CMPT 276: Group Project

Spring 2020

Phase 4- Group #9

Original Plan

Our original plan actually did not stray too far from our final product. Initially we had the setting of a software engineer being bullied by their boss at work and as a means to get back at their boss they would deposit items collected around the office into their morning coffee. This whole setting was wiped out due to reasons: First, we initially planned on making sprites for an entire office environment but we deemed it would cost too much time. Second, after programming the movement of the enemy, the movement reminded us of the movement from the ghosts from Super Mario brothers, and other horror games where the enemy subtly chases the player from behind. Instead we found a free sprite sheet for a medieval dungeon type setting and instead of collecting items for the coffee and avoiding nosy coworkers, the player would now collect treasure while avoiding dungeon monsters and traps.

In terms of gameplay, pretty much everything stayed the same. The whole premise of collecting items to open the exit point while avoiding enemies and traps never changed. Just the aesthetic.

For the UML, we did stray a lot further from our original UML diagram but some things remained the same. As we mentioned before in phase two, we tried to implement a maze-like game that uses a 2D array with all the overlying objects contained within it. But after scrapping that idea and opting out for smoother gameplay and movement with all eight directions, it resulted in changing a lot of the interactions between the game objects that we originally planned in our UML. For example, our original idea had a maze that stored the player and monitored all of the player's movements and interactions. But instead, we decided to have the player be responsible for its own interactions and have it simply placed on the JFrame, on top of the map class that has been placed there previously. If we programmed our game using our original UML design, the movement would have been blocky and grid-based and therefore, we strayed from our design to allow us to have all the movable game objects to move smoothly in the JFrame. Lastly, while programming the game, we realized that classes such as walls, hallways, and rooms were completely redundant since they could simply be stored as coordinates within the map and the Graphics 2D library handled a lot of the sizing of these features. So we decided to remove them completely and instead, had the map encapsulate these classes as game objects that would draw out either a walkway or a wall on a position depending on if the map interpreted those sets of coordinates to be a wall or not. We also additionally added some extra classes and draw methods that were not described in our UML that simply handle the UI of the game and give the game a prettier aesthetic. The remaining game objects more or less remained faithful to our original concept.

Design

Outside of aesthetics, the design of our game never changed. As stated above we started off with an office setting, but agreed to move on to more of a dungeon setting after experiencing limitations with time.

To break down our design, we will list each of our components that were required as of phase 1's criteria.

Player

The player's design was incredibly simple. The only functionalities we were interested in implementing was the player movement (in all four directions) and interaction with other objects (such as enemies, doors, traps, and rewards). We started by placing the player in an empty map and tried to get it moving. After that was successful we added walls and made sure the player wasn't able to walk through them. We next added an enemy, a door, and items which will be discussed in further detail below. Basically once the player was working, we kept adding more objects; focusing the design around how the player interacts with them.

Map

We knew right from the get go we were going to use a 2D array to build the map. Instead of hard coding a level each time, we implemented a way for a map to be built by reading in a text file using a Scanner. This was extremely helpful as it gave us a visual aid when constructing maps without having to run the game every time. Our first map was just a bunch of 0's and 1's inside the array. 0 represented open space, and 1 represented a wall which the player and enemy were not allowed to walk through. We would then paint squares on the screen, white for 0 and black for 1, and with that our first map was created!

As the development process went on, we decided to go the extra mile and use a tileset to represent the map. We made a tile class and found a free spritesheet online to give it that extra aesthetic. After some changes on how wall collisions worked with our new sprites, and a way to load in the sprites, our map class was finally complete.

Enemy

To implement the enemy class, we took the players code for detecting wall collisions and then implemented a way for the enemy to move towards the player in the shortest path possible whenever the player moved. We used a rectangle as a hitbox for both the player and enemy to detect collisions between the two. If the enemy collides with the player, the player loses their life and both are sent back to their initial positions.

Rewards, Traps, Doors, and Bonus Rewards

These are grouped together as they all followed the same principle in how they interacted with the player. For rewards, traps and bonus rewards, if the player walked into them something would happen. Traps would cause a negative effect by deducting points and removing a life. Rewards would give the player more score, and Bonus Rewards gave additional score as well as set a new respawn point for the player (like a checkpoint).

Doors were similar in walls in respect that players and enemies were not allowed to walk through them unless they were opened. We thought it would be cool to lock hidden rewards behind doors and add as an incentive for players to explore our levels. That idea also gave birth to rigging some doors with traps so that when they're opened they would spawn an enemy. Doors were a cool function to add since the player now had an option to open doors or leave them closed.

The Game States

In order to separate our game into states (menu, tutorial, level1, level2, etc.) we made a game state manager. What it did was draw whatever state the manager was currently in. It was super easy to implement and also made our game seem elegant. Now we could pause the game, give the player the option to restart it, or a level selection screen for example.

Important Lessons Learned

PULL AND MERGE FREQUENTLY! That is something we really struggled with the beginning of development. We also had an issue of forgetting to work independently on our own branches so it caused a ridiculous amount of merging conflicts in the middle of development which we wasted valuable time fixing.

We also found that communication was also key, it was quite hard once we were forced to work from home due to the pandemic but prior to that we were meeting up at campus 1-2x a week and would have a very constructive group session. Fortunately, discord exists and that ended up helping a lot with coordinating and dispersing work in phase 2.

We definitely learned a lot about programming in Java and using the Java libraries to create the game. A lot of the members have never used JFrame and JPanel to create a Java application before and moreover when it came to testing and using JUnit. It was especially difficult in the beginning when we were all trying to figure out how everything works, but through this struggle, we definitely learned the importance of teamwork and the efficiency of it. Considering that many of us are seeking co-op jobs that are all about working within teams, having the experience of programming in teams was one of the most important and invaluable things that we learned throughout this project. We appreciate the professor and TA for teaching us all of these techniques that have helped us build this game. The

experience and knowledge gained from this semester will be an invaluable asset towards our future growth. THANK YOU!! :)

Artifacts

To run the game we suggest you follow these steps. 1. mvn clean 2. mvn compile 3. mvn exec:java -Dexec.mainClass="com.mycompany.app.Game".

To run the tests on our game it's also quite simple. 1. mvn clean 2. mvn compile 3. mvn test. The tests will be under project/276_game/target/surefire-reports

After entering 1. mvn clean and then 2. mvn site javadoc:javadoc. The javadocs can be found under project/276_game/target/site/apidocs/com/mycompany/app/

Also here is a little link to our game for the features:

https://youtu.be/RA-F3cmQUHo