Phase 2 Report -- Group 13

☐ The overall approach to implementing the game

When starting implementation, we started off by delegating who would do which classes and we did them separately. Then we checked in every so often to see where we were all at and discuss what our next steps were going to be. As a group, we usually collaborated to solve any issues and tried different methods to make progress. As the deadline approached and we were getting near to completing the code, we discussed more with each other to fix any issues and make cosmetic decisions. Everyone had some contribution to coding but for the members who were weaker in that aspect, they contributed more in ways that didn't involve coding(e.g., this report). We used JPanel to implement our game interface and drew images in different directions for animation purposes. We used 0,1,2 to represent our map. One represents the walls/barriers where the player is unable to walk across. Zero represents the tiles that the player is able to walk on and two represents special tiles; the end cell that the player walks into after collecting all regular rewards. Using our UML diagram as a guide, we made the planned classes. As we progressed through implementing what was on the UML diagram and researching different ways of implementing some methods, we changed, removed, and added classes and methods so that the code was more organized and flexible.

☐ Adjustments and modifications

- added a Trap class that wasn't included in our diagram due to oversight; implements the required functionality of punishment.
- removed GameRunner class; deemed unnecessary as the initially planned methods for this class was put into other classes
- added Input class for taking keyboard input from the user; was originally planned to be within the Gatherer class
- added CollisionChecker class to check if the main character can access certain points and block the player if not allowed.
- Renamed Character class to Entity
- Instead of Entity being a class to be implemented, it became a class to be inherited
- added AssetSetter class that creates and sets objects and entities at certain locations on the map
- added Background class and BGManager class. It is related to creating and displaying the background
- added Object class, which is a class that's inherited by the regular reward, bonus reward and trap
- Renamed Board class to GameWindow
- added Tile_Manager, EndTitle, and Game_Tiles; for creating and displaying the current view of tiles to the screen
- add worldmap.txt to store the map of our game
- add gameStat to display different ending scenes

- add gameState class to implementing pause and resume states of the game
- Overall added/removed methods/members of classes as we saw fit. They were either not necessary or we thought it would be better to implement the classes in a new way different from how we originally planned

☐ Management process and division of roles and responsibilities

- Our group first met up on Feb 22. We help each us to install Maven and divide the roles and responsibilities:

  Maggie Tan -- Trap(punishment) and Scavenger(moving enemy)
  Utsav Anantbhat -- GameWindow
  Swara Desai -- Gatherer(main character)
  Hongrui Qu -- BonusResource(bonus rewards) and RegularResource(rewards)

- On March 05, We revised some of our design in phase 1 and continue to finish the functionality.
- Over the week, we hold regular meetings to solve any difficulties and share our progress.
- On March 12, We check with the code and worked on the report. We finalized the remaining things that need to be implemented and the members that aren't as strong in coding, continue on the report
- On March 19, final check before the submission deadline.

☐ External libraries

java.awt: We use this library to colour the appearance of objects in the game interface and implement all the graphics.
javax.swing.JPanel: We use this library to implement our game interface.
java.io: We use this library for exception handling and file management
java.text.DecimalFormat: We use this library for formatting numbers
java.lang.Math: We use this library to calculate the absolute value of the distance between scavengers and gatherer
java.util.Scanner We use this library to load our map data.
javax.imageio We use this library for images processing.

☐ Method to enhance the quality of code

- We merge changes to the main branch constantly and commit helpful comments.
- We write comments within different class files that help others understand the purpose and functionality.
- Each of the class files is retrieved by at least two of us to ensure the correct functionality.
- We maintain consistency in naming parameters or functions.

- We keep code as simple as possible

☐ Biggest challenges

Since we have little practical experience working with Maven along with git version control, we could occasionally experience technical issues that prevented us from executing git commands, such as push and pull.
There were some functionality overlaps between classes (code duplication) that could have been avoided by communicating more clearly in the initial stages of coding.