3D Minesweeper

Nicholas Cabrera, Ayomikun Gbadamosi, & Dara Oseyemi

Abstract

This project aims to create an optimized three-dimensional version of the well-known Minesweeper game. In order to achieve this, we used the THREE.js mesh features in tandem with texturing and ray tracing from mouse clicks to create an interactive experience for the user to interact with an assortment of cubes. The success of the 3D Minesweeper game was primarily assessed qualitatively, using parameters such as enjoyability and smoothness of the gameplay. The smoothness of the gameplay can be defined as noticing loading speeds and lag time in the camera movements. Experiments were conducted throughout the game development process to evaluate game performance as well as general playing experience. Some of these experiments included generating several new fields to ensure sufficiently random field generation, loading highly customized levels to check that there is high variability in the types of levels that can be played, and both playing levels to completion and purposely failing levels to confirm that the game can run independently. The 3D Minesweeper game developed in this project has the following key features: mouse click-dominant controls scheme, starting menu for user customization of the subsequently generated game field, and a three-dimensional cubic game field that can be interacted with using the same rules and objectives as the original Minesweeper game. The final game product is fully functional with fluid camera movements, reasonable loading times, intuitive controls, and player access to highly customizable, randomized three-dimensional game fields.

Introduction

Minesweeper is a widely known and successful computer game that is enjoyed by individuals around the world. The single-player, logic-based game is typically played on a two-dimensional rectangular board, as shown in Figure 1, consisting of a matrix of covered squares that the user's click can uncover. The goal of the game is to locate a predetermined number of randomly-placed mines by using numerical hints, which are revealed in the squares that do not contain the mines, in the shortest amount of time possible; the user must avoid the squares containing the mines during this time because the game ends if a mine is clicked on by the player.[1] While the game concept presents its own challenges due to the implicit time constraint, it also leaves a significant amount of room for expansion to produce a more complex game. The game has the potential to expand in many ways, including varying the number of tiles, the number of bombs, and the base formatting and configuration of the board. Upon

[1]"Minesweeper," from Wolfram MathWorld, accessed May 3, 2022, https://mathworld.wolfram.com/Minesweeper.html#:~:text=Minesweeper%20is%20single%2Dplayer%20logic,a%20mine%2C%20the%20game%20ends.

exploring the ways the game board itself can be manipulated to increase the game's difficulty, it was found that the inherent two-dimensional playing space can be changed to allow for three-dimensional gameplay. Thus, this project aims to adapt the original Minesweeper game's fundamental rules to be played using a three-dimensional board space. Upon creating a 3D Minesweeper game, the players will have access to a new rendition of the game once playing the original Minesweeper game begins to feel tedious and repetitive. Many Minesweeper players who start off enjoying the game find that the lack of diversity in the gameplay style causes them to eventually abandon the game altogether. With a new game version, these players can once again return to one of their favorite games and encounter new challenges that will ultimately enhance their problem-solving capabilities.

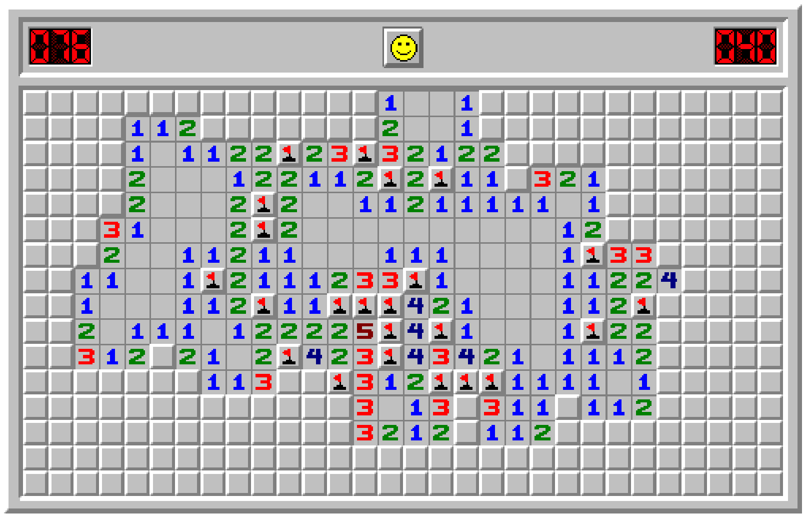


Figure 1. Original 2D Minesweeper Game

Previously, there have been several other instances where originally two-dimensional games were adapted to be played in three dimensions, both in the history of the course and in the real world. One such example comes from the project from Spring 2021 of 3D Tetris, which gave a lot of original inspiration for the project due to one of our members discussing with the said group in the past. The project aimed to rethink the idea of the 2D game of Tetris into the scope of 3D by creating entirely new pieces to restructure towards 3D as well as offer intuitive controls for the project.[2] Thus, it began our process of not only thinking of how we need to restructure Minesweeper in order to work in the new 3D plane but also gave us an idea of the challenges we would face when working with THREE.js shapes.

With the Minesweeper game, in particular, there already exist many online applications that display a variety of approaches to implementing the game in three dimensions, each with its own set of unique strengths and flaws. One of the commonly recurring weaknesses of these 3D Minesweeper games is that the view of the cubes' locations in the larger three-dimensional matrix and their revealed values is often unclear or consistently obstructed as the player is completing the game, as shown in Figure 2, where the cubes are the three-dimensional manifestation of the covered squares. Also, these games are relatively repetitive amongst themselves because they each typically contain only one standard, filled three-dimensional shape, which is commonly the cube, for the player to click through; the shape is not able to be changed as the user plays multiple rounds, which quickly results in the same tedious gameplay problem that discourages players. However, despite these drawbacks, all of these games do

[2]EdGartner, "Edgartner/3DTetris," GitHub, accessed May 1, 2022, https://github.com/EdGartner/3DTetris.

successfully capture the essence of the original Minesweeper game and translate it into a three-dimensional experience.

The critical information gained from observing the other 3D Minesweeper game attempts was applied in determining the ideal approach for this project's implementation of the game. Initially, the approach involved utilizing the pre-existing THREE.js faces in order to manage the various shapes involved in the games; however, it was quickly made evident that a more efficient approach would be to develop a mesh from scratch that encompasses all the game components. To test the game controls, a hollow cube of smaller cubes was initially created as a baseline product, and this baseline was then built upon over time. Since the original Minesweeper game was already heavily mouse-click based, this remained the primary control method in this program, allowing the player to have fine control and increased maneuverability around the space. To ensure that the value of each revealed cube is clearly conveyed to the player, textures were created and applied from a map to each of the meshes that had the corresponding value to a representative texture. Once the texture-mapping system was performing optimally, the program controls were expanded to more accurately emulate those of the original Minesweeper game by establishing a system of being able to flag blocks, a staple feature of the original game to keep track of the positions of possible or confirmed mines. Though, at first, the number of bombs present in the game and the configuration of the three-dimensional board were static, player customization was later added for both of these features in the form of an HTML landing page (See Figure 3) when the mechanics of the static version of the game were fully developed. Such a customization feature essentially eliminates the aforementioned issue of monotonous gameplay. This approach to a 3D Minesweeper game should work well in most circumstances, regardless of player experience level, because the controls were purposefully designed to be relatively intuitive and straightforward, and the game's objective, even despite the new three-dimensional board, is well-known and clear. Furthermore, the game was developed and run on Windows 10 computers with average processing power, so it is expected to run smoothly with any other, better systems. This is because the nature of the game is not very graphics-intensive, with the longest loading time occurring at the beginning of the program when a large section of the cubes is revealed at once, usually occurring more in larger playing fields and improving over time as the player clears them. Overall, the straightforward nature of the already popular game makes it very compatible with the outlined implementation approach.

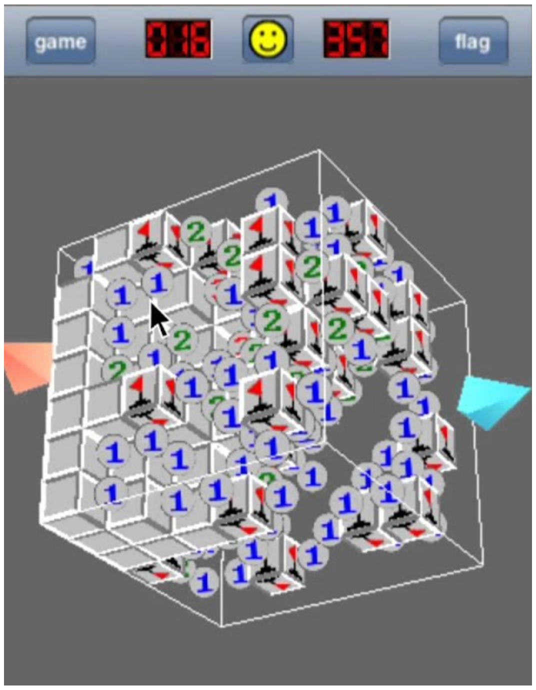
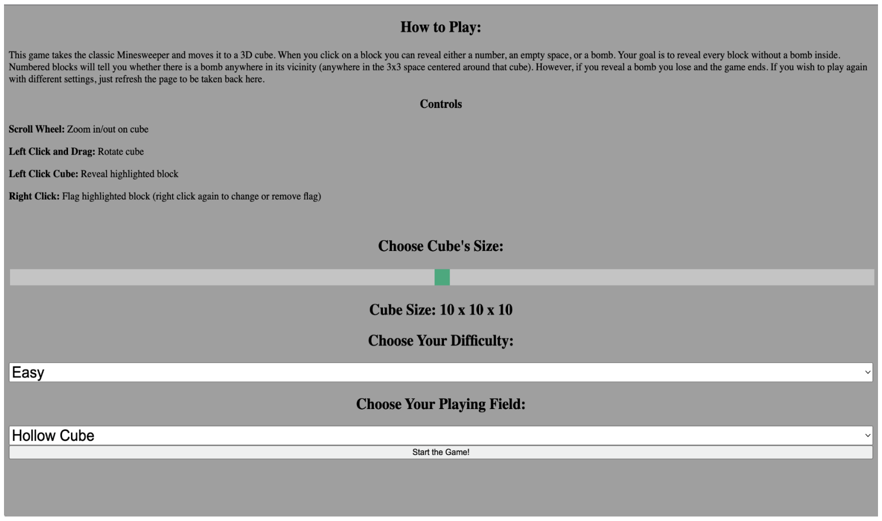


Figure 2. Example 3D Minesweeper Online Game



How to Play:

This game takes the classic Minesweeper and moves it to a 3D cube. When you click on a block you can reveal either a number, an empty space, or a bomb. Your goal is to reveal every block without a bomb inside. Numbered blocks will tell you whether there is a bomb anywhere in its vicinity (anywhere in the 3x3 space centered around that cube). However, if you reveal a bomb you lose and the game ends. If you wish to play again with different settings, just refresh the page to be taken back here.

Controls

Scroll Wheel: Zoom in/out on cube

Left Click and Drag: Rotate cube

Left Click Cube: Reveal highlighted block

Right Click: Flag highlighted block (right click again to change or remove flag)

Choose Cube's Size:

Cube Size: 10 x 10 x 10

Choose Your Difficulty:

Easy

Choose Your Playing Field:

Hollow Cube

Figure 3. 3D Minesweeper Level Customization Menu

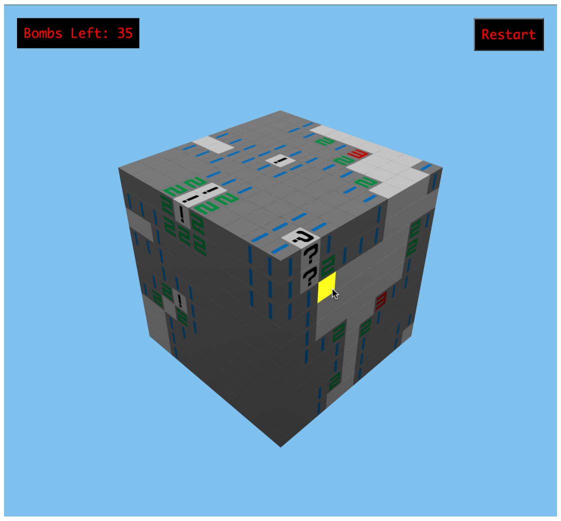


Figure 4. Example 3D Minesweeper Level

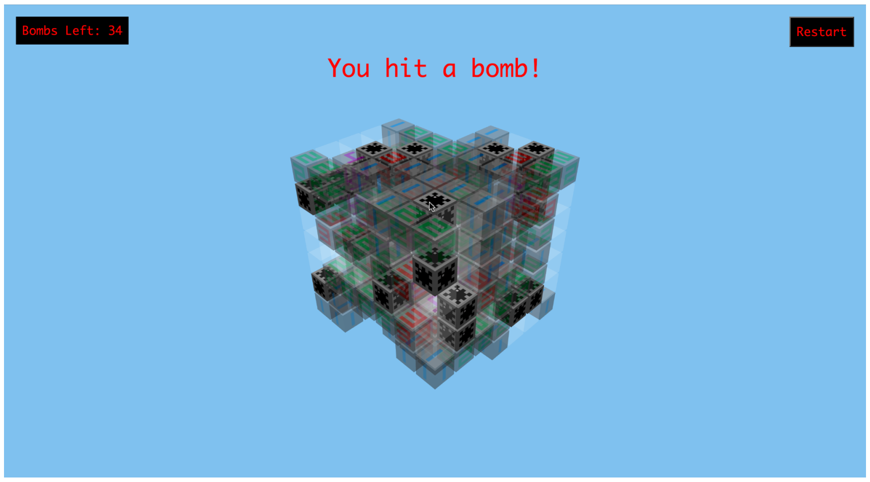


Figure 5. 3D Minesweeper (Filled Cube) Level Loss Display

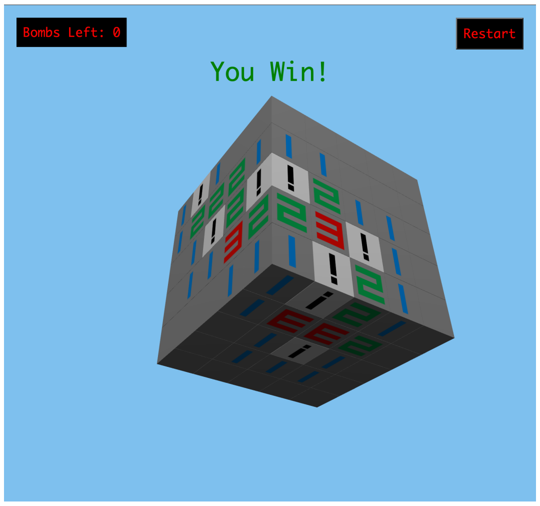


Figure 6. 3D Minesweeper Level Victory Display

<u>Methodology</u>

Cube Object Creation (Cube.js)

The first step in executing our approach involved creating the smaller geometric shapes, or the three-dimensional version of the clickable tiles in the original Minesweeper game, that make up the larger game board. At first, the planned implementation was to store information for each face of a cube and then use that information to determine what displays when the user clicks on the particular cube face. Upon realizing that this approach was overly complicated and would lead to creating an excessive number of objects, the use of the basic THREE.js cube mesh was opted for instead of loading GTLF files[3]. Each cube mesh was made to hold instance variables related to the status of the specific cube, including whether it contains a bomb, how many of the adjacent or diagonal cubes contain bombs in the 3x3 space, and whether the cube has been revealed or flagged by the user. While the change in plan allowed for a significant decrease in the number of objects that must be stored when creating a full game field, it also made it difficult to vary the shape of these board pieces, as will be discussed in a later section because a new mesh would have to be created.

To convey the information regarding the cube's status to the player, when necessary, as the game is in progress, image files containing different textures were created by us, mapped to a specific cube value or meaning, and then applied to the cubes based on their current value or status. Each image file contains a texture with a number between 1 and 26 because 26 is the maximum of a cube's neighbors that can contain a bomb when considering a three-by-three space, a bomb, and either a question mark or an exclamation point, which are used to indicate a flagged cube by the user's input. It is important to note that the bomb texture is only applied to cubes when the player clicks on a bomb cube, and it is applied to all existing bomb-containing cubes at once. These textures were stored in a separate folder from the rest of the app files, so the image files had to repetitively be passed to other functions to avoid errors at build time. A possibly better approach would have been to place the issues with the CubeGrids file, which will be discussed next, but when the solution was attempted, issues relating to filing paths not being recognized arose. Furthermore, the textures used were very simple; it might have been beneficial to find or create a unique art style for the game.

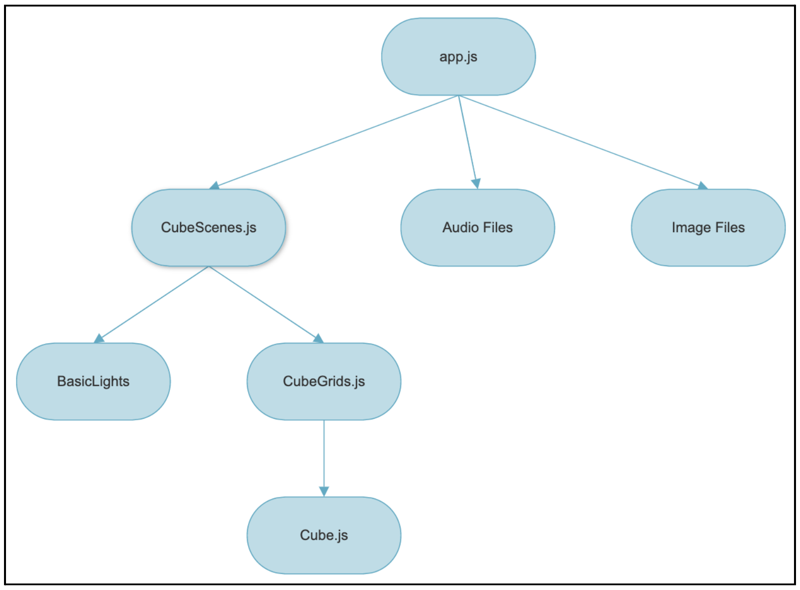


Figure 7. High-level 3D Minesweeper Project Architecture Summary

Full Game Field Generation (CubeGrids.js)

The next step is to be able to assemble multiple cube objects into a larger three-dimensional shape or game field for the player to interact with. During this process, the cube positions in the field must be initialized, the randomized cube locations of the specified number of bombs must be determined, and the values of neighboring cubes must be computed based on how many of their neighbors contain bombs. Our method of completing

[3]"THREEjs Box Geometry Library," Three.js – JavaScript 3D Library, accessed May 1, 2022, https://threejs.org/docs/api/en/geometries/BoxGeometry.html.

these tasks starts with taking input values from the user for board size, difficulty in the form of an integer(1, 2, or 3), and a boolean value for whether the board shape should be filled or hollow. Based on the user selections, the square of the input size number of cube objects is created and arranged to create a larger 3D cube that is either hollow or filled, depending on the user choice that determines the value of the boolean. Once the cubes are placed, the Math.random function is used to select indices from the array of cube objects randomly; bombs are placed in the cubes at the selected indices. After bomb placement, the function iterates through all of the cubes to compute each cube's value by checking how many of the neighboring cubes within a three-by-three cubic space contain bombs. Aspects of the player's interaction with the game field are also simultaneously carried out by functions in this file, including highlighting unrevealed cubes the mouse hovers over, flagging cubes by right-clicking, recursively revealing cubes and their neighbors when applicable, indicating when the game as been won or lost (See Figures 5-6), and determining cube transparency when playing on a filled cube grid. The level of transparency is based on cube size, becoming more transparent with a larger grid of cubes. Even despite creating the transparency feature, transparency is still relatively obscure at high levels; thus, a better strategy may have been to have the user be able to toggle on and off transparent blocks, but the time constraint greatly limited this possibility. A different method could have been utilized to create separate files for filled and hollow game fields; however, one file containing all of the functions was found to be a better option to preserve conciseness. Furthermore, the number of bombs set for each difficulty level could have been a direct choice from the player, but to avoid creating particularly extreme levels, this was decided to be the best choice.

Mouse Click Control Scheme (CubeScenes.js)

After the user enters their selections for the level customization, the player's mouse clicks must be linked to the display functions outlined in the previous section in order for the player to be able to interact with the cubes and change the game field; thus, this file implements the underlying mechanics of the Minesweeper game that allow the user to flag, highlight, and reveal cubes. Therefore, the flow of the function calls through this file are as follows: the player's mouse movement or click triggers the function in this file, causing a specific change to the game field, and then the corresponding function in CubeGrids.js is called to execute and display the change. A different possible approach would have been to combine the user interaction-related functions in CubeGrids.js and Cubescene.js into one file, but this used approach felt more natural when carrying out the project workflow. Additionally, functions to create text on the screen that tracks the total number of bombs and the number of bombs that have been flagged are located in this file.

Menu Page for Level Customization (app.js)

Creating a level customization page involved first creating an HTML page for the user to interact with. This page includes interactive sliders and dropdowns that provide input parameters to the functions that generate a game field based on the board size, level difficulty, and grid type (filled or hollow) choices for the user. These HTML parameters are then removed when the user clicks the start button. Some of the code for implementing this functionality was borrowed from the Beehive Simulation project[4] and W3.[5] Further stylization of this page would have been ideal if not for the project's time constraints. By pressing the start button, audio is loaded and the canvas for the game field is generated through the collaboration of the other files (See Figure 7). It also generates a reset button that creates a new scene with the previous input parameters the user selected; however, the loading speed of this feature tends to decrease as the number of cubes present in the game field increases. The basic lighting, BasicLights.js, is used from the starter code, though it may have been a good idea to find better lighting effects. Event Listeners that act as controls for the mouse, camera rotation, and cube selection are also located in this file. An alternative option would have been to also look into implementing keyboard controls and how to change between the two control schemas to accommodate all player preferences.

Game Audio

As an added feature, background music was added to the game to allow for a different sense of ambiance for each of the game-level options. Several audio files for music and for sound effects, which help convey victory, loss, and cube selections to the player, were imported. Already existing music from outside published games, specifically from the game Genshin Impact[6], were utilized. An alternate approach would have been to create original music to use in the game or to find three more closely related music tracks with different tempos to create distinct environments. An example of this can be observed in the popular Kahoot game, where music tempo depends on the amount of time the players have to answer the question.[7] At the suggestion of the TA, parts of the code to implement this feature were borrowed from the Beehive Simulator project from last year. [8]

Results

The success of the project can be assessed qualitatively, using factors such as the overall enjoyability of the game and the smoothness of the gameplay. In order to test the success of the game, several experiments were performed. Among these tests was the repetitive generation of cubes in the gameboard to check that the cubes' values and the bombs' placement were feasibly random. Also, experiments were run using varying degrees of player optimization to assess, with more tests being done at high levels of customization, to ensure that a diversity of level difficulties and scales can be achieved by users (See Figure 4). Additionally, several levels were

[4] skyliu10, "Skyliu10/BeeHiveSim," GitHub, accessed May 1, 2022, https://github.com/skyliu10/BeeHiveSim.
[5] "How to - Range Sliders," How To Create Range Sliders, accessed May 1, 2022, https://www.w3schools.com/howto/howto_js_rangeslider.asp.
[6] "Genshin Impact," Genshin impact – step into a vast magical world of adventure, accessed May 1, 2022, https://genshin.hoyoverse.com/en/.
[7] "Kahoot!," Kahoot!, accessed May 1, 2022, https://kahoot.it/.
[8] Skyliu10, Beehive Simulator.

played to completion to verify that the game can consistently, smoothly run all the way through independently, meaning without any back-end code manipulation. We purposely failed some levels as well by clicking on the bomb-containing cubes to confirm that the game can recognize and respond to instant losses.

The results of the continuous evaluation of the game quality indicate that further optimization of the background algorithms to display the game would be ideal because, though it is capable of running and being played with particularly large game board sizes, it runs noticeably slower until more cubes have been revealed. Players can play full games with consistently random generated fields without any errors or difficulties, and the game successfully identifies when the player has lost the round. Lastly, the game's controls, including the camera movement around the cube, remain smooth at varying levels of player customization. The player customization is fully operational, allowing for the reliable generation of game fields corresponding to the user's chosen parameter values. The final 3D Minesweeper game product is simple and easy to understand for all players, even with the added complexity of the three-dimensional gameplay.

Discussion

The approach utilized in implementing the 3D Minesweeper game concept was far more successful than originally anticipated. The final gameplay feels fairly intuitive, and there are rarely problems with the game controls and the camera movement. The longest loading time of the game occurs after the player starts the game and when one cube selection reveals several neighboring cubes at once (typically happens for one of the first few sections on easy large, filled cubes grids). However, as mentioned previously, as the player plays and removes cubes, the rendering time of the game improves significantly, allowing for more seamless and smooth gameplay throughout the level. Furthermore, the approach greatly simplified the process of storing the data for each cube and then utilizing the data to calculate the current status of all neighboring cubes, as we store fewer items than storing the individual faces and can rely on mesh positions instead of neighboring faces for the location of mines. Although the approach worked, there are other variations of this approach that could have led to an even more polished game. Another approach that would have been used involves allowing the user to choose the control scheme of the game, allowing the choice between mouse click-dominant or a keypress-dominant control, such as WASD keys and arrow keys to control a cursor and camera, respectively. This would satisfy all players with different control preferences for computer games. In this alternate approach, presets for generating a specific type of preferred field can be saved for the user, allowing them to play the same type of level multiple times upon repeated visits instead of inputting it each time. Finally, a different type of algorithm for generating random cube positions, values, and neighbors could have also been used in this other approach, as opposed to the fairly rudimentary approach of just inputting a random position in the array of cubes and making it a bomb, in order to create a more interesting and diverse set of game fields.

For future work in enhancing the gameplay of 3D Minesweeper, the game can be expanded to include other shapes and meshes instead of just a cube for the individual clickable spaces in the larger game field geometry. An example of this could be creating a three-dimensional game field matrix made up of pyramids or any other polygonal shapes; implementing this feature within the scope of this project was made impossible by time constraints. Additionally, the overall game field matrix shape can be varied; for example, the larger field shape could be like that of a 'plus' sign made up of smaller rectangles that serve as the clickable components of the game. As previously stated, the game's runtime could be optimized, especially to achieve maximum efficiency at exceptionally crowded and complex levels. Visual edits to the game field can also be made by providing users with options of stylish textures to put on the individual shapes within the game field and the background. The design of the initial startup menu can also be modified to be more visually appealing to users. The last change that could be made is to create the extra control mode options for the users, an example scheme of which could be moving around with the arrow keys and rotating with the WASD keys.

Throughout the process of designing and implementing the game, much knowledge was gained, and several lessons were learned that would allow for a more fluid development process in the future. To allow for a more efficient project workflow, it was made clear that delegating roles and tasks to team members early on in the process would allow for a more organized workflow, as well as clearly outlining a plan of action for creating the relevant application. Moreover, organizing the code into several integrated files and then compiling them together helps keep track of the various components necessary for the successful operation of the application, which consequently simplifies the debugging process. It was noted that it is important to explore new features in graphics to create a more seamless user experience, such as creating a control schema and other methods of displaying a sufficient amount of information to the users. The last and possibly most significant lesson learned during this project involves finding the ideal implementation of the application idea while keeping in mind that the implementation needs to be flexible enough to make changes later on as new, unexpected problems are encountered.

Conclusion

In summary, the goal of the project, which was to create an interesting and dynamic expansion of the Minesweeper game into a three-dimensional space, was accomplished as effectively as it could have been since, given the time constraints, the game is fully functional with highly variable levels and ready to be played by users. As established earlier, several next steps could be taken to further develop the game, such as implementing new control schemes and adding new field geometry options. Present issues with the game that can be revisited in the future involve fixing any obscure bugs within the program that could potentially arise from corner case inputs from the user, revising the algorithms behind the game display to improve runtime, and a number of other optimizations.

Contributions

All three project group members contributed in refining the original project plan, finalizing the list of features to be included in the final game, and determining the ideal approach for creating the 3D Minesweeper game. The work was divided such that Nick worked on the code for the startup menu for level customization and game audio, and Dara worked on the code for cube object creation, game controls, and game canvas creation. Ayo worked on code for game field generation and noted the methods, approaches, and logic used while developing the game and utilizing this information to compile the project report.

Works Cited

EdGartner. “Edgartner/3DTetris.” GitHub. Accessed May 1, 2022. https://github.com/EdGartner/3DTetris.

“Genshin Impact.” Genshin impact – step into a vast magical world of adventure. Accessed May 1, 2022. https://genshin.hoyoverse.com/en/.

“How to - Range Sliders.” How To Create Range Sliders. Accessed May 1, 2022. https://www.w3schools.com/howto/howto_js_rangeslider.asp.

Kahoot! Accessed May 1, 2022. https://kahoot.it/.

“Minesweeper.” from Wolfram MathWorld. Accessed May 1, 2022. https://mathworld.wolfram.com/Minesweeper.html#:~:text=Minesweeper%20is%20single%2Dplayer%20logic,a%20mine%2C%20the%20game%20ends.

skyliu10. “Skyliu10/BeeHiveSim.” GitHub. Accessed May 1, 2022. https://github.com/skyliu10/BeeHiveSim.

“THREEjs Box Geometry Library.” Three.js – JavaScript 3D Library. Accessed May 1, 2022. https://threejs.org/docs/api/en/geometries/BoxGeometry.html.

Referenced Projects/ Libraries for Code Implementation

- Beehive Simulation: https://github.com/skyliu10/BeeHiveSim
- 3D Tetris : https://github.com/EdGartner/3DTetris/blob/main/3D%20Tetris/COS426%20Final%20Project%20Write-Up.pdf
- THREE.js Library: https://threejs.org/docs/api/en/geometries/BoxGeometry.html
- Genshin Impact: https://genshin.hoyoverse.com/en/
- W3: https://www.w3schools.com/howto/howto_js_rangeslider.asp.
- Bomb Sound Effect: https://youtu.be/Y3y_Oq6M5mw

- Victory Sound Effect: Taken from Happy Wheels video game files, https://youtu.be/lcJH8JtgZoE
- Highlight Sound Effect: Taken from Persona 4 Golden video game files