COMS W4172: 3D UIs and AR—Spring 2020

Prof. Steven Feiner
Date out: February 4, 2020
Date due: February 25, 2020

Assignment 1: On a Roll

Introduction

For your first assignment, you will be building your own mobile phone game! This will be an improved "Platformer" version of the Unity Roll-a-Ball tutorial, but with touch-based input and other features. Your task is to design and build the controls, environment, game conditions, obstacles, and user interface. As long as you meet the requirements described below, feel free to get creative and make your game as interesting and fun as you want!

Note: Unity is a powerful development environment. However, that power comes at the price of a sizable learning curve: You'll need to get comfortable with a workflow that may be different from what you're used to, along with what will probably be an unfamiliar language, C# (although you should already know the basic object-oriented programming concepts underlying it). Therefore, ***please start as early as possible on this assignment,*** so you can explore and become comfortable with the editor, and begin designing your game enough in advance that you'll have time to come to office hours if you need assistance. Do take advantage of Unity's extensive online Manual and Scripting API Reference, along with its many free tutorials. And remember that only **one** of your **four** late days may be used on this first assignment; so, the latest you can turn it in will be midnight on February 26, using that late day.

Please start by either working your way through the Unity Roll-a-Ball tutorial (or reviewing it, if you've done it previously). However, you do not need to use the actual code (or any of the simple Unity primitive-based models) from that tutorial.

Assignment Requirements

1. **Touch-Based Movement:** The ball is controlled by touch-based input. If the player touches the phone screen at a point on the ground , the ball should **roll, *not* slide,** in the direction of the touched location. How this works however, is up to you. (Does the ball gain speed from multiple taps? Or from holding the touch for a longer period of time?) The ball should also be able to "Jump" upwards in order to get over obstacles. Again, how this is done is up to you (e.g., it might be accomplished by tapping on the ball directly). You should use the Unity Touch struct to accomplish this, along with raycasting to cast a ray into the scene through the point you touched from the camera's perspective. Please see the Camera.ScreenPointToRay() function and the section below on "Raycasting".
2. **User Interface:** You should use the Unity UI to present the player with a Canvas object on the screen that shows:
   a. Controls

i. A Start Button to begin the game.
ii. A Restart Button after the game is over.
iii. Any other controls you wish to have to activate actions or events in the game, such as a Jump button or speed slider.

b. Current Game Status
   i. See “Time and Points” below.

3. **Platforms:** Your game will be structured into **five** different platforms (areas that your ball must traverse) and each platform should have at least three collectibles (described below). Each platform can look however you like and can be at a different height than the other platforms, as long as it meets the requirements.
   1. **Platform 1** is the starting area, and the place to which the ball’s position is reset when the Restart button is pressed. A ramp should lead to Platform 2.
   2. **Platform 2** should have obstacles (each could be as simple as a cylinder) around which the player must navigate the ball. Each obstacle should be solid, preventing the ball from passing through it. To get to Platform 3, the ball must be carried by the “Rotating Objects” described below.
   3. **Platform 3** should contain some object that grows and shrinks (and could additionally translate and rotate) periodically, so that it could knock the ball off of the platform if the player does not time the ball’s movement correctly. To get to Platform 4, there should be another ramp leading to a gap that can only be crossed by building up enough speed using a “Velocity-Boost Area” (described below).
   4. **Platform 4** should have a “Rotating Floor” (described below) at about the same level as the ground. If the ball rolls onto the Rotating Floor, the ball’s movement should be affected accordingly. To get to Platform 5, there should be a “Falling Bridge” (described below).
   5. **Platform 5** signifies the end of the game. Once the player reaches it, the game should notify the player that they have won, display the score and time, and display the Restart Button.
4. **Lights and Cameras:** The ball should be illuminated by a spot light that translates and/or rotates to follow the ball. (You are welcome to include additional lights to improve the appearance of the scene.) There should be two cameras that the player will be able to switch between by using the UI. The first camera should follow the ball at a set offset and there should be some way to rotate the camera to improve the player’s view. The second camera will be a stationary camera through which the player can view the entire game environment.
5. **Rotating Objects:** You must include in your game Rotating Objects that convey the ball from Platform 2 to Platform 3. Here’s how they should be configured:
   a. At least four objects should rotate much like a Ferris wheel around an axis parallel to the ground of Platform 2.
   b. Each object in (a) should be associated with a secondary set of objects that rotates around it, much like a carousel on an axis perpendicular to the ground of Platform 2.
   c. Here is a video of the kind of rotations we are looking for. Note that the ball should be able to “hitch a ride” on the objects of (a) and/or (b) to get from Platform 2 to Platform 3.

6. **Rotating Floor:** While Platform 4 is stationary, there should be an object (e.g., a plane) that is barely above the platform floor. This object should be constantly rotating about an axis perpendicular to the floor and will affect the ball's velocity if it rolls over it.
7. **Size-Changing Obstacle:** This object periodically grows and shrinks. For example, it could be a cube that grows and shrinks over time, enabling it to knock the ball off of the platform if the player does not time the ball's movement correctly. In addition, this object could also translate and rotate.
8. **Selection and Speed Change:** For the Rotating Objects, Rotating Floor, and Size-Changing Obstacle above, you will need to implement speed controls. If the player touches any of these three, a slider should appear to control the movement speed of the object touched: the rotational speed of the Rotating Objects or Rotating Floor, and scaling speed of the SIze-Changing Obstacle.
9. **Velocity-Boost Areas:** This is a surface or area in the game that increases the ball's velocity if it comes in contact with it. If the ball and rolls onto this area, its speed should increase **but its direction of motion should stay the same.** This would allow the ball to make a jump it normally couldn't, or reduce the time it takes to traverse an area.
10. **Falling Bridge:** This is a bridge the ball must cross, but which incrementally breaks apart as the ball contacts it. For example, the bridge could be made up of cubes, each of which starts to fall under the influence of gravity (or due to a change in the cube's constraints) a short time after the ball contacts that cube. Crossing the bridge successfully would thus require having the ball move off a cube quite soon after touching it. Hint: To accomplish this, you could implement the time delay by using Coroutines (e.g., with WaitForSeconds()) or the Invoke() function. The bridge should function similarly to the one in this video. The bridge should be reset to its original state when the game restarts..
11. **Time and Points:** Your game should display two pieces of data, both of which should be reset to zero when the game is restarted:
    a. Time: The time elapsed since the game started, which should be constantly updating.
    b. Points: The number of collectibles the player has acquired by colliding with the collectibles spread throughout the platforms. See the "Roll-a-Ball" tutorial for an example of how to implement Collectibles.
12. **Collectibles:** Much like in the Roll-a-Ball tutorial, your game will include Collectibles. These can be diamonds, coins, crystals, or whatever you want, as long as they disappear upon colliding with the ball and a point is added to the score in the UI.
13. **Out-of-Bounds Area:** This should be an area that the ball touches if it falls off one of the platforms. When the ball touches the Out-of-Bounds area, it should be reset to an initial position of your choice in order to continue playing the game (e.g., the start of this platform or a location on the previous platform).
14. **Assets:** You should use at least two free downloaded models. These could include a unique look for your ball, a well-made ramp, interesting looking platforms or obstacles, or environmental objects such as trees. (All other objects can be made from simple Unity primitives, much as was done in Roll-a-ball.) Since we do not want you to have to create your own 3D models, you can download models (*free ones only, please*) for all objects in your scene from the Unity Asset Store, or any other source (e.g., the ones listed on our IA

page), providing you have permission and cite each source properly in your documentation. Alternatively, you can load a model (.fbx file, .obj file, or other supported formats) obtained elsewhere by dragging its file into the **Project** View in the Unity Editor Window. Any associated textures should be added to a "Textures" folder placed next to the loaded model in the **Project** View. You can also import an entire directory at once. While you are also welcome to create any of your objects directly from Unity primitives, you should use at least two downloaded models. Again, any model that you use must be free. Also, while the Unity Asset Store has many free assets to use in projects, including sounds and music, characters, animations, environments, special effects, and more, **you may not use any scripts/code that comes with these assets**. For example, if you wish, you may download an actual Ferris-wheel model asset, but you may not use any scripts that came with it.

Example gameplay with a possible version of this assignment

1. The player starts the game and is presented with the title of the game and a start button. After the button is pushed, the game starts and the rest of the UI appears around the edges of the screen, including the timer that starts from 0.
2. The player touches any point on the ground area of Platform 1 and the ball starts rolling in the direction of that point. The player touches the same point multiple times and the ball increases in speed, but accidentally goes too far and rolls off of the platform. The ball hits the Out-of-Bounds area and the ball's position is reset to its original starting point.
3. The player is more careful this time and moves the ball up a ramp to reach Platform 2. Once there, the player moves the ball around several obstacles and reaches the rotating objects. The player moves the ball onto one of the rotating objects and the ball is raised up near Platform 3. The player touches the ground on Platform 3 and the ball rolls off of the rotating object and onto Platform 3.
4. After reaching Platform 3, the player sees that the next obstacle is a cube that is constantly changing size. When small, the path is clear and the ball can move forward; when large, the path is blocked. If the cube is currently growing, there is a chance that the growing cube will knock the ball off the platform. The player navigates the ball past this obstacle and sees a gap that the ball must get across by going through a Velocity Boost Area that will make it possible to go up a ramp and successfully make the jump to Platform 4. The player moves the ball through the Velocity-Boost area, the velocity of the ball is increased, and it makes the jump to Platform 4.
5. On Platform 4, the player sees that the middle of the floor is rotating. The player tries to move the ball through this part of the floor, but its velocity is changed by the rotation of the floor and it almost falls off. The player moves the ball past this obstacle and on to the last one; the falling bridge.
6. As the ball starts rolling across the bridge, each cube in the bridge that the ball has made contact with starts to fall a little while later (as if its gravity had been turned on one second after colliding with the ball.) The player successfully navigates the ball to Platform 5 and the game is over. The player can read their score and time and can press the Restart button if they want to play again.

Hints

Documentation. Before starting this assignment, please look through the Unity Manual, to get a better feel for the Editor and see the reference page on Input for a comprehensive overview of its functionality. You can also take advantage of some of the many other free Unity Tutorials (in addition to Roll-a-ball).

Development approach. To do this assignment well, you should think *carefully* about how you structure your scene graph hierarchy. Which objects should you use and how should they be arranged in the hierarchy? (The relationship between a parent and its children is important, while the order in which siblings are listed should not matter for this assignment.) How should the transformations that you apply to your objects be composed to achieve the required effects? Begin with just the ball and touch-based input to experimentally verify that your approach works, so you can modify it early on if necessary. Next, incrementally add the five platforms, the first ramp, the Out-of-Bounds area. Then create the functionality of the more complicated features, such as the Rotating Objects and Falling Bridge, adding them to the scene as you progress. Test early, test often!

Ten Usability Heuristics for User Interface Design. Your user interface should follow the Nielsen usability heuristics. We will not grade your work based on the aesthetics of its models (since we do not want you to buy any assets and do not assume you have any modeling skills) or on physical realism. Other than that, we will be evaluating your work with the heuristics in mind.

Hierarchy. Understanding rotation is crucial here. Note that when an object is rotated, its descendants will also rotate. Therefore, if you want object *B* to act as if it were a descendant of object *A*, but not be affected by *A*'s rotation (e.g., to have *A* rotate at a different rate than *B*), the easiest way to do this is to create empty GameObject *A′*, make *A* and *B* both children of *A′*, where *A* is centered at *A′* and *B* is offset from *A′,* and then rotate *A* and B individually. If you do this, transforming *A′* will transform all its descendants, but *A* and *B* can each have its own independent rotation.

Model Files. Each model file you find will, when you find it, most likely *not* be of an appropriate scale relative to the other objects you're using. Therefore, be prepared to apply a scale transform to one or more of your models to bring them up or down to a reasonable size. In addition, note that some models may contain too many polygons for your mobile device to render your scene at a reasonable frame rate. *Before you get too enamored of any model, please try it out on your device in context with the rest of your scene to make sure that it will work well.* See also How do I fix the rotation of an imported model?

Raycasting. When using the Physics.Raycast function, you will be returned a RaycastHit object. The RaycastHit object contains a reference to a Collider. The Collider contains a reference to the GameObject to which it is attached. You can use that reference to determine the object with which a raycast collided through the screen. You will use raycasting for your

touch-based input, in that when you touch the screen, a rayt will be cast through the point you touched (from the camera's perspective.) You will accomplish this using the Camera.ScreenPointToRay() function.

Rotation and Orbiting. In Unity, the Transform.Rotate *method* specifies a *relative* orientation change that will be composed with the current orientation. Transform.Rotate needs to be given the amounts to rotate as Euler angles about *x*-, *y*-, and *z*-axes, either as three separate floats or as a Vector3. It will apply a rotation to the object about the *z*-axis, *x*-axis, and *y*-axis (*in that specific order,* as discussed in class). You should also read about and use Transform.RotateAround. Please be sure that you understand the relativeTo parameter of Transform.Rotate and how Transform.{right,up,down} differ from Vector3.{right,up,down}. In strong contrast to the Transform.Rotate method, the Transform.{rotation,localRotation,eulerAngles,localEulerAngles} *properties* set the *absolute* orientation of a Transform (i.e., will override and ignore its current orientation) and do this in ways that offer many possibilities for you to do the *wrong* thing: *none* of the elements of Transform.{rotation,localRotation} is an angle (they are the components of a quaternion) and *none* of the angles of Transform.{eulerAngles,localEulerAngles} should be set individually or incremented!

Play Mode. While Play Mode provides a very useful way of debugging your app, it is not a reliable indicator of how it will run on your mobile device. *Make sure you test directly on your device whenever you introduce a new imported asset.* While debugging, you will want to include secondary controls that are guaranteed to work on your desktop or laptop in Play Mode, such as Input.getMouseButton(). Since these controls may affect performance slightly, you will want to disable them when running on your mobile device. (Or if you feel comfortable with your app's performance on your device, you can choose to support both modes.)

Unity Remote. Unity Remote makes it possible to run an application in Play Mode on your laptop or desktop, while having its output and input appear on and be obtained from your mobile device, all without any need to deploy to that device. This can make debugging far more pleasant and less time consuming than it would be if you had to deploy every time you changed anything. However, it is important to note that Unity Remote will not provide a reliable indicator of how the application will run on your mobile device and is known for having "issues." This is especially true regarding *rendering* and *timing*, since Unity Remote renders everything on the computer on which the editor is running and sends the rendered content as compressed images through the cable to your mobile device. As with Play mode, *make sure you test directly on your device whenever you introduce a new imported asset.* Overall, as the documentation warns, "Bear in mind that Unity Remote is only really intended to give a quick approximate check of how your game will look and feel when running on the device. Make sure that you occasionally do a full build and test the 'real' app." (Better still, replace "occasionally" with "frequently" here.)

Textures. When you load in a model and associated texture in two separate load steps, the texture might not connect to the model automatically. In order to connect the texture to the model, drag the texture onto the Texture box in the Material component of your model in the

Inspector View. When deploying to mobile, materials and textures (that are not already placed on objects in a deployed scene) it is important to place texture and material files into a dedicated folder (Assets->Resources). This will ensure that they are deployed with the application executable and can be found by your code (usually executed in a Resources.Load function).

Transparency. It might be tempting to make some objects semitransparent. But please note that objects that have any portions that are not opaque are treated differently from opaque objects in how and when they are rendered, because of the hardware rendering algorithm used in current interactive graphics systems. *This should not be a problem if you have only a single planar object that is not opaque*. However, this can cause multiple non-opaque objects to render incorrectly and inconsistently as their positions change relative to each other and the camera. Since you will be using a transparent sphere to show the bounds within which a plate can move, we strongly suggest that you do not make any other objects transparent.

Multi-Touch (optional). The user might want to use multiple fingers on one or both hands. To make this possible, you are *optionally* welcome to support multi-touch interaction. See Multi Touch Input for more information.

What You Should Turn In

A Compressed/Zipped file containing your entire project. You can do this by navigating to the file where your Unity projects are stored and compressing it there. This file will contain:

1. Your Unity project in its entirety.
    a. **Do not include the app executable (or the XCode project for iOS).**
2. Your **well-commented** code.
3. Your readme.txt file that includes
    a. Your name
    b. UNI
    c. Date of submission
    d. Computer platform
    e. Mobile platform, OS version, and device name
    f. Description of your project, what you did, and how you accomplished it.
    g. Any problems you overcame (both coding and technical)
    h. A list of any free assets you used.
4. Finally, a brief (under five minutes, please) video demonstrating your application's features. While we are not expecting a painstakingly scripted production, please practice before recording it and do some simple editing to remove unnecessary pauses. Submit this as a link in your README file and in the CourseWorks File Upload comment to an ***unlisted*** video on YouTube or Google Drive. The upload time of your video will be the time at which we will consider it to have been submitted.

How to submit

Please compress all files in your submission into a single zip file, remembering to include any needed data files. Please follow the naming convention “YOURUNI_Assignment1.zip” for your submission. Name your video “YOURUNI_Assignment1,” upload it as an *unlisted* video on YouTube and include the URL in your submission, as described below. Submission should be done through CourseWorks, following these steps:

1. Log into CourseWorks.
2. Select Assignments from the left-hand navigation pane.
3. Click the Submit Assignment button in the top right corner.
4. The Submit Assignments page will load. Choose your zipped project using the browse dialog window that appears after pressing “Choose File.”
5. After choosing your project, copy the URL of your ***unlisted*** video upload into the Comments field beneath the File Upload section.
6. Press “Submit.”

Please try to submit well before the deadline, since CourseWorks can sometimes become busy and slow. You can resubmit multiple times. (Note: CourseWorks will save your previous comments, so you don’t need to re-enter your URL if it has not changed, but CourseWorks will clear your previous upload from the File Upload section.) You can add a file you previously uploaded by clicking “Click here to find a file you've already uploaded,” expanding the Unfiled folder and selecting your file, then pressing “Submit.”

Immediately after uploading your submission to CourseWorks, please check it by downloading it, creating a new project with which to test it, and reading through its README.txt file. We will not accept as an excuse that you accidentally uploaded the wrong file. We also regret that we *cannot* accept files that are on your computer, even if they appear to have the correct date.

Remember, you can use only a *single* late day on this assignment, so start early! And, have fun!

Grading

Input and Player Control	10 points
Motion	25 points
Platforms & Structure	25 points
User Interface (Usability Heuristics)	25 points
Lights & Cameras	10 points
Documentation (including video)	5 points
Total	**100 points**