Lab #1

CSE3541/5541 (Spring 2024)

Due Date: Tuesday, Jan 23 by 11:59pm

Guidelines

- The TAs/graders will NOT overcome compiler or debugging errors for you.
- All make-ups for lab must be accompanied by a documented and verifiable excuse well before the deadline. Given the severity of the emergency please inform me as soon as possible. See the syllabus.
- Lab submissions will NOT be accepted via email to me or the grader.
- Work on the lab on your own. **No group work**!

Objectives

The purpose of this lab is to introduce you to the Unity development environment and have you start to code in C#. This lab contains detailed steps to follow. This will not be true for future labs, which will be less prescriptive and will have more implementation requirements.

Collaboration and Piazza

Please read the course policy on Academic Misconduct in the syllabus. You may discuss the lab with your classmates ONLY at a high level. You must formulate your own solution (all code must be authored by you alone) without any help from others or third party sources (e.g., the internet).

Do not post any part of your solution or spoilers on Piazza.

In your lab report (see below), you need to list with whom you have discussed the lab.

Task I (Preliminaries)

Task IA: Start early

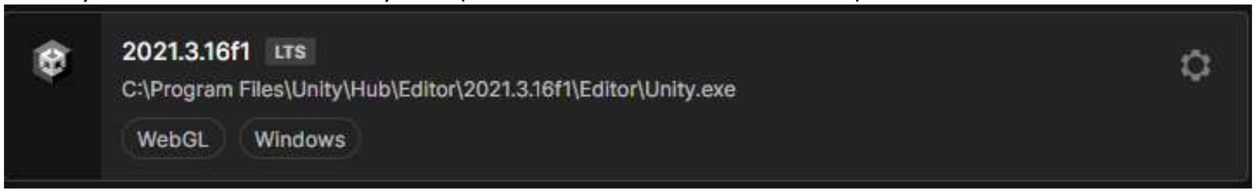
You will write a solution in Unity that moves a character using events from the new input system that allows various devices. **A sample solution is provided to you on Carmen where you found this assignment description file**.

Please read: The expectation is that you look at a lab assignment early and determine how much time you will need to complete it. Only you know your programming and debugging skills, which will differ with many of your peers. **In most all cases, if you wait to start a lab close to the deadline it will most likely result in a late submission**.

In addition, you need to allocate time to write the lab report that you will submit with your solution.

Task IB: Unity Hub, Unity Editor, Follow tutorials

1. **Computer requirement**. This course requires that you must have a computer to use (Windows or Mac – no Linux) to complete the labs. The CSE computer labs do not have Unity installed on them (due to licensing issues). If you have an older computer then you should use a newer one because Unity will run too slowly.
2. **Install the Unity Hub**. You must install the Unity Hub first **before** installing a Unity Editor. Use the following tutorial: https://learn.unity.com/tutorial/install-the-unity-hub-and-editor#.
3. **Install a Unity Editor**. Install a recent version of a Unity editor (2021.1, 2021.2, or higher) when the Unity Hub is installed (previous step) and running. Check that the WebGL platform has been installed as well. If it didn't install automatically already then use these instructions: https://docs.unity3d.com/hub/manual/AddModules.html. Here is an example of what you should see in your "Installs" in the Unity Hub (notice WebGL at the bottom left):


2021.3.16f1 LTS
C:\Program Files\Unity\Hub\Editor\2021.3.16f1\Editor\Unity.exe
WebGL Windows

4. **Install Visual Studio.** Make sure you install at least version 2019. An alternative is to install Visual Code, but computer science students should use a real tool.
5. **Quickly learn C# or brush up on your skills.** Use Programming in C# web page provided by Professor Crawfis: https://web.cse.ohio-state.edu/~crawfis.3/cse459_CSharp/index.html.
6. For those interested in game development, I would start working on the Junior Programming Pathway at https://learn.unity.com/. Carl is pretty entertaining at 2x speed.

Task IC: Incrementally build and maintain versions of your solution with "git"

Use the version control system **git** (not Github) and run it locally on your machine. Maintain multiple versions of your solution as you progress in your work. This is the same what you did in your project course (i.e., 390X courses). Maintain a repo for each lab that you implement this semester. **Do NOT maintain your solution Github. Making your solution available publically is a violation of academic misconduct according to University policy.**

Task II (New Project)

Use the following steps from the Unity Hub (refer to these steps for future labs) to create your project (You can refer to step 5 called "Setting Up Projects With Unity Hub" here https://learn.unity.com/tutorial/project-configuration-with-unity-hub-1?uv=2019.2#5d924899edbc2a131ad3ca27):

1. Create a new empty (3D) Unity project.
2. Create (if not already created) new folders under Assets called **Scenes (probably already there)**, **Scripts**, **Materials**, and **Input**. You can rename the default SampleScene in the folder Scenes.
3. Set the default namespace for any scripts (See Task V below) to your LastnameFirstname.Lab1 (e.g., ShareefNaeem.Lab1).
4. Check (change?) that the color space is set to **Linear** (See https://docs.unity3d.com/Manual/LinearRendering-LinearOrGammaWorkflow.html).
5. Save the scene by selection **File→Save Scene**. Include your name as part of the scene file name (e.g., ShareefMovingObject). You will use the built-in play, pause, and step buttons to view your animation.

Task III (New Input System)

You are required to use the new "Input System" in all labs for user input. Watch and **use** the following two videos about the new Input System (particularly with C# events) using the input requirements below. You can watch the videos first and then refer to them while following my steps or follow both simultaneously.

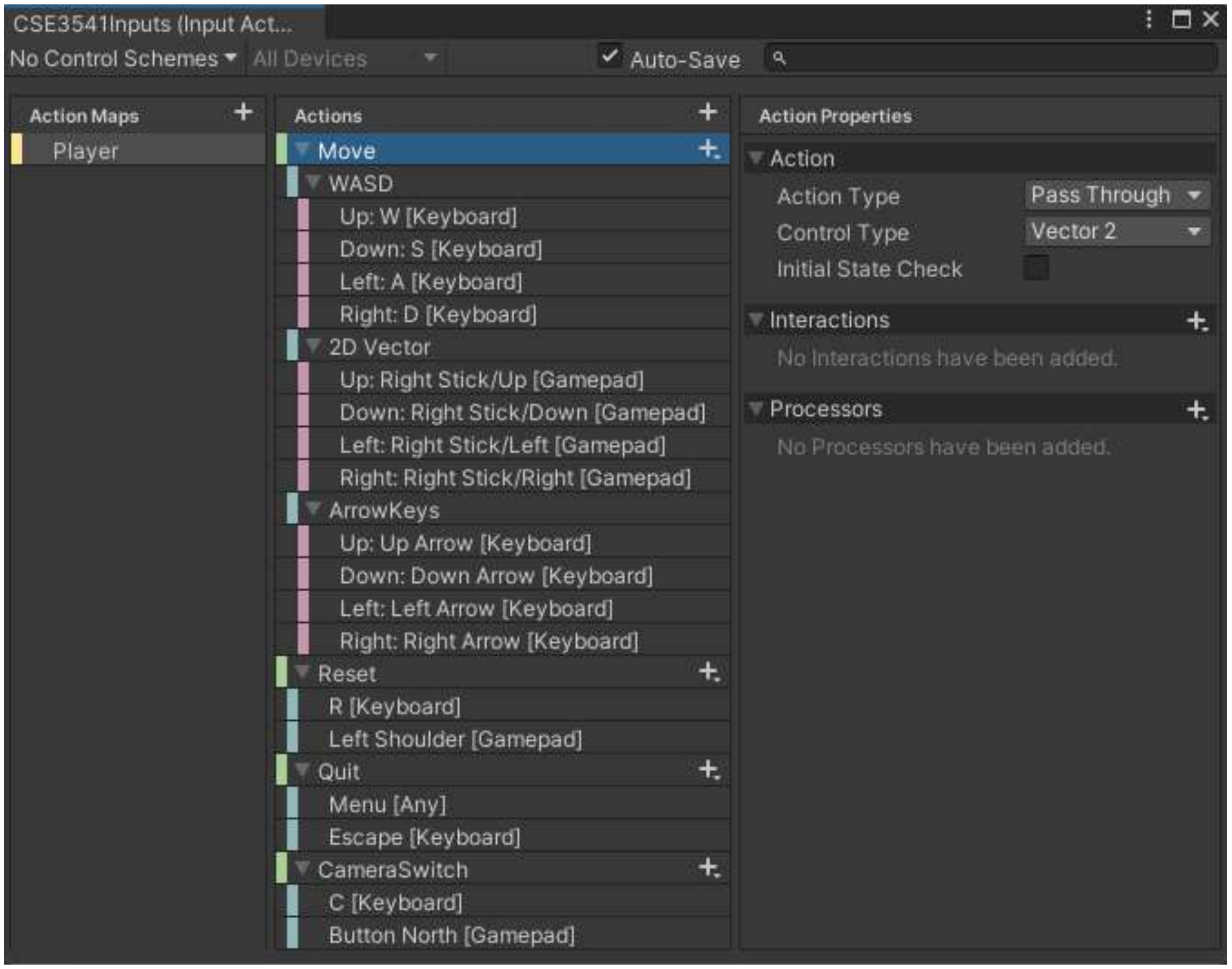
- https://www.youtube.com/watch?v=YHC-6I_LSos (16 minutes, 45 seconds)
- https://www.youtube.com/watch?v=kGykP7VZCvg (36 minutes, 58 seconds)

Input requirements:

- Import the package Input System using the Package Manager (found under "Unity Registry"): https://docs.unity3d.com/Packages/com.unity.inputsystem@1.0/manual/Installation.html#installing-the-package.
- When it prompts you to replace the old input system, say **yes**. Unity will need to restart.
- Call the ActionMap "Player" (see below).
- For the following steps (see screen shots on the next page for what yours should look like), if you are unable to find the "Path" then try to type it in the search bar:
    - Add an action (I called mine "Move") and set the "Action Type" and "Control Type" as shown below.
        - Using "Add Up\Down\Left\Right Composite" (uses 2D vector as a composite) to add WASD, Gamepad controls, and arrow keys:
            - WASD(wasd) keys on the keyboard for movement
            - Arrow keys on the keyboard for movement (same as WASD)
            - Gamepad right-stick for movement
    - Add an action (I called mine "Reset") and define two bindings with the r key on keyboard and Left shoulder on gamepad to reset the position

- Add an action (I called mine “Quit”) and define two bindings with the Esc on the keyboard and Menu button on the gamepad to quit
- Add an action (I called mine “CameraSwitch”) and define two bindings with the C on the keyboard and North button (Y on Xbox One controller) to switch cameras

Screen shot of what yours should look like:



CSE3541Inputs (Input Act...
No Control Schemes ▾ All Devices ▾ ✓ Auto-Save

```
Action Maps        Actions                                   Action Properties
  Player           ▼ Move                                    ▼ Action
                     ▼ WASD                                      Action Type          Pass Through
                         Up: W [Keyboard]                        Control Type         Vector 2
                         Down: S [Keyboard]                      Initial State Check
                         Left: A [Keyboard]                  ▼ Interactions
                         Right: D [Keyboard]                     No Interactions have been added.
                     ▼ 2D Vector                             ▼ Processors
                         Up: Right Stick/Up [Gamepad]            No Processors have been added.
                         Down: Right Stick/Down [Gamepad]
                         Left: Right Stick/Left [Gamepad]
                         Right: Right Stick/Right [Gamepad]
                     ▼ ArrowKeys
                         Up: Up Arrow [Keyboard]
                         Down: Down Arrow [Keyboard]
                         Left: Left Arrow [Keyboard]
                         Right: Right Arrow [Keyboard]
                   ▼ Reset
                       R [Keyboard]
                       Left Shoulder [Gamepad]
                   ▼ Quit
                       Menu [Any]
                       Escape [Keyboard]
                   ▼ CameraSwitch
                       C [Keyboard]
                       Button North [Gamepad]
```

- In the Inspector of the Input Action Asset, add a root namespace (LastNameFirstName.Input), i.e. use your full name, and have it generate the C# class (Toggle “Generate C# Class”). I like to place this into the Scripts folder. You can take a look at this script if you like. Most of it is JSON initialization (see https://www.youtube.com/watch?v=YHC-6I_LSos).

Task IV (Scene)

You will construct a scene consisting of a floor, two walls, and a character that will roam around this environment using user input. First, learn about and familiarize yourself with the Unity Editor that you installed after installing the Unity Hub. A good place to start is here: https://learn.unity.com/tutorial/explore-the-unity-editor-1# and look at other relevant tutorials in https://learn.unity.com/.

1. Create the environment (floor and two walls)

- Add an empty GameObject to your scene
  - Set the name to "*Environment*"
  - Position of zero; No rotation; Scale of one
  - Add children to this GameObject
    - A plane going from (-25, 0, 0) to (25, 0, 100) in global (World) coordinates
      - Set the name to "*Floor*"
      - A plane is 10 meters by 10 meters by default; Scale the plane using (5, 1, 10) to achieve the position defined above
    - Two cubes going along the *z*-axis of the plane on each *x*-axis edge.
      - Set the names to these as "*LeftEdge*" and "*RightEdge*".
- Mark the root of the Environment node as **static** in the Inspector (and all of its children).
- For the floor, add simple textures in order to provide some context. It can be hard to tell your character is moving. Import any packages you experiment with into a different Unity project and then only export (or copy over) 1 or 2 of these materials. They can get quite large.

  Here is a small useful one:

  https://assetstore.unity.com/packages/2d/textures-materials/gridbox-prototype-materials-129127

  Other choices:

  https://assetstore.unity.com/packages/2d/textures-materials/stone/tileable-cobblestone-texture-21235

  https://assetstore.unity.com/packages/2d/textures-materials/tiles/hand-painted-seamless-tiles-texture-vol-5-162143

  Note your submission should be less than 3MB!!!

- Create a hierarchical character.
  - Here are some examples that took less than 3 minutes to make:

    They are comprised of 2, 8, 8, and 12 primitives, respectively. The third one uses some transformations of the second one.

  - Use an Empty GameObject (highly suggested for collisions and a local coordinate frame, more on this later) as the base or one of the default Unity Primitives.
    - Give your empty game object a name, e.g. Sally or Fred.
    - Make sure the children are all above the plane.

- Give the character an initial position: Some inspiration:

  https://www.youtube.com/watch?v=8wm9ti-gzLM

  https://www.youtube.com/watch?v=3VQl0scK5HM

  https://www.artstation.com/artwork/ZWeZN (don't make this complicated, but you could make an entire voxel character)

- In the Materials folder, create several basic materials. Note (for this lab) do not use pre-built models or textures for the character. Check out a color palette to help with nice looking material colors (hint: avoid fully saturated colors (255, 0, 0), (0, 255, 0) and (0, 0, 255). https://coolors.co/, https://blog.templatetoaster.com/color-palette-generators/.

Task V (Unity Scripts)

You will follow the instructions below and use the provided skeleton code in your solution to write Unity

```
using UnityEngine;
using UnityEngine.InputSystem;

namespace ShareefSoftware
{
    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();
        }
    }
}
```

scripts to define the logic of the application. Change the namespace from "ShareefSoftware" to LastnameFirstname.Lab1 (e.g., ShareefNaeem.Lab1). In my solution I write less than 30 lines of code, with only 1-11 lines of executable code!

Quit Handler

Try to always add this as one of the very first things you do in future labs. Otherwise, you have a build that you cannot quit out of. Follow the tutorial here (https://www.youtube.com/watch?v=YHC-6I_LSos) for how to use C# events and subscribe to them. There are some more subtle issues if you are interested in a more professional approach (e.g., Moving vs MoveRequested, and QuitRequest vs Quitting events). This quit handler (above) will work for both the Editor as well as a finished built game when the quit event is fired. It simply calls *Application.Quit*() when outside a build or sets *isPlaying* to false if running in the Unity editor. Of course, it will not work yet in your scene, as we have not created an instance of this class. For this, we use a new class, InputManager (below).

Movement

You should write a script (MonoBehavior) to handle the move events. To the left is my skeleton. It takes a GameObject that it will move (indicated by the [SerializeField] attribute, as well as a movement speed (in m/s). These are assigned in the Unity Editor for each Scene / usage. To reduce coupling, an Initialization method is added which is passed in the InputAction associated with movement. Make sure you enable the InputAction and save a reference to it for the FixedUpdate method. Again, we need to create an instance of this class in order for our movement to work. This will be handled next.

```
using UnityEngine;
using UnityEngine.InputSystem;

namespace ShareefSoftware
{
     public class MovementControl : MonoBehaviour
     {
         [SerializeField] private GameObject playerToMove;
         [SerializeField] private float speed = 5f;
         private InputAction moveAction;

         public void Initialize(InputAction moveAction)
         {
…
         }
         private void FixedUpdate()
         {
…
         }
     }
}
```

```
using ShareefSoftware;
using ShareefSoftware.Input;
using UnityEngine;

namespace ShareefSoftware
{
    public class InputManager : MonoBehaviour
    {
        [SerializeField] private MovementControl movementController;
        private CSE3541Inputs inputScheme;

        private void Awake()
        {
            inputScheme = new CSE3541Inputs();
            movementController.Initialize(inputScheme.Player.Move);
        }
        private void OnEnable()
        {
            var _ = new QuitHandler(inputScheme.Player.Quit);
        }
    }
}
```

Input Manager

Our InputManager class will handle all of the initialization of the input and associated handlers. It is what Unity and others would call the PlayerInput class. Above is the basic structure. We will add to this as the lab is worked out. Note that the class CSE3541Inputs is the name I called my InputActionMap asset, so this is the name it chose automatically when it generated the class. This class is now automatically recreated every time we make a change to the InputMap. Since this is derived from MonoBehaviour, we can add it to a GameObject in our scene to create it. Unity will then call its Awake and OnEnable methods when we start our game.

Cameras

For this project, we cannot see the characters face with an over the shoulder camera (or no rotations on the character or camera). We will add 4 cameras:

1) Typical 3rd person Rear-facing camera
2) Front-facing camera (rotate about y-axis 180 degrees).
3) Side camera
4) Top-down camera

For the 4th camera, make it an orthographic camera (if you need help: https://www.youtube.com/watch?v=9CjQKVtshno). Zoom in on your character some (except maybe for the top-down camera). If you play your "game" now, the character will move outside the camera. We will fix this with a script that moves the camera to follow the player. To do this, we will use a very simple script attached to **each** camera. Set the position of the gameObject (its transform) to the position of the target plus the offset. There is one line of code for you to add. You will need to set the offset vector in the Unity Editor for each camera.

```
using UnityEngine;

namespace ShareefSoftware
{
    class FollowWithOffset : MonoBehaviour
    {
        [SerializeField] private Transform target;
        [SerializeField] private Vector3 offset;

        private void Update()
        {
…
        }
    }
}
```

We also need a small piece of code (aren't they all nice and small!) to switch cameras. For this lab, we will cycle through an array or list of cameras. This will show you how easy it is to add arrays as [SerializeField]'s that can be set in the Unity editor. We will keep an index of the current camera and then activate the next camera each time the event is fired. For this to work, you need to enable the next camera **before** disabling the active camera. Here is a skeleton to get you started. Once you have this, you can initialize it in the Input Manager.

```
using UnityEngine;

namespace ShareefSoftware
{
    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)
        }

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

Reset / Respawn Character

Write a script (MonoBehaviour) to store the initial position of the character on Awake or Start. I called mine PlayerRespawner, but Respawner would have been a better name. Add a method called Respawn, that will set the position back to the initial position. Also save off the rotation (a Quaternion) and reset it as well (transform rotation). Now write a ResetRequestHandler or just ResetHandler that is much like the

```
using ShareefSoftware;
using ShareefSoftware.Input;
using UnityEngine;

namespace ShareefSoftware
{
    public class InputManager : MonoBehaviour
    {
        [SerializeField] private MovementControl movementController;
        [SerializeField] private CameraSwitcher cameraSwitcher;
        [SerializeField] private PlayerRespawner playerRespawner;
        private CSE3541Inputs inputScheme;
        private ResetHandler resetHandler;

        private void Awake()
        {
            inputScheme = new CSE3541Inputs();
            movementController.Initialize(inputScheme.Player.Move);
            resetHandler = new ResetHandler(inputScheme.Player.Reset, playerRespawner);
        }
        private void OnEnable()
        {
            var _ = new QuitHandler(inputScheme.Player.Quit);
            var nextCameraHandler = new NextCameraHandler(inputScheme.Player.CameraSwitch, this.cameraSwitcher);
        }
    }
}
```

QuitHandler in that it subscribes to the performed event of an InputAction passed in an Initialize method. In this case though, also pass in a PlayerRespawner (or Respawner) instance and store that. When the event fires, simply call Respawn. See the final InputManager on how this is initialized.

Input Manager (final)

Now that we have a camera switcher, and a PlayerRespawner in addition to our QuitHandler and MovementController, we can create and / or initialize all of our input functionality. Note, this could be split across many classes. Here it shows all of the "glue" needed to get our simple control scheme working in one class, so high cohesions, but somewhat higher coupling.

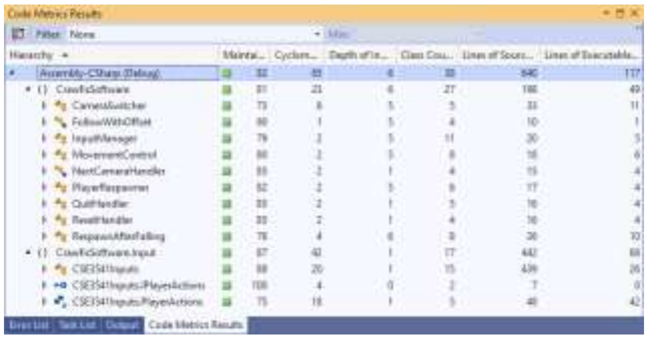
Documentation / Lab Report

Write a **detailed report** that describes on what you have implemented. This is a report, **so please do not just provide short answers to the questions below as this won't receive full credit**. For question 10, do not leave it blank or say there are none.

- Document your source code
- **Include a short introduction that introduces the reader to the problem (as you would in any engineering lab report)**
- **Include a few screen shots of your renderings (these should be figures in the lab report to illustrate items to explain, i.e. provide explanations in your text)**
- **Include all your source code in this report** (except the autogenerated Input class)
    - **Provide short high level descriptions of what each code snippet does (code does not explain itself)**
- **Include answers to the following questions in your report:**

1) State with whom you have had discussions about the lab
2) State what you have implemented
3) Did you do any of the extra credit? If so, explain in detail and include images.
4) How hard or easy did you find this lab?
5) Have you ever worked in Unity before?

6) Have you programmed in C# much?
7) What happens if you wire-up your Movement Controller to a part or child of your character?
8) What scripts could you reuse for other projects?
9) Which scripts could you not reuse?
10) Comment on the code structure given in this lab. Can you suggest any improvements to the structure?
11) What are the advantages and disadvantages of the code structure?
12) Run Analyze->Code Metrics in Visual Studio and report the results (summarize and show a readable table). See example.
13) What are the scale values of children objects for your character? How does this work?
14) What happens when you move past the end of the plane (before and after the extra credit)?



Code Metrics Results

Filter: None

Hierarchy	Maintai...	Cyclom...	Depth of In...	Class Cou...	Lines of Sourc...	Lines of Executable...
Assembly-CSharp (Debug)	82	65	6	30	640	117
CowficSoftware	81	21	6	27	186	49
CameraSwitcher	73	8	5	3	33	11
FollowWithOffset	90	1	5	4	10	1
InputManager	79	2	5	11	30	5
MovementControl	80	2	5	8	18	6
NextCameraHandler	85	2	1	4	15	4
PlayerRespawner	82	2	5	8	17	4
QuitHandler	88	2	1	3	16	4
ResetHandler	88	2	1	4	16	4
RespawnAfterFalling	78	4	6	3	36	10
CowficSoftware.Input	87	42	1	17	442	68
CSE3541Inputs	88	20	1	15	439	26
CSE3541Inputs.IPlayerActions	100	4	0	2	7	0
CSE3541Inputs.PlayerActions	75	18	1	5	40	42

Error List | Task List | Output | Code Metrics Results

Lab Submission

In addition to your report (**submit this as a PDF file**), submit a **Unity package**.

Within Unity, right-click the scene file and choose Export Package … This ensures that only the files used in the scene are exported (as well as all scripts as it cannot tell sometimes). Save the package as YourlastnameLab1. This should result in a file called YourLastNameLab1.unitypackage being created in the folder you specified (hint, use Documents or some other space). This is the file you should turn in for grading.
Under the Lab1 assignment on Carmen, submit both the PDF file and the unitypackage file using File Upload option (not the Buckeye Box option).

You can test to determine if you have made the package correctly by creating a new Unity project, open the package file from a file browser, import all of the files in it, open the scene within Unity, and click play.

GRADING

Lab 1 will be graded based on the following criteria:

- Is the lab submitted in the correct format and following our file naming conventions? (20 points)
- Was a **thorough** report submitted in addition to the lab implementation? (20 points)
- Is the code well written and formatted to be readable? (8 points)
- Is the Scene set-up properly with good names? (8 points)
- Is the motion control correct (orthogonal translation using the WASD keys) and can the character's position be reset (pressing R)? (20 points)
- Do the arrow keys and gamepad function the same as the keyboard? (8 points)

- Does the camera switching work as expected? (12 points)
- Is the student's name on the report and used in the namespaces? (4 point)

Extra Credit

For extra credit as well as greater complexity / challenge consider the following (each worth 3 points for a maximum of 9 points). You must complete all steps for each. Ambitious students should do all three!

A. Gravity, collisions and a better respawn (3 points)
   1) Add a Box Collider to the root of your character as well as a RigidBody. You may want to freeze the rotations.
   2) Add a script that checks if a GameObject's y-position falls below a threshold (falls off of the plane). If so, it calls the Respawn
   3) Rather than call Respawn right away, start a Coroutine that waits for a few seconds and then respawns
   4) Have the character respawn a couple of meters above the plane (use a SerializeField for this distance)

B. Multiple characters and character switching (3 points)
   1) Create 3 additional characters
   2) Write a script to switch between characters
   3) Add a new Input binding and event handler for switching characters (Left-Ctrl for previous player and Right-Control for next player, Dpad left/right in gamepad)
   4) (option 1) Hide all other characters and move the next character to the current characters position
   5) (option 2) Have all characters visible and switch which one you are moving (and hence the camera)

C. Local multiplayer (3 points)
   1) Watch the Input Systems on local multiplayer
   2) Look at the included sample (within PackageManager->Input System) for local multiplayer.
   3) Create 2 additional characters
   4) Have WASD control one character, arrow keys control a second character and the gamepad a third character