CMPT 276 - Phase 2 Report

Group 22:

Michael Plunkett (mplunket@sfu.ca)
Dina Zeng (dhz1@sfu.ca)
Sameer Hossain (sameerh@sfu.ca)
Salman Rafiei (salman_rafiei@sfu.ca)

Overview of Implementation:

The implementation of our game took several steps. First, we set up the classes based on our UML diagrams. Generally, this step includes creating the new class, writing attributes for it, and writing getters and setters. Then, we pieced together the relationship between each class by figuring out which classes depend on what and how the game flow would work. The game flow was simple at the beginning where the player would spawn and move to the exit; Gradually, the flow became more complicated as we added exit conditions and obstacles. In parallel to writing the classes and setting up the game flow, we also designed the user interface and implemented it within our program.

We make extensive use of git branches to allow team members to work in parallel and keep track of distinct features we were implementing.

As this was the first time anyone in our group was making a video game in Java, we drew inspiration from YouTuber RyiSnow's series "How To Build a 2D Game in Java'' to create the game's framework and learn the fundamentals of Java game development. We also have used his sprites during the initial development of our game just so that we didn't test our game with a bunch of different coloured boxes. Our intention in the next phase is to use his tile editor to import our own tilesets; however due to time constraints, we will be using open source game art and sounds for now (RyiSnow, 2022).

Adjustments:

- UI classes:
    - We added the UI classes to the program; they were missing in the initial design phase
- Add door class
    - We needed an starting and exit position for the game
- Enemy class is abstract
    - We never need to create an instance of the enemy class and all subclasses are based off of Enemy class with their own implementations.
- Renamed 'Character' class to 'Player'
    - To distinguish it from Java's java.lang.character class

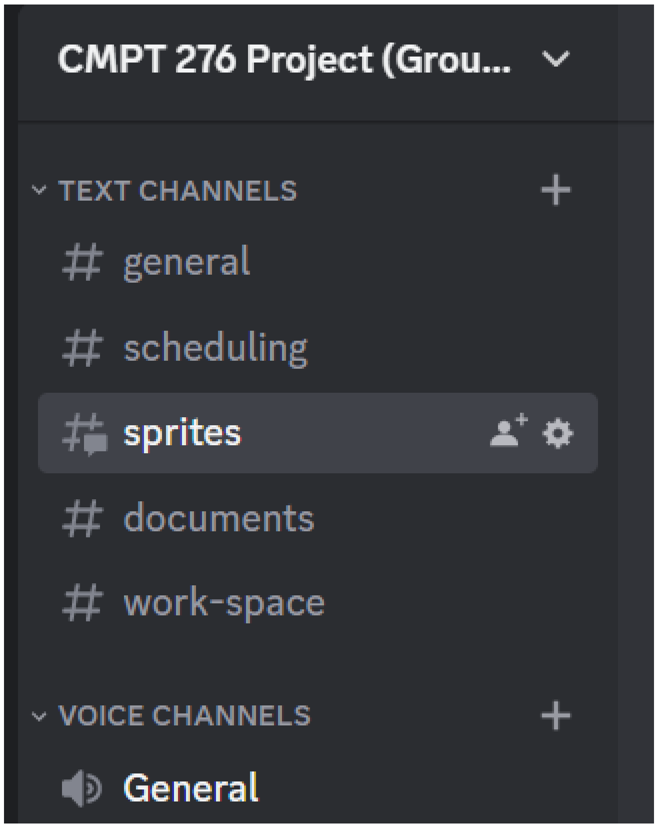
- Created a generic ‘Entity’ class for characters in the game to inherit from
    - This abstract class was created so all entities can be related
- Added collision detection and player hitbox
    - Created so we can check if the player hits an enemies and ends the game
- Add a door class which extends the item class.
    - it made sense that it was an interactable object that ends the level
- Implemented UI and game states
    - Needed an UI so users can play the game and game states so that we can start and end the game appropriately
- Implemented title screen
    - For aesthetics
- Added a randomized movement pattern type of enemy (Bats) to add variety in how enemies move
- Deleted Board, Game, Door (under GameMap), Tiles
    - These classes became obsolete in our implementation

Roles and Responsibilities:

- Initial structure of classes/methods: Dina
- Javadoc comments: Michael, Sameer, Dina
- GamePanel: Sameer, Michael
- Collision: Sameer
- UI: Sameer
- Items:
    - Keys - Sameer, Michael
    - Spikes - Michael
    - Potions - Michael
- Entities:
    - Player - Sameer
    - Bats - Sameer
    - Skeletons - Michael
    - Slimes - Michael
- Health and points: Sameer, Michael
- Map Design: Salman, including all of the below
    - Map layout
    - Item and enemy placement
    - Creating world text files
- Report: Dina, Sameer

We did not assign concrete roles and responsibilities throughout our implementation process. Each person worked on a feature or section that was connected to the code that needed to be done. The features were loosely assigned based on the availability of each person and their skill sets. Whenever a teammate needed help, another teammate would step in and work on the feature with them.

In terms of communication, we set up a discord server to keep track of our documentation, update our project design, and schedule meetings. We held meetings once a week to keep each other updated on our progress.



<u>External Libraries Used:</u>

- Sprites:
  - Author youtube: https://www.youtube.com/@RyiSnow
  - Note, might change sprites later to these: https://pixel-poem.itch.io/dungeon-assetpuck

● Java extensions	(javax)
○ Swing	(javax.swing)
○ ImageIO	(javax.imageio)

● Abstract Window Toolkit (AWT)	(java.awt)
● Utility	(java.util)

Quality of Code Considerations:

Initially, most classes had long lines of spaghetti code and needed to be reconfigured into their own classes. We wanted to keep our classes highly cohesive and loosely coupled. We strongly considered the 4 principles of object oriented programming: abstraction, inheritance, polymorphism, and encapsulation and implemented the principles consistently throughout our program.

For example, all of the Entities in our program inherit from our Entity class; they share attributes and methods where necessary so we could have highly cohesive classes. Classes that do not depend on Entity or should not be associated with Entity were separated from Entity to ensure that our classes are loosely coupled.

Additionally, all classes were appropriately grouped in directories: all entities and related classes (Player, Enemy) were grouped under “People”, all items and related classes were grouped under “Drops”, all game/map elements and related classes were grouped under “ GameMap”, the UI class was under “UI”, and all game flow classes were grouped under “app”.

To keep track of what was written and how each method works, we wrote JavaDoc comments for each method. This also has the added benefit of allowing us to understand each other’s code.

Generally, the principle for each class was “if this method/class/attribute can be associated with another method/class/attribute, try to merge them and delete the unnecessary code”. We wanted to ensure our code was as effective and efficient as possible and redundant code only made our program unreadable to each other.

Challenges:

The biggest challenge we faced during phase 2 of the project was implementing the game flow and understanding how each class in our project would communicate with each other. Since there are so many moving parts, it’s difficult to know exactly what is necessary for each class to have while keeping our program as a whole cohesive. In the end, most of our classes rely heavily on the GamePanel class, which is responsible for controlling the high level flow of the game with methods such as run(), update(), and changeGameState().

Another issue we faced during phase 2 was the design of the sprites. None of us had experience in designing sprites so we decided that it would be more efficient if we searched for a free stock sprite on the internet. Since we had a fantasy dungeon theme, our options for sprites were limited from the internet and we ultimately decided to reach out to a couple of artists to see if we could use their sprites with permission for our assignment.

Finally, randomly spawning and despawning the bonus reward drops turned out to be more challenging than expected. We chose to store these random bonus rewards in an ArrayList since it is mutable, however we are always adding, removing, and displaying elements from this array we ran into problems with concurrent modification exceptions. Since none of us have very much experience working with threads before, it was hard to debug this.

References:

1. Ryisnow. (n.d.). *How to Make a 2D Game in Java*. YouTube. Retrieved March 14, 2023, from https://www.youtube.com/playlist?list=PL_QPQmz5C6WUF-pOQDsbsKbaBZqXj4qSq