# Trapped in Firestone: An Escape-the-Room Game

By Khanh Vu, Yashodhar Govil and Urvashi Uberoy

## Introduction and Background

Escape-the-room games have become increasingly popular in the past few years, with many versions online, in virtual reality, or in an actual setup room. However, many older and more popular online editions, which are free and thus more accessible to the general population, are made in Flash, and are hence 2D, which takes away from the immersive experience that makes this genre of game so successful. So, we wanted to make an escape-the-room game that was an immersive 3D experience. We also wanted our game to appeal to the Princeton community to give it a personal touch and make it relatable for the entire student body.

Therefore, we decided to create a Firestone-themed Escape-the-Room game. We wanted to make it special to Princeton as well as to COS 426, so the room is set in Firestone, and the mission is to find Professor Finkelstein to submit your Dean's Date Assignment for the class, as he has been locked inside Firestone library. We took a little field trip to Firestone library to scout out potential rooms, and when we found one we liked in Firestone's New Wing, we made our 3D rendering as close to the original as possible. We hope our game will give the player the feeling that they are actually in Firestone and also hope that it will contribute to the still-growing reserve of 3D games.

## Methodology Overview

The main parts during the implementation phase were:

1. Game logic design: testing the game to check whether the order of events and control flow is smooth.
2. Scene setup: assembling meshes and geometries to make the room look like it is part of Firestone Library.
3. Player interaction: making sure the player can move in all directions, rotate the view, and select objects in the most intuitive way possible.
4. The inventory: adding items that the player selected to their "inventory" so that they could view them again and use them to solve puzzles.
5. Notifications and hints: clicking certain objects should trigger hints or messages to the player, instructing them on what to do next or where to go.

## Game Design and Storyline

For game logic design, we surveyed other online escape-the-room games for inspiration, getting a sense of the style of storytelling and the design of game logic. Accordingly, we came up with the following clues/ Easter eggs that the player can collect in order to unlock the secret door and find Professor Finkelstein:

1. Keys (by the white sofa): These unlock the keyhole at the top of the chest of drawers, and reveal a scrap of paper that is a hint to try to fix the fusebox.
2. Book (on the table in the middle of the bookshelves): On picking this up and clicking on its icon in the inventory, the player is shown an images of the *Rights, Rules and Responsibilities* book which has the number "957" in the top-right corner. This number is the PIN to unlock the fusebox.
3. Fuse (on the middle round table in the back of the room): Once the fusebox is opened with the PIN, the fuse can be added into the fusebox to activate the security system. On doing so, the prox sensor near the door turns from red to green, telling the player that they can now open the door with a keycard.
4. Keycard (near the red chairs): Once the security system is reactivated, the keycard can be used on the prox sensor to open the door and find Professor Finkelstein!
5. Check (on the table with the laptop): A check from the Housing Office for $1000 for messing up room draw (Easter egg for rising seniors)
6. Phone (on the leftmost round table in the back): Professor Finkelstein's phone that he left in this room before he got locked! Clicking on it shows a series of messages between Austin and Professor Finkelstein, with Professor Finkelstein asking Austin to come unlock him.

**Scene Setup**

For 3D scene-setting, related work in player controls, and object intersection is mostly based in Three.js. For object modeling, we used 3DWarehouse to download Collada objects (like bookshelves, lamps, chairs, etc.) and then loaded them into our scene. We manually positioned the objects with reference to the layout of Firestone's New Wing room. Note that, based on the logic flow, the scene's special clue objects are the 6 items listed above. Additionally, the scene also has special objects that these clues can be applied to, namely the drawer keyhole, the fusebox, and the prox sensor.

**Player Interaction with the Scene**

For player interaction, there were two sub-parts that we focused on. The first one was player navigation control, allowing the player to move around in the scene, and the second was the player interaction with objects in the scene.

For player navigation control, we made extensive use of Pointer Lock Control, which allows the player to move around the scene steadily by using the keyboard arrow keys. However, as the cursor is hidden when pointer lock mode is turned on, it is not useful for selecting objects. Thus, we made another feature: when the Enter key is pressed, the scene is frozen and the player can use their cursor to select items. When they press Enter again, the scene is unfrozen, and pointer lock control is once again activated.

Player interaction with objects in the scene include collision avoidance and selection (picking up objects). Selection of objects was done by raycasting. If the closest intersection of the ray from the camera to the mouse was within a certain distance of an object's position, and the player clicked, the player will have selected that item. For clue objects, such as the key, phone, or prox, the items are small enough that we can use this method with a relatively small radius. Once the player has clicked, we set the clue's "visible" feature to false, making it seem like the player has picked it up and out of the scene. To apply these clues to objects in the scene, the player can first select the clue in the inventory (read below) and then click on the object in the scene. If the application is correct, for example, prox to prox sensor, an event will be triggered.

**Handling Collisions: An Attempt or Two**

To make sure that the player stayed within the walls of the scene, we were able to implement a simple version of collision detection by confining the y-axis of the camera to never move above the "human eye level" of 25 units, as well as restricting the player's movements to within the x and z coordinates of the four walls. We did that by first updating the camera position as normal, adding the change in time multiplied by the velocity (which we get by multiplying 500 with the direction of movement, gotten from player keyboard input). Then we check whether this new position is out of bounds, i.e. beyond +/- 200 bounds for both x and z. If yes, we would undo our last update of the camera position and put the player back to the old position by adding a negative of the last velocity times time.

For object collision detection, we first tried to use Three.js's Raycaster to detect intersection with the closest object. However, we realized that the mouse position is "frozen" during pointer lock mode, so detecting intersection is not feasible. We then tried to use the distance between the camera and the object to determine when to stop the player from moving towards that object, but the object's position is just a point within the object that doesn't take in account the size of the rest of the object. Even when we create a bounding box for each object and ensuring the camera position does not violate those boxes, the player's movement in the scene becomes jumpy, at points even unnavigable, as the number of objects in the scene are too numerous and close together. Future work, if time permits, would be to use the confines of each object to determine whether collision is taking place, similar to the way we handled wall collision.

**The Inventory**

We wanted there to be some way for the player to keep track of the clues they have found, and to refer back to them as necessary. The player a certain sense of satisfaction when the 3D items disappear from the screen and appear in 2D form in their inventory. There were a number of possible approaches to making an inventory. We could have it be a pop-up window that players could pull up or dismiss at will. The advantage of a pop-up window is that it would allow more screen space for the rest of the game. However, this might be somewhat unintuitive and clunky to use. Another approach was a static inventory that would always be visible to the player, which was significantly

easier to implement. The only real drawback of the static inventory is the reduced screen space. But making it a thin bar at the bottom of the screen didn't take away that much space from the game, so we decided to go with the latter approach.

Overall, the inventory is implemented as an HTML div element that has some CSS styling to make it more aesthetically appealing. The items in the inventory are managed by some Javascript code that changes the contents of the inventory div element as required. This script also maintains a list of items in the inventory, and other parts of the code can add to or remove from the inventory. The items themselves are represented by simple 2D icons, represented as png files. The alternative would have been some sort of 3D rendering of the objects in the inventory, but we decided that this approach was not worth the performance burden and the significantly increased difficulty. Stylistically, too, it didn't really make much sense.

The last important function of the inventory is to display images on the screen for the relevant items, which, when enlarged, gives clues in the form of written messages. For example, selecting the book should show the image of the book, which is important in solving the puzzle. This is implemented using the same blocker and instructions div that are shown at the beginning. These divs are made visible or invisible as appropriate and their contents are changed to reflect the information we want to show.

**Notifications and Hints**

The notifications and hints were implemented in the same way as the item information displays. Namely, we used a blocker div placed over the play-screen to display this information. The blocker does not interfere with the game. We make it visible or invisible depending on the circumstances. We also change the innerHTML of the blocker as required to display the notifications and hints. This blocker is also linked to the interaction/exploration modes, so it appears when an event is triggered (such as the player picking up an item, or applying a clue to the correct receiving object), and disappears when the player enters the navigation/exploration mode.

Just as with the inventory, there was an alternative to implement the notifications as pop-up windows that can be dismissed after reading. We decided to proceed with the blocker since it was used to display the instructions at the start and the framework already existed. Furthermore, using the same blocker for all the notification, hints, and item displays was more convenient and elegant. It also gave the game a sense of aesthetic congruity.

**Discussion and Conclusion**

Overall, we wanted the game to function properly without any major bugs and the game logic and flow to make sense. As the creators, we tested the game extensively, and also asked our fellow classmates to try playing it. We found that the game did not have any major bugs, but it was a bit slow to load, due to the numerous amount of objects in the scene. If the computer on which the game

is run is running many applications at the same time, or if the browser the player is using is clustered, the game would take up to 15-20 seconds to load, and would draw a fair amount of battery from the device. In rare occasions, the player's movement would also become jumpy. Additionally, we also found that, although the game logic flow was appropriately challenging but still solvable, the initial placement of certain clues made the objects too difficult for players to find. We adjusted the difficulty of the game by leaving the clues in more conspicuous places like on chairs and tables.

Another piece of player feedback was that there were not many things that were "clickable" (objects that could be added to the inventory), which can bore or frustrate the player. Therefore, in addition to our four clues in the original logic flow, we added a few more "Easter egg" items that were either there for the player's amusement or gave the player hints on how to proceed. Besides the delay in the initial loading of the game, we are satisfied with the aesthetics, the functionality, and the logic of the game!

If we have had more time, we would have added collision detection, so that players cannot walk through objects. This would add to the game's realistic feel. We would also add more rooms to increase the complexity of this puzzle (all the while maintaining its solvability), and even add human figures for the player to interact with. We would also have liked to increase the performance of our code, reduce lag and jumpy movements, and make the scene less heavy to load.

**Final Remarks**

Through this project, we encountered similar concepts to the ones introduced in Assignment 3 (Raytracer), but this time we were on the applications end of things. Therefore, this project was very rewarding in the sense that it gave us a glimpse of the real-world applications of computer graphics and broadened our exposure to the power of Three.js.

**Acknowledgements**

Thank you to Professor Finkelstein and all the TAs and preceptors for a great semester!

**Code:** All the code for this project can be found on: https://github.com/ygovil1/find-finkelstein
**Webpage:** Our game can be found at: escaping-firestone.herokuapp.com