Approach

Our approach to implementing our game was to first have the game window show up using Java Swing. Here we had to calculate how big we wanted the window to be and the size of each “tile” of the game would be. This led to us learning that we needed to use a thread to keep the game running continuously. From there we went to implement characters, starting with the main character and how he moves. We created a Mobile class that would represent the moving entities on the map. A MainCharacterTV class was then created and to make him move around the window, we implemented a key handler that takes in input from the keyboard and moves the character around accordingly.

After having the character successfully show up on screen and move, we decided to start creating the map by tiling the window. Here, we implemented a tile manager that would tile the window to its perfect dimensions with the images we provided it with. Since we wanted a 3D look to our game, we also created a wall manager that would load in walls after the tiling was done. When an entity or object is right above a wall, it will give the effect that it has depth and that it's actually behind it.

With the map loaded, we then implemented the moving enemies and the rewards. Here we extended the moving objects class and static objects class and created enemies and rewards respectively. Since these were extensions of already existing classes, drawing them on the window was simple. What really was important when implementing these objects was how they interacted with the main character, such as the game ending if the character was met with an enemy or the character’s health increasing after picking up a power up. These enemies are also implemented to follow the character around and to minimize getting stuck on walls.

The way we spawned our enemies may seem unorthodox, but it was implemented with game balancing kept in mind. If they were to spawn randomly, they could all have been spawned on one side of the map and the player could not have traversed through to get to the exit.

The next step in our approach was collision detection, and to accomplish this we added a collision checker class that would check the tiles that the character was on and see if that certain tile was a wall or not. If the tile was, for example, just a floor tile, then the character would be able to move freely into that tile, but if the tile was a wall the character would get stuck and could not move past that wall.

For the start screen, it was a matter of having a variable indicating whether the game state was in “play mode” or “menu mode”. From there, we added the title of the game and a picture of our main character. The main menu also has option selector, where the currently selected option is indicated with “>”. Once you have a selection, simply press enter and the game will start or quit with the window closing automatically.

Health for the main character was included by having a health variable be part of the MainCharacterTV class. The game will check the player's health every time the game updates and changes the on-screen health bar accordingly. Every "tick" of the game will subtract a portion of the player's health.

Since traps extend the StaticObject class, it was just a matter of changing how the character interacts with them. Rather than have the player's health increase when they collide with the trap like the battery power up, the game ends, telling the player they have lost.

One of the requirements stated there needed to be multiple rewards that must be acquired by the player to win, so we implemented multiple key cards on the map that must be collected so that the player can exit the level.

For the implementation of how the player loses, we decided that once either the player has collided with an enemy or trap or has run out of "battery" the game will end abruptly and tell the player that they have died. This was very similar to the menu screen as there is another option selector with a retry option, bringing the player back to the starting position of the game and a quit option which would bring the player back to the main menu screen.

Attempting to follow our original plan, we also implemented a second level to our game. To enter this second level, the player must first collect all three key cards from the first level and then go to the stairs at the end of the level. Once they have done this, they are met with a screen informing them that they have beat this level and that they can either continue or quit the game. Once the player continues, a new level is loaded with a different map where enemies, key cards and traps are placed in different locations. This has the same win condition as the first level, and once they have completed that, they are met with the winning exclamation that they "escaped the lab".

A feature that we decided to add was to implement a maximum number of retries that the player has at each level. Each time the player would die on the level, their number of retries would decrease by one. Once their retries become zero, the game is over and they would be forced to go to the main menu, where they would restart the game.

Adjustments

After the feedback from Phase1, we made major adjustments to our overall game design. These adjustments are outlined below.

Rather than having a menu class to display the main menu or pause screen, we changed the design to have a UI class. This would also take care of on-screen elements such as the health bar, and drawing menu screens for the main menu, the instructions screen, the win screen and the lose screen. This class is essentially in charge of static menu elements that need to be drawn.

Moreover, we decided to change the Map class to a wall and tile manager. These would be drawn separately, giving a 3D effect, as mentioned before. We discovered that we could load the map with a text file, so we integrated that into our system, where different integers (0 or 1) would represent a wall or a tile.

Instead of a single Game class that handles user input and displaying the game itself, we split the Game class into two separate classes. One for handling user input, and another for setting up the game. For user input, we added a KeyHandler class to handle input from the keyboard. This class sets keys to control the direction of the player moving and selecting menu options. For the setup of the game, we added a GamePanel class that initializes the entity positions, the health of the main character, displays all components and the game board itself. This class also handles resetting the main character and other entities to default positions and health after completing or losing a level.

For the positioning of our entities, we removed the Position class entirely. We decided that it didn't fit our game design and made it unnecessarily complicated. By removing the Position class, we implemented an Entity class that takes care of all static objects and moving objects and their respective positions on the map.

For the traps, rewards, regular and bonus class, we decided to combine them into one StaticObject class that encompassed all non-moving or static objects. This was also applied to the enemies class, where they were changed into a MovingObject class along with the main character. Furthermore, we removed one of the two enemies we had initially planned to add, as we later realized that it was unnecessary.

Additional features we did not have in the initial design are a key card reward that is to be collected for the player to win the game. On top of the main character's health, we added a score. We also added options for the player, such as continuing to play or quitting the game. For retrying the level, we added a limit to the number of retries. Of course, these were missed by our use cases in phase 1, so we made sure that these features were included in phase 2.

To inform the player how to play, we added an information screen that is accessible by pressing "i" when playing the game. Instead of having a pause button on screen, we opted to have the player press "p" instead to pause the game. These options in game are also presented to the player in the main menu.

Management Process

Everyone participated in planning out how we should start implementing the game. We had weekly meetings where we discussed what we should do and where we could ask for some help if needed. We planned out a schedule where we had goals to finish certain parts by this date. Furthermore, we communicated every change and update made to the project to avoid work being done twice as well as merge conflicts. For division of responsibilities, we opted to have everyone contribute evenly to the project, so everyone was part of implementing the game as well as the document.

External Libraries

For the GUI, we used Java Swing and Graphics2D because it was very intuitive and can be implemented well into our game. Graphics2D was extremely important to our game as it was the main drawing tool used to render our main character, the enemies, the rewards and map. Additionally, the Java Abstract Window Toolkit that was used to hold the sprites for our entities as buffered images.

Measures Taken

In general, measures we took to ensure the quality of our code was doing code review of each other's responsibilities. We also referenced the textbook and the lecture slides regarding design principles and to make sure we were following general coding etiquette suggested by the textbook.

More specifically, our initial goal was to get a functional game working. After, we reviewed the code, simplifying and removing functions that were repetitive or doing similar tasks. Once this was complete, we began adding more functionality to the game such as levels, options for the player, improvements to the map design, and additional sprites and graphics to make the game more visually appealing. We added descriptive comments and JavaDocs comments throughout the implementation process to ensure everyone understood the functionality of each method used.

After each new functionality was added, we tested the new additions by playing the game from start to finish, ensuring there were no bugs. If bugs were found while playing, we inspected our changes, making sure all game aspects were being updated or reset properly - specifically with the levels.

Challenges

A big challenge we faced during our implementation was redesigning our plan due to the feedback received from Phase 1. We spent a significant amount of time redesigning our class structure and the inheritance between classes.

A factor that created challenges was that not everyone was very familiar with Java, as this was their first experience with it. There was definitely a learning curve when we started with the project.

One challenge that was prevalent was how we would implement collisions between entities. We had to determine at which x value and y values where two entities would intercept and have an action occur.

Another issue we faced was a bug that occurred when the main character was hugging a wall on his left, he would get stuck when the player tries to go down. This was very frustrating, as we did not know what was causing this, but after going through our code meticulously, we found out that we had accidentally made a mistake in the calculation.

An issue that later occurred was the enemies having to follow the main player around. Here we had to figure how the enemy would react to the movement of the player and make the correct movement towards the player.