# Interactive Design for Escape Room in AR - Team 7

*Gabby Kang (gkang9), Keyi Lu (klu20), Esther Wang (wwang177)*
*Dec 2023*

# 1. Introduction and Background

An escape room is a real-life adventure game where participants work together to decipher riddles and puzzles based on provided clues. Typically, this activity revolves around a storyline, and participants find themselves confined to a physical space until they successfully solve all the challenges within a set timeframe. Implementations of escape rooms experience limitations in flexibility due to prop or casting requirements. On the other hand, virtual reality based escape rooms can lack the "physicality" of moving around a room and investigating objects due to lack of environment realism and haptic feedback. Augmented reality (AR) has the potential to outshine both real-world and virtual-world escape room alternatives by combining exploration physicality with prop flexibility.

Our project explores possible methods of implementing AR in escape rooms by specifically focusing on different types of player interactions with the real and virtual space. We additionally leverage the use of handheld devices (which ought to be considered ubiquitous) as a window into the virtual space, allowing us to visualize virtual content. In particular, we primarily use marker-based AR to implement dynamic visualization, player-modified environments, and player perspective manipulation. We additionally attempted to implement the use of the Transmission Control Protocol (TCP) for multiplayer collaboration; however, due to limitations in time, we were unable to fully integrate our multiplayer communication into our player interactions.

The following report describes the motivation for development of each player interaction, as well as methodology and the end use case (ie: given greater development time and asset integration). We then describe the utilization of each action in our demo, as well as the corresponding scripts/game objects that were required to implement the player interaction. Fully documented code and demo can be found in the github repo.

**Repository Link:** https://github.com/Micky1001/IntroAR/tree/main

# 2. Core Features Methodology

The following section describes motivation, methodology, and end use case scenario (when applicable) for each interaction.

## 2.1 Fundamentals - Marker Detection and Virtual Touch

We use Vuforia-based Marker detection to provide AR functionality by presenting a reference feature to display virtual objects. In a fully-developed escape room, these markers can be seamlessly integrated into the environment; for example, specific wallpaper patterning, image representations of 3D scenes, or a 3D multi target could be used as markers and be more easily hidden in the environment.

Additionally, a fundamental requirement to interact with any virtual environment is to create an approximation of virtual touch. In our case, we deploy for handheld devices and use the touchscreen as the approximation of our virtual touch. Virtual objects in a 3D augmented reality space can be projected into a 2D pixel coordinate using the Camera.WorldToScreenPoint approximation. Alternatively, we can directly convert the 2D coordinates of the touchscreen touch into a ray using Camera.ScreenPointToRay, and we can use standard physics raycasting to check for interactions with game object colliders.

## 2.2 Dynamic Visualization

Virtual objects can be instantaneously loaded and unloaded leading to dynamic visualization of the same space. In Unity, the GameObject.setActive() method toggles any GameObject between loaded and unloaded states, and an external boolean can be used as a control for the state toggle. In particular, joint loading and unloading of groups of game objects can lead to delivery of hint information. In an end use case scenario, dynamic visualization can be expanded to completely change the environment based on theming, storyline, or progression; for example, we can implement a UI button to act as a visualization toggle between past, present, and future escape room states—leading to alternate perspectives of the same room. We can also take advantage of the flexibility of virtual objects to reuse or recontextualize the same marker as the game state advances by loading different types of virtual content on the marker.

## 2.3 Player-Modified Environments

Virtual objects are flexible and are not bound by traditional physics. Additionally, we can force the player's virtual touch to interact with the virtual object in a pre-defined, non-realistic way. Combining these two traits, the game environment can easily be updated via player input. For example, players can drag a virtual object to a different location by using any screen point to world point conversion in conjunction with the virtual touch implementation discussed in 2.1. Alternatively, players can utilize dynamic visualization (2.2) to add or remove objects from the virtual world. The virtual world thereby becomes an extremely malleable, creative environment that goes far beyond what is currently achievable in pure reality escape rooms. Our end goal for

player modified environments is to play with the extent of the boundaries by allowing the player to instantiate and move unrealistic objects (ie: large pillars, "organic" virtual creatures, etc) or place objects in unrealistic positions (ie: suspending objects in air, placing objects "outside" the boundary of the room, etc).

### 2.4 Player-Perspective Manipulation

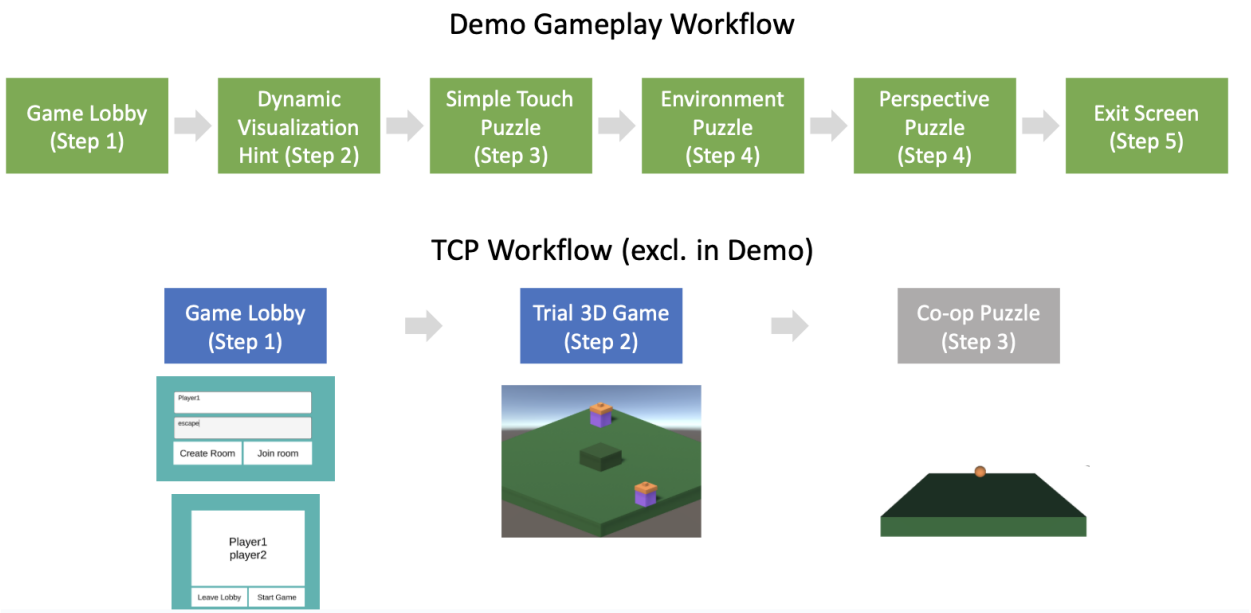
Physical interactions are classic during the conventional escape room experience. Therefore, some kind of physical interaction requirement would be ideal to combine with our virtual mobile interactions. The idea we came up with is to ask the player to move and rotate the camera to align the virtual objects on the screen.

### 2.5 Transmission Control Protocol (TCP)

A key component of escape rooms is the collaborative effort to escape. TCP is a transport layer protocol in computer networking that establishes and maintains connection among multiple devices on a network. PUN is a high level abstraction and an API service that streamlines the multiplayer networking environment in Unity. In our project specifically, we adopted the client-server model that involves a host (master client) and subordinate clients. The flow usually involves the master client creating a “game room” for other clients to join; only the master client can start the game depending on the setup. During the gameplay, all clients’ inputs and updates are sent to master client to process, and broadcasts the updated game state to all clients.

# 3. Gameflow Structure

The following section discusses a summary how each of the possible interactions is implemented in our demo and at which step they are implemented at.




## 3.1 Fundamentals - Marker Detection and Virtual Touch

A set of three buttons of different colors (red, green, blue) are displayed on a specific detected marker. Players must touch the buttons in a specific sequence via the touchscreen to complete the puzzle. Touches are only registered when the player touches the round, colored portion of the button; touching the button base will not register as a touch and will not register as an entry in the sequence. Our demo implements the simple raycasting conversion discussed in 2.1 to register collisions, but our button prefab is structured such that the collider surrounds the colored part of the button (ie: the top and not the base of the button). We additionally keep track of a sequence index variable to ensure correct successive order of the sequence.

## 3.2 Dynamic Visualization

Our demo uses dynamic visualization to provide the hint sequence for the button puzzle described in 3.1. Using a UI button, different state IDs are toggled in increasing sequential order, ranging from 0 to 3. Once the state ID == 4, the state ID is reset to 0. At each different state, different information is visualized; state 0 loads nothing, state 1 loads 3 red cubes, state 2 loads 2 green cubes, and state 3 loads 1 blue cube (corresponding to a sequence of (1, 2, 3) = (blue, green, red). Previously loaded information is unloaded when the next state is toggled as per Unity's built in GameObject.SetActive() method.

### 3.3 Player-Modified Environments

We developed a simple Tic Tac Toe-esque environment to demonstrate player-modified environments. Players are requested to "Tie" a game of tic tac toe by adding more components to the 3x3 board which already has 5 spots filled in.. This is the most complicated part of our demo and consists of three major parts: (i) a draggable inventory panel, (ii) instantiation of a game object at the same location as the draggable panel, and (iii) game object property specification. Our demo uses an inventory with two objects, a sphere and a cross, and the inventory is enabled/disabled with dynamic visualization depending on game progression. Each object is associated with its own Canvas UI Panel. The UI panels are defined by a specific draggable functionality, such that when we drag our finger across the screen we move the panel as well. Upon release of the finger, the panel is reset to its original position; we then use the raytracing for screen-to-world coordinates (2.1) to project the 2D touch screen coordinate into the virtual world.

Upon collision, the object that corresponds to that panel will be instantiated at the raytraced location. Each of the two objects are rigidbody, gravity affected, and draggable objects, but they have different functionality. We additionally categorize object properties by providing their own property script (either XCollision or SphereCollision), which checks for collisions with individual Tic Tac Toe grid entries. Each Tic Tac Toe grid entry is tagged with a relationship to "X" or "O" to indicate the correct answer; when a draggable game object collides with any of the objects, we check if the game object matches the grid and update the game state accordingly.

### 3.4 Player-Perspective Manipulation

We designed a perspective puzzle that requires the player to physically maneuver their AR camera in the virtual world. A multi-target is used to optimize the positioning of the spheres. The spheres are placed on a virtual plane, which is tilted with respect to the up surface of the multi-target. The program will access all spheres' screen positions in real-time. The player needs to change the position and the angle of the camera to align all the spheres within a certain range to complete the task.

### 3.5 Transmission Control Protocol (TCP)

We were able to successfully implement synchronization across multiple players in the same game room (TCP step 1), spawn players depending on the number of clients, and for players to interact with each other in a 3D game scene in Unity (TCP step 2). However, there were some technical challenges when we implemented the multiplayer co-op game (TCP step 3) that involves two players working together to shoot a target (one controls position, while the other one controls force magnitude). There were some issues with Vuforia AR camera as we moved on from the gameroom lobby to gameplay and touchscreen features that could be addressed in the future.

### 3.6 Demo Specific Backend Management

The players are led to a main menu when they start the game, where they have two options - Escape(play) or Quit. The main gameplay scene will load after the “play button” is tapped. The timer script will start to count the time used during the gameplay as soon as the main scene is loaded. Also, the timer will keep track of the puzzle completion, because we want to stop to count the time when the player has finished all the puzzles. After the gameplay is finished, we would like an ending menu where the player can either choose to quit or replay the game if they’d like to. Also, a display of the time of the last gameplay so they can see if this time they would finish the puzzles faster.

# 4. Example Scripts

### 4.1 Fundamentals - Marker Detection and Virtual Touch

***Game Object Structure***

- Vuforia Image Target
    - Button Prefab - Red
    - Button Prefab - Green
    - Button Prefab - Blue
- Button Manager

***GKButtonManagerScript.cs***

- Controls the Virtual Touch and sequencing interactions for the three buttons used in the three button set puzzle.

### 4.2 Dynamic Visualization

***Game Object Structure***

- Vuforia Image Target
    - Red Objects
        - Red Cube x 3
    - Blue Objects
        - Blue Cube x 2
    - Green Object
        - Green Cube x 1
- Filter UI Button Manager

***GKUIButtonScript.cs***

- Controls the dynamic visualization of red, green, and blue objects sets upon triggers from a UI button.

### 4.3 Player-Modified Environments

***Game Object Structure***

- Vuforia Image Target
    - Tic Tac Toe Grid
        - Tic Tac Toe Individual Grid Entry PreFab x9
        - Tic Tac Toe Default Sphere Entries x 4
        - Tic Tac Toe Default Cross Entry x1
    - Draggable Sphere
    - Draggable Cross
    - Plane
- Inventory Canvas Group
    - Sphere Inventory Panel
    - Cross Inventory Panel
- Tic Tac Toe Manager

***GKInventoryManager.cs***

- Loads or unloads the individual inventory panels when the inventory is toggled. The script additionally accounts for game progression, restricting total visualization until the first puzzle is completed.

***GKInventoryPanel.cs***

- Provides functionality for the draggable UI panel, including the instantiation of the associated game object upon release of the UI panel

***GKDraggable.cs***

- Provides functionality for the draggable 3D game object, placing the 3D game object at the ray traced corresponding world coordinate for a given touch.

***GKSphereCollision.cs***

- Identifies correct placement of Sphere objects in the Tic Tac Toe grid by looking at collision tags and updating booleans accordingly.

***XCollision.cs***

- Identifies correct placement of Cross objects in the Tic Tac Toe grid by looking at collision tags and updating booleans accordingly.

## 4.4 Player-Perspective Manipulation

***Game Object Structure***

- Vuforia Multi Target

***KLPuzzleManager.cs***

- This script read the positions of the spheres from the perspective puzzle. This also determines every frame if all spheres are aligned .

## 4.5 Transmission Control Protocol (TCP)

***GameManager.cs***

- This script ties together the key functionality, including get the action maker's player ID, spawning player and connection to different scene

***NetworkManager.cs***

- This script implements the core functionality of all networking related events, including creating, joining and leaving rooms

***ShootingSlider.cs***

- This script implements two self-oscillating slider bars that allows player to control the shooting position and force magnitude in Vuforia with AR camera, without implementing TCP functionalities yet

### 4.6 Demo Specific Backend Management

***Timer.cs***

- This script keeps track of the time the player has used in the current gameplay and displays the time counting throughout the game. Also, this script call the next scene(ending) and the timer stops when all the puzzles are finished.

***MainMenu.cs***

- Provides functions for buttons on the menus.

***EndSceneManager.cs***

- Display the time used in the last gameplay on the ending screen

***Activator.cs***

- Reads the booleans from GKBoolean to activate the puzzles when the previous ones are finished.

## 5. Appendix and Reference


- Trade offs of different unity networking solutions
- Unity Realtime Multiplayer, Part 2: TCP, UDP, WebSocket Protocols
- Photon PUN