Assignment 1: Island in the Sea

Introduction

This assignment will be the first Unity project you develop on your own, after having completed Assignment 0 & Assignment 0.5. You will be using Unity 2017.3 and C# to design a 3D environment containing objects with which the user can interact using your mobile device’s touchscreen. Your environment should include an island in the sea, a lighthouse on the island, a boat sailing around the island, a bird and a fish circling the boat, and bottles floating on the sea.

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 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) 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 from the sources listed on our IA page, so please do not use any purchased models when doing this assignment. But do make sure to use at least one downloaded model.)

Sea, Island, and Lighthouse

The sea should be a polygon parallel to the world xz-plane. The should be an island protruding from the sea. The island’s geometry can be relatively simple, such a Unity primitive, or more complex if you like. On the island, there should be a lighthouse.

Boat

There should be at least one boat in your environment. The boat(s) should sail on the sea, orbiting the island, and pointing in a direction tangent to the orbit. Your boat(s) should heave, sway, surge, roll, pitch, and yaw according to some function of your choosing that affects at least one of these degrees of freedom; but please make sure that the boat is always partially in the water and never fully submerged. Do not worry about trying to make the boat trajectory be physically correct!

Bottles

There should be multiple bottles distributed on the sea. Each bottle should move (“bob”) up and down relative to the sea (according to some function of your own choosing), but should always remain at least partially immersed.
Figure 1 shows a simplified, schematic, bird’s-eye view of how the environment should look.

In addition to the island, the lighthouse, the boat, and the bottles, there should also be a bird flying around the boat and a fish swimming and jumping around the boat. Figure 2 shows a simplified schematic view of how the bird and the fish should move relative to the boat.

**Bird**
The *bird* should orbit the boat at a radius and height of your choosing (as long as it’s above water). The axis that the bird orbits should pass through and be attached to the boat, perpendicular to the boat’s deck. Thus, it should move with the boat. (While Figure 2 shows a mast, your boat does not need to have one.) We’re not done with the bird, however. Since the bird has gotten a bit confused from all that circling, it should also rotate about a separate axis that passes through its body, parameterized any way you would like.

**Fish**
The *fish* will also orbit an axis that passes through and is attached to the boat. However, the axis about which the fish orbits should point in the direction in which the boat is sailing. Thus, the fish should swim underneath and jump over the boat as the fish traverses its orbit. (Please make the orbital radius vary over time, just enough to make the fish look like it’s not orbiting in a
perfect circle relative to the boat. Even better, can you also make the fish clearly spend less
time in the air than it does in the water by varying its speed?)

Figure 2. A schematic illustration of how the bird and the fish move relative to the boat. The fish is shown offset from
the boat here with a misleadingly tilted orbit. The fish should instead orbit the boat such that the fish passes directly
above and below the boat during its orbit.

Lights
There should be at least two light sources in the scene: One should be the sun (a stationary
directional light) and the other should be near the top of the lighthouse (a spot light that rotates
around a vertical axis). You do not need to make any scene geometry that corresponds to either
light.

Selection
The user should be able to select the boat, lighthouse, and any bottle in the scene using your
mobile device’s touchscreen. (Direct selection using the touchscreen can be achieved using the
Ray object. See the Physics.Raycast and Physics.RaycastAll functions and the ray casting
tutorial.) Note that the sea and island (or any kind of sky that you might choose to have) does
not need to be selectable by the user.

Once an object has been selected, you should change it visually in some way to indicate this. (If
you’d like, you can also indicate selection through audio or vibration, in addition to visually.) For
example, the object’s texture could change to a texture that marks it as selected. Please think
about which approach(es) would be most effective.
Initially, no object should be selected. At most one object should be in the selected state at a time. Touching a different selectable object than the currently selected object should deselect the currently selected object and select the newly touched object. (Note that touching the island or sea should not deselect the currently selected object if the island or sea is not selectable.) You should also provide some way to deselect the currently selected object without selecting another object.

**Control Panels**

Referring to the Unity *UI System*, create a partially transparent control panel (see the *Canvas* documentation and the *Canvas Manual*) for each selectable object except the bottles. 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 a camera control panel to control the camera mode (see the discussion of the Camera below), which should be visible at all times. (If you would, instead, like to have a way to hide the camera control panel, please make sure your app starts with it visible and provides clear documentation on how to make it visible again.)

**Object Actions**

*Boat*

When a *boat* (you might have decided to create more than one) is selected, that boat and its orbiting bird and fish should stop moving, pausing at their current poses until the boat has been deselected.

The control panel for a boat should allow the user to adjust the speed at which the boat orbits the island and any other properties of the boat’s trajectory that you would like the user to control.

If the boat collides with a bottle, the bottle should be destroyed. There should be some visual effect indicating the collision other than simply having the bottle disappear from that point on. (To detect collision, please refer to the Unity documentation for *Collider*, including its messages such as *Collider.OnTriggerEnter*, and the *Colliders as Triggers* video tutorial.)

*Bird and Fish*

When a bird or fish is selected, that object should stop moving, pausing at its current pose relative to the boat until it has been deselected. Note that when a boat is selected (and its bird and fish have paused), the bird and fish should still remain selectable, and if the bird or fish is selected, its boat and the other object orbiting it should be deselected and start moving again.
The control panel for a bird or fish should allow the user to adjust the speed at which that object orbits its boat and any other parameters you would like the user to control.

**Lighthouse**
When the lighthouse is selected, the user should be able to change the speed at which its spot light rotates and any other parameters of the spot light you would like the user to control.

**Bottles**
A bottle is selected when the user touches it. That bottle should follow the user’s finger over the sea as long as the finger stays in contact with the screen. The bottle should be deselected when the user removes their finger from the screen. If the bottle collides with the island or lighthouse, or is moved out of the sea, the bottle should remain in the last location it occupied in the sea before that occurred and should be deselected; if the bottle collides with the boat, it should be destroyed, as described above. In all these cases, no additional selection should be allowed until the finger is lifted from the screen. You are welcome to ignore collisions between the bottle and any bird, fish or other object. Please look again at the Unity documentation for Collider and its messages.

Whenever an object resumes moving after it has been paused, it (and any descendants) should resume from its current position and orientation. That is, it should not make a discontinuous jump in position or orientation.

**Camera**
Your user should be able to use the camera control panel to place the camera in any of the following three modes:

*Scene mode* (the default): The camera should be initially located and oriented such that everything in the scene is within its frustum. No explicit translation or rotation of the camera is necessary. Try to think of how you might have the position, orientation, and/or other attributes of the camera change automatically as the boat(s) move to keep everything within the camera frustum.

*Lighthouse mode*: The user should be able to “attach” the camera to the lighthouse. The camera should rotate to track the boat such that the boat is always in view. If you have more than one boat, please provide a way to change the boat being viewed. The camera and lighthouse spot light are separate and their positions and speeds should not, in general, be the same.

*Boat mode*: The user should be able to “attach” the camera to the boat. In this mode, the user does not need to control the rotation or position of the camera through user input. Instead, the camera should always be looking at the direction in which the boat is sailing (see Transform.LookAt). If you have more than one boat, please provide a way to change the boat to which the camera is attached.
Note that objects that are outside the bounds established by the camera’s near and far clipping planes will *not* be visible; so, please bear this in mind when setting the clipping planes.

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, please 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 its functionality.

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 sea, the island, and the boat to experimentally verify that your approach works, so you can modify it early on if necessary. And please keep your orbits simple at first.

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: 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 enamoured 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?](#)

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 the object with which a ray cast through the screen collided.

Regarding 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). 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!

Regarding Play Mode: While Play Mode is a very useful way of debugging your app, 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. While debugging, you will want to include secondary controls that are guaranteed to work on 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.)

Regarding 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.

Regarding 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 quite differently in how and when they are rendered, relative to opaque objects, 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 nonopaque objects to render incorrectly and inconsistently as their positions change relative to each other and the camera.

**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.txt file with:
  - Your name & UNI
  - Date of submission
  - Computer Platform
  - Mobile Platform & OS version & Device name
  - Project title
○ Project directory overview
○ Special instructions, if any, for deploying app
○ Instructions for using app
○ Missing features
○ Explanation of bugs in your code and Unity
○ Asset sources (cite everything you didn’t create yourself)

- A brief video demonstrating your application’s features. Please 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 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 unlisted 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 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, please check it by downloading it, creating a new project with which to test it, and reading its README.txt file. We will not accept excuses that the wrong file was accidentally uploaded.

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