Project Scripts

About

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

public class About : MonoBehaviour
{
    public void PlayGame()
    {
        SceneManager.LoadScene("Classroom");
    }
    public void QuitGame()
    {
        Application.Quit();
    }
    public void Menu()
    {
        SceneManager.LoadScene("MainMenu");
    }

}
```

DialogueManager

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

public class DialogueManager : MonoBehaviour
{
    public NPC npc;

    bool isTalking = false;

    float distance;
    float curResponseTracker = 0;

    public GameObject player;
    public GameObject dialogueUI;

    public Text npcName;
    public Text npcDialogueBox;
    public Text playerResponse;


    // Start is called before the first frame update
    void Start()
    {
        dialogueUI.SetActive(false);
```

```
}
void Update()
{
    if (Input.GetKeyDown(KeyCode.Escape))
    {
        Application.Quit();
    }
}
void OnMouseOver()
{
    if(Input.GetAxis("Mouse ScrollWheel") < 0f)
    {
        curResponseTracker++;
        if (curResponseTracker >= npc.playerDialogue.Length - 1)
        {
            curResponseTracker = npc.playerDialogue.Length - 1;

        }
    }
    else if(Input.GetAxis("Mouse ScrollWheel") > 0f)
    {
        curResponseTracker--;
        if(curResponseTracker < 0f)
        {
            curResponseTracker = 0;
        }
    }


    distance = Vector3.Distance(player.transform.position, this.transform.position);
    if(distance <= 30f)
    {
        //trigger dialogue
        if(Input.GetKeyDown(KeyCode.E) && isTalking==false)
        {
            StartConversation();
        }
        else if (Input.GetKeyDown(KeyCode.E) && isTalking == true)
        {
            EndDialogue();
        }
        if(curResponseTracker == 0 && npc.playerDialogue.Length >= 0)
        {
            playerResponse.text = npc.playerDialogue[0];
            if(Input.GetKeyDown(KeyCode.Return))
            {
                npcDialogueBox.text = npc.dialogue[1];
            }
        }
        else if (curResponseTracker == 1 && npc.playerDialogue.Length >= 1)
        {
            playerResponse.text = npc.playerDialogue[1];
            if (Input.GetKeyDown(KeyCode.Return))
            {
                npcDialogueBox.text = npc.dialogue[2];
            }
        }
        else if (curResponseTracker == 2 && npc.playerDialogue.Length >= 2)
```

```
            {
                playerResponse.text = npc.playerDialogue[2];
                if (Input.GetKeyDown(KeyCode.Return))
                {
                    npcDialogueBox.text = npc.dialogue[3];
                }
            }
        }
    }

    void StartConversation()
    {
        isTalking = true;
        curResponseTracker = 0;
        dialogueUI.SetActive(true);
        npcName.text = npc.name;
        npcDialogueBox.text = npc.dialogue[0];
    }

    void EndDialogue()
    {
        isTalking = false;
        dialogueUI.SetActive(false);
    }

}
```

GameMenu

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

public class GameMenu : MonoBehaviour
{
    public void PlayGame()
    {
        SceneManager.LoadScene("Classroom");
    }
    public void QuitGame()
    {
        Application.Quit();
    }
    public void About()
    {
        SceneManager.LoadScene("About");
    }

}
```

NPC

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

[CreateAssetMenu(fileName = "NPC file", menuName = "NPC Files Archive")]
public class NPC : ScriptableObject
{
    public string name;
    [TextArea(3, 15)]
    public string[] dialogue;
    [TextArea(3, 15)]
    public string[] playerDialogue;

}
```

Player

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

public class Player : MonoBehaviour
{
    private GameObject triggeringNpc;
    private bool triggering;

    public GameObject npcText;

    void Start()
    {

    }
    void Update()
    {
        if (triggering)
        {
            npcText.SetActive(true);
            if (Input.GetKeyDown(KeyCode.E))
            {
                print("congratulations");
            }
        }
        else
        {
            npcText.SetActive(false);
        }
    }

    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "NPC")
        {
            triggering = true;
            triggeringNpc = other.gameObject;
        }
```

```
    }

    void OnTriggerExit(Collider other)
    {
        if (other.tag == "NPC")
        {
            triggering = false;
            triggeringNpc = null;
        }
    }
}
```

First Person Controller

```
using System;
using UnityEngine;
using UnityStandardAssets.CrossPlatformInput;
using UnityStandardAssets.Utility;
using Random = UnityEngine.Random;

#pragma warning disable 618, 649
namespace UnityStandardAssets.Characters.FirstPerson
{
    [RequireComponent(typeof (CharacterController))]
    [RequireComponent(typeof (AudioSource))]
    public class FirstPersonController : MonoBehaviour
    {
        [SerializeField] private bool m_IsWalking;
        [SerializeField] private float m_WalkSpeed;
        [SerializeField] private float m_RunSpeed;
        [SerializeField] [Range(0f, 1f)] private float m_RunstepLenghten;
        [SerializeField] private float m_JumpSpeed;
        [SerializeField] private float m_StickToGroundForce;
        [SerializeField] private float m_GravityMultiplier;
        [SerializeField] private MouseLook m_MouseLook;
        [SerializeField] private bool m_UseFovKick;
        [SerializeField] private FOVKick m_FovKick = new FOVKick();
        [SerializeField] private bool m_UseHeadBob;
        [SerializeField] private CurveControlledBob m_HeadBob = new CurveControlledBob();
        [SerializeField] private LerpControlledBob m_JumpBob = new LerpControlledBob();
        [SerializeField] private float m_StepInterval;
        [SerializeField] private AudioClip[] m_FootstepSounds;    // an array of footstep
sounds that will be randomly selected from.
        [SerializeField] private AudioClip m_JumpSound;           // the sound played
when character leaves the ground.
        [SerializeField] private AudioClip m_LandSound;           // the sound played
when character touches back on ground.

        private Camera m_Camera;
        private bool m_Jump;
        private float m_YRotation;
        private Vector2 m_Input;
        private Vector3 m_MoveDir = Vector3.zero;
        private CharacterController m_CharacterController;
        private CollisionFlags m_CollisionFlags;
        private bool m_PreviouslyGrounded;
        private Vector3 m_OriginalCameraPosition;
```

```
private float m_StepCycle;
private float m_NextStep;
private bool m_Jumping;
private AudioSource m_AudioSource;

// Use this for initialization
private void Start()
{
    m_CharacterController = GetComponent<CharacterController>();
    m_Camera = Camera.main;
    m_OriginalCameraPosition = m_Camera.transform.localPosition;
    m_FovKick.Setup(m_Camera);
    m_HeadBob.Setup(m_Camera, m_StepInterval);
    m_StepCycle = 0f;
    m_NextStep = m_StepCycle/2f;
    m_Jumping = false;
    m_AudioSource = GetComponent<AudioSource>();
            m_MouseLook.Init(transform , m_Camera.transform);
}


// Update is called once per frame
private void Update()
{
    RotateView();
    // the jump state needs to read here to make sure it is not missed
    if (!m_Jump)
    {
        m_Jump = CrossPlatformInputManager.GetButtonDown("Jump");
    }

    if (!m_PreviouslyGrounded && m_CharacterController.isGrounded)
    {
        StartCoroutine(m_JumpBob.DoBobCycle());
        PlayLandingSound();
        m_MoveDir.y = 0f;
        m_Jumping = false;
    }
    if (!m_CharacterController.isGrounded && !m_Jumping && m_PreviouslyGrounded)
    {
        m_MoveDir.y = 0f;
    }

    m_PreviouslyGrounded = m_CharacterController.isGrounded;
}


private void PlayLandingSound()
{
    m_AudioSource.clip = m_LandSound;
    m_AudioSource.Play();
    m_NextStep = m_StepCycle + .5f;
}


private void FixedUpdate()
{
    float speed;
```

```
            GetInput(out speed);
            // always move along the camera forward as it is the direction that it being
aimed at
            Vector3 desiredMove = transform.forward*m_Input.y +
transform.right*m_Input.x;

            // get a normal for the surface that is being touched to move along it
            RaycastHit hitInfo;
            Physics.SphereCast(transform.position, m_CharacterController.radius,
Vector3.down, out hitInfo,
                               m_CharacterController.height/2f, Physics.AllLayers,
QueryTriggerInteraction.Ignore);
            desiredMove = Vector3.ProjectOnPlane(desiredMove, hitInfo.normal).normalized;

            m_MoveDir.x = desiredMove.x*speed;
            m_MoveDir.z = desiredMove.z*speed;


            if (m_CharacterController.isGrounded)
            {
                m_MoveDir.y = -m_StickToGroundForce;

                if (m_Jump)
                {
                    m_MoveDir.y = m_JumpSpeed;
                    PlayJumpSound();
                    m_Jump = false;
                    m_Jumping = true;
                }
            }
            else
            {
                m_MoveDir += Physics.gravity*m_GravityMultiplier*Time.fixedDeltaTime;
            }
            m_CollisionFlags = m_CharacterController.Move(m_MoveDir*Time.fixedDeltaTime);

            ProgressStepCycle(speed);
            UpdateCameraPosition(speed);

            m_MouseLook.UpdateCursorLock();
        }


        private void PlayJumpSound()
        {
            m_AudioSource.clip = m_JumpSound;
            m_AudioSource.Play();
        }


        private void ProgressStepCycle(float speed)
        {
            if (m_CharacterController.velocity.sqrMagnitude > 0 && (m_Input.x != 0 ||
m_Input.y != 0))
            {
                m_StepCycle += (m_CharacterController.velocity.magnitude +
(speed*(m_IsWalking ? 1f : m_RunstepLenghten)))*
                               Time.fixedDeltaTime;
```

```
            }

            if (!(m_StepCycle > m_NextStep))
            {
                return;
            }

            m_NextStep = m_StepCycle + m_StepInterval;

            PlayFootStepAudio();
        }


        private void PlayFootStepAudio()
        {
            if (!m_CharacterController.isGrounded)
            {
                return;
            }
            // pick & play a random footstep sound from the array,
            // excluding sound at index 0
            int n = Random.Range(1, m_FootstepSounds.Length);
            m_AudioSource.clip = m_FootstepSounds[n];
            m_AudioSource.PlayOneShot(m_AudioSource.clip);
            // move picked sound to index 0 so it's not picked next time
            m_FootstepSounds[n] = m_FootstepSounds[0];
            m_FootstepSounds[0] = m_AudioSource.clip;
        }


        private void UpdateCameraPosition(float speed)
        {
            Vector3 newCameraPosition;
            if (!m_UseHeadBob)
            {
                return;
            }
            if (m_CharacterController.velocity.magnitude > 0 &&
m_CharacterController.isGrounded)
            {
                m_Camera.transform.localPosition =
                    m_HeadBob.DoHeadBob(m_CharacterController.velocity.magnitude +
                                        (speed*(m_IsWalking ? 1f : m_RunstepLenghten)));
                newCameraPosition = m_Camera.transform.localPosition;
                newCameraPosition.y = m_Camera.transform.localPosition.y -
m_JumpBob.Offset();
            }
            else
            {
                newCameraPosition = m_Camera.transform.localPosition;
                newCameraPosition.y = m_OriginalCameraPosition.y - m_JumpBob.Offset();
            }
            m_Camera.transform.localPosition = newCameraPosition;
        }


        private void GetInput(out float speed)
        {
```

```
            // Read input
            float horizontal = CrossPlatformInputManager.GetAxis("Horizontal");
            float vertical = CrossPlatformInputManager.GetAxis("Vertical");

            bool waswalking = m_IsWalking;

#if !MOBILE_INPUT
            // On standalone builds, walk/run speed is modified by a key press.
            // keep track of whether or not the character is walking or running
            m_IsWalking = !Input.GetKey(KeyCode.LeftShift);
#endif
            // set the desired speed to be walking or running
            speed = m_IsWalking ? m_WalkSpeed : m_RunSpeed;
            m_Input = new Vector2(horizontal, vertical);

            // normalize input if it exceeds 1 in combined length:
            if (m_Input.sqrMagnitude > 1)
            {
                m_Input.Normalize();
            }

            // handle speed change to give an fov kick
            // only if the player is going to a run, is running and the fovkick is to be
used
            if (m_IsWalking != waswalking && m_UseFovKick &&
m_CharacterController.velocity.sqrMagnitude > 0)
            {
                StopAllCoroutines();
                StartCoroutine(!m_IsWalking ? m_FovKick.FOVKickUp() :
m_FovKick.FOVKickDown());
            }
        }


        private void RotateView()
        {
            m_MouseLook.LookRotation (transform, m_Camera.transform);
        }


        private void OnControllerColliderHit(ControllerColliderHit hit)
        {
            Rigidbody body = hit.collider.attachedRigidbody;
            //dont move the rigidbody if the character is on top of it
            if (m_CollisionFlags == CollisionFlags.Below)
            {
                return;
            }

            if (body == null || body.isKinematic)
            {
                return;
            }
            body.AddForceAtPosition(m_CharacterController.velocity*0.1f, hit.point,
ForceMode.Impulse);
        }
    }
}
```