MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY
BHOPAL (M.P.)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MAJOR PROJECT

Multiplayer FPS using UNITY

Session 2017-2018

SUBMITTED BY:

Ayush Bhandari (141112269)
Himanshu Lal (141112077)
Aniket Kumar Singh (141112057)
Tarun Garg (141112052)

UNDER THE GUIDANCE OF:
Prof. Meenu Chawla

MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY
BHOPAL (M.P.)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that Tarun Garg, Ayush Bhandari, Himanshu Lal and Aniket Kumar Singh students of B.Tech 4th Year (Computer Science & Engineering), have successfully completed their project "**Multiplayer FPS using UNITY**" in partial fulfillment of their minor project in Computer Science & Engineering.

PROF. MEENU CHAWLA

(Project Guide)

ACKNOWLEDGEMENT

With due respect, we express our deep sense of gratitude to our respected guide Prof. Bholanath Roy, for his valuable help and guidance. We are thankful for the encouragement that he has given us in completing this project successfully. His rigorous evaluation and constructive criticism was of great assistance. It is imperative for us to mention the fact that this minor project could not have been accomplished without the periodic suggestions and advice of our project coordinator Prof. Bholanath Roy. We are also grateful to our Lab Coordinators for permitting us to utilize all the necessary facilities of the college. Needless to mention is the additional help and support extended by our respected HOD, Dr. R. K. Pateriya, in allowing us to use the departmental laboratories and other services. We are also thankful to all the other faculty, staff members and laboratory attendants of our department for their kind cooperation and help. Last but certainly not the least; we would like to express our deep appreciation towards our family members and batch mates for providing the much needed support and encouragement.

DECLARATION

We, hereby, declare that the following report which is being presented in the Minor Project Documentation entitled “Multiplayer Game using UNITY” is the partial fulfilment of the requirements of the fourth year Major Project in the field of Computer Science And Engineering. It is an authentic documentation of our own original work carried out under the able guidance and the dedicated co-ordination of Prof. Meenu Chawla. The work has been carried out entirely at Maulana Azad National Institute of Technology, Bhopal. The following project and its report, in part or whole, has not been presented or submitted by us for any purpose in any other institute or organization. We, hereby, declare that the facts mentioned above are true to the best of our knowledge. In case of any unlikely discrepancy that may possibly occur, we will be the ones to take responsibility.

Ayush Bhandari (141112269)

Himanshu Lal (141112077)

Aniket Kumar Singh (141112057)

Tarun Garg (141112052)

CONTENTS

LIST OF FIGURES

1. ABSTRACT

The video game industry is still relatively young compared to others, but in such a short period of time, hardware improvements have made leaps and bounds making video games one of the biggest businesses around the world. Today the game industry clocks about a revenue of 108 billion dollar. Generations by generation, video games have been changing. The video game industry is the economic sector involved with the development, marketing and sale of video and computer games.

The aim of this project is to develop a Multiplayer 3D game using the Unity Gaming Engine for modelling and rendering and C# for character and object interaction. Unity is a cross-platform game engine development by Unity Technologies, which is primarily used to develop video games and simulations for computers, consoles and mobile device. The engine targets the Direct3D on Windows.

2. INTRODUCTION

Making games is a creative and technical art form. Video game development is the process of creating a video game. The effort is undertaken by a game developer, who may range from a single person to an international team dispersed across the globe.

Game development, production, or design is a process that starts from an idea or concept. Often the idea is based on a modification of an existing game concept. The game idea may fall within one or several genres. Designers often experiment with different combinations of genres. Different companies have different formal procedures and philosophies regarding game design and development. There is no standardized development method; however, commonalities exist. The genre of game developed here is a **First-Person Shooter (FPS)** game**.**

First-person shooter (FPS) is a video game genre centred around gun and other weapon-based combat in a first-person perspective; that is, the player experiences the action through the eyes of the protagonist. The genre shares common traits with other shooter games, which in turn makes it fall under the heading action game. Since the genre's inception, advanced 3D and pseudo-3D graphics have challenged hardware development, and multiplayer gaming has been integral.

FIG 1. FIRST PERSON SHOOTER PERSPECTIVE

The above game image represents a first person shooter perspective while the one below gives the third person shooter perspective.

FIG 2. THIRD PERSON SHOOTER PERSPECTIVE

3. LITERATURE REVIEW AND SURVEY

3.1 Definition

First-person shooters are a type of three-dimensional shooter game, featuring a first-person point of view with which the player sees the action through the eyes of the player character. They are unlike third-person shooters, in which the player can see (usually from behind) the character they are controlling. The primary design element is combat, mainly involving firearms.

First person-shooter games are also often categorized as being distinct from light gun shooters, a similar genre with a first-person perspective which use light gun peripherals, in contrast to first-person shooters which use conventional input devices for movement. Another difference is that first-person light-gun shooters like Virtua Cop often feature "on-rails" (scripted) movement, whereas first-person shooters like Doom give the player more freedom to roam.

3.2 Game design

Like most shooter games, first-person shooters involve an avatar, one or more ranged weapons, and a varying number of enemies.[8] Because they take place in a 3D environment, these games tend to be somewhat more realistic than 2D shooter games, and have more accurate representations of gravity, lighting, sound and collisions. First-person shooters played on personal computers are most often controlled with a combination of a keyboard and mouse. This system has been claimed as superior to that found in console games, which frequently use two analog sticks: one used for running and sidestepping, the other for looking and aiming. It is common to display the character's hands and weaponry in the main view, with a head-up display showing health, ammunition and location details. Often, it is possible to overlay a map of the surrounding area.

3.3 Combat and power-ups

First-person shooters often focus on action gameplay, with fast-paced and firefights, though some place a greater emphasis on narrative, problem-solving and logic puzzles. In addition to shooting, melee combat may also be used extensively. In some games, melee weapons are especially powerful, a reward for the risk the player must take in maneuvering his character into close proximity to the enemy. In other situations, a melee weapon may be less effective, but necessary as a last resort. "Tactical shooters" are more realistic, and require teamwork and strategy to succeed; the player often commands a squad of characters, which may be controlled by the game or by human teammates.

First-person shooters typically give players a choice of weapons, which have a large impact on how the player will approach the game.[5] Some game designs have realistic models of actual existing or historical weapons, incorporating their rate of fire, magazine size, ammunition amount, recoil and accuracy.

3.4 Level design

First-person shooters may be structurally composed of levels, or use the technique of a continuous narrative in which the game never leaves the first-person perspective. Others feature large sandbox environments, which are not divided into levels and can be explored freely. In first-person shooters, protagonists interact with the environment to varying degrees, from basics such as using doors, to problem solving puzzles based on a variety of interactive objects. In some games, the player can damage the environment, also to varying degrees: one common device is the use of barrels containing explosive material which the player can shoot, destroying them and harming nearby enemies.

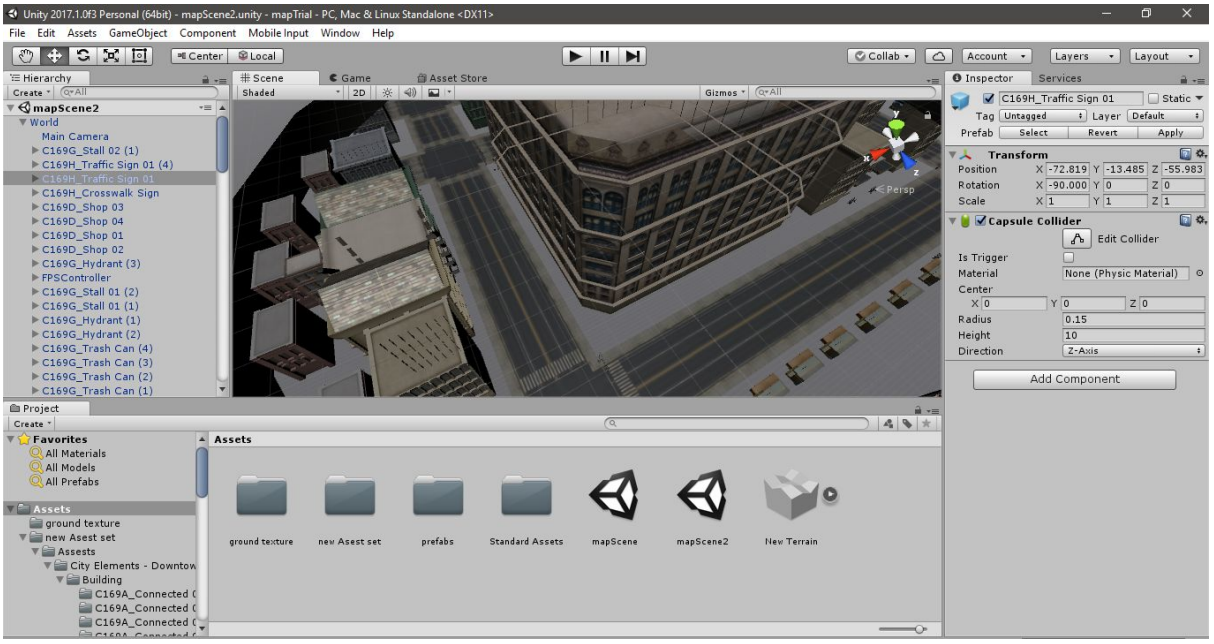

FIG 3. LEVEL DESIGN IN UNITY

3.5 Multiplayer

More recent first-person shooters utilize the internet for multiplayer features, but local area networks were commonly used in early games.

First-person shooters may feature a multiplayer mode, taking place on specialized levels. Some games are designed specifically for multiplayer gaming, and have very limited single player modes in which the player competes against game-controlled characters termed "bots". Massively multiplayer online first-person shooters allow thousands of players to compete at once in a persistent world. Large scale multiplayer games allow multiple squads, with leaders issuing commands and a commander controlling the team's overall strategy. Multiplayer games have a variety of different styles of match.

The classic types are the deathmatch (and its team-based variant) in which players score points by killing other players' characters; and capture the flag, in which teams attempt to penetrate the opposing base, capture a flag and return it to their own base whilst preventing the other team from doing the same.

Other game modes may involve attempting to capture enemy bases or areas of the map, attempting to take hold of an object for as long as possible while evading other players, or deathmatch variations involving

limited lives or in which players fight over a particularly potent power-up. These match types may also be customizable, allowing the players to vary weapons, health and power-ups found on the map, as well as victory criteria. Games may allow players to choose between various classes, each with its own strengths, weaknesses, equipment and roles within a team.

4. METHODOLOGY AND WORK DESCRIPTION

This project will entail development of a game using the Unity engine with the following characteristics-

- A 3D rendered environment with detailed prefabs, structures, terrain designs and landforms etc.
- First person perspective i.e. the whole narrative will be based on the first-person view of the main protagonist/user.
- Real Physics mechanics using Nvidia's PhysX. For example, Gravity, collision detection, rotation & revolution, speed of objects and other such applications are handled by the physics engine within the game.
- Dynamic Lightning and shadow mechanics using shadow maps that bend and generate shadows according to the object that blocks the light path.
- Character modelling using Autodesk Maya for designing the characters and objects for the environment.

4.1 Subtask Involved

- Designing and constructing a 3D rendered environment with detailed prefabs, structures, terrain designs and landforms.
- Constructing the camera models for the first-person perspective for the protagonist.
- Implementing the Physics mechanics in the simulation and blending it into the game engine.
- Designing Lightning, shadow and reflection mechanics using shadow maps which can generate shadows according to the object that blocks the light path.
- Character modelling using Autodesk Maya for designing the characters and objects for the environment.
- Developing a small narrative/story for the game to progress in.

4.2 Constraint

All game assets (art, sound, levels, and so on) are procedurally generated 1 by 1 using model creators which was time consuming nad hectic task.

5. DEVELOPMENT PROCESS

The whole development process is divided into 3 phases.

Phase	Scope	Output
Phase 1	Game Design	Asset development
Phase 2	Coding	Design Implementation
Phase 3	Testing	Working prototype of the game

5.1 Phase 1 - AssetDevelopment

5.1.1 3D Modelling

3D modelling is the use of computers to create images and graphics that look to have three dimensions.
Advanced 3D animation software programs like Autodesk Maya and iClone as well as some coding knowledge are required to be able to create your own 3D models.

The basic process involves connecting sets of points with lines, curved surfaces, and other geometric data to make wireframe models.

Assets for the game are designed in this phase. The process includes

- Conceptualizing Assets

  The motivation for creating a new game asset can come from a variety of sources. It might be that the game needs a specific quest item, or perhaps the game world needs more art direction.The factors involved at this stage of asset creation can vary, depending on *why* the asset is being made.

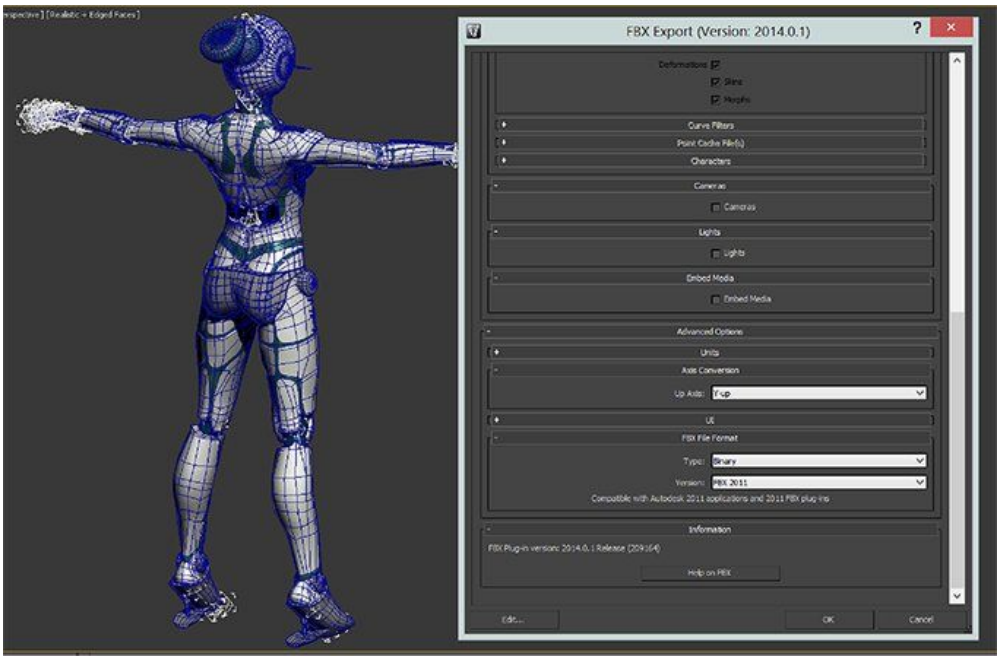

FIG 4. 3D MODEL CREATED IN iCLONE

- 3D Modeling

  Once settled on the look and feel of an asset, some time is spend on breaking down the design into its basic 3D shapes iClone. iColne is used because it suits many of the personal preferences, but alternatively, it can be used on any other 3D modeling app like Blender, 3Ds max, or zBrush,

- Scene Organization

  Once there are more than a few objects in the scene,Then objects are grouped together in the Outliner based on two criteria:

  - If objects categorically belong together, They are put together into one group.
  - If objects need to *move* together, They are put into one group and then adjust the pivot for the group.

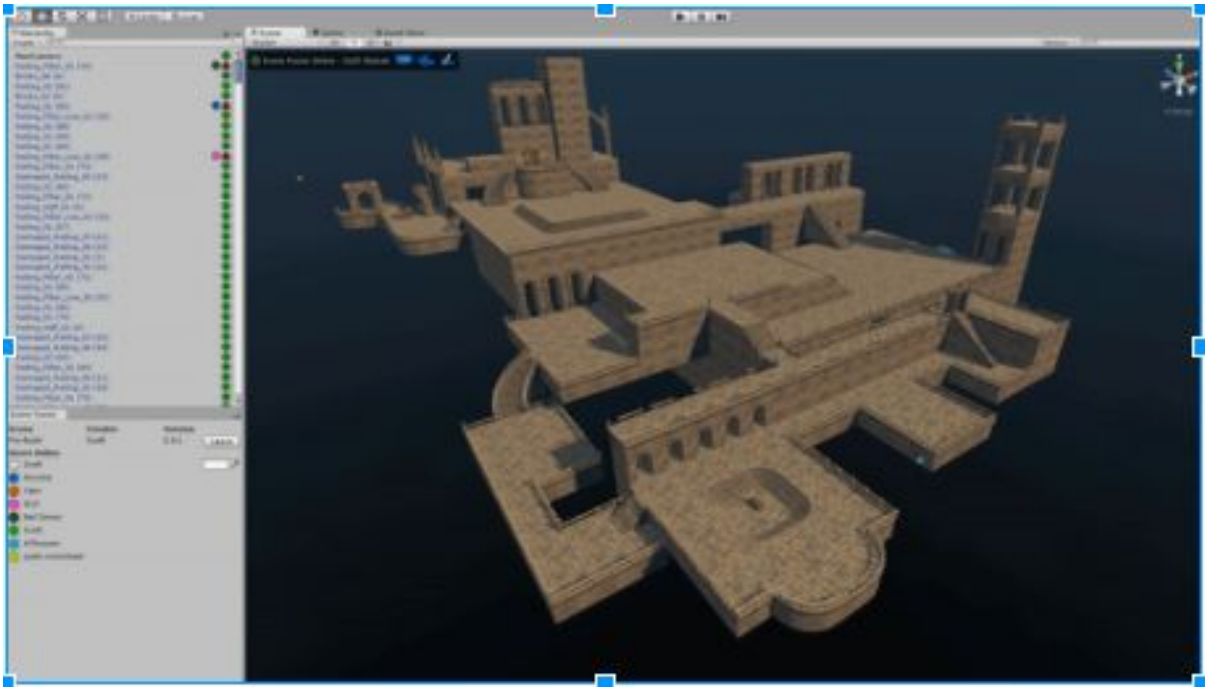

FIG 5. SCENE ORGANIZATION

- UV Mapping

  Once the model and scene is organized, next step comes UV mapping, which is the process of flattening a 3D object into 2D space so that texture maps can be applied.

- Procedural Texturing and Game Engine Import

  In this final step, I'll describe texture mapping is done and asset workflow is outlined such that it can go from a plain 3D model to a prop that's ready for a game.

5.2 Phase 2 - Coding

In the third phase the main logic of the game is implemented. Coding is done in C# on unity platform which helps importing 3d assets in game workflow and dynamically control their behaviour using code.

It includes coding to achieve the following tasks:

- Control player from fps perspective and its movement through the map.
- Physical interaction of the player with in game objects like guns.
- Weapon animation like muzzle flash ,recoil etc.
- Managing multiple client connections to a single host for multiplayer environment.
- Managing healthbars of the players.
- Deciding on their respawn locations. A new opponent must not spawn near his enemy.

5.3 Phase 3- Integration and Testing

This is the last phase of development in which the different modules are combined and their compatibility is tested.
Once successfully connected the fully functional game is tested with various players.

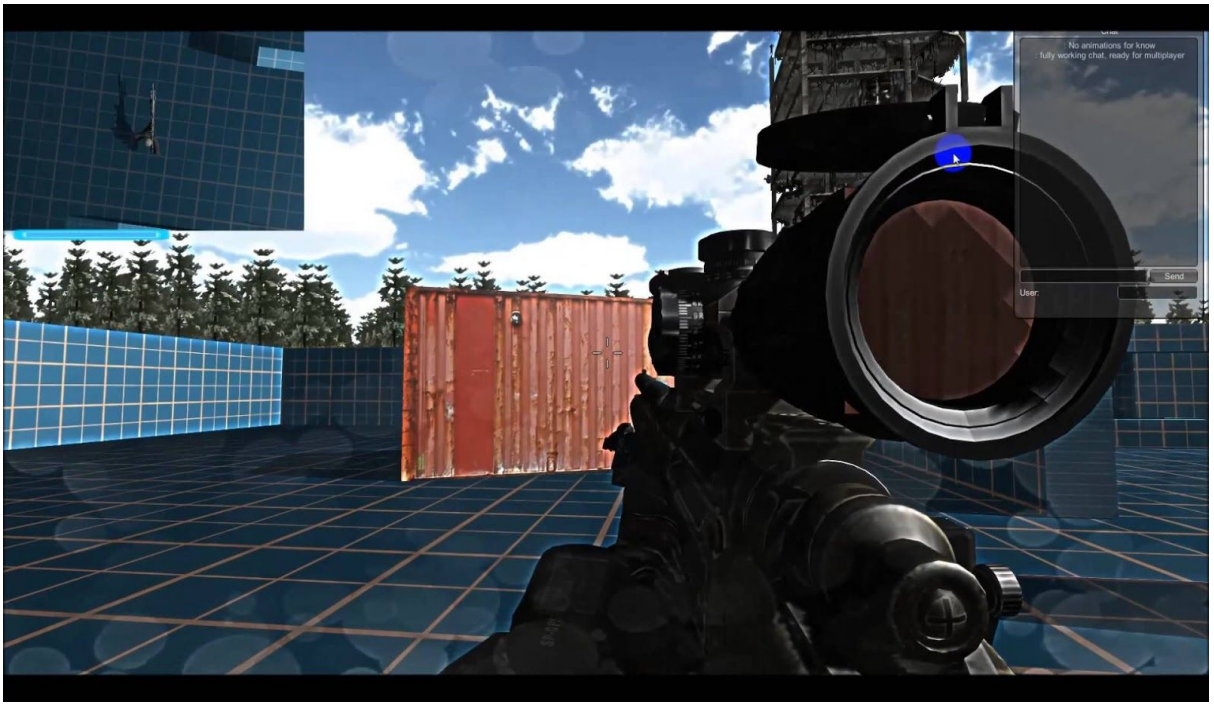

FIG 6. IN GAME FOOTAGE OF AN FPS

6. Softwares Used

Softwares:

- Unity Game Development Editor
- Visual Studio for coding
- Sublime Editor
- iClone Character Creator & 3D Animation
- iClone 3DXchange
- Blender

Libraries Used:

- Unity Gaming Engine
- Physx
- Realvision 3D

7. Future Scope

The Current market for gaming is huge with games and game developers coming out in the market everyday.With technological advancements, the quality of games have also drastically improved ,and for the better. Games are created, not only for entertainment purposes but also for educational purposes. Also with the introduction of Augmented Reality(AR) and Virtual reality(VR) in gaming world, the games are closer to reality than ever.The limit is only our imagination.

This project proposes an interactive way of entertainment and in can be updated to include:

- 3D Virtual Reality Environment
- More Maps options
- Player Customization options
- Weapons Customization options
- Interaction through VR Glasses and Bluetooth Controller

8. References

[1] Unity Game Development Resources.

[2] Udemy online course of " Unity 3D Game Development" by Jose Bortilla

[3] iClone Documentation

[4] 3D Animation Tutorial- www.awn.com/animationworld

[5] Documentation of 3D-Xchange

[6] Other Online resources For Various API's

[7] "https://www,youtube.com"-For Other Tutorials

9. Source Code

9.1 CursorController.cs

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

public class CursorController : MonoBehaviour {

	void Start () {
//		Cursor.lockState = CursorLockMode.Locked;
	}

	// Update is called once per frame
	void Update () {
		ControlCursor ();
	}

	void ControlCursor() {
		if (Input.GetKeyDown (KeyCode.Tab)) {
			if (Cursor.lockState == CursorLockMode.Locked) {
				Cursor.lockState = CursorLockMode.None;
			} else {
				Cursor.lockState = CursorLockMode.Locked;
			}
		}
	}

}
```

9.2 PlayerHealth.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;
```

```
public class PlayerHealth : NetworkBehaviour {

	[SyncVar]
	public float health = 100f;
	public void TakeDamage(float damage) {
		if (!isServer) {
			return;
		}
		health -= damage;
		print ("DAMAGE RECEIVED");
		if (health <= 0f) {

		}
	}
}
```

9.3 NetworkManager.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

public class NetManager : NetworkManager {
	private bool firstPlayerJoined;
	public override void OnServerAddPlayer(NetworkConnection conn,
short playerControllerID) {
		GameObject playerObj = Instantiate (playerPrefab,
Vector3.zero, Quaternion.identity);
		List<Transform>spawnPositions=
NetworkManager.singleton.startPositions;

		if (!firstPlayerJoined) {
			firstPlayerJoined = true;
			playerObj.transform.position = spawnPositions
[0].position;
		} else {
			playerObj.transform.position = spawnPositions
[1].position;
		}
```

```
            NetworkServer.AddPlayerForConnection (conn, playerObj,
playerControllerID);

        }
        void SetPortAndAddress() {
            NetworkManager.singleton.networkAddress = "localhost";
            NetworkManager.singleton.networkPort = 7777;
        }

        public void HostGame() {
            SetPortAndAddress ();
            NetworkManager.singleton.StartHost ();
        }

        public void JoinGame() {
            SetPortAndAddress ();
            NetworkManager.singleton.StartClient ();
        }

}
```

9.4 FPSHandsWeapon.cs

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

public class FPSHandsWeapon : MonoBehaviour {

        public AudioClip shootClip, reloadClip;
        private AudioSource audioManager;
        private GameObject muzzleFlash;

        private Animator anim;

        private string SHOOT = "Shoot";
        private string RELOAD = "Reload";

        void Awake () {
            muzzleFlash = transform.Find ("MuzzleFlash").gameObject;
```

```
        muzzleFlash.SetActive (false);

        audioManager = GetComponent<AudioSource> ();
        anim = GetComponent<Animator> ();
    }

    public void Shoot() {
        if (audioManager.clip != shootClip) {
            audioManager.clip = shootClip;
        }
        audioManager.Play ();

        StartCoroutine (TurnMuzzleFlashOn ());

        anim.SetTrigger (SHOOT);
    }

    IEnumerator TurnMuzzleFlashOn() {
        muzzleFlash.SetActive (true);
        yield return new WaitForSeconds (0.05f);
        muzzleFlash.SetActive (false);
    }

    public void Reload() {
        StartCoroutine (PlayReloadSound ());
        anim.SetTrigger (RELOAD);
    }

    IEnumerator PlayReloadSound() {
        yield return new WaitForSeconds (0.8f);
        if (audioManager.clip != reloadClip) {
            audioManager.clip = reloadClip;
        }
        audioManager.Play ();
    }

}
```

9.5 FPSWeapons.cs

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

public class FPSWeapon : MonoBehaviour {

	private GameObject muzzleFlash;

	void Awake () {
		muzzleFlash = transform.Find ("Muzzle Flash").gameObject;
		muzzleFlash.SetActive (false);
	}

	public void Shoot() {
		StartCoroutine (TurnOnMuzzleFlash ());
	}

	IEnumerator TurnOnMuzzleFlash() {
		muzzleFlash.SetActive (true);
		yield return new WaitForSeconds (0.1f);
		muzzleFlash.SetActive (false);
	}
}
```

9.6 FPSController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

public class FPSController : NetworkBehaviour {

	private Transform firstPerson_View;
	private Transform firstPerson_Camera;

	private Vector3 firstPerson_View_Rotation = Vector3.zero;
```

```
public float walkSpeed = 6.75f;
public float runSpeed = 10f;
public float crouchSpeed = 4f;
public float jumpSpeed = 8f;
public float gravity = 20f;

private float speed;

private bool is_Moving, is_Grounded, is_Crouching;

private float inputX, inputY;
private float inputX_Set, inputY_Set;
private float inputModifyFactor;

private bool limitDiagonalSpeed = true;

private float antiBumpFactor = 0.75f;

private CharacterController charController;
private Vector3 moveDirection = Vector3.zero;

public LayerMask groundLayer;
private float rayDistance;
private float default_ControllerHeight;
private Vector3 default_CamPos;
private float camHeight;

private FPSPlayerAnimations playerAnimation;

[SerializeField]
private WeaponManager weapon_Manager;
private FPSWeapon current_Weapon;

private float fireRate = 15f;
private float nextTimeToFire = 0f;

[SerializeField]
private WeaponManager handsWeapon_Manager;
private FPSHandsWeapon current_Hands_Weapon;

public GameObject playerHolder, weaponsHolder;
public GameObject[] weapons_FPS;
```

```
        private Camera mainCam;
        public FPSMouseLook[] mouseLook;

        private Color[] playerColors = new Color[] {new Color(0, 44, 255,
255),
                new Color(252, 208, 193, 255), new Color(0, 0, 0, 255)};
        public Renderer playerRenderer;

        void Start () {
                firstPerson_View = transform.Find ("FPS View").transform;
                charController = GetComponent<CharacterController> ();
                speed = walkSpeed;
                is_Moving = false;

                rayDistance = charController.height * 0.5f +
charController.radius;
                default_ControllerHeight = charController.height;
                default_CamPos = firstPerson_View.localPosition;

                playerAnimation = GetComponent<FPSPlayerAnimations>
();

                weapon_Manager.weapons [0].SetActive (true);
                current_Weapon = weapon_Manager.weapons
[0].GetComponent<FPSWeapon> ();

                handsWeapon_Manager.weapons [0].SetActive (true);
                current_Hands_Weapon =
handsWeapon_Manager.weapons
[0].GetComponent<FPSHandsWeapon> ();

                if (isLocalPlayer) {
                        playerHolder.layer = LayerMask.NameToLayer
("Player");

                        foreach (Transform child in playerHolder.transform) {
                                child.gameObject.layer =
LayerMask.NameToLayer ("Player");
                        }

                        for (int i = 0; i < weapons_FPS.Length; i++) {
```

```
                        weapons_FPS [i].layer =
LayerMask.NameToLayer ("Player");
                }

                weaponsHolder.layer = LayerMask.NameToLayer
("Enemy");

                foreach (Transform child in weaponsHolder.transform)
{
                        child.gameObject.layer =
LayerMask.NameToLayer ("Enemy");
                }
        }

        if (!isLocalPlayer) {
                playerHolder.layer = LayerMask.NameToLayer
("Enemy");

                foreach (Transform child in playerHolder.transform) {
                        child.gameObject.layer =
LayerMask.NameToLayer ("Enemy");
                }

                for (int i = 0; i < weapons_FPS.Length; i++) {
                        weapons_FPS [i].layer =
LayerMask.NameToLayer ("Enemy");
                }

                weaponsHolder.layer = LayerMask.NameToLayer
("Player");

                foreach (Transform child in weaponsHolder.transform)
{
                        child.gameObject.layer =
LayerMask.NameToLayer ("Player");
                }
        }

        if (!isLocalPlayer) {
                for (int i = 0; i < mouseLook.Length; i++) {
                        mouseLook [i].enabled = false;
                }
```

```
            }

            mainCam = transform.Find ("FPS View").Find ("FPS
Camera").GetComponent<Camera> ();
            mainCam.gameObject.SetActive (false);

            if (!isLocalPlayer) {
                  for (int i = 0; i < playerRenderer.materials.Length; i++) {
                        playerRenderer.materials [i].color = playerColors
[i];
                  }
            }

      }

      public override void OnStartLocalPlayer() {
            tag = "Player";
      }

      void Update () {

            if (isLocalPlayer) {
                  if (!mainCam.gameObject.activeInHierarchy) {
                        mainCam.gameObject.SetActive (true);
                  }
            }

            // IF WE ARE NOT THE LOCAL PLAYER
            // E.G. MEANING WE ARE NOT RUNNING THIS CODE
            // ON OUR OWN COMPUTER
            if (!isLocalPlayer) {
                  // IF THIS IF STATEMENT IS TRUE
                  // AND WE EXECUTE THE return CODE
                  // ALL CODE THAT IS WRITTEN BELOW THE return
                  // WILL NOT BE EXECUTED
                  return;
            }

            PlayerMovement ();
            SelectWeapon ();
      }
```

```
        void PlayerMovement () {
                if (Input.GetKey (KeyCode.W) || Input.GetKey (KeyCode.S)) {
                        if (Input.GetKey (KeyCode.W)) {
                                inputY_Set = 1f;
                        } else {
                                inputY_Set = -1f;
                        }
                } else {
                        inputY_Set = 0f;
                }

                if (Input.GetKey (KeyCode.A) || Input.GetKey (KeyCode.D)) {
                        if (Input.GetKey (KeyCode.A)) {
                                inputX_Set = -1f;
                        } else {
                                inputX_Set = 1f;
                        }
                } else {
                        inputX_Set = 0f;
                }

                inputY = Mathf.Lerp (inputY, inputY_Set, Time.deltaTime *
19f);
                inputX = Mathf.Lerp (inputX, inputX_Set, Time.deltaTime *
19f);

                inputModifyFactor = Mathf.Lerp (inputModifyFactor,
                        (inputY_Set != 0 && inputX_Set != 0 &&
limitDiagonalSpeed) ? 0.75f : 1.0f,
                        Time.deltaTime * 19f);

                firstPerson_View_Rotation = Vector3.Lerp
(firstPerson_View_Rotation,
                        Vector3.zero, Time.deltaTime * 5f);
                firstPerson_View.localEulerAngles =
firstPerson_View_Rotation;

                if (is_Grounded) {

                        // HERE WE ARE GONNA CALL CROUCH AND
SPRINT
                        PlayerCrouchingAndSprinting();
```

```
                moveDirection = new Vector3 (inputX *
inputModifyFactor, -antiBumpFactor,
                        inputY * inputModifyFactor);

                moveDirection = transform.TransformDirection
(moveDirection) * speed;

                // HERE WE ARE GONNA CALL JUMP
                PlayerJump();
        }

        moveDirection.y -= gravity * Time.deltaTime;

        is_Grounded = (charController.Move (moveDirection *
Time.deltaTime) & CollisionFlags.Below) != 0;

        is_Moving = charController.velocity.magnitude > 0.15f;

        HandleAnimations ();

    }

    void PlayerCrouchingAndSprinting() {
        if (Input.GetKeyDown (KeyCode.C)) {

            if (!is_Crouching) {
                is_Crouching = true;
            } else {
                if (CanGetUp ()) {
                    is_Crouching = false;
                }
            }

            StopCoroutine (MoveCameraCrouch());
            StartCoroutine (MoveCameraCrouch());
        }

        if (is_Crouching) {
            speed = crouchSpeed;
        } else {
            if (Input.GetKey (KeyCode.LeftShift)) {
```

```
                speed = runSpeed;
            } else {
                speed = walkSpeed;
            }
        }

        playerAnimation.PlayerCrouch (is_Crouching);

    }

    bool CanGetUp() {
        Ray groundRay = new Ray (transform.position,
transform.up);
        RaycastHit groundHit;

        if (Physics.SphereCast (groundRay, charController.radius +
0.05f,
            out groundHit, rayDistance, groundLayer)) {

            if (Vector3.Distance (transform.position,
groundHit.point) < 2.3f) {
                return false;
            }
        }

        return true;
    }

    IEnumerator MoveCameraCrouch() {
        charController.height = is_Crouching ?
default_ControllerHeight / 1.5f : default_ControllerHeight;
        charController.center = new Vector3 (0f,
charController.height / 2f, 0f);

        camHeight = is_Crouching ? default_CamPos.y / 1.5f :
default_CamPos.y;

        while (Mathf.Abs (camHeight -
firstPerson_View.localPosition.y) > 0.01f) {

            firstPerson_View.localPosition = Vector3.Lerp
(firstPerson_View.localPosition,
```

```
                        new Vector3 (default_CamPos.x, camHeight,
default_CamPos.z),
                        Time.deltaTime * 11f);

                yield return null;
            }
        }

        void PlayerJump() {
            if (Input.GetKeyDown (KeyCode.Space)) {

                if (is_Crouching) {

                    if (CanGetUp ()) {
                        is_Crouching = false;

                        playerAnimation.PlayerCrouch
(is_Crouching);

                        StopCoroutine (MoveCameraCrouch());
                        StartCoroutine (MoveCameraCrouch());
                    }

                } else {
                    moveDirection.y = jumpSpeed;
                }
            }
        }

        void HandleAnimations() {
            playerAnimation.Movement
(charController.velocity.magnitude);
            playerAnimation.PlayerJump (charController.velocity.y);

            if (is_Crouching && charController.velocity.magnitude > 0f) {
                playerAnimation.PlayerCrouchWalk
(charController.velocity.magnitude);
            }

            // SHOOTING
            if(Input.GetMouseButtonDown(0) && Time.time >
nextTimeToFire) {
```

```
                nextTimeToFire = Time.time + 1f / fireRate;

                if (is_Crouching) {
                        playerAnimation.Shoot (false);
                } else {
                        playerAnimation.Shoot (true);
                }

                current_Weapon.Shoot ();
                current_Hands_Weapon.Shoot ();
        }

        if (Input.GetKeyDown (KeyCode.R)) {
                playerAnimation.ReloadGun ();
                current_Hands_Weapon.Reload ();
        }

    }

    void SelectWeapon() {
        if (Input.GetKeyDown (KeyCode.Alpha1)) {

                if (!handsWeapon_Manager.weapons
[0].activeInHierarchy) {
                        for (int i = 0; i <
handsWeapon_Manager.weapons.Length; i++) {
                                handsWeapon_Manager.weapons
[i].SetActive (false);
                        }

                        current_Hands_Weapon = null;

                        handsWeapon_Manager.weapons [0].SetActive
(true);
                        current_Hands_Weapon =
handsWeapon_Manager.weapons
[0].GetComponent<FPSHandsWeapon> ();
                }

                if (!weapon_Manager.weapons [0].activeInHierarchy) {
                        for (int i = 0; i <
weapon_Manager.weapons.Length; i++) {
```

```
                                weapon_Manager.weapons [i].SetActive
(false);
                        }

                        current_Weapon = null;
                        weapon_Manager.weapons [0].SetActive (true);
                        current_Weapon = weapon_Manager.weapons
[0].GetComponent<FPSWeapon> ();

                        playerAnimation.ChangeController (true);
                    }
                }

                if (Input.GetKeyDown (KeyCode.Alpha2)) {

                    if (!handsWeapon_Manager.weapons
[1].activeInHierarchy) {
                        for (int i = 0; i <
handsWeapon_Manager.weapons.Length; i++) {
                            handsWeapon_Manager.weapons
[i].SetActive (false);
                        }

                        current_Hands_Weapon = null;

                        handsWeapon_Manager.weapons [1].SetActive
(true);
                        current_Hands_Weapon =
handsWeapon_Manager.weapons
[1].GetComponent<FPSHandsWeapon> ();
                    }

                    if (!weapon_Manager.weapons [1].activeInHierarchy) {
                        for (int i = 0; i <
weapon_Manager.weapons.Length; i++) {
                            weapon_Manager.weapons [i].SetActive
(false);
                        }

                        current_Weapon = null;
                        weapon_Manager.weapons [1].SetActive (true);
```

```
                        current_Weapon = weapon_Manager.weapons
[1].GetComponent<FPSWeapon> ();

                        playerAnimation.ChangeController (false);
                    }
                }
                if (Input.GetKeyDown (KeyCode.Alpha3)) {

                    if (!handsWeapon_Manager.weapons
[2].activeInHierarchy) {
                        for (int i = 0; i <
handsWeapon_Manager.weapons.Length; i++) {
                            handsWeapon_Manager.weapons
[i].SetActive (false);
                        }

                        current_Hands_Weapon = null;

                        handsWeapon_Manager.weapons [2].SetActive
(true);
                        current_Hands_Weapon =
handsWeapon_Manager.weapons
[2].GetComponent<FPSHandsWeapon> ();
                    }
                    if (!weapon_Manager.weapons [2].activeInHierarchy) {
                        for (int i = 0; i <
weapon_Manager.weapons.Length; i++) {
                            weapon_Manager.weapons [i].SetActive
(false);
                        }
                        current_Weapon = null;
                        weapon_Manager.weapons [2].SetActive (true);
                        current_Weapon = weapon_Manager.weapons
[2].GetComponent<FPSWeapon> ();

                        playerAnimation.ChangeController (false);
                    }
                }
            }
}
```