Paradigma

Yadiel, Ariel y Omar

General overview: Our videogame paradigma is a survival first person shooter inspired by similar games of the genre such as vampire survivors and Devil Daggers. In our game you spawn in the middle of a coliseum and the goal is to survive an endless horde of running zombies and wizard skeletons as long as possible to rack up as many points as you can, like in old arcade survival games. It is also important to note we took great inspiration from everything we learned in class to the point where we also used many of the class scripts as cornerstones to achieve our goal in the project.

Our first steps were in deciding were we wanted our game to take place, and we ended up deciding for a free coliseum asset store we found, and after this we took it upon ourselves to look at some enemy prefabs and decided upon a wizard skeleton with a wand for our ranged unit and to fit in the theme we added a zombie skeleton from the mixamo page, with an integrated running animation. Our game consists of pure survival to the max like the original arcades, so the shooter enemies and the runner enemies which we created, are constantly spawning and if you're not careful you will get hit and hence your life will be forfeit giving you a game over. Considering we took mayor inspiration from the games mentioned in our general overview, we also decided it would be best to make it so that you the player, and the enemy we encounter only have 1

health point, not only does this make the game more skill focused but it also adds to the stress and hardcore element we wanted to incorporate into our game.

Our coloseium is a sprite downloaded for free from the internet, however it is a sprite that works with the NavMesh package which is what we will be utilizing as the AI for the game. The NavMesh project works in a very peculiar way in the fact that it can automatically trace the places in which the ai can stand on/walk through. This package was crucial to our project due to it being so focused on survival of the main character from thye endless onslaught of enemies. We utilized the navmesh on the runner characters as a way for the to get close to the player without them noticing and for them to die instantly upon collision. On the other hand we utilized it on the shooters as a tracking device for them, the shooters do not follow u around the same way the runner enemies do but they do constantly shoot at you from a distance so taking care of them is a wise choice before u continue to slay the rest of the runners.

The player will be spawned in the middle of the coliseum and will have to survive as long as possible, an important feature the player has is the ability to switch between two weapons, a shotgun and an assault rifle, these two weapons will indeed prove to be pivotal to the maximal survival in different situations, if one is surrounded by many mobs it is indeed wise to utilize the shotgun to clear all the enemies, however when the field is clear utilizing the assault rifle is indeed the wiser choice. An important thing to keep in mind is that there is a limited amount of ammunition before you have to reload and in

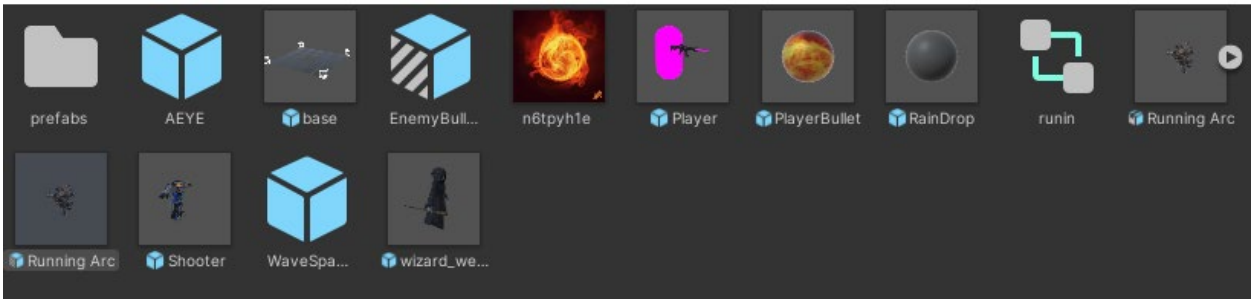
this amount of reload time you will be vulnerable to the attack of the enemies and the runners.

Conforming the themes that were discussed in class that came in handy when making the project, we certainly made use of “primitives” for what is essentially the most important part of the game, the damage/healthbar and scores. The damage the user deals, although it instantly kills the enemies, was managed with this type of primitive structure and the amount of points you can get by killing enemies was also managed by this type of manipulation.

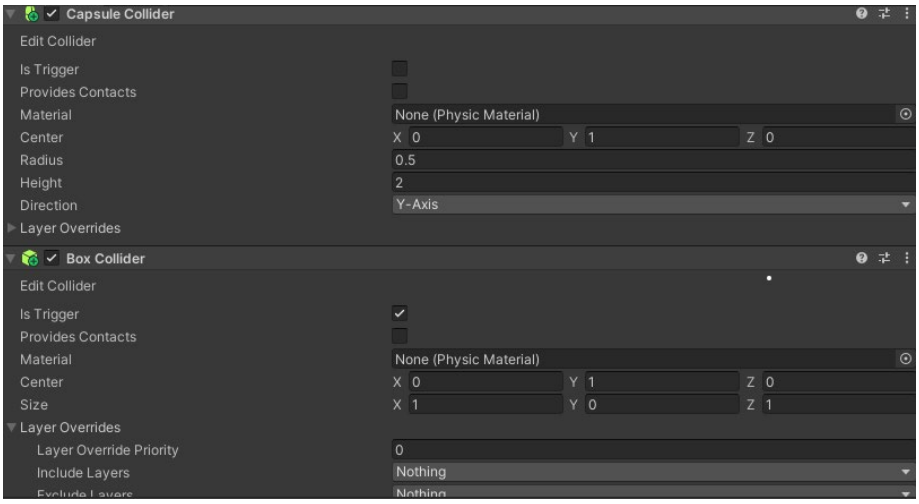
In terms of prefabs and hierarchy prefabs played an essential role in our game in many aspects. The spawners used pre-established prefabs which can be found in the assets/prefabs folder where you will be able to find a “Running Arc” prefab and “wizard_weapon_legacy DEMO” prefab which are the enemies used for the main game itself. In that same folder one will be able to find the bullets used by the wizards and the

prefab for the player, not to mention an incredibly important prefab called “AEYE” which is what our enemies use as AI to locate the player and either chase or shoot at him.

Conforming the colliders there is not a lot of fancy stuff going on in with them like in other parts, the only real colliders that were implemented were for the hitboxes of both the enemies and the player. An important note to mention is that the runner enemies do in fact have two colliders, a box collider which acts as a trigger for attacking

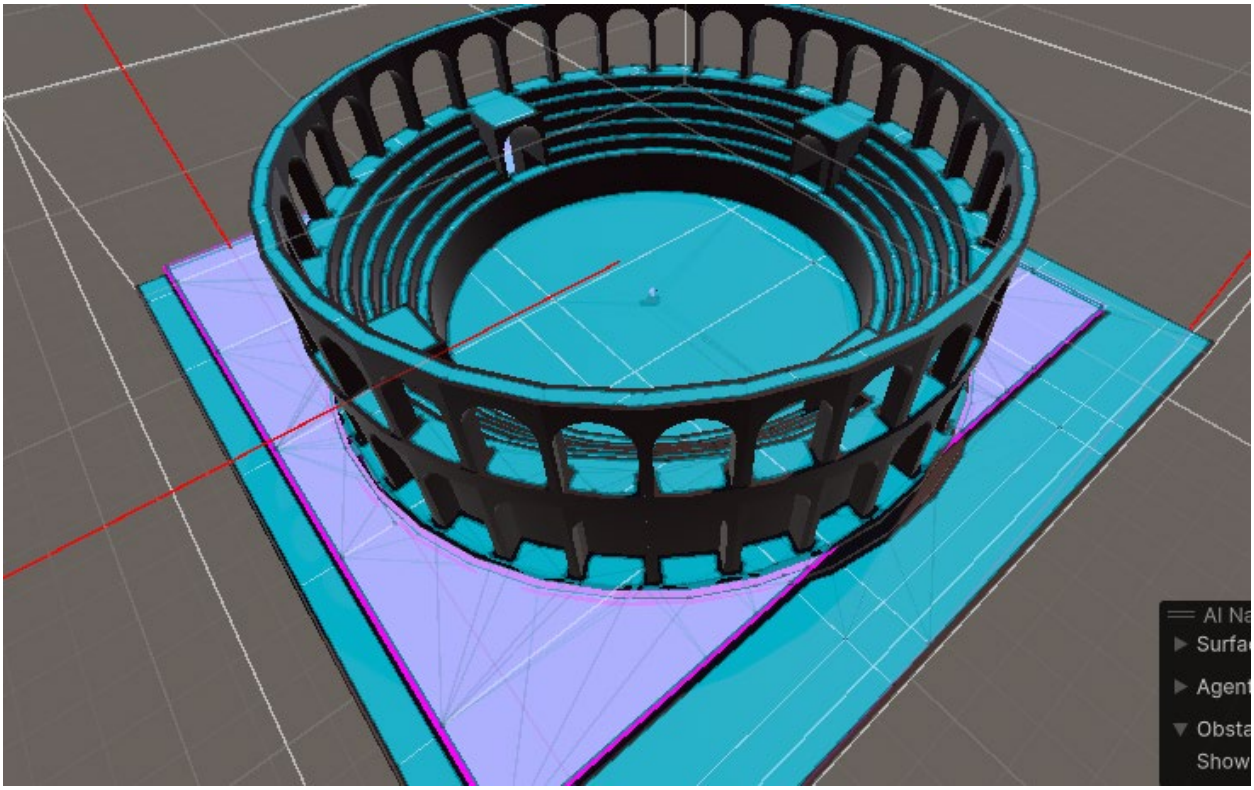


the enemy and a capsule collider which acts as the collider which registers the bullets damage, for some reason which i simply can’t explain if one of the two colliders is taken off the enemy breaks and just can’t take damage or deal damage for that matter, the interaction between the two colliders is unknown to me however if both are turned on it works fine.

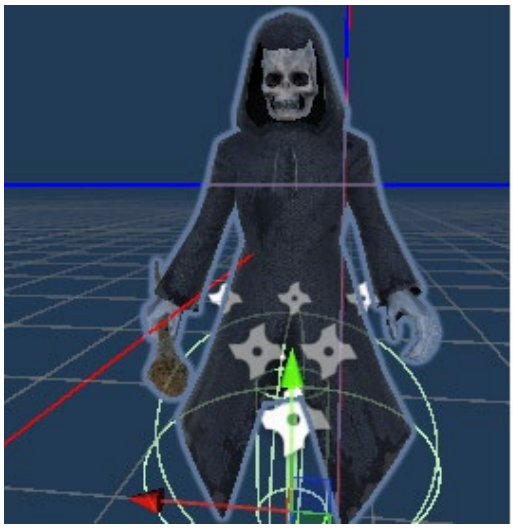


Our map consisted of a free coliseum blender asset found in this link https://www.turbosquid.com/3d-models/free-lwo-model-arena-roman/516687 and enlarged it quite a bit, while also utilizing the AI Navigation package and baked the model so it had a ground to use as ai ground, and we also put a flat plane just below the baked terrain so our characters and enemies wouldn't just, faze out of existence

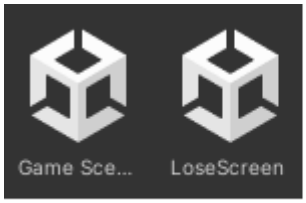
through the arena.



As for textures, we used a mix of many different textures for different things. Our arena had a simple yet effective material change which made it black and had it look more ominous. As for our enemies, here is where it gets interesting. We imported a mixamo asset called SKELETONZOMBIE and a free unity demo asset called FantasyMonsterWizard. The guns also came in a package and had a wide array of options for them, from a package called Ishikawa1116 and although we decided to only use the shotgun and rifle with the bullets, the package had more options to choose from.



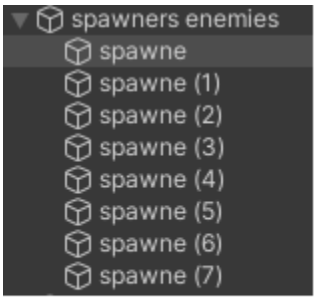
The event programing part we simply took inspiration from what we learned in class and implemented it in our scripts/project. We simply had our main scene and the loosing screen which would be called upon once the character was destroyed by either



the shooters or chaser enemies.

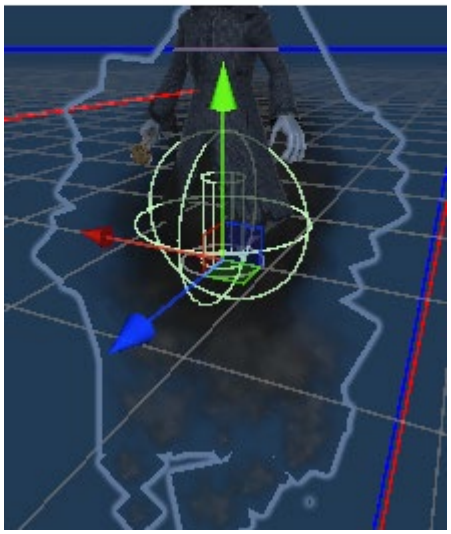
Now when we come talk about the spawning of the enemies that was an absolute headache, not the spawning itself but the implementation of it in correlation with utilizing the AI Navigation package. Something we did not know until Yadiel painstakingly found out how to fix it after hours of trying is that enemies **MUST** spawn in baked area for their ai to work properly, if that is not the case then not only will the ai from NavMesh not work, but it will throw two errors, “"SetDestination" can only be called on an active agent that has been placed on a NavMesh.” and “"Resume" can only be

called on an active agent that has been placed on a NavMesh.” these two errors were certainly a gigantic headache to deal with and the fact is that they only happened on spawned enemies. If an enemy was in the scene originally they would work just fine with the navmesh ai, this is an error which although fixed, we do not know what causes the difference in treatment from spawned enemies to enemies which are already located in the scene itself, all we can say is that the fix we found was to place the spawners essentially directly in the baked area so they would not only stop throwing errors but also attack the player when spawned.

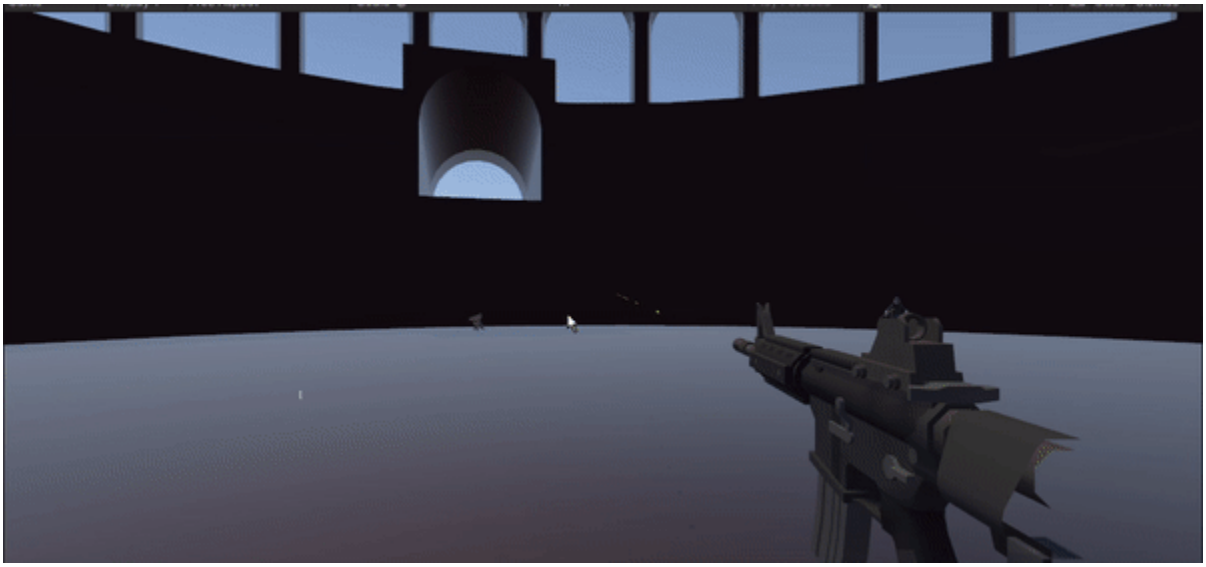


On our game the only real physics comes from the players bullets and the fact that we had to put a plane so the characters and enemies didn’t just faze through the floor. Adding to that we also did not use any illumination from anywhere and just used the base camara light the scene adds when u make a new scene in unity.

We did not make any manual particle system however, our guns do in fact have a lot of fancy particles which were integrated by Omar after a lot of work, not only does shooting have a particle system but also the collisions of bullets and the act of

reloading. These parts were edited from a the gun package we installed to suit our needs and wants for the game. Another particle system implemented in the game is from our skeleton shooter enemy, their base animation has their lower body constantly spewing out smoke, which although was not implemented by us, did in fact come from the asset we downloaded from unity. 

It is important to note that we took great lengths to have as much interactivity with our enemies as possible with our knowledge, as thus we were able to implement the animations our enemies came with into a perpetual loop to look as fancy as we could make it, our runners are constantly sprinting at the enemy with their arms flapping around endlessly which looks incredibly creepy and for our shooter we decided to utilize the attack animation it had and loop it infinitely as well considering it does not move and all it can do is just stand there menacingly shooting at you until either you or it dies.



IMPORTANT TO NOTE FRAME RATE IN GIF DOES NOT REPRESENT ACTUAL GAMEPLAY, GIFS HAVE LOWER FRAME RATE NATURALLY IF THEY ARE GOING TO BE LONGER***

First Person Shooting Mechanic

- The FPS mechanic enables players to experience the game from a first-person perspective, engaging in combat scenarios with enemies using a variety of firearms and weapons.

Objective:

- The primary goal is to provide players with an immersive and interactive shooting experience, strategic thinking, and combat skills.

Implementation:

- The FPS Mechanic is made from a Cylinder prefab that contains Children and Scripts for it to make the player move and look around the map. Canvas contains the crosshair for the FPS.

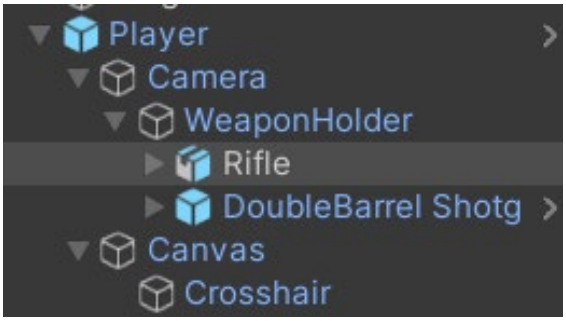
i. **FPSController:** The script handles player movement based on input from the Unity Input System.

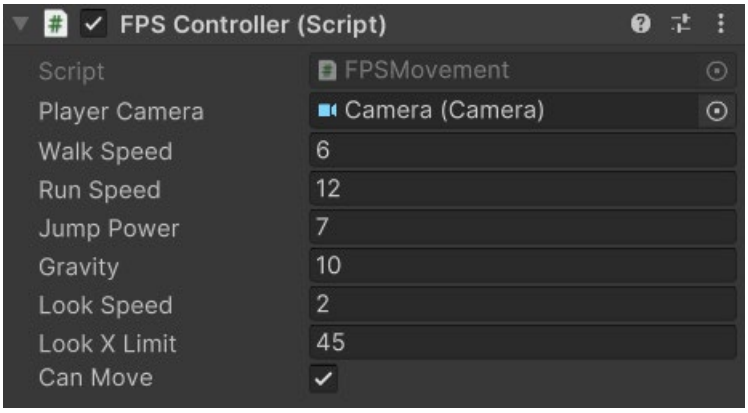
a) The walkSpeed and runSpeed variables determine the movement speed, with the option to run by pressing the Left Shift key.

b) The curSpeedX and curSpeedY variables calculate the current movement speed in the forward and right directions.

c) The Vector3 moveDirection variable represents the overall movement direction, combining forward and right vectors.

d) The character's vertical movement is controlled by the jumpPower variable, allowing the player to jump when the "Jump" button is pressed.





ii. **Camera and Player Rotation:** The script manages the rotation of the player character and the attached camera.

a) The lookSpeed variable controls the sensitivity of the camera's vertical and horizontal rotation.

b) The lookXLimit variable restricts the vertical rotation of the camera, preventing it from exceeding a certain angle.

c) The rotationX variable stores the current vertical rotation of the camera.

d) Player camera rotation is controlled by the mouse input along the Y-axis, clamped to the specified limits.

e) The player character rotates based on the mouse input along the X-axis, affecting horizontal rotation.

f) The script ensures that these rotations only.

```
5   [RequireComponent(typeof(CharacterController))]
    0 references
6   public class FPSController : MonoBehaviour
7   {
        1 reference
8       public Camera playerCamera;
        2 references
9       public float walkSpeed = 6f;
        2 references
10      public float runSpeed = 12f;
        1 reference
11      public float jumpPower = 7f;
        1 reference
12      public float gravity = 10f;
13
14
        2 references
15      public float lookSpeed = 2f;
        2 references
16      public float lookXLimit = 45f;
17
18
        6 references
19      Vector3 moveDirection = Vector3.zero;
        4 references
20      float rotationX = 0;
21
        4 references
22      public bool canMove = true;
23
24
        4 references
25      CharacterController characterController;
```

```
26      void Start()
27      {
28          characterController = GetComponent<CharacterController>();
29          Cursor.lockState = CursorLockMode.Locked;
30          Cursor.visible = false;
31      }
32
```

```
33      void Update()
34      {
35
36          #region Handles Movment
37          Vector3 forward = transform.TransformDirection(Vector3.forward);
38          Vector3 right = transform.TransformDirection(Vector3.right);
39
40          // Press Left Shift to run
41          bool isRunning = Input.GetKey(KeyCode.LeftShift);
42          float curSpeedX = canMove ? (isRunning ? runSpeed : walkSpeed) * Input.GetAxis("Vertical") : 0;
43          float curSpeedY = canMove ? (isRunning ? runSpeed : walkSpeed) * Input.GetAxis("Horizontal") : 0;
44          float movementDirectionY = moveDirection.y;
45          moveDirection = (forward * curSpeedX) + (right * curSpeedY);
46
47          #endregion
48
49          #region Handles Jumping
50          if (Input.GetButton("Jump") && canMove && characterController.isGrounded)
51          {
52              moveDirection.y = jumpPower;
53          }
54          else
55          {
56              moveDirection.y = movementDirectionY;
57          }
58
59          if (!characterController.isGrounded)
60          {
61              moveDirection.y -= gravity * Time.deltaTime;
62          }
63
64          #endregion
65
```

```
66          #region Handles Rotation
67          characterController.Move(moveDirection * Time.deltaTime);
68
69          if (canMove)
70          {
71              rotationX += -Input.GetAxis("Mouse Y") * lookSpeed;
72              rotationX = Mathf.Clamp(rotationX, -lookXLimit, lookXLimit);
73              playerCamera.transform.localRotation = Quaternion.Euler(rotationX, 0, 0);
74              transform.rotation *= Quaternion.Euler(0, Input.GetAxis("Mouse X") * lookSpeed, 0);
75          }
76
77          #endregion
78      }
79  }
```

iii. **Weapon Switch:** This script adds a sway effect to the weapon based on mouse movement.

- public int selectedWeapon: Represents the index of the currently selected weapon.
- In Start(), the SelectWeapon() function is called to set the initial weapon.

a) Update Loop:

- Update(): Checks for input to switch weapons.

- Use the mouse scroll wheel to cycle through weapons.
- Uses number keys (1, 2, 3) to directly select a weapon.
- Calls SelectWeapon() when a change in the selected weapon occurs.

b) SelectWeapon():

- Iterates through each child transformation of the weapons holder.
- Activates the selected weapon while deactivating others.

```
3   public class WeaponSwitch : MonoBehaviour
4   {
        12 references
5       public int selectedWeapon = 0;
6
7       // Start is called before the first frame update
        0 references
8       void Start()
9       {
10          SelectWeapon();
11      }
12
13      // Update is called once per frame
        0 references
14      void Update()
15      {
16          int previousSelectedWeapon = selectedWeapon;
17
18          if (Input.GetAxis("Mouse ScrollWheel") > 0f)
19          {
20              if (selectedWeapon >= transform.childCount - 1)
21                  selectedWeapon = 0;
22              else
23                  selectedWeapon++;
24          }
25
26          if (Input.GetAxis("Mouse ScrollWheel") < 0f)
27          {
28              if (selectedWeapon <= 0)
29                  selectedWeapon = transform.childCount - 1;
30              else
31                  selectedWeapon--;
32          }
33
```

```
34          if (Input.GetKeyDown(KeyCode.Alpha1))
35          {
36              selectedWeapon = 0;
37          }
38
39          if (Input.GetKeyDown(KeyCode.Alpha2) && transform.childCount >= 2)
40          {
41              selectedWeapon = 1;
42          }
43
44          if (Input.GetKeyDown(KeyCode.Alpha3) && transform.childCount >= 3)
45          {
46              selectedWeapon = 2;
47          }
48
49          if (previousSelectedWeapon != selectedWeapon)
50          {
51              SelectWeapon();
52          }
53      }
54
        2 references
55      void SelectWeapon()
56      {
57          int i = 0;
58          foreach (Transform weapon in transform)
59          {
60              if (i == selectedWeapon)
61                  weapon.gameObject.SetActive(true);
62              else
63                  weapon.gameObject.SetActive(false);
64
65              i++;
66          }
67      }
68  }
```

iv. **Weapon Sway:** This script adds a sway effect to the weapon based on mouse movement.

a) Sway Settings:

- [SerializeField] private float smooth;: Controls the smoothness of the sway effect.
- [SerializeField] private float swayMultiplier;: Determines the intensity of the sway effect.

b) Update Loop:

- Update(): Retrieves mouse input and applies a sway effect to the weapon's rotation.
- Input.GetAxisRaw("Mouse X") and Input.GetAxisRaw("Mouse Y") capture mouse movement.
- Calculates rotations (rotationX and rotationY) based on mouse input.
- Interpolates between the current rotation and the target rotation using Quaternion.Slerp.
- Applies the resulting rotation to the weapon's local rotation.

```
5   public class WeaponSway : MonoBehaviour
6   {
7
8       [Header("Sway Settings")]
        1 reference
9       [SerializeField] private float smooth;
        2 references
10      [SerializeField] private float swayMultiplier;
11
12      // Update is called once per frame
        0 references
13      void Update()
14      {
15          float mouseX = Input.GetAxisRaw("Mouse X") * swayMultiplier;
16          float mouseY = Input.GetAxisRaw("Mouse Y") * swayMultiplier;
17
18
19          Quaternion rotationX = Quaternion.AngleAxis(-mouseY, Vector3.right);
20          Quaternion rotationY = Quaternion.AngleAxis(mouseX, Vector3.up);
21
22          Quaternion targetRotation = rotationX * rotationY;
23
24          transform.localRotation = Quaternion.Slerp(transform.localRotation, targetRotation, smooth * Time.deltaTime);
25      }
26  }
```

c) Input Handling:
- Input.GetAxis("Mouse ScrollWheel"): Checks the mouse scroll wheel for weapon switching.
- Input.GetKeyDown(KeyCode.Alpha1), Input.GetKeyDown(KeyCode.Alpha2), etc.: Checks for number key presses for direct weapon selection.

d) Edge Cases:
- Checks for edge cases when scrolling or using number keys to ensure the selected weapon index stays within valid bounds.

e) SelectWeapon():
- Deactivates all weapons except the one at the selected index.
- Uses a loop to iterate through each child transformation of the weapons holder.

- Activates the selected weapon and deactivates others based on the index.

```
public class WeaponSway : MonoBehaviour
{

    [Header("Sway Settings")]
    1 reference
    [SerializeField] private float smooth;
    2 references
    [SerializeField] private float swayMultiplier;

    // Update is called once per frame
    0 references
    void Update()
    {
        float mouseX = Input.GetAxisRaw("Mouse X") * swayMultiplier;
        float mouseY = Input.GetAxisRaw("Mouse Y") * swayMultiplier;


        Quaternion rotationX = Quaternion.AngleAxis(-mouseY, Vector3.right);
        Quaternion rotationY = Quaternion.AngleAxis(mouseX, Vector3.up);

        Quaternion targetRotation = rotationX * rotationY;

        transform.localRotation = Quaternion.Slerp(transform.localRotation, targetRotation, smooth * Time.deltaTime);
    }
}
```

v. **Gun Script**: The Gun script is responsible for handling shooting mechanics, including damage calculation, range, fire rate, and impact effects. The script interacts with other game objects and components, such as the FPS camera, particle systems for muzzle flash, impact effects, and an animator for reloading animations.

a) Initialization: The Start() method initializes the gun's ammo count and disables the script by default.

b) OnEnable(): The OnEnable() method resets the ammo count and ensures that reloading is not in progress when the gun is enabled.

c) Update Loop: The Update() method checks for input to shoot and triggers the Shoot() function if conditions are met.

d) Reload Mechanism: The Reload() coroutine handles the reload process, including animations and delays.

e) Shoot Mechanism: The Shoot() method executes the shooting logic, including muzzle flash, ammo consumption, spread calculation, raycasting, and impact effects.

```
4   public class Gun : MonoBehaviour
5   {
        1 reference
6       public float damage = 10f;
        1 reference
7       public float range = 100f;
        1 reference
8       public float fireRate = 15f;
        1 reference
9       public float impactForce = 30f;
        4 references
10      public float spread = 0f;
11
        2 references
12      public int maxAmmo = 10;
        1 reference
13      public int numberOfPellets = 5; // Change the type to int
        4 references
14      private int currentAmmo;
        1 reference
15      public float reloadTime = 1f;
        4 references
16      private bool isReloading = false;
17
        2 references
18      public Camera fpscamera;
        1 reference
19      public ParticleSystem muzzleflash;
        1 reference
20      public GameObject impactEffect;
21
        2 references
22      public Animator animator;
23
        2 references
24      private float nextTimeToFire = 0f;
25
```

```
26      void Start()
27      {
28          currentAmmo = maxAmmo;
29      }
30
        0 references
31      void OnEnable()
32      {
33          isReloading = false;
34          animator.SetBool("Reloading", false);
35      }
36
        0 references
37      void Update()
38      {
39          if (isReloading)
40          {
41              return;
42          }
43          if (currentAmmo <= 0)
44          {
45              StartCoroutine(Reload());
46              return;
47          }
48
49          if (Input.GetButton("Fire1") && Time.time >= nextTimeToFire)
50          {
51              nextTimeToFire = Time.time + 1f / fireRate;
52              Shoot();
53          }
54      }
55
```

```
56      IEnumerator Reload()
57      {
58          isReloading = true;
59
60          animator.SetBool("Reloading", true);
61
62          yield return new WaitForSeconds(reloadTime - .25f);
63          yield return new WaitForSeconds(1f);
64
65          currentAmmo = maxAmmo;
66          isReloading = false;
67      }
68
```

```
    void Shoot()
    {
        muzzleflash.Play();

        currentAmmo--;

        for (int i = 0; i < numberOfPellets; i++)
        {
            float spreadX = UnityEngine.Random.Range(-spread, spread);
            float spreadY = UnityEngine.Random.Range(-spread, spread);

            Vector3 direction = fpscamera.transform.forward + new Vector3(spreadX, spreadY, 0);

            RaycastHit hit;
            if (Physics.Raycast(fpscamera.transform.position, direction, out hit, range))
            {
                UnityEngine.Debug.Log(hit.transform.name);

                Life lifeComponent = hit.transform.GetComponent<Life>();
                if (lifeComponent != null)
                {
                    lifeComponent.TakeDamage(damage);
                }

                if (hit.rigidbody != null)
                {
                    hit.rigidbody.AddForce(-hit.normal * impactForce);
                }

                GameObject impactGO = Instantiate(impactEffect, hit.point, Quaternion.LookRotation(hit.normal));
                Destroy(impactGO, 2f);
            }
        }
    }
}

```

This script is designed to work in conjunction with the WeaponSwitch script, allowing for seamless weapon switching and shooting mechanics.

IMPORTANT TO NOTE FRAME RATE IN GIF DOES NOT REPRESENT ACTUAL GAMEPLAY, GIFS HAVE LOWER FRAME RATE NATURALLY IF THEY ARE GOING TO BE LONGER***