Crossy Road VR Game Documentation

University of Illinois at Champaign-Urbana
Yuxi Gu
May 2017

Table of Contents

Introduction ………………………………………………………………………….. 3

Implementation of Crossy Road VR……………………………………………………… 3

User-level Instructions For Playing the Game …………………………………….…. 8

List of Illustrations

Figures

Figure 1. Game Scene on The Unity 5.2.2 Platform ……………………………………… 3

Figure 2. GeneratLaneBox ………………………………………………………….. 5

Figure 3. GeneratRiverBox ……………………………………………………… 5

Figure 4. Input Values of Button X, Y, A and B…………………………………… 6

Figure 5. Start Menu ………………………………………………………………... 8

Figure 6. Interface Of Change Characters …………………………………………….. 8

Figure 7. Model of An Xbox Controller ……………………………………………..... 9

Figure 8. Game Scene2 on The Unity 5.2.2 Platform ………………………………... 10

Figure 9. Game Over Interface……………………………………………………… 11

Tables

Table 1. Function Descriptions1……………………………………………………… 4

Table 2. Function Descriptions2……………………………………………………… 5

Table 3. Mappings Between Buttons and Their Movements …………………………....... 9

Table 4. GameObjects And Their Corresponding Functionalities ……………………….... 11

Introduction

Crossy Road VR is a virtual reality game that was implemented on the Unity 5.2.2 platform. Players can play this game use the Oculus Rift, which is a virtual reality headset, to visually view the scene of the game and use an Xbox controller to control the movement of the avatar after hitting "Play" on the Unity 5.2.2 platform. The main idea of this game is to move as far as possible without dying. The avatar in this game will come across various obstacles while moving forward, such as moving cars, and each of the obstacles will cause the immediate death of the avatar. In order to get a higher score, players needs to cross as many roads as possible. This documentation is composed of two separate parts, which are the game implementation and the user-level instructions for playing the game. In the implementation part, the configuration of the scenes, as well as the details of setting up an Xbox controller and the basic operations of building the user interfaces, will be illustrated in three separate sections respectively. And in the user-level instructions part, information of how to access and go through the game tutorial, basic operations of changing the character, the way that the scoring system works, the corresponding functionalities of specific GameObjects and information of how to use an Xbox controller to control the avatar and how to restart or exit the game will be explained in seven separate sections.

Implementation of Crossy Road VR

1. The Configuration of The Scenes

The scene of the game was auto-generated by C# script. An example of the scene was shown in Figure1. Game Scene on The Unity 5.2.2 Platform.

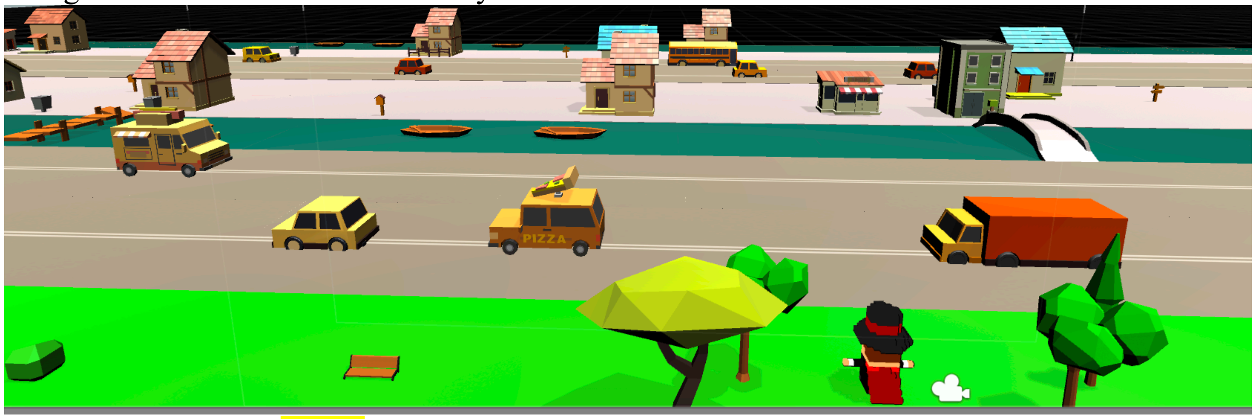


Figure1. Game Scene on The Unity 5.2.2 Platform

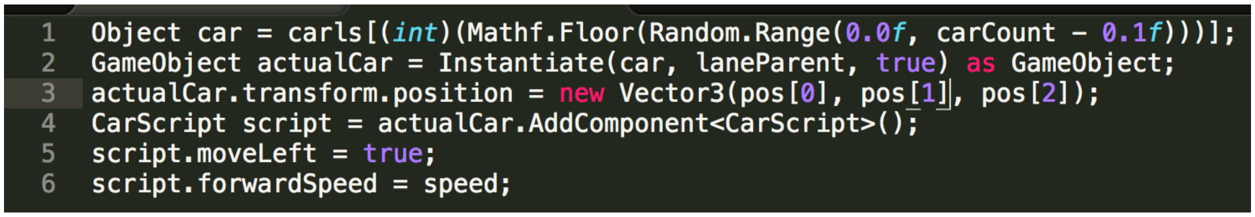
The main idea of the building the whole scene was to use a random generator to select the next type of sub-scene (Lane, Wood, Neighborhood or River) to be generated, where each sub-scene was generated by a separate function written in C#. There were two types of GameObjects in these sub-scenes, one was stationary GameObjects and another was moving GameObjects. An example of generating a moving GameObject and an example of generating a stationary GameObject were described in the following contents to illustrate the implementation of the scenes in detail.

1. Moving GameObjects

Only cars and boats were moving GameObjects in this game. The generation of a car was used as an example here. A code snippet of a function called **GenerateLaneBox(Vector3 pos, Transform laneParent, float spd)** that generated a moving car was shown in Figure 2. GeneratLaneBox. And Table 1. Function Descriptions1 explained the input arguments and described the specific functionalities of GenerateLaneBox and each of the function used in GenerateLaneBox in detail.

Function Name	Input Arguments Explanation	Description
GenerateLaneBox (Vector3 pos, Transform laneParent, float spd)	pos: The initial desired position of the car, represented in a form of Vector3(x, y, z). • pos[0]: The desired x position. • pos[1]: The desired y position. • pos[2]: The desired z position.	Returns a moving car that is a children of **laneParent**, and the car is moving at a constant speed that equals **spd,** starting from position **pos. (Word that** marked bold indicates it's one of the input arguments.)
	laneParent: The parent of the car.	
	spd: The desired speed of the car.	
Random.Range(float min, float max)	min: The lower bound of the random number to be generated.	Returns a random float number between and **min** (inclusive) and **max** (inclusive).
	max: The upper bound of the random number to be generated.	
Mathf.Floor(float f)	f: The float number passed in this function.	Returns the largest integer smaller to or equal to f.
Instantiate(Object original, Transform parent, bool instantiateInWorldSpace);	original: An existing prefab that you want to use to create the new GameObject.	Returns a Gameobject that its 3D model is **orginial,** its parent is **parent**, and set it's position relative to world space or parent space based on the value of **instantiateInWorldSpace.**
	parent: The parent of the GameObject to be created.	
	instantiateInWorldSpace: This variable is true when the position of the GameObject to be created is relative to world space. If the position is relative to the parent space, this variable is set to false.	
myObject.AddComponent<ScriptName>	myObject: The targeted GameObject to be attached.	Attach a script (in Javascript or C#) called **ScriptName** to **myObject**.
	ScriptName: The file name of the script that you want to attach to myObject.	

Table 1. Function Descriptions1



```
1  Object car = carls[(int)(Mathf.Floor(Random.Range(0.0f, carCount - 0.1f)))];
2  GameObject actualCar = Instantiate(car, laneParent, true) as GameObject;
3  actualCar.transform.position = new Vector3(pos[0], pos[1], pos[2]);
4  CarScript script = actualCar.AddComponent<CarScript>();
5  script.moveLeft = true;
6  script.forwardSpeed = speed;
```

Figure 2. GeneratLaneBox

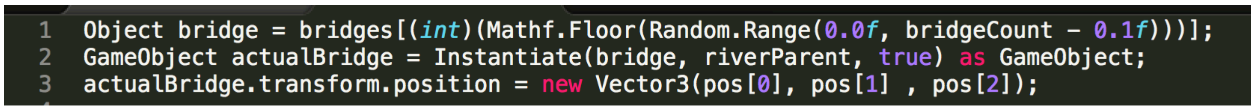
What Line 1 did is to randomly select a car prefab (a prefab is basically a 3D model used to visually represent an object of any type) from an array called carls, which contained a list of prefab of taxi, van, car, school bus and pizza car. The random selection was accomplished by Mathf,Floor(Random.Range(0.0f, carCount - 0.1f)). This closure generated a random integer number within the range of [0, carCount - 1]. And the number generated indicated the car prefab selected, where 0 represent the first car prefab in the array called carls and carCount - 1 represented the last car prefab. Line 2 created a GameObject called actualCar using the prefab selected in Line 1 and assigned actualCar to be the children of laneParent, which referred to the sub-scene of type "lane". Line 3 set the actualCar's initial x, y and z position to be the desired position. Line 4-6 Attached a script called "CarScript" to actualCar, then set its direction (could be either left or right) and speed to make it running on the road.

2. Stationary GameObjects

All GameObjects except cars and boats were stationary GameObjects in this game. The generation of a bridge was used as an example here. A code snippet for the function called **GenerateRiverBox(Vector3 pos, Transform riverParent)** was shown in Figure 3.GenerateRiverBox. And Table 2. Function Descriptions2 explained the input arguments and described the specific functionalities of GenerateRiverBox. For functions used in both of GenerateRiverBox and GenerateLaneBox, such as Random.Range, please use Table 1. Function Descriptions1 for reference.

Function Name	Input Arguments Explanation	Description
GenerateRiverBox(Vector3 pos, Transform riverParent)	pos: The desired position of the bridge, represented in a Vector3(x, y, z). • pos[0]: The desired x position. • pos[1]: The desired y position. • pos[2]: The desired z position.	Returns a bridge that is a children of **riverParent** and is placed in position **pos.** **(Word that** marked bold indicates it's one of the input arguments.)
	riverParent: The parent of the bridge.	

Table 2. Function Descriptions2



```
1  Object bridge = bridges[(int)(Mathf.Floor(Random.Range(0.0f, bridgeCount - 0.1f)))];
2  GameObject actualBridge = Instantiate(bridge, riverParent, true) as GameObject;
3  actualBridge.transform.position = new Vector3(pos[0], pos[1] , pos[2]);
```

Figure 3.GenerateRiverBox

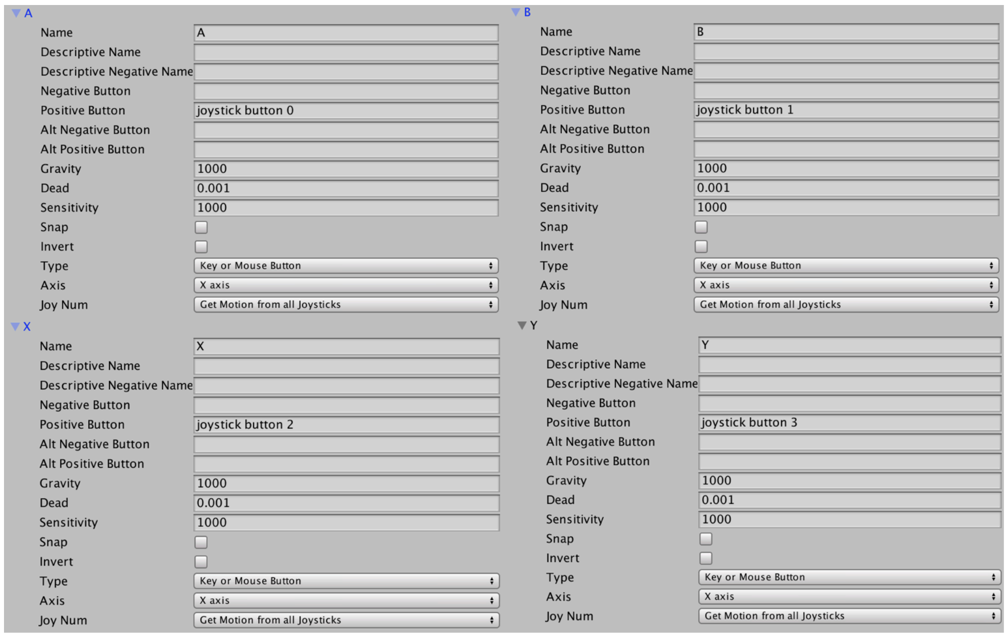
Line 1 randomly selected a bridge prefab from an array called bridges, which contains two different type of bridges. The random selection was accomplished by Mathf,Floor(Random.Range(0.0f, bridgeCount - 0.1f)). This closure generated a random integer number within the range of [0, bridgeCount - 1]. And the number generated indicated the bridge prefab selected, where 0 represent the first bridge prefab in the array called carls and carCount - 1 represented the last one. Line 2 created a GameObject called actualBridge using the prefab selected in Line 1 and assigned actualBridge to be the children of riverParent, which referred to the sub-scene of type "River". Line 3 set the actualBridge's x, y and z position to be the desired position.

And the compositions of the sub-scenes were described in the following details:

- *Lane*
  - Each lane was composed of a rectangular area of road and a lot of randomly generated cars that were running in left or right direction.
- *Wood*
  - Each wood was composed by a rectangular area of meadow and different types of stationary decorative GameObjects (trees, benches, pavilion, etc).
- *Neighborhood*
  - Each neighborhood was composed of a rectangular area of terrain, different types of stationary houses and various decorations.
- *River*
  - Each river was composed of a rectangular area of a river, two kinds of bridges and moving boats.

2. Set Up Connections Between The Xbox Controller And The Game

The inputs that were related to the Xbox controller's four buttons (X, Y, A, B) in "Project Settings" was set up first. The input values for setting up button X, Y, A and B were showed in Figure 4. Input Values of Button X, Y, A and B. Then Input.GetButtonDown("X"), Input.GetButtonDown("Y"), Input.GetButtonDown("A") and Input.GetButtonDown("B") were called in the C# script that controlled the movements of the avatar to get actual input values from these four buttons in the Xbox Controller.



▼ A

Name	A
Descriptive Name	
Descriptive Negative Name	
Negative Button	
Positive Button	joystick button 0
Alt Negative Button	
Alt Positive Button	
Gravity	1000
Dead	0.001
Sensitivity	1000
Snap	☐
Invert	☐
Type	Key or Mouse Button
Axis	X axis
Joy Num	Get Motion from all Joysticks

▼ B

Name	B
Descriptive Name	
Descriptive Negative Name	
Negative Button	
Positive Button	joystick button 1
Alt Negative Button	
Alt Positive Button	
Gravity	1000
Dead	0.001
Sensitivity	1000
Snap	☐
Invert	☐
Type	Key or Mouse Button
Axis	X axis
Joy Num	Get Motion from all Joysticks

▼ X

Name	X
Descriptive Name	
Descriptive Negative Name	
Negative Button	
Positive Button	joystick button 2
Alt Negative Button	
Alt Positive Button	
Gravity	1000
Dead	0.001
Sensitivity	1000
Snap	☐
Invert	☐
Type	Key or Mouse Button
Axis	X axis
Joy Num	Get Motion from all Joysticks

▼ Y

Name	Y
Descriptive Name	
Descriptive Negative Name	
Negative Button	
Positive Button	joystick button 3
Alt Negative Button	
Alt Positive Button	
Gravity	1000
Dead	0.001
Sensitivity	1000
Snap	☐
Invert	☐
Type	Key or Mouse Button
Axis	X axis
Joy Num	Get Motion from all Joysticks

Figure 4. Input Values of Button X, Y, A and B

3. Build User Interface

We built each interface with a canvas, which had a panel as its children. Add we added different components in the panel of each canvas, including buttons and texts, based on the functionality of each user interface. Examples of user interfaces in this game were showed in Figure 5. Start Menu and Figure 6. Interface Of Change Characters. Also, the detailed implementation for switching one interface to another interface (or the scene of the game) was to set up the visibilities of interfaces. Examples of switching the start menu to other interfaces were used here. Initially, all interfaces except the start menu was set to be invisible and every button in this game was attached with a listener. When player pressed "Tutorial" or "Change Characters" in the start menu, the listener attached to this button was triggered. Then the start menu was set to be invisible and the user interface for tutorials or change characters was set to be visible. However, if player pressed "Start", then all interfaces was set to be invisible. Overall, building user interfaces was much simpler than implementing the auto-generated rendering part since all components (e.g. buttons, canvas, panels) needed for user interfaces were already provided in the Unity 5.2.2 platform.

User-level Instructions For Playing the Game

1. **How to Access And Go Through the Game Tutorial**
    - In the start menu (Figure 5. Start Menu), there is a button called "Tutorial". Press button "X" in the keyboard will allow players to view the page for tutorials. And pressing the button called "Back To Home" in the tutorial page will navigate players back to the main menu.

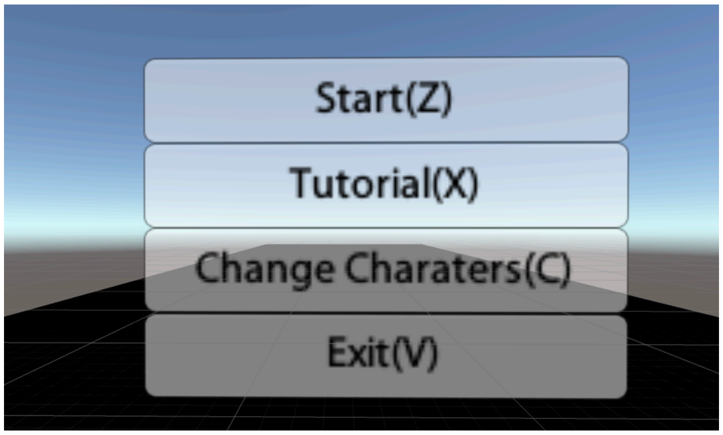


Figure 5. Start Menu

2. **How to Change Characters**
    - In the start menu (Figure 5. Start Menu), there is a button called "Change Characters". Players can press button "C" in the keyboard to enter the user interface for changing a character (Figure 6. Interface Of Change Characters). And players can also press button "U" and "O" in the keyboard to view the previous character and the next character and press button "I" to select the current character as the avatar. If a certain character is selected, the game will start automatically.

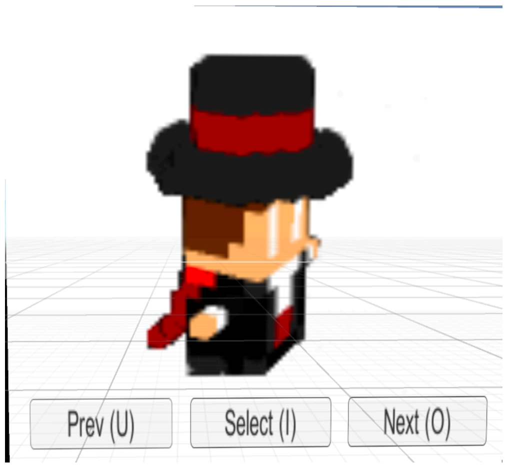


Figure 6. Interface Of Change Characters

3. **Use the Xbox Controller**

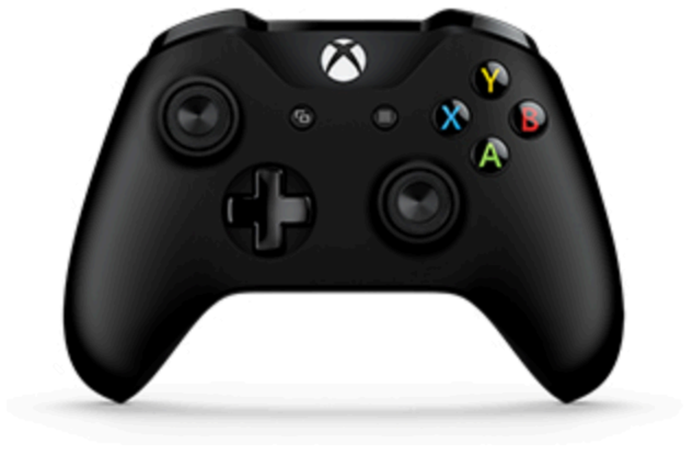


Figure 7. Model of An Xbox Controller

- In this game, players should only use four buttons(X, Y, A, B) in the Xbox Controller (Figure 7. Model of An Xbox Controller) to control the movement of the avatar. The detailed information about the mappings between buttons and their movements is in Table 3. Mappings Between Buttons and Their Movements.

Button	Movement
Y	Move Forward
A	Jump
X	Turn Left
B	Turn Right

Table 3. Mappings Between Buttons and Their Movements

4. **Scoring System**
   - The score of the game is determined by the number of roads the player crossed. The final score of the player = 10 * the number of roads the avatar crossed.

5. **Information of Specific GameObjects**
   - The following graph (Figure 8. Game Scene2 on The Unity 5.2.2 Platform) is a screenshot on the Unity 5.2.2 platform when the game starts running. One important thing to notice is that the actual game view in the Oculus Rift is not the same as the scene displayed on the Unity due to some external factors (lighting, shading, etc.). However, since it's difficult to obtain screenshots in an actual VR environment, screenshots on the Unity5.2.2 platform are used here to show the visual representation of different GameObjects, such as cars, houses and bridges.

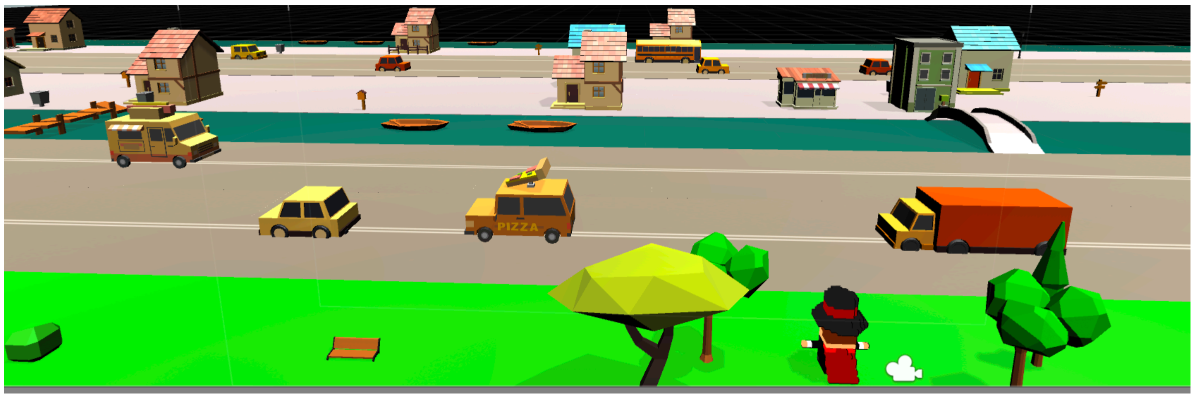


Figure 8. Game Scene2 on The Unity 5.2.2 Platform

Different types of GameObjects have different functionalities. The information about mappings between GameObject and its functionality are contained in Table 4. GameObjects And Their Corresponding Functionalities.

GameObject	Corresponding Functionality
River	If avatar fall into the river, the game will end.
Bridge	Avatar can cross the bridge without falling into the river.
Boat	Avatar can jump onto the boat, and then jump to the land to cross the river.
Car	If avatar is hit by a car of any type, the game will end.
House	If avatar collides with a house of any type, the game will end.
GameObject that doesn't belong to any of the GameObjects above(including Trees, Bench, MailBox, etc)	They are just for decoration. If the avatar collides with a GameObject that belongs to this category, the game will still continue.

Table 4. GameObjects And Their Corresponding Functionalities

6. Restart The Game

If the avatar is dead, an interface that's similar to Figure 9. Game Over Interface will pop up. The first questionmark will be replaced with the reason of death ("Drown In The River", "Hit By A Car" or "Hit The Wall") and the second questionmark will be replaced with the final score of the player when the game ends. If the player want to play the game again, he/she could click "Restart Game" button and then he/she will be navigated to the start menu of the game.

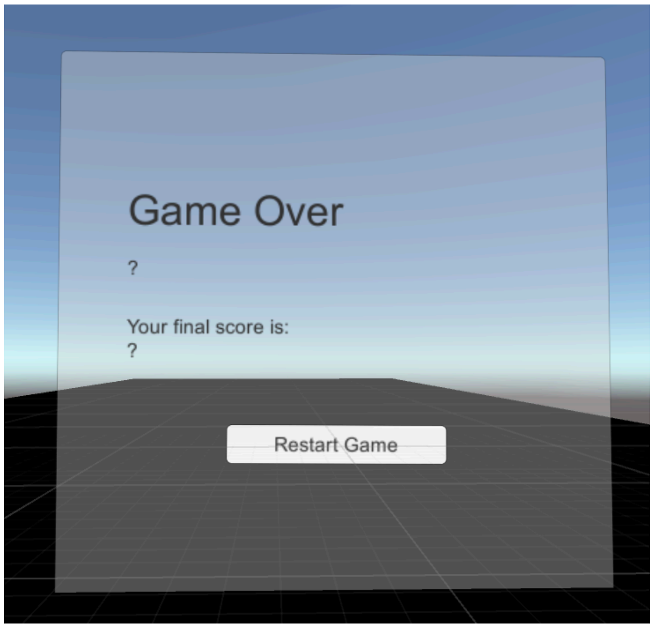


Figure 9. Game Over Interface

7. **Exit The Game**
    - If players want to the exit the game, just click the "Exit" button in the start menu (Figure 5. Start Menu) by pressing "V" in the keyboard. Then players will exit the game and return to their computer desktop.