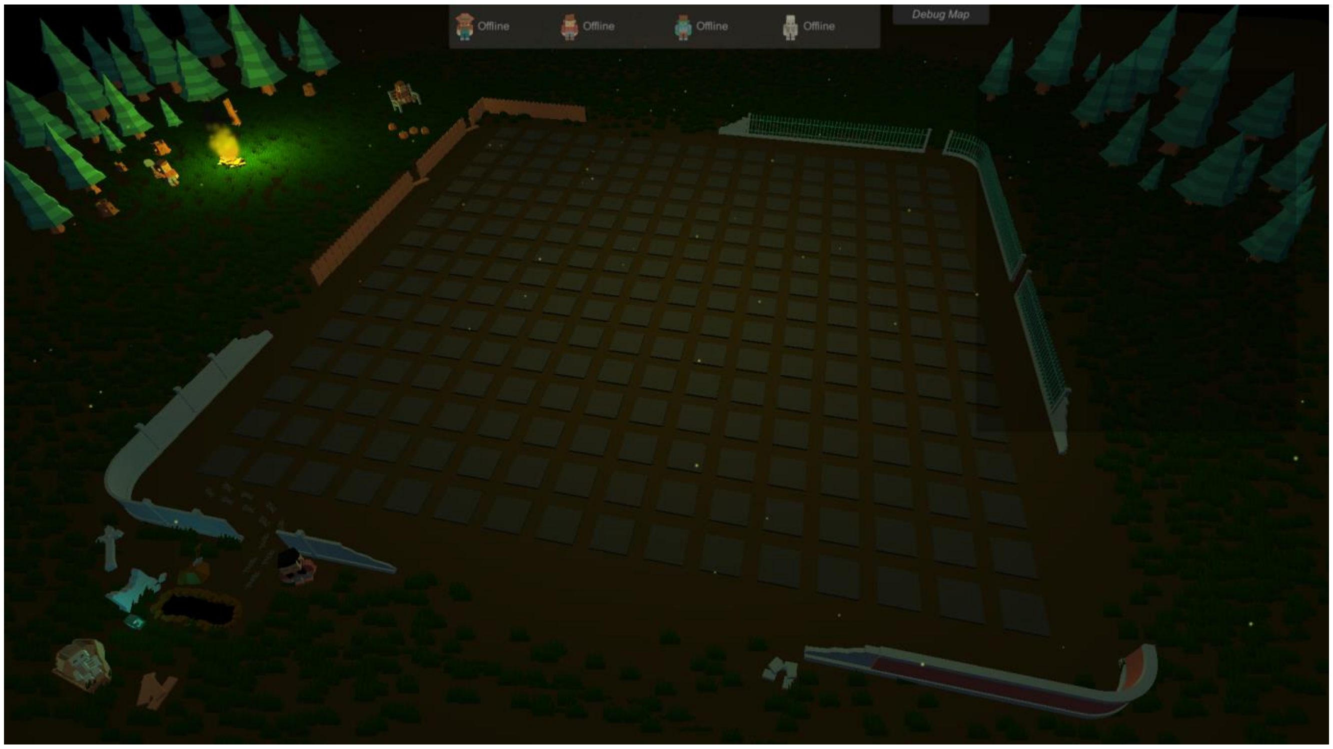
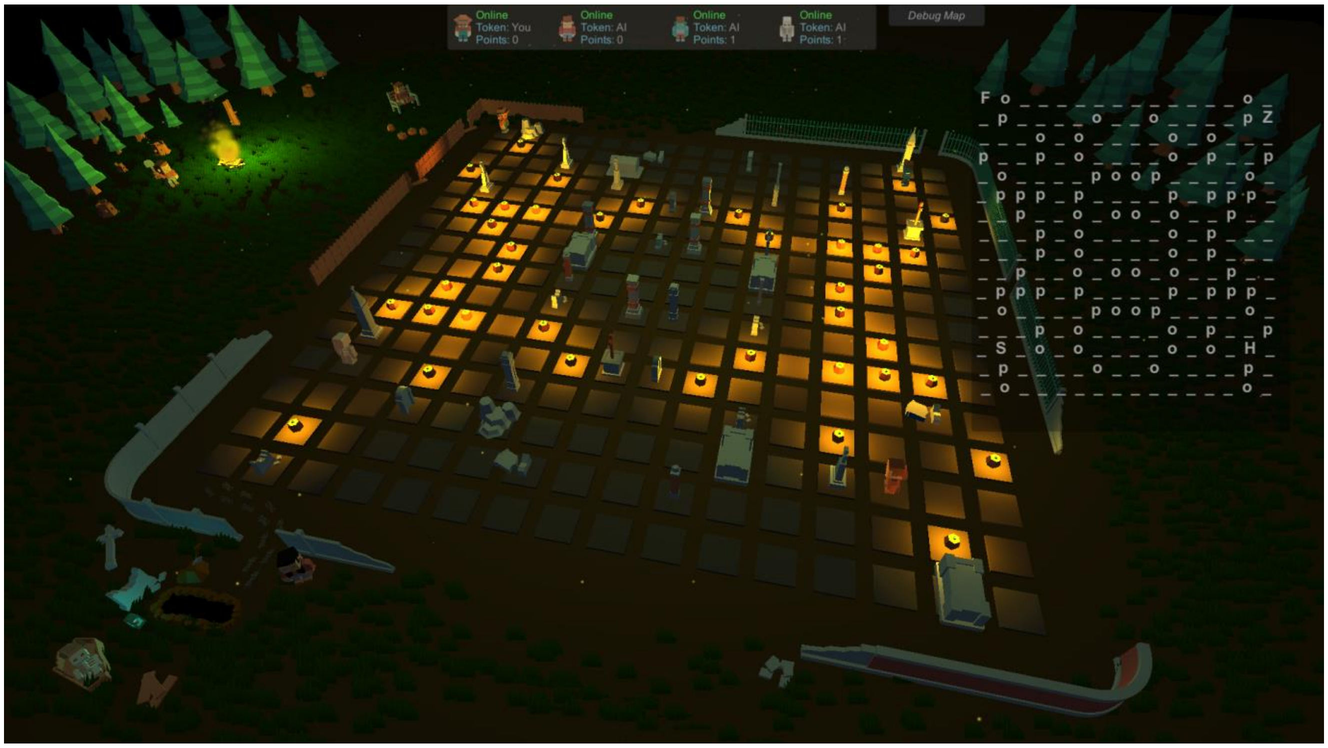


Figure 2.6.5.1 Map before preparation

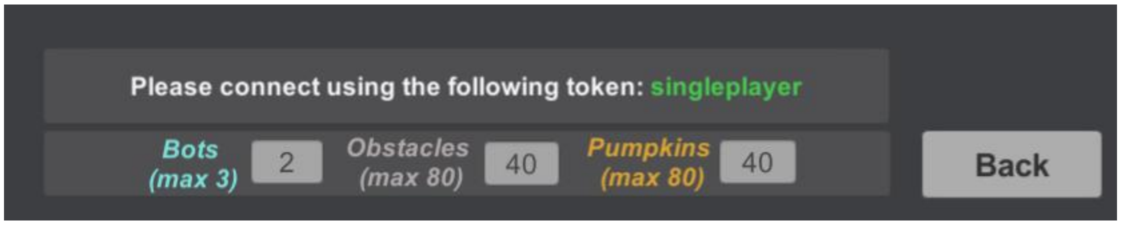


```
F o _ _ _ _ _ _ _ _ _ _ _ _ o _
_ p _ _ _ _ o _ _ o _ _ _ _ p Z
_ _ _ o _ o _ _ _ _ o _ o _ _ _
p _ _ p _ o _ _ _ _ o _ p _ _ p
_ o _ _ _ _ p o o p _ _ _ _ o _
_ p p p _ p _ _ _ _ p _ p p p _
_ _ p _ _ o _ o o _ o _ _ p _ _
_ _ _ p _ o _ _ _ _ o _ p _ _ _
_ _ _ p _ o _ _ _ _ o _ p _ _ _
_ _ p _ _ o _ o o _ o _ _ p _ _
_ p p p _ p _ _ _ _ p _ p p p _
_ o _ _ _ _ p o o p _ _ _ _ o _
_ _ _ p _ o _ _ _ _ o _ p _ _ p
_ S _ o _ o _ _ _ _ o _ o _ H _
_ p _ _ _ _ o _ _ o _ _ _ _ p _
_ o _ _ _ _ _ _ _ _ _ _ _ _ o _
```

Figure 2.6.5.2 Map after preparation

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 [2.6.5] and last but not the least, there are AIs to play with. 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.


Please connect using the following token: singleplayer

Bots (max 3): 2
Obstacles (max 80): 40
Pumpkins (max 80): 40

Figure 2.6.5 Single player parameters panel

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

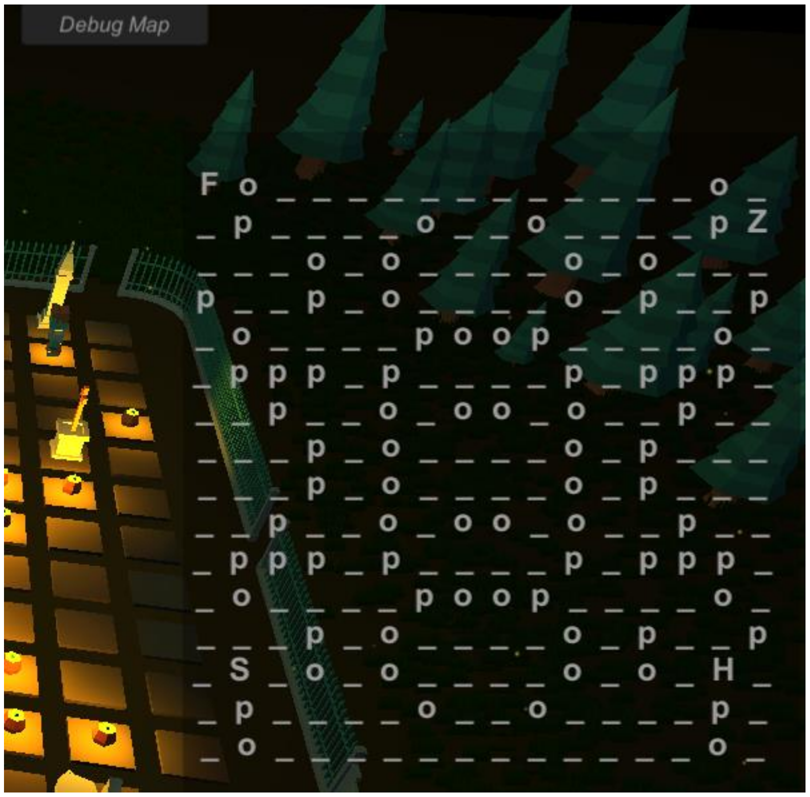
should, leading to a bug. Next, each players command is sent to a handler that will execute it. The handler checks if the player is able to perform that specific action and if so, it gets executed, otherwise, nothing happens. The available actions are: MOVE X Y, BOMB.
At the end of the round, the players 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.

2.6.6 Map Debugger

Map debugger is a tool that helps a user to see what is going on the map [2.6.6]. 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.



```
F o _ _ _ _ _ _ _ _ _ _ _ _ o _
_ p _ _ _ _ o _ _ o _ _ _ _ p Z
_ _ _ o _ o _ _ _ _ o _ o _ _ _
p _ _ p _ o _ _ _ _ o _ p _ _ p
_ o _ _ _ _ p o o p _ _ _ _ o _
_ p p p _ p _ _ _ _ p _ p p p _
_ _ p _ _ o _ o o _ o _ _ p _ _
_ _ _ p _ o _ _ _ _ o _ p _ _ _
_ _ _ p _ o _ _ _ _ o _ p _ _ _
_ _ p _ _ o _ o o _ o _ _ p _ _
_ p p p _ p _ _ _ _ p _ p p p _
_ o _ _ _ _ p o o p _ _ _ _ o _
_ _ _ p _ o _ _ _ _ o _ p _ _ p
_ S _ o _ o _ _ _ _ o _ o _ H _
_ p _ _ _ _ o _ _ o _ _ _ _ p _
_ o _ _ _ _ _ _ _ _ _ _ _ _ o _
```

Figure 2.6.6 Map debugger

2.7 Strong Points

- The games can be played with no internet connection
- The response time is instant, apart from the web platforms where thousands of users are connected together, this platform relies only on the speed of the connected players, which in our case can be a maximum of 4.
- Easy hosting of the server on the internet because it supports command line parameters and also comes in two forms: executable and source code
- The system is built in such a way that adding a new game is simple, besides the information about the game, all the classes are generic. There is nothing more to modify except for the Game Manager class and the game rules for the python server and client.
- Customizable map while playing single player.
- Map debugger for an easier implementation of the strategy.
- User friendly interface and easy to use program.

Future work

Unity Game

- Add unity player the possibility to create a server from the game.
- Add more games to the platform.
- Improve the strategy of the AI and create a league system for the players.
- Create a match finder scene where the players would be able to access random games all over the world.
- Create a debugging system inside the single player game, allowing the player to pause the game.
- Adding more details about the current game like: the last executed command, current command, a dashboard with the current winner.
- Add sound system for a better game feedback.
- Character builder. Allow players to create their own character.
- Messages system. Allow players to send messages using their character.
- Creating a build for Web, Android, iOS.
- Create the possibility to write code inside the Unity Game.

Server

- Reduce the data traffic in order to be able to host the server on an internet domain.

Client

- Add some helper functions for the player and more instructions.
- Add support for different coding languages.

Conclusion

Bibliography

[1] Prof. Teodora Bakardjieva. Introduction to Computer Networking
https://www.vfu.bg/en/e-Learning/Computer-Networks-Introduction_Computer_Networking.pdf

[2] Data packet https://www.techopedia.com/definition/6751/data-packet

[3] Computer networks and data communication
https://peda.net/kenya/ass/subjects2/computer-studies/form-4/itcn

[4] Vangie Beal. Network and communication
https://www.webopedia.com/TERM/N/network.html

[5] Network Address
https://www.techopedia.com/definition/20969/network-address

[6] Vivek Shah. Reasons why unity3d is so much popular in the gaming industry.
https://medium.com/@vivekshah.P/reasons-why-unity3d-is-so-much-popular-in-the-gaming-industry-705898a2a04

[7] What is a socket. https://www.tutorialspoint.com/unix_sockets/what_is_socket.htm

[8] Socket Programming
https://www.cs.dartmouth.edu/~campbell/cs60/socketprogramming.html

[9] Client-Server Network https://computrnetworking.wordpress.com/2012/02/20/client-server-network/