Deccan Education Society's
Kirti M. Doongursee College of Arts, Science and Commerce
[NAAC Accredited: "A Grade"]

T. Y. BSc. [Computer Science]

Project Documentation

Seat Number [ ]

Roll No: 64

Student ID: 681400

Department of Computer Science and Information Technology

Department of Computer Science and Information Technology
Deccan Education Society's
Kirti M. Doongursee College of Arts, Science and Commerce
[NAAC Accredited: "A Grade"]

C E R T I F I C A T E

This is to certify that Mr. / ~~Miss.~~ **SHLOK G. SHIVKAR** of T. Y. BSc. (Computer Science) with Seat No. **64** has completed the Project Implementation – USCS603 under my supervision in this College during the year 2020-2021.

Lecturer-In-Charge

Date: / / 2021

H.O.D.
Department of
Computer Science & IT

Date: / / 2021

Examined by:

Date: / / 2021

Remarks:

_____

_____

A Project Report

On

"The Forest"

Game Development : Unity 3D Project

Submitted in partial fulfillment of the requirement of

University of Mumbai

For the Degree of

Bachelor of Computer Science

Submitted By

SHLOK G. SHIVKAR

Under the Guidance of

Prof. SIDDHESH KADAM

Department of Computer Science and Information Technology

Deccan Education Society's

Kirti M. Doongursee College of Arts, Science and Commerce

[NAAC Accredited: "A Grade"]

Mumbai

(2020-2021)

INDEX

A Unity 3D Project

“ The Forest ”

Designed and Developed

by

Shlok G. Shivkar

Introduction

"The Forest" is a single player 3D shooting game designed and developed in unity game engine, in which the player starts in the jungle filled with cannibals and boars, the sole purpose of the player is to survive as long as possible using the given tools and weapons.

Requirements

★ System Requirements :

- Intel Core i3 Dual Core.
- 4 GB RAM (6~8 GB Recommended).
- 4 GB free hard disk space (80 GB Recommended).
- Windows 7 and above.

★ Software :

- Unity 2.4.3 (Version 2020.3.1f1)

★ Technology :

- Unity 2.4.3 (Version 2020.3.1f1)
- Raycasting

★ Language :

- C Sharp (C#).

Methodology

This project is based on 3D Game Development

- In this project, I have used assets and animations from the unity asset store and compiled them to get the final game
- Unique animations have been added to each model to give it a smooth gameplay using animator controller (For shooting, enemy animations etc.)
- As for shooting, Raycasting technology has been used in this game

The player has first person camera and can access 6 offensive weapons

- Axe
- Revolver
- Shotgun
- Assault Rifle
- Spear
- Bow

And has to eliminate 2 types of enemies

- Cannibals
- Boars

Flowchart / Diagrams

Animation

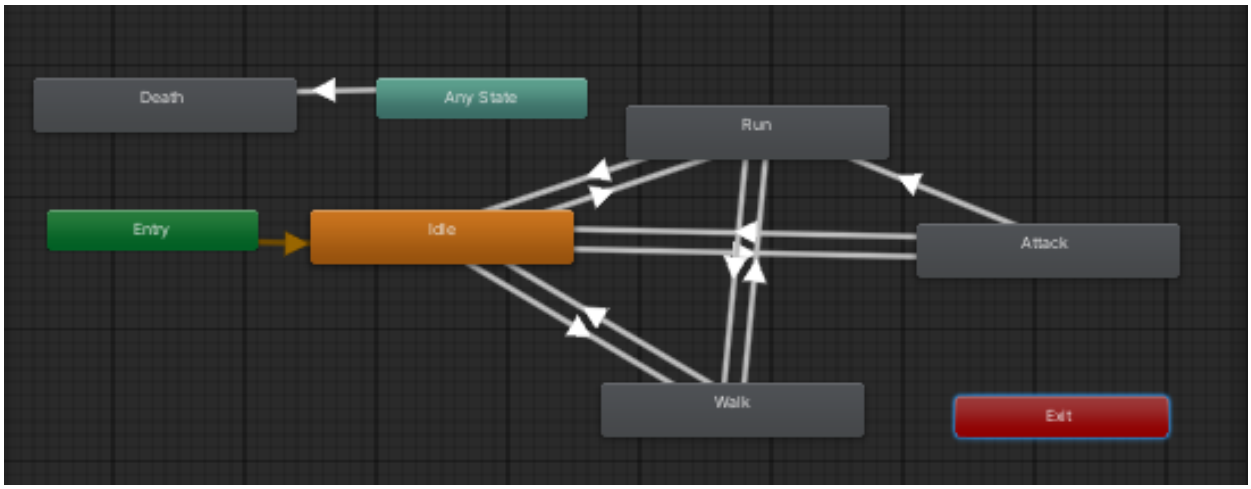

Figure No. 1 : Traversal of player states

Enemies

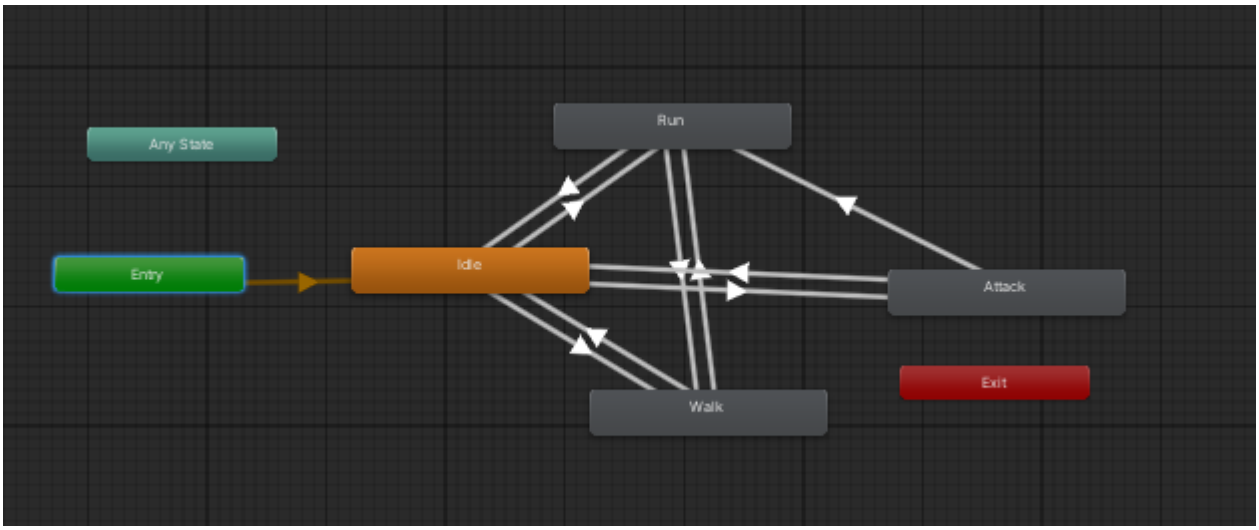

Figure No. 2 : Enemy Attack Animation

Set of Weapons

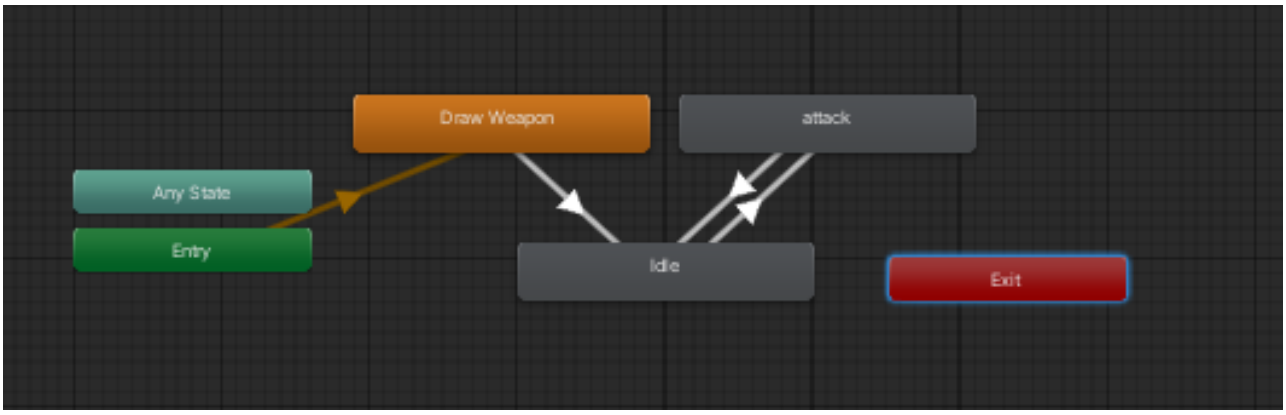

Figure No. 3 : Animation for Axe, Revolver, Shotgun, and Rifle

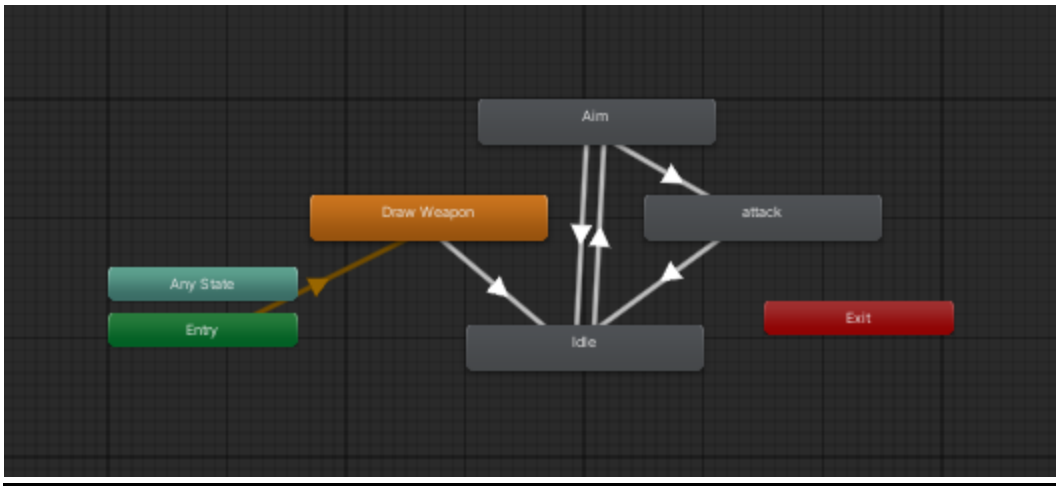

Figure No. 4 : Animation for Spear and Bow

Gameplay Screenshots

Main Menu

Figure No. 5

Options Menu

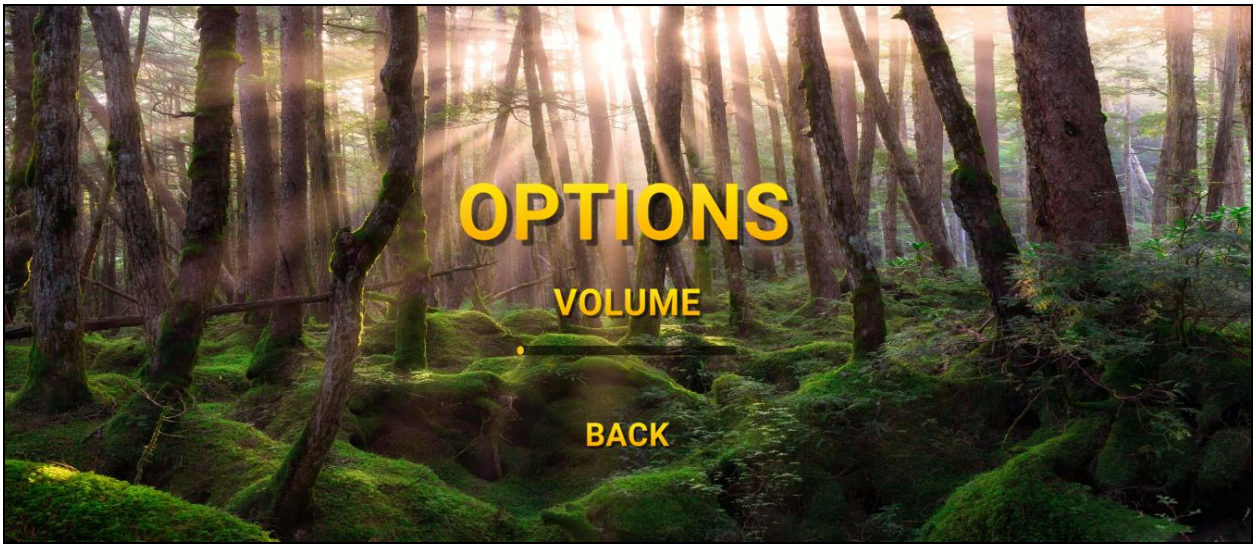

Figure No. 6

Gameplay Screenshots

Start position / Axe

Figure No. 7

Revolver

Figure No. 8

Gameplay Screenshots

Shotgun

Figure No. 9

Assault Rifle

Figure No. 10

Gameplay Screenshots

Bow

Figure No. 11

Spear

Figure No. 12

Enemies' Screenshots

Cannibal

Figure No. 13

Boar

Figure No. 14

Source Code

★ **Menu.cs**
//main menu

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

public class menu : MonoBehaviour
{
    public void PlayGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }

    public void QuitGame()
    {
        Debug.Log("Quit");
        Application.Quit();
    }

}
```

★ **Pmovement.cs**

//player movement

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

public class pmovements : MonoBehaviour
{

    private CharacterController character_Controller;

    private Vector3 move_Direction;

    public float speed = 5f;
    private float gravity = 20f;

    public float jump_Force = 10f;
    private float vertical_Velocity;

    void Awake()
    {
        character_Controller = GetComponent<CharacterController>();
    }

    void Update()
    {
        MoveThePlayer();
    }

    void MoveThePlayer()
    {
        move_Direction = new Vector3(Input.GetAxis(Axis.HORIZONTAL), 0f, Input.Ge
tAxis(Axis.VERTICAL));

        move_Direction = transform.TransformDirection(move_Direction);
        move_Direction *= speed * Time.deltaTime;

        ApplyGravity();

        character_Controller.Move(move_Direction);
    }
```

```
    void ApplyGravity()
    {
        if(character_Controller.isGrounded)
        {
            vertical_Velocity -= gravity * Time.deltaTime;
            PlayerJump();
        }
        else
        {
            vertical_Velocity -= gravity * Time.deltaTime;
        }

        move_Direction.y = vertical_Velocity * Time.deltaTime;

    }
    void PlayerJump()
    {
        if(character_Controller.isGrounded && Input.GetKeyDown(KeyCode.Space))
        {
            vertical_Velocity = jump_Force;
        }
    }
}
```

★ **Mlook.cs**

//camera

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

public class mlook : MonoBehaviour
{
    [SerializeField]
    private Transform playerRoot, lookRoot;

    [SerializeField]
    private bool invert;

    [SerializeField]
    private bool can_Unlock = true;

    [SerializeField]
    private float sensitivity = 5f;

    [SerializeField]
    private int smooth_Steps = 10;

    [SerializeField]
    private float smooth_Weight = 0.4f;

    [SerializeField]
    private float roll_Angle = 10f;

    [SerializeField]
    private float roll_Speed = 3f;

    [SerializeField]
    private Vector2 default_Look_Limits = new Vector2(-70f, 80f);

    private Vector2 look_Angles;

    private Vector2 current_Mouse_look;
    private Vector2 smooth_Move;

    private float current_Roll_Angle;
```

```
    private int last_Look_Frame;




    // Start is called before the first frame update
    void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;
    }

    // Update is called once per frame
    void Update()
    {
        LockAndUnlockCursor();
        if(Cursor.lockState==CursorLockMode.Locked)
        {
            LookAround();
        }
    }

    void LockAndUnlockCursor()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if(Cursor.lockState == CursorLockMode.Locked)
            {
                Cursor.lockState = CursorLockMode.None;
            }
            else
            {
                Cursor.lockState = CursorLockMode.Locked;
                Cursor.visible = false;
            }
        }
    }

    void LookAround()
    {
        current_Mouse_look = new Vector2(Input.GetAxis(MouseAxis.MOUSE_Y), Input.
GetAxis(MouseAxis.MOUSE_X));

        look_Angles.x += current_Mouse_look.x * sensitivity * (invert ? 1f : -
1f);
        look_Angles.y += current_Mouse_look.y * sensitivity;
```

```
        look_Angles.x = Mathf.Clamp(look_Angles.x, default_Look_Limits.x, default
_Look_Limits.y);

        //current_Roll_Angle = Mathf.Lerp(current_Roll_Angle, Input.GetAxisRaw(Mo
useAxis.MOUSE_X) * roll_Angle, Time.deltaTime * roll_Speed);
        //to be implemented in further stages

        lookRoot.localRotation = Quaternion.Euler(look_Angles.x, 0f, 0f);
        playerRoot.localRotation = Quaternion.Euler(0f, look_Angles.y, 0f);
    }

}
```

★ **Sprintcrouch.cs**

//sprint and crouch movement

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

public class sprintcrouch : MonoBehaviour
{

    private pmovements playerMovement;

    public float sprint_Speed = 10f;
    public float move_Speed = 5f;
    public float crouch_Speed = 2f;

    private Transform look_Root;
    private float stand_Height = 1.6f;
    private float crouch_Height = 1f;

    private bool is_Crouching;

    private pfootsteps player_Footsteps;

    private float sprint_Volume = 1f;
```

```
    private float crouch_Volume = 0.1f;
    private float walk_Volume_Min = 0.2f, walk_Volume_Max = 0.6f;

    private float walk_Step_Distance = 0.4f;
    private float sprint_Step_Distance = 0.25f;
    private float crouch_Step_Distance = 0.5f;

    private PlayerStats player_Stats;

    private float sprint_Value = 100f;
    private float sprint_Threshold = 10f;


    void Awake()
    {
        playerMovement = GetComponent<pmovements>();

        look_Root = transform.GetChild(0);

        player_Footsteps = GetComponentInChildren<pfootsteps>();

        player_Stats = GetComponent<PlayerStats>();
    }

    void Start()
    {

        player_Footsteps.volume_Min = walk_Volume_Min;
        player_Footsteps.volume_Max = walk_Volume_Max;
        player_Footsteps.step_Distance = walk_Step_Distance;

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

    void Sprint()
    {
        if(sprint_Value > 0f)
        {
            if(Input.GetKeyDown(KeyCode.LeftShift) && !is_Crouching)
            {
```

```
            playerMovement.speed = sprint_Speed;

            player_Footsteps.step_Distance = sprint_Step_Distance;
            player_Footsteps.volume_Min = sprint_Volume;
            player_Footsteps.volume_Max = sprint_Volume;
        }
    }

    if(Input.GetKeyUp(KeyCode.LeftShift) && !is_Crouching)
    {
        playerMovement.speed = move_Speed;

        player_Footsteps.step_Distance = walk_Step_Distance;
        player_Footsteps.volume_Min = walk_Volume_Min;
        player_Footsteps.volume_Max = walk_Volume_Max;


    }

    if(Input.GetKey(KeyCode.LeftShift) && !is_Crouching)
    {
        sprint_Value -= sprint_Threshold * Time.deltaTime * 1.5f;
        if(sprint_Value <= 0f)
        {
            sprint_Value = 0f;
            playerMovement.speed = move_Speed;
            player_Footsteps.step_Distance = walk_Step_Distance;
            player_Footsteps.volume_Min = walk_Volume_Min;
            player_Footsteps.volume_Max = walk_Volume_Max;
        }

        player_Stats.Display_StaminaStats(sprint_Value);
    }

        else
        {
            if(sprint_Value != 100f)
            {
                sprint_Value += (sprint_Threshold /2f) * Time.deltaTime;
                player_Stats.Display_StaminaStats(sprint_Value);

                if(sprint_Value > 100f)
                {
                    sprint_Value = 100f;
                }
```

```
                }
            }
    }

    void Crouch()
    {
        if(Input.GetKeyDown(KeyCode.C))
        {
            if(is_Crouching)
            {
                look_Root.localPosition = new Vector3(0f, stand_Height, 0f);
                playerMovement.speed = move_Speed;

                player_Footsteps.step_Distance = walk_Step_Distance;
                player_Footsteps.volume_Min = walk_Volume_Min;
                player_Footsteps.volume_Max = walk_Volume_Max;

                is_Crouching = false;
            }
            else
            {
                look_Root.localPosition = new Vector3(0f, crouch_Height, 0f);
                playerMovement.speed = crouch_Speed;

                player_Footsteps.step_Distance = crouch_Step_Distance;
                player_Footsteps.volume_Min = crouch_Volume;
                player_Footsteps.volume_Max = crouch_Volume;
                is_Crouching = true;
            }
        }
    }
}
```

★ **Pfootsteps.cs**
//player footsteps

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

public class pfootsteps : MonoBehaviour
{

    private AudioSource footstep_Sound;

    [SerializeField]
    private AudioClip[] footstep_Clip;

    private CharacterController character_Controller;

    [HideInInspector]
    public float volume_Min, volume_Max;

    private float accumulated_Distance;

    [HideInInspector]
    public float step_Distance;


    void Awake()
    {
        footstep_Sound = GetComponent<AudioSource>();

        character_Controller = GetComponentInParent<CharacterController>();

    }

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

    void CheckToPlayFootStepSound()
    {
        if (!character_Controller.isGrounded)
```

```
            return;

        if(character_Controller.velocity.sqrMagnitude > 0)
        {
            accumulated_Distance += Time.deltaTime;

            if(accumulated_Distance > step_Distance)
            {
                footstep_Sound.volume = Random.Range(volume_Min, volume_Max);
                footstep_Sound.clip = footstep_Clip[Random.Range(0, footstep_Clip
.Length)];
                footstep_Sound.Play();

                accumulated_Distance = 0f;

            }
        }
        else
        {
            accumulated_Distance = 0f;
        }
    }
}
```

★ **AttackScript.cs**

//axe atk

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

public class AttackScript : MonoBehaviour
{
    public float damage = 2f;
    public float radius = 1;
    public LayerMask layerMask;

    void Update ()
    {
        Collider[] hits = Physics.OverlapSphere(transform.position, radius, layer
Mask);
```

```
        if(hits.Length>0)
        {
            hits[0].gameObject.GetComponent<HealthScript>().ApplyDamage(damage);
            gameObject.SetActive(false);
        }
    }
}//class
```

★ **Playerattack.cs**

//atk with all oher weapons

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

public class playerattack : MonoBehaviour
{
    private WeaponManager weapon_Manager;

    public float fireRate = 15f;
    private float nextTimeToFire;
    public float damage = 20f;

    private Animator zoomCameraAnim;
    private bool zoomed;

    private Camera mainCam;

    private GameObject crosshair;

    private bool is_Aiming;

    [SerializeField]
    private GameObject arrow_Prefab, spear_prefab;

    [SerializeField]
    private Transform arrow_Bow_StartPosition;

    void Awake()
    {
```

```
        weapon_Manager = GetComponent<WeaponManager>();

        zoomCameraAnim = transform.Find(Tags.LOOK_ROOT).transform.Find(Tags.ZOOM_
CAMERA).GetComponent<Animator>();

        crosshair = GameObject.FindWithTag(Tags.CROSSHAIR);

        mainCam = Camera.main;
    }


    // Start is called before the first frame update
    void Start()
    {

    }

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

    void WeaponShoot()
    {
        if(weapon_Manager.GetCurrentSelectedWeapon().fireType == WeaponFireType.M
ULTIPLE)
        {
            if(Input.GetMouseButton(0) && Time.time>nextTimeToFire)
            {
                nextTimeToFire = Time.time + 1f / fireRate;

                weapon_Manager.GetCurrentSelectedWeapon().ShootAnimation();
                BulletFired();
            }
        }
        else
        {
            if(Input.GetMouseButtonDown(0))
            {

                //handle axe
                if(weapon_Manager.GetCurrentSelectedWeapon().tag == Tags.AXE_TAG)
                {
```

```
                    weapon_Manager.GetCurrentSelectedWeapon().ShootAnimation();
                }

                //handle shoot
                if(weapon_Manager.GetCurrentSelectedWeapon().bulletType == Weapon
BulletType.BULLET)
                {
                    weapon_Manager.GetCurrentSelectedWeapon().ShootAnimation();
                    BulletFired();
                }
                else
                //arrow or spear
                {
                    if(is_Aiming)
                    {
                        weapon_Manager.GetCurrentSelectedWeapon().ShootAnimation(
);

                        if(weapon_Manager.GetCurrentSelectedWeapon().bulletType =
= WeaponBulletType.ARROW)
                        {
                            ThrowArrowOrSpear(true);
                            //throw arrow
                        }
                        else if (weapon_Manager.GetCurrentSelectedWeapon().bullet
Type == WeaponBulletType.SPEAR)
                        {
                            ThrowArrowOrSpear(false);
                            //throw spear
                        }
                    }
                }
            }//if input get mouse btn 0
        }//else
    }//weapon shoot

    void ZoomInAndOut()
    {
        if(weapon_Manager.GetCurrentSelectedWeapon().weapon_Aim==WeaponAim.AIM)
        {
            if(Input.GetMouseButtonDown(1))
            {
                zoomCameraAnim.Play(AnimationTags.ZOOM_IN_ANIM);
                crosshair.SetActive(false);
            }
```

```
            if (Input.GetMouseButtonUp(1))
            {
                zoomCameraAnim.Play(AnimationTags.ZOOM_OUT_ANIM);
                crosshair.SetActive(true);
            }
        }// if we need to zoom te weapon

        if(weapon_Manager.GetCurrentSelectedWeapon().weapon_Aim == WeaponAim.SELF
_AIM)
        {
            if(Input.GetMouseButtonDown(1))
            {
                weapon_Manager.GetCurrentSelectedWeapon().Aim(true);
                is_Aiming = true;

            }
            if (Input.GetMouseButtonUp(1))
            {
                weapon_Manager.GetCurrentSelectedWeapon().Aim(false);
                is_Aiming = false;

            }
        }



    }//zoom in and out

    void ThrowArrowOrSpear(bool throwArrow)
    {
        if (throwArrow)
        {
            GameObject arrow = Instantiate(arrow_Prefab);
            arrow.transform.position = arrow_Bow_StartPosition.position;
            arrow.GetComponent<ArrowBowScript>().Launch(mainCam);
        }
        else
        {
            GameObject spear = Instantiate(spear_prefab);
            spear.transform.position = arrow_Bow_StartPosition.position;
            spear.GetComponent<ArrowBowScript>().Launch(mainCam);
        }
    }
    void BulletFired()
```

```
    {
        RaycastHit hit;

        if(Physics.Raycast(mainCam.transform.position,mainCam.transform.forward,o
ut hit))
        {
            if(hit.transform.tag == Tags.ENEMY_TAG)
            {
                hit.transform.GetComponent<HealthScript>().ApplyDamage(damage);
            }
        }
    }
}//class
```

★ **PlayerAxeWooshSound.cs**

//sound for axe swing

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

public class PlayerAxeWooshSound : MonoBehaviour
{
    [SerializeField]
    private AudioSource audioSource;
    [SerializeField]
    private AudioClip[] woosh_Sounds;

    void PlayWooshSound()
    {
        audioSource.clip = woosh_Sounds[Random.Range(0,woosh_Sounds.Length)];
        audioSource.Play();
    }

}
```

★ **HealthScript.cs**

//script for enemy and player health

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

public class HealthScript : MonoBehaviour
{

    private EnemyAnimator enemy_Anim;
    private NavMeshAgent navAgent;
    private EnemyController enemy_Controller;
    public float health = 100;
    public bool is_Player, is_Boar, is_Cannibal;

    private bool is_Dead;

    private EnemyAudio enemyAudio;

    private PlayerStats player_Stats;

    void Awake()
    {
        if(is_Boar || is_Cannibal)
        {
            enemy_Anim = GetComponent<EnemyAnimator>();
            enemy_Controller = GetComponent<EnemyController>();
            navAgent = GetComponent<NavMeshAgent>();

            enemyAudio=GetComponentInChildren<EnemyAudio>();
        }
        if(is_Player)
        {
            player_Stats = GetComponent<PlayerStats>();
        }
    }



    public void ApplyDamage(float damage)
    {
```

```
        if(is_Dead)
        return;

        health -=damage;
        if(is_Player)
        {
            player_Stats.Display_HealthStats(health);
        }

        if(is_Boar||is_Cannibal)
        {
            if(enemy_Controller.Enemy_State ==EnemyState.PATROL)
            {
                enemy_Controller.chase_Distance = 50f;
            }
        }

    if(health <= 0)
    {
        PlayerDied();
        is_Dead = true;
    }// apply damage
}

void PlayerDied()
    {
        if(is_Cannibal)
        {
            GetComponent<Animator>().enabled = false;
            GetComponent<BoxCollider>().isTrigger = false;
            GetComponent<Rigidbody>().AddTorque(-transform.forward*50f);

            enemy_Controller.enabled = false;
            navAgent.enabled = false;
            enemy_Anim.enabled = false;

            StartCoroutine(DeadSound());

            EnemyManager.instance.EnemyDied(true);
        }

        if(is_Boar)
        {
            navAgent.velocity = Vector3.zero;
            navAgent.isStopped=true;
```

```
            enemy_Controller.enabled = false;

            enemy_Anim.Dead();

            StartCoroutine(DeadSound());
            EnemyManager.instance.EnemyDied(false);
        }

        if(is_Player)
        {
            GameObject[] enemies = GameObject.FindGameObjectsWithTag(Tags.ENEMY_T
AG);

            for(int i = 0; i<enemies.Length; i++)
            {
                enemies[i].GetComponent<EnemyController>().enabled = false;
            }

            EnemyManager.instance.StopSpawning();
            GetComponent<pmovements>().enabled = false;
            GetComponent<playerattack>().enabled = false;
            GetComponent<WeaponManager>().GetCurrentSelectedWeapon().gameObject.S
etActive(false);


        }

        if(tag == Tags.PLAYER_TAG)
        {
            Invoke("RestartGame",3f);
        }
        else
        {
            Invoke("TurnOffGameObject",3f);
        }

    }//player died

    void RestartGame()
    {
    UnityEngine.SceneManagement.SceneManager.LoadScene("SampleScene");
    }

    void TurnOffGameObject()
    {
```

```
        gameObject.SetActive(false);
    }
    IEnumerator DeadSound()
    {
        yield return new WaitForSeconds(0.3f);
        enemyAudio.Play_DeadSound();
    }
}
```

★ **PlayerStats.cs**

//keep track and display health and stamina

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


public class PlayerStats : MonoBehaviour
{
    [SerializeField]
    private Image health_Stats, stamina_Stats;


    public void Display_HealthStats(float healthValue)
    {
        healthValue /= 100f;
        health_Stats.fillAmount = healthValue;
    }

    public void Display_StaminaStats(float staminaValue)
    {
        staminaValue /= 100f;
        stamina_Stats.fillAmount = staminaValue;
    }
}//class
```

New title weapon scripts

★ **ArrowBowScript.cs**
//script for bow and spear

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

public class ArrowBowScript : MonoBehaviour
{

    private Rigidbody myBody;

    public float speed = 30f;

    public float deactivate_Timer = 3f;

    public float damage = 15f;

    void Awake()
    {
        myBody = GetComponent<Rigidbody>();
    }

    // Start is called before the first frame update
    void Start()
    {
       Invoke("DeactivateGameObject", deactivate_Timer);
    }

    public void Launch(Camera mainCamera)
    {
        myBody.velocity = mainCamera.transform.forward * speed;

        transform.LookAt(transform.position + myBody.velocity);
    }
    void DeactivateGameObject()
    {
        if(gameObject.activeInHierarchy)
        {
            gameObject.SetActive(false);
        }
```

```
    }

    void OnTriggerEnter(Collider target)
    {
        //after we touch a target deactivate arrow
    }
}//class
```

★ **WeaponHandler.cs**

//activate hitbox when attacking and sounds

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

public enum WeaponAim
{
    NONE,
    SELF_AIM,
    AIM
}

public enum WeaponFireType
{
    SINGLE,
    MULTIPLE
}

public enum WeaponBulletType
{
    BULLET,
    ARROW,
    SPEAR,
    NONE
}


public class WeaponHandler : MonoBehaviour
{

    private Animator anim;
```

```
public WeaponAim weapon_Aim;

[SerializeField]
private GameObject muzzleFlash;

[SerializeField]
private AudioSource shootSound, reload_Sound;

public WeaponFireType fireType;

public WeaponBulletType bulletType;

public GameObject attack_Point;

void Awake()
{
    anim = GetComponent<Animator>();
}

public void ShootAnimation()
{
    anim.SetTrigger(AnimationTags.SHOOT_TRIGGER);
}

public void Aim(bool canAim)
{
    anim.SetBool(AnimationTags.AIM_PARAMETER, canAim);
}

void Turn_On_MuzzleFlash()
{
    muzzleFlash.SetActive(true);
}

void Turn_Off_MuzzleFlash()
{
    muzzleFlash.SetActive(false);
}

void Play_ShootSound()
{
    shootSound.Play();
}
```

```
    void Play_ReloadSound()
    {
        reload_Sound.Play();
    }

    void Turn_On_AttackPoint()
    {
        attack_Point.SetActive(true);
    }

    void Turn_Off_AttackPoint()
    {
        if(attack_Point.activeInHierarchy)
        {
            attack_Point.SetActive(false);
        }
    }
}
```

★ **WeaponManager.cs**
//select weapons

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

public class WeaponManager : MonoBehaviour
{

    [SerializeField]
    private WeaponHandler[] weapons;

    private int current_Weapon_Index;


    // Start is called before the first frame update
    void Start()
    {
        current_Weapon_Index = 0;
        weapons[current_Weapon_Index].gameObject.SetActive(true);
    }
```

```
    // Update is called once per frame
    void Update()
    {
        if(Input.GetKeyDown(KeyCode.Alpha1))
        {
            TurnOnSelectedWeapon(0);
        }
        if (Input.GetKeyDown(KeyCode.Alpha2))
        {
            TurnOnSelectedWeapon(1);
        }
        if (Input.GetKeyDown(KeyCode.Alpha3))
        {
            TurnOnSelectedWeapon(2);
        }
        if (Input.GetKeyDown(KeyCode.Alpha4))
        {
            TurnOnSelectedWeapon(3);
        }
        if (Input.GetKeyDown(KeyCode.Alpha5))
        {
            TurnOnSelectedWeapon(4);
        }
        if (Input.GetKeyDown(KeyCode.Alpha6))
        {
            TurnOnSelectedWeapon(5);
        }

            }
    void TurnOnSelectedWeapon(int weaponIndex)
    {
        if(current_Weapon_Index == weaponIndex)
        {
            return;
        }
        weapons[current_Weapon_Index].gameObject.SetActive(false);
        weapons[weaponIndex].gameObject.SetActive(true);
        current_Weapon_Index = weaponIndex;
    }
    public WeaponHandler GetCurrentSelectedWeapon()
    {
        return weapons[current_Weapon_Index];
    }

}
```

New title Enemy scripts

★ **EnemyController.cs**
//control enemy states

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

public enum EnemyState{
    PATROL,
    CHASE,
    ATTACK
}
public class EnemyController : MonoBehaviour
{

    private EnemyAnimator enemy_Anim;
    private NavMeshAgent navAgent;

    private EnemyState enemy_State;

    public float walk_Speed = 0.5f;
    public float run_Speed = 4f;
    public float chase_Distance = 7f;
    private float current_Chase_Distance;
    public float attack_Distance = 1.8f;
    public float chase_After_Attack_Distance = 2f;

    public float patrol_Radius_Min = 20f, patrol_Radius_Max = 60f;
    public float patrol_For_This_Time = 15f;
    private float patrol_Timer;
    public float wait_Before_Attack = 2f;
    private float attack_Timer;

    private Transform target;

    public GameObject attack_Point;

    private EnemyAudio enemy_Audio;
    void Awake()
```

```
{
    enemy_Anim = GetComponent<EnemyAnimator>();
    navAgent= GetComponent<NavMeshAgent>();

    target = GameObject.FindWithTag(Tags.PLAYER_TAG).transform;

    enemy_Audio = GetComponentInChildren<EnemyAudio>();

}

// Start is called before the first frame update
void Start()
{
    enemy_State = EnemyState.PATROL;
    patrol_Timer = patrol_For_This_Time;
    attack_Timer = wait_Before_Attack;
    current_Chase_Distance = chase_Distance;

}

// Update is called once per frame
void Update()
{
    if(enemy_State == EnemyState.PATROL)
    {
        Patrol();
    }

    if(enemy_State == EnemyState.CHASE)
    {
        Chase();
    }

    if(enemy_State == EnemyState.ATTACK)
    {
        Attack();
    }
}
void Patrol()
{
    navAgent.isStopped = false;
    navAgent.speed = walk_Speed;

    patrol_Timer += Time.deltaTime;
```

```
        if(patrol_Timer>patrol_For_This_Time)
        {
            SetNewRandomDestination();
            patrol_Timer = 0f;
        }

        if(navAgent.velocity.sqrMagnitude>0)
        {
            enemy_Anim.walk(true);
        }
        else
        {
            enemy_Anim.walk(false);
        }

        //test distance between enemy and player
        if(Vector3.Distance(transform.position,target.position)<=chase_Distance)
        {
            enemy_Anim.walk(false);
            enemy_State = EnemyState.CHASE;
            enemy_Audio.Play_ScreamSound();
        }

    }
    void Chase()
    {
        //enable agent to move again
        navAgent.isStopped = false;
        navAgent.speed = run_Speed;
        navAgent.SetDestination(target.position);
        //set player pos as enemy target
        if(navAgent.velocity.sqrMagnitude>0)
        {
            enemy_Anim.run(true);
        }
        else
        {
            enemy_Anim.run(false);
        }

        if(Vector3.Distance(transform.position, target.position) <= attack_Distan
ce)
        {
            enemy_Anim.run(false);
            enemy_Anim.walk(false);
```

```
            enemy_State = EnemyState.ATTACK;

            //reset the chase distance to previous one
            if(chase_Distance != current_Chase_Distance)
            {
                chase_Distance = current_Chase_Distance;

            }
            else if (Vector3.Distance(transform.position, target.position)>chase_
Distance)
            {
                //player run away fom enemy

                //stop running
                enemy_Anim.run(false);
                enemy_State = EnemyState.PATROL;

                //reset the patrol timer
                //can calculate the new patrol destination right away
                patrol_Timer = patrol_For_This_Time;

                if(chase_Distance != current_Chase_Distance)
                {
                    chase_Distance=current_Chase_Distance;
                }



            }

        }
    }
    void Attack()
    {
        navAgent.velocity = Vector3.zero;
        navAgent.isStopped = true;

        attack_Timer += Time.deltaTime;

        if(attack_Timer > wait_Before_Attack)
        {
            enemy_Anim.Attack();
            attack_Timer = 0f;

            enemy_Audio.Play_AttackSound();
```

```
        }

        if(Vector3.Distance(transform.position, target.position) > attack_Distanc
e + chase_After_Attack_Distance)
        {
            enemy_State = EnemyState.CHASE;

        }

    }

    void SetNewRandomDestination()
    {
        float rand_Radius = Random.Range(patrol_Radius_Min, patrol_Radius_Max);
        Vector3 randDir = Random.insideUnitSphere * rand_Radius;
        randDir += transform.position;

        NavMeshHit navHit;
        NavMesh.SamplePosition(randDir,out navHit,rand_Radius,-1);
        navAgent.SetDestination(navHit.position);
    }

void Turn_On_AttackPoint()
    {
        attack_Point.SetActive(true);
    }

    void Turn_Off_AttackPoint()
    {
        if(attack_Point.activeInHierarchy)
        {
            attack_Point.SetActive(false);
        }
    }

    public EnemyState Enemy_State
    {
        get; set;
    }

}
```

★ **EnemyAnimator.cs**

//control enemy animations

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

public class EnemyAnimator : MonoBehaviour
{

    private Animator anim;


    // Start is called before the first frame update
    void Awake()
    {
        anim=GetComponent<Animator>();
    }

    public void walk(bool walk)
    {
        anim.SetBool(AnimationTags.WALK_PARAMETER, walk);
    }

    public void run(bool run)
    {
        anim.SetBool(AnimationTags.RUN_PARAMETER, run);
    }


    public void Attack()
    {
        anim.SetTrigger(AnimationTags.ATTACK_TRIGGER);
    }

    public void Dead()
    {
        anim.SetTrigger(AnimationTags.DEAD_TRIGGER);
    }

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

```
    }
}
```

★ **EnemyAudio.cs**

//enemy sounds

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

public class EnemyAudio : MonoBehaviour
{
    private AudioSource audioSource;
    [SerializeField]
    private AudioClip scream_Clip, die_Clip;

    [SerializeField]
    private AudioClip[] attack_Clips;

    void Awake()
    {
        audioSource = GetComponent<AudioSource>();

    }

    public void Play_ScreamSound()
    {
        audioSource.clip = scream_Clip;
        audioSource.Play();
    }

    public void Play_AttackSound()
    {
        audioSource.clip = attack_Clips[Random.Range(0,attack_Clips.Length)];
        audioSource.Play();
    }

     public void Play_DeadSound()
    {
        audioSource.clip = die_Clip;
```

```
        audioSource.Play();
    }


}
```

★ **EnemyManager.cs**
//spawning enemies

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

public class EnemyManager : MonoBehaviour
{
    public static EnemyManager instance;

    [SerializeField]
    private GameObject boar_Prefab, cannibal_Prefab;

    public Transform[] cannibal_SpawnPoints, boar_SpawnPoints;

    [SerializeField]
    private int cannibal_Enemy_Count, boar_Enemy_Count;
    private int initial_Cannibal_Count, initial_Boar_Count;

    public float wait_Before_Spawn_Enemies_Time = 10f;

    void Awake()
    {
        MakeInstance();
    }

    void Start()
    {
        initial_Cannibal_Count = cannibal_Enemy_Count;
        initial_Boar_Count = boar_Enemy_Count;

        SpawnEnemies();

        StartCoroutine("CheckToSpawnEnemies");

    }
```

```
    void MakeInstance()
    {
        if(instance == null)
        {
            instance = this;
        }
    }

    void SpawnEnemies()
    {
        SpawnCannibals();
        SpawnBoars();
    }

    void SpawnCannibals()
    {
        int index = 0;

        if(index >= cannibal_SpawnPoints.Length)
        {
            index = 0;
        }

        for (int i=0;i<cannibal_Enemy_Count;i++)
        {
            Instantiate(cannibal_Prefab,cannibal_SpawnPoints[index].position,Quat
ernion.identity);
            index++;
        }
        cannibal_Enemy_Count = 0;
    }

    void SpawnBoars()
    {
        int index = 0;

        if(index >= boar_SpawnPoints.Length)
        {
            index = 0;
        }

        for (int i=0;i<boar_Enemy_Count;i++)
        {
            Instantiate(boar_Prefab,boar_SpawnPoints[index].position,Quaternion.i
dentity);
```

```
            index++;
        }
        boar_Enemy_Count = 0;
    }

    IEnumerator CheckToSpawnEnemies()
    {
        yield return new WaitForSeconds(wait_Before_Spawn_Enemies_Time);
        SpawnCannibals();
        SpawnBoars();
        StartCoroutine("CheckToSpawnEnemies");
    }

    public void EnemyDied(bool cannibal)
    {
        if(cannibal)
        {
            cannibal_Enemy_Count++;
            if(cannibal_Enemy_Count>initial_Cannibal_Count)
            {
                cannibal_Enemy_Count = initial_Cannibal_Count;
            }
        }
        else
        {
            boar_Enemy_Count++;
            if(boar_Enemy_Count>initial_Boar_Count)
            {
                boar_Enemy_Count = initial_Boar_Count;
            }
        }
    }

    public void StopSpawning()
    {
        StopCoroutine("CheckToSpawnEnemies");
    }


}
```

New title helper script

★ **Taghelp.cs**

//some tags to help for faster coding

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Axis
{
    public const string HORIZONTAL = "Horizontal";
    public const string VERTICAL = "Vertical";
}
public class MouseAxis
{
    public const string MOUSE_X = "Mouse X";
    public const string MOUSE_Y = "Mouse Y";
}

public class AnimationTags
{
    public const string ZOOM_IN_ANIM = "ZoomIn";
    public const string ZOOM_OUT_ANIM = "ZoomOut";

    public const string SHOOT_TRIGGER = "Shoot";
    public const string AIM_PARAMETER = "Aim";

    public const string WALK_PARAMETER = "Walk";
    public const string RUN_PARAMETER = "Run";
    public const string ATTACK_TRIGGER = "Attack";
    public const string DEAD_TRIGGER = "Dead";
}
public class Tags
{
    public const string LOOK_ROOT = "Look Root";
    public const string ZOOM_CAMERA = "FP Camera";
    public const string CROSSHAIR = "Crosshair";
    public const string ARROW_TAG = "Arrow";

    public const string AXE_TAG = "Axe";

    public const string PLAYER_TAG = "Player";
    public const string ENEMY_TAG = "Enemy";
}
```

Future Development

This project is far from being completed. There are quite a few ideas that can be implemented in the current scenario

My further plans with this project is to implement a Story mode or an objective oriented mechanic which makes the game more interactive and interesting

New weapons and enemies may also be developed depending on the assets available including a final boss.

References

- www.google.com
- www.stackoverflow.com
- www.unity.com
- www.unity3d.com
- www.youtube.com