Phase 2 Report

Overall Approach

In the first phase of our project, we conducted multiple planning sessions to outline our objectives and goals. As we progressed to the implementation phase, our team assigned specific roles to each member based on their areas of expertise. Josh, Rahul, and Tawheed worked on developing the engine and bridge between the engine and UI, while Jay was responsible for designing the UI. We adopted the Agile SDLC model to manage our project, which allowed us to plan and prioritize the necessary tasks using a TODO list. Our team implemented the features incrementally, testing them before concluding each iteration as successful. This approach enabled us to continually adapt to changes and incorporate feedback from stakeholders, resulting in a more refined and optimized final product.

Adjustments/Modifications from Phase 1

One major adjustment we had to change from phase 1 in our UML diagram is where the move functions for the player and enemies would be located. We had initially placed them in the entities package because it made sense that the entities themselves would control their own movement, however during the beginning of implementation, we realized it would be better to have the controls and movement in the state package, so only one instance of the map had to be initialized, and we wouldn't need to pass the map object around as a parameter. This would reduce coupling and increase synchronization between enemy and player movements. Other than that, the main outline of our Use cases and UML diagrams have remained pretty much the same. Inherently with planning a big project like this we could not plan out all the functions at the beginning, so the UML diagram, especially for packages like State, are very lackluster in comparison to our actual implementation, however the structure of the package remains. We also were able to settle on the theme of which the game will follow. In Phase 1, our mockups included generic images that we grabbed off google images as placeholders as we didn't know what direction we wanted to go in aesthetically and thematically. We then stumbled upon an asset pack of a dungeon crawler / catacombs theme that really fit the vibe of what we were going for, and so we decided to use it.

Management Process

During phase 1 we split the games architecture into 4 packages, state, UI, map, and entities. Thus with 4 members, we split it 1 person per package. Obviously this doesn't mean that we are limited to just our package, but rather it is encouraged that we add and update other packages as we think of new and better implementations and methods. An example of this was with the entities package. The first iteration had implemented a Point variable from the map package for each entity, and was setting the coordinates of it through x and y getters and setters. It was then updated by another member to be passed in a Point object, and to simply set this.location to the new point. The Entities and Map package were the first two to be finished, so focus was then divided up between the State and UI package. Jay took on the UI package by himself, while Tawheed, Josh, and Rahul focused on the State package.

External Libraries

As of right now we have only implemented one major external library and that's for implementing the GUI for the game. For the GUI we decided on using JavaFX as the library of choice. We had a couple of options including Swing, AWT, slick2D, and LITIengine, all of which we discovered could be used to implement a 2D game like how we intended, however we decided upon JavaFX. The reason we decided to stick with JavaFX is covered in the biggest challenges section. Other supplementary libraries we included in our game were for the back end logic, which were libraries such as ArrayList, List, and Random. We used ArrayLists and Lists in order to store entities, due to the fact that they can store non primitive data types, such as the enemy, player, trap, and reward classes we designed, and they are resizable, letting us increase entity count depending on levels. We implemented the Random library to aid in enemy movement. If the player is close to the player we have a system in place where they "track" the player and move towards them, however, if the player is out of the enemies range, they simply move in a random direction. We also used the random library for maze generation, with Tawheed using Prim's randomized algorithm for generating the structure of the maze, and placing all the entities initially, that being the enemies, the rewards, and the traps.

Measures Taken to Enhance Quality of code

We have taken several measures to enhance the quality of our code. Following the notes we saw in class about Object Oriented Programming and Design Principles, we made sure to implement as many Design Principles as we could. We made sure to use enums to improve readability and have a standardized way of referencing the type of enemy and cell we would be dealing with. We also implemented Interfaces, however simple they might be in our implementation, to prevent code duplication and have similar classes follow a blueprint for what they are supposed to do. We also strived for a high level of cohesion, grouping all the functions that work on the same data together in the same package, and in the same file. The game.java file we have acts as the driver, instantiating the game, placing enemies, and controlling movement with collision detection. We believe we also have low coupling as data isn't being passed unnecessarily between packages. We also followed and implemented the Law of Demeter so functions weren’t violating the single dot rule when doing inner function calls. I believe that thorough planning in Phase 1 was successful as we were able to systematically plan out packages and classes before starting implementation, leading us to very little to almost no backtracking in trying to remedy any issues with implementation or code quality.

Biggest Challenges

We faced a few challenges throughout this phase but nothing massive. One thing that was unfortunate was that a team member fell ill for roughly a week, so communication, collaboration, and progress on their part was temporarily halted. The other teammates luckily picked up the slack and helped continue progress with the game. Speaking of communication, in order to avoid any challenges, we have been holding almost daily scrum meetings over discord to communicate any issues we might be having, to cover what we have been working on, and what we might need help with implementing.
Something that posed an issue to all members, not specifically with the game, was version control. Being new to using git and not having any prior experience, we faced a couple of hiccups with branches. As this was our first collaborative project, we initially pushed everything to the master branch, but quickly ran into several merge conflicts and other issues, resulting in a chaotic mess of trying to figure out how to remedy the issues and backtracking versions. In order to address the numerous issues we were running into, we realized the need for separate branches for each developer's work, and obtaining confirmation before merging. This approach helped to streamline the development process and minimize the risk of errors and conflicts in the codebase. However, it did prove difficult at times as master wasn't constantly updated, so trying to pull certain files from certain branches and keep everything working at the same time did end up being a little bit of a nightmare, but these issues are resolved simply from gaining experience with version control. There was a big disconnect as UI and game logic was implemented separately, thus, issues arose when trying to bridge the gap between them and connect the two. As we were using JavaFX to handle inputs from the keyboard, it posed a challenge on how to transfer the keystrokes over from the UI package into the State package so Game.java could use the inputs to move the character in the actual game implementation on the 2d array of cells.
JavaFX proved to be a challenge we weren't expecting due to the fact that we had virtually no experience with it prior to this project. After a lot of debate and research, alongside recommendations from fellow peers, we decided that it was the right choice due to an easier learning curve compared to other graphic libraries. In a class where it's our first time using version control and working in groups, we opted for an easier library to lessen the stress accompanying implementing this game. As for learning JavaFX, we had to learn how the library operates, with its whole hierarchy system. Due to our inexperience, we would often change scenes everytime we wanted to update the view instead of simply clearing the canvas and redrawing, which would be much more efficient. Also, JavaFX needs to be initialized on the main thread, so some work was needed to make sure that works. We faced an issue with integrating UI and Game logic, something we didn't account for in the UML, and thus we had to create a new package

called EventHandler that acted as a bridge between what is essentially the front and back end of the game