**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
  + Show your instructor that you can use both controllers.
* Open the Odyssey Input Example project
  + This project is designed to show how to work with each
  + More information can be found in the Unity API for XR Input
    - <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 controller prefab as follows:
      * **Controller prefab *(an empty game object)*  
         |--- Controller Model *(the visual model)*  
         |--- Grab Point *(an empty game object)***
* The Controller Script will perform the following
  + Position and orient the controller in real time
  + A debug method that will output the horizontal and vertical joystick values to the console window
  + Demonstrate to your instructor that you have this working

**LAB REQUIREMENTS**

**Setup**

* Add to your controller script support for both the left and right controllers.
  + Implement a boolean into your controller script to differentiate between the left and right controller.
  + This boolean will set variable of XRNode that you will use to get the device information.
    - Your previous examples may have hard coded this previously
  + Hook up the second panel to display input information from the right hand controller
* Continue with your Unity project from Lab1
  + Import the Lab2 Package and use the scene provided in the Package
  + Set up the headset and two controllers as you did previously scene
    - Hint: did you make a prefab of the player?
* Create the following script files
  + *TargetableObject*
    - Does not have any members or methods
  + *GrabbableObject*
    - Inherits from *TargetableObject*
    - Has only two members, but no methods
    - Is attached to objects that can be grabbed
  + *InteractableObject*
    - Inherits from *TargetableObject*
    - Does not have any members
    - Only has a virtual method *OnInteract*
      * This method will be empty and do nothing
      * It will be overridden in child classes for specific behavior
  + RespawnButton
    - Inherits from *InteractableObject*
    - *OnInteract* override will have respawn functionality
  + *ExitApplicationButton*
    - Inherit from *InteractableObject*
    - *OnInteract* override will have exit application functionality

**OBJECT HEIRARCHY**

*The following chart shows the object inheritance*

*Classes that are in BOLD with an ‘\*’ are scripts that will be attached to game objects*

|  |  |  |
| --- | --- | --- |
|  | *MonoBehavior*  **|---** TargetableObject **|--- GrabbableObject \*   |---** InteractableObject **|--- RespawnButton \*   |--- ExitApplicationButton \*** |  |

**Targeting Objects**

* 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
* When a controller’s raycast results with a hit on a object with a script of the *TargetableObject type*, 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”
  + Store a reference to this object to a member of the controller
  + Grab\Interact range can be implanted in multiple ways
    - single value for both Grab & Interact
    - separate values for Grab & Interact
* The controller’s ray cast should be visible in game and Scene views.
  + Use a line render component to draw this line.
  + Set the color of this line based on 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 type of *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 is currently grabbed *(hint: use a bool)*
    - A reference to the controller grabbing it *(hint: pass ‘this’)*
* 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 an object is grabbed
* 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:
  + <https://www.youtube.com/watch?v=IEV64CLZra8>

**Object Interaction**

* When the Trigger button is pressed, the controller checks if it has a valid targeted object of the type of *InteractableObject*
* If the controller has an *InteractableObject*, then it will call OnInteract on the object.

**Respawn Button**

* In the provided scene there is a pillar with a box next to it.
  + This consists of a few different objects.
  + There is a model by a pillar that user will target.
    - This object is the respawn button
  + A trigger above the pillar is the spawn area for new objects
    - If you are missing this trigger, add one
* When OnInteract is called:
  + check the pillar’s spawn area for an existing *GrabbableObject*.
    - * If one exists: Do nothing.
      * If none exist: Instantiate a new *GrabbableObject*

**Exit Application**

* Create a new object to be interacted with to exit the application
  + You can reskin the Respawn Button Model for your purpose
* When OnInteract is called:
  + Exit the application
  + Handle both of these cases
    - Running standalone project
    - From inside the Unity Editor

**Extra Credit**

* Add virtual methods for OnTarget\OnUntarget in *TargetableObject.* 
  + Call theses methods from the controller as appropriately
  + Extend the Spawn Button to switch textures based on Targeting
* 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.
* Implement a throw mechanic and try to hit the platforms not part of the main platform.
  + Documents and Tutorials on this topic:
    - <https://www.youtube.com/watch?v=ltxoFzXcGQA>
    - <https://docs.unity3d.com/ScriptReference/XR.CommonUsages.html>
* Use billboards in the world to display HUD-like information
  + 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**