Tajreen Ahmed (ttahmed), Aniela Macek (amacek), Lillian Meng (lmeng)
COS 426 Final Project

Goal

The goal of our project was to create a miniature puzzle adventure in the style of the *Professor Layton* video game series. We wanted to include puzzles that incorporated the various graphics techniques we explored this semester. To achieve this goal, we decided to implement two games in particular: a gravity maze and a pitcher pour game.

Related Works

The major work we used as a reference when creating these puzzles was the *Professor Layton* video game series. We decided that the puzzles used in these games would be better than any puzzles we could come up with ourselves, and chose to implement these puzzles in our own style.

Methodology

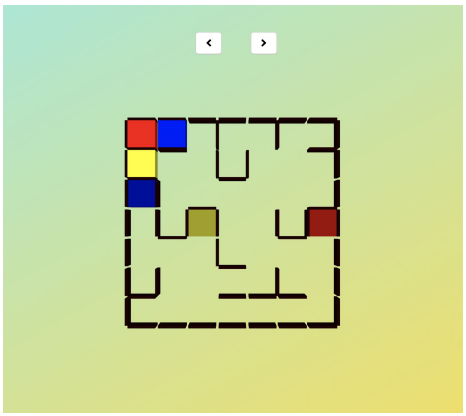
Each of the two puzzle-types involved separate implementation strategies, threaded together with a dedicated landing page and navigation between puzzles.

Initial Technical Decisions

To begin with, we needed to decide whether we wanted a 2D or 3D implementation. While we were inspired by *Professor Layton*, we wanted our own creation of *Enigma* to use a more modern interface and graphical style. We ultimately decided on a 2.5D style, where we are restricted to a flat view from a stationary camera but can still work with 3D objects.

With this settled, our implementation naturally lent itself to using Three.js. The remainder of our project is implemented with Javascript, HTML/CSS and is hosted on Heroku for easy deployment at enigma-puzzles.herokuapp.com.

Gravity Maze



Description

The goal of the gravity maze is to rotate a maze so that three color coded blocks end up in their corresponding goal positions. The box can be put into 4 different rotations, and whenever the orientation changes gravity will pull all the blocks down until they collide with either a wall from the maze or another block.

Implementation

To create the gravity maze, we needed to incorporate gravity, collisions, and animation to display the blocks moving between positions and the rotation of the maze. The walls blocks, and goal positions were rendered with cube objects, whose positions were determined by the positions of the objects within the puzzle. The positions of these walls and blocks were stored separately in 7x7 arrays, so that locations could be precisely tracked, and then converted to approximate locations in the scene for rendering.

One of the important parts of the puzzle that needed to be incorporated was the rotation of the maze itself. In the *Professor Layton* version of the puzzle, the entire maze would rotate whenever the orientation changed, so that when gravity affected the blocks, they would always move down. One possible way we considered implementing this functionality was by rotating the entire puzzle and all of the objects within it. However, this would make position conversions more difficult as we tried to track the locations of the blocks and the walls they might collide with. Instead, we decided to rotate the camera in the scene, so that the positions of the cubes remain the same when rotation occurs but the puzzle itself appears to be moving to the user.

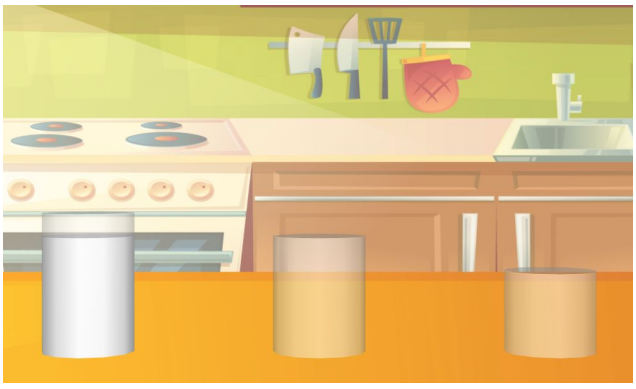
The next parts of the gravity maze that needed to be incorporated were gravity and collisions. Since the camera, not the puzzle itself, was rotating in the scene, the direction of gravity needed to change whenever rotation occurred. The blocks then needed to stop whenever they collided with either a wall or another block. One approach we considered was simply implementing collisions the way we did in Assignment 5, with a force being applied to objects when the rendered locations

collided. While this would make the motion realistic, precision was more important for the game so that the blocks didn't accidently get stuck in positions that they shouldn't get stuck in because of the forces applied to them. We decided that the best way to achieve this precision would be to use arrays to track collisions, so that a block would stop as soon as it was above another block or a wall in the current orientation. The gravity function checks the positions of the blocks and walls using the 7x7 arrays, and stops motion as soon as a collision occurs.

The final aspect that needed to be incorporated was animation, so that the blocks appeared to be moving smoothly because of gravity, rather than instantly move from one position to another. In order to accomplish this, we created functions to update the positions of the blocks as they moved from one array position to the next. These functions would check if the current position matched the position the block should be in, and move the block down by a set increment if they had not yet reached their final position. By re-rendering for each of these iterations of the scene during motion, the scene became animated with the blocks moving smoothly from one position to the next.

As an extra functionality, we decided to also come up with a way to generate random gravity puzzles which were always solvable by some set of moves. In order to do this without creating unsolvable puzzles, we first created a maze with walls in random positions, and randomly chose initial positions for the three blocks in the puzzle. The puzzle then performed a set number of random moves, and recorded the final positions of the blocks after the moves and used them as the goal positions for the puzzle. Finally, the blocks were returned to their initial positions, and the puzzle was rendered for solving. This ensured that there was some sequence of moves that would allow the blocks to end up in the goal positions.

Pitcher Pour



Description

In the game, there are three pitchers of varying capacities; the first can hold 10 quarts, the second 7 quarts, and the last 3 quarts. At the start, the 10-quart pitcher is filled completely with milk. The user can pour the milk between the

pitchers but cannot choose how much is poured; milk is transferred until one pitcher is full or the other is empty. The goal of the game is to distribute the milk such that there are 5-quarts of milk in the 10-quart pitcher and the 7-quart pitcher.

Implementation

The pitchers and the milk are both implemented as cylinder objects to properly convey their intended glass-like shape. Each milk object is positioned at the same location in the Three.js scene as its corresponding pitcher. To ensure that the milk is visible through the pitcher, we modified the material of the pitcher so that it was transparent and had a partial opacity.

In creating the pitcher and milk objects, one of our goals was to ensure that their placement in the scene was responsive. To enhance the overall atmosphere of the puzzle, we had set the background of the page to be an image of a kitchen with a wooden table. Initially, the positions of the pitcher and milk objects had been hard-coded to sit on the table when viewed from a full-screen browser. We realized that the realism of the scene fell apart when the browser was resized as the resizing would often cause the objects to shift off of the table. To account for this, we positioned the objects relative to the canvas size of the scene and made sure to re-render the scene and change the parameters of the camera when the browser is resized.

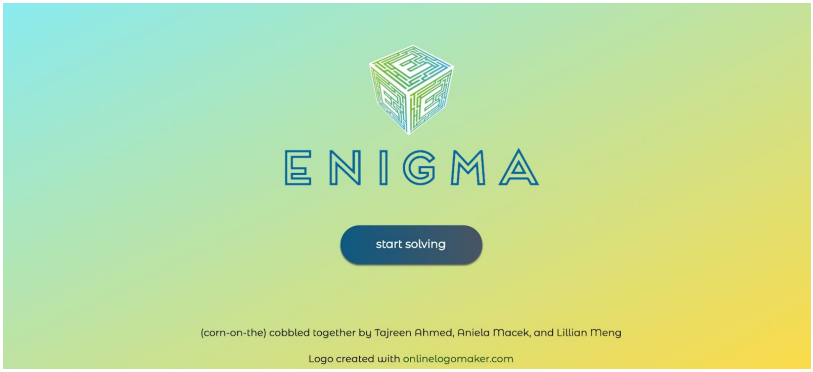
Once the pitchers and milk were created and placed properly in the scene, the next step was to implement the pouring functionality. To do so, we created a function that would trigger upon a click from the user. It takes the mouse click and converts its coordinates from screen coordinates to the coordinates in the scene. Then, we used a raycaster to check for any intersections with the scene. If a pitcher is intersected, we consider it to be “selected.” Once two pitchers have been selected, the milk is poured.

To animate the process of milk being poured from one pitcher to the other, we incrementally removed the previous milk cylinder from the scene and added a new milk cylinder of smaller or larger height. The scene is re-rendered between each step.

Interface: Integration & User Experience

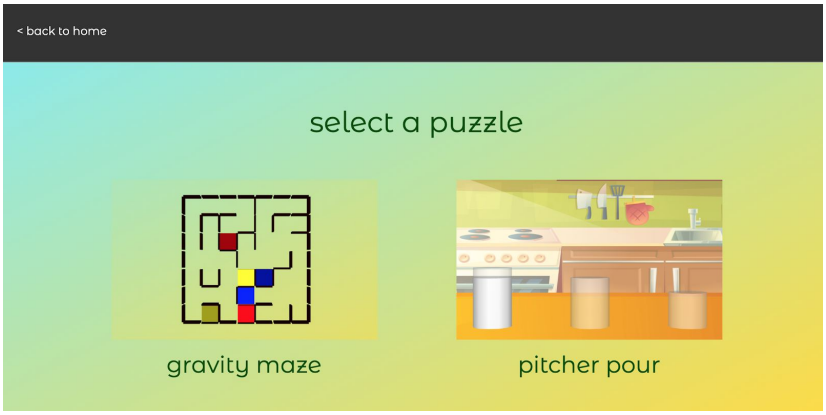
Since we implemented two separate puzzles, it was important to ensure a smooth user experience across the *Enigma* site and game experience. We sought to achieve this through several design decisions.

Landing page



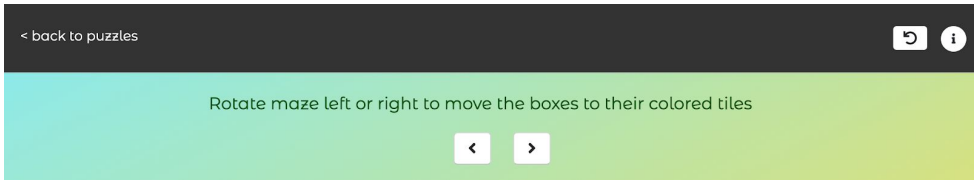
A clean landing page provides an approachable introduction to the game and establishes the initial theme. Our *Enigma* logo is placed on a spinning cube to add dynamism to the page and to serve as an engaging focal point. This was implemented in Three.js by applying our logo as a texture to all sides of a 3D cube. The cube's x and y rotation values are then gradually incremented just prior to each call to render.

Puzzle select page



Including a puzzle select page is necessary to allow users to choose and switch between puzzles. The preview images provide information at-a-glance. Users will naturally express interest by mousing over the appropriate images, and at this point we fade-in more detailed information to facilitate decision-making and level difficulty selection.

Consistent header



We realized that similar header functionality is needed across all puzzles: navigation back to puzzle selection, resetting puzzles, and revisiting game

instructions. A consistent header across all puzzle pages serves as a subtle but effective way to thread all of the puzzles together.

Aesthetics

Maintaining similar 2.5D graphical style was important. The landing page intentionally mixes a spinning 3D cube with an 2D button to enter level select. The gravity mazes allow some form of 3D-ness to be observed in the blocks that move around, but fixing the camera to only view from one side somewhat reduces this to a 2D view. The pitcher pour game also adheres to this 2.5D style, setting the 3D glass cylinders against a 2D-style backdrop. Other design decisions were made to work toward our goal for a more modernized overall aesthetic: serif font, simple green/blue/grey color palette, and non-shaded buttons.

Results

In the latter stages of implementation, we shared our project with friends to receive user feedback we could use to iterate on and polish our project.

One observation we made regarding the gravity maze was that users naturally gravitated towards using the arrow keys on the keyboard rather than using the clickable buttons on the screen. We then went back and implemented key press controls to accommodate this natural user inclination.

Additionally, we realized that users should be able to re-access the instructions for each puzzle, particularly for the pitcher puzzle, where the instructions can be slightly more complex. We added a simple icon button for this in the header bar. We also added a button to allow user resets, reducing gameplay frustration and thereby retaining engagement. These changes, alongside smaller iterations, allowed us to create a polished project that we hope provides an intuitive and satisfying user experience.

Our final project has a wide array of functionalities. We have varying levels of difficulty for our gravity maze, including one option that generates a randomized maze. Our pitcher pour puzzle successfully animates level changes in the liquids and responds effectively to user interaction. Overall, *Enigma* is designed with user experience in mind.

Conclusion

Overall, we were very pleased with the puzzles we created. We were able to incorporate many different techniques that we learned throughout the semester such as rendering, collisions, ray tracing, and coordinate transformations.