Our Implementation approach followed the evolutionary process model. We start off by implementing the game panel and the main function to create the window pop-up for the game and continuously add on implementations such as the cells of the map, the entities (player, objects and enemies) and the UI class implementations. We check the functionality after each class addition and use pre-existing libraries for purposes such as creating threads, panels, and graphics.

In terms of initial design, we added the use cases for picking up rewards and running into enemies and the results of points being changed. We decided to not implement the login use case as it will require additional storage of login information in the game and instead just instantiate a new game each time. We added new classes to help manage the composite objects such as CellManage which creates the map with different Cells. We also used abstractions such as SuperObjects and Entity to group similar attributes of the classes that extend them.

For division of work, Yizhou Lu offered to work on the code for the midway progress deadline as he was not present during phase 1. Due to timing and potential merge conflict issues, the codes were completed by the two members Haoxin Wan and Janet Wang. Haoxin Wan worked on the initial GamePanel, Main, Player, Collision and Object related classes while Janet Wang worked on the Cells and UI classes and updating the other classes to use these functions. Lin Li worked on the javadoc and comments for all the codes while Yizhou Lu researched information for the libraries we imported and used in the codes and did an outline for the report. The final report is written by Janet Wang.

The following built-in packages were used and their main function are listed:

- java.io.IOException(Gives checked exception that indicates a problem);
- javax.imageio.ImageIO(Read and Write in Image data);
- java.lang.Thread.sleep(When a processor is finished, let it sleep and give cpu to another processor);

java.awt.*(Use the package Awt);

1. java.awt.image.BufferedImage;(Java awt image is going to create and modify image, bufferedImage is an image with accessible buffer data)

2. java.awt.event.KeyEvent(Generate event);
3. java.awt.event.KeyListener(Listener interface to receive keyboard event);
4. java.awt.Color(Control Image Color);
5. java.awt.Dimension(Control Image Dimension);
6. javax.swing.JPanel(Organize components and layouts);
7. java.awt.Graphics(Generate Graphic);
8. java.awt.Graphics2D(Gives control of 2 dimension of the graph);
9. java.awt.Rectangle(Specified x and y coordinates' spaces);

javax.swing.*(Use the package Swing);

1. javax.swing.JPanel(Organize components and layouts);
2. javax.swing.JFrame(Foundation of creating Graphic Java Application);

Our codes right now have some redundancy in implementation and should be further simplified through uses of methods and function calls to decrease the number of lines of code. We created different classes for managing different functions as well as separating concerns into separate packages allowing modularity to increase readability. The main game is grouped within the GamePanel class that uses most of the classes and packages and this structural set-up will increase the testability of the code in future phases as the codes are modularized. The separation of concern will also make it easier to reuse parts of the codes for other functions. For example, additional entities (like NPCs) may be added in the future to expand the game. The overall set-up allows for flexibility by making it adaptable and applicable across different scenarios.

The biggest challenge during this phase was trying to gauge our workload and dividing the work and responsibilities properly. The overall timing was hard to gauge as well and a lot of the workload ended up being done during the last few days before the deadline. All of our members are relatively new to Java so we struggle a lot with learning and coding the assignment at the same time which further increases our own expected workload. Working concurrently with other members also proved to be a struggle as there are often merge conflicts when we update codes in the same file. In terms of timing, we feel the need to incorporate a schedule for future phases to avoid the timing challenge we encountered.