1. How hard or easy did you find this lab?

   It was hard to get into the new input system, other than that everything was fairly easy enough.

2. Have you ever worked in Unity before?

   Yes, but only in 2d.

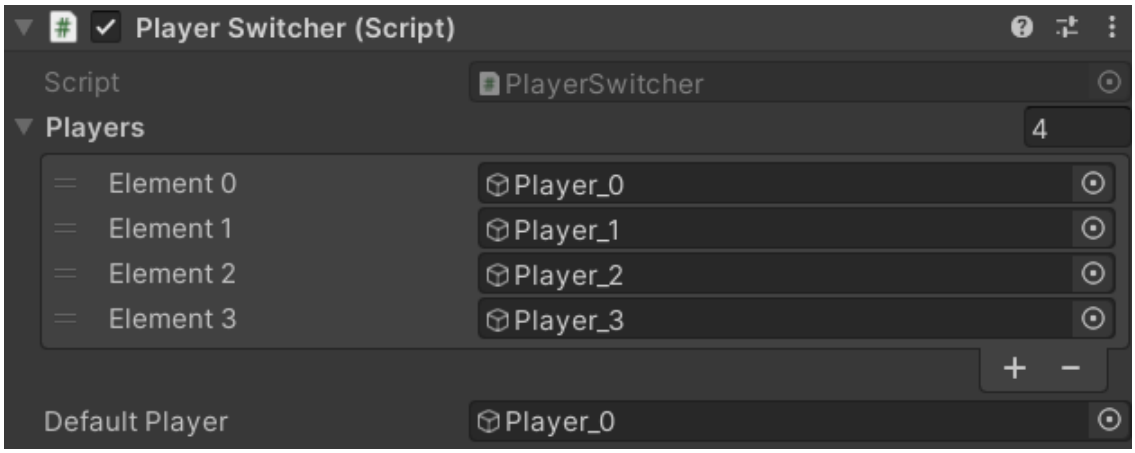
3. Have you programmed in C# much?

   I only programmed in C# in CSE 3902.

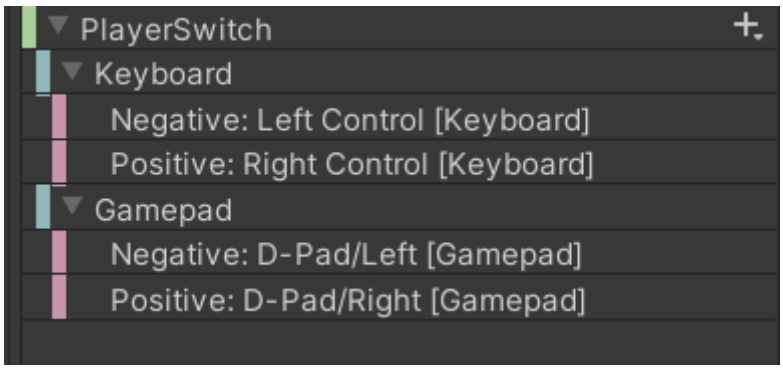
4. Did you do any of the extra credit? If so, explain in detail and include images.

   Yes.

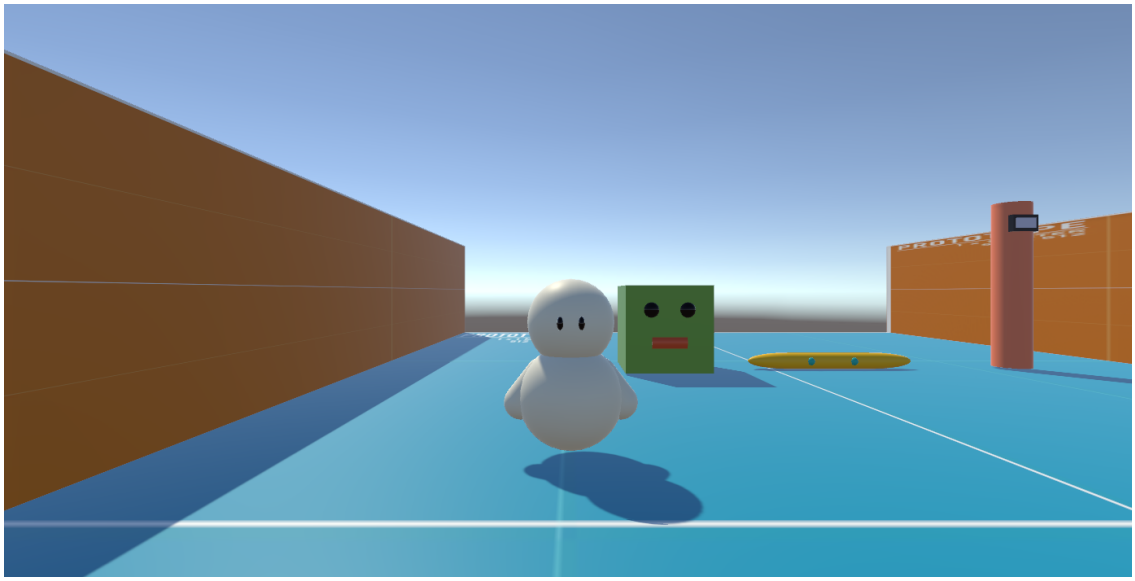
   I created a new module that handles player switch.

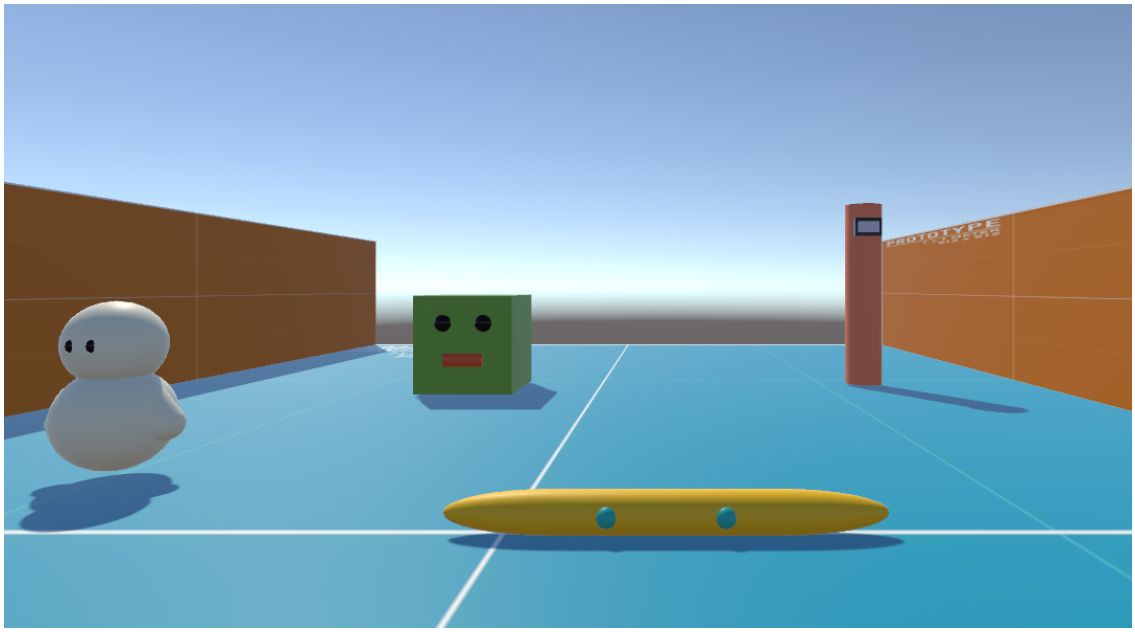


   New input bindings:



   I chose option 2 (Have all characters visible and switch which one you are moving (and hence the camera)):

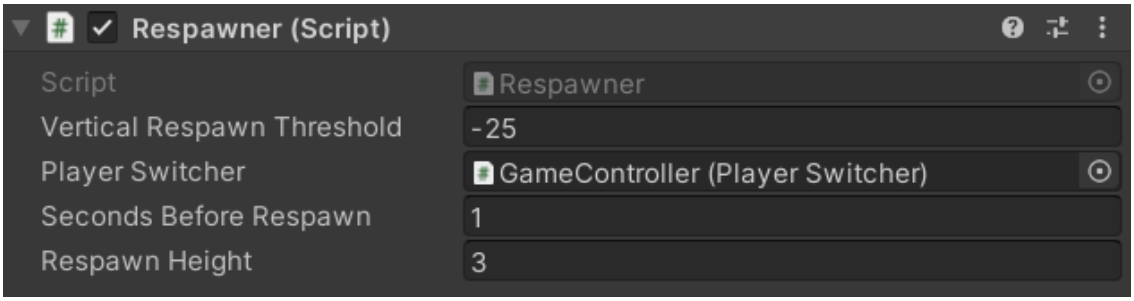




As showed in the screenshots, switching to different players causes different camera position and movement input.

Also for each player, I added a box collider and rigidbody component

and modified Respawner script to meet the requirements



```
private void FixedUpdate()
{
    if (playerSwitcher.GetCurrentPlayer().transform.position.y <
verticalRespawnThreshold)
        StartCoroutine("WaitAndRespawn");
}

IEnumerator WaitAndRespawn()
{
    yield return new WaitForSeconds(secondsBeforeRespawn);
    Respawn();
}
```

5. What happens if you wire-up your Movement Controller to a part or child of your character?

   Before doing the extra credit, only the part of the character will move. But after extra credit parts, my script automatically gets the current active character through PlayerSwitcher.GetCurrentPlayer(), so it does not matter.

6. What scripts could you reuse for other projects?

   In my opinions, all scripts can be reused with possible modifications.

7. Which scripts could you not reuse?

   None.

8. What are the bugs you have found in this implementation / architecture? Hint: there are some. Any ideas on how to fix these?

   Object not set to a reference.

   Eventually, I found Awake() methods are not guaranteed to be called before all OnEnable() methods. Awake() is only guaranteed to be called before the OnEnable() method in the same object. Therefore, for example, using an instance in OnEnbale() of object A while setting that instance to a reference in Awake() of object B can result in null reference errors.

9. Run Analyze->Code Metrics in Visual Studio and report the results (summarize and show a readable table). See example.

Hierarchy	Maintainability I...	Cyclomatic Com...	Depth of Inherita...	Class Coupling	Lines of Source c...	Lines of Executable ...
Assembly-CSharp (Debug)	81	81	5	41	830	158
YangBaihua.Input	86	47	1	17	543	89
InputActions	88	16	1	15	540	23
InputActions.IPlayerActions	100	7	0	2	10	0
InputActions.PlayerActions	71	24	1	5	69	66
YangBaihua.Lab1	80	34	5	29	287	69
CameraSwitcher	72	5	5	4	42	13
FollowWithOffset	88	1	5	6	11	1
InputManager	75	2	5	13	25	6
MovementControl	76	2	5	9	26	8
NextCameraHandler	85	2	1	4	15	4
PlayerSwitcher	77	9	5	3	47	14
PlayerSwitchHandler	83	2	1	4	16	4
QuitHandler	85	2	1	5	16	4
ResetHandler	85	2	1	4	17	4
Respawner	79	7	5	12	42	11

   High maintainability, medium cyclomatic complexity, low depth of inheritance, medium coupling, not too many lines of code.

10. What are the scale values of children objects for you character? How does this work?

    Most of them are between (0.1, 10). They depend on how I want the character to look like. If I want a bigger head, for example, I enter some values to see which is closer to my expectation, then slide the value slightly to make it more precise.

11. What happens when you move past the end of the plane (before and after the extra credit)?

    Before, nothing happens. Player stays in the air. After, player falls.

Source Code:

```
using YangBaihua.Input;
using UnityEngine;

namespace YangBaihua.Lab1
{
    public class InputManager : MonoBehaviour
    {
        [SerializeField] private MovementControl movementController;
        [SerializeField] private CameraSwitcher cameraSwitcher;
        [SerializeField] private Respawner respawner;
        [SerializeField] private PlayerSwitcher playerSwitcher;
        private ResetHandler resetHandler;
        private InputActions inputScheme;

        private void Awake()
        {
            inputScheme = new InputActions();
            movementController.Initialize(inputScheme.Player.Move);
            resetHandler = new ResetHandler(inputScheme.Player.Reset,
respawner);

        }

        private void OnEnable()
        {
            var _ = new QuitHandler(inputScheme.Player.Quit);
            var nextCameraHandler = new
NextCameraHandler(inputScheme.Player.CameraSwitch, cameraSwitcher);
            //var resetHandler = new ResetHandler(inputScheme.Player.Reset,
respawner);
            var playerSwitchHandler = new
PlayerSwitchHandler(inputScheme.Player.PlayerSwitch, playerSwitcher);
        }
    }
}
```

```
using UnityEngine;

namespace YangBaihua.Lab1
{
    public class CameraSwitcher : MonoBehaviour
    {
        [SerializeField] private Camera[] cameras;
        [SerializeField] private Camera defaultCamera;
        private int index = 0;

        void Start()
        {
            index = 0;
```

```
            // Loop through each camera and disable it.
            // Enable the default camera
            // (optional) make sure next camera is
            // not the default (if more than one)

            for (int i = 0; i < cameras.Length; i++)
            {
                var camera = cameras[i];
                camera.enabled = true;
                camera.gameObject.SetActive(false);

                if (camera.Equals(defaultCamera))
                    index = i;
            }

            defaultCamera.gameObject.SetActive(true);

        }
        public void NextCamera()
        {
            // Enable the next camera
            // then disable the current camera

            int newIndex = index + 1;
            if (newIndex >= cameras.Length)
                newIndex = 0;

            cameras[newIndex].gameObject.SetActive(true);
            cameras[index].gameObject.SetActive(false);

            index = newIndex;
        }
    }
}
```

```
using UnityEngine.InputSystem;

namespace YangBaihua.Lab1
{
    public class PlayerSwitchHandler
    {
        private PlayerSwitcher playerSwitcher;

        public PlayerSwitchHandler(InputAction action, PlayerSwitcher
playerSwitcher)
        {
            action.performed += PlayerSwitch_performed;
            action.Enable();
            this.playerSwitcher = playerSwitcher;
        }

        void PlayerSwitch_performed(InputAction.CallbackContext obj)
        {
            playerSwitcher.SwitchPlayer(obj.ReadValue<float>() > 0);
        }
    }
```

```
}
```

```
using UnityEngine;

namespace YangBaihua.Lab1
{
    public class PlayerSwitcher : MonoBehaviour
    {
        [SerializeField] private GameObject[] players;
        [SerializeField] private GameObject defaultPlayer;
        private int index = 0;

        private void Start()
        {
            for (int i = 0; i < players.Length; i++)
            {
                var player = players[i];
                player.gameObject.SetActive(true);

                if (player.Equals(defaultPlayer))
                    index = i;
            }
        }

        public void SwitchPlayer(bool next)
        {
            int newIndex = index;
            if (next)
            {
                newIndex = index + 1;
                if (newIndex >= players.Length)
                    newIndex = 0;
            }
            else
            {
                newIndex = index - 1;
                if (newIndex < 0)
                    newIndex = players.Length - 1;
            }

            index = newIndex;

        }

        public GameObject GetCurrentPlayer()
        {
            return players[index];
        }
        public GameObject[] GetPlayers()
        {
            return players;
        }
    }
}
```

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

namespace YangBaihua.Lab1
{
    public class Respawner : MonoBehaviour
    {
        [SerializeField] private float verticalRespawnThreshold;
        [SerializeField] private PlayerSwitcher playerSwitcher;
        [SerializeField] private float secondsBeforeRespawn;
        [SerializeField] private float respawnHeight;
        private Dictionary<GameObject, Vector3> respawnPointDict;

        private void Awake()
        {
            respawnPointDict = new Dictionary<GameObject, Vector3>();
        }

        private void Start()
        {
            foreach(GameObject player in playerSwitcher.GetPlayers())
            {
                Vector3 position = player.transform.position + Vector3.up *
respawnHeight;
                respawnPointDict.Add(player, position);
            }
        }

        public void Respawn()
        {
            var target = playerSwitcher.GetCurrentPlayer().transform;
            target.GetComponent<Rigidbody>().velocity = Vector3.up *
target.GetComponent<Rigidbody>().velocity.y;
            target.GetComponent<Rigidbody>().angularVelocity = new Vector3(0f,
0f, 0f);
            target.position =
respawnPointDict[playerSwitcher.GetCurrentPlayer().gameObject];
        }

        private void FixedUpdate()
        {
            if (playerSwitcher.GetCurrentPlayer().transform.position.y <
verticalRespawnThreshold)
                StartCoroutine("WaitAndRespawn");
        }

        IEnumerator WaitAndRespawn()
        {
            yield return new WaitForSeconds(secondsBeforeRespawn);
            Respawn();
        }
    }
}
```

```
using UnityEngine;
```

```
namespace YangBaihua.Lab1
{
    public class FollowWithOffset : MonoBehaviour
    {
        [SerializeField] private Transform target; // not used
        [SerializeField] private Vector3 offset;
        [SerializeField] private PlayerSwitcher playerSwitcher;

        private void Update()
        {
            gameObject.transform.position =
playerSwitcher.GetCurrentPlayer().transform.position + offset;
        }
    }
}
```

```
using UnityEngine;
using UnityEngine.InputSystem;

namespace YangBaihua.Lab1
{
    public class MovementControl : MonoBehaviour
    {
        [SerializeField] private PlayerSwitcher playerSwitcher;
        [SerializeField] private GameObject playerToMove; // no need to be
serialized
        [SerializeField] private float speed = 5f;
        private InputAction moveAction;
        private float horizontal;
        private float vertical;

        public void Initialize(InputAction moveAction)
        {
            moveAction.Enable();
            this.moveAction = moveAction;
        }

        private void FixedUpdate()
        {
            playerToMove = playerSwitcher.GetCurrentPlayer();

            horizontal = moveAction.ReadValue<Vector2>().x;
            vertical = moveAction.ReadValue<Vector2>().y;
            Vector3 moveDirection = Vector3.forward * vertical + Vector3.right *
horizontal;
            playerToMove.transform.position += moveDirection * speed *
Time.deltaTime;
        }

    }
}
```

```
using UnityEngine.InputSystem;
```

```
namespace YangBaihua.Lab1
{
    public class NextCameraHandler
    {
        private CameraSwitcher cameraSwitcher;
        public NextCameraHandler(InputAction action, CameraSwitcher
cameraSwitcher)
        {
            action.performed += NextCamera_performed;
            action.Enable();
            this.cameraSwitcher = cameraSwitcher;
        }

        private void NextCamera_performed(InputAction.CallbackContext obj)
        {
            cameraSwitcher.NextCamera();
        }
    }
}
```

```
using UnityEngine;
using UnityEngine.InputSystem;

namespace YangBaihua.Lab1
{
    public class QuitHandler
    {
        public QuitHandler(InputAction quitAction)
        {
            quitAction.performed += QuitAction_performed;
            quitAction.Enable();
        }

        private void QuitAction_performed(InputAction.CallbackContext obj)
        {
#if UNITY_EDITOR
            UnityEditor.EditorApplication.isPlaying = false;
#endif
            Application.Quit();
        }
    }
}
```

```
using UnityEngine.InputSystem;

namespace YangBaihua.Lab1
{
    public class ResetHandler
    {
        private Respawner respawner;

        public ResetHandler(InputAction action, Respawner respawner)
        {
```

```
            action.performed += Reset_performed;
            action.Enable();
            this.respawner = respawner;
        }

        void Reset_performed(InputAction.CallbackContext obj)
        {
            respawner.Respawn();
        }
    }
}
```