Game Development in Unity Using Oculus Quest VR

Christopher Todd
California State University-Stanislaus, Turlock, CA
ctodd1@csustan.edu

Abstract — **The Oculus, which is owned by Facebook, is quickly becoming the most popular medium for an interactable 3D gaming experience by way of Virtual Reality. Unity Real-Time Development Platform is one of few options that game developers have to create VR games for the Oculus which makes it a desirable platform to learn. This paper will address all the components that went into the creation of a game that runs on the Oculus Quest using Unity. It will present some of the methods necessary to allow the game to implement procedurally generated stages and a racing vehicle to be the object the player takes control of to accomplish the task of completing each stage.**

I. Motivation

Game development is an area of computer science that is most often seen as a niche. However, with the ever-increasing popularity of virtual reality in most recent years, game development is, by far, one of the best fields in computer science to learn. Not only is VR becoming more and more popular with the gaming industry, it is also become an aid to help with training up medical workers, military personnel, and even teenagers helping them become more confident behind the wheel of a vehicle [1] [2] [3]. With these industries using VR, it is the motivating factor for learning how to create a video game which is just beginning stages of utilizing a much more complex piece of technology for a greater purpose.

II. Related Work

Virtual Reality has made it possible for games to be more interactive amongst the player base. The following aspects of these released games will be implemented in the game discussed in this paper.

A. *VR being used in iRacing*

iRacing is a racing simulator that has expanded their gaming platform to support Virtual Reality which immersive their player base into one of the most realistic vehicle racing games currently on the market [4].

B. *Procedural generation found in Borderlands*

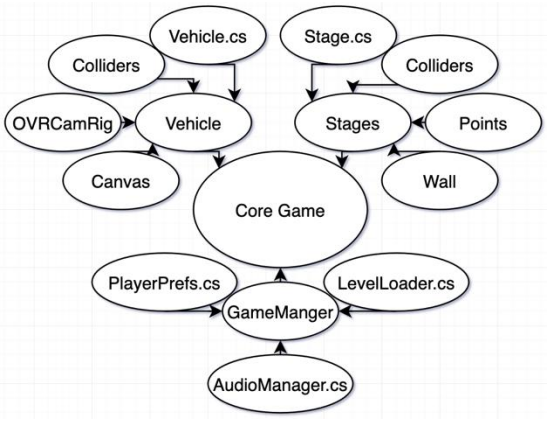
As for procedural generation to create an infinite level system, which is usually created for higher replay value amongst the player base, procedural generation is used in games such as Borderlands which spawns procedurally generated weapons when looting boxes, killing enemies, and completing quests [5].

III. Method

The project desired is a racing-like game that will procedurally generate stages with the goal of completing as many stages as possible before the time runs out. A vehicle is instantiated at the beginning of each attempted game run with the player sitting in the seat of the vehicle with their vision immersed in the Oculus Quest's virtual reality. The Oculus controllers will allow the player to drive the vehicle through each consecutive stage until the game has ended.

Each stage generated will have a randomly generated point value that must be achieved by collecting point pillars that have been given randomly generated set values. Each point pillar has walls surrounding it to create an obstacle for the car to maneuver around. Each stage is guaranteed to have enough pillars to collect to achieve the point value for that stage. When the player has collected enough points that is equal or greater than the stage's total point value to be earned, the wall that is leading to the next stage will be lifted up giving the player a new stage to drive to and complete. When the vehicle enters each new stage, the current collected points will be set back to '0' and the last stage is destroyed.

An LCD screen will be placed on the middle console for the player to have real-time feedback of important values needed to achieve the games purpose. These values include the timer which counts down by seconds, the current points collected in the given stage, and the speed of the vehicle. The current speed of the vehicle is important when collecting the pillars because it will be added as a bonus to the current points collected which gives the player an ability to get to the next stage quicker, possibly collecting few pillars. The game ends when the timer reads '0' or the player has fallen over the edge of the stage.



As shown in the figure, the Core Game has three main components that make the game mechanics work: Vehicle, Stages, and Game Manager. Each main component has subcomponents that lend to its functionality as well.

A. Vehicle

The vehicle has many methods to make its functionality work in sync with the game's mechanics. Taking a closer look under the hood, the vehicle is equipped with a canvas, an OVRCameraRig, Colliders, and as assortment of vehicle scripts.

1. Canvas

The canvas is normally used for user interface. The canvas object is attached as a child of the vehicle in order for it to stay static to the LCD screen that is placed on the middle console. The canvas has a scoreUI object, speedUI object, and timeUI object.

When the player collects points, the scoreUI object has a script that increments the value of a score instance variable and then runs a function to display the score to the screen of the vehicle.

```
public void AddPoints(int points){
    score += points;
    scoreUI.text = score.ToString();
```

The speedUI object has a script that gets an update every frame of the current speed of the vehicle and translates the speed's value to the LCD screen.

```
void Update(){
    int currentSpeed = (int)carController.CurrentSpeed;
    speedUI.text = currentSpeed.ToString();}
```

The timeUI object has a script that manages the timer countdown as well as a method that allows for increasing the timer by X number of seconds based on the point value collected.

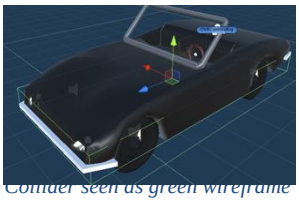
```
private void Update(){
    if(timer <= 0f){EndGame();return;}
    timer -= Time.deltaTime;
    timeUI.text = ((int)timer).ToString();}
```

2. OVRCameraRig

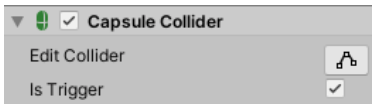
In Unity, a game object can be saved and stored in the assets folder as a prefab. A Prefab usually consists of multiple game objects as children to the main game object/prefab. Prefabs help speed up the gaming development process because once a game object is created for a specific purpose, the prefab can be replicated or instantiated elsewhere in the game. All this to say that the OVRCameraRig is a prefab a part of the Oculus library that was imported as an asset package through the package manager. This prefab comes with all of the components and scripts necessary for the Oculus Quest to be used as the virtual reality camera in this game. This camera is positioned at a normal head level in the driver's seat of the vehicle.

3. Colliders

Colliders in Unity are used for either collisions or triggers. The collider is attached as a component of a given game object with scale and offset in relation to the object's transforms. It is good to give the collider a shape that is in close relation to the shape of the object it is being attached to. Collisions will use the physics engine to show the displacement of two game objects that have colliders attached to them when they meet at an exact location. The displacement is based on object A and object B's current velocity, mass, direction, etc. when they collide.



Collider seen as green wireframe box around vehicle



However, to change a collider into a trigger, it is as simple as checking a box in the inspector located on the collider's component. When the trigger is true, it no longer interacts with other colliders by displacing the game object, but instead, it triggers an event within a script attached to the game object giving action to the collider that passed through another collider. For example, if I wanted to have the vehicle collect points as it passed through a game object that is a visual representation of a coin, it wouldn't make sense to have the collider on the coin to be a non-trigger. The reason for this is that the vehicle would bounce off of the coin instead of passing through the coin.

Changing a collider's trigger value to true allows a script to use the OnTriggerEnter method which will then be programmed to determine what happens at the point the trigger collider meets either another trigger collider or a normal collider. In this case, it makes sense to collect a coin by passing a method which increases the score by X amount of points and then destroys the game object that was attached to the collider as the vehicle drives through the coin. There is also a similar method that can be called if the collision's trigger checkbox was false which is the OnCollisionEnter method. Note that one is a Collision method and the other is a Trigger method.

```
private void OnTriggerEnter(Collider other) {
    if (other.gameObject.tag == "vehicle") {
        scoreUI.AddPoints(pointsToCollect);
        timeUI.AddTime(timeToAdd);
        Destroy(gameObject); } }
```

4. Vehicle.cs

Vehicle.cs is not a script itself, rather a representation of twelve (12) scripts that are attached to various components of the vehicle such as the colliders, wheel effects, LCD screen which controls the user interface, car audio, car controller, steering wheel rotation, steering wheel colliders, break lights, suspension, and skid trails. The vehicle's functionality is something that was not programmed from scratch. It was stripped down as parts from a standard asset vehicle found on the Unity Asset Store. The vehicle was sculpted by a 3D object model artist named Preston Linderman.

The vehicle was made into a prefab in order to be instantiated at the beginning of every round. The scripts attached to the vehicle work in sync with one another by way of methods and instance variables being passed around throughout the scripts to give the vehicle its functionality.

B. Stages

Stages is where the game's procedural generation happens. A single stage is prefabbed with multiple game object children where it is instantiated at the beginning of each game attempt or when the vehicle transfers over from one stage area to the next.

The methods that create the random environment for the player comes together when each subcomponent works in tandem with each other to create each unique stage layout. The key components for the Stages consist of a Wall, Points, Colliders, and Stages.cs.

1. Wall

A wall is attached at the north end of each instantiated stage. It displays a point value that is randomly generated using an RNG (Random Number Generator). The player must collect this point value that is on the wall in order to advance to the next stage. The wall is lifted up six (6) meters when the score on the LCD screen is greater than or equal to the number on the wall. As you can see in the screen capture above, the wall says 800 and the score on the LCD in tiny font says 892. This makes the condition true in the script, therefore, raising the wall up to allow the vehicle to pass through to the next stage.

```
public void TransformWall(){
    if (scoreUI.score >= scoreNeededToAdvance
        && !isWallMoved) {
        transform.position += raiseWallUp;
        isWallMoved = !isWallMoved; }}
```

2. Points

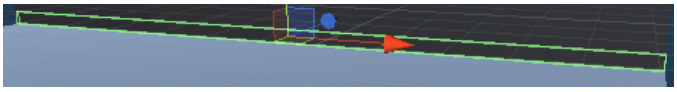
```
for(int i = 0; i < wall.GetComponent<MoveWall>()
    .numberOfObstacles; i++){
    int x = Random.Range(-5, 5) * 10;
    int z = Random.Range(-3, 5) * 10;
    GameObject obstacle = Instantiate
        (obstacles[Random.Range(0, obstacles.Length)],
        new Vector3((float)x, 0f,
        transform.parent.transform.position.z + (float)z),
        Quaternion.identity);
    obstacle.transform.parent = transform;}
```

Points are collected and added to the score for every point pillar driven through. Point pillars are a child of a point-obstacle-track that is a prefab. When a new stage is instantiated, a random number generator is used a few times to calculate the number of points needed to complete the stage in order to move on to the next stage and the position in which the point obstacle track prefabs will be instantiated throughout the individual stage.

```
else{
    numberOfObstacles = Random.Range(5, 11);
    scoreNeededToAdvance = numberOfObstacles * 100;}
myWallScore.SetWallScoreText(scoreNeededToAdvance.ToString());
```

A numberOfObstacles variable is created on each wall script and a random number is generated from 5 to and including the possibility of 10. This numberOfObstacles value is then used to calculate the score needed to progress to the next stage by multiplying it by one-hundred (100) and attaching the result to a scoreNeededToAdvance variable which is used to be the value displayed on the wall and used as the conditional value needed to be achieved to raise the wall to progress to the next stage.

3. Colliders



```
private void OnTriggerExit(Collider other){
    if (other.gameObject.tag == "vehicle"){
        FindObjectOfType<RaceAreas>().ProgressToNextStage();} }
```

Each stage prefab has a trigger collider that is positioned at the location of the wall. The event that is triggered when the vehicle passes through the trigger collider is an OnTriggerExit method which will only trigger the event when the collider exits the parameter of the collider.

```
public void ProgressToNextStage() {
    Destroy(GameObject.Find("RaceArea" + (stage-1).ToString()));
    GameObject g = Instantiate(raceArea, new Vector3(0f, 0f, stage++ * 100f),
        Quaternion.identity);
    g.transform.parent = FindObjectOfType<RaceAreas>().gameObject.transform;
    g.name = "RaceArea" + stage.ToString();
    FindObjectOfType<VRVehicles.Car.Canvas.ScoreUI>().score = 0;
    FindObjectOfType<VRVehicles.Car.Canvas.TimeUI>().timer = 10f; }
```

4. Stages.cs

Stages.cs is not a script itself, rather a representation of six (6) scripts that are attached to various components of the Stage prefab such as the collider which ends the game if the car falls off the edge, to each north wall to move it upwards when the player has achieved required score to advance, to the Stages prefab which has important methods for stage creation, to the wall canvas which displays the score needed to advance to the next stage, and to a child object named Tracks which helps delegate where each point collector goes based on an RNG.

C. Game Manager

The Game Manager may look like it doesn't hold much value, but it does have an important purpose. It is a game object that doesn't get destroyed on new scene loaded. This means that it can hold scripts that needs to be passed from scene to scene that have instantiated an object that is holding data that must be maintained throughout the gameplay. It also carries many of the important scripts that are not meant to be attached to temporary game objects because temporary game objects usually get destroyed.

1. PlayerPrefs.cs

```
public class PlayerPrefsController : MonoBehaviour{
    const string BEST_STAGE_KEY = "best stage";

    public static float GetBestStage(){
        return PlayerPrefs.GetFloat(BEST_STAGE_KEY);}

    public static void SetBestStage(float stage){
        if(stage > GetBestStage()){
            PlayerPrefs.SetFloat(BEST_STAGE_KEY, (stage - 1));}}

    public void ResetBestStage(){
        PlayerPrefs.SetFloat(BEST_STAGE_KEY, 0f);
        if (FindObjectOfType<CurrentBestStage>()) {
            FindObjectOfType<CurrentBestStage>().UpdateCurrentBestStage();}}}
```

PlayerPrefs.cs is a script that will record the greatest stage completed into a user file on the Oculus Quest. This greatest stage completed is displayed on the Title Screen and End Game Screen.

2. AudioManager.cs

```
public class AudioManager : MonoBehaviour {
    public Sound[] sounds;
    void Awake(){
        foreach(Sound sound in sounds) {
            sound.audioSource = gameObject.AddComponent<AudioSource>();
            sound.audioSource.clip = sound.clip;
            sound.audioSource.volume = sound.volume;
            sound.audioSource.pitch = sound.pitch;
            sound.audioSource.loop = sound.loop;  } }
    public void PlaySound(string name){
        Sound s = Array.Find(sounds, sound => sound.name == name);
        if(s == null) {
            Debug.LogWarning("Sounds: " + name + " not found!");
            return; }
        s.audioSource.Play(); }}
```

The AudioManager.cs takes care of the sounds in the game. It takes in a sound array of all sound clips and attaches an AudioSource to each sound clip. This also has a method that plays any sound when the string name of the clip is passed into the parameters.

3. LevelLoader.cs

```
public void LoadGame()
{
    SceneManager.LoadScene("Core Game");
}
```

The LevelLoader.cs contains the methods in order to load scenes such as Title Screen scene, Core Game scene, End Game scene, etc. These methods are called on menu buttons and events to trigger the new scene.

IV. RESULT

There are many benefits that the Unity Game Engine has in developing a 3D interactable game with the Oculus Quest. One being that Unity makes it very easy to create a game for Virtual Reality given the many pre-built packages that take care of the synchronization with the Oculus Quest interface. The results of the previous section are shown in the video below.

To start, right click and click Play

V. CONCLUSION

To conclude, game development using Unity for the Oculus Quest is fun and exciting. The learning curve is sometimes a little steep, but if one is motivated, learning to be proficient in some of the core concepts of game development is possible.

REFERENCES

[1] M. Sattar, S. Palaniappan, A. Lokman, A. Hassan, N. Shah, Z. Riaz, "Effects of Virtual Reality training on medical students' learning motivation and competency.," *Pakistan Journal of Medical Sciences*, vol. 35, no. 3, pp. 852-857, 2018.

[2] M. Velichko, "VR Military Training – the Next Step of Combat Evolution," Jasoren, 2019. [Online]. Available: https://jasoren.com/vr-military-training-the-next-step-of-combat-evolution/.

[3] W. Vlakveld, M. R. E. Romoser, H. Mehranian, F. Diete, and D. L. Fisher, "Does the experience of crashes and near crashes in a simulator-based training program enhance novice driver's visual search for latent hazards?," *Transportation Research Record*, no. 2265, pp. 153-160, 2011.

[4] "Simracing takes on a whole new look with a Virtual Reality headset!," iRacing, 2020. [Online]. Available: https://www.iracing.com/virtual_reality/.

[5] "Weapons," Fandom, [Online]. Available: https://borderlands.fandom.com/wiki/Weapons. [Accessed 2020].