Week 9: Programming (Part 1 – Player)

Welcome to Week 9!

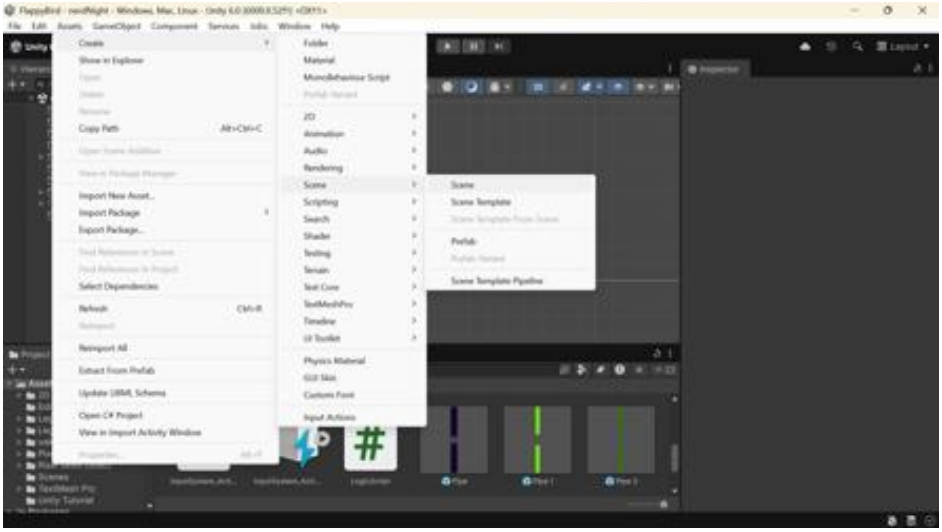
Introduction to Programming the Player

9.1—Introduction

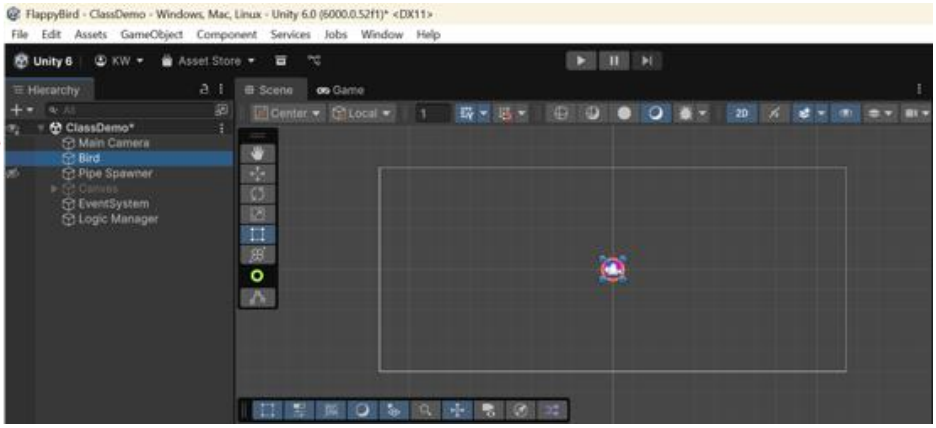
This week starts the development of a Unity project called “Flying Bird” that will mark the final project for Unity 101. The focus for this week is on player controls. The Bird will be created and animated.

9.2—Bird sprite

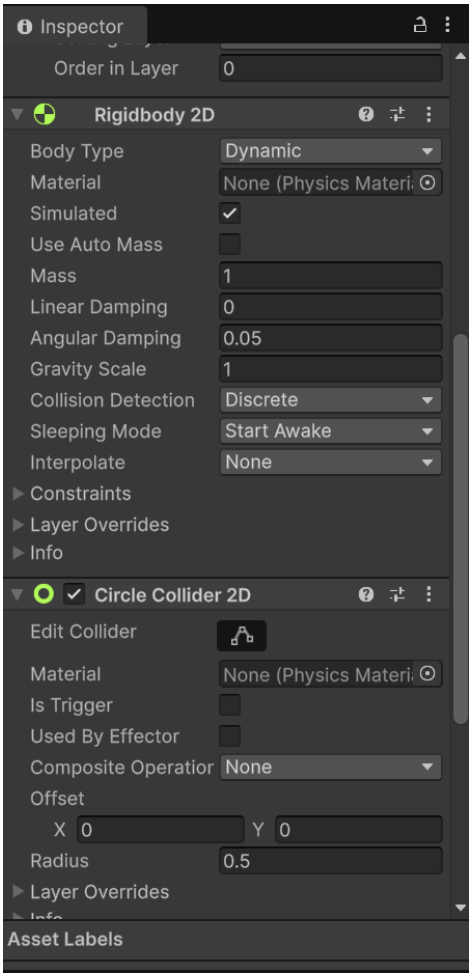
Start by creating a new 2D Unity project using the Unity Hub. Once the project has opened in the Unity Editor, save a new scene. From the top menu, select Assets -> Create -> Scene. This scene will hold the completed Flying Bird game. All the objects, behaviors, and interactions for the game will be built in this scene.



Next, add a 2D sprite to the scene to represent the bird. Create a Circle 2D Sprite and rename it “Bird”. This sprite acts as a temporary placeholder rather than the final visual design of the character. Using a simple shape allows you to focus on game mechanics –like movement and collisions—before spending time on artwork and visual polish. At this stage, functionality is the priority, and visual refinement will come later.



Once the Bird sprite has been added, open the Inspector panel to configure its behavior. Two components need to be added to the Bird object: Rigidbody2D and Circle Collider 2D. The Rigidbody 2D component allows the Bird to respond to physics, like gravity, while the Circle Collider 2D defines the area used to detect collisions with other objects in the game. Together, these components allow the Bird to interact realistically with the game world.

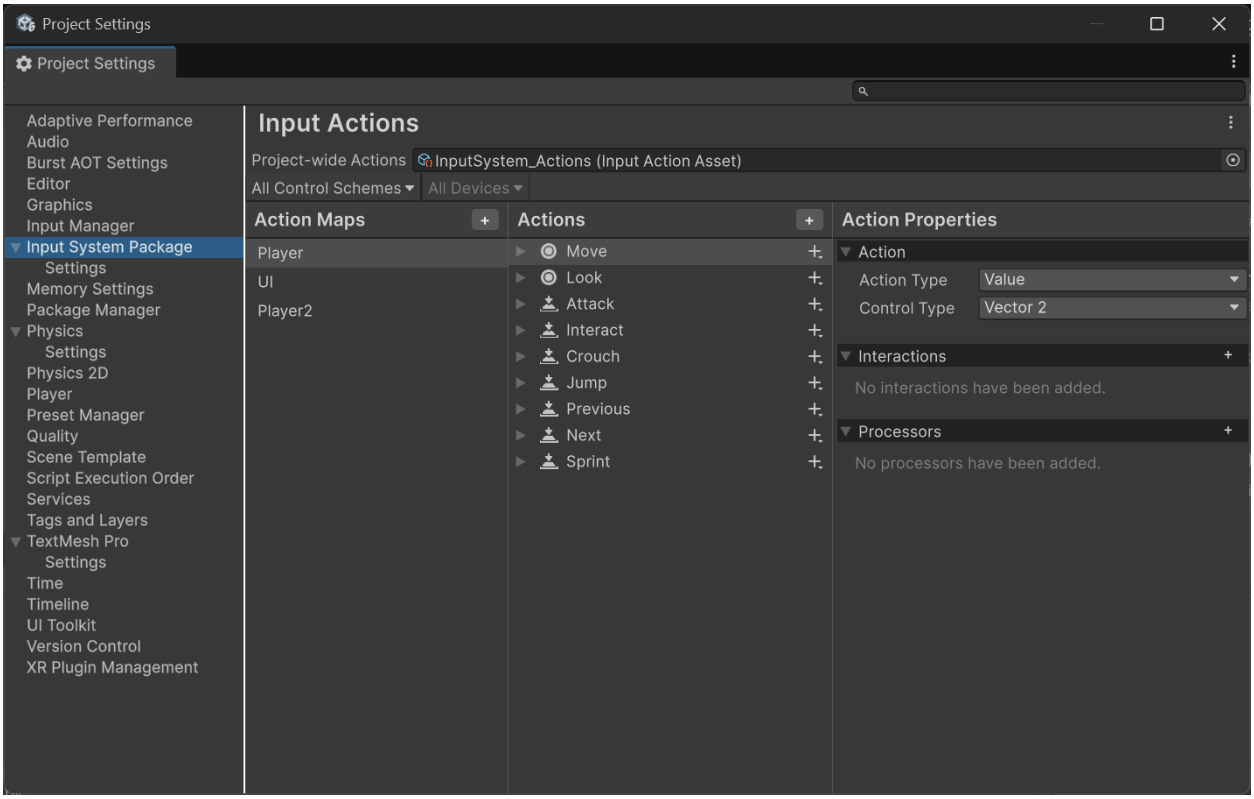


9.3- Input Actions

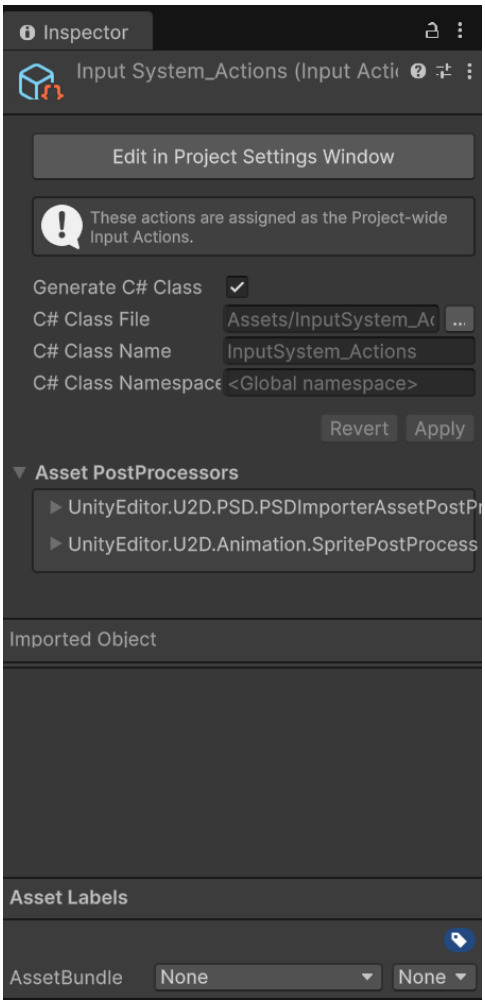
To begin setting up player input, open the Unity Package Manager by selecting Window -> Package Manager from the top menu. Use the Package Manager to import the Input System package into your project. This package provides modern, flexible tools for handling player input and will be used throughout the game to control actions such as jumping.

[Placeholder for image: Unity menu showing Window ->Package Manager and the Input System package selected] [Fig.4 Refer to the Week 6 materials for additional context on the Input System]

After the Input System package has been installed, open Project Settings by selecting Edit ->Project Settings. In the Input Actions section, create a default Input Action Asset. Unity will automatically generate an asset containing common player actions. Take time to review the contents of this asset, paying close attention to the Player Action map and the Jump action within it. The Jump action should be mapped to the spacebar, as this will allow the player to make the bird jump when the spacebar is pressed.

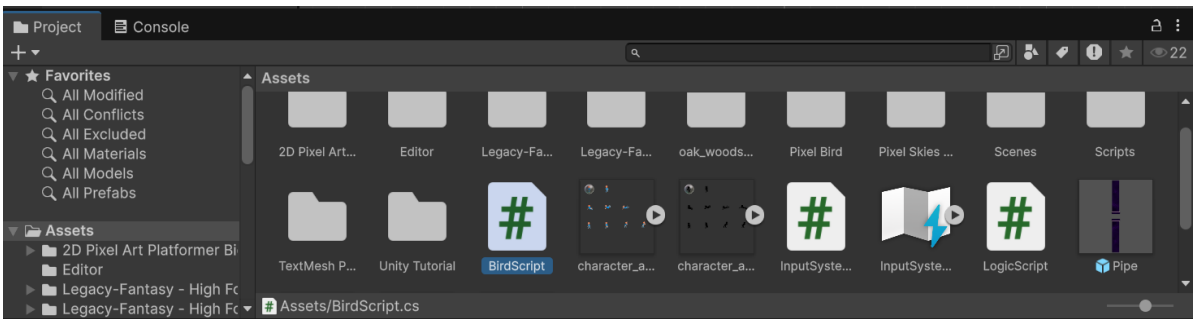


Next, locate the newly created Input Action Asset in the Asset panel. By default, this file is named “InputSystem_Actions.” Select the asset to display its settings in the Inspector panel. Enable the option labeled “Generate C# Class.” [Fig.5]]This setting caused Unity to automatically create a strongly typed C# class from the Input Action Asset. The generated class can then be referenced directly in custom scripts, making it easier and sfer to work with player input in code.



9.4 – Scripting

Begin by creating a new C# script in your Unity project and name it “BirdScript”. This script will be responsible for controlling how the bird responds to the player input and how it moves within the game world. Once created, this script should be attacted to the Bird GameObject in the scene.



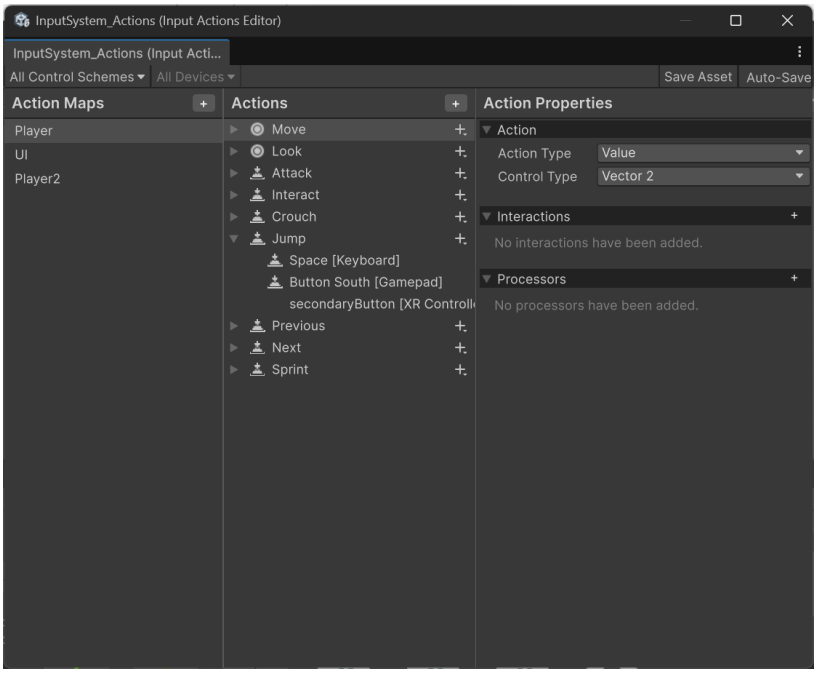
Inside the BirdScript, define two public variables. One variable should use the Rigidbody2D data type, which will allow the Bird’s Rigidbody2D component’s properties to be modified through

the code. The second variable should use the Input System Actions data type, which allows the script to recognized when the player presses a button on their keyboard or controller. By keeping these variables public, they can be easily assigned and adjusted within the Unity Inspector.

```
1   using UnityEngine;
2   using UnityEngine.InputSystem;
3
    Unity Script (1 asset reference) | 1 reference
4   public class BirdScript : MonoBehaviour
5   {
6
7
8       public Rigidbody2D myRigidbody;
9       private InputSystem_Actions controls;
10
```

Next, add an Awake function to the script, placing it before the Start function. The Awake function is called when the script instance is first loaded, even before the game beings running. In this function, create a new instance of the Input System Actions class. This ensures that the player's input controls are initialized early, before any gameplay logic begins. Initializing input in Awake helps prevent missed input or timing issues when the game starts.

After setting up the Awake function, create an OnEnable function. This function is automatically called whenever the script becomes active. Inside it, enable the Player Action Map, which contains the player-related inputs such as jumping. When the default Input Actions asset was created, Unity automatically generated both a Player map and a UI map. For this game, the Player map is the one we want to use. Enabling it ensures that player inputs are recognized while the Bird is active.



Following this, create an OnDisable function. This function is called when the script is disabled or the GameObject is no longer active. In this function, disable the Player Action Map and remove any input callbacks that were added earlier. This step is important for preventing unintended input behavior, such as triggering actions when the player is paused or when switching to a different control mode, like a menu.

In the Start function, configure the Bird's Rigidbody2D so that gravity is initially turned off by setting its simulated property to false. This prevents the Bird from falling as soon as the game loads. Instead, gravity will only begin affecting the Bird once the player chooses to start by pressing the jump button. This approach gives the player control over when gameplay begins.

Finally, create an OnJump function that is triggered when the player presses the jump button. When this function is called, it first checks whether the Bird's Rigidbody2D is currently being simulated by Unity's physics system. If gravity is not yet active, the function enables it so that the game begins at the player's first input. The function applies an upward velocity to the Bird using vector2.up multiplied by a flapping value, which controls how strong the jump is. This upward movement simulates the bird moving upward from flapping its wings.

9.4 – Debugging

For debugging purposes, you may optionally include a Debug.Log statement inside the OnJump function. This message will appear in the Unity Console whenever the jump button is pressed, making it easier to confirm that the input is being detected correctly before the Bird moves upward.

9.5 – Completed Script

This script connects player input to physics-based movement, allowing the Bird to begin falling only when the player is ready to jump upward in response to input.

```
public SpriteRenderer sprite;
    public Rigidbody2D myRigidbody;
    public float flapping;

    private InputSystem_Actions controls;

    private void Awake()
    {
        controls = new InputSystem_Actions();
    }

    private void OnEnable()
    {
        controls.Player.Enable();
        controls.Player.Jump.performed += OnJump;
    }

    private void OnDisable()
    {
        controls.Player.Jump.performed -= OnJump;
```

```
    controls.Player.Disable();
}

private void Start()
{
    myRigidbody.simulated = false;
}
public void OnJump(InputAction.CallbackContext context)
{
    //

    if(myRigidbody.simulated==false)
    {
        myRigidbody.simulated = true;
    }

    Debug.Log("Button is pressed!" );
    myRigidbody.linearVelocity = Vector2.up * flapping;
}
```

9.6—Exercises

Reinforcement

The reinforcement section is used to check the understanding of the information that was presented in the PowerPoint/document. Please answer the questions in your own words.

R—9.1 What does the InputSystem_Actions class represent in this script?

R—9.2 What does it mean when the Jump action is “performed”?

R—9.3 Why is controls.Player.Enable() called in OnEnable()?

R—9.4 Why is controls.Player.Disable() called in OnDisable()?

R—9.5 Why is myRigidbody.simulated set to false at the start of the game?

R—9.6 What does Vector2.up*flapping do to the object’s movement?

R—9.7 Why is Debug.Log(“Button is pressed!”) useful during development?

Project

The project section’s tasks are used in work toward creating the final Flying Bird game project for the end of the course.

P—9.1 Create a new 2D Unity project titled “Final Project”. Import the Input System package.

P—9.2 Add a 2D sprite named “Bird” to scene and attach a C# script called “ClassBirdScript”

P—9.3 Add the following components to the Bird sprite: Rigidbody2D and Circle Collider 2D.

P—9.4 Create the Input Action assets in the Project Settings. Then, generate the C# input actions class.

P—9.5 Create code that Enables and Disables the Player Input map. Subscribe the jump action to a method that will handle jumping.

P—9.6 Create a method that runs when the jump action is performed.