CS135 Winter 2022 - Lab 1

Lab 1 was a Collaboration between Diego Vega (dvega007) and Sumachai Suksanguan (ssuks001)

Parts and Explanation

All Assignments were completed and tested within Unity. Part 1 was done primarily with the FPSController, while Part 2 was developed and tested using OVR Input with an Oculus Quest 2.

Part 1

Part 1 was an Introduction into Unity, and as a result, the materials and instructions given to us made the experience easy.

Part 1.1 - The Room

Create a Cubic Room. Add Lighting to the Room. Add a Planet and a Moon orbiting it. Script both the Lighting and the Planet, as well as including a Teleport and Exit Feature. This was really straightforward, as most of the complexities came from the logic required to script the light!

Light Switch

I wanted to be able to use an array of light colors, but instead made it work by just doing a simple switch case. It will cycle through the colors white, red, blue, green, and white!

Orbit

This one was also really self explanatory. As all you had to do was create the objects and then in the script, add the line `this.transform.Rotate(new Vector3 (0,2,0));` into the `void Update()` function.

Room Switch and Quit

This one was also really simple. After creating our second room (in Part 2), set the coordinates of the transformation to that room and bind it to the '2' key with `if(Input.GetKeyDown("2"))`. Likewise, we bind the coordinates of room 1 to the '1' key using `if(Input.GetKeyDown("1"))`. We use the same transformation function as in Orbit, but rather than the Rotation, we want to change the position.

It's important to note that this script should be attached to the FPSController, as its behaivor is what we are automating.

Then, we bind an application quit to the `Esc` key, and use a bit of C# and Unity Commands to exit the application.

Part 1.2 - The Room 2

This one is similar to Part 1, but we are creating a room using cubes instead of Planes. By changing the scaling of the cubes to an extremely low value, we are able to make them look 2D, but viewable from all angles (as opposed to planes). The Room will be a hexagon with a slanted roof. In this case, its a hexagonal room with a slab of concrete forced into the ceiling at an angle, sue me I'm lazy.

We then add textures to the Walls!

Room Switch

Similar to Part 1.1, we need to implement the teleport back to Room 1 Feature. This is simple, as we just include the `if(Input.GetDownKey("1"))` into our previous room_switch script as mentioned before.

Trigger Zone

We create a box collider that is placed on the floor, and when triggered with another Collider (in this case, our FPSController), will drop a ball on top of us. Creating the collider was simple, and then simply adding the void function below allowed us to drop the ball whenever it was triggered:

```
void OnTriggerEnter(Collider other){
  ball.GetComponent<RigidBody>().useGravity = true;
}
```

> *Note: Ball refers to a `GameObject` called Ball referenced earlier in the code.*

Part 2

Part 2 was creating the Cat and Mouse game, with significantly less hand-holding. All this was done by creating a Open-Box using several thin Cube objects, a skyline, and a directional light emulating the sun.

Part 2.1 - The Room 3

Oh boy. Was this a doozy. First of all, creating the Room, Skybox, and all that Other Stuff was the easy part. Even the texture mapping was easy. The hard part of the game was actually scripting all the logic that had to go with it! In doing so, we learned how to reference variables in other scripts, so that it doesn't just feel like you're working through on script at a time. Let's break down the logic, shall we?

Light Logic

The prompt asked us to do at least 4 Cubes. So, we did 5 (Go Further Beyond!). It also said that every 3 seconds, it should randomly turn on light on. Alright, let's start small (we will implement trigger zones later!) Let's create the logic for those lights so that every three seconds a random light will turn on! In this case, we will be using `public float timeLeft` as a variable to track how much time we have left.

```
void allOff(){
  light1.enabled = false;
  ...
  light5.enabled = false;
 }

 void turnON(int lightNum){
  switch(lightNum){
    case 1:
      light1.enabled = true;
      break;
    ...
    case 5:
      light5.enabled = true;
    default:
      break;
    }
}

void Start(){
  allOff();
  int temp = Random.Range(1,6);
  turnON(temp);
}

void Update(){
  timeLeft -= Time.deltaTime;
  if(timeLeft < 0){
    timeLeft = 3;
    allOff();
    int temp = Random.Range(1,6);
    turnON(temp);
}
```

> *Note: We are using Unity's `Random` function, rather than the native C# one! Note that `Random.Range(int x, int y)` has gives a range that is inclusive x and exclusive y!*

Alright. So the code is pretty straight forward. At start, turn on a random light, 1 through 5. Then, every frame, subtract time passed, and if the time remaining ever drops under 0, reset the timer, turn off all lights, and turn on a new random one, 1 through 5. That was on of the easier ones...

Trigger Logic

The trigger logic was the logic that took up the majority of the time. Anyways, we created 5 trigger zones that are directly on the cubes and lights, and we know that if our Collider is within them, and they press 'A', they should receive a point. That point will then be updated on a scoreboard, that will be referenced later. Much like above, we are using `public game my_game` to allow the trigger script to access the functions and member variables of the entire game, such as `timeLeft` and `score`.

```
void OnTriggerStay(Collider other){
  if(OVRInput.GetDown(OVRInput.Button.One)){
    my_game.timeLeft = -1;
    my_game.score++;
    my_game.scoreboard.text = "Score: " + my.game_score;
  }
}
```

> *Note: We set `my_game.timeLeft` to -1 so that when it is referenced later in the `void Update()` function it will flash a new light automatically! Remember, a new light will flash as soon as `timeLeft` is less than 0, hence why we set it to -1 in here.*

Also, in order to avoid having multiple trigger zones active at once, we utilize the `void allOff()` and `void turnON(int lightNum)` functions to enable and disable our trigger zones:

```
void allOff(){
  light1.enabled = false;
  trigger1.enabled = false;
  ...
  light5.enabled = false;
  trigger5.enabled = false;
}
```

```
void turnON(int lightNum){
  switch(lightNum){
    case 1:
      light1.enabled = true;
      trigger1.enabled = true;
      break;
    ...
    case 5:
      light5.enabled = true;
      trigger5.enabled = true;
      break;
    default:
      break;
   }
}
```

Scoreboard Logic

Scoreboard Logic is actually pretty simple. We simply access the scoreboard scripting by incuding:

```
public TextMesh scoreboard;

scoreboard.text = "Your Text Here";
```

Scoreboard will just need a score variable that is incremented whenever the user scores a point!

Quit Logic

This one is also straightforward, as it just relies on knowledge of OVRInput and similar logic from Part 1. All we need to do is add a conditional statement in `void Update()` that checks to see if 'B' is pressed down at any point.

```
void Update(){
  ...
  if(OVRInput.GetDown(OVRInput.Button.Two)){
    #if UNITY_EDITOR
      UnityEditor.EditorApplication.isPlaying = false;
    #else
      Application.Quit();
    #endif
  }
}
```

Part 2.2 - VR Experiences

We both tried Multiple VR Experiences as partners. Diego wrote about his first time experiences with *Superhot VR* and *Beat Saber*, while Chai wrote about his experiences in *Zenith: The Last City* and *Cook-Out: A Sandwich Tale*. Detailed experience information can be found here.