Assignment 3: Go Fish!

Kevin Li
(UNI: kl3285)
COMS 4172
Professor Steven Feiner
March 28, 2024

Introduction

This report explains how users interact with the applications developed for Assignment 3, specifically: 1) A mobile AR pond builder, and 2) a headset-based AR fishing game. The two applications are intended to work together, wherein users craft a virtual environment on their phones and then immerse themselves in the fishing game using a VR headset (i.e., Meta Quest 2). The document is organized into two primary sections, each detailing the functionalities and design philosophies underpinning the mobile and headset applications.

Part One: Phone-Based AR Fishing Pond Builder*

Upon opening the application, the user is greeted with the main interface, consisting of interactable buttons for 'Reset,' 'Delete Selection,' 'Undo,' and 'Redo' functionalities. Concurrently, the phone's primary camera activates, signaling to the user the need to engage with their surrounding space as part of the application experience (Figure 1).

In addition to the UI elements mentioned above, the interfaces feature a dropdown menu positioned at the bottom of the screen. By default, the pond option is preselected, guiding the users to place a pond first, which is required by the application and provides a more natural workflow. Adjacent to the dropdown is a vertical slider, strategically placed on the left side of the interface. This slider can be used to adjust the pond's size prior to its placement in the scene (Figure 1). The choice of a vertical orientation of the slider is deliberate, aiming to intuitively reflect the action of increasing or decreasing size in a manner that is natural. This design decision is rooted in Jakob Nielsen's second usability heuristic, 'Match Between the System and Real World,' enhancing the user's intuitive control over the functionality. Additionally, this slider disappears upon selecting other

spawnable objects in the dropdown, indicating to the user that the slider is only applicable to the pond.

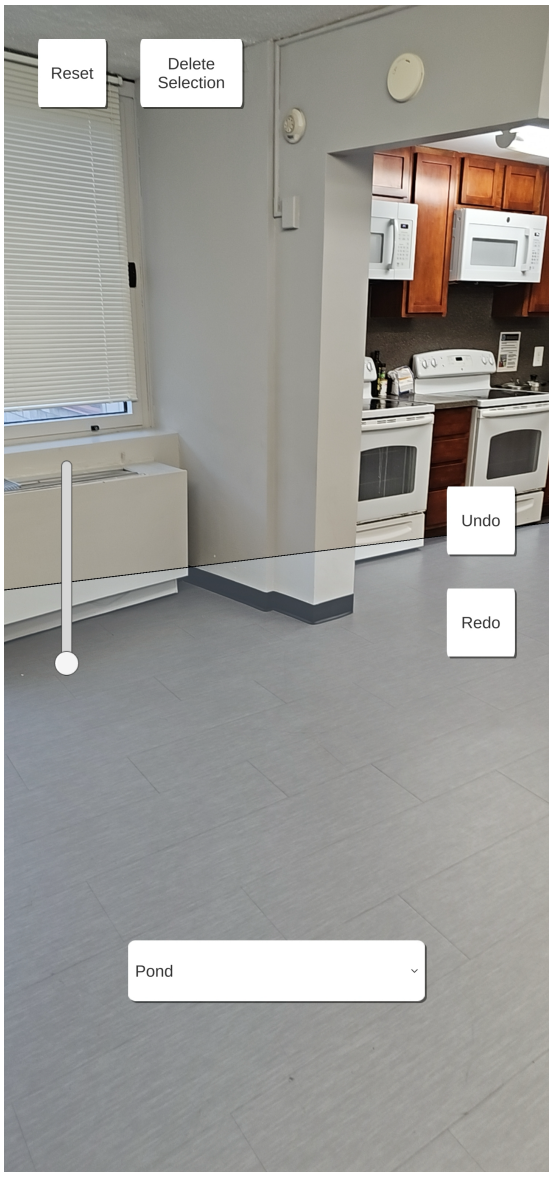

Figure 1: Application open on mobile device rendering UI elements and detected plane (transparent white material with black outline).

To begin placing objects in the virtual space, users should move the mobile camera around in their physical environment for the AR Plane Manager to generate planes for placeable AR objects. The detected planes are visually represented with a transparent white material and a distinct black outline, making it evident to users where AR objects can be positioned without obscuring the view of the real world. Moreover, the plane manager works in real time, alerting the user of any newly detected planes or extended surface area, effectively addressing Nielsen's first heuristic, 'Visibility of System Status.'

To place an object, the user can tap on a point on the detect plane. If the object can be placed, it will be rendered. If not, i.e., the user is attempting to spawn a second pond or spawn fish outside of the pool, the attempt will be rejected. The object will be initially spawned but quickly deleted, indicating to the user that the system did register the tap but that the action is invalid. This again, conforms with Nielsen's first heuristic. Note that the system requires users to first place a pond and then a boat to prevent the likelihood of erroneous manipulations.

All objects other than the pond, boat, and reference objects (again, to prevent possible invalid operations) support selection, manipulation, and deletion. This system was implemented using Unity's XR Interaction toolkit. Spawned AR objects can be selected by simply tapping on them. Deselection can be accomplished by tapping on the object again or tapping elsewhere in the scene (Figure 2). Selected objects are rendered in an orange highlight and can be manipulated in various ways. To move an object, a user drags it across the screen with a single finger. Rotational adjustments are made through a two-finger circular motion. Additionally, selected objects can be removed from the scene via the 'Delete Selection' button. It's worth nothing that during the

selection phase, objects can temporarily overlap or move beyond their designated areas, such as placing a pond object outside its bounds. However, this is only allowed while the object is selected. If the manipulation is not allowed, the object will revert to its original position and rotation upon deselection. This system is underpinned by a Command Pattern-based architecture, ensuring that all interactions are reversible, which is further explained below.

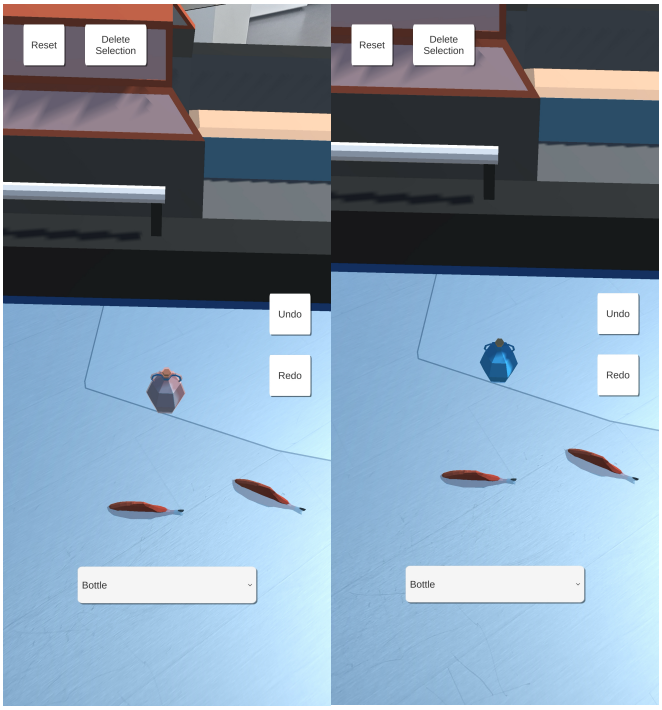

Figure 2: Bottle selected on the left and deselected on the right after tapping elsewhere on the screen.

The application has a history system, incorporating Undo and Redo functionalities, built using the Command Pattern design paradigm. This approach ensures that every object manipulation action (spawn, delete, translate, rotate) is encapsulated within commands that adhere to an 'ICommand' interface class mandating Undo() and Redo() methods. A global redo and undo stack are also used to manage user actions. Implementation details can be found in the submitted code.

To enhance user experience by minimizing the risk of unintended scene resets, the application has a fail-safe mechanism. Upon initiating a scene reset through the ‘Reset’ button, a secondary confirmation window appears, presenting a cautionary message regarding the irreversible loss of data should the reset proceed (Figure 3). This feature is a direct reflection of Nielsen’s fifth usability heuristic, ‘Error Prevention.’

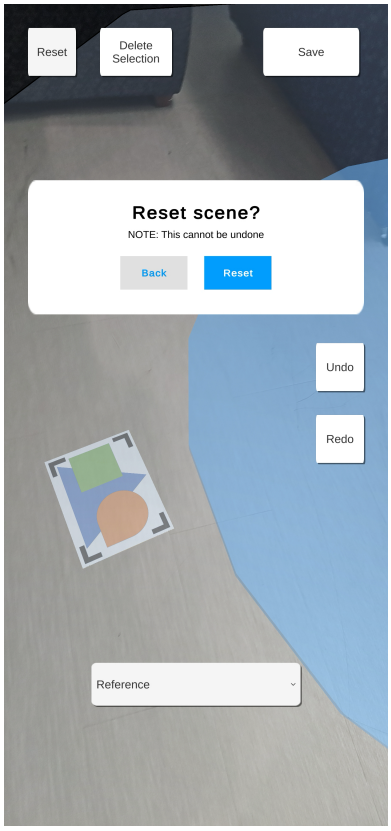

Figure 3: Screenshot of application showing the process of resetting the scene.

To save a scene, the user must have placed a ‘Reference’ object in the scene. A ‘Save’ button only appears when a ‘Reference’ object is present in the scene. Otherwise, the button is inactive, effectively informing the user of the need of a reference object. The created scene information is saved as a .csv file and can subsequently be transferred to a Meta Quest 2 headset to set up Part

Two. A window, detailing the file path for the user to easily find the file on their phone, pop ups after the file has been successfully saved (Figure 4). This features not only streamlines the process of transferring the scene to a VR headset but also aligns with the principles of keeping users informed.

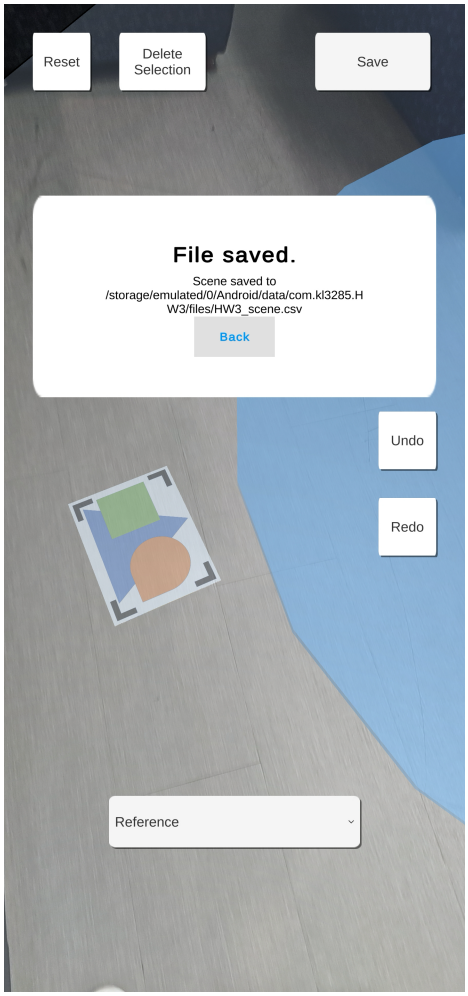

Figure 4: Screenshot of application after saving the scene as a .csv file with a window detailing file path.

**NOTE: All required features for Part One have been implemented.*

Part Two: Headset-Based AR Fishing Game

After saving the scene as a .csv file using the mobile application from Part One, the file can be manually transferred to a USB-connected-computer using Android File Transfer. Following the transferal, the .csv file (named 'HW3_scene) can be imported into the 'Assets/Resources' folder within the Unity project. Additional example scenes (i.e., 'HW3_scene_example') used for testing can also be found in the same directory.

After the .csv file is transferred, the VR application can be deployed to a Meta Quest device. Upon opening the application, passthrough is automatically enabled. The user is constantly informed by viewport-mounted textual instructions, ensuring that important information is always readily accessible. The initial instructions guide the user to align the virtual reference on their left controller with the marker in the physical environment. Following successful placement of the virtual marker, the user pressed on the 'A' button to load in the entire scene. The clearly displayed instructions align with Nielsen's tenth heuristic by providing clear and easily accessible instructions (Figure 5).

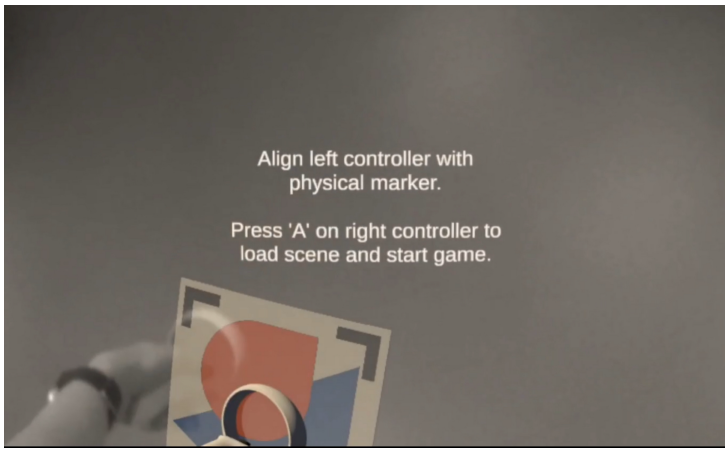

Figure 5: Initialization scene of application with clear instructions and virtual reference object following left controller.

Following the successful placement of the scene, the player should walk to the boat and start playing the game. The score is now displayed and constantly updated during gameplay. To further guide the player, additional instructions are provided, indicating the need to prepare the fishing rod with bait (Figure 6). The user can interact with the fishing rod by positioning either controller close to it and pressing the grip trigger button. Both the fishing rod and the baits also change color upon interaction, either when a controller is nearby or when they are successfully grabbed (Figure 7). This visual feedback is rooted in Nielsen's first usability heuristic and keeps users informed of system status.

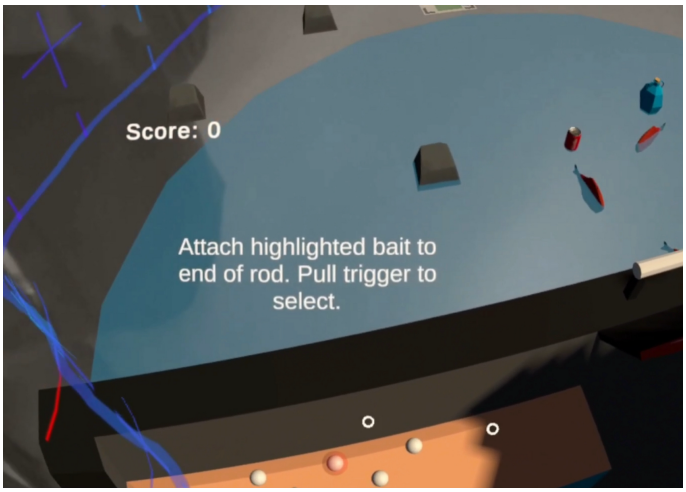

Figure 6: Instructions and score displayed to the player after scene load.

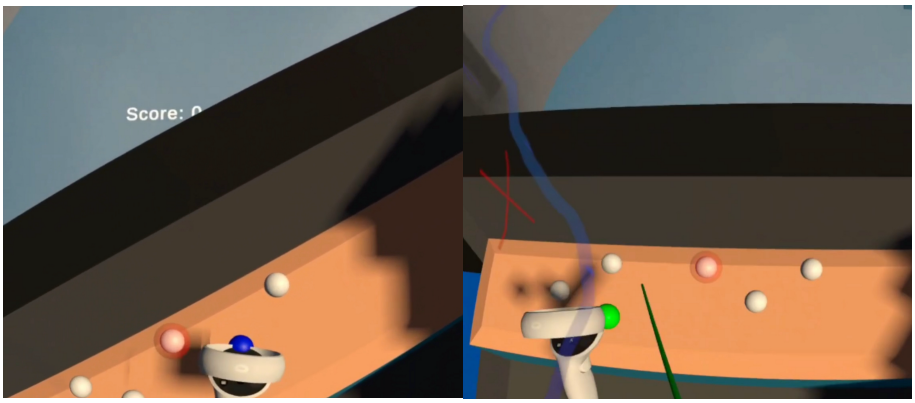

Figure 7: (Left) Bait is rendered in blue when in contact with controller and (Right) rendered in green when being grabbed.

To bait a rod, the user can either use a virtual hand or a virtual pointer. For the first option, the user can move a controller to a bait, select it by holding the grip button, and drag it to the tip of the rod, For the latter option, the user can point the controller ray to a bait from a distance, select it using the grip trigger button, and then drag it to the tip of the rod. Only the highlighted bait can be successfully baited; all other baits will not snap to the rod. The bait must also touch the tip of the rod for the rod to be baited.

Once the rod has been baited, new instructions are displayed, informing the user to cast bait by pressing ‘A’ on the right controller. To introduce an element of randomness, the distance that the bait is cast is proportional to the hold time of the button, i.e., if the player holds ‘A’ for a longer time, the bait will be shot out by a greater distance (Figure 8).

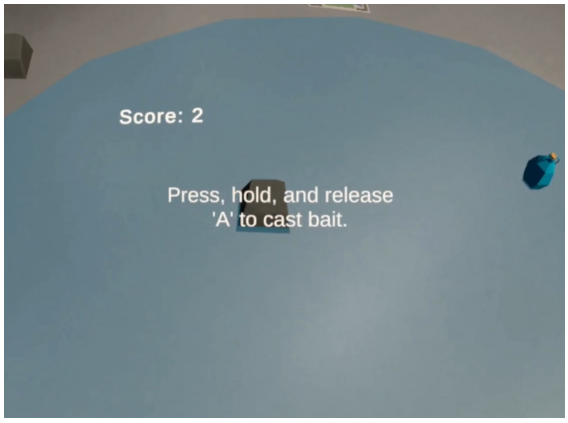

Figure 8: Instructions displayed after attaching bait to rod informing player to cast bait by pressing, holding, and releasing the ‘A’ button.

After the bait has been casted, the instructions to cast bait are no longer displayed. The player can move the controller and the rod will move accordingly, with the distance between the rod and the bait fixed (unless the bait hits external objects). If a fish or environment object enters the capture

range of the casted bait, the player will be alerted by a new displayed message, prompting them to reel with the ‘A’ button, following which the player can reel or move the rod away. Caught fish will disappear from the scene and increment the player’s score by 3. Caught environment objects will stay put and decrement the player’s score by 1.

Even if no objects enter the bait’s capture range, the player can still reel using the same ‘A’ button. If nothing was caught, the player’s score is not affected and they can repeat the process of baiting the rod, casting a bait, and reeling in. The fishing process is repeated until all fish in the scene have been caught, after which the game is over and the user can easily restart the game by pressing ‘A,’ a feature designed with Nielsen’s seventh heuristic (Flexibility and Efficiency of Use) in mind (Figure 9).

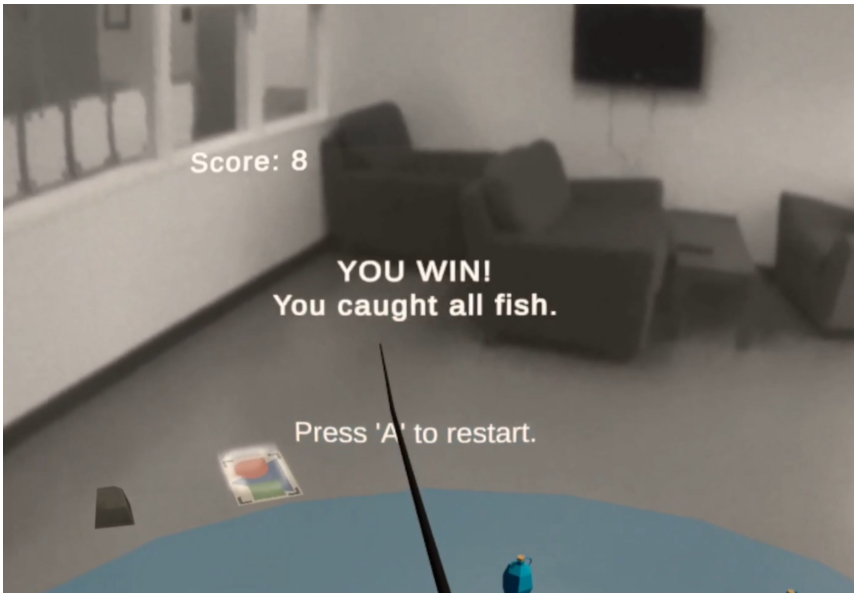

Figure 9: Game over text and restart instructions.

Simplifying the control scheme is important to enhancing user experience. As detailed above, ‘A’ is the primary input for the majority of gameplay interactions—ranging from scene loading to casting bait, reeling in fish, and executing game resets—players are provided with a streamlined and intuitive gameplay experience. This decision reduces cognitive load on users, sparing them from the need to memorize complex button layouts for different actions.

**NOTE: All required features for Part Two have been implemented.*