Phase 2 Report

High-Level Overall Approach

The game has a main `Game` class with a **"game loop"** - a constant while loop that **(1)** updates the game information, and **(2)** draws all the game assests on screen. Each "tick" is one game loop iteration. It creates this game loop by using Java Threads. Classes (main character, enemies, rewards, board, GUI) each have a draw and update method(s) that are called by `Game`.

The `MainCharacter` is controlled using `WASD`, which is programmed by using Java's `KeyListener`. **Collision** is handled by `CollisionHandler`. Each game entity has a "hit-box": if the hit-boxes of two entities intersect (e.g. moving enemy and wall, main character and reward, etc.), then this stops movement/collects the reward/etc.

Moving enemies (`Slime`s) have very **simple pathing**. They move towards shortest *x*-distance to the player, then towards the shortest *y*-distance to the player.

Rewards (`Star`s & `Coin`s) have their positions set by the `Game` or `Spawner`s.

The board (`Map`) is loaded from a .csv file. Each number corresponds to a `Cell`. It is surrounded by "void" from its four sides, and has interior walls to mimic a maze-like cave.

The Screen, GUI, and other assests is drawn using Java's Swing, primarily `JPanel` and `Graphics2D`. The game window also uses `JFrame` from the same library.

Modifications from Initial Design

New helper methods were needed for classes to implement unforseen functionality; For example, `Game` needed a way to create a window and format its timer to display the time on-screen; `Entity` needed more methods to interact with its hit-box for collision detection; `Map` needed more methods to create and interact with its cells.

New fields were needed or **pre-existing fields** were **modified** to fit the system better. `Entity` uses a `HashMap` instead of an `Array` to store its image frames to easily access them with keys like "up" or "down" instead of with indices; `Map` creating a 2D-grid of `Cell`s slowed the game significantly, so the design was changed to holding a 2D-grid of `int`s that's parsed and used to index a 1D `Cell` array; `MainCharacter` needed a field to keep track of the regular rewards (`Star`s) in order to exit the map.

New classes were needed. One `Screen` class was split into two: `Screen` which was primarily screen settings, and `GUI` which handled drawing the win, lose, and score displays. `SlimeSpawner` was created to further abstract `MovingEnemySpawner`.

Management Process

Roles and responsibilities were **flexible**. Team members were open to work on parts of the game that were free. Quick communication was made to know who was doing what.

Our Half-way Deadline

Initially, our goal was to create a correctly moving main character with movable enemies. However, we managed only to create a correctly-moving character. This was likely due to troubleshooting Maven and an initial lack of goal setting.

Plan and Milestones

Our plan was **reactive**: a team member would choose to work on a class that was avaiable, make their separate branch, and communicate to the team when it was okay to merge. Any team members who were merging at the same time would coordinate who would merge first and who would deal with possible merge conflicts. We used this approach to complete our two main milestones, our half-way deadline above, and the game. Smaller miletones were made naturally from the worked taken on by teammates.

External Libraries

No external libraries were used to, for e.g., create the GUI or parse a file. All libraries are internal (from `java`/`javax`).

Quality Measures

We **tried to ensure low-coupling** by limiting the dependency/references of the main `Game` class by other classes, for e.g. having the `MainCharacter` accessing the `Map` directly rather than through `Game` since the main character does not need to interact with several main public methods of `Game` such as those that handle the game loop. In general, we tried to limit class references by only having certain methods as an interface between them, for e.g. most entities have a draw method that takes `MainCharacter` to draw them relative to the prayer, but has no `MainCharacter` field.

We **aimed for high-cohesion** by trying to have classes only "do one thing", like `Game` only controlling the game loop, and `Map` only loading and drawing the 2D-grid.

Biggest Challenges

Since our plan was more flexible, it was susceptible to **vagueness in tasks** and **sensitive to lack of communication**. When this happened, our main solution was to trying our best to stay understanding towards each team member's current situation, which ultimately helped us move forward as a team.