Introduction to Games Programming

Adam Al-Bsoul

Student ID: S1718672

Preface and scenes

A Main scene was created. A level roughly modelled after the Counter Strike map “de_dust2” was created using cubes and planes.

An FPSController was used as the player, and was tagged “Player”. 3 weapons were imported (a pistol, a machine gun and a sniper rifle) which were positioned in front of the FPSController and parented to the FirstPersonCharacter. The models were downloaded off of free3d.com. The clipping issues were solved by putting guns in a new layer, making the main camera render everything except for the guns, and making a new camera just for the guns to be rendered on. The gun camera was parented to the main camera, the only layer it renders was set to the gun layer, depth field was changed to 1 so that it gets rendered first, depth only was chosen for clear flags.

A crosshair was set in the centre of the canvas. It is just a raw image with a .png image of a crosshair.

A barrel was placed in the scene as a target. Box Collider and Rigid Body components were added to the barrel.

A muzzle flash was made by creating a plane with a material which has an image of a muzzle flash. The collider was removed. The plane was resized, rotated and placed in front of the barrel of each gun. A point light was created as a child object of the plane.

A shooting script was made. The fire rate, damage and impact force was changed for the pistol, machine gun and sniper rifles.

A script to change the weapons with the scroll wheel or number keys was made. An empty object was made and called “WeaponHolder”. It was parented to the FirstPersonCharacter.

A reload animation was made so that the player can know when the gun is getting reloaded. This was made in Unity’s Animation window, just by lowering the guns on the X axis when they’re reloading. A weapon idle animation was made by just lowering a gun a smaller amount on the X axis.

3 menu scenes were made: MainMenu, Options and EndScene.

There are 13 scripts in total. They are described in each section of this document. *The explanation is in the form of italic comments, "//".*

Persistent Data

The score in this script is incremented each time the barrel gets destroyed.
GameData:

```
public class GameData : MonoBehaviour
{
	public Text playerName;
	public int score;
	public Text scoreText;

	private void Start()
	{
	// the score at the start is 0
	playerName.text = PlayerName.charName;
	}

	private void Update()
	{
	PlayerPrefs.SetInt("Score", score);
	DontDestroyOnLoad(this);
	}
}
```

PlayerName is stored, along with other values
PlayerName:

```
public class PlayerName : MonoBehaviour
{
	public InputField nameField;
	public static string charName;
	public static Text displayName;
	public static Text endScore;

	public void OnSubmit()
	{
	charName = nameField.text; // when the player types in their name, it saves the value to the nameField var
	Debug.Log("name: " + charName);
```

```
        DontDestroyOnLoad(this);
        }

        private void Start()
        {
        GameObject player = GameObject.Find("Player");
        GameData gD = player.GetComponent<GameData>();
        endScore.text = PlayerPrefs.GetInt("Score").ToString();
        }
}
```

First Person Controls.

m9Ammo/m4Ammo/SniperAmmo are essentialy the same. They are added to the respective ammo boxes (red for pistol ammo, brown for machine gun ammo, grey for sniper ammo):
m9Ammo:

```
public class m9Ammo : MonoBehaviour
{
        public int x;
        public int y;
        public int z;
        // Update is called once per frame
        void Update()
        {
        transform.Rotate(new Vector3(x, y, z) * Time.deltaTime); // rotates the object
        }

        void OnTriggerEnter(Collider a)
        {
        if (a.gameObject.tag == "Player") // if the player enters the collider
        {
        Debug.Log("entered");
        GameObject M9 = GameObject.Find("M9"); // find the pistol
        MachineGunScript mgs = M9.GetComponent<MachineGunScript>(); // get the script component of the pistol
        mgs.reserveAmmo += 20;  // add 20 ammo to the reserve
        Destroy(gameObject); // delete the ammo box
        }
        }
}
```

If the player somehow tries to exit the map, by trying to jump over the walls, the player will get teleported to the spawnpoint.
RespawnScript:

```
public class RespawnScript : MonoBehaviour
{

	public GameObject spawnPoint; // var for the point where the player should teleport to

	private void OnTriggerEnter(Collider other)
	{
	if (other.gameObject.tag == "Respawn") // if the player enters the trigger of the respawn collider
	{
	this.transform.position = spawnPoint.transform.position; // get the position values of the spawnpoint and change the current position values to those of the spawnpoint
	}
	}
}
```

When the player enters the end scene trigger, the scene is switched to the end scene.
EndSceneTrigger:

```
public class EndSceneTrigger : MonoBehaviour
{
	void OnTriggerEnter(Collider other)
	{
	if (other.gameObject.tag == "Player") // if the object that enters the collider is tagged as Player
	{
	Application.LoadLevel("EndScene"); // EndScene loads
	}
	}
}
```

Weapon holder is an empty game object with the guns added as child objects of it
WeaponSwitcher:

```
public class WeaponSwitcher : MonoBehaviour
{
	public int selectedWeapon = 0;
	public Text m9;
	public Text m4;
	public Text sniper;

	// Use this for initialization
```

```
void Start()
{
SelectWeapon();
}

// Update is called once per frame
void Update()
{
int previousSelectedWeapon = selectedWeapon;

if (Input.GetAxis("Mouse ScrollWheel") > 0f)
{
if (selectedWeapon >= transform.childCount - 1)
selectedWeapon = 0;
else
selectedWeapon++;
}
if (Input.GetAxis("Mouse ScrollWheel") < 0f)
{
if (selectedWeapon <= 0)
selectedWeapon = transform.childCount - 1;
else
selectedWeapon--;
}

if (Input.GetKeyDown(KeyCode.Alpha1)) // if "1" is pressed ...
{
selectedWeapon = 0; // the selected weapon changes to the 0th weapon
}

if (Input.GetKeyDown(KeyCode.Alpha2) && transform.childCount >= 2)
{
selectedWeapon = 1;
}

if (Input.GetKeyDown(KeyCode.Alpha3) && transform.childCount >= 3)
{
selectedWeapon = 2;
}

if (previousSelectedWeapon != selectedWeapon) // enables the scroll function
{
SelectWeapon();
```

```
        }

        if (selectedWeapon == 0) // if the selected weapon is the 0th one, ie. the m9 pistol
        {
        m9.color = Color.red; // the text changes color to red to notify the player of the change
        m4.color = Color.green;
        sniper.color = Color.green;
        }

        if (selectedWeapon == 1)
        {
        m9.color = Color.green;
        m4.color = Color.red;
        sniper.color = Color.green;
        }

        if (selectedWeapon == 2)
        {
        m9.color = Color.green;
        m4.color = Color.green;
        sniper.color = Color.red;
        }
        }

        void SelectWeapon()
        {
        int i = 0;
        foreach (Transform weapon in transform)
        {
        if (i == selectedWeapon)
        weapon.gameObject.SetActive(true);
        else
        weapon.gameObject.SetActive(false);
        i++;
        }
        }
}
```

Rigid Bodies & Colliders

Rigid body were placed on the barrel prefab. A box collider was placed on the barrel prefab. They are manipulated in scripts. A box collider was placed on the collectible ammo boxes.

Spawning Targets

When the player enters a trigger, barrels spawn in one of the 5 spawnpoints.
SpawnObject:

```
public class SpawnObject : MonoBehaviour
{
	public Transform[] spawnPoints;
	public GameObject prefab;
	public AudioSource spawnSound;

	void OnTriggerEnter() // when the player enters the trigger
	{
	int spawnPointIndex = Random.Range(0, spawnPoints.Length); // a random spawnpoint is chosen
	spawnSound.GetComponent<AudioSource>().Play(); // plays sound to notify player of the spawning
	Instantiate(prefab, spawnPoints[spawnPointIndex].position, spawnPoints[spawnPointIndex].rotation); // spawns a prefab (barrel) in the position of the spawnpoint
	}
}
```

Destroying Targets

When the player shoots, the fire rate, force of impact, damage are dependent on the gun the player is equipped with. If the ammo in the current clip is less or equal to 0, the gun gets reloaded. If the player presses the R key, the player reloads the gun automatically. When the gun is reloaded, the ammo needed for the reload gets subtracted from the reserve ammo.
MachineGunScript:

```
public class MachineGunScript : MonoBehaviour
{
	// Shooting
	public float damage = 10;
	public Camera fpsCam;
	public float impactForce = 30;
	public float fireRate = 15f;
	private float nextTimeToFire = 0;

	// Sound
	public AudioClip audShot;
	public AudioClip audReload;
```

```
public AudioSource hitSoundEffect;

// Muzzle flash
public GameObject muzzleFlash;
float timeToDisable = 0.0f;

// Crosshair
public GameObject crosshair;
public GameObject crosshairRed;

// Ammo
public int maxAmmo = 10;
int lastClip = 0;
public int reserveAmmo = 60;
public int reloadAmmo;
private int currentAmmo;
public float reloadTime = 1f;
private bool isReloading = false;
public Animator animator;
public Text magAmmoText;
public Text reserveAmmoText;

void Start()
{
if (currentAmmo == -1)
currentAmmo = maxAmmo; // equates the current clip to the maximum it can hold
// draws the default crosshair
crosshair.SetActive(true); // white crosshair gets displayed
crosshairRed.SetActive(false); // red crosshair doesn't get displayed
}

void OnEnable()
{
isReloading = false;
animator.SetBool("Reloading", false);
}


// Update is called once per frame
void Update()
{

if (isReloading)
```

```
        return; // doesn't go further down the code of the Update() method

        if (currentAmmo <= 0 && reserveAmmo > 0 || (Input.GetKey(KeyCode.R))) // if the player runs out of ammo and has enough ammo, the game reloads the gun automatically, or the player can press R to reload manually
        {
        StartCoroutine(Reload()); // starts the coroutine for relading
        return;// doesn't go further down the code of the Update() method
        }

        if (Input.GetButton("Fire1") && Time.time >= nextTimeToFire && currentAmmo > 0) // checks if the player has clicked LMB, checks if it's time to shoot, checks if there is enough ammo to shoot
        {
        nextTimeToFire = Time.time + 1f / fireRate; // sets the firerate by dividing the current time (+1 second) by a public fire rate value

        Shoot(); // starts the Shoot() method
        }

        if (timeToDisable <= 0.0f) // if the time isn't to shoot
        {
        muzzleFlash.SetActive(false); // the muzzle flash doesn't display
        }
        else
        {
        timeToDisable -= Time.deltaTime;
        }

        magAmmoText.text = currentAmmo.ToString(); // displays current ammo
        reserveAmmoText.text = reserveAmmo.ToString(); // displays reserve ammo

        // change color to red if there isn't much ammo
        if (currentAmmo <= 3) // if ammo is low...
        {
        magAmmoText.color = Color.red; // ... the color of the text notifies the player by changing the text to red
        }
        if (reserveAmmo <= 3)
        {
        reserveAmmoText.color = Color.red;
        }
        if (currentAmmo > 3) // if ammo isn't low ...
```

```
        {
        magAmmoText.color = Color.green; // ... the color is green
        }
        if (reserveAmmo > 3)
        {
        reserveAmmoText.color = Color.green;
        }
        }

        IEnumerator Reload()
        {
        isReloading = true;
        muzzleFlash.SetActive(false);
        Debug.Log("Reloading");

        AudioSource audio = GetComponent<AudioSource>();
        audio.PlayOneShot(audReload);
        animator.SetBool("Reloading", true);

        yield return new WaitForSeconds(reloadTime - .25f);
        animator.SetBool("Reloading", false);
        yield return new WaitForSeconds(.25f);

        // Replaces used ammo (lastClip) with ammo from reserve (reserveAmmo); then resets
used ammo counter (lastClip)
        if (reserveAmmo > 0)
        {
        currentAmmo = maxAmmo;
        reserveAmmo = reserveAmmo - lastClip;
        lastClip = 0;
        }

        // Makes sure that the reserve ammo doesn't become something ilogical like a negative
number
        if (reserveAmmo < 0)
        {
        reserveAmmo = 0;
        }

        isReloading = false;
        }
```

```
        void Shoot() // If the player clicks LMB, raycast checks if it's the target. If it is, it deals
damage
        {
        currentAmmo--; // subtracts one bullet from current clip
        lastClip++; // adds 1 bullet to int that is going to be subtracted from reserveAmmo when
player reloads

        // Muzzleflash
        muzzleFlash.SetActive(true);
        timeToDisable = 0.02f;

        //GetComponent<AudioSource>().Play();
        AudioSource audio = GetComponent<AudioSource>();
        audio.PlayOneShot(audShot);

        // Shooting
        RaycastHit hit;
        if (Physics.Raycast(fpsCam.transform.position, fpsCam.transform.forward, out hit)) // a
raycast comes out of the player's camera
        {
        Debug.Log(hit.transform.name);

        Target target = hit.transform.GetComponent<Target>(); // if the target is hit
        if (target != null)
        {
        target.TakeDamage(damage); // takes damage
        hitSoundEffect.GetComponent<AudioSource>().Play();
        }

        if (hit.rigidbody != null) // if a rigidbody is hit
        {
        hit.rigidbody.AddForce(-hit.normal * impactForce); // force is added to the rigid body,
multiplied by a public var for the impactForce
        }
        // Coroutine to show red crosshair
        StartCoroutine(CrosshairShot());
        }

        crosshairRed.SetActive(true); // displays the red crosshair when the player is shooting
        crosshair.SetActive(false);

        }
```

```
        public IEnumerator CrosshairShot()
        {
        yield return new WaitForSeconds(0.1f);
        // Crosshair changes color to red when the gun is shooting
        crosshair.SetActive(true);
        crosshairRed.SetActive(false);
        }
}
```

If the health of the barrel reaches 0 or less, it gets destroyed.
Target:

```
public class Target : MonoBehaviour
{
        public float health = 10f;
        public GameObject explosionPrefab;
        public AudioSource boomSound;

        private void Start()
        {
        GameObject Player = GameObject.Find("Player");
        GameData gameData = Player.GetComponent<GameData>();
        }

        public void TakeDamage(float amount)
        {
        health -= amount; //subtracts the amount value from health
        if (health <= 0f) // if the value of health is less than or equal to 0
        {
        StartCoroutine(Die()); // Die() coroutine starts
        }
        }

        private void Update()
        {
        GameObject Player = GameObject.Find("Player");
        GameData gameData = Player.GetComponent<GameData>();
        gameData.scoreText.text = "Score: " + gameData.score; // displays the score
        DontDestroyOnLoad(this);
        }

        public IEnumerator Die()
        {
```

```
        if (explosionPrefab != null)
        {
        Instantiate(explosionPrefab, transform.position, Quaternion.identity); // explosion effect spawns in the position of the barrel
        yield return new WaitForSeconds(1f); // waits for the explosion effect to finish
        }
        Destroy(gameObject); // deletes the object
        {
        GameObject Player = GameObject.Find("Player");
        GameData gameData = Player.GetComponent<GameData>();
        gameData.score++; // increments score
        }
        }
}
```

Graphical User Interface

Coroutines were used to play sound of button clicks before another scene gets loaded, except for the quit button, which exits the game immediately.

ButtonManager:

```
public class ButtonManager : MonoBehaviour
{
        public Text endScore;

        private void Start()
        {
        GameObject Player = GameObject.Find("Player");
        GameData gameData = Player.GetComponent<GameData>();
        endScore = gameData.scoreText;
        }

        void Update()
        {
        Cursor.visible = true; // displays the cursor
        Cursor.lockState = CursorLockMode.None; // unlocks the cursor
        }

        // coroutines that play the click sound and then load the scenes
        IEnumerator DelayPlay()
        {
        GetComponent<AudioSource>().Play();
        yield return new WaitForSeconds(
```

```
        GetComponent<AudioSource>().clip.length);
        Application.LoadLevel("Main");
        }

        IEnumerator DelayMainMenuBtn()
        {
        GetComponent<AudioSource>().Play();
        yield return new WaitForSeconds(
        GetComponent<AudioSource>().clip.length);
        Application.LoadLevel("MainMenu");
        }

        IEnumerator DelayOptions()
        {
        GetComponent<AudioSource>().Play();
        yield return new WaitForSeconds(
        GetComponent<AudioSource>().clip.length);
        Application.LoadLevel("Options");
        }
        // sets the coroutines to methods
        public void DelayPlay(string Main)
        {
        StartCoroutine(DelayPlay());
        }

        public void DelayMainMenu(string MainMenu)
        {
        StartCoroutine(DelayMainMenuBtn());
        }

        public void DelayOptions(string Options)
        {
        StartCoroutine(DelayOptions());
        }

        public void QuitGame() // when the player chooses to quit the game, the game exits
instantly, without a delay for sound
        {
        Application.Quit();
        }

}
```

Audio

Music is seamless across all scenes.
MusicDontDestroy:

```
public class MusicDontDestroy : MonoBehaviour
{
	private void Awake()
	{
	GameObject[] objs = GameObject.FindGameObjectsWithTag("BGM"); // creates an array of objects with the bgm (background music) tag
	if (objs.Length > 1) // if there are more than 1 objects with the bgm tag, ie. if more than 1 instances of music are playing
	Destroy(this.gameObject); // then the object gets destroyed to insure that only 1 instance of music is playing
	DontDestroyOnLoad(this.gameObject); // the music is played seamlessly throughout scenes, because the object with the audio source doesn't get destroyed
	}
}
```

Volume changes by sliding the slider, ie. by making the value of the slider the same as the value of the volume.
VolumeSlider:

```
public class VolumeSliderScript : MonoBehaviour
{
	public Slider slider; // var for volume slider
	public AudioSource volumeAudio; // var for audioSource

	private void Awake()
	{
	if (slider)
	{
	GetComponent<AudioSource>().volume = PlayerPrefs.GetFloat("CurVol"); // gets the value from PlayerPrefs
	slider.value = GetComponent<AudioSource>().volume; // equates the value of the slider to the volume value of the audio
	}
	}

	public void VolumeControl(float volumeControl)
	{
```

```
        GetComponent<AudioSource>().volume = volumeControl; // sets the volume of
AudioSource to var volumeControl
        PlayerPrefs.SetFloat("CurVol", GetComponent<AudioSource>().volume); // equates the
value of the slider to the volume value of the audio
        PlayerPrefs.Save(); // saves the value to PlayerPrefs
        }

        private void Update()
        {
        VolumeControl(slider.value); // updates the value of the slider every frame
        }
}
```

Extra Features

- *3 guns, which can be switched with the mouse wheel or number keys*
- *Text color changes to red to show which gun is selected*
- *When ammo is low (less than 3), the ammo text changes color to red*
- *Crosshair color changes to red when the gun shoots*