Game Title : COVID RUSH

Group 1

Member List

吳岱容 / 310551067
李諭樹 / 310552024
呂思函 / 0716207

Introduction

【Immerse Experience】

The story fits current reality. Plots, music, and game mode will give players nervous feelings and excitement. They can easily immerse themselves in the game world, and have similar but more exciting experiences different from reality.

【New Game Mode】

Players can not only interact with NPCs but also collect supplies in game. There are several significant characteristics in our game, such as music switches terrains combined with Tag's (鬼抓人) elements, and getting to the safe regions to avoid infected, etc.

【Fascinating Scenes】

Because changing terrains and safe regions is the main element in COVID RUSH. There will be several fascinating scenes switching during the game.

Game Story

Severe disease has stroken the world for a period of time. The governments of countries try hard to have citizens vaccinated. They also strongly appeal to their citizens to wear masks in order to protect themselves and others.

You have gradually noticed the situation of spreading disease, and you also find out that some regions are severe and some are not. People tend to get rid of dangerous regions and swarm into safe regions. While they enter the safe region, they still try to grab more supplies they can.

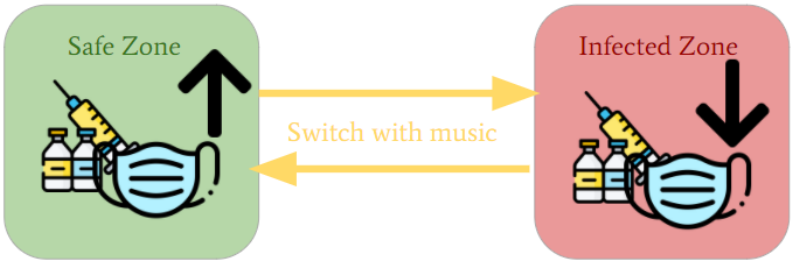
Watching those raving behavior, you are conscious of making yourself out of threat right now!

Game Control

Our game is played mainly with mouse and keyboard (WASD).

1. Player Control:
   The player can use WASD keys to control their character to move left and right and press space key to let the character jump. Also, the player can move the mouse horizontally to control the character's orientation.
2. Camera Control :
   The player can move his mouse to control the camera's orientation, and scroll the wheel to control the distance.

In our game, there are safe zones (green) and infected zones (red). The player needs to avoid infected zones and stay on the safe one.



The safe zones and infected zones will switch with the music. If the player is in infected zones, their HP or supply will decrease,and if they become zero, the game will be over, but if the player is in safe zones, he will be fine. The game's target is to find the goal in the game scene or to live until the song is over.

There is some supply to help the player survive. Also, there are infected enemies which chase the player if he gets too close to them. If the infected enemies touch the player, the player's HP will decrease.

System Configuration(player, npc, props)

1. props :
   a. Masks and Needles are the keys to get through the game
   b. Vaccine is the prop that can increase the player's life value(time of game)
   c. different amount in three levels :
      i. City (largest scene)
         1. Mask : 5
         2. Needle : 5
         3. Vaccine : 10
      ii. Grass (scene with traps)
         1. Mask : 5

2. Needle : 5
3. Vaccine : 5

iii. Crypt (smallest scene)

1. Mask : 5
2. Needle : 5
3. Vaccine : 5

2. tags : were set to handle different props
    a. Global
        i. Red : step on the red bricks will minus the player's life value
        ii. Green : green bricks are safe zones
        iii. Props_Facemask : UI will show that the facemask icon is increasing
        iv. Props_Needle : UI will show that the needle icon is increasing
        v. Props_Vaccine : UI will show the increasing life value
        vi. Player
        vii. Enemy
            1. Enemy_Facemask : UI will show that the facemask icon is decreasing
            2. Enemy_Needle : UI will show that the needle icon is decreasing
    b. Grass
        i. Water : minus life value if the player go on water and traps

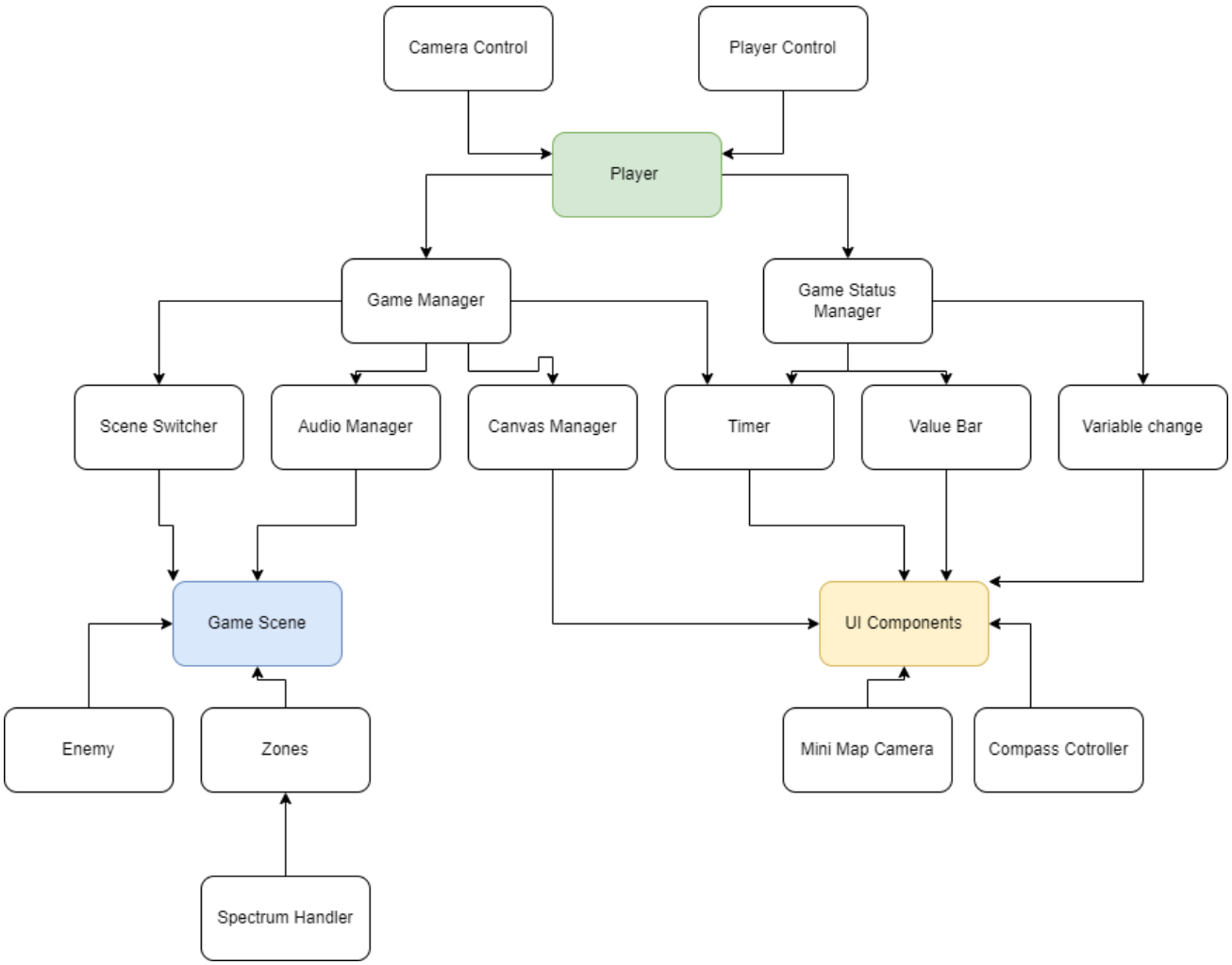
Game System Architecture

Our system architecture can be roughly splitted into three parts, **the player**, **user interface (UI) components**, and the **objects in the game scene**. The player object is controlled by the player control script and the camera script, and it will respond with the user's control and interact with our game to make effect on some of the game objects.

The game status is mainly managed by the two managers, **Game Manager** and **Game Status Manager**. These two manage the smaller components and interact with each other in an **event-driven-development** way, which we will explain later in the technical section. These smaller components and managers build up our game scene and UI, and also change them according to the current status.

In the game scene, there also are enemies which will chase the player. The concept of the movement script will also be introduced in the technical section.

The different types of zones in the game scene are controlled by **Spectrum Handler**, which will listen and respond to the sound in the scene. This component is used to control the bricks' status, since it is not affected by the player's action (only related to the background music), we separate it from the two managers.



Game System component description

Camera Control

Control the player camera. The camera is attached behind the player character. Users can control the camera using their mouses.

Player Control

The script that controls the player character's movement and animation states. It is controlled by the inputs of the keyboard or mouse.

Game Manager

The highest manager in our structure. It is responsible for controlling the game flow.

Game Status Manager

Interact with each other and notify some registered functions according to the status change.

Scene Switcher

There are several scenes in the first page (i.e. start menu). This switcher controls the active scene and registers the game scene changed related functions.

Audio Manager

Managing the audio sources we use in the game scene, and registering the functions related.

Canvas Manager

Managing some of the UI components (ex. dashboard), and registering the functions related.

Timer

Handling the timer on the right corner, which notifies the player remaining time, and registering the functions related.

Value Bar

Presenting the value changes with the bar (HP and item values), and registering some related functions.

Variable Change

Presenting the variables' value with text, and registering some related functions.

Enemy

Enemies in the game scene, which will chase the player, and if the player gets too close to them, the player's HP and items will decrease. The algorithm of chasing will be described in the technical section.

Zones

The green and red squares on the game scene floor. They will switch types with the music, and this action is controlled by the **Spectrum Handler**. If the player stands on green ones, they will be safe, but if he stands on the red ones, his HP and items will decrease.

Spectrum Handler

Catching the spectrum data in the game scene and controlling the square types. Details will be described in the technical section.

Mini Map Camera

Showing the mini map on ui.

Compass Controller

Showing the orientation of the player character is facing.

Technical Section

Event-driven development (EDD)

Different from HW3, we create our own event center. The main reasons include following 2:

1. Some components may be destroyed during scene transition while some may not, which leads to creating unrefered game objects. See figure .
2. Event-subscribed orientation saves time on tracing components. See detail below.

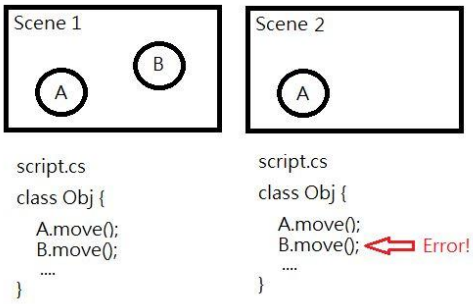


Figure . Class Obj has 2 objects A & B. On the right hand side (i.e. in scene 2) B is removed, but we still need script.cs to control A. In this case, script will crush because we can not access B (with NullException)

The mechanism of EDD shows in figure . The left hand side is called “publisher”, in contrast to the right hand side is called “subscriber”. Once there are publishers’ variables updating, the updating values will be pushed to some topics(i.e. EventType). At the same time, the subscribers who registered to those topics will receive these new values. The implementation of Event Manager could be done by a simple manipulation on a dictionary object.

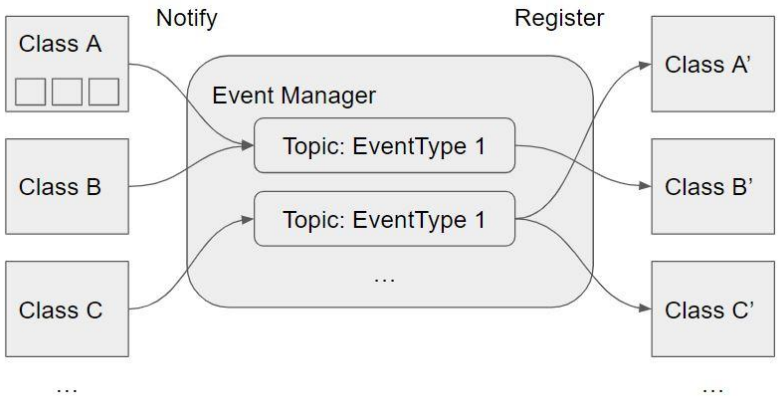


Figure . The mechanism of EDD. The smaller blocks in Class A represent instances.

Now let’s look back at figure if we apply EDD on this case. Figure shows the result. Obviously, whether B is in scene 2 or not won’t affect the execution result, because both A and B are subscribers, while class Obj is a publisher.

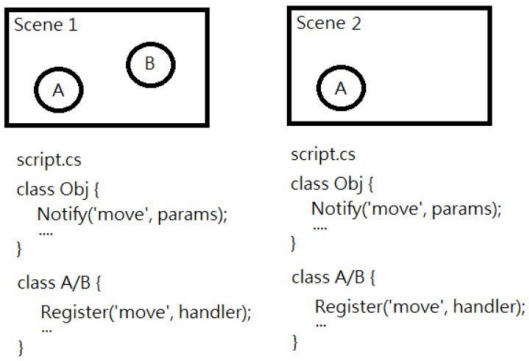


Figure . Apply EDD in the case of figure .

Spectrum Data

Different from usual rhythm games which arrange hit nodes manually. We use Unity API - “GetSpectrumData” to catch the time to switch zones automatically. This API listens to the sounds in the scene and uses Fast Fourier Transform to split the sound wave, and then save the values into an array. By this, we can control the game objects along with the music.

This API is usually used to perform some audio to visual effect, which we can usually see in a music video.

In our program, there are three components in control of the zones switching:

1. Spectrum Handler:
   This listens to the sounds in the scene, and it will catch the specific spectrum we need, and then pass the values to other components.
2. Spectrum Beat:
   This takes use of the spectrum data values we get from the spectrum handler, and is in order to control the zones in the scene. It makes the zones flicker with the rhythm in the play scene to give the player some clues and also performs the visual effect.
3. Spectrum Hit:
   This takes use of the spectrum data values we get from the spectrum handler, and is in order to control the zones in the scene. It switches the zones’ type with the music, including tags and the materials.

NPCs’ Movement

NPCs are those who will grab the player's resources, such as facemasks and needles. If the collision between NPCs and the player occurs, black and red particles will burst out, just like blood and virus. The followings are the mechanism of NPCs' movement:

1. Random scatter NPCs in three levels

a.
```
public GameObject ene_mask;
public GameObject ene_needle;
private int xPos;
private int zPos;
private int objectToGenerate;
private int objectQuantity=0;

[SerializeField]
public int total;
public int xrange1;
public int xrange2;
public int zrange1;
public int zrange2;
```

b. In Enemy.cs :

   i. scatter two different NPCs to grab the player's different supplies

      1. facemask
      2. needle

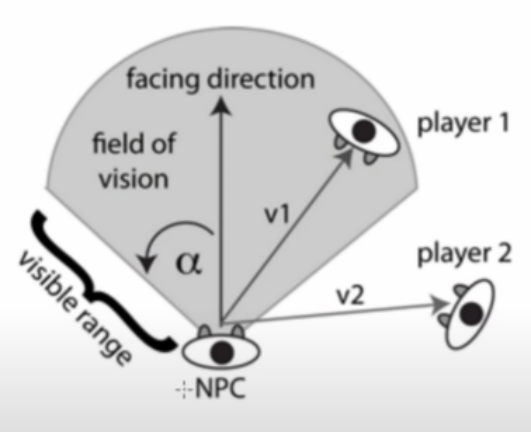
   ii. enable to input variables outside the script

   iii. which can easily set variables in different levels

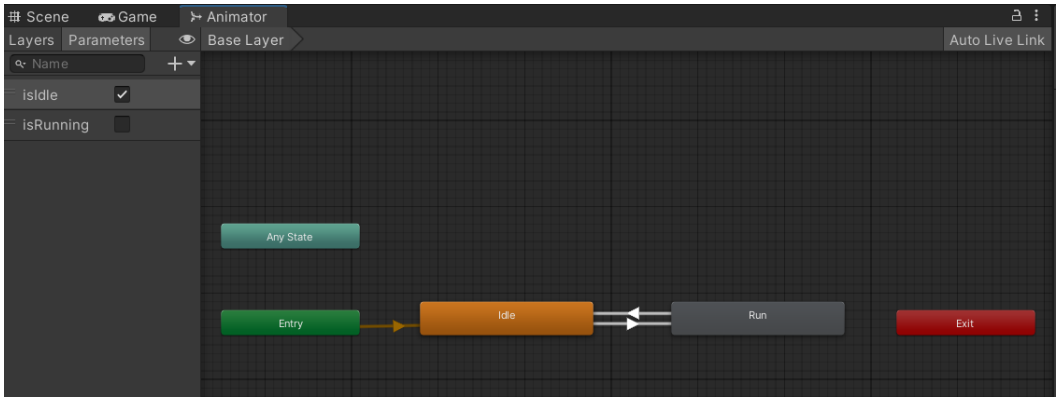
c.
```
// Start is called before the first frame update
void Start()
{
    StartCoroutine(GenerateObjects());
}

IEnumerator GenerateObjects()
{
    while (objectQuantity < total)
    {
        objectToGenerate = Random.Range(1, 3);
        // xPos = Random.Range(-106, -5);
        // zPos = Random.Range(-44,56);
        xPos = Random.Range(xrange1, xrange2);
        zPos = Random.Range(zrange1, zrange2);
        // Instantiate(ene, new Vector3(xPos, 0, zPos), Quaternion.identity);


        if (objectToGenerate == 1)
        {
            Instantiate(ene_mask, new Vector3(xPos, 0, zPos), Quaternion.identity);
        }
        if (objectToGenerate == 2)
        {
            Instantiate(ene_needle, new Vector3(xPos, 0, zPos), Quaternion.identity);
        }

        yield return new WaitForSeconds(0.1f);
        objectQuantity++;
    }
}
```

d. In Enemy.cs :
    i. Use Random.Range()
    ii. set random positions for each NPC in the level
    iii. then randomly choose to scatter two different kinds of NPCs

2. The NPCs will chase the player if the player is too close to the NPC.
    a.
```
// Update is called once per frame
void Update()
{
    Vector3 direction = player.position - this.transform.position;
    float angle = Vector3.Angle(direction, this.transform.forward);
    float dis = Vector3.Distance(player.position, this.transform.position);

    if (dis < 30 && angle < 90)
    {
        direction.y = 0;
        this.transform.rotation = Quaternion.Slerp(this.transform.rotation,
                                  Quaternion.LookRotation(direction), 0.1f);

        anim.SetBool("isIdle", false);
        this.transform.Translate(0, 0, 0.01f);
        anim.SetBool("isRunning", true);
    }
    else
    {
        anim.SetBool("isIdle", true);
        anim.SetBool("isRunning", false);
    }
}
```
    b. compute distance between the player and the NPC
    c. detect the chasing range
    d. if the player is inside the chasing range, the NPC will turn to the player's direction and chase forward
3. Set perspective for NPCs to fit the reality. (You can't see others behind you or out of perspective)
    a.



4. Use the animator to control NPCs' animations between idle and chasing.

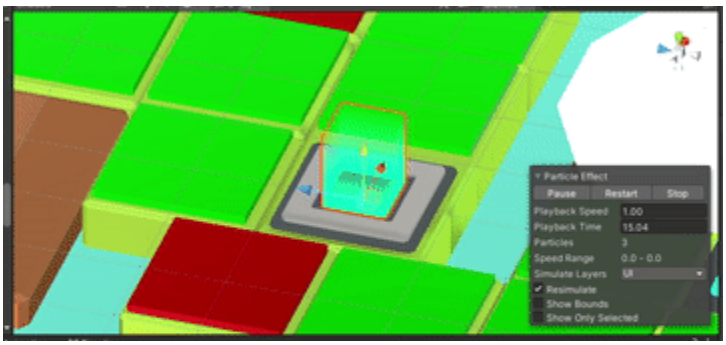


Particle Systems

1. Picking Props
   a. 

2. Collision with NPCs
   a. 

   b. burst particles like blood and virus
   c. so the color was chosen as red and black
3. Terrain decoration

a. 

b. learning different kinds of particle system
c. practice them in our project

Tasks and Contributions of member(s)

Name	%	#Code Lines	Contribution & Artworks
吳岱容	33	356	1. Player control (Keyboard and mouse). 2. Player camera (Following the player and mouse control). 3. Player animation states. 4. Song edition of each level. 5. Spectrum detection and brick control.
李諭樹	33	953	1. Design & implement scenes connection. 2. Design & implement game workflow. 3. Design & implement all the UI components. 4. Design & implement all the UI layout. 5. Construct the game manager. 6. Build-up event store(event manager). 7. Build-up the audio manager. 8. Connect the player, game manager, and UI all together.
呂思函	33	274	1. Set three levels : City, Grass, and Crypt scatter props set bricks of different colors to show safe and dangerous zone 3. NPCs chase the player in a certain range 4. NPC animation state 5. Particle System : Pick props and collision between NPCs 6. Particle System : Grass level 7. Several sound effect 8. Game Intro

Milestones

Figure and show the idea of our milestones. The key point is that **we followed our plan at the beginning, while it was harder and harder to distinguish jobs independently**. We re-scheduled our plan slightly, so one of us was responsible for integrating finished objects into the project, another worked on Game AI, and the other one charged in UI combination. But we still started doing integrated tests in January.

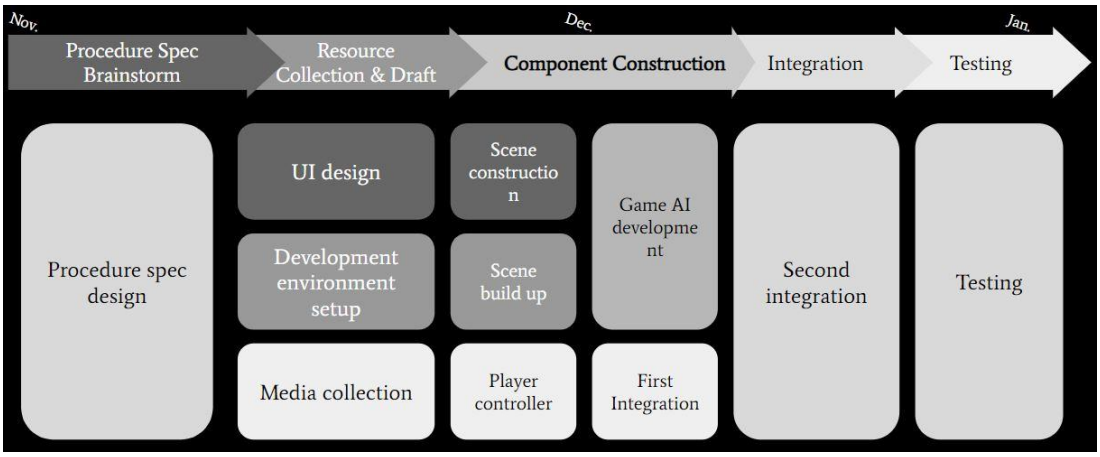


Figure . Expected milestones.

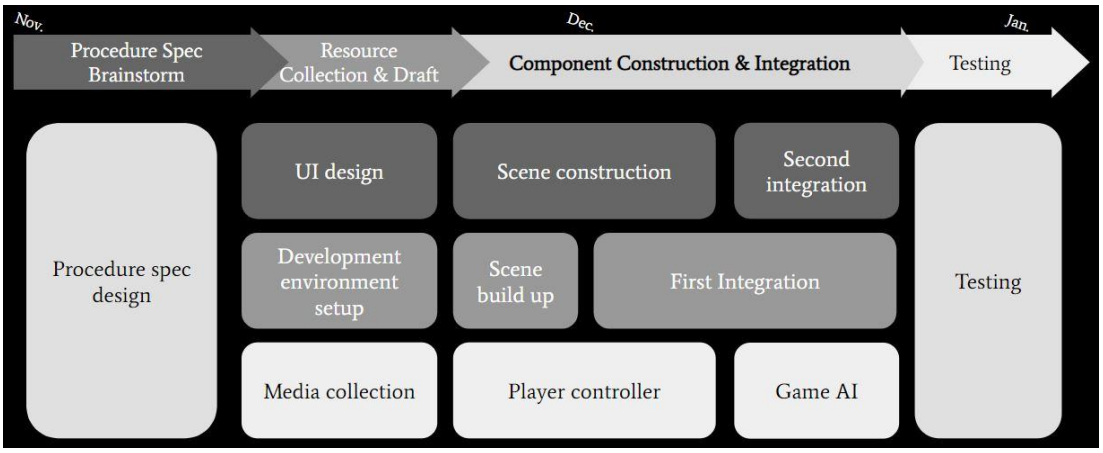


Figure . Actual milestones.

SWOT analysis

Table . SWOT analysis of our project.

Table gives the idea of SWOT analysis of our project. Basically our game has high interaction with players. It could be developed as a first person shooter game or a mult-player online game in the future. However, due to the little time we have, these ideal goals turn out to be our weakness in the end.

Discussion

Event-driven development

Though in the above section we mentioned the advantages of EDD, there are several disadvantages in our case:

1. It is not convenient to debug.
2. Need EventType to control the amount of the events.

It could be possible that there are so many events that decrease the performance. Figure shows that even a tiny object may also occupy the channel for game managers. In this class, we propose to use multiple event managers for different types of components. For example, like figure shows, we use 2 event managers. One for the UI part, and the other for system components. The handshake between 2 managers will be interesting: we could embed another middleware here to filter out unused messages.

Spectrum Data

The reason we choose to use this API is that this function can help us automatically catch the timing we want to switch the zones' types (the zones in our game will change with music). If we

do it in the normal way of building a rhythm game, we will have to map the node manually, and this might take a lot of time to do the synchronization and debugging.

However, this API still has its drawback. This API should be used with the Audio Listener, which means it will listen to all the sounds in the game scene, and therefore, the mechanism may be affected by the other sound effects in the game scene. Also, if the song is very complicated, it might be very difficult to catch the beats.

To overcome the problems above, we propose two methods. One is to store more spectrum data and do a more complicated analysis, but this method we need to understand the FFT, and also there isn't so much documentation describing this API, moreover, the parameter and the usage may be different from each song. The other method is to edict the music clip. By adding specific sounds into the timing we want to switch the zones, to let the Spectrum Handler catch the beats more easily. In the end, we choose the second one. By using this method, we only need to edit the clip, and also we can choose the timing we want without doing the heavy synchronization work.

Component reused

It would be frustrating if we want to apply a change on the same components distributed in the whole project. For example, there are several canvases used in the project, while it is not necessary to load all of them into the same scene. However, once we want to change their behaviors we may update it one-by-one manually. Indeed it would be convenient to control them by a script. So the question here turns into the following: how to write a extenbibly component? To deal with such a stuff, we should not add components based on their usage. **We should think about their behaviors instead**. For example, instead of binding the variable with life bar, timer, volume with Lifebar, Timer, and Volume classes respectively, we could use a variable displayer which handles these cases in the same case. That is, reusing the function-oriented components.

Limitation

1. Currently our game couldn't handle multi-player at the same time.
2. You may use a PC with **keyboards, mouses, and speakers** to play this game to get the ultimate gaming experience.
3. Because of the size of our project, we can't upload it to make it a free online game.
    a. Github, Unity Play, itcho.io, and Simmer.io, etc.

Future Work

1. Conquer the limitations we met during this period of development.
    a. multi-player
    b. online game

2. Several game settings have to be fulfilled, such as
    a. exit
    b. introductions of game mode
    c. more information during game, ex : digit of minus life value

Conclusion

Building a game is not very easy, it includes many knowledge areas. We need programming skills and also basic physics knowledge to implement our game. To make our game look fantasy, we may also need some art skills. Most importantly, we have to learn teamwork, since a game program is such a big program that if we want to build a very fine one, team working is definitely required.

In this final project, we build a 3D game on ourselves, and learn lots of things. Including, using creativity, programming, working with teammates, and presentation skills.

Acknowledgement

This idea is somehow inspired by the Crypt of the NecroDancer (developed by Brace Yourself Games) and Super Mario 3D Land (developed by Nintendo).

Reference List

1. Unity Asset Store
2. Sketchfab
3. CGTrader
4. Youtube