**OVERVIEW**

In this lab, you will learn how work with the left and right hand controllers. You will learn to get input data from the controllers. You’ll then build a way to target objects in the world. You’ll extend this targeting feature to build a grab mechanic and an interact mechanic.

**IN-CLASS ASSIGNMENT**

* **You must use the PCs in Little 231 for this assignment**
* **This is an in-class assignment that is due by the end of the lab.**
* Setup the Odyssey with two controllers via the HMD Portal
* Launch and run the provided OdysseyInput Unity project, and show the Professor that you can use both controllers.
  + This project along the information provided in the following link(s) will provide you with enough information to programmatically access, control and track each controller.
    - Unity XR Input documentation: <https://docs.unity3d.com/Manual/xr_input.html>
* Setup the left controller.
  + Create a controller prefab consisting of an appropriately sized cylinder with a material applied to it, and a visible location that grabbed objects will attach themselves to.
    - Structure the preFab as follows:  
       ***Controller prefab (Empty Game Object)  
       |--- Cylinder Model  
       |--- Grab Point***
  + Add a script to your controller preFab. Your script should
    - Real-time track, position and orient the controller
      * If the controller clips too early as it gets closer to your face fix it, and make things look more realistic. Clipping is not a problem in itself, as long as the clipping doesn’t look unrealistic. For example, a controller shouldn’t clip one foot away from your face.
      * Output the Vector2 horizontal and vertical joystick values to the console window and demonstrate to the Professor that you have this working properly

**LAB REQUIREMENTS**

**Setup**

* When you finish setting up the left controller, set up and hook up the right controller.
  + You should have both controllers working now.
* Continue with your Unity project from Lab1
* Import the Lab2 Package and use the scene provided in the Package
* Your headset (camera) and two controllers need to be structured so as to be children of an Empty Game Object denoted as Player
* Create the following script files
  + *TargetableObject*
  + *GrabbableObject*
    - Will inherit from *TargetableObject*
    - Will be attached to objects that can be grabbed
  + *InteractableObject*
    - Will inherit from *TargetableObject*
    - Will include a virtual method OnInteract
      * This method will be empty and do nothing
      * It will be overridden in child classes for specific behavior

**The Controller’s Raycast**

* Each controller will need to the ability to perform a raycast
  + By default, this will be performed on each gameplay update
  + But there will be situations where it won’t be needed
* The use the controller’s position for the ray’s origin
* Use the controller’s Forward vector (from its Transform) for the direction of your ray
* The controller’s ray cast should be visible in game and Scene views.
  + Use a line render component to draw this line.
* When a controller’s raycast results with a hit on a *TargetableObject* object, that object is considered to be a “*targeted object”* by the controller.
* If the controller has a targeted object and it is within the grab\interact range, then the object is a “valid targeted object”
  + You can use a single value to determine this range.
  + You can choose to instead implement two separate values for grab and interact objects.
* The controller’s ray cast should change color to indicate different states.
  + These include by not exclusive to:
    - Ray cast does not result in a targeted object
    - Has a targeted object (targeted but not in the grab\interact range)
    - Has a valid targeted object (targeted and in the grab\interact range)

**The Grab Mechanic**

* An object can be grabbed under the following conditions:
  + If the controller does not have a grabbed object already
  + The controller has a valid targeted object that is also a *GrabbableObject*
  + The valid target object is not already grabbed by another controller.
* The grip button will activate the grab on a valid *GrabbableObject*
  + The targeted object will attach itself to the controller’s *Grab Point*
    - Parent the targeted object the Controller
  + The targeted object is considered “Grabbed” by the controller.
  + The controller will report to the targeted object’s *GrabbableObject* script
    - that it currently grabbed
    - a reference to the controller grabbing it
* The Grab remains active while the grip button is held down.
  + The grabbed object will mirror orientation and position of the controller.
  + The controller will not perform a raycast while the grab remains active
* The grabbed object will be released when the grip button is released
  + The grabbed object is detached and no longer moves with the controller.
  + The controller will report to the grabbed object’s *GrabbableObject* script that it is no longer grabbed and nulls out the reference to the controller.
  + The object is now no longer considered “Grabbed” by the controller.
* Helpful Tutorial with some usable ideas:  
  // HOW TO PICK UP AND PLACE / THROW OBJECTS WITH C# - Mini Unity   
  // Tutorial - EASY & FUN
  + <https://www.youtube.com/watch?v=IEV64CLZra8>

**Object Respawn**

* In the provided scene there is a pillar with a box next to it.
  + The pillar is the respawn location
  + The box is the respawn button
* Create a new script *RespawnButton*
  + Will inherit from *InteractableObject*
  + Will override *OnInteract* with respawn functionality
* When a controller’s raycast targets the respawn button
  + Change the material on respawn button object to indicate the controller is targeting it.
  + The controller’s raycast will change color due to targeting an object
* When the Trigger button is pressed on a targeted respawn button
  + check the pillar’s spawn area trigger for an existing *GrabbableObject*.
    - * If one exists: Do nothing.
      * If none exist: Instantiate a new *GrabbableObject*.

**Exit Application**

* Implement a way to exit the game for both situations:
  + A running standalone project
  + from inside the Unity Editor

**Extra Credit**

* Allow for the case when an object that is grabbed by one controller is then grabbed by the other controller will move over to the other controller.
  + Make this look smooth
* Implement a throw mechanic and try to hit the platforms not part of the main platform.
  + Documents and Tutorials on this topic:
    - Vive Minute Tutorials in Unity - Throwing Objects – 6:31 minutes

<https://www.youtube.com/watch?v=ltxoFzXcGQA>

* + - Unity link pertaining to Odyssey controller linear and angular velocities

<https://docs.unity3d.com/ScriptReference/XR.CommonUsages.html>

* Add billboards in the world
  + This is a stand-in for a HUD-based UI…
    - … plus many VR games do this already
  + Example things to show include, but not exclusive to
    - current score
    - time left
    - hits
    - misses
* Make this project into a mini-game
  + Have a game over\pre-game state
  + Have a way to start or reset the mini-game

**SUBMISSION**

Submit to Canvas a zip file with the following

* Remove the old Lab01 standalone (installable) build folder
  + The size of your project will quickly get unwieldly if you don’t remove this folder.
* Structure your submission folder as follows  
   ***You turn in folder  
   |--- Standalone, installable Lab02 build (Only latest version)  
   |--- Unity project ready to build and run from within Unity***
* Remove files that can be rebuilt from the Unity project to reduce the size of the project folder
* Name you zipfile like this **<YourName>\_AG231\_Lab2**