# Phase 4 Report

This report discusses the final game and reflects on our experience as a team.

## The Game

In this section, we provide a **brief description** of our game, **how our game evolved** throughout the course of development, and what **lessons we have learned** along the way.

### Game Description

In our game, you play as a pink-haired girl who awakens and finds herself in an underground mine. Unbeknownst to her, she has been transported into a world different from hers. To escape the mines, she must collect all five star pieces that when combined together, can allow her to leave and return to her world. But there are roaming slimes and magical traps that she must carefully maneuver around to not let her chances of escape slip away from her. Along her way, there are foreign coins scattered around the mine floor that she can pick up, which might serve useful if she makes it to the outside world.

### Faithfulness to original design/plan

We stayed true to our main idea of being inside an underground mine as the player traverses through it. Conceptually, however, we moved away from a general mining theme of collecting ores and diamonds to a more general fantasy theme inspired by Japanese role playing games (JRPGs).

In terms of software design, we generally **did not stray away too much from our original UML class diagram**, closely following our class hierarchy. For example, the class `Entity` -- from which we derived our `MainCharacter` class or `Rewards` class -- was powerful in abstracting the general notion of a "game entity," so we found no need to change its inheritance structure. We can say similar for the `Spawner` class which sets the "spawns" or positions of its entities. However, we did found that we need more concrete subclasses to spawn specific enemies, which we did not account for in the UML diagram, only having abstract spawners. Hence, this led to the creation of the `SlimeSpawner` , `TrapSpawner` , `CoinSpawner` , and `StarSpawner` . Moreover, during testing, we found that sometimes we needed to extract classes from other classes in order to test certain methods. For example, to test how player key input interacts with player movement, we created a new class `PlayerMovementHandler` to abstract key inputs from movement to make testing easier.

However, we did **add or update many class methods and fields**. This is because some logic we planned for before implementation did not work too well while developing. For example, the `Screen` class originally had a `displayWinScreen()` and `displayLoseScreen()` method, but was scrapped for an entire new `GUI` class that would handle all the UI-related drawing to screen. The functionality of drawing to screen required more work than originally thought -- the logic could not fit within two methods. Furthermore, we added to the `Spawner` class methods to `update()` and `draw()` the entities it spawned to easily control those entities as the game ran. These methods also helped in randomly spawning `Coin` objects, our bonus reward, which we did not fully consider how that would be implemented initially. As a last example, how we initially planned image loading for `Entity` objects proved to be confusing when trying to get certain images when kept in an array. We refactored this by using a HashMap instead to use meaningful key phrases like `up1` or `idle_left` to get specific images.

Overall, we followed the class inheritance structure of our UML class diagram quite closely, only adding new subclasses when necessary or updating key fields and methods when poor or confusing design was found.

### Reflection and lessons learned

In terms of team dynamics, we were able to **develop the game relatively smoothly** as we overcame bugs, initially poor design, and branch management. As mentioned in our Phase 2 report, we each worked on separate branches and coordinated merges, and that ethic continued throughout the testing phase and debugging/refactoring phases. However, we still had some **issues in being somewhat vague** in when to complete our short-term goals and the goals themselves. Often, it would be a teammate picking up whatever work was needed to do, communicating that work to the team, and working on it in a separate branch. Though this was rather adaptive, there usually was not clear deadlines we set for ourselves as a team other than the actual project deadlines.

So we learned the importance of **time management**, specifically setting and readjusting expectations of completing tasks accordingly. We at least tried to communicate the times and days we would be available, but we could have been more specific in what we expected to do and complete by an estimated date, which would likely help in pacing ourselves.

In summary, our team was able to work together with **overall consistent communication**, with the caveat that **our time could have been managed better**.

## Tutorial

We created a **video demo** that can be watched here.