Phase 2 Report

Group Four Members:

- Ana Premovic
- Regan Li
- Matin Keivanloo
- Zhaoyue Yuan

Topics Discussed:

1. Describe overall approach to game implementation
2. State & justify adjustments & modifications to initial design of project (shown in class diagrams and use cases from phase 1)
3. Explain management process of this phase, and division of roles and responsibilities
4. List external libraries used (ex: for GUI) & briefly justify reasons for choosing libraries
5. Describe measures taken to enhance quality of code
6. Discuss biggest challenges faced during phase

Approach to Game Implementation:

- Discuss project scope and git to ensure a uniform concrete understanding of expectations
- Dividing & assigning tasks to each group member, organize by importance
- Refer to UML diagram for implementation of required classes
- Discuss and agree upon an IDE to minimize intangible issues (minimize variables to limit avoidable time spent problem solving - allocate to implementing)
- Emphasis on communication whenever issues or other topics arise.

One of the first things we did as a group was organize ourselves and split up individual tasks, so that we could all begin implementation right away without confusion of what each member was working on. Then we discussed requirements of the project as well as what was expected of us as a group so that there would be no discrepancies in our overall understanding, which could be time wasted that would be better utilized towards implementing classes. In essence, we agreed that it was of utmost importance that our group was organized and in agreement with elements such as project scope, roles and responsibilities, the IDE utilized, and implementation methods so that we could reduce potential conflicts that otherwise would have been easily avoidable given proper, thorough, planning. In terms of the game implementation itself, we all referred to our UML diagram created during phase 1 to deduce which classes required to implement first, in order to build our game off of a solid foundation. Fortunately, a couple of group members possessed prior experience working with the Java language, so they were able to suggest implementation methods and external libraries that we should make use of.

Adjustments & Modifications Made to Initial Design:

- Merged gameLogic and Screen classes into a single class
- Added KeyHandler to take both WASD and arrow keys as character inputs
- Added Tile and TileManager classes
- Did not add quicksand as obstacle/static enemy
- Did not add evil bunnies as obstacle/dynamic enemy
- Did not implement “glow up” feature

Explanation of Adjustments:

When considering the alteration of the initial design, we made a significant amount of changes. In fact, the only classes we added were the Tile and TileManager classes, which replaced the “2DBoard” class in the UML diagram of phase 1. As a group, we decided that having a “Tile” and “TileManager” class would provide better flexibility if we were to incorporate a variety of different tiles into the “floor” of our game. We also implemented a KeyHandler class, which takes both WASD and arrow keys as inputs in order to move the main character around — catering to users of all kinds, regardless of preference of input. We decided to do away with the idea from phase 1 where we would add a glowing effect behind the score/carrots collected, though we thought that would be a distraction, so we did not implement the glow effect. We also initially planned to have multiple static and dynamic enemies in lava and quicksand then evil snakes and bunnies respectively. However, we decided to discard the implementations of quicksand as well as evil bunnies because we decided that the game board would become cluttered and have too much going on, which takes away from the immersion of the game.

Management Process - Division of Roles & Responsibilities:

In our first meeting, we immediately handled the division of roles & responsibilities in order to begin work promptly while minimizing confusion as well as maximizing group productivity. During the meeting, we decided that with our varying class schedules, it would be more efficient to stay in touch online rather than commuting out of our way to meet on campus, which can take away from valuable work time. So, from then on, we used Discord as our main form of communication for discussions, updates, what was being worked on, ideas, suggestions, and any issues that arose during phase 2. Whenever a member began working on implementation or when one pushed new changes to git, it was communicated via Discord. Everyone responded punctually when needed, and our meticulous communication as well as organization is what allowed us to stay on the same page and sail through phase 2 with ease.

- **Ana**
  - Created & worked on Screen class
  - Created & worked on RewardDisplay class
  - Implemented display of game rewards
  - Implemented game loop
  - Worked on CollisionDetector class (collisions with barriers & rewards)
  - Worked on KeyHandler class
  - Created & worked on MainCharacter class
  - Worked on Board class
  - Implemented score keeping
  - Implemented winning and game over logic
  - Writing & editing report
  - Found main character, enemy, barrier, & trap images

- **Regan**
  - Created & worked on KeyHandler class (key press, key released)
  - Created & worked Tile class
  - Created & worked TileManager class (getting tile images from resources)
  - Worked on GameRunner class
  - Worked on Screen class
  - Worked on Enemy class
  - Worked on MainCharacter Class
  - Writing & editing report
  - Found bonus reward & barrier images

- **Matin**
  - Worked on MenuLogic class
  - Start menu
  - Game over menu
  - Score bar
  - Time elapsed tracker
  - Writing & editing report
  - Found barrier, cave background tile, reward, & start menu images

- **Zhaoyue**
  - Created & worked on Enemy class
  - Implemented enemy pathfinding algorithm
  - Worked on CollisionDetector class
  - Writing & editing report
  - Found background tile image

External Libraries & Justification:

```
import javax.imageio.ImageIO; // Used to get images and display them (background,
tiles, character sprites, etc
import javax.swing.JButton; // Used to implement start and exit buttons (GUI)
import javax.swing.JFrame; // Used to display game window
import javax.swing.JPanel; // Used to display a menu bar that allows exit of game
import java.awt.Image; // Used to draw images to game window
import java.io.IOException; // Used to catch exceptions when we use try statement
import java.awt.Graphics; // Used to draw graphics & images to game window
import java.awt.event.ActionEvent; // Used to track actions as such as a button press
import java.awt.event.ActionListener; // Used to handle events that are tracked
import java.awt.Dimension; // Used to get and manipulate size of component/image
import java.awt.Font; // Used to manipulate fonts that appear in the GUI
import java.util.ArrayList; // Used to create a list of enemy
import java.io.InputStream; // Used to read files such as an image or audio
import java.awt.event.KeyEvent; // Used to track key events (key press, release, etc)
import java.awt.event.KeyListener; // Used to handle key events - perform actions
import java.io.InputStream; // Used to represent stream of bytes - reads data from
input source
import java.io.InputStreamReader; // Used to read bytes from input stream & convert
into characters
import java.io.BufferedReader; // Used to read data from character input stream,
buffers characters to provide an efficient method of reading data such as arrays
import java.awt.image.BufferedImage; // Used to show & manipulate images in GUI
import java.io.File; // Used to manipulate files & their paths
import java.util.Random; // Used to generate random numbers (random X & Y for rewards)
```

Measures Taken to Enhance Quality of Code:

We did not make many changes to enhance the quality of the code, but the adjustments we made created a substantial impact. First of all, we realized that it would be more sensible to merge both the gameLogic and Screen classes into a singular class, since it provides much easier access to all the methods compared to having them in separate classes. In terms of displaying object sprites, instead of displaying object sprites in their own respective classes, we decided to handle them in classes made specifically for display, in the ImmovableObjectDisplay and Screen classes. In general, measures that we took to enhance the overall quality of our code were to incorporate abstraction in terms of keeping most of the asset displaying in its own class, as well as keeping the enemy algorithm in its own class. In order to maintain and preserve simplicity within our software, every class (except “Screen”) that we created possessed only a single purpose, limiting potential confusion.

Difficulties Faced During Phase 2:

General Challenges:

- Git issues (merging, pulling, pushing)
- Slight difficulty finding sprites that fit aesthetic of our game
- Difficulty writing pathfinding algorithm — needed to choose between BFS, DFS, A*
- Restart button causing weird behavior such as movable characters having double speed
- Index out of bounds error due to turtle going off screen

Explanation of challenges faced:

Due to our group's lack of experience in working collaboratively and with git, we inevitably faced a couple of issues during phase 2. The most common issues that we would run into while working with git were errors merging, pulling, and pushing — specifically pushing and merging branches. However, we were able to resolve these common issues through thorough reading of git documentation online. Another miniscule issue that we faced during phase 2 was that there was slight difficulty in finding object sprites that all fit each other as well as the aesthetic of the game that we were targeting. It seemed easier than we thought, but we ended up finding decent ones, and were able to make slight alterations to them in photoshop in order to suit our own requirements. In terms of the actual writing of code, one of the dilemmas that we faced was which pathfinding algorithm to implement for the snakes, from Breadth First Search (BFS), Depth First Search (DFS), or A start (A*). We ended up deciding on implementing the A* algorithm for the enemy because both BFS and DFS were easier to implement, however their time complexities were high, so we decided to go with the A* algorithm, which is more complex, but efficient. One of the more grueling difficulties during this phase was dealing with weird behavior that our restart button was generating. Occasionally, when we would restart the game, we noticed that both the turtle as well as the snakes were moving at double their set speed. It was an arduous debugging effort, because we were uncertain about the underlying cause. However, we did suspect that it was a result of two game threads running simultaneously as a result of the restart button. Through some trial and error as well as removing the "startGameThread" methods from the game over screen method of the restart button implementation, we were able to overcome the issue. The last of our notable difficulties is also related to the restart button, where the "KeyHandler class" would be mishandled on restart, ceasing it from functioning properly. Upon restarting after a win, none of the movement would be functional unless the up key was pressed first, triggering the upReleased boolean. We realized that this was because our game thread would end when the turtle touched the ocean (exit) tile, however the upPressed boolean would still be true. To remedy this issue, we made the ocean tile solid, which prevented the turtle from crawling off the screen upon a win.