Edinburgh Napier UNIVERSITY | School of Computing

Can player immersion in a tabletop role-playing game be improved using an interactive tabletop companion application?

Development Report

Submitted in partial fulfilment of

the requirements of Edinburgh Napier University

for the Degree of

Computing and User Experience

April 2021

Word Count: 2556

Contents

1. Project Definition

The artefact is an Android application designed to improve player immersion during a game of the tabletop RPG (role-playing game) Dungeons and Dragons. The artefact was created for the Android mobile platform, as few virtual tabletop applications exist for mobile. The artefact's aim is to enhance the player experience and improve immersion within the RPG's world.

2. Methodology and Implementation

The development of the artefact followed the iterative design process, whereby functionality was added piece by piece and tested throughout, until all requirements were completed. Development progressed in stages, loosely following the Gantt chart timeline outlined in this project's Scoping Document (See Appendix A).

2.1 Android Studio

Initial development started on the Android Studio platform, before the finalisation of the Scoping Document. Android Studio is an IDE (Integrated Development Environment) and allows users to develop both the visual and back-end of an application i.e., the XML files that handle the visual aspects of an application, and Java class files that contain the code behind the application. An initial framework was developed for switching between three different "views" or screens (See Appendix B), however after doing research and exploring the capabilities of Android Studio, it was deemed that another platform would be better suited, as Android Studio has some limitations regarding custom GUI (Graphical user Interface) elements. Android Studio does allow the creation of custom components (based on their base layout classes: "View" and "ViewGroup"), which can be altered to create any number of custom UI elements. Therefore, it would have been possible to have created the artefact using Android Studio, but it would have been an arduous task, as almost all the UI elements would have to be custom and have taken a long time to produce. Hence, the Android Studio framework was abandoned, and Unity was chosen as the development environment instead. Unity is primarily used for game design and had all the functionality required to create the artefact in a shorter time span and with fewer visual limitations.

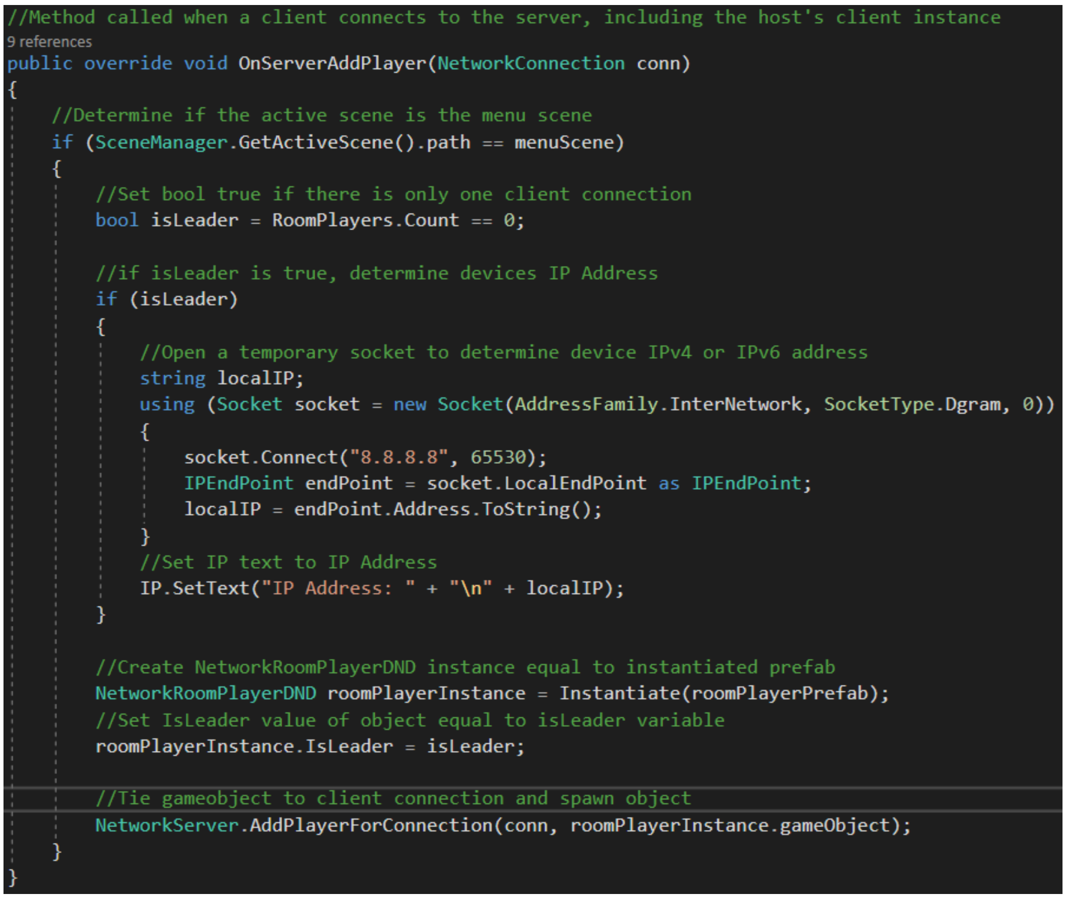
2.2 Unity

When development had moved to Unity, a whole new strategy had to be developed. Starting again at the beginning of the iterative process, research was required to understand how to develop a grid-based system for the artefact. Initial research focussed on following tutorials to create chess games made within the Unity engine (VR with Andrew, 2018). This was a good starting point as not only does chess work to a grid, but each piece has rules regarding their movement. After more research, a series of tutorial videos were found that relate to the creation of a custom grid and tile map system (Code Monkey, 2019b). This approach was then adopted as the basis for the creation of the grid system in the artefact. Following this, the next area researched was multiplayer. Unity, at the time of artefact creation, had no inbuilt multiplayer solution. The simplest solution was to use a plugin/package called Mirror ("Mirror Networking – Open Source Networking for Unity", n.d.). Mirror is a networking library based on Unity's abandoned UNET networking system. Another tutorial series was followed to understand how to create a local multiplayer system for the artefact (Dapper Dino, 2020). Once basic functionality for both the grid system and multiplayer were added, the process of refining each and integrating more systems began.

Unity is structured using "Scenes", which are containers for all, or parts of a game or application (i.e., levels). Within a Scene, there are Game Objects, which, as a base, contain only positional coordinates within said scene. Components can be added onto Game Objects to add functionality, these might be a sprite or image in a 2D game, or a 3D object with accompanying textures. Scripts are a type of component that adds C# code to a given Game Object or Scene, with almost limitless possibilities. A Prefab is a Game Object that is saved as a template, and any number instances of it can be created within a scene, already containing any attached scripts or other components. In a multiplayer game, it can be thought of that a Game Object on one device is the same as its equivalent on another device, but only if they are linked in some way through networking code.

2.2.1 Network Manager

The most important element within the artefact is the Network Manager object and adjoining script. The Network Manager handles multiplayer functionality within both the lobby scene (default scene) and game scene. When a player hosts a game, they create both a client and server instance run from the same device, which other devices can connect to as clients by connecting to the host device's IP address (all games are hosted through port 7777, this is the default used by Mirror but can be changed). The custom Network Manager class inherits from Mirror's own Network Manager class; however, most functions were overwritten to execute custom code. The “OnServerAddPlayer()” method (Fig. 1) in the “NetworkManagerDND” class is called on the server when a client connects, this includes the hosts own client connection as they are both the server and client, and determines which player is the leader and generates player objects and ties them to clients.



```csharp
//Method called when a client connects to the server, including the host's client instance
9 references
public override void OnServerAddPlayer(NetworkConnection conn)
{
    //Determine if the active scene is the menu scene
    if (SceneManager.GetActiveScene().path == menuScene)
    {
        //Set bool true if there is only one client connection
        bool isLeader = RoomPlayers.Count == 0;

        //if isLeader is true, determine devices IP Address
        if (isLeader)
        {
            //Open a temporary socket to determine device IPv4 or IPv6 address
            string localIP;
            using (Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, 0))
            {
                socket.Connect("8.8.8.8", 65530);
                IPEndPoint endPoint = socket.LocalEndPoint as IPEndPoint;
                localIP = endPoint.Address.ToString();
            }
            //Set IP text to IP Address
            IP.SetText("IP Address: " + "\n" + localIP);
        }

        //Create NetworkRoomPlayerDND instance equal to instantiated prefab
        NetworkRoomPlayerDND roomPlayerInstance = Instantiate(roomPlayerPrefab);
        //Set IsLeader value of object equal to isLeader variable
        roomPlayerInstance.IsLeader = isLeader;

        //Tie gameobject to client connection and spawn object
        NetworkServer.AddPlayerForConnection(conn, roomPlayerInstance.gameObject);
    }
}
```

Figure 1: OnServerAddPlayer() method

When all clients are connected to the host, the host can begin the game and launch everyone into the game scene. This calls methods “ServerChangeScene()” and “OnServerSceneChanged()” (See Appendix C). The first of these methods handles changing the client objects from being instances of the “RoomPlayer” class to instances of the “GamePlayer” class, and transfers across relevant data including names, icons, and character sheet data. The second of these methods is responsible for creating the spawn system game object on the server and all clients.

2.2.2 Spawn System

The “SpawnSystem” script is responsible for handling spawning of the client objects and host objects (See Appendix D). During development, it was decided that the spawn locations for player tokens should be determined at run time, as there should only be ‘x’ number of spawn locations equal to the number of players in the game. When the “SpawnPlayer()” method is called, it determines if the connection parameter sent to it is the host or a client and spawns a prefab on all clients with control tied to said connection, with client connections spawning the “player prefab” and the host connection the “host prefab”.

2.2.3 Client Controls

When a user selects to join a game, rather than to host, they are guided through an interface for selecting their player token and setting up their character sheet data and connecting to an IP Address (see fig. 2, Appendix E). The data gathered through this UI is saved using several scripts (See Appendix F) and persists through application restarts. A character sheet display was developed, using one of Kenney’s UI packs ("Kenney • UI Pack: RPG Expansion", n.d.), and a script created for populating and handling said character sheet (See Appendix G).

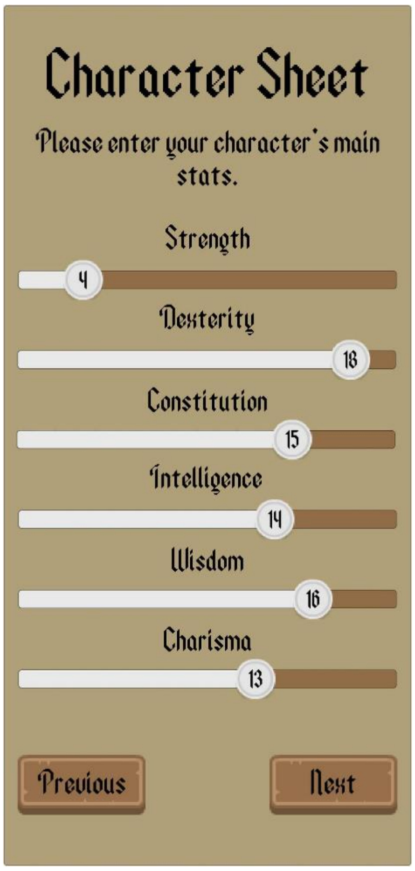


Figure 2: Main Stats selection

The initial system for moving player tokens between grid cells used A* pathfinding to determine the least “costly” path between a given start and end point ("Introduction to A*", n.d.). This approach was abandoned after considering that players should have a greater control over which grid cells they move between. However, the code used for gathering reference to neighbouring cells would be reused to calculate whether a given cell is occupied by a "wall" object and hence not a valid movement. The movement code for clients is held within the "DNDCombatUnit" script (See Appendix H), which is a component of "PlayerCharacter" prefab game objects spawned on the server by the Spawn System. The methods responsible for moving an object on the grid contain some of the most complicated code in the artefact. In short, when a client touches a square within the immediate radius of their token, that cell is added to a list of Vector3 objects (Vector3 objects contain three float values; x, y, and z coordinates), the server then spawns a transparent "path" object on these cells to indicate where the token will move (Fig. 3), and finally when the client triggers their tokens movement, their token will traverse between the list of Vector3 locations using an asynchronous piece of code known as a coroutine. Whilst the client token is in motion, all movement code is disabled using a simple Boolean variable. This same code is reused by the host in the "DNDHost" script.

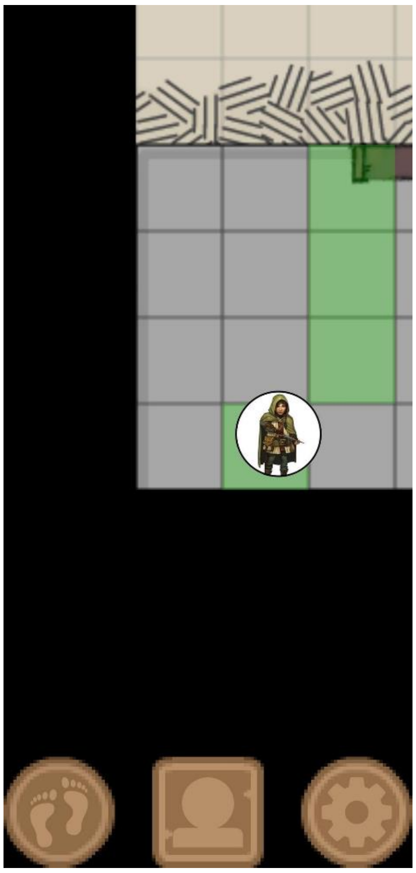
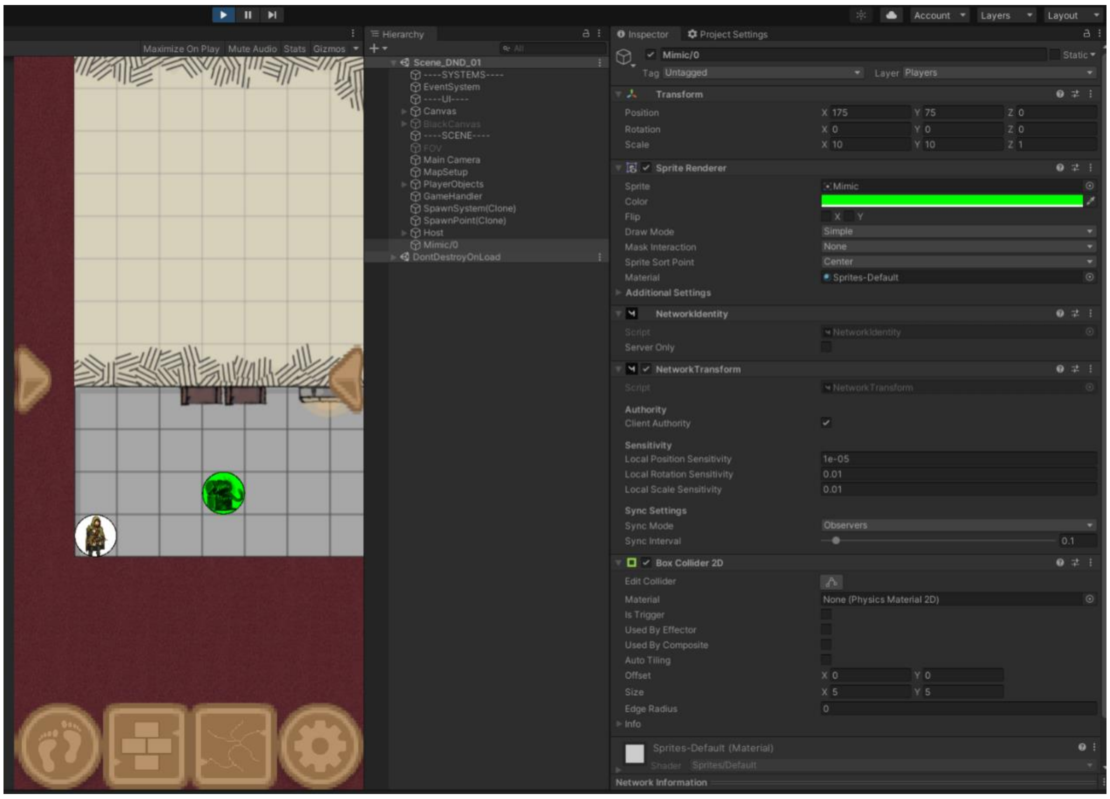


Figure 3: Player Character moving over Path Game Objects

2.2.4 Host Controls

Unlike client connections, the host does not have a token associated with them. Instead, the host can spawn instances of monsters onto the grid and move them. A method for selecting which monster the host is moving was devised using Unity's Raycast2D feature (See Appendix I). By attaching a collider to the monster prefab, when one is spawned onto the grid, the host can click said monster to select it, and then click grid squares to move said monster in the same manner as clients move their tokens (Fig. 4).



```
Mimic/0                                   Static
Tag Untagged          Layer Players
Transform
  Position      X 175   Y 75   Z 0
  Rotation      X 0     Y 0    Z 0
  Scale         X 10    Y 10   Z 1
Sprite Renderer
  Sprite              Mimic
  Color
  Flip                X  Y
  Draw Mode           Simple
  Mask Interaction    None
  Sprite Sort Point   Center
  Material            Sprites-Default
  Additional Settings
NetworkIdentity
  Script              NetworkIdentity
  Server Only
NetworkTransform
  Script              NetworkTransform
  Authority
  Client Authority    ✓
  Sensitivity
  Local Position Sensitivity   1e-05
  Local Rotation Sensitivity   0.01
  Local Scale Sensitivity      0.01
  Sync Settings
  Sync Mode           Observers
  Sync Interval       0.1
Box Collider 2D
  Edit Collider
  Material            None (Physics Material 2D)
  Is Trigger
  Used By Effector
  Used By Composite
  Auto Tiling
  Offset        X 0   Y 0
  Size          X 5   Y 5
  Edge Radius   0
  Info
Sprites-Default (Material)
  Shader  Sprites/Default
Network Information
```

Figure 4: A monster Prefab in Unity Inspector

A "state machine" was developed to handle the host's controls, as the host has multiple functions that they can activate with only one finger touch inputs (Creation of Walls, Rough Terrain, selecting Monsters, and moving Monsters). The state machine uses an Enum that contains multiple values and a state variable that can be read by conditional statements to determine which code to run (See Appendix J).

2.2.5 Fog of War/View Distance

During initial research, a method for creating a 360° circular mesh was discovered and tested (Code Monkey, 2019a), which would enable the background map to be obscured to the clients. Using the Universal Render Pipeline package ("Universal Render Pipeline (formerly LWRP) | Unity", n.d.), this could also allow for selected objects outside of the mesh to be masked or not rendered entirely. The mesh is altered using the Raycast2D feature; the ray casts radiate from a central point out to a fixed distance, and upon collision with a collider component, they return the position of the collider, which then updates the mesh. This constantly updates, meaning if any new colliders come within the mesh's area, the mesh will update accordingly. (Fig. 5).

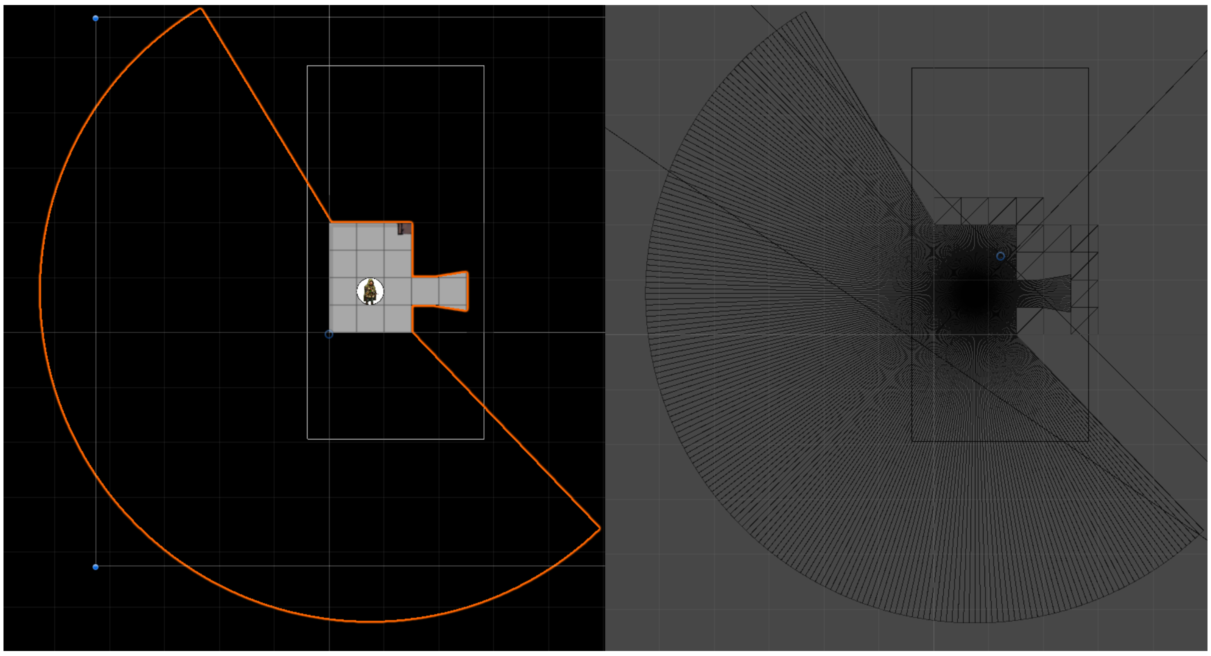


Figure 5: Shaded and Wireframe view of the Field of View mesh

Despite being one of the first features researched, this was one of the last features to be introduced. Given more time, several issues would be fixed, primarily to do with efficiency. The radius of the Field of View mesh extends for approximately 60ft or 12 grid cells, as this is the average visible distance for player characters with "darkvision". The mesh extends beyond the bounds of the background map, whilst this does not compromise any of the artefact's features, it is not very resource efficient and could be solved by adding a collider to the bounds of the background map.

2.3 Networking

The primary challenge when developing the artefact was that everything that happens on one instance of the artefact must be updated on all connected instances. Mirror contains several components and classes that enable this to happen.

2.3.1 Network Transform

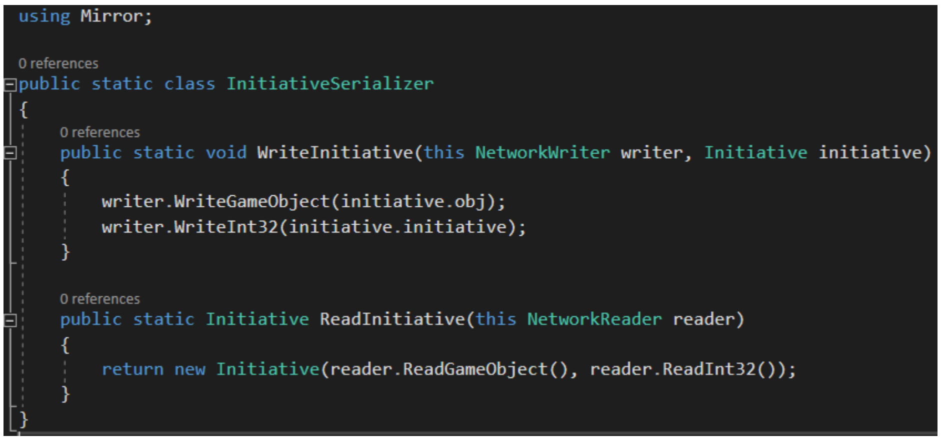
By attaching the Network Transform component to both the player token prefab and the monster prefab, when the game object moves on one instance, its positional data updates on all instances (See Appendix K). This was the simplest solution to the problem of updating player positions.

2.3.2 Server Commands and Client Remote Procedure Calls

Certain scripts, such as “DNDHost” and “DNDCombatUnit”, inherit from Mirror’s Network Behaviour class, which contains several useful features. By adding a “Command” tag before a method, said method can be called from the client but the code within is run on the server. This allows for individual client instances to send information to the server. On the other hand, the “ClientRpc” tag, means that the method can only be run from the server, but the code is executed on all client instances. By conjoining these two together, a client can send information to all other clients through the server. This was used in several ways, including updating monster sprites from the host to all connected clients (See Appendix L).

2.3.3 Synchronised Variables

Standard variable types can be prefaced with the “SyncVar” tag, which will automatically update their values across all clients when one instance is changed. This was used to synchronise a list of custom objects across all clients, containing a game object reference and initiative roll. To do this, a custom class had to be created for Mirror’s code to understand how to serialize the object (Fig. 6).



```
using Mirror;

0 references
public static class InitiativeSerializer
{
    0 references
    public static void WriteInitiative(this NetworkWriter writer, Initiative initiative)
    {
        writer.WriteGameObject(initiative.obj);
        writer.WriteInt32(initiative.initiative);
    }

    0 references
    public static Initiative ReadInitiative(this NetworkReader reader)
    {
        return new Initiative(reader.ReadGameObject(), reader.ReadInt32());
    }
}
```

Figure 6: Initiative Serializer class

2.4 Asset Development

The technical components of the delivered artefact were deemed more important than the visual and aesthetics of the artefact, hence more time was dedicated to the functionality of the artefact than the creation of UI and other visual elements. As mentioned above, assets were borrowed where appropriate from sources such as Kenney's UI packs ("Kenney • UI Pack: RPG Expansion", n.d.) and google images. Images sourced from google image search were filtered through creative commons and modified for use using Adobe Photoshop.

2.4.1 Maps

Initial development goals included the ability for users to upload custom map images, however, due to technical limitations and time constraints, this goal was altered. The issues that arose when dealing with user uploaded images stemmed from a problem sending said images across the local network between devices. The solution would likely require the creation of a method to split the byte information of an image into a series of data packets, broken down on one end and reassembled at the other. However, working on this would have eaten up a considerable amount of development time, to the detriment of the other systems present in the artefact. Instead, two map assets were created for users to choose between. These map assets were created using the free online tool Dungeon Scrawl ("Dungeon Scrawl", 2021), and then refined for use using Adobe Photoshop. The capability to scale the grid and player tokens to any background map still exist within the artefact, even though this could not be fully realised within the time frame.

2.4.2 Player Tokens

As above, the original development goal was to have players upload their own image to be used as a player token, and again this goal was altered. Five images were gathered through creative commons from google images, and edited in Adobe Photoshop, before bein g added to the artefact. These images were chosen specifically to represent the supplied character sheets for the Dungeons and Dragons campaign "Lost Mine of Phandelver".

3. Future Work

There are several areas where the project could have been improved, espe cially considering changes made during development due to unforeseen limitations and time constraints. With more time, a solution to the problem of user uploaded images could be addressed. By further exploration of Mirror's capabilities or researching alternatives, such as Unity's upcoming multiplayer solution, this issue could be resolved.

During development and initial research, the idea to create a map builder within the application was explored. A custom grid solution was created within the artefact; however, Unity contains its own grid and tile map system which could have been used to allow users to generate their own map from sprite assets. This development route would have been difficult to engineer, as Unity contains both Grid and Tile map Game Objects, but they are designed to be used within the Unity Editor environment, and not in a finished application. Tile maps work by "painting" sprite assets onto a Grid so that they connect, much like a jigsaw. Because of this, tile assets would have to have been created, taking up a larger portion of development time.

Lastly, after development had concluded, the Unity developers had begun working on their own multiplayer networking solution, dubbed MLAPI ("Unity Multiplayer Networking | MLAPI", 2021). Alongside this, Unity is also working on a new xml-based UI system, which was available during development but is still within the beta phase and is unstable. Returning to this project in the future, these new solutions might be a better alternative to those used within the artefact.

4. Conclusion

The lack of virtual tabletop applications for mobile platforms promotes the need for this work. Having access to a virtual tabletop during a tabletop RPG session, in a form that most users would have access to, would be beneficial to the player experience. With further development, and after having learned from mistakes and design limitations, this project could be developed into a successful commercial product.

References

- Code Monkey. (2019a). Field of View Effect in Unity (Line of Sight, View Cone) [Video]. Retrieved from https://www.youtube.com/watch?v=CSeUMTaNFYk&ab_channel=CodeMonkey

- Code Monkey. (2019b). Grid System in Unity 9Heatmap, Pathfinding, Building Area) [Video]. Retrieved from https://www.youtube.com/watch?v=waEsGu--9P8&ab_channel=CodeMonkey

- Dapper Dino. (2020). How To Make A Multiplayer Game In Unity - Client-Server - Mirror Networking [Video]. Retrieved from https://www.youtube.com/watch?v=5LhA4Tk_uvI&ab_channel=DapperDino

- Dungeon Scrawl. (2021). Retrieved 22 April 2021, from https://dungeonscrawl.com/

- Introduction to A*. Retrieved 22 April 2021, from http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

- Kenney • UI Pack: RPG Expansion. Retrieved 22 April 2021, from https://www.kenney.nl/assets/ui-pack-rpg-expansion

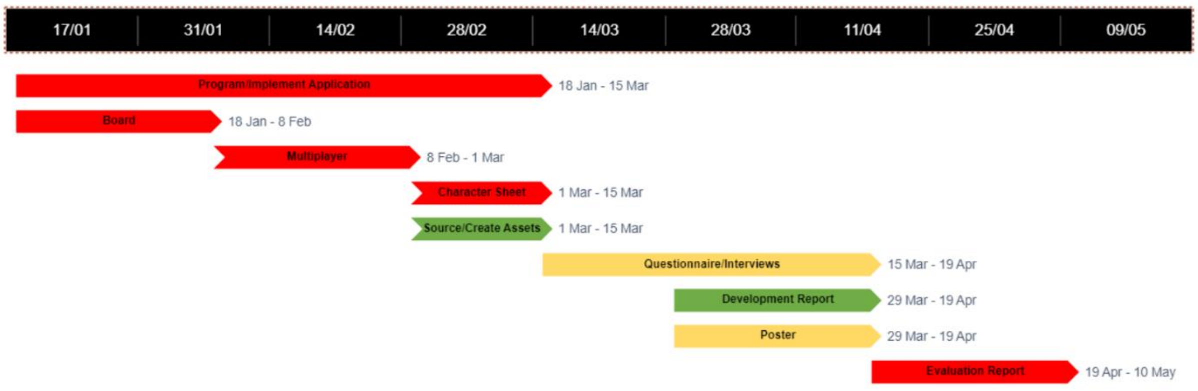
- Mirror Networking – Open Source Networking for Unity. Retrieved 22 April 2021, from https://mirror-networking.com/

- Unity Multiplayer Networking | MLAPI. (2021). Retrieved 22 April 2021, from https://docs-multiplayer.unity3d.com/

- Universal Render Pipeline (formerly LWRP) | Unity. Retrieved 22 April 2021, from https://unity.com/srp/universal-render-pipeline

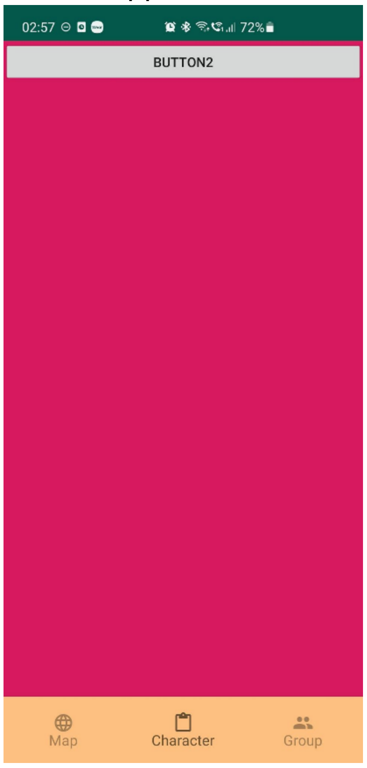
- VR with Andrew. (2018). How to make 2D Chess in Unity - Board (Pt. 1) [Video]. Retrieved from https://www.youtube.com/watch?v=tG1T8J7c7F0&ab_channel=VRwithAndrew

Appendices

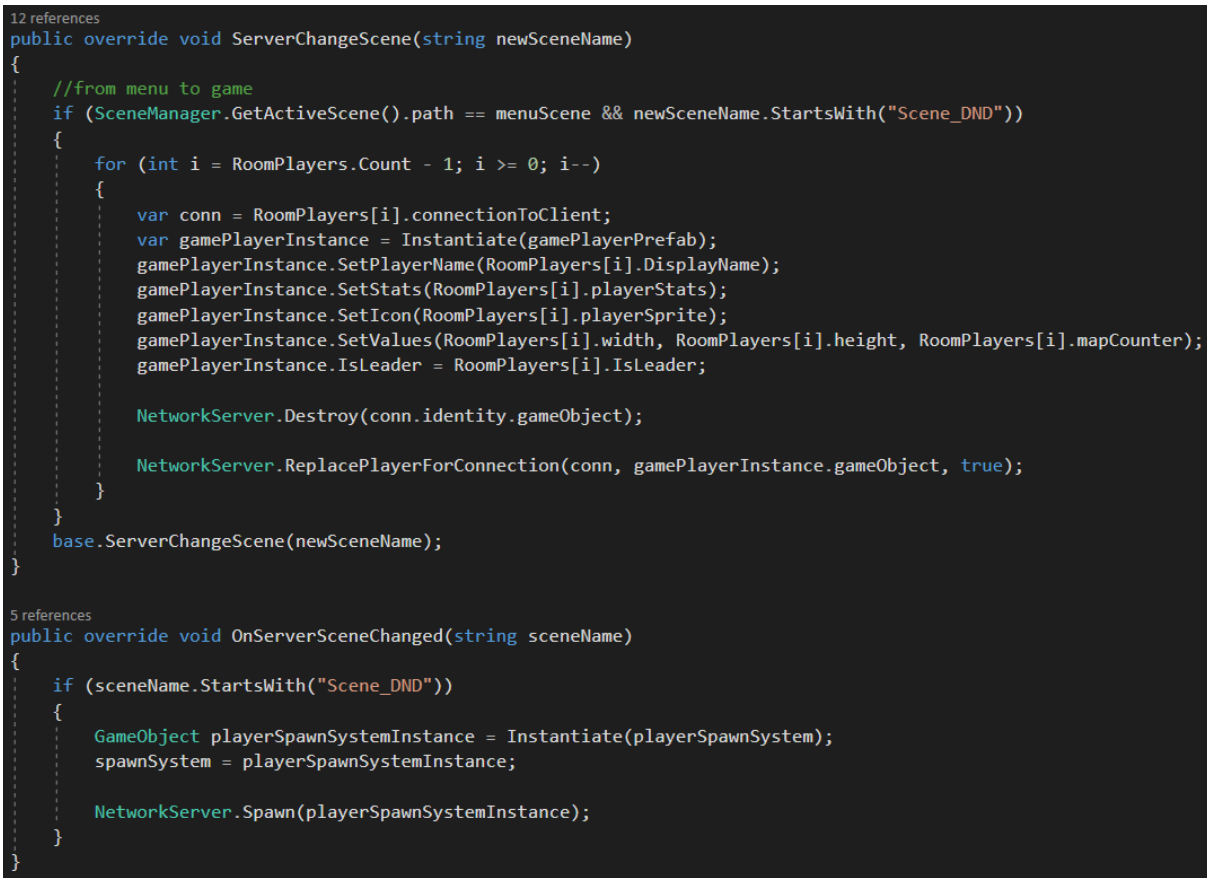


Appendix A

Appendix B

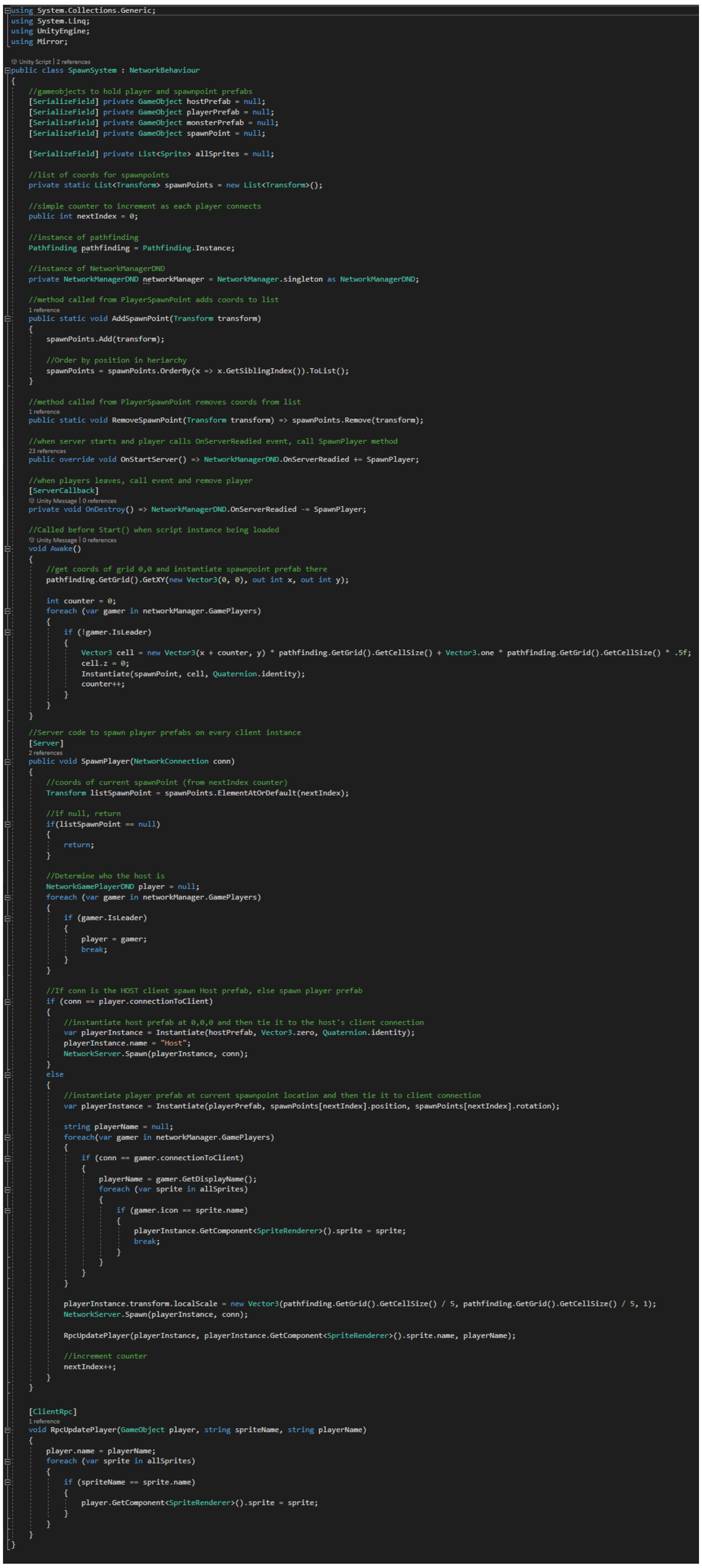


Appendix C



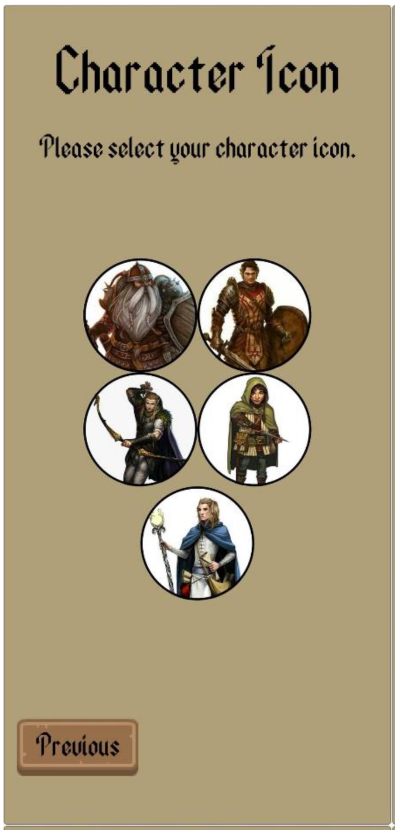
```
12 references
public override void ServerChangeScene(string newSceneName)
{
    //from menu to game
    if (SceneManager.GetActiveScene().path == menuScene && newSceneName.StartsWith("Scene_DND"))
    {
        for (int i = RoomPlayers.Count - 1; i >= 0; i--)
        {
            var conn = RoomPlayers[i].connectionToClient;
            var gamePlayerInstance = Instantiate(gamePlayerPrefab);
            gamePlayerInstance.SetPlayerName(RoomPlayers[i].DisplayName);
            gamePlayerInstance.SetStats(RoomPlayers[i].playerStats);
            gamePlayerInstance.SetIcon(RoomPlayers[i].playerSprite);
            gamePlayerInstance.SetValues(RoomPlayers[i].width, RoomPlayers[i].height, RoomPlayers[i].mapCounter);
            gamePlayerInstance.IsLeader = RoomPlayers[i].IsLeader;

            NetworkServer.Destroy(conn.identity.gameObject);

            NetworkServer.ReplacePlayerForConnection(conn, gamePlayerInstance.gameObject, true);
        }
    }
    base.ServerChangeScene(newSceneName);
}

5 references
public override void OnServerSceneChanged(string sceneName)
{
    if (sceneName.StartsWith("Scene_DND"))
    {
        GameObject playerSpawnSystemInstance = Instantiate(playerSpawnSystem);
        spawnSystem = playerSpawnSystemInstance;

        NetworkServer.Spawn(playerSpawnSystemInstance);
    }
}
```

Appendix D

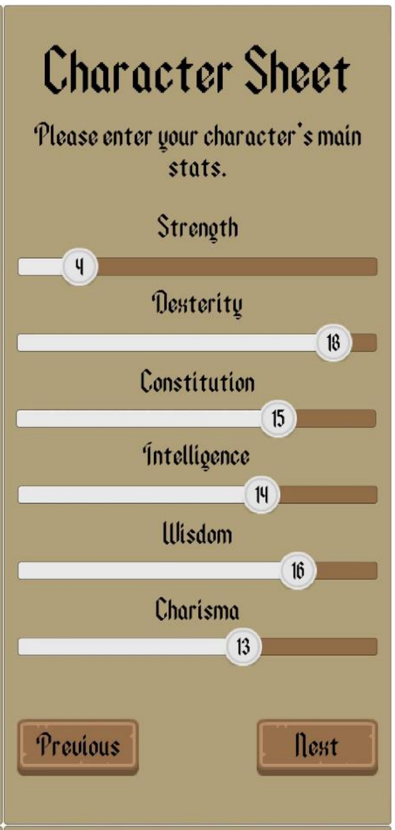


```
using System.Collections.Generic;
using System.Linq;
using UnityEngine;
using Mirror;

Unity Script | 2 references
public class SpawnSystem : NetworkBehaviour
{
    //gameobjects to hold player and spawnpoint prefabs
    [SerializeField] private GameObject hostPrefab = null;
    [SerializeField] private GameObject playerPrefab = null;
    [SerializeField] private GameObject monsterPrefab = null;
    [SerializeField] private GameObject spawnPoint = null;

    [SerializeField] private List<Sprite> allSprites = null;

    //list of coords for spawnpoints
    private static List<Transform> spawnPoints = new List<Transform>();

    //simple counter to increment as each player connects
    public int nextIndex = 0;

    //instance of pathfinding
    Pathfinding pathfinding = Pathfinding.Instance;

    //instance of NetworkManagerDND
    private NetworkManagerDND networkManager = NetworkManager.singleton as NetworkManagerDND;

    //method called from PlayerSpawnPoint adds coords to list
    1 reference
    public static void AddSpawnPoint(Transform transform)
    {
        spawnPoints.Add(transform);

        //Order by position in heriarchy
        spawnPoints = spawnPoints.OrderBy(x => x.GetSiblingIndex()).ToList();
    }

    //method called from PlayerSpawnPoint removes coords from list
    1 reference
    public static void RemoveSpawnPoint(Transform transform) => spawnPoints.Remove(transform);

    //when server starts and player calls OnServerReadied event, call SpawnPlayer method
    23 references
    public override void OnStartServer() => NetworkManagerDND.OnServerReadied += SpawnPlayer;

    //when players leaves, call event and remove player
    [ServerCallback]
    Unity Message | 0 references
    private void OnDestroy() => NetworkManagerDND.OnServerReadied -= SpawnPlayer;

    //Called before Start() when script instance being loaded
    Unity Message | 0 references
    void Awake()
    {
        //get coords of grid 0,0 and instantiate spawnpoint prefab there
        pathfinding.GetGrid().GetXY(new Vector3(0, 0), out int x, out int y);

        int counter = 0;
        foreach (var gamer in networkManager.GamePlayers)
        {
            if (!gamer.IsLeader)
            {
                Vector3 cell = new Vector3(x + counter, y) * pathfinding.GetGrid().GetCellSize() + Vector3.one * pathfinding.GetGrid().GetCellSize() * .5f;
                cell.z = 0;
                Instantiate(spawnPoint, cell, Quaternion.identity);
                counter++;
            }
        }
    }

    //Server code to spawn player prefabs on every client instance
    [Server]
    2 references
    public void SpawnPlayer(NetworkConnection conn)
    {
        //coords of current spawnPoint (from nextIndex counter)
        Transform listSpawnPoint = spawnPoints.ElementAtOrDefault(nextIndex);

        //if null, return
        if(listSpawnPoint == null)
        {
            return;
        }

        //Determine who the host is
        NetworkGamePlayerDND player = null;
        foreach (var gamer in networkManager.GamePlayers)
        {
            if (gamer.IsLeader)
            {
                player = gamer;
                break;
            }
        }

        //If conn is the HOST client spawn Host prefab, else spawn player prefab
        if (conn == player.connectionToClient)
        {
            //instantiate host prefab at 0,0,0 and then tie it to the host's client connection
            var playerInstance = Instantiate(hostPrefab, Vector3.zero, Quaternion.identity);
            playerInstance.name = "Host";
            NetworkServer.Spawn(playerInstance, conn);
        }
        else
        {
            //instantiate player prefab at current spawnpoint location and then tie it to client connection
            var playerInstance = Instantiate(playerPrefab, spawnPoints[nextIndex].position, spawnPoints[nextIndex].rotation);

            string playerName = null;
            foreach(var gamer in networkManager.GamePlayers)
            {
                if (conn == gamer.connectionToClient)
                {
                    playerName = gamer.GetDisplayName();
                    foreach (var sprite in allSprites)
                    {
                        if (gamer.icon == sprite.name)
                        {
                            playerInstance.GetComponent<SpriteRenderer>().sprite = sprite;
                            break;
                        }
                    }
                }
            }

            playerInstance.transform.localScale = new Vector3(pathfinding.GetGrid().GetCellSize() / 5, pathfinding.GetGrid().GetCellSize() / 5, 1);
            NetworkServer.Spawn(playerInstance, conn);

            RpcUpdatePlayer(playerInstance, playerInstance.GetComponent<SpriteRenderer>().sprite.name, playerName);

            //increment counter
            nextIndex++;
        }
    }

    [ClientRpc]
    1 reference
    void RpcUpdatePlayer(GameObject player, string spriteName, string playerName)
    {
        player.name = playerName;
        foreach (var sprite in allSprites)
        {
            if (spriteName == sprite.name)
            {
                player.GetComponent<SpriteRenderer>().sprite = sprite;
            }
        }
    }
}
```

Appendix E


Character Icon

Please select your character icon.


Character Sheet

Please enter your character's main stats.

Strength 4

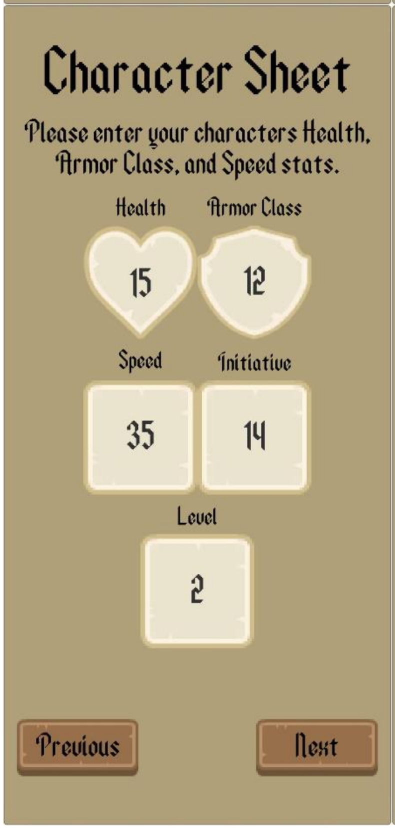
Dexterity 18

Constitution 15

Intelligence 14

Wisdom 16

Charisma 13


Character Sheet

Please enter your characters Health, Armor Class, and Speed stats.

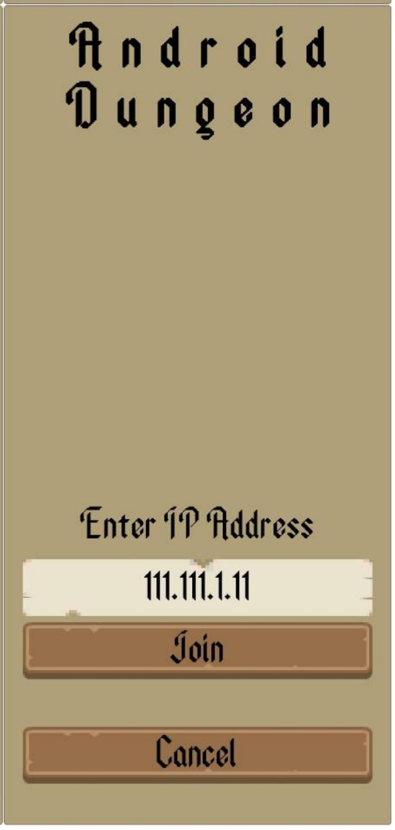
Health 15

Armor Class 12

Speed 35

Initiative 14

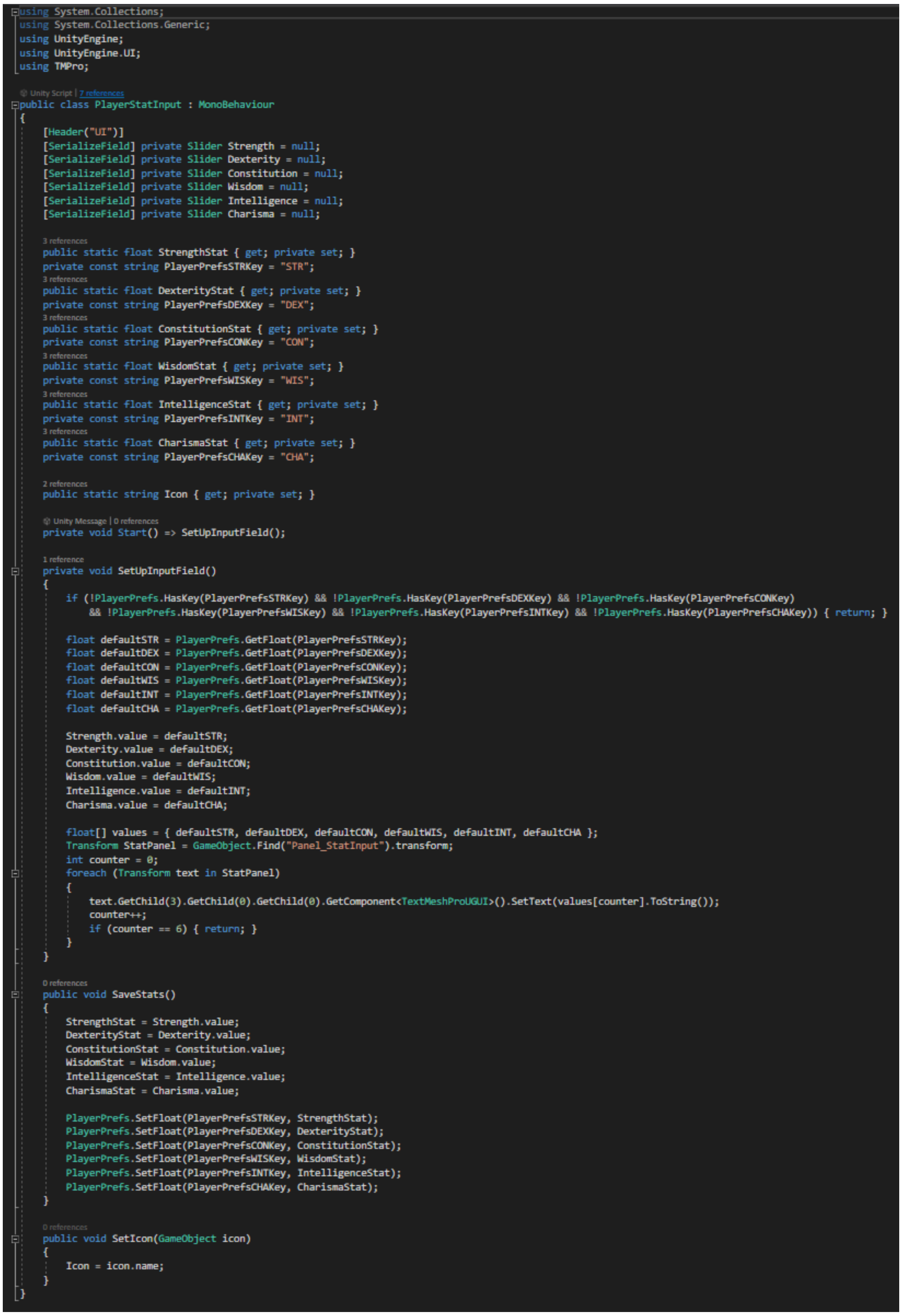
Level 2


Android Dungeon

Enter IP Address

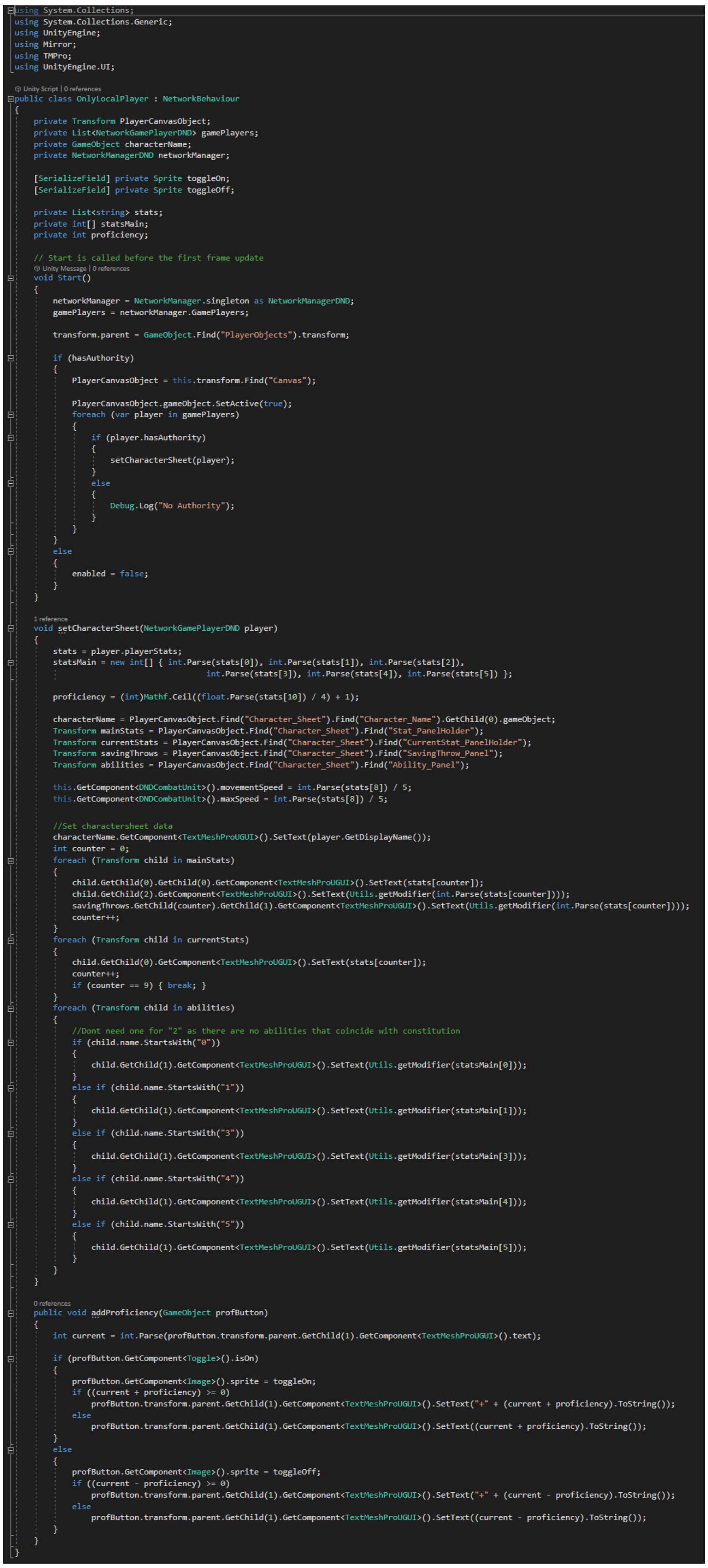
111.111.1.11

Appendix F



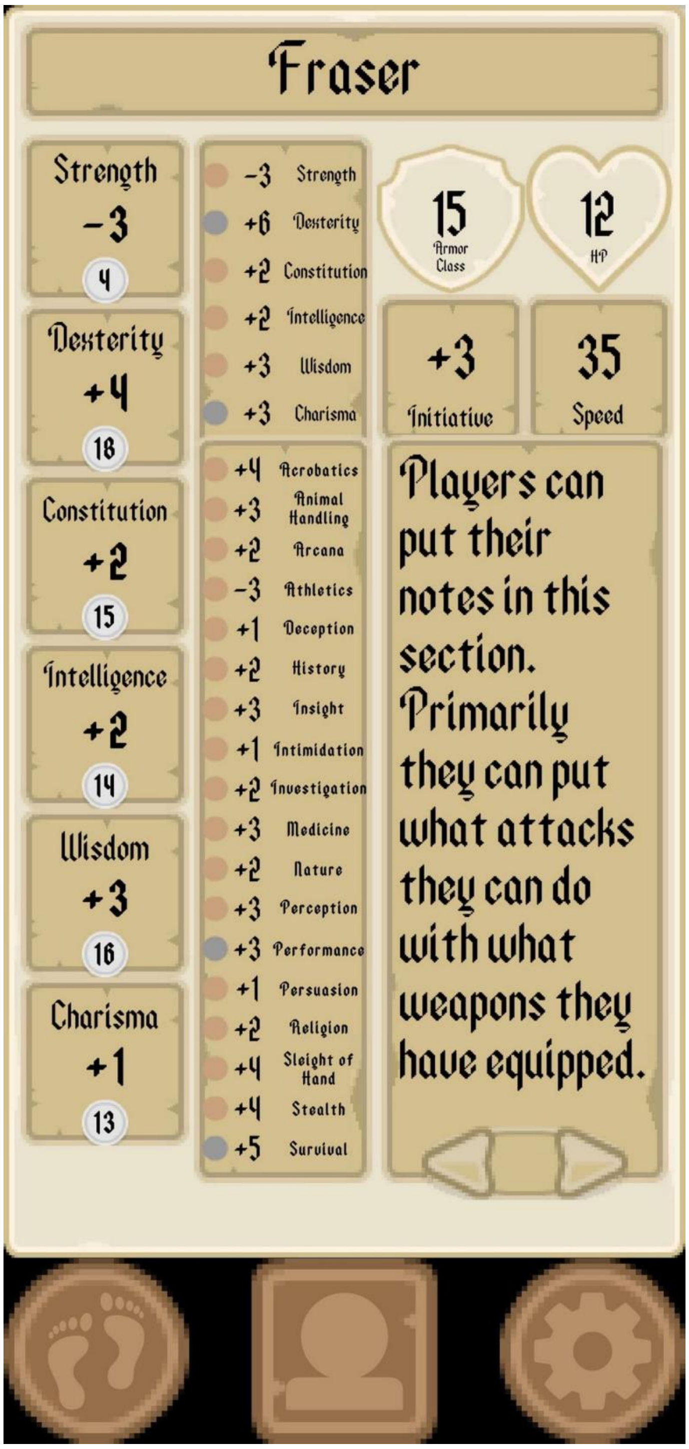
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

Unity Script | 7 references
public class PlayerStatInput : MonoBehaviour
{
    [Header("UI")]
    [SerializeField] private Slider Strength = null;
    [SerializeField] private Slider Dexterity = null;
    [SerializeField] private Slider Constitution = null;
    [SerializeField] private Slider Wisdom = null;
    [SerializeField] private Slider Intelligence = null;
    [SerializeField] private Slider Charisma = null;

    3 references
    public static float StrengthStat { get; private set; }
    private const string PlayerPrefsSTRKey = "STR";
    3 references
    public static float DexterityStat { get; private set; }
    private const string PlayerPrefsDEXKey = "DEX";
    3 references
    public static float ConstitutionStat { get; private set; }
    private const string PlayerPrefsCONKey = "CON";
    3 references
    public static float WisdomStat { get; private set; }
    private const string PlayerPrefsWISKey = "WIS";
    3 references
    public static float IntelligenceStat { get; private set; }
    private const string PlayerPrefsINTKey = "INT";
    3 references
    public static float CharismaStat { get; private set; }
    private const string PlayerPrefsCHAKey = "CHA";

    2 references
    public static string Icon { get; private set; }

    Unity Message | 0 references
    private void Start() => SetUpInputField();

    1 reference
    private void SetUpInputField()
    {
        if (!PlayerPrefs.HasKey(PlayerPrefsSTRKey) && !PlayerPrefs.HasKey(PlayerPrefsDEXKey) && !PlayerPrefs.HasKey(PlayerPrefsCONKey)
            && !PlayerPrefs.HasKey(PlayerPrefsWISKey) && !PlayerPrefs.HasKey(PlayerPrefsINTKey) && !PlayerPrefs.HasKey(PlayerPrefsCHAKey)) { return; }

        float defaultSTR = PlayerPrefs.GetFloat(PlayerPrefsSTRKey);
        float defaultDEX = PlayerPrefs.GetFloat(PlayerPrefsDEXKey);
        float defaultCON = PlayerPrefs.GetFloat(PlayerPrefsCONKey);
        float defaultWIS = PlayerPrefs.GetFloat(PlayerPrefsWISKey);
        float defaultINT = PlayerPrefs.GetFloat(PlayerPrefsINTKey);
        float defaultCHA = PlayerPrefs.GetFloat(PlayerPrefsCHAKey);

        Strength.value = defaultSTR;
        Dexterity.value = defaultDEX;
        Constitution.value = defaultCON;
        Wisdom.value = defaultWIS;
        Intelligence.value = defaultINT;
        Charisma.value = defaultCHA;

        float[] values = { defaultSTR, defaultDEX, defaultCON, defaultWIS, defaultINT, defaultCHA };
        Transform StatPanel = GameObject.Find("Panel_StatInput").transform;
        int counter = 0;
        foreach (Transform text in StatPanel)
        {
            text.GetChild(3).GetChild(0).GetChild(0).GetComponent<TextMeshProUGUI>().SetText(values[counter].ToString());
            counter++;
            if (counter == 6) { return; }
        }
    }

    0 references
    public void SaveStats()
    {
        StrengthStat = Strength.value;
        DexterityStat = Dexterity.value;
        ConstitutionStat = Constitution.value;
        WisdomStat = Wisdom.value;
        IntelligenceStat = Intelligence.value;
        CharismaStat = Charisma.value;

        PlayerPrefs.SetFloat(PlayerPrefsSTRKey, StrengthStat);
        PlayerPrefs.SetFloat(PlayerPrefsDEXKey, DexterityStat);
        PlayerPrefs.SetFloat(PlayerPrefsCONKey, ConstitutionStat);
        PlayerPrefs.SetFloat(PlayerPrefsWISKey, WisdomStat);
        PlayerPrefs.SetFloat(PlayerPrefsINTKey, IntelligenceStat);
        PlayerPrefs.SetFloat(PlayerPrefsCHAKey, CharismaStat);
    }

    0 references
    public void SetIcon(GameObject icon)
    {
        Icon = icon.name;
    }
}
```

Appendix G.1



```csharp
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Mirror;
using TMPro;
using UnityEngine.UI;

Unity Script | 0 references
public class OnlyLocalPlayer : NetworkBehaviour
{
    private Transform PlayerCanvasObject;
    private List<NetworkGamePlayerDND> gamePlayers;
    private GameObject characterName;
    private NetworkManagerDND networkManager;

    [SerializeField] private Sprite toggleOn;
    [SerializeField] private Sprite toggleOff;

    private List<string> stats;
    private int[] statsMain;
    private int proficiency;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        networkManager = NetworkManager.singleton as NetworkManagerDND;
        gamePlayers = networkManager.GamePlayers;

        transform.parent = GameObject.Find("PlayerObjects").transform;

        if (hasAuthority)
        {
            PlayerCanvasObject = this.transform.Find("Canvas");

            PlayerCanvasObject.gameObject.SetActive(true);
            foreach (var player in gamePlayers)
            {
                if (player.hasAuthority)
                {
                    setCharacterSheet(player);
                }
                else
                {
                    Debug.Log("No Authority");
                }
            }
        }
        else
        {
            enabled = false;
        }
    }

    1 reference
    void setCharacterSheet(NetworkGamePlayerDND player)
    {
        stats = player.playerStats;
        statsMain = new int[] { int.Parse(stats[0]), int.Parse(stats[1]), int.Parse(stats[2]),
                                int.Parse(stats[3]), int.Parse(stats[4]), int.Parse(stats[5]) };

        proficiency = (int)Mathf.Ceil((float.Parse(stats[10]) / 4) + 1);

        characterName = PlayerCanvasObject.Find("Character_Sheet").Find("Character_Name").GetChild(0).gameObject;
        Transform mainStats = PlayerCanvasObject.Find("Character_Sheet").Find("Stat_PanelHolder");
        Transform currentStats = PlayerCanvasObject.Find("Character_Sheet").Find("CurrentStat_PanelHolder");
        Transform savingThrows = PlayerCanvasObject.Find("Character_Sheet").Find("SavingThrow_Panel");
        Transform abilities = PlayerCanvasObject.Find("Character_Sheet").Find("Ability_Panel");

        this.GetComponent<DNDCombatUnit>().movementSpeed = int.Parse(stats[8]) / 5;
        this.GetComponent<DNDCombatUnit>().maxSpeed = int.Parse(stats[8]) / 5;


        //Set charactersheet data
        characterName.GetComponent<TextMeshProUGUI>().SetText(player.GetDisplayName());
        int counter = 0;
        foreach (Transform child in mainStats)
        {
            child.GetChild(0).GetChild(0).GetComponent<TextMeshProUGUI>().SetText(stats[counter]);
            child.GetChild(2).GetComponent<TextMeshProUGUI>().SetText(Utils.getModifier(int.Parse(stats[counter])));
            savingThrows.GetChild(counter).GetChild(1).GetComponent<TextMeshProUGUI>().SetText(Utils.getModifier(int.Parse(stats[counter])));
            counter++;
        }
        foreach (Transform child in currentStats)
        {
            child.GetChild(0).GetComponent<TextMeshProUGUI>().SetText(stats[counter]);
            counter++;
            if (counter == 9) { break; }
        }
        foreach (Transform child in abilities)
        {
            //Dont need one for "2" as there are no abilities that coincide with constitution
            if (child.name.StartsWith("0"))
            {
                child.GetChild(1).GetComponent<TextMeshProUGUI>().SetText(Utils.getModifier(statsMain[0]));
            }
            else if (child.name.StartsWith("1"))
            {
                child.GetChild(1).GetComponent<TextMeshProUGUI>().SetText(Utils.getModifier(statsMain[1]));
            }
            else if (child.name.StartsWith("3"))
            {
                child.GetChild(1).GetComponent<TextMeshProUGUI>().SetText(Utils.getModifier(statsMain[3]));
            }
            else if (child.name.StartsWith("4"))
            {
                child.GetChild(1).GetComponent<TextMeshProUGUI>().SetText(Utils.getModifier(statsMain[4]));
            }
            else if (child.name.StartsWith("5"))
            {
                child.GetChild(1).GetComponent<TextMeshProUGUI>().SetText(Utils.getModifier(statsMain[5]));
            }
        }
    }

    0 references
    public void addProficiency(GameObject profButton)
    {
        int current = int.Parse(profButton.transform.parent.GetChild(1).GetComponent<TextMeshProUGUI>().text);

        if (profButton.GetComponent<Toggle>().isOn)
        {
            profButton.GetComponent<Image>().sprite = toggleOn;
            if ((current + proficiency) >= 0)
                profButton.transform.parent.GetChild(1).GetComponent<TextMeshProUGUI>().SetText("+" + (current + proficiency).ToString());
            else
                profButton.transform.parent.GetChild(1).GetComponent<TextMeshProUGUI>().SetText((current + proficiency).ToString());
        }
        else
        {
            profButton.GetComponent<Image>().sprite = toggleOff;
            if ((current - proficiency) >= 0)
                profButton.transform.parent.GetChild(1).GetComponent<TextMeshProUGUI>().SetText("+" + (current - proficiency).ToString());
            else
                profButton.transform.parent.GetChild(1).GetComponent<TextMeshProUGUI>().SetText((current - proficiency).ToString());
        }
    }
}
```

Appendix G.2



Fraser

Strength	-3	4
Dexterity	+4	18
Constitution	+2	15
Intelligence	+2	14
Wisdom	+3	16
Charisma	+1	13

○	-3	Strength
●	+6	Dexterity
○	+2	Constitution
○	+2	Intelligence
○	+3	Wisdom
●	+3	Charisma

○	+4	Acrobatics
○	+3	Animal Handling
○	+2	Arcana
○	-3	Athletics
○	+1	Deception
○	+2	History
○	+3	Insight
○	+1	Intimidation
○	+2	Investigation
○	+3	Medicine
○	+2	Nature
○	+3	Perception
●	+3	Performance
○	+1	Persuasion
○	+2	Religion
○	+4	Sleight of Hand
○	+4	Stealth
●	+5	Survival

15 Armor Class

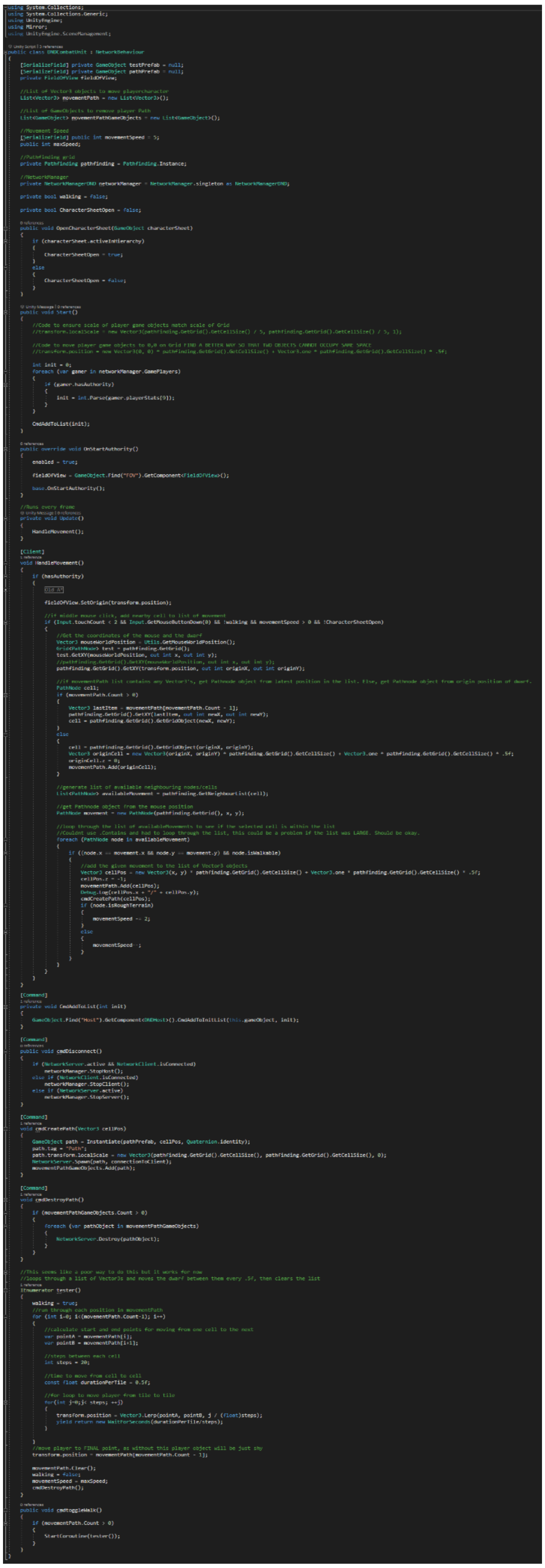
12 HP

+3 Initiative

35 Speed

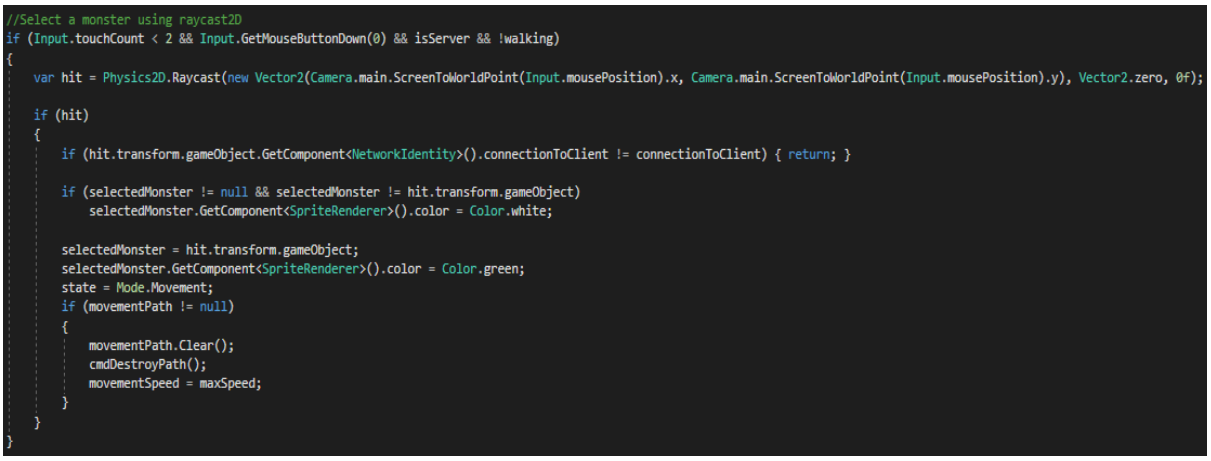
Players can put their notes in this section. Primarily they can put what attacks they can do with what weapons they have equipped.

Appendix H



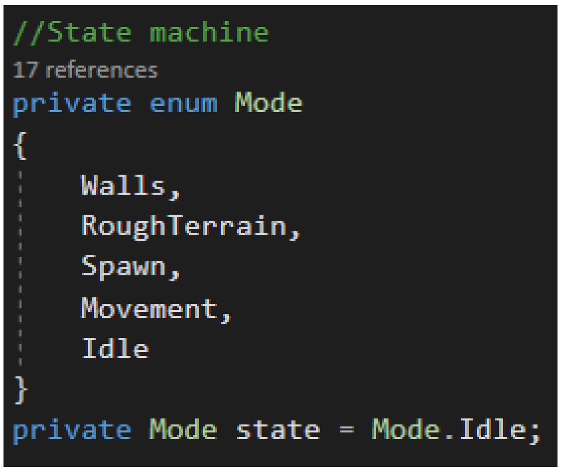
```csharp
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Mirror;
using UnityEngine.SceneManagement;

Unity Script | 3 references
public class DNDCombatUnit : NetworkBehaviour
{
    [SerializeField] private GameObject testPrefab = null;
    [SerializeField] private GameObject pathPrefab = null;
    private FieldOfView fieldOfView;

    //List of Vector3 objects to move playercharacter
    List<Vector3> movementPath = new List<Vector3>();

    //List of GameObjects to remove player Path
    List<GameObject> movementPathGameObjects = new List<GameObject>();

    //Movement Speed
    [SerializeField] public int movementSpeed = 5;
    public int maxSpeed;

    //Pathfinding grid
    private Pathfinding pathfinding = Pathfinding.Instance;

    //NetworkManager
    private NetworkManagerDND networkManager = NetworkManager.singleton as NetworkManagerDND;

    private bool walking = false;

    private bool CharacterSheetOpen = false;

    0 references
    public void OpenCharacterSheet(GameObject characterSheet)
    {
        if (characterSheet.activeInHierarchy)
        {
            CharacterSheetOpen = true;
        }
        else
        {
            CharacterSheetOpen = false;
        }
    }

    Unity Message | 0 references
    public void Start()
    {
        //Code to ensure scale of player game objects match scale of Grid
        //transform.localScale = new Vector3(pathfinding.GetGrid().GetCellSize() / 5, pathfinding.GetGrid().GetCellSize() / 5, 1);

        //Code to move player game objects to 0,0 on Grid FIND A BETTER WAY SO THAT TWO OBJECTS CANNOT OCCUPY SAME SPACE
        //transform.position = new Vector3(0, 0) * pathfinding.GetGrid().GetCellSize() + Vector3.one * pathfinding.GetGrid().GetCellSize() * .5f;

        int init = 0;
        foreach (var gamer in networkManager.GamePlayers)
        {
            if (gamer.hasAuthority)
            {
                init = int.Parse(gamer.playerStats[9]);
            }
        }

        CmdAddToList(init);
    }

    0 references
    public override void OnStartAuthority()
    {
        enabled = true;

        fieldOfView = GameObject.Find("FOV").GetComponent<FieldOfView>();

        base.OnStartAuthority();
    }

    //Runs every frame
    Unity Message | 0 references
    private void Update()
    {
        HandleMovement();
    }

    [Client]
    1 reference
    void HandleMovement()
    {
        if (hasAuthority)
        {
            Old A*

            fieldOfView.SetOrigin(transform.position);

            //if middle mouse click, add nearby cell to list of movement
            if (Input.touchCount < 2 && Input.GetMouseButtonDown(0) && !walking && movementSpeed > 0 && !CharacterSheetOpen)
            {
                //Get the coordinates of the mouse and the dwarf
                Vector3 mouseWorldPosition = Utils.GetMouseWorldPosition();
                Grid<PathNode> test = pathfinding.GetGrid();
                test.GetXY(mouseWorldPosition, out int x, out int y);
                //pathfinding.GetGrid().GetXY(mouseWorldPosition, out int x, out int y);
                pathfinding.GetGrid().GetXY(transform.position, out int originX, out int originY);

                //if movementPath list contains any Vector3's, get Pathnode object from latest position in the list. Else, get Pathnode object from origin position of dwarf.
                PathNode cell;
                if (movementPath.Count > 0)
                {
                    Vector3 lastItem = movementPath[movementPath.Count - 1];
                    pathfinding.GetGrid().GetXY(lastItem, out int newX, out int newY);
                    cell = pathfinding.GetGrid().GetGridObject(newX, newY);
                }
                else
                {
                    cell = pathfinding.GetGrid().GetGridObject(originX, originY);
                    Vector3 originCell = new Vector3(originX, originY) * pathfinding.GetGrid().GetCellSize() + Vector3.one * pathfinding.GetGrid().GetCellSize() * .5f;
                    originCell.z = 0;
                    movementPath.Add(originCell);
                }

                //generate list of available neighbouring nodes/cells
                List<PathNode> availableMovement = pathfinding.GetNeighbourList(cell);

                //get Pathnode object from the mouse position
                PathNode movement = new PathNode(pathfinding.GetGrid(), x, y);

                //loop through the list of availableMovements to see if the selected cell is within the list
                //Couldnt use .Contains and had to loop through the list, this could be a problem if the list was LARGE. Should be okay.
                foreach (PathNode node in availableMovement)
                {
                    if ((node.x == movement.x && node.y == movement.y) && node.isWalkable)
                    {
                        //add the given movement to the list of Vector3 objects
                        Vector3 cellPos = new Vector3(x, y) * pathfinding.GetGrid().GetCellSize() + Vector3.one * pathfinding.GetGrid().GetCellSize() * .5f;
                        cellPos.z = -1;
                        movementPath.Add(cellPos);
                        Debug.Log(cellPos.x + "/" + cellPos.y);
                        cmdCreatePath(cellPos);
                        if (node.isRoughTerrain)
                        {
                            movementSpeed -= 2;
                        }
                        else
                        {
                            movementSpeed--;
                        }
                    }
                }
            }
        }
    }

    [Command]
    1 reference
    private void CmdAddToList(int init)
    {
        GameObject.Find("Host").GetComponent<DNDHost>().CmdAddToInitList(this.gameObject, init);
    }

    [Command]
    0 references
    public void cmdDisconnect()
    {
        if (NetworkServer.active && NetworkClient.isConnected)
            networkManager.StopHost();
        else if (NetworkClient.isConnected)
            networkManager.StopClient();
        else if (NetworkServer.active)
            networkManager.StopServer();
    }

    [Command]
    1 reference
    void cmdCreatePath(Vector3 cellPos)
    {
        GameObject path = Instantiate(pathPrefab, cellPos, Quaternion.identity);
        path.tag = "Path";
        path.transform.localScale = new Vector3(pathfinding.GetGrid().GetCellSize(), pathfinding.GetGrid().GetCellSize(), 0);
        NetworkServer.Spawn(path, connectionToClient);
        movementPathGameObjects.Add(path);
    }

    [Command]
    1 reference
    void cmdDestroyPath()
    {
        if (movementPathGameObjects.Count > 0)
        {
            foreach (var pathObject in movementPathGameObjects)
            {
                NetworkServer.Destroy(pathObject);
            }
        }
    }

    //This seems like a poor way to do this but it works for now
    //loops through a list of Vector3s and moves the dwarf between them every .5f, then clears the list
    1 reference
    IEnumerator tester()
    {
        walking = true;
        //run through each position in movementPath
        for (int i=0; i<(movementPath.Count-1); i++)
        {
            //calculate start and end points for moving from one cell to the next
            var pointA = movementPath[i];
            var pointB = movementPath[i+1];

            //steps between each cell
            int steps = 20;

            //time to move from cell to cell
            const float durationPerTile = 0.5f;

            //for loop to move player from tile to tile
            for(int j=0;j< steps; ++j)
            {
                transform.position = Vector3.Lerp(pointA, pointB, j / (float)steps);
                yield return new WaitForSeconds(durationPerTile/steps);
            }

        }
        //move player to FINAL point, as without this player object will be just shy
        transform.position = movementPath[movementPath.Count - 1];

        movementPath.Clear();
        walking = false;
        movementSpeed = maxSpeed;
        cmdDestroyPath();
    }

    0 references
    public void cmdtoggleWalk()
    {
        if (movementPath.Count > 0)
        {
            StartCoroutine(tester());
        }
    }
}
```

Appendix I



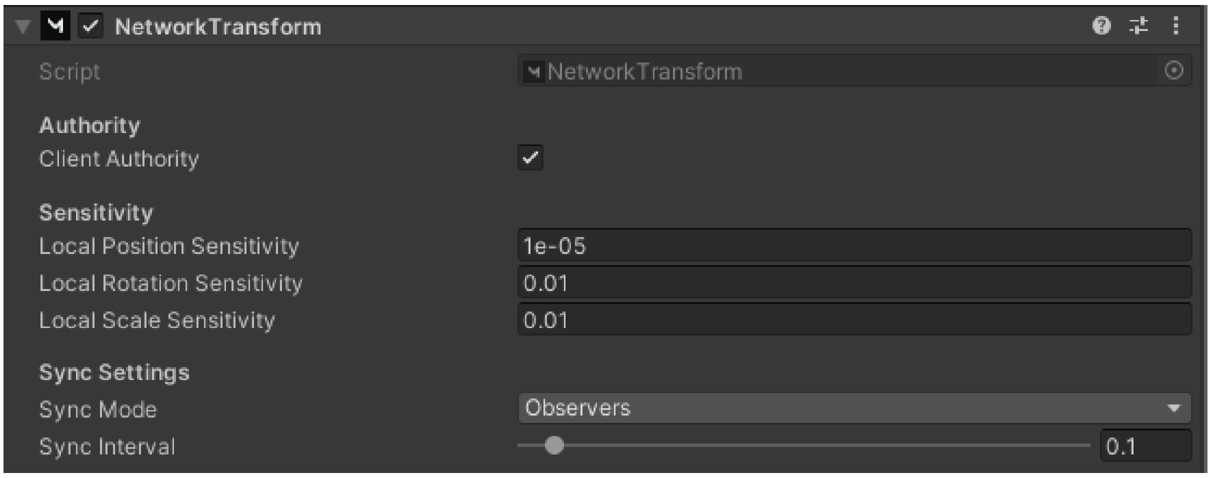
```
//Select a monster using raycast2D
if (Input.touchCount < 2 && Input.GetMouseButtonDown(0) && isServer && !walking)
{
    var hit = Physics2D.Raycast(new Vector2(Camera.main.ScreenToWorldPoint(Input.mousePosition).x, Camera.main.ScreenToWorldPoint(Input.mousePosition).y), Vector2.zero, 0f);

    if (hit)
    {
        if (hit.transform.gameObject.GetComponent<NetworkIdentity>().connectionToClient != connectionToClient) { return; }

        if (selectedMonster != null && selectedMonster != hit.transform.gameObject)
            selectedMonster.GetComponent<SpriteRenderer>().color = Color.white;

        selectedMonster = hit.transform.gameObject;
        selectedMonster.GetComponent<SpriteRenderer>().color = Color.green;
        state = Mode.Movement;
        if (movementPath != null)
        {
            movementPath.Clear();
            cmdDestroyPath();
            movementSpeed = maxSpeed;
        }
    }
}
```

Appendix J



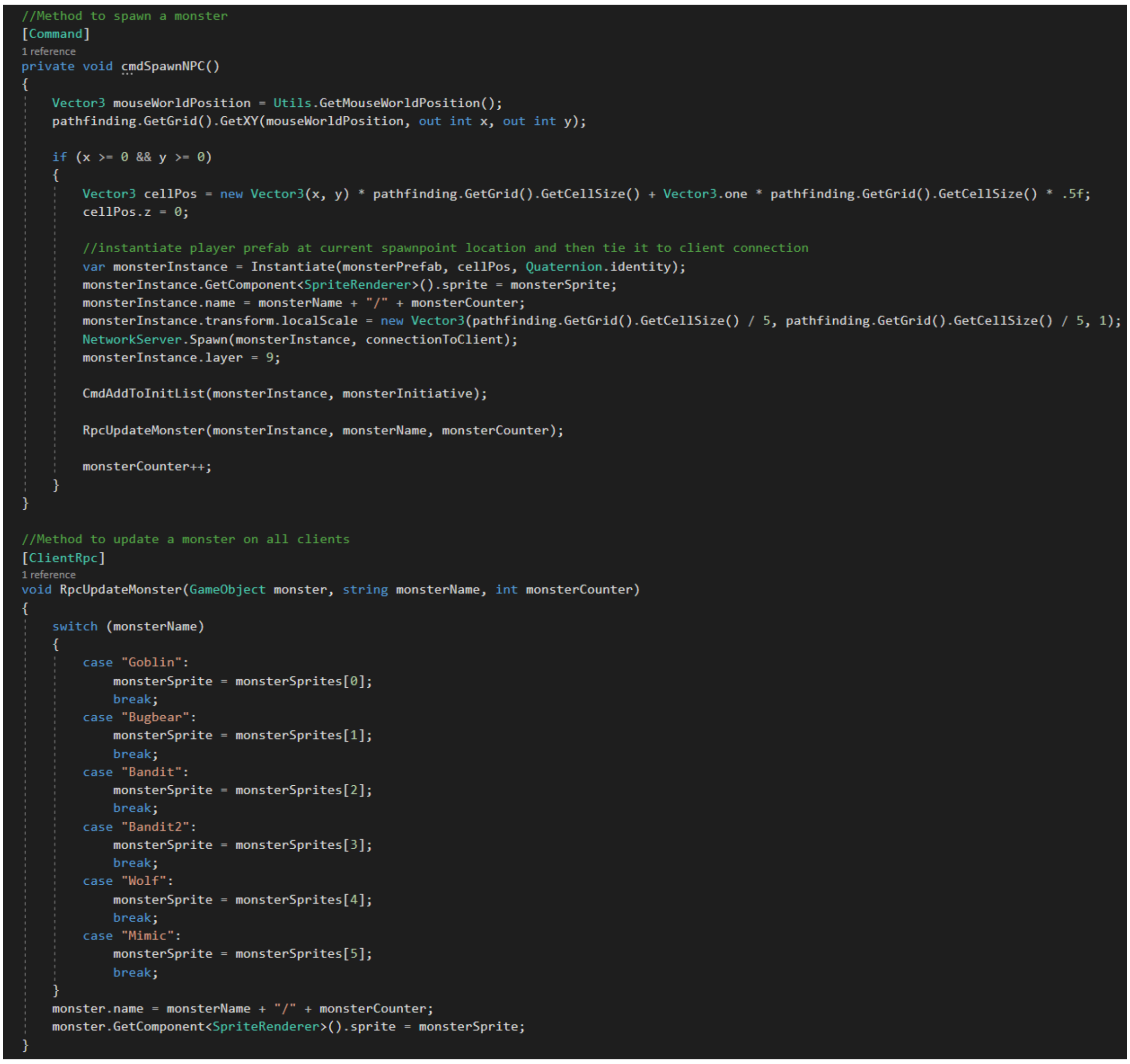
```
//State machine
17 references
private enum Mode
{
    Walls,
    RoughTerrain,
    Spawn,
    Movement,
    Idle
}
private Mode state = Mode.Idle;
```

Appendix K



Script	NetworkTransform
Authority	
Client Authority	✓
Sensitivity	
Local Position Sensitivity	1e-05
Local Rotation Sensitivity	0.01
Local Scale Sensitivity	0.01
Sync Settings	
Sync Mode	Observers
Sync Interval	0.1

Appendix L



```
//Method to spawn a monster
[Command]
1 reference
private void cmdSpawnNPC()
{
    Vector3 mouseWorldPosition = Utils.GetMouseWorldPosition();
    pathfinding.GetGrid().GetXY(mouseWorldPosition, out int x, out int y);

    if (x >= 0 && y >= 0)
    {
        Vector3 cellPos = new Vector3(x, y) * pathfinding.GetGrid().GetCellSize() + Vector3.one * pathfinding.GetGrid().GetCellSize() * .5f;
        cellPos.z = 0;

        //instantiate player prefab at current spawnpoint location and then tie it to client connection
        var monsterInstance = Instantiate(monsterPrefab, cellPos, Quaternion.identity);
        monsterInstance.GetComponent<SpriteRenderer>().sprite = monsterSprite;
        monsterInstance.name = monsterName + "/" + monsterCounter;
        monsterInstance.transform.localScale = new Vector3(pathfinding.GetGrid().GetCellSize() / 5, pathfinding.GetGrid().GetCellSize() / 5, 1);
        NetworkServer.Spawn(monsterInstance, connectionToClient);
        monsterInstance.layer = 9;

        CmdAddToInitList(monsterInstance, monsterInitiative);

        RpcUpdateMonster(monsterInstance, monsterName, monsterCounter);

        monsterCounter++;
    }
}

//Method to update a monster on all clients
[ClientRpc]
1 reference
void RpcUpdateMonster(GameObject monster, string monsterName, int monsterCounter)
{
    switch (monsterName)
    {
        case "Goblin":
            monsterSprite = monsterSprites[0];
            break;
        case "Bugbear":
            monsterSprite = monsterSprites[1];
            break;
        case "Bandit":
            monsterSprite = monsterSprites[2];
            break;
        case "Bandit2":
            monsterSprite = monsterSprites[3];
            break;
        case "Wolf":
            monsterSprite = monsterSprites[4];
            break;
        case "Mimic":
            monsterSprite = monsterSprites[5];
            break;
    }
    monster.name = monsterName + "/" + monsterCounter;
    monster.GetComponent<SpriteRenderer>().sprite = monsterSprite;
}
```