CMPT 276 Phase 2: Report

Group 8: Darren Jennedy, Setu Patel, Steve Lam, Samuel Wong

During phase 2 of our project, which happens between February 24 and March 24, we were tasked to implement our game, “How Not to Fail University”.

Overall Approach:

Our approach to implementing the game utilizes some of the principles of the scrum method. Our group has short meetings every 2-3 days, where we discuss what we’ve done since the last meeting, answer questions we might have regarding the tasks we were given, and divide up responsibilities. Our team keeps track of our progress and elements we need to implement, similar to a scrum backlog. What differentiates our method from traditional scrum methods is that we do not have a designated scrum master. We also do not have traditional scrum sprints and sprint reviews, because the project is not designed to have multiple iterations. In meetings, rather than the scrum master asking members about their progress, we independently show what we’ve done since the last meeting, and determine together what tasks need to be done.

Adjustments from Phase 1:

While the implementation of our game does not change any of the use cases drafted in phase 1, it does introduce changes in the software architecture from our initial class diagram on phase 1.

During implementation of our game, we realized that to maintain separation of concerns, we needed more classes than what we predicted in phase 1, so we added classes to help achieve that, such as adding the result and effect modules, to communicate the game result and game effect, respectively. We also added a few more modules, such as movable entities and positioning modules, to increase modularity. Furthermore, we realized that some of the classes we designed for phase 1 are redundant, such as the DirectionOrNone enumeration in the movement class in our original UML diagram, so we trimmed it out from our final submission.

Another concern we missed in phase 1 was designing the class diagrams for graphic rendering, such as image loading or GUI menus. We added those classes in our final submission, such as the gui module where we have our menus and the MainFrame class, a class to maintain the game frame. We also added sound effects to the game, so we had to create a SoundEffects class to generate them.

Management Process and Division of Roles:

As mentioned previously, our team has routine meetings where we decide on the division of responsibilities. Because of our frequent meetings, we do not have to assign one person to strictly finish one aspect of the game. Instead, each individual's tasks are determined during each meeting, such that we can divide the workload fairly. This management system ensures that each team member is doing all they can to contribute. Our management system also allows team members to collaborate on solving issues we might come into. Whenever a member has a problem they don't understand, they could bring it up in one of our regular meetings and other team members can help them solve the issue. This results in each member contributing in almost all aspects of the game, instead of just one.

That being said, our team does assign primary responsibilities to each member, which means that each member will put most of their effort in their respective roles. Our team assigns these responsibilities during meetings, and members will be assigned new responsibilities after they finish their previous tasks. Therefore, while each member contributes in more than one part of the game, they have their own individual responsibilities to attend to.

List of Members' Primary Responsibilities:

Darren : Enemy generation and movement logic, GUI menus (such as the main menus and ending screens), and crafting report
Setu : Collectable generation (rewards, punishment, bonus reward), player movement (together with Samuel), and sound effects
Samuel: Maze generation and player movement (together with Setu), and setting up game sprites
Steve : Game loop, graphic rendering, and general game architecture (eg: setting up position validations, classes to communicate game effects, etc)

External Libraries:

Our team decided not to use any external libraries, instead utilizing Java's AWT and Swing class, which is part of Java Foundation Classes (JFC). We learned that AWT and Swing was powerful enough for our needs of basic image rendering and GUI menu creation. The reason we choose not to use external libraries is because our team found that we didn't have enough time to learn to use new libraries, such as LWJGL, which requires learning the basics of OpenGL. As most of our team members are fairly inexperienced, we didn't want to delay implementing the game by learning new libraries. We also attempted to use a game engine named LITIENGINE (https://litiengine.com/). We were planning to use only their rendering library to help render graphics, but we decided not to use it due to the lack of documentation and tutorials.

Measures to Enhance Quality of Code:

We took extensive measures to adhere to good design principles and create low coupling classes. For example, we created separate interfaces to represent the state of the ending of the game, in order to reduce coupling on the game or the world classes. The interfaces are created to communicate information without introducing interdependence, to increase cohesion and reduce coupling.

We also strived to adhere to the principles of object oriented design, such as abstraction and modularity by using inheritance when necessary. For example, we abstracted our entities into two partitions; moving entities (MoveableEntity class) and non-moving entities (Collectable class), and utilized inheritance and polymorphism to reduce repeating code and distinguish unique features. We also made classes more modular by making sure that each class is only responsible for one problem, thus increasing our separation of concerns. An example of this is separating the movement algorithm for our enemy class and creating a new module, the movement generator, to handle deciding the correct movement according to the algorithm given. By using good design principles, we managed to organize our code which helped us structure the game.

Biggest Challenges we Faced:

The main challenge our team faced was learning how a game worked and adapting to new technologies. As most of our team members were inexperienced in programming, we had to learn a lot of information in a short notice. Learning how to use Git, Maven, AWT, Swing, etc. on the fly was a new experience for most of us, and having a limited time frame to learn and implement a whole game from scratch is a big challenge for us, especially during these times of remote classes. For instance, learning to manage merge conflicts and branches was taking more time than we'd have liked. Learning how a game worked, how the game loop needs to be incremented and how to update the game state was also very tasking for us. However, over time, we became more used to the task and learned to adapt to the situation and work with the time we had.

Another big challenge we faced was trying to improve the quality of our code. As this was most of our team members' first big project, we had some trouble deciding on the structure of our code and how to uphold good design principles. For instance, at one point, our team had a monolithic maze class, where the maze class would not only generate the maze, but also generate other entities, as the collectibles and the player. However, we learned to work together to find a solution to decouple the maze class by delegating the generation of the entities in the world class. In our final submission, we managed to have low coupling in our classes to create a good separation of concerns.