Basil El-Hindi

Project 3 Analysis Document

I knew tackling this project would have many moving components, and so it would be important to be consistently using a clean, modular framework as I built the code from the ground up. While each individual component is easy enough to program in a vacuum, dealing with putting all the pieces together turned out to be a massive headache.

To begin the project, I first introduced the basics that would be needed for movement and playing the game, and the character I created closely followed the FPS character we made in Project 2. However, from there I had many diverging tasks.

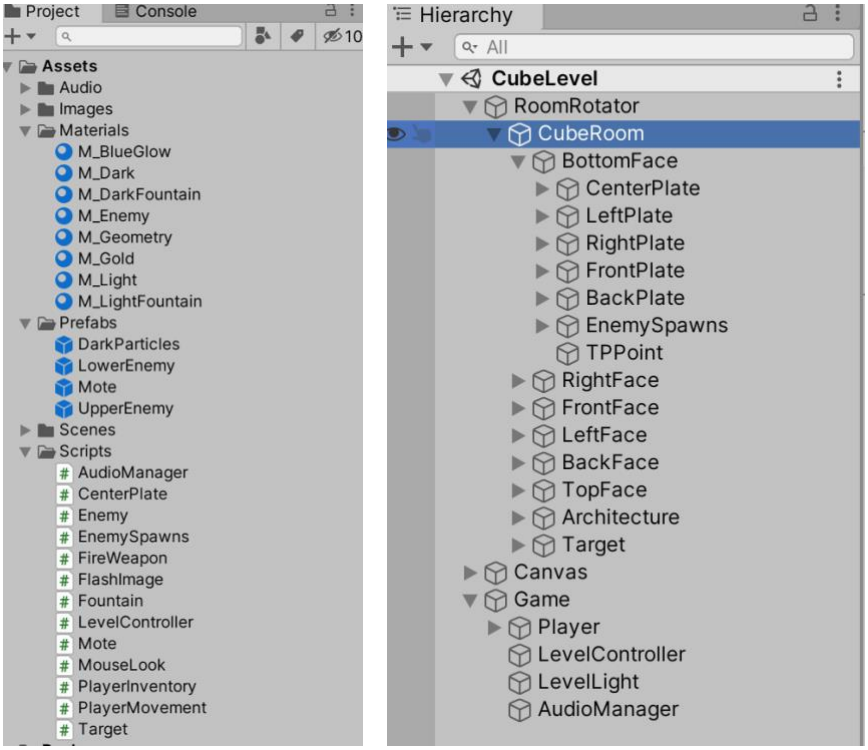
Ultimately, I knew I would be building the cube I had in steps, and since each face of the cube was functionally identical, I decided it would be best to first program everything I could on a singular face before moving on to program them all together. The first component after getting movement with a character controller was enemies that die on raycast hit, but then additionally the player affects items the enemy drops, so the player has to communicate to the enemy which then creates objects with those specifications. I also had to implement an inventory for the player, and that logic took some time as there are certain rules around holding the items that I implemented according to the source material, shown in the screenshot below.

```
public void AddMote(bool shadow)
{
    if(!_fullMotes)
    {
        AudioManager.Instance.PlaySound(_motePickupSound);
        _holdingMotes = true;
        _moteTimer = 29.9f;
        if(shadow == _shadow)
        {
            _motesHeld++;
            if(_motesHeld == _maxMotes)
            {
                SetFullMotes();
            }
        }
        else
        {
            _shadow = shadow;
            _motesHeld = 1;
        }
    }
}

void SetFullMotes()
{
    _fullMotes = true;
    _fullIndicator = Instantiate(_Mote,
        _fullMotePoint.position, transform.rotation,
        gameObject.transform);
    _fullIndicator.GetComponent<Mote>().SetDark(_shadow);
    _weapon.SetDisabled(true);
    _persp.TogglePerspective();
}

void RemoveMotes()
{
    _motesHeld = 0;
    _holdingMotes = false;
    if(_fullMotes)
    {
        _persp.TogglePerspective();
    }
    _fullMotes = false;
    _weapon.SetDisabled(false);
    Destroy(_fullIndicator);
}
```

The next mechanic to implement was what drove the rotation of the room to progress towards the goal, and that concept took communication between 4 different classes to coordinate what needed to happen. After that happened, I needed to add the actual objective in to begin testing, and that required building out the whole 3D level, which had many bugs that I spent a lot of time dealing with. Ultimately, I had a level controller manage the level, including storing the information about the current state of the cube and where the next rotation would be based on the player actions. Lastly, I added enemy spawns, and between hand adding the enemy spawns and the geometry blocks the level itself took a long time to create. The coroutine used to rotate the level and change the game states is shown here.

```
IEnumerator RotateRoom()
{
    yield return new WaitForSeconds(1);
    _flashImage.StartFlash(1, 1, Color.white);
    _faces[(int)_currentFloor].GetComponentInChildren<CenterPlate>().SetActive(false);
    Quaternion startingRotation = _cubeRoom.transform.rotation;
    AudioManager.Instance.PlaySound(_teleportSound);
    Vector3 targetRotation = Vector3.zero;
    float elapsedTime = 0f;
    if(_nextRotation == Rotation.Forward)
    {
        targetRotation = 90f * Vector3.right;
    }
    else if(_nextRotation == Rotation.Backward)
    {
        targetRotation = -90f * Vector3.right;
    }
    else if(_nextRotation == Rotation.Clockwise)
    {
        targetRotation = -90f * Vector3.forward;
    }
    else if(_nextRotation == Rotation.Counterclockwise)
    {
        targetRotation = 90f * Vector3.forward;
    }
    else
    {
        Debug.Log("Error determining rotation");
    }
    SetNewFloor();
    _nextRotation = Rotation.None;
    while(elapsedTime < 1f)
    {
        elapsedTime += Time.deltaTime;
        _cubeRoom.transform.rotation = Quaternion.Slerp(startingRotation, Quaternion.Euler(targetRotation), elapsedTime);
        yield return new WaitForEndOfFrame();
    }
    _transitioning = false;
    _player.GetComponent<CharacterController>().enabled = false;
    _player.transform.position = _tpPoints[(int)_currentFloor].position;
    _player.GetComponent<CharacterController>().enabled = true;
    ActivateFountains();
    yield return 0;
}
```

I kept my project clean and organized throughout. Each of the faces were identical save for some of their values, and the rest of the architecture was fit under the grouping game object, as well as keeping the project assets organized for easy finding.



Lastly, I spent some time polishing the VFX and SFX. It didn't end up looking as polished as I would have liked, but the time I spent coding was able to get a lot of complex development done. The background music is a youtube rip of the actual soundtrack used in the Destiny level, and the sound effects were made on ChipTone.