CS 175: Roll-A-Bunny

Final Project Report

Grace Zhang, Yong Li Dich

Contents

1 Roll-A-Bunny Overview

For our CS 175 project, we decided to create a game called **Roll-A-Bunny** by building on what we've learned in class, and extending it to create this cross-platform game in Unity.

1.1 Goal

Roll-A-Bunny is a combination of two classics: Whack-A-Mole + Roll-A-Ball, as well as our love for bunnies in CS 175. The goal of Roll-A-Bunny is to roll your player (a sphere) over all 9 bunnies while collecting as many yellow collectibles (12 in total) as possible in under 60 seconds. You win if you are able to roll over all 9 bunnies and your score will be the number of collectibles you were able to collect in the process. If you are unable to roll over all 9 bunnies within 60 seconds, you will unfortunately lose the game.

Throughout these 60 seconds, the bunnies will pop up and down in their respective positions on the 3 by 3 grid and the yellow collectibles will be rotating in place, scattered across the surface.

1.2 Navigation

Your player can be controlled via the arrow keys or the WASD keys. There are 4 walls that enclose the 3 by 3 space and your player will be limited to this area.

1.3 Scoring / Metrics

- Bunnies - each bunny will only appear in its designated cell on the 3 by 3 grid, and upon collision with a bunny, it will disappear and the bunny counter will be incremented by 1.
- Yellow Collectibles - upon collision with a yellow collectible, the collectible will disappear and your score will be incremented by 1.

2 Project Components and Hierarchy

We focused on project organization by keeping our project hierarchy as organized as possible: in our hierarchy, we grouped all the bunnies together, all the yellow collectibles together, all the walls together, and all the text together.

2.1 Bunnies

2.2 Terrain

2.3 Walls

We created 4 identical walls to enclose the bunnies and our player, so that the player does not roll off the screen. These walls are duplicates of each other and their transforms were calculated and updated based on the size of our square-shaped ground.

2.4 Collectibles

The 12 yellow collectibles are yellow cubes that are all based on the same prefab. All cubes are colored yellow to draw attention to themselves as collectibles that will gain the user points. Additionally, each collectible rotates in place around the surface of the play area.

2.4.1 Rotating Collectibles

Each of the collectibles rotate in place. We were able to achieve this rotation effect by rotating the transform of each of our yellow collectibles in the `Update()` function so that the transform of each of our collectibles is updated in each frame. In our `Update()` function, we apply a rotation by a certain vector to the transforms of our collectibles, smoothed by a time delta (so that the rotations are frame rate independent and not extremely fast and jerky as they would be without the smoothing).

2.5 Text

We have 4 different texts in our game:

1. Bunny Count Text - we have a floating bunny counter in the upper left hand corner of our game. This number represents the number of bunnies that you have been able to roll over so far in the game. This text is automatically updated every time there is a trigger collision between the player (the sphere) and any of the bunnies (which are tagged as "Bunny"). More specifically, when such a trigger collision occurs, we deactivate the bunny, increment our bunny counter, and update our bunny count text.

2. Score Text - we have a score displayed in the upper left hand corner of our game. This number represents the number of yellow collectibles that you have been able to collect so far in the game. This text is automatically updated every time there is a trigger collision between the player (the sphere) and any of the yellow collectibles (which are tagged as "Collectibles"). More specifically, when such a trigger collision occurs, we deactivate the yellow collectible, increment our collectibles counter, and update our score text.

3. Time Remaining Text - we have the time remaining displayed in the upper right hand corner of our game. This number represents the amount of time you have remaining to roll over all 9 bunnies as well as collect as many yellow collectibles you can. The time remaining starts at 60 seconds and is decremented every frame based on the time that has passed. This serves as a count down of how much time you have left before your game is scored.

4. Win Text - once you complete the game, the win text will display over your player. The win text will either display "You Win!" if you won the game by rolling over all 9 bunnies, or "You lose :(" if you were unable to roll over all 9 bunnies within the 60 seconds.

3 Camera Positioning

We wanted the user to be able to focus on his/her player's actions and achieved this by updating our camera's position so that it is always following the player's position.

We first attempted to update the camera position to be relative to the player's position by making the camera a child of the player, which unfortunately led to a very roll-heavy perspective of our scene because the player's sphere is rotating rapidly to move around and so the camera's point of view rotates with it. Thus, we took an alternative approach to capture the effect of our camera following our player:

3.1 Camera Following Player

We were able to capture the effect of our camera following our player by keeping our camera and player as two separate game objects with no hierarchical relationship between them. We realized that what we wanted to achieve was to make the camera's position, relative to the player's position, remain constant.

Thus, we introduced the idea of having an offset value. We first placed the camera in a position that is able to capture our player and its surrounding space as desired. Then, in our camera's `Start()` function, we compute an offset that is the initial difference between the camera's position and the player's position. Our goal is to have the camera and player have this same offset at every frame throughout the duration of the game, so we update the camera's transform at every frame in the function `LateUpdate()` so that it is located at the player's position plus the offset. Updating the camera's transform in `LateUpdate()` rather than `Update()` allows us to ensure that we will update our camera's position only after all items have been processed in `Update`, more specifically ensuring that the position of the camera is updated only after the player's position has been updated in that particular frame. This process allows the camera to maintain the same relative position to our player throughout the game, achieving the effect of having our camera follow our player, allowing the user to focus on his/her player's behavior.

4 Design

4.1 Collision Detection

4.2 Prefabs

4.3 Assets

5 Challenges

5.1 Learning Unity

We were both new to Unity and have never coded in C# before. As such, there was a steep learning curve for us, as we had to familiarize ourselves with the new programming language, framework, and terminology. We did so by completing many different online tutorials: [1], [2], [3].

Luckily, we had already learned a lot in CS 175 this semester and were able to identify similar ideas in Unity as those that we learned and implemented in the course. We realized that Unity provides many great wrappers

5.2 Learning

test

References

[1] Unity Interactive Tutorials,
https://unity3d.com/learn/tutorials/s/interactive-tutorials

[2] Roll-A-Ball Tutorial,
https://unity3d.com/learn/tutorials/s/roll-ball-tutorial

[3] Whack-A-Mole Tutorial,
https://www.youtube.com/watch?v=m4M7VAn-bYk&t=736s