Google Cardboard & Virtual Reality

EEC 687

PROJECT PROGRESS REPORT 3

Jaimal Bhairavia | 2668554
Arpit Patel | 2657673

November 9th, 2016

This progress report is going to be majorly technical in terms of all the functions and components used to achieve the desired working of the game so far. We are still working on modifying the game manifolds. The different sections cover scripts and scripting, steps to expand(create) a scene, use of all properties as per requirements,

Scripting in Unity 3D

Scripting is the most integral part of the Unity. Developers can create any and every kind of function through scripts. Every function, irrespective of it being predefined or developer defined, is always created in terms of scripts coded in C#. In our MazedOut application, we have certain requirements to be fulfilled to achieve the gameplay we have planned. Hence, these requirements are made possible via scripts. As we know, scripting is nothing but coding in C# and linking it with our main program control which is GvrMain. The script can be included into project and linked from inspector panel of GvrMain. Selecting add a component and further selecting add a new script. This will directly take us to MonoDevelop which is the in-house editor of unity. The complete code is written here and debugged here.

Here is code of the **control.cs** script

```
using UnityEngine;
using System.Collections;
public class control : MonoBehaviour{
    private bool walking = false;
    private Vector3 spawnPoint;
    // Use this for initialization
    void Start ()
    {
        spawnPoint = transform.position;
    }
    // Update is called once per frame
    void Update ()
    {
        if (walking)
        {
                transform.position = transform.position + Camera.main.transfo
                rm.forward * .4f * Time.deltaTime;
        }
        if (transform.position.y < -10f)
        {
                transform.position = spawnPoint;
        }
        Ray ray = Camera.main.ViewportPointToRay (new Vector3 (.5f, .5f, 0));
        RaycastHit hit;
        if (Physics.Raycast (ray, out hit))
        {
            if (hit.collider.name.Contains ("plane"))
            {
```

```
                walking = false;
            }
            else
            {
                walking = true;
            }
        }
    }
}
```

The given code is attached to GvrMain prefab so whenever we execute the project, i.e. run the game, it will automatically render the connected script files, here it is control.cs. To begin the code, we have used two different libraries which are predefined in the unity with some specific functions. The library System.collection has a package called Namespace which provides a wide variety of interfaces and classes which are collection of components such as lists, queues, arrays, hash tables and dictionaries.

Further, we created another class named control which is derived from MonoBehaviour base class for all the classes. Next, we created two variables, **walking** for inspecting the walking mode of the first-person character (the camera in our case) and **spawnPoint** is used for setting up the starting point of the game and as a reference point for other functions. transform.position is a keyword provided by unity to monitor the position of a particular object with which it is attached to. So, transform.position is used to detect any kind of movement.

The Update function is called on every frame, meaning, this function will be called several number of times per second. Whenever the walking condition will be true, we make the camera move ahead in the direction of the camera and the speed of the movement is 0.4 meters per frame. This can be increased or decreased to modify the speed of walking. Next condition is to find the position of the character. The code is written in such a manner that if the position is not detected on the plane, the game restarts and the character is placed on the start point i.e. spawnpoint, and will begin the game again. Further, we created a Physics Ray object to find if our character is near any object and if it is whether it is colliding or not. If there is a collision, we make the walking variable false, meaning, stop the character from moving until a collision is no more detected. The walking resumes in any direction the character faces.

Till now we have achieved auto walking functionality including stopping whenever it collides with any kind of object. Furthermore, in previous version of the game, all the walls of the maze were see through if the character came in close enough to collide with them. This minor bug is now fixed. Details explained in later section.

Expanding the Maze:

Citing the previous version of the maze, we felt that the maze dint look like a real. This lead to a huge urge in us to expand the maze in terms of scale and structure. Hence, we tried to increase the size and complexity of the maze with a few changes in its color schemes (still working on the color schemes to make it look better). We started all over again with a fresh project and started building the scene. Plenty of walls were added in random arrangements. These walls were nothing but cubes of different sizes and orientation. Setting them in an exact form with respect to their vertices was a tedious task which involved detailed positioning such that the walls tend to make a close room like environment. Moreover, whenever moving in the maze, one should have enough space to move around. We spaced the walls in such a manner that moving around is possible. Making the walls thinner compared to the last version of the maze helped us to make that happen. We were successful to achieve the desired level of detail and structural kinesis.

The steps for extending the maze (Building all over again):

1. A new project is created and the cardboard SDK is imported. Moreover, a few materials and images are downloaded and imported in unity to use it in the project. Further, a plane is created and its size is transformed to a scale of 2 to make our workspace large enough to comprise the maze. An image which had a maze template is downloaded from the internet and imported in unity. This image is then superimposed on the plane and used as a base to build the maze in an easy manner. Figure no. 1 illustrates the plane with the maze template.

Figure no. 1

2. We start creating walls by inserting cubes in different size and orientation. The walls in the image below are cubes stretched out in terms of axes and a wood material is applied on them. These walls have the same property and hence can be duplicated and arranged.

Figure no. 2

3. The next image displays the complete maze built after all arrangements are made. This maze is ready to be used to implement all the cardboard functions like the reticle and the event trigger. The left image is the side view and the right image had the top view of the maze created.

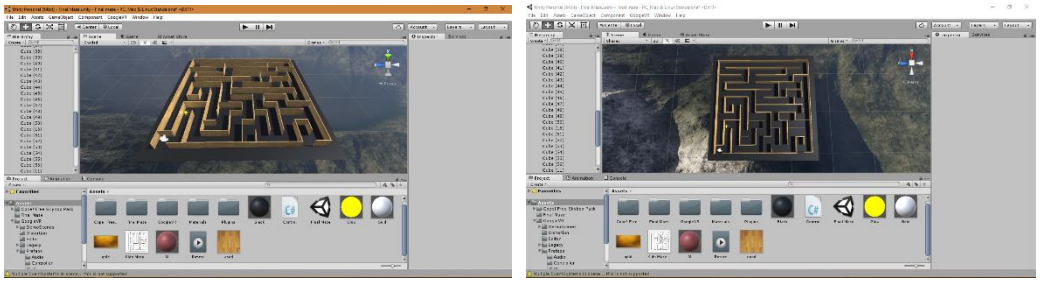
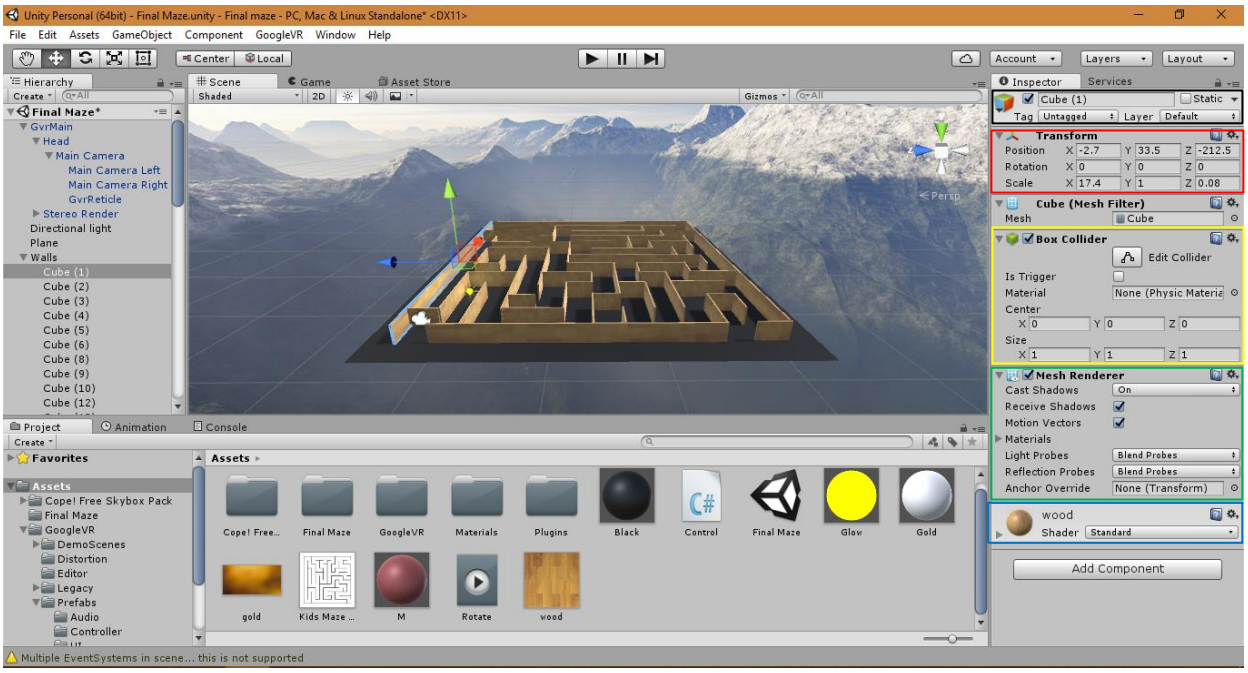


Figure no. 3

Properties of Game objects:

Let's talk about a few properties of the objects in the game. We have been speaking about the objects all through since the beginning. All the objects have a basic mesh filter and a mesh renderer which are required to give it a 3D view. Shadows, motion vectors, light and reflection (Green box) and the material and shade (Blue box) are customizable per our requirements. These are the default the properties. One important component of an object is the collider.

For a cube, it's a box collider (yellow box), for a sphere it's a sphere collider and so on. The property of this component is that any kind of event triggered is because of interaction with this collider. For example, if a pointer enters a cube, it enters the box collider and hence an event is detected. The collider opens some humungous possibilities to play around with what can be done with the object once an event is occurred. All the properties are highlighted in the different boxes in the image below.



Problems Solved:

The following problems that persisted in the previous version of MazedOut have been solved as of now.

1) Auto walk feature of stopping whenever the user looks at the ground was not working. We found out that it was an issue of the physics raycaster as our plane wasn't opaque. So, once we fixed that problem, this one fixed itself without any extra effort.
2) Another issue was the one we had with the walls such that when the character went close to a wall, it became see through disrupting and confusing the complete view. This problem was solved by changing the property of the camera. Clipping plane is the name of the property, and it means that we can set the minimum and maximum focal length of the camera. As we reduced the near i.e. the minimum focal length to 0.01, the angular drag reduced from a preset of 1 to 0.01. This means the camera can see as close as 0.01 meter of the object.

Future Work:

- We have figured out the click event trigger codes and its integration with the event system. When the user clicks on any GameObject, we can define any function in the script to be performed on that event. With respect to our game, this would be like counting the coins when they are picked up and then destroying them once they are collected.
- As we have created a new maze which is larger in size, we have two different mazes and so we plan to improvise the game-play by setting the game in two different stages. The smaller maze would be form the first stage and when the player completes the maze under a preset time, he or she can proceed to the next stage if he or she wishes to. This will be achieved by creating a button which will be required to be clicked.
- At the end of the first level we will create invisible object which when collides with the character will end that stage and a scoreboard would appear with two options of replaying the same stage or proceeding to the next. The next level button will only be displayed if a certain criterion is fulfilled (for now let's assume it a pre-decided time). Next level of the maze which is four times bigger in size and will also have coins to collect. So, for this level the score will be calculated based on two parameters being the number of coins and the time taken to complete the game.
- The game can have numerous expansion ideas. We would list out the one's we have thought about in the final report. Unity and Virtual reality get more and more interesting as we go deeper and deeper into them.