Group 16 Phase 2 Report

Approach to implementing the game

We took an object-oriented approach to implement the game. The modularity allows us to create more understandable and maintainable code. It also allows us to more easily divide work among group members by assigning different modules to each member to implement. We implemented the game through several increments that each adds more features. This allows for faster development and easier debugging of each feature.

Adjustments and modifications to the initial design of the project

We added some classes to handle map generation, sounds and sprites. We also made some classes singleton to make some features more easily implemented. Although this decreases modularity and increases coupling, fields and methods inside a class are still highly cohesive.

The management process

We had several in-person meetings to determine what we would be implementing each week and what we struggled with implementing the last week on our own. This allows members to catch up with what others are working on and ensures modules would work together.

The division of roles and responsibilities

When discussing the division of roles and responsibilities, we came to a conclusion that each member should work on features that they feel most comfortable; some members have more experiences on Java; other members are more comfortable on other aspects.

Joshua: Implementing UI components (Reset button, exit button, time elapsed), menu state, setting up maven configuration
Paolo: Implemented game engine, running and win states, gameboard, main character, rewards, punishment, scoreboard, audio and sprite helper classes. Animated sprites for main character, enemy, reward.
Leon: Implementing enemy’s searching algorithm.
Zac: Implementing classes for map generation and level advancement.

---

external libraries used (briefly justify the reason(s) for choosing the libraries)

1. Import java.util.ArrayList and java.util.List to implement depth first search of the moving enemy.

2. javax.imageio.ImageIO, to read image assets, this is the most frequently mentioned library on the internet, and therefore it's easy to look up examples and error solutions.

3. javax.sound.sampled.Clip, to read sound assets, this library is easy enough to fulfill our needs without the complications of reading complex documentations.

---

the measures to enhance the quality of your code

1. We create classes such that the contents are highly cohesive to ensure high maintainability.

2. We use interfaces and abstractions whenever possible to provide loose coupling and increase code reusability.

3. We used design patterns like singleton and factory methods to solve problems as they are proven practices.

4. We would review each other's code to provide a different perspective on how something could be implemented differently with a higher quality.

---

the biggest challenges you faced during this phase

1. Time commitment.
    - Since every team member has their own commitments in other classes, it's difficult to find the time to actually code this project. We also had difficulty arranging meetings to catch up on what others have coded and agree on what to implement on the next increment.

2. Underestimation of the project complexity.

- When reading the project description, we thought the project was easy to implement, it turned out that it’s harder than we thought and we overpromised on certain features that require substantial time commitment.

3. Java coding experience.
    - As mentioned previously, some members have never code Java before and are not familiar with object-oriented design. As a result, more time are devoted in learning and getting used to Java rather than developing the code

4. Git experience.
    - Some members are not highly experienced in Git so they have to spend time getting used to using it.