Assignment 1: Taking Unity For A Spin

Introduction

This assignment will be the first Unity project you develop on your own, after having completed Assignment 0 & Assignment 0.5. You'll be using Unity and C# to design a virtual 3D amusement park with which the user can interact using your mobile device’s touchscreen. Your amusement park should include a carousel/teacup amusement-park ride and a flying drone.

Since we do not want you to have to create your own 3D models, you are free to download models from the Unity Asset Store, or any other source (e.g., the ones listed on our TA Wiki), 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) 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. (There are plenty of free models available, so you should not have to purchase any.)

Platform and Passenger Objects

Your amusement-park ride should have a static planar base (the ground), a rotating platform, and at least three passenger objects: a “galloping” passenger object, a spinning teacup passenger object, and a creative third passenger object of your own imagining. The platform should be a descendant of the base and each passenger object should be a descendant of the platform in the scene graph hierarchy (but, see the hint about hierarchy below).

The platform should rotate around an axis parallel to the Y axis at its center. The “galloping” passenger object should oscillate up and down like a carousel horse as the platform rotates. The spinning teacup passenger object should orbit a point on the platform and simultaneously rotate about its own axis.

The third passenger object should have at least one intermediate ancestor between it and the platform, and the passenger object and at least one of those intermediate ancestors should have transforms that change during execution. For example:
In this example, PassengerObjChild (the object on or in which a passenger would be seated) changes its transform relative to PassengerObjParent, which changes its transform relative to the platform in the scene graph hierarchy.

**Drone**
The drone is any object that is moving in the air, that can give a moving, aerial view of the scene. The drone should be relatively small when compared to the passenger objects, but should not be difficult to see. The user should be able to control the drone’s direction, orientation, and speed, as discussed below.

**Lights**
You will need at least one light in the scene graph in order to view the scene.

**Selection**
The user should be able to select any object in the amusement park (base, platform, passenger objects and drone) using your mobile device’s touchscreen. (Direct selection using the touchscreen can be achieved using the Ray object. See the Physics.Raycast function and the ray casting tutorial.)

Once an object has been selected, you should change it in some way to indicate this visually. (If you’d like, you can also indicate this sonically.) For example, the object could change texture to a texture that marks it as selected. (To put that another way, even though all the objects have different textures, when one is selected, you could make apply a different texture to it or do something else that indicates that it is the selected object.) Please think about which approach would be most effective.

Initially, no object should be selected. At most one object should be in the selected state at a time. You should also support some way to deselect the currently selected object without selecting another object. You are welcome to use an on-screen button to enable selection, to avoid accidental selection as the user navigates the scene.

**Control Panels**
Referring to the Unity UI, create a partially transparent control panel (see the Canvas documentation and the Canvas Manual) for each object. These panels should be placed in a position of your own choosing. Each panel should be visible only while the object to which it belongs is selected. Thus, none of the object control panels should be visible when the application is initialized. The panels will contain controls (made with Interaction Components and Visual Components) for the parameters of the actions you assign to your objects (see next section for details).

There should also be one camera control panel, which should be visible at all times.

**Object Actions**
**Base**
When the planar base is selected, all the objects in the scene should stop moving, pausing at their current pose.

**Platform**
When the platform is selected, it and its descendants should initially stop moving, pausing at their current poses. You should give the user the ability (e.g., by interacting with the platform's control panel) to manipulate the speed and rotation direction of the platform.

**Passenger Objects**
When a passenger object is selected, it and its descendants should initially stop moving, pausing at their current poses. You should give the user the ability (e.g., by interacting with its control panel) to manipulate its movement (including the movement of its descendants). When the object is selected, a control panel should appear that has different controls depending on the type of object.

- Teacup: The user should be able to adjust both rotation speeds, both rotation directions, and the offset of the cup from the point on the platform around which the cup orbits.
- “Galloping” object: The user should be able to adjust the speed and range with which the passenger object moves up and down.
- Third passenger object: The user should be able to adjust at least three parameters that influence the of this third class of object. These parameters should match your description of the movement of the object in your readme.

Once the base, platform, or any passenger object is deselected, it (and any descendants) should resume moving, taking into account its orientation at the time at which it was deselected.

**Drone**
The drone should be able to move freely about the environment. The user should be able to control the drone using a GUI menu. The user should decide the direction and speed of the drone. If the user sets the speed of the drone to some non-zero value, it should begin moving along its forward axis. The user should be able to set the drone to any orientation.

If the drone “collides” with an object, you should indicate on-screen that this has happened and reposition the drone to a “safe” location. (To accomplish this, please refer to the Unity documentation on trigger collisions and the video tutorial).

**Camera**
Your user should be able to place the camera in one of three modes:

*Default mode*: The camera should be initially located and oriented such that your entire ride is visible. In this mode, the camera should only be able to move in the XZ plane, bounded by min-max XZ coordinates of your choosing. It should be controllable using a virtual analog stick or a directional pad. Note that Unity comes with a virtual joystick (MobileSingleStickControl),
which you are welcome to use (see the TA wiki list of Unity Tips). To use it, import it using:

(Menu bar) Assets→Import Package→CrossPlatformMobileInput. Then, after it is imported,

(Project View) Assets→Standard Assets→CrossPlatformMobileInput→Prefabs→
MobileSingleStickControl.

The camera should be able to rotate as well: The user should be able to control the camera’s pitch and yaw. The pitch should be limited to −80° to +80°, the yaw should be unconstrained, and the camera should not roll. When the user returns the camera to this mode, it should be placed at the position and orientation that it last had in the mode.

Rider mode: The user should be able to “attach” the camera to any passenger object, such that the camera is placed in the eyes of a rider on the passenger object. When the camera is attached as the rider, the camera’s position and orientation no longer need to be directly controlled by the user, as they were in Default mode. In addition, the camera should be affected by the motion of the passenger object. Keep in mind that, in general, the rider will not be positioned at the center of the passenger object.

Drone mode: The user should be able to “attach” the camera to the drone, such that it is rigidly mounted to the drone, translating and rotating with it. When the camera is attached to the drone, the camera’s position and orientation no longer need to be directly controlled by the user, as they were in Default mode.

Note that objects that are outside the bounds established by the camera’s near and far clipping planes will not be visible.

Hints

Before starting this assignment, please note that there is an extensive collection of Unity Tutorials. We strongly suggest that you review the Roll-a-Ball tutorial. Additionally, you may want to look through the Unity Manual, to get a better feel for the Editor.

Please see the Unity reference page on Input for a comprehensive overview of the functionality of both the desktop and mobile input classes. Please read through the document to familiarize yourself with the features and limitations of both classes.

To do this assignment well, you should think carefully about how you structure your scene graph hierarchy. Which nodes should you use and how should they be arranged in the hierarchy relative to each other? (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 base, platform, and one simplified passenger object to experimentally verify that your approach works, so you can modify it early on if necessary.
Regarding 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.

Regarding model files: The model files you find will most likely be of an inappropriate 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 attached to a particular model, please try it out on your device in context with the rest of your scene to make sure that it will work well.

Regarding ray casting: 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 what object was collided with, using the ray cast through the screen.

Regarding rotation: In Unity, the “Rotate” function in the “Transform” class specifies a relative orientation change that will be composed with the current orientation. In contrast, the Rotation, LocalEulerAngles, and EulerAngles variables will set the absolute orientation (i.e., will override the current orientation). The Rotate function needs to be given the amount to rotate in the 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).

Regarding the Unity virtual joystick (MobileSingleStickControl): Unfortunately, this asset is not well documented; but it is a useful feature. Please see the Developing with Unity and Vuforia guide on the TA wiki for instructions on how to use the virtual joystick. You are also welcome to create a Directional Pad instead, using standard GUI assets, or use any C# joystick equivalent that you write yourself or find on the internet (though you must provide credit).

Regarding Play Mode: There are two sets of importable Standard Assets: desktop and mobile. The mobile assets may use particular features in the mobile APIs that do not work on the desktop. So while debugging, you will want to include secondary controls that are guaranteed to work on in Play Mode, such as Input.getMouseButton(). Since these controls may affect performance slightly, you will want to disable them when not debugging, or if you feel comfortable with your app’s performance on your device, you can choose to support both modes.
A second note on Play Mode: Play Mode is a very useful way of debugging your app, but it is not fully indicative of how it will run on your mobile device. Make sure you test fully on your device whenever you introduce a new imported asset.

A note on 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.

What to submit

Your submission should include:

- The entire Unity project folder compressed.
  - Do not include the app executable (or the XCode project for iOS).
- A readme file with:
  - Your name & UNI
  - Date of submission
  - Computer Platform
  - Mobile Platform & OS version & Device name
  - Project title
  - Project directory overview
  - Description of the intended motion and purpose of your third passenger object
  - Special instructions, if any, for deploying app
  - Instructions for using app
  - Missing features
  - Explanation of bugs in your code and Unity
  - Asset sources
- A brief video demonstrating your application’s features.

How to submit

Please compress all files in your submission except for the video 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 to a private URL on Google Drive or YouTube and include the URL in your submission as described below. Submission will be done through CourseWorks (Canvas) using the following the 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 private video upload into the Comments field beneath the File Upload section.

6. Press “Submit.”

Please try to submit before the deadline, since CourseWorks can sometimes become busy and slow. You can re-submit multiple times. (Note: Canvas will save your previous comments, so you don’t need to re-enter your URL if it has not changed, but Canvas 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.”

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