CS304: Game Programming - Fall2019

Date: Nov 22, 2019

Assignment # 3, 4

Due: Part 1 is due Thursday, Nov 29, in class timing and Part 2 is due Tuesday, Dec 10, in class timing.

Late policy: Up to 6 hours late: 5% of the total; up to 24 hours late: 10%, and then 20% for each additional 24 hours. Submission instructions will be given later.

Plagiarism: Sharing of source code and copying of assignments is highly discouraged. Failing to comply will lead to zero marks in assignment.

The game's structure is based on the *Unity Roll-A-Ball tutorial* (see https://unity3d.com/learn/tutorials/s/roll-ball-tutorial).

Part 1: Make the following modifications to the Roll-A-Ball tutorial:

Replace ball with cube: Replace the rolling ball with a cube, called the *Player*. (Generally, you can pick any shape you like, provided that it is rotationally asymmetrical, so that it is possible to distinguish which side is the front.)

Material: Replace the material/texture applied to floor and pickups of your own choice.

Gliding motion: Rather than rolling, the Player glides on or just above the ground. The left and right arrow keys (or the 'A' and 'D' keys) cause the Player to rotate counter-clockwise and clockwise, respectively. The up and down arrow keys (or the 'W' and 'S' keys) cause the Player to move forward and backward, relative to the direction it is facing. (As in the Roll-a-Ball tutorial, you can use Unity's Input.GetAxis commands to access these inputs.)

Camera follows behind Player: Rather than placing the camera above the entire scene, the camera follows the Player from behind and slightly above, as is common in action-adventure games (see Fig. 1).

Easy to win: To make the grading simpler, rather than requiring the Player to collect all the pickups, there should be a public variable called pickupQuota in your PlayerController class that will allow the grader to control the number of pickups that must be collected in order to win the game. The grader should be able to adjust the value of this variable in the Unity editor (in the Inspector window for the Player object).

Audio: Add background music and audio clips to different events like when player collects pickup, hits side walls or when maximum pickup quota achieved.

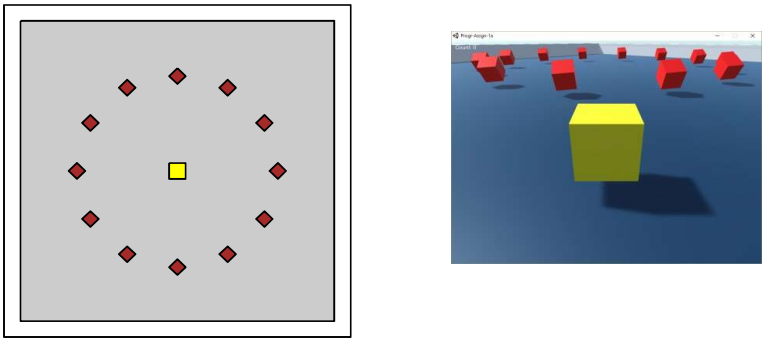

Figure 1: Part 1: A top-down view of the initial game layout and a sample view with the player in the south-east corner looking north-west.

The rest of the game is the same as Roll-A-Ball. The Player starts at the center of board, moves around and collects pickups by running over them. The number of pickups collected should be displayed. When the Player has collected the given quota of pickups, an appropriate message should be displayed in console window.

Part-2: In this part you will create a two-level game. The first level will consist of Part 1, and the second level will be a shooting game. Before reading the description below, you may want to just try playing the game. (See the link below for the sample executable.)

Environment: The environment consists of square platform with walls forming a maze-like structure (see Fig. 2). You are free to creatively modify this arrangement.

Figure 2: Part 2: A top-down view of the initial game layout and a sample view. We have replaced the default Unity skybox with a solid color.

Player: As in Part 1, the Player object is a cube (or any rotationally asymmetric shape) the moves in the same manner as in Part 1, but has two additional capabilities.

- Whenever the space bar is hit, the Player jumps vertically, and gravity pulls it back down. Repeatedly hitting the space bar (while in the air) causes Player to jump higher and higher. In our implementation, the player's motion controls function even when the player is in the air. You can choose to do the same or to disable motion while in the air.
- Whenever the left mouse button is pressed, the player shoots a projectile horizontally in the direction that it is facing. In our implementation, the projectile is a prefab consisting of a group of five green spheres, all encased within a capsule collider.

Pickups: The pickups can be rendered as in Part 1 (rotating cubes), but they should also oscillate up and down, somewhat like a yo-yo. Unlike the Player, which is controlled by gravity, the pickups should oscillate in a periodic, sinusoidal manner.[1] Their oscillations should *not* appear to be synchronized with each other. (This can be done through the use of a random-number generator, or you can define some public variables in your controlling script that can be adjusted within the Unity editor.)

Scoring: Whenever the Player fires a shot that hits a pickup, the Player is awarded the pickup. When a sufficient number of pickups are hit, the player wins. As in Part 1, the number of pickups needed to win should be adjustable through a public variable.

Hitting a Pickup: Unlike Part 1, hitting a pickup is fatal to the Player. You lose immediately.

Start Menu: The game begins in a start menu with three buttons: One to play Part 1, one to play Part 2, and one to "Quit", which ends the program. As each of the levels is ended, you return to this menu.

[1] This is because we want you to combine kinematic game objects with those that are not kinematic, that is, controlled by physics.

(There is a nice online tutorial for creating such menus. We will provide a link in the class Projects page.)

Quit: From within Parts 1 or 2, it should be possible to quit the level at any time by hitting either the 'Q' or ESC keys. This should return control to the start menu, from which either level can be restarted. Also, from the start menu, the game can be terminated by hitting the "Quit" button.

Delaying between Levels: We would like the transition to the start menu to involve a short delay.[2]As in Part 1, at the end of the level, you should print a short message to the center of the screen (e.g., "You Won!", "You Lost!", or "Quitting"). Then, after a short delay (we used 2 seconds) the Start Menu screen should appear.

Exploding pickups: When a pickup is hit (either by being shot or being hit by the Player object) an explosion effect should result. The easiest way to implement this is to use a Unity particle system. (There are numerous online tutorials about generating explosions. Feel free to steal one, but remember to cite your sources.)

Dimming the Screen: One of the nice effects in our implementation is that when displaying text in the center of the sceen, we dim the screen but still allow the background to show through. See if you can figure out how to do this. Hint: It only takes a few lines of code and involves a semi-transparent UI element.)

Prioritizing: There are quite a few items listed above. They are listed in order from most-important to least-important. If you cannot finish all of them, we will give partial credit, but you will maximize your score if you complete them in top-down order.

[2] This is mostly to give you exposure to coroutines.