Lab 2: Scripting

CS410/510: VR/AR Development

Ehsan Aryafar earyafar@pdx.edu
Sam Shippey sshippey@pdx.edu

Outline

- Scripting
- Exporting projects
- VR

Deliverable

- A capsule will walk, run, and jump.
- It won’t fall through the floor.
- It can collide with objects, optionally moving them.

Scripting: Review

- Last time: ECS

Scripting: Review

- Last time: ECS
  - Entity/Component/System

Scripting: Review

- Last time: ECS
  - Entity/Component/System (GameObject, Component, internals, etc)

Scripting: Review

- Last time: ECS
  - Entity/Component/System (GameObject, Component, internals, etc)
  - Think of scripts as components that tell systems what to do.

Scripting: Review

- Last time: ECS
    - Entity/Component/System (GameObject, Component, internals, etc)
    - Think of scripts as components that tell systems what to do.
- The parts of scripts we already know
    - Scripts are written in a (mostly) object-oriented style

Scripting: Review

- Last time: ECS
  - Entity/Component/System (GameObject, Component, internals, etc)
  - Think of scripts as components that tell systems what to do.
- The parts of scripts we already know
  - Scripts are written in a (mostly) object-oriented style
  - We’ve already seen most of the functions we’re writing today

Scripting: Review

- Last time: ECS
  - Entity/Component/SystemSystem (GameObject, Component, internals, etc)
  - Think of scripts as components that tell systems what to do.
- The parts of scripts we already know
  - Scripts are written in a (mostly) object-oriented style
  - We’ve already seen most of the functions we’re writing today
  - We don’t even need to manipulate many of the components we’re working with.

Scripting: Review

- Last time: ECS
  - Entity/Component/SystemSystem (GameObject, Component, internals, etc)
  - Think of scripts as components that tell systems what to do.
- The parts of scripts we already know
  - Scripts are written in a (mostly) object-oriented style
  - We’ve already seen most of the functions we’re writing today
  - We don’t even need to manipulate many of the components we’re working with.
- What we might not know
  - Unity API

Scripting: Review

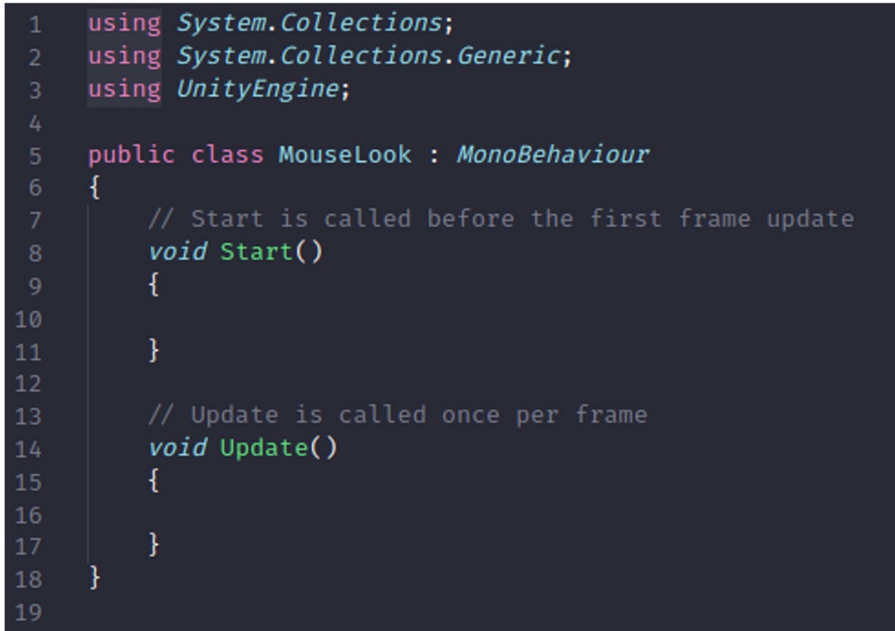
- Last time: ECS
    - Entity/Component/System (GameObject, Component, internals, etc)
    - Think of scripts as components that tell systems what to do.
- The parts of scripts we already know
    - Scripts are written in a (mostly) object-oriented style
    - We’ve already seen most of the functions we’re writing today
    - We don’t even need to manipulate many of the components we’re working with.
- What we might not know
    - Unity API
    - C#

In the editor

- Open up your project from last time
- Right-click under Assets
  - Create->C# Script
  - Name it “MouseLook”
- Repeat above for another one called “KeyboardMovement”
- Open “Mouselook” in your favorite (text) editor

Empty script

- You should now be staring at an empty script
- MonoBehaviour
- Start()
- Update()

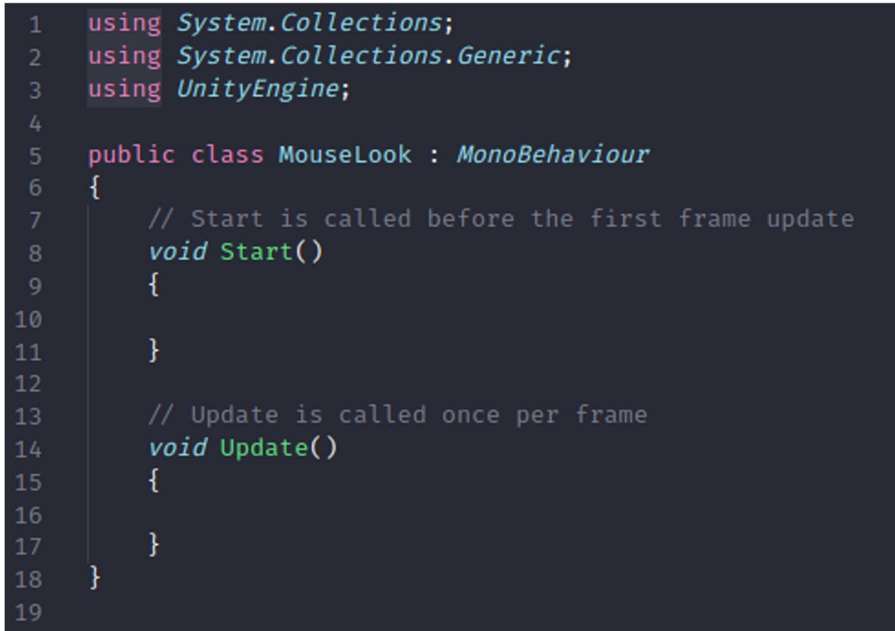


```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MouseLook : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

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

    }
}

```

Scripts here are adapted from a number of forum answers.

MonoBehaviour

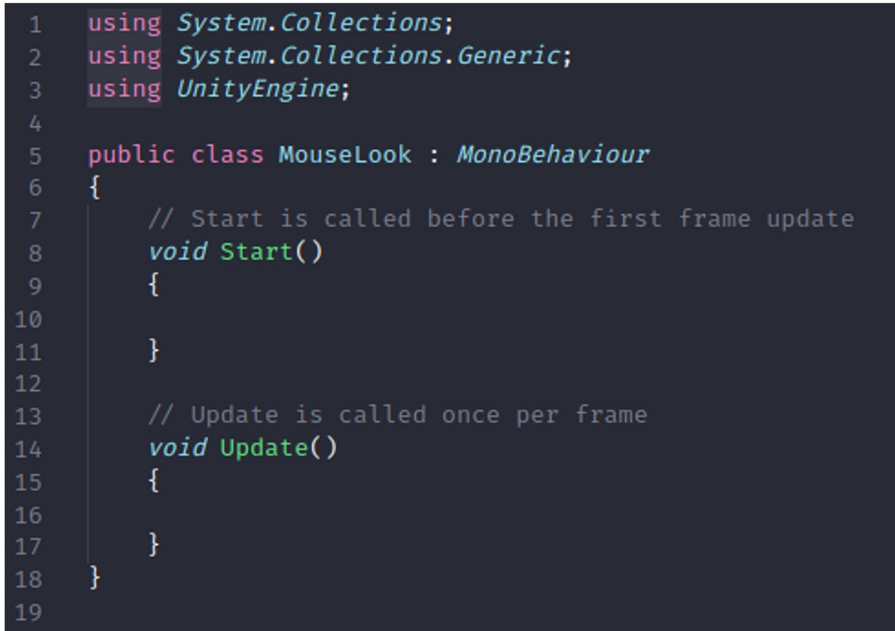
- Base class which exposes the bare minimum of the Unity API (a : b is inheritance, like in C++)
- Includes methods like
  - Start
  - Update
  - FixedUpdate
  - OnBecameVisible
  - OnCollisionEnter
  - … and more



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

public class MouseLook : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

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

    }
}
```

MonoBehaviour

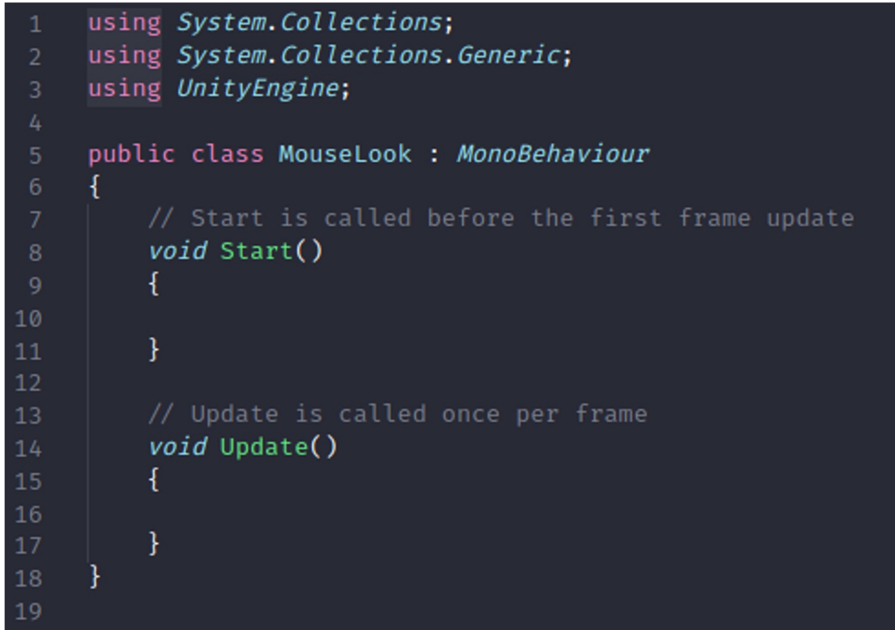
- Base class which exposes the bare minimum of the Unity API (a : b is inheritance, like in C++)
- **Core of writing scripts: Derive from MonoBehaviour and overload whatever functions you need to get the effect that you want.**



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

public class MouseLook : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

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

    }
}

```

MonoBehaviour

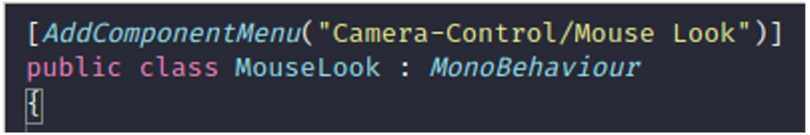
- We will implement a common control scheme: The mouse controls your camera, keyboard controls 4-direction movement.



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

public class MouseLook : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

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

    }
}
```

MouseLook

- Start with the mouse
- First, ergonomics: Move this from the scripts section to the camera control section and give it a descriptive name.



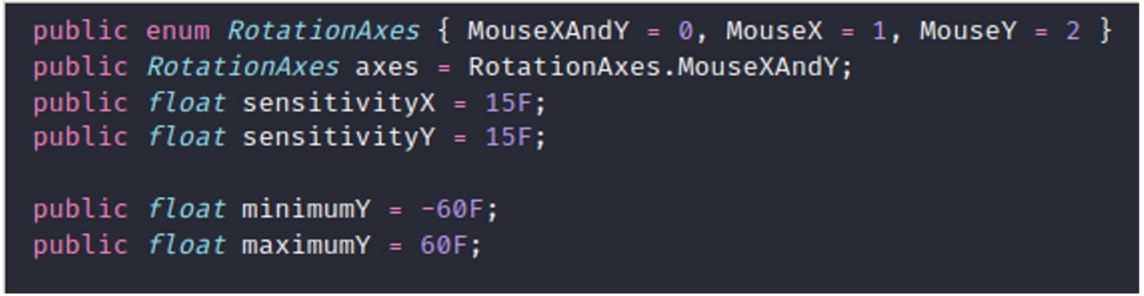
```
[AddComponentMenu("Camera-Control/Mouse Look")]
public class MouseLook : MonoBehaviour
{
```

MouseLook

- Scripts can expose internal variables to the inspector
- This lets game designers -- who might or might not know anything about scripting -- manipulate variables in your script. This is awesome- The fewer people touching code the better.
- Public data values are exposed to the inspector. Let's think about some settings we might want.

MouseLook

- Scripts can expose internal variables to the inspector
- Which axes we can control (With a wrapper Enum around it to make it easier to read)
- Sensitivity for each axis
- Minimum and maximum vertical angle



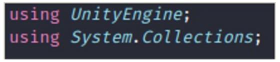
```
public enum RotationAxes { MouseXAndY = 0, MouseX = 1, MouseY = 2 }
public RotationAxes axes = RotationAxes.MouseXAndY;
public float sensitivityX = 15F;
public float sensitivityY = 15F;

public float minimumY = -60F;
public float maximumY = 60F;
```

MouseLook

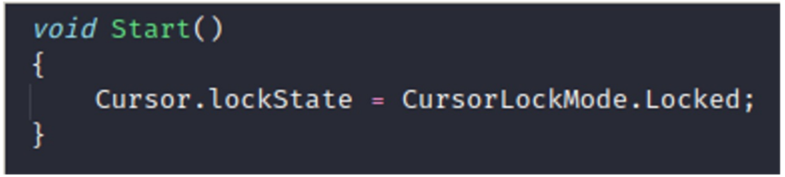
- Private data members keep track of internal state and are not exposed to the inspector.
- In our case we just need one- A float RotationY to make our restrictions on min/max angle easier to read.
- Set its initial value to 0F

MouseLook

- Add the following namespace inclusions
- Start function
  - Runs once, when the script is initialized (Remember that technically this is a component. Kind of.)
  - All we want to do here is keep the user's cursor locked in the game window so it doesn't fly out when they move.



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



```
void Start()
{
    Cursor.lockState = CursorLockMode.Locked;
}
```

MouseLook

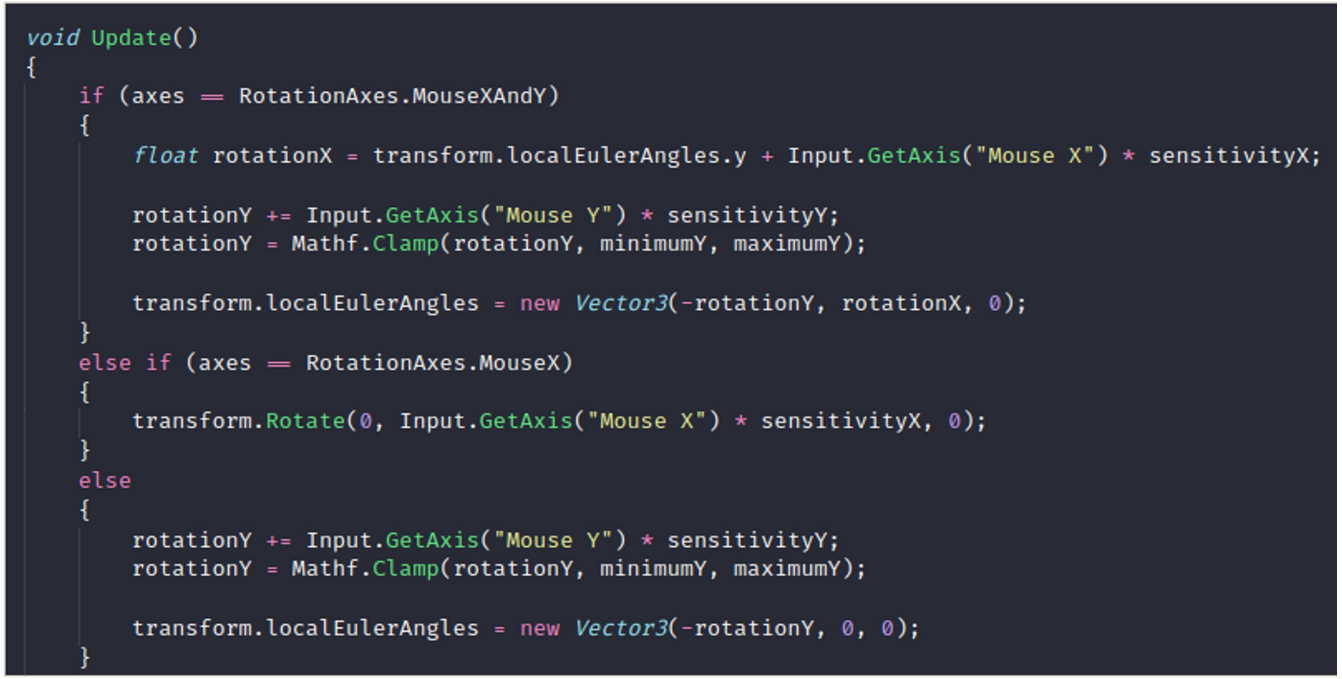
- The Update function
  - Called once per frame

MouseLook

- The Update function
    - Called once per frame
    - This is where we do things that are sensitive to user inputs- Like moving the camera. If the camera only moved on physics updates it would be nauseating.
    - We have 3 cases to deal with just based on settings:
        - X and Y axes
        - Just X
        - Just Y

MouseLook

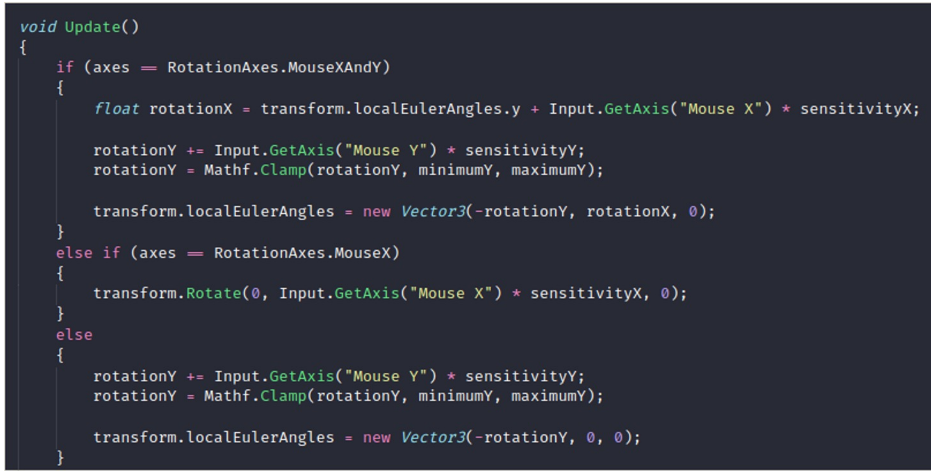
- The Update function



```
void Update()
{
    if (axes == RotationAxes.MouseXAndY)
    {
        float rotationX = transform.localEulerAngles.y + Input.GetAxis("Mouse X") * sensitivityX;

        rotationY += Input.GetAxis("Mouse Y") * sensitivityY;
        rotationY = Mathf.Clamp(rotationY, minimumY, maximumY);

        transform.localEulerAngles = new Vector3(-rotationY, rotationX, 0);
    }
    else if (axes == RotationAxes.MouseX)
    {
        transform.Rotate(0, Input.GetAxis("Mouse X") * sensitivityX, 0);
    }
    else
    {
        rotationY += Input.GetAxis("Mouse Y") * sensitivityY;
        rotationY = Mathf.Clamp(rotationY, minimumY, maximumY);

        transform.localEulerAngles = new Vector3(-rotationY, 0, 0);
    }
```

MouseLook

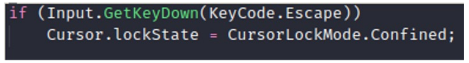
- The Update function
  - Virtual axes: Defined within Unity’s input handling system.
  - Clamp: rotationY is equal to rotation unless it’s greater than minimumY or more than maximumY. Then it’s equal to the corresponding min/max value.



```
void Update()
{
    if (axes == RotationAxes.MouseXAndY)
    {
        float rotationX = transform.localEulerAngles.y + Input.GetAxis("Mouse X") * sensitivityX;

        rotationY += Input.GetAxis("Mouse Y") * sensitivityY;
        rotationY = Mathf.Clamp(rotationY, minimumY, maximumY);

        transform.localEulerAngles = new Vector3(-rotationY, rotationX, 0);
    }
    else if (axes == RotationAxes.MouseX)
    {
        transform.Rotate(0, Input.GetAxis("Mouse X") * sensitivityX, 0);
    }
    else
    {
        rotationY += Input.GetAxis("Mouse Y") * sensitivityY;
        rotationY = Mathf.Clamp(rotationY, minimumY, maximumY);

        transform.localEulerAngles = new Vector3(-rotationY, 0, 0);
    }
```

MouseLook

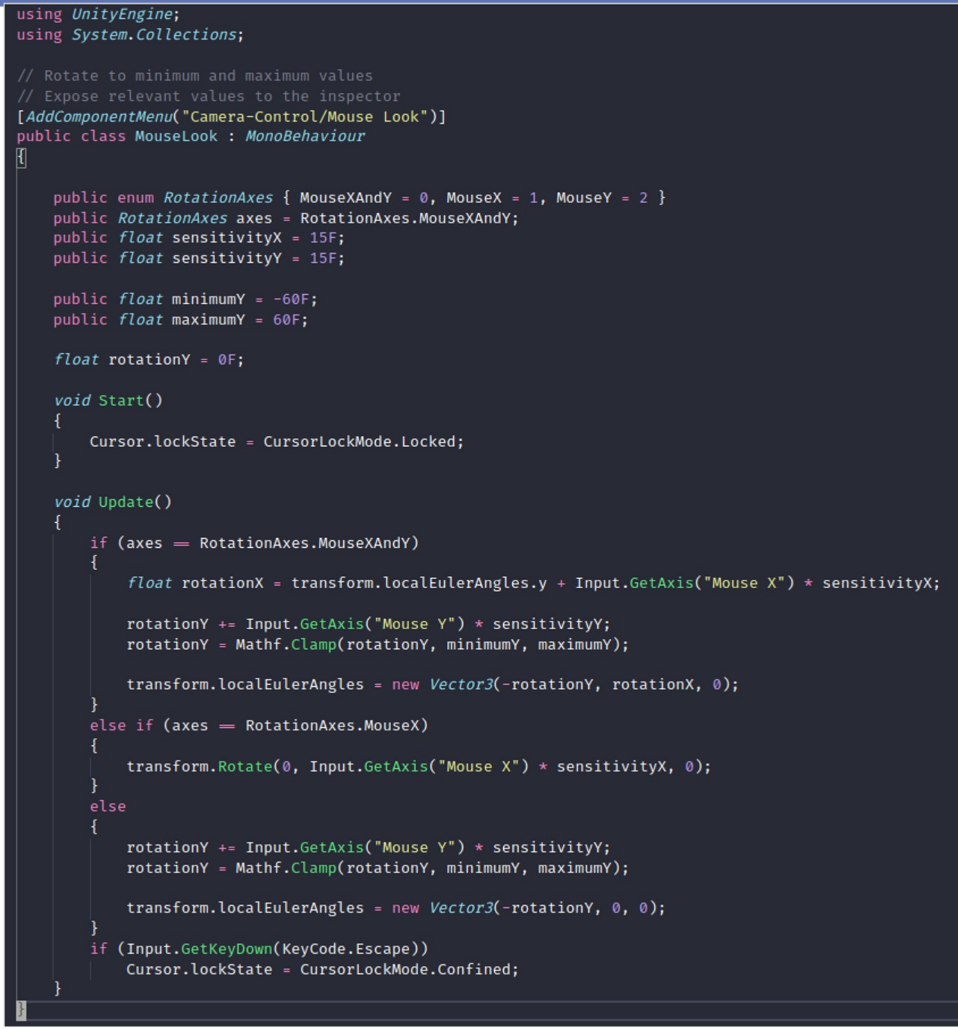
- The Update function
- Escaping the window
- Lets the user out of the game



```
if (Input.GetKeyDown(KeyCode.Escape))
    Cursor.lockState = CursorLockMode.Confined;
```

MouseLook

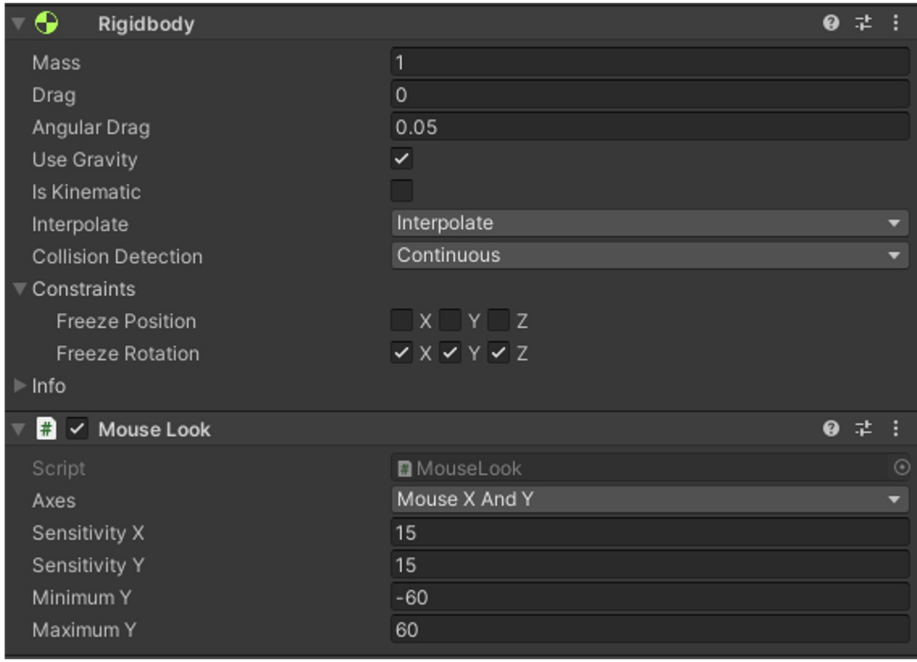
- Full script



```
using UnityEngine;
using System.Collections;

// Rotate to minimum and maximum values
// Expose relevant values to the inspector
[AddComponentMenu("Camera-Control/Mouse Look")]
public class MouseLook : MonoBehaviour
{

    public enum RotationAxes { MouseXAndY = 0, MouseX = 1, MouseY = 2 }
    public RotationAxes axes = RotationAxes.MouseXAndY;
    public float sensitivityX = 15F;
    public float sensitivityY = 15F;

    public float minimumY = -60F;
    public float maximumY = 60F;

    float rotationY = 0F;

    void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;
    }

    void Update()
    {
        if (axes == RotationAxes.MouseXAndY)
        {
            float rotationX = transform.localEulerAngles.y + Input.GetAxis("Mouse X") * sensitivityX;

            rotationY += Input.GetAxis("Mouse Y") * sensitivityY;
            rotationY = Mathf.Clamp(rotationY, minimumY, maximumY);

            transform.localEulerAngles = new Vector3(-rotationY, rotationX, 0);
        }
        else if (axes == RotationAxes.MouseX)
        {
            transform.Rotate(0, Input.GetAxis("Mouse X") * sensitivityX, 0);
        }
        else
        {
            rotationY += Input.GetAxis("Mouse Y") * sensitivityY;
            rotationY = Mathf.Clamp(rotationY, minimumY, maximumY);

            transform.localEulerAngles = new Vector3(-rotationY, 0, 0);
        }
        if (Input.GetKeyDown(KeyCode.Escape))
            Cursor.lockState = CursorLockMode.Confined;
    }
}
```

MouseLook

- Adding the script to the player
- Click Add Component in the inspector with the Player object selected. Search “MouseLook”. Add component. Set values to taste.



Rigidbody

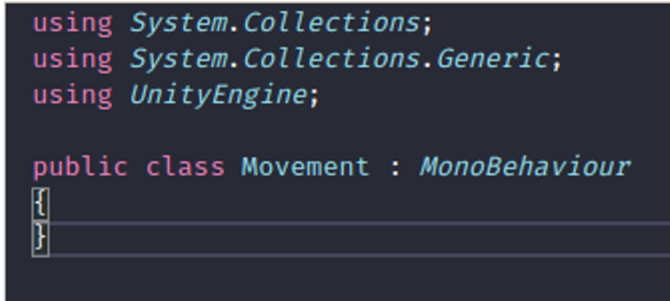
Mass	1
Drag	0
Angular Drag	0.05
Use Gravity	✓
Is Kinematic	
Interpolate	Interpolate
Collision Detection	Continuous
Constraints	
Freeze Position	X Y Z
Freeze Rotation	✓ X ✓ Y ✓ Z
Info	

Mouse Look

Script	MouseLook
Axes	Mouse X And Y
Sensitivity X	15
Sensitivity Y	15
Minimum Y	-60
Maximum Y	60

KeyboardMovement

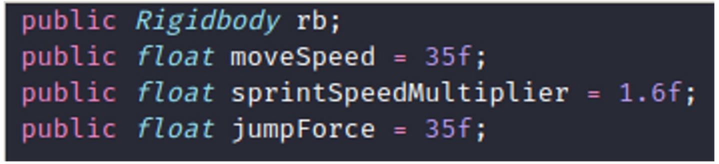
- 4 direction movement script
- Some other Unity API functions
- Manipulating other components



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

public class Movement : MonoBehaviour
{
}
```

KeyboardMovement

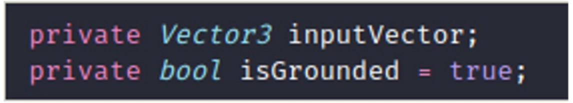
- Exported values:
    - The rigidbody we want to move around
    - Some design elements that should be able to be tweaked without opening a text editor.



```
public Rigidbody rb;
public float moveSpeed = 35f;
public float sprintSpeedMultiplier = 1.6f;
public float jumpForce = 35f;
```

KeyboardMovement

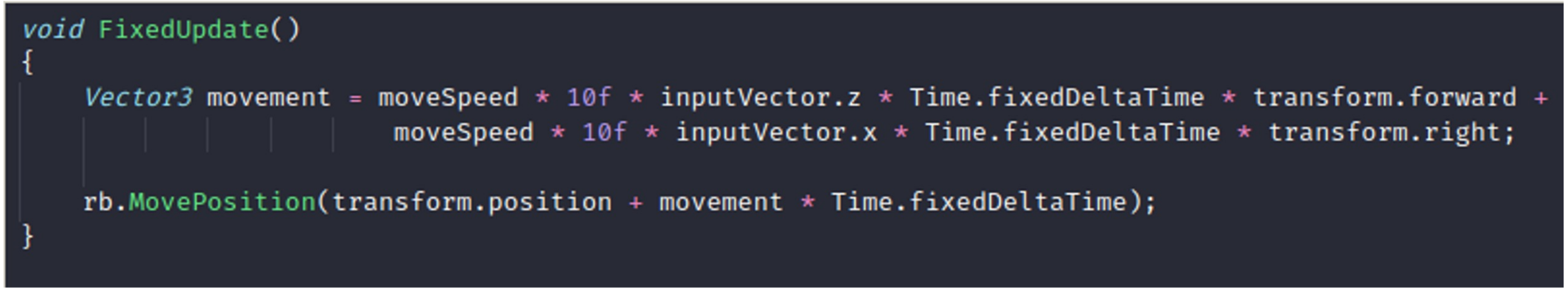
- Private values:
  - inputVector
  - isGrounded
- These change often and it would make no sense for them to be public.



```
private Vector3 inputVector;
private bool isGrounded = true;
```

KeyboardMovement

- Movement
- Use FixedUpdate: Although we could try to scale our input vector and make sense of it, time only goes so small and it’s awkward to do. Doing it a fixed number of times per second keeps our player in-sync with the physics engine and avoids bugs.



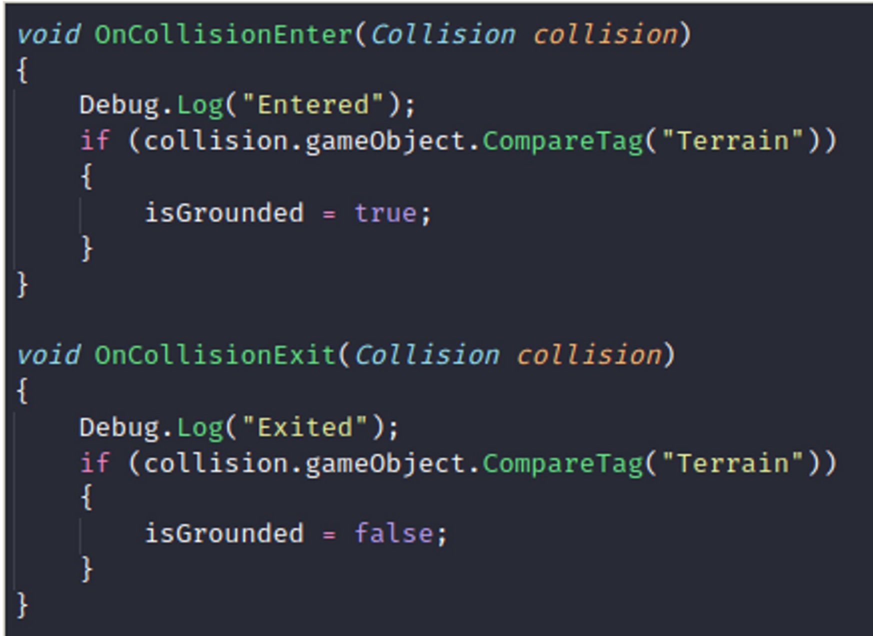
```
void FixedUpdate()
{
    Vector3 movement = moveSpeed * 10f * inputVector.z * Time.fixedDeltaTime * transform.forward +
                        moveSpeed * 10f * inputVector.x * Time.fixedDeltaTime * transform.right;

    rb.MovePosition(transform.position + movement * Time.fixedDeltaTime);
}
```

KeyboardMovement

- Movement
- Use FixedUpdate: Although we could try to scale our input vector and make sense of it, time only goes so small and it’s awkward to do. Updating it a fixed number of times per second keeps our player in-sync with the physics engine and avoids bugs.
- This doesn’t get input yet- Just applies the update that we’ll get in the Update() function.

KeyboardMovement

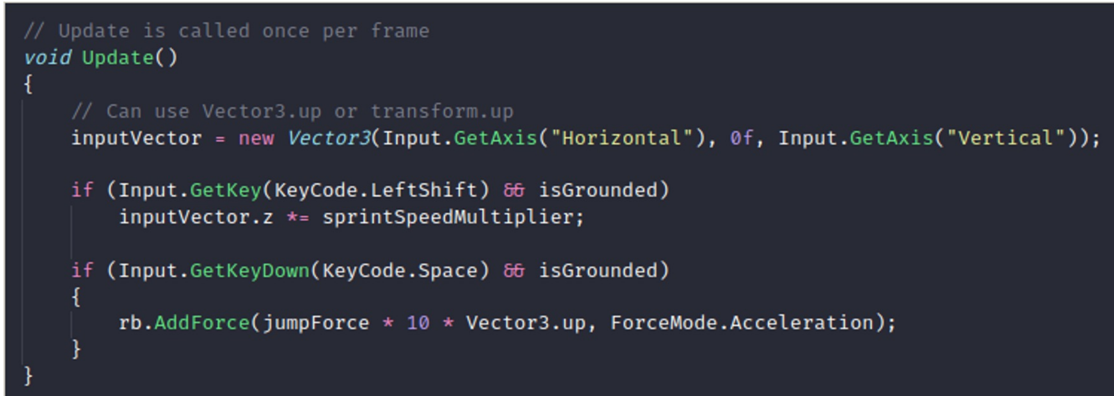
- Jumping and sprinting
  - Sprinting will just increase the speed at which we run.
  - Jumping will apply a force to the rigidbody.
- First, collision detection: We want to only let the player jump if they’re standing on the ground.



```
void OnCollisionEnter(Collision collision)
{
    Debug.Log("Entered");
    if (collision.gameObject.CompareTag("Terrain"))
    {
        isGrounded = true;
    }
}

void OnCollisionExit(Collision collision)
{
    Debug.Log("Exited");
    if (collision.gameObject.CompareTag("Terrain"))
    {
        isGrounded = false;
    }
}
```

KeyboardMovement

- Jumping and sprinting
  - Sprinting will just increase the speed at which we run.
  - Jumping will apply a force to the rigidbody.
- Finally, input handling in the update function.



```
// Update is called once per frame
void Update()
{
    // Can use Vector3.up or transform.up
    inputVector = new Vector3(Input.GetAxis("Horizontal"), 0f, Input.GetAxis("Vertical"));

    if (Input.GetKey(KeyCode.LeftShift) && isGrounded)
        inputVector.z *= sprintSpeedMultiplier;

    if (Input.GetKeyDown(KeyCode.Space) && isGrounded)
    {
        rb.AddForce(jumpForce * 10 * Vector3.up, ForceMode.Acceleration);
    }
}
```

KeyboardMovement

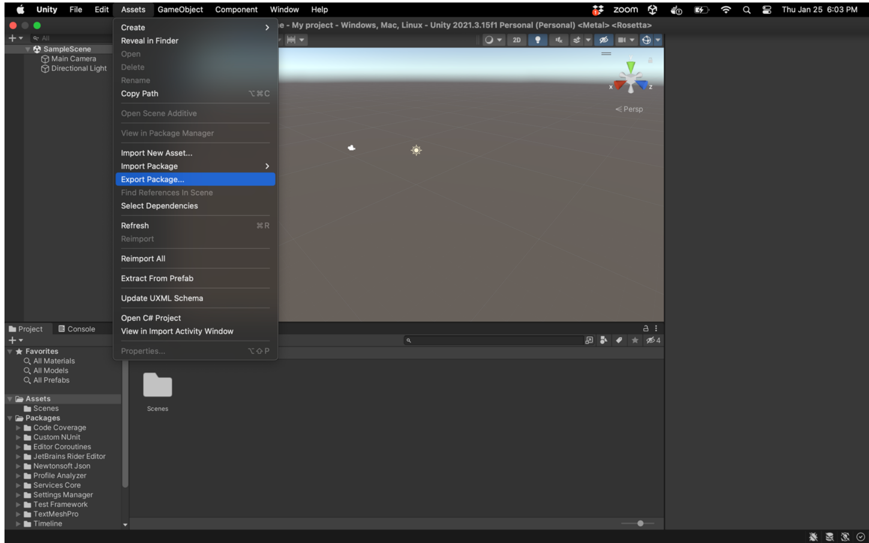
- Takeaway: Scripts let us extend the functionality of Unity by supplying “hooks” that can be used to manipulate already existing systems and even create entirely new ones.

Outline

- Scripting
- Exporting projects
  - How to turn in the first lab homework
- VR

Exporting

- Go to Assets->Export Package
- Leave everything checked
- Save somewhere you'll remember as **.unitypackage**
- **This is what you'll turn in.**



Outline

- Scripting
- Exporting projects
- VR

State of VR in Unity in 2024

- The VR ecosystem is split between major vendors
- Unity does make an effort to unite this stuff
- This stuff is of interest to students with a VR headset on hand immediately
- Quite a lot better than it was 4 years ago!

VR Standards and SDKs

- Oculus
  - Meta’s headsets
- Playstation VR
- OpenXR
  - Everyone else (Vive, Valve, HoloLens, Oculus)

Outline

- Scripting
- Exporting projects
- VR

- I will be here for the rest of the class time to answer questions and debug scripts
- Please don’t hesitate to ask questions if you’re stuck!