Testing Report
Rage Quit
The Fumbling Five:

Yong Lu, Jared Tran, Jeremy Euchi, Brooke Clement, Victor Chung

Yong:

Breakable item animations:

Each breakable item has two components. The original unbroken model and the broken variant. I tested each item the same way. I loaded the item into a separate testing scene that I made and then tested the objects with the destructible.cs script. This testing was done by simply left clicking on the model and seeing if the model broke properly. I made sure the models had the proper physics, collisions, and interactions with the player. Although the testing was the same, the bugs were different on some objects.

- Audi: One of the bugs found was that the broken model of the Audi is one-sided see-through from the inside out. I couldn't change this bug unless changing the material and texture of the Audi but that would ruin the aesthetics.
- Door: Door was breaking the wrong way: the wooden shards were aligned at a 90 degree angle in the x coordinate compared to the original door.
- Iphone: Scaling issues with the original model and the breaking model. Fixed by reimporting the original model into Blender and changing the scaling in Blender.
- Lamp: Had the issue of the broken parts "teleporting" after broken. Fixed by moving the broken pieces into the object container.
- Monitor: The monitor and macbook have an interesting bug. During testing, they didn't work on Victor's computer but only Victor's computer. Everyone else had it working and the final release build works for everyone including Victor.
- Macbook: Same issue as monitor.
- Shower: Issues with the textures where one side of the glass was see through and the other side of the glass was solid black. Fixed this by replacing the shower glass texture with the Iphone glass texture.
- Toilet: Broke smoothly. No issues.
- Wall lights: Broke smoothly. No issues.
- TV: Teleporting issue. Fixed the same way
- Foot table: Breaks smoothly just like wood. No issues.

Destructible.cs: Used to test each individual item. How it works: When the user clicks on the object, the object is deleted and at the same time, a broken model of the object is instantiated on the spot to simulate a "breaking" effect. This is the basis for all breaking in the game. This script is later incorporated into Victor's Destructible Interactable script.

Jared:
Breakable item animations:
Each breakable item has an unbroken model and a broken model. Items were first tested in the Jared Test scene by using one of the scripts listed below, then moved into the main scene if they appeared to behave correctly. Certain objects exhibited unexpected behaviors. A recurring issue with some objects involved the broken model appearing at different coordinates than their original models, resulting in the model seemingly disappearing.

- Chair: Broke smoothly. No issues.
- Table: Broke smoothly. No issues.
- Vases: Destroyed version appeared in a different location than unbroken version, causing the object to look like it had disappeared. Fixed by making the coordinates of the destroyed version (0, 0, 0) so that it appeared in the same place as the original.
- Plate: Destroyed version appeared in a different location than unbroken version, causing the object to look like it had disappeared. Fixed by making the coordinates of the destroyed version (0, 0, 0) so that it appeared in the same place as the original.
- Sink: Broke smoothly. No issues.
- Plant: Broke smoothly. No issues.
- Closet: Issues with incorrect rotation and position of parts of the destroyed version caused the object to look very unrealistic when breaking. Later found that the back side of some parts of the closet were untextured, making it look like those parts had disappeared. Fixed by fixing the rotation and position and by making rotated clones of the parts with untextured backs to make them appear to be single parts with both sides textured. Also removed small problematic parts of the destroyed version like handles and door frames.
- Painting: Did not break properly, fixed by fixing issues with rigid bodies in the destroyed version.
- Glass cup: Broke smoothly. No issues.

Originally used scripts such as Destroy_bottle that would test if an object broke if the player collided with it. Later switched to the Destructible.cs script mentioned above and finally Victor's Destroy Interactable script for testing in the main scene.
Red screen effect:

- Timer.cs
- When timeRemaining was less than or equal to 10, the transparent red image was made visible

Jeremy:

- Options: A slider UI is connected in the game that connects the music and a full screen checkbox. In the build, it's simple to test if the slider changes the music's volume and if the game can go in and out of full screen.

- PauseMenu: If the user presses 'ESC', the program checks if the game is paused and the end screen is not activated. It then appropriately calls either 'Pause' or 'Resume' functions. To pause the game, we show the cursor on screen and set the pause menu UI to activate on screen and set the time of the game to be 0. If we resume the game, we make sure the end screen is not active and then do the opposite of pause, hiding the menu and resuming the time of the game.
- Timer: Every frame of the game, it checks if the time is still running. If we reached 0, we show the cursor, enable the end screen and leaderboard UI's, and show the user their final score. Finally, the restart button calls the Restart function which loads the scene again.
- Leaderboard: I programmatically created a GUI table that shows the top 5 scores saved. Using PlayerPrefs in Unity, we can load and save json strings containing name value pairs that are the highscores. We only store the top 5 and we check the two edge cases of initially empty and more than 5. I also sort the top 5 after a new entry is accepted. The input field can only be in a proper name format which is handled by Unity so I don't need to worry about bad input. I only needed to test for an empty json file and test that the lowest score is deleted so it maintains only 5 scores saved.

Brooke:

- Initial Tutorial: After the player loads into the game after pressing play on the main menu, if it is the first playthrough of the game a tutorial UI will pop up. The player can press Q to go straight into the game, or Esc to go into the pause menu. The timer does not start when this is open. Otherwise, if the game is replayed from the ending screen, this initial tutorial will not pop open again. This was tested by loading the game and playing through multiple times in order to check if it only pops during the first round, and if it connects correctly to either the playable game or the pause menu (depending on which key is pressed).
    - PopupScript.cs

- Pause Menu Tutorial: After going to the pause menu in game, there is a button that takes the player to the tutorial screen. Pressing Q will take the player back to the pause menu, and pressing Esc will take the player back to the game. While the pause menu or tutorial is up, the cursor should be visible, and should only be visible after exiting the Pause Menu to the game. Tested by clicking through the options in the pause menu, checking to see if cursor was visible the whole time, and if the correct screen was loaded after pressing Q or Esc.
    - PauseMenu.cs

- Asset Prefabs and Materials: Created layout of game (both inside and outside), and gave box colliders to objects so the player can't walk/fall though.. This was tested just by

loading into the game and observing through the camera if everything looked in order, and making sure the player cannot phase through objects.
    - Most furniture prefabs and materials, excluding car, painting, cup, and electronics for interior
    - Skybox, land, and trees for exterior

- Lighting: Used realtime lighting to illuminate the area. Initially tried using baked lighting, but the team decided to use dynamic objects (which are not compatible with baked lighting) for most assets, and so we swapped to realtime. This could be seen in the main scene without loading in, but was tested by loading in to see how it worked with the player camera.

Victor:

Player Animation Controller:
- Unity's Animation Controller
- For player's animation, a few parameters were made and set inside InputHandler.cs, DestroyInteractable.cs and PickUpMallet.cs in order to change from one animation to another
- Tested in Unity Scene View and Play View

Camera:
- CameraBreathing.cs
    - Zoom in and out while the player character is standing still to imitate a breathing effect.
    - Tested in Unity Scene View and Play View
- CameraController.cs
    - Control the camera movement in the game. Most important being the sensitivity of movement and where the camera should stop top imitate the extent of a person's point of view
    - Tested in Unity Scene View and Play View
- CameraSwaying.cs
    - Random swaying so the camera doesn't feel static at any given point
    - Tested in Unity Scene View and Play View
- CameraZoom.cs
    - On right mouse click, zoom in the camera so player can have a better view
    - Tested in Unity Scene View and Play View

Player:
- FirstPersonController.cs
    - Control player's movement
    - Tested in Unity Scene View and Play View

- HeadBob.cs
    - Random swaying of the head so the camera doesn't feel static at any given point and imitate breathing
    - Tested in Unity Scene View and Play View

Input Handler:

- InputHandler.cs
    - This is the main manager of the system, it takes in input from keyboard and mouse and pass that to the corresponding script
    - Tested in Unity Scene View and Play View

Interaction System:

- Hoverable.cs
    - Check when the player is hovering over an object they can interact with
    - Tested in Unity Scene View and Play View
- IHoverable.cs
    - Public class to set and get values from Hoverable.cs
    - Tested in Unity Scene View and Play View
- IInteratable.cs
    - Public class to set and get values from InteractableBase.cs
    - Tested in Unity Scene View and Play View
- InteractableBase.cs
    - Base class where the other Interaction script derive from
    - Tested in Unity Scene View and Play View
- InteractionController.cs
    - Manager for the interaction system.
    - Tested in Unity Scene View and Play View
- InteractionData.cs
    - Hold the data for the interaction and pass it to the corresponding script
    - Tested in Unity Scene View and Play View
- InteractionUI.cs
    - Create the UI for the interaction system
    - Tested in Unity Scene View and Play View
- InteractionUIPanel.cs
    - Create the UI panel for the interaction system
    - Tested in Unity Scene View and Play View

Score:

- Score.cs
    - Scoring system that is use to print the player's score to the screen
    - Tested in Unity Scene View and Play View

Other helper Script etc.