Concept

Genre: Minigame Collection

Core Gameplay

- The player is given a set of simple minigames to choose from and play.
- Each game would have its own rules, controls, and other properties depending on what would be best suited for the individual minigame.
- There should be a main hub where the game options are presented.
- The score for each game should be viewable both from the hub area and within the game. This way, a player knows their score, and will feel motivated to try and beat it.

List of Potential Minigames

- Mini-golf
- Claw machine
- **Shooting Gallery**
- Ski-ball
- Rhythm Game
- **Whack-a-mole**

First Minutes of Gameplay

- The player spawns into a hub world with several minigames to choose from.

- The player is asked to enter a name that their scores will be listed under.
- Ideally, each game would have a ‘carnie’ telling you to play their game. This would add character to the game.
- The player would choose a game to play and be entered into that game.
- After the player has completed their game, they would be given a score and returned to the hub world to start again.

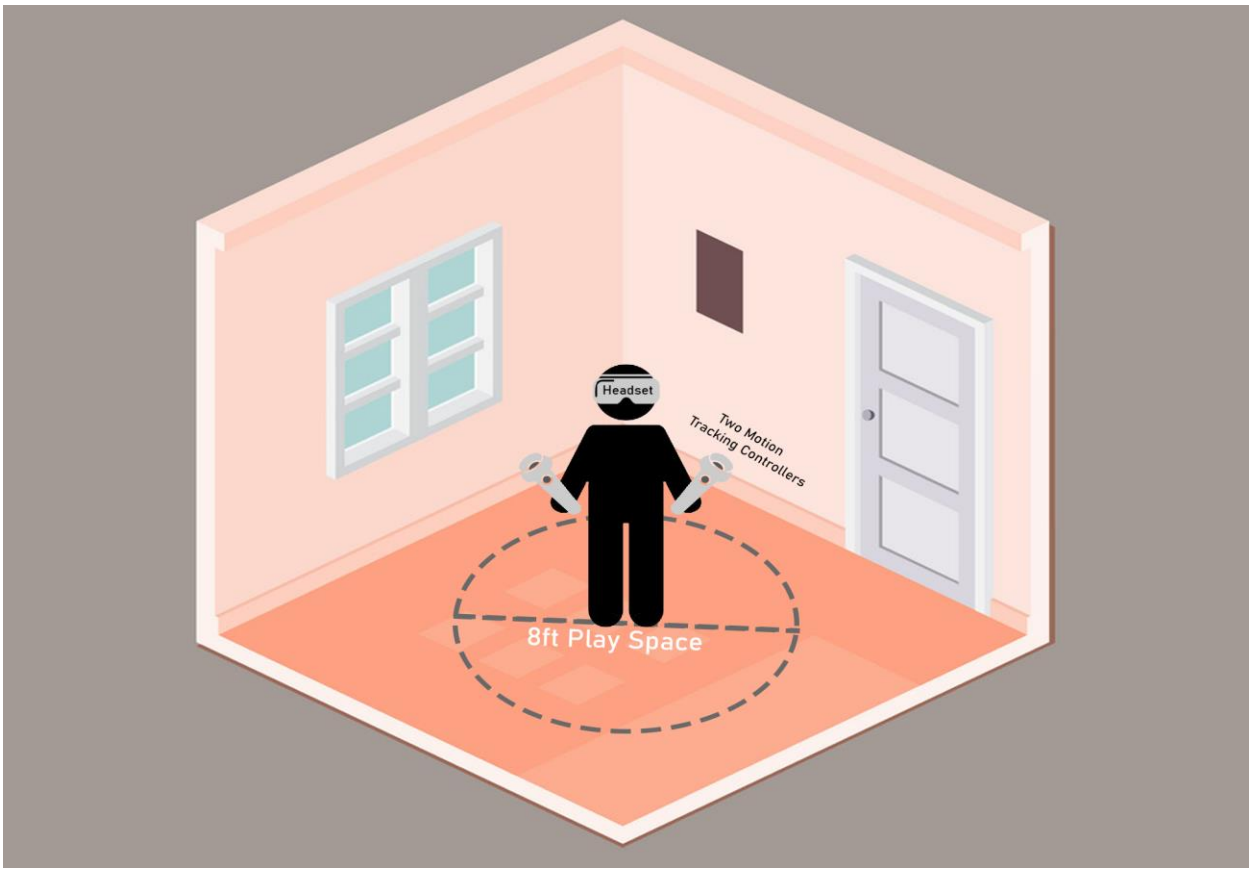
Game Requirements

Minimum

- A hub area with game selection
- At least one playable minigame (more can be added later)
- The ability to track several of the highest scores for each minigame (with persistence)
- Models of the controllers or simulated hands in game
- Live score tracking in game via a toggle menu
- Music and SFX

System Requirements

- The player will need a VR headset.
- Any commercially available headset should work, but the game will be designed with the Meta Quest 2 in mind.
- The player will need one controller for each hand.
- Play space should be at least large enough to comfortably extend arms fully in both directions. (i.e. at least 8ft diameter)
- The play space does not need to be larger than this because the player will not need to move, only to turn and swing their arms. However, any extra space is nice.



Models

- Vr Controllers
    - Purpose: Vr Controllers will represent the player's hands as they play.
    - Status: Not Collected: I'll spend more time looking once I start working on VR integration into the project.
- Ring Target
    - Purpose: These will be launched during the shooting gallery minigame for the player to shoot out of the air.
    - Status: Collected: I personally created this model using blender.
- Standing Target
    - Purpose: These will stand up at certain intervals for the player to shoot.
    - Status: Collected: I personally created this model using blender.
- Gun
    - Purpose: The player will shoot this during the shooting gallery minigame.
    - Status: Collected: Sourced from the unity asset store.
- Misc Environmental Props
    - Purpose: These are things like rocks and trees that will be used to decorate the environment so the player is not on a flat plane.
    - Status: Collected: Sourced from the Toon Fantasy Nature pack.
- Hub Building
    - Purpose: The hub will be indoors and will need various decorative assets and textures so that it will be appealing to the player.

    - Status: Not Collected: I have some ideas and leads, but have yet to decide what the final hub should look like.
- Character Sprites
    - Purpose: Characters will be placed at certain locations in the game to give tutorials give the game a more fun atmosphere.
    - Status: In Progress: Some sprites have been made, but many of them will be done at a later date.

Technical Writeup and Challenges

Shooting Gallery Minigame Description

In the shooting gallery minigame, the player will be transported to a scene and presented with several targets to shoot at. Targets can be of two types:

- Standing Targets
    - Standing targets start out by laying flat on the ground attached to a post. When activated, they will stand up straight and wait until they are shot or until some amount of time passes. If an active standing target is shot, it will turn green, fall to the ground, and grand points to the player. If some time passes and the target is not shot, it will turn red, fall to the ground, and be marked as a miss.
- Launched Targets
    - Launched targets are simple. They are launched through the air for the player to shoot. When shot, the target will turn green and grand points to the player. If the target falls below a certain height, it is destroyed to prevent many targets from building up and falling forever. When the target is destroyed, it is marked as a miss if it has not been shot.

The core of the game logic essentially revolves around providing the player with targets to shoot. Right now, this is done randomly, so a random target will be activated or launched every 1 – 3 seconds. However, there will likely be some amount of scripting done in the future to provide a pre-planned sequence of targets during key moments of the game, such as the final seconds.

Game Management

Right now, game management is split into three manager scripts that communicate with each other:

- GameManager
    - This is the typical GameManager Component that is present in most games. It's role is to listen for global actions like quitting the game, and to coordinate the other managers.
- ScoreManager
    - The ScoreManager class handles everything related to score tracking and persistence. I thought it best to make this it's own class instead of including it's functionality in GameManager because it would keep the code better organized and would help to maintain a healthy division of responsibility in my project. I'll go into more detail about the ScoreManager class in a later section of this documentation.
- MinigameManager
    - This is a script / Component that is used to control the game logic of a particular minigame. For example, the ShootingGalleryMinigameManager Component exists to coordinate the targets in the shooting gallery minigame. It collects all of the targets in the scene and tells them when to activate.
    - One of the ideas of the Minigame Manager is that it only exists for as long as the minigame is being played. When a minigame is started, the GameManager creates a MinigameManager Component. When a minigame is over, the GameManager destroys this Component.

- I designed this component the way I did so that it would be as modular as possible. Even though I will likely only make one minigame during this class, a minigame gallery is no gallery with only one minigame. For this reason, I want to show that I am thinking about how to make this project in such a way that it could be expanded upon in the future.

Score Tracking

One major challenge of this project was tracking score. Each of the targets knows if it has been hit or missed, and the ScoreManager knows what to do when a target is hit or missed, but coordinating these together was more complicated than I would have expected.

Initially, I thought to use the unity event system. This would essentially involve each target having an event that the ScoreManager could subscribe to in order to find out when a target was hit or missed. However, this had several issues. First, ScoreManager would have to subscribe to every event, which is difficult to do because it doesn't have a list of targets stored like the MinigameManager does. Second, the launched targets are created at the time of their launch, so a list of targets would have to be dynamically updated as targets are created and destroyed.

My second idea was for the MinigameManager to read a boolean that was already stored in each target's script. I thought this would be easier because the MinigameManager already has a list of all the targets. However, this functionally ended up being worse than the first idea because the list was still not dynamically updated, and now the MinigameManager would have to communicate to the ScoreManager by going through the GameManager.

Finally, I remembered using the SendMessage functionality in my CS485 project and decided to use that. Essentially, each target would store a reference to the GameManager that they could send a message to when hit. The downside of this approach is that now the targets are communicating with the GameManager instead of the ScoreManager, but I decided to go through with it anyway.

Score Management and Persistence

Now that the ScoreManager was being informed of target hits and misses it needed to do two things, calculate a score, and save it. Calculating the score was easy enough: just create some variables and update their values when necessary.

However, persistence was a bit of a challenge. This was primarily because I had no idea how to go about it. In general, I understood the concept of persistence, and I at least knew that the scores would need to be saved to a file. I knew that simply writing unlabeled information to a file in a specific order would probably work, but would be a lot of work, and very prone to error. Fortunately, I'm taking CS482 (Machine Learning) this term, and we primarily use Python in that class, so the first thing that came to my mind was the data dictionary. I thought that storing the score data as a set of key-value pairs would make it easy to access specific values without much complex logic. A few short google searches later, and I learned that the Unity Engine has native support for reading and writing JSON files.

At this point I split up my internal variables so that the persistent values would be a part of a data structure, and the temporary values would be simple private fields. This way, I could just serialize the data structure anytime I wanted to save data, and deserialize it into a new instance of the structure whenever I wanted to load data.

Next Steps

- VR Integration
    - Currently, the game is not playable. One of the top priorities going forward needs to be integrating VR into the project so that it can become playable.
    - To integrate VR into the game, I will need to get models of VR controllers to represent the hands of the player. I will then need to track the positions of the controllers in the game to the position of the controllers in the real world, as well as connect the orientation and position of the camera to that of the headset in the real world.
- Functional Gun
    - This is another reason the game is not yet playable. I have the model and sounds for a gun, but its functionality still needs to be scripted.
    - In theory this is easy to do. The process should basically be: when the gun is fired, call a function. The function will play a sound, update the animation controller, and perform a raycast, sending a message to the first object hit by the ray.
- Dialogue System
    - There is no point to having NPCs in the world if they cannot talk to the player. A dialogue system will add flavor to the game by allowing NPCs to talk to you, as well as add a simple and seamless tutorial into the game.

    - This should not be anything too complex, realistically it should not be more than text bubbles that appear above NPCs. A player should be able to press a button to advance to the next line of dialogue.
- Hub World
    - Although there is only one minigame for more, I think it is important to build support for more down the line. For this reason, I would like a small hub world where players can choose what game they would like to play.
    - The hub should be a small area with NPCs to talk to, portals to minigames, and signs that display settings and high scores.

Citations

Assets (Paid assets were obtained via Humble Bundle)

- https://assetstore.unity.com/packages/3d/props/wooden-pbr-table-112005
    - Model of a table used in shooting gallery scene
- https://assetstore.unity.com/packages/3d/props/guns/wild-west-revolver-0-61712
    - Model of a revolver used in shooting gallery
- https://assetstore.unity.com/packages/3d/environments/landscapes/toon-fantasy-nature-215197
    - Source of environmental props, maps, and skyboxes
- https://assetstore.unity.com/packages/audio/music/orchestral/total-music-collection-89126
    - Source for all music used
- Natalee Zatkoff
    - Source of all character sprites