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.

Figure 1: Roll-A-Bunny in action. Our player (the white sphere) navigates a bumpy terrain to hit a dancing bunny or rotating yellow collectible cube for points. The walls of the playfield protect the sphere from falling out of bounds, and the transparency of the walls allows for a better user experience. The user's score and count of remaining bunnies are at the upper left hand corner of the screen; the instructions (and later, game results) are at the top center of the screen; and the remaining time (starting with 60 seconds) is at the upper right hand corner of the screen.

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 nine bunnies while collecting as many yellow collectibles (twelve in total) as possible in under 60 seconds. You win if you are able to roll over all nine 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 nine 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.

Once you reach the end of the game after 60 seconds, the game will be frozen in "Game Over" mode. If you would like to play again, you can hit the "space" key to restart the game.

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 bunnies left counter will be decremented 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

The bunnies were an asset we downloaded from the Unity 3D asset store. In the bunny-spirit of CS 175, we use them to represent moles as part of our inspired whack-a-mole game. The bunny asset included the bunny's graph library/scene graph of the rabbit materials and textures, scene, and vector animations of the bunny's movement. The bunnies are constructed using similar ideas that were covered in CS 175. They have a mesh over the different bones of the rabbit, which are connected as nodes within a scene graph. Then, an animation of the repetitive dancing movement can be made.

The nine bunnies we have in the game are randomly chosen to pop up in their designated cells in 1.5 second intervals. These random bunny appearances are controlled within `GameController.cs`, where we select a bunny to appear by randomly selecting a bunny from our array of bunnies. In `GameController.cs`, we have an array of bunnies holding bunny game objects.

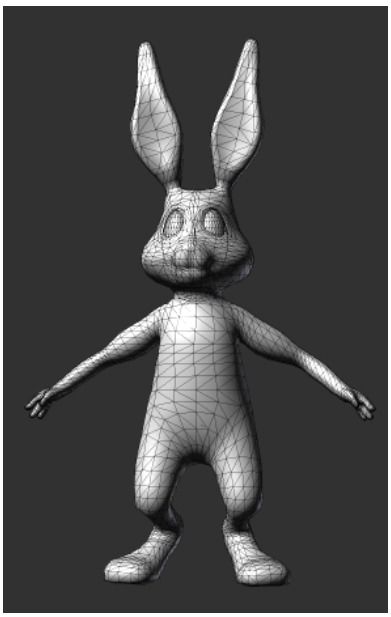

Figure 2: An image of our bunny mesh.

2.1.1 Random Bunny Appearances

In each random appearance of a bunny, in order to update the position of the bunny, we linearly interpolate between its local position and its target position depending on the circumstance. Unity has a `Vector3.Lerp(start position, target position, speed)` function that allows us to accomplish this more easily and efficiently. Similar to the lerp function we introduced in class, this means we pass in transform vectors such as the local position of the rabbit and the target position of the rabbit. This lerp functionality allows us to smoothly move the bunnies up and down.

The target position is updated when we randomly select the next bunny to appear. For example, we may choose to select bunny B1, when our algorithm indicates that B1 has been randomly selected. B1 would have been selected by randomly generating a number in the range of the length of our bunnies array, which contains all of our bunny game objects.

We also have a timer set for how long each bunny will appear on the screen. This timer is initially set to 1.25 seconds, and after 1.25 seconds, the bunny B1 will hide or disappear since this timer will have run out. The bunny B1 could also disappear before this timer runs out if our player rolls over the bunny, decrementing the number of bunnies the user has left to roll over.

2.2 Terrain

We created a 3D terrain to increase the difficulty of the game and more closely simulate reality. The 3D terrain has a terrain collider so the player would have to move up and down small bumps and hills in order to navigate the game field. We were able to accomplish this by using the terrain game object and its brush to raise and lower terrains in a natural fashion.

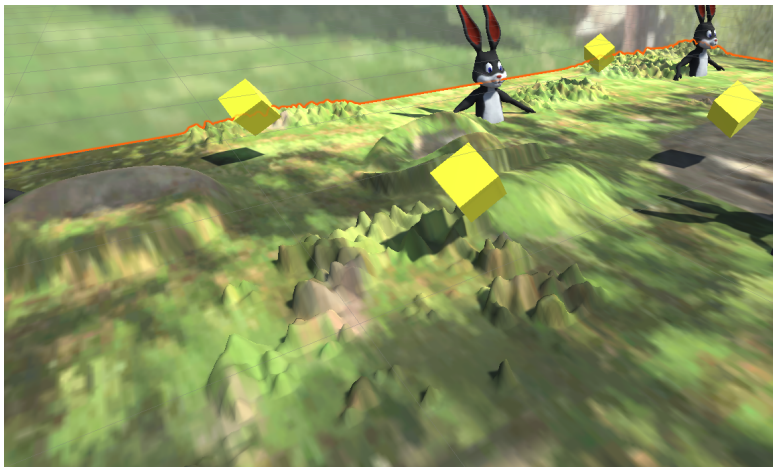

Figure 3: An image of a subsection of our terrain.

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.

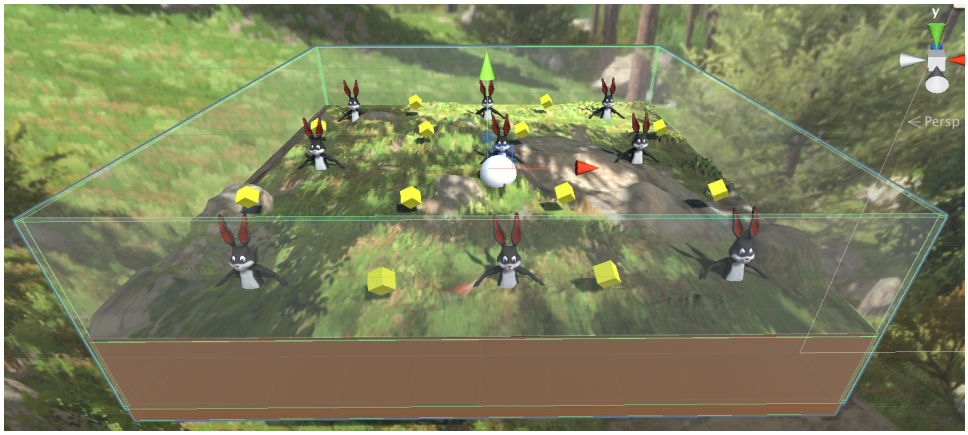

Figure 4: An image of our 4 transparent walls that enclose our play field.

2.4 Collectibles

The twelve 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.

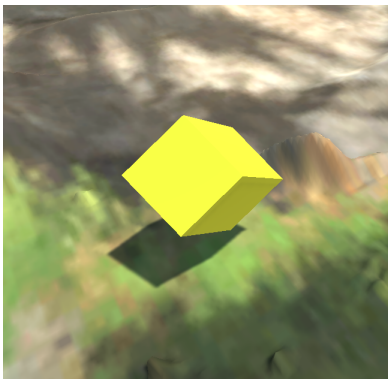

Figure 5: An image of our rotating yellow collectible.

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 bunnies counter in the upper left hand corner of our game. This number represents the number of bunnies out of nine that you have remaining in the game to roll over. 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, decrement our bunnies 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 nine 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. Info Text - at the beginning of the game, the info text will display instructions describing the objective of the game. Once you complete the game after 60 seconds, the info text will display the results of the game. This text will conditionally display the following:

    - "You Win - full score!" if you won the game by rolling over all nine bunnies and collected all twelve collectibles
    - "You win with X points!" if you roll over all nine bunnies but do not collect all twelve collectibles

- “You lose :(” if you were unable to roll over all nine bunnies within the allotted 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.

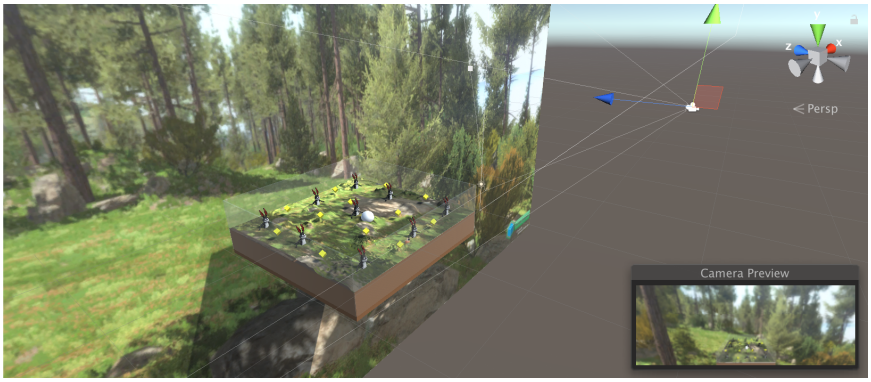

Figure 6: An image of our camera's position relative to the player's position. This offset is maintained throughout our game.

4 Design

We focused on our project's overall design so that our game runs efficiently and our codebase is maintainable.

4.1 Collision Detection

We use trigger collision detection to update our bunnies counter and collectibles counter upon player collision with a bunny and yellow collectible respectively. In particular, we had originally set up our bunnies and yellow collectibles without rigid body components, so they were considered to be static. This meant that Unity would recalculate our static collider cache every frame.

In order to increase the efficiency of our game, we added rigid body components to all of our bunnies and yellow collectibles to update them to be dynamic colliders, eliminating the need to recalculate the static collider cache every frame. Additionally, we updated them to be kinematic rigid bodies so that they will not react to physics forces and instead can be animated and moved by its transform (otherwise, gravity would pull them down and through the ground since they are triggers). In this way, we have efficient collision detection and are able to update our bunnies counter and collectibles counter upon the appropriate types of collisions.

To elaborate on our collision detection process, we also use the tag system to detect which objects our player is colliding with at each collision. We do this by introducing different tags for our bunny and collectibles prefabs. We then use the built-in function `CompareTag()` to efficiently compare the tag of any game object to the string value of our tag of choice.

4.2 Prefabs

We have used prefabs to make our codebase more well-structured and maintainable. In particular, prefabs allow us to create GameObjects with specific components, property values, and child GameObjects as a reusable Asset. This allows us to avoid duplication of code and instead create instances of the same GameObject when needed in the game.

In particular, we took advantage of prefabs in our implementation of our bunnies and yellow collectibles. We have a bunny prefab and a collectible prefab, which we were able to utilize to create instances of our nine bunnies and twelve yellow collectibles. This allows us to have more easily maintainable code as we can now make changes to all of our bunnies or collectibles at once, rather than having to update each bunny or collectible individually.

4.3 Assets

We have spruced up our project and have made it look more similar to reality by taking advantage of different Unity assets.

We downloaded an animated bunny asset as part of our game for users to try to roll over all nine of our bunnies. We also create prefabs using our assets to allow for more efficient updates of the same game objects. For example, a bunny has its own prefab so when one bunny is updated, all the other bunnies are also updated.

Throughout the game creation process, we stored different scenes to work with for code management purposes. We also use terrain and background assets to help us build and simulate a forest environment.

Figure 7: An example of our bunny asset, which we use to generate our nine bunnies in Roll-A-Bunny.

5 Challenges

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], [4].

We learned a lot about C#, object oriented programming, hierarchical ordering of game objects, timers in games, and state-tracking (for example, via our bunnies counter and collectibles counter). We also learned to reload a scene and freeze a scene, which are interesting parts of the Unity game engine library that we have incorporated. Controlling the camera and assigning the point of view of the player were also important considerations of this project, and we spent quite some time discussing the benefits and drawbacks of various implementations.

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 that accomplish tasks similar to those we achieved in our problem sets - it was very rewarding to utilize these built-in functions while having a good sense of what is going on under the hood.

6 Further Work

While we are very excited with how our game turned out, we definitely also have ideas on how we can improve our game in the future. In particular, we would like to further take advantage of the various Unity assets that ware available to create more than just bunnies; instead, we could potentially roll over bunnies, robots, squirrels, etc.

We also hope to be able to include more physics. Specifically, when the ball rolls over a patch of sand, perhaps some of the sand can fly outward in response. Similarly, when we roll over a bunny, perhaps there is some visible impact on the surrounding terrain that we could simulate.

Thanks for all the help this semester - we've had lots of fun and learned a lot!

References

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

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

[3] Unity Scripting Reference,
https://docs.unity3d.com/ScriptReference/

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