CMPT 276 - Phase 2 Report

Andy Lu, Andy Cheng, Martin Lau, Jackson Nguyen

Overall Approach

The beginning of the project was very disorganized. We came to realize our game was very under-specified in many areas, due to our lack of experience in general with object-oriented design, game development, and general lack of extensive team projects. This led to unwittingly developing 2 versions of our foundational game systems (game loop, rendering) at once.

Once we had these foundational systems, we became more organized, and were able to split up our work on more specific features and tasks. We continued to work with both systems, as we didn't know which was better, and eventually continued with just one of them as it developed more well-formed features.

As our design was under-specified, we generally added classes as we needed them.
Our team would meet 2-3 times a week, and for each member we discussed our progress since the last meeting, any issues we had, our plans for until the next meeting, when we would meet next, exchange implementation ideas, and any other topics we needed to bring up.

Each member would develop on their own branch of code, and during meetings we would go over and merge as we saw fit. In hindsight, a feature-based branch structure where each feature had a branch, not each member, would have worked better.

Modifications to initial design

Not put in:

- Board
- Button
- Slider
- Health
- Score
- Non-Moving Entity
- Consequences
- Structure

Various methods/variables in classes were not implemented

Justifications:

- ❏ Board: It was replaced by a GameInfo class containing the same attributes. More helper methods were added to it, it became more of a jack-of-all-trades class.
- ❏ Button: Unnecessary, functionality was provided by Java Swing
- ❏ Slider: Features requiring sliders were not implemented, so Slider was not needed.

- ❏ Health: Now it is an element of the Player class and is rendered by the Display class instead.
- ❏ Score: Same as Health.
- ❏ Non-Moving entity: The Entity class doesn't move to begin with, so there's no need to specify a non-moving variant.
- ❏ Consequences: Both the Reward and the Trap abstract subclass now directly inherits from Entity, adding a consequence class seemed unnecessary.
- ❏ Structure: The Door and the Wall subclasses now inherit directly from Entity, adding the Structure class felt unnecessary.

Newly put in:

Addition of many new Entity subclasses
(Floor, Ladder, Door, Sword, Exit)

Added classes to support Entities
(EntityFactory, Pathfinder, EntityStateHandler, Node, NodeComparator)

Added new classes for game systems
(Game, Display, KeyInput, MazeGenerator)

Added new classes to support game systems
(GameInfo, ImageUtils, UserInterface)

Justifications:

The game was initially very under-specified. The addition of many of these classes were a result of fixing blindspots in our initial design, and later as helper classes to support those fixes.

Classes that were added due to under-specification
- ❏ Game to handle essential game loop
- ❏ Display to handle rendering the game to the screen
- ❏ Pathfinder to handle pathfinding for Enemies
- ❏ EntityStateHandler to handle Player attributes (health, keys, score)
- ❏ KeyInput to handle taking user input
- ❏ New Entity classes to support basic game functions (displaying floors to the screen, for example)

Classes added to support existing classes
- ❏ EntityFactory added to more easily create Entities
- ❏ Node, Node Comparator to support Pathfinder algorithm
- ❏ GameInfo to support passing basic game data around (gameBoard, entityList)
- ❏ ImageUtils to support obtaining images for rendering
- ❏ UserInterface to consolidate information on the UI elements

Management Process, Delegation of Roles and Responsibilities

In general, we held Scrum meetings and followed the Agile development process. In our meetings, we talked about progress made since last meeting, talked about problems with code, and delegated tasks to do for the next meeting. We basically had one sprint for Phase 2, which was just implementing all of the requirements for the game. We also set up a Trello board outlining all of the to-dos for the game, and each one of us chose a to-do to work on and eventually finish. In this sense, the delegation of roles and responsibilities was largely self-motivated; we set up the roles to have, and each person chose which one they wanted to work on.

External Libraries:

Java Swing - Not an external library, but is the one we used for our GUI. We used Swing because it is a built-in library in Java, so it was easier to use. Also, Swing has good features for drawing onto the screen with the JComponent class.

AWT - also not an external library, but it was very useful. For rendering, we used the Canvas class of AWT to properly size the game onto the screen. For images, we used AWT's BufferedImage and image transformation classes for convenient import and manipulation of images. For user input, we used AWT's KeyEvent and KeyListener classes to obtain the user's key presses. For game logic, we used AWT's Point class to have a convenient object to store entity coordinates. All of these above use cases show why we used AWT to enhance our code; it simply has too many usages.

Measures taken to enhance code:

Some features of the game were restructured during development when their designs became less manageable going forward.
Example: EnemyAI was a class that originally handled pathfinding for Enemies. However, it required giving Enemy access to potentially troublesome information, and calculations were being done with each enemy that could be consolidated. Pathfinder was created as a restructuring of EnemyAI to address these issues.

Packages: packages were added to organize files coherently. Currently, there are 3: 'display', 'entity', and 'game'. 'display' and 'entity' are self-explanatory. As for 'game', it contains classes relevant to foundational systems, such as the game loop or user input.

Documentation: Efforts were taken throughout the project to document code. Comments were used to explain specific code segments and for their function. JavaDocs were used for many functions and classes to explain their general purpose, what they contained, and how they should be used.

<u>Biggest challenges faced during this project:</u>

A challenge that we faced during this project was time. We did not get to implement everything that we had planned for the game. We could not fully prioritize our time for this project because we are also taking other courses that also have deadlines for work that needs to be done or they have a part-time job that they need to be at. So, some days not everyone is working on the project because they have other work from courses that are due. Additionally, we had an assignment due for this course that was the same week as our phase 2 project. This set us back a lot of time because each individual in our group works at our own pace, so there were times where half of us were working on the assignment and the other half working on the project. Juggling between other classes, assignments and learning how to implement features to our game, took a tax of the limited amount of time we had received for completing this phase.

Another challenge we faced during this project was learning the language, and creating a game for it. None of us were really familiar with Java and Java Swing, so we had to self learn how the language works, and how we can use our knowledge to implement features. We spent the majority of our time not programming, but researching how we can program. For us, learning a new language is a hard task, and having to make a functioning game with a ton of features was a huge challenge that we all faced.