Game Programming

Sci-Fi Demo: Lecture I

Izzet Fatih Senturk

Setup

- Create a new project: Sci-Fi Demo in 3D
- Download assets from: https://e1.pcloud.link/publink/show?code=XZiOaJZccqPinVfpaVqh1vohIekfBt2QMGV
- Import the asset pack to the project
  - Make sure all is selected and then import
- Fix Errors in the scripts
  - Change [RequireComponent(typeof (GUITexture))] to [RequireComponent(typeof(Texture))]
  - Change GUIText to Text (add: **using** UnityEngine.UI)
- Select Level from Game Models folder and drag & drop to the Hierarchy window

Lighting in Unity

- The Level already comes with a Sun Light
  - Remove Directional Light from the Scene
- Set Camera position (5,2,8) and rotation (0,180,0)
- Go to Level -> Market -> Lights -> SunLight
  - Position of the light doesn't matter, it's the sun light
  - Set Light Intensity: 2
  - Change Color as you wish
- Right click Lights and add a new Light -> Point Light
  - Place it above the fish sign
  - Set Range, Intensity, and Color
- Add two Light -> Reflection Probes and set their bounds
- Select Market Floor Tile Sand material and adjust Metallic, Smoothness, Normal Map

Player Setup

- Create 3D Object -> Capsule
  - Rename: Player
  - Set the position
- Character controller is mainly used for third-person or first-person player control that does not make use of RigidBody physics
  - Allows you to easily do movement constrained by collisions without having to deal with a rigidbody
- Remove CapsuleCollider and add a new component: CharacterController instead
- Create a Scripts folder under Assets in the Project window
- Create a new C# script under Scripts named as Player_sc
  - Add the script to the Player

Player Movement

- Create a variable as a handler for CharacterController
- Create a variable for velocity and serialize it

```
private CharacterController _controller;
[SerializeField]
1 reference
private float _speed = 3.5f;
```

- Get CharacterController component at Start

```
void Start()
{
    _controller = GetComponent<CharacterController>();
}
```

- Get direction, calculate velocity and apply movement

```
void Update()
{
    Vector3 direction = new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical"));
    Vector3 velocity = direction * _speed;
    _controller.Move(velocity * Time.deltaTime);
}
```

Adding Gravity

- The Player is currently hovering above the ground. Create a gravitational force to drag the Player down

```
private float _gravity = 9.81f;
```

- Apply the gravitational force on the y axis at Update before movement

```
void Update()
{
    Vector3 direction = new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical"));
    Vector3 velocity = direction * _speed;
    velocity.y -= _gravity;
    _controller.Move(velocity * Time.deltaTime);
}
```

Code Improvement

- Define a new method to calculate movement and simplify Update function

```
void Update()
{
    CalculateMovement();
}

1 reference
public void CalculateMovement()
{
    float horizontalInput = Input.GetAxis("Horizontal");
    float verticalInput = Input.GetAxis("Vertical");
    Vector3 direction = new Vector3(horizontalInput, 0, verticalInput);
    Vector3 velocity = direction * _speed;
    velocity.y -= _gravity;
    _controller.Move(velocity * Time.deltaTime);
}
```

Local Space vs World Space

- Make sure the camera moves with the movement of the Player
  - Drag the Main Camera and make it the child of the Player
- The first person look
  - Select the Main Camera and set x,y,z = 0
  - Zoom in on the Player and see that the camera is in the Player
  - Move the camera above (at the level around the Player's head) (y=~.66)
- Save and test it
  - The movement coices are backwords
  - We will fix it now

Local Space vs World Space

- Look towards the proper direction
  - Set y rotation on the Main Camera to 0
- Convert movement direction from local space to world space
  - Use Transform.TransformDirection method from the scripting API
- Set Player rotation 180 to look towards the shark at the beginning

```
public void CalculateMovement()
{
    float horizontalInput = Input.GetAxis("Horizontal");
    float verticalInput = Input.GetAxis("Vertical");
    Vector3 direction = new Vector3(horizontalInput, 0, verticalInput);
    Vector3 velocity = direction * _speed;
    velocity.y -= _gravity;

    velocity = transform.transform.TransformDirection(velocity);
    _controller.Move(velocity * Time.deltaTime);
}
```

Mouse Look

- Look left and right
  - When the mouse moves at the x axis
- Look up and down
  - When the mouse moves at the y axis
- Modify Player's Transform.Rotation with Mouse X and Mouse Y
  - Create a new C# script: LookX_sc
  - Attach the script to the Player

```
[SerializeField]
1 reference
private float _sensitivity = 1f;

// Update is called once per frame
0 references
void Update()
{
    float _mouseX = Input.GetAxis("Mouse X");
    Vector3 newRotation = transform.localEulerAngles;
    newRotation.y += _mouseX * _sensitivity;
    transform.localEulerAngles = newRotation;
}
```

Mouse Look

- Create a new C# script: LookY_sc
  - Attach the script to the Player (doesn't quite work actually!)
- How can the Player look up and down while moving?
- Create an empty Game Object for LookY: LookY
  - Put LookY in the Player as a child
  - Set x,y,z of LookY = 0
  - Put the Main Camera inside LookY as a child
  - Attach the LookY script to LookY game object

```
[SerializeField]
1 reference
private float _sensitivity = 1f;

// Update is called once per frame
0 references
void Update()
{
    float _mouseY = Input.GetAxis("Mouse Y");
    Vector3 newRotation = transform.localEulerAngles;
    newRotation.x += _mouseY * _sensitivity;
    transform.localEulerAngles = newRotation;
}
```