Phase 2 Report

Group Number: 5

Group Members:
Joshua Kim
Jad Alriyabi
Marco Lai
John Sweeney

Date of Submission: 19/3/2022

Project Implementation Approach:

Our overall approach to implementing the game involved us working in teams of two based on the Scrum Process Flow. We worked individually but collaboratively on different aspects of the game. Our phase 2 report provides detailed information on how we collaborated. Our team had a back and forth discussion on the best design parameters for our game. Individual game functions are contained within respective classes, with a notable feature of our approach being the main class as well as the handler class.

We implemented game functions for basic functions elements of the game, such as rewards and punishments in their own dedicated classes. When the game runs, the game is initialized with the Game class. After the game class is initialized as the main method, the game class, calls the Menu class. The menu class implements two buttons, one that allows the user to start the game and another that allows the user to quit the game. Once the user has started the game, the handler class delivers updates to the Game class on user input and where animations should be rendered. The handler controls user input and graphics updates from the relevant functions such as KeyInput.

Adjustments and Modifications:

We intended to implement a Board class that would project all game objects and user input on a map in our initial design. Instead, we imported the "Java.awt.image.BufferedImage + Java.awt.image.BufferedStratgey" libraries; then, we created a Buffer Image Loader Class that handles and manipulates the image data. This class loads the png and outputs as the background of the game's map.

Also, in our initial design, a Reward class with two subclasses was supposed to be implemented. We realized that there is no need to create separate classes for specific rewards since it would deem the code to be repetitive. So instead, we decided that one class for Rewards was sufficient and that the reward object imports images from our sprite sheet class so that we had a variety of rewards.

Most of the implementation of the Escape the Bank game has been complemented. However, due to conflicting schedules, features such as player collision, scorebox, health box still need to be implemented. We plan to fully implement these features and their use cases before phase 3 is due.

Management process and division of roles:

We tried to distribute our tasks very evenly. We all had our midterms and other commitments for most of the time this project was out, so we sparsely met. Although whenever we did talk, we came out with very fruitful results. Every Tuesday and Thursday, we would meet online or in-person to collaborate; we got most of the work done in the final week before the phase was due. Other times we were off researching how to implement the game's functionality.

Collectively, we decided to implement the Scrum Process Flow to help manage the division of roles and responsibilities. Jad and Marco formed a team and were responsible for the back-end portion of the project, while Joshua and John were responsible for the front end. Our scrum meeting would last 10-20 minutes every Tuesday and Thursday to discuss what obstacles we have come across and the upcoming goals. Every week, we met to assess what features on the sprint backlog need to be implemented; so far, implementing the Scrum model has been successful. Because most team members were busy with midterms and assignments, Marco decided to implement the majority of the initial code initially, followed suit by the rest of the team with appropriate inputs on the project.

Libraries:

Library Name	Purpose of Implementation
`java.awt.Canvas`	Rectangular area where the user can draw or trap input from the user
`java.awt.Color`	Creates colour by using the given RGB(Red, Green, Blue) values
`java.awt.Graphics`	Allows a program to draw onto components that can be on devices or screens
`java.awt.Graphics2D`	Provides control over geometry, coordinate transformations, colour management, and text layout of graphical objects
`java.awt.image.BufferStrategy`	Allowed us to organize complex memory on a particular Canvas or Window
`java.awt.image.BufferedImage`	To handle and manipulate the image data

java.awt.event.KeyAdapter	Adapter class for receiving keyboard events
java.awt.event.KeyEvent	Indicates a keystroke event generated when a key is pressed, released, or typed.
java.awt.Rectangle	Specifies an area in a coordinate space that is enclosed by the Rectangle (x,y) in the coordinate space, its width, and its height.
java.io.IOException	To throw a failure in Input & Output operations
javax.imageio.ImageIO	An interface to be implemented by objects that can determine the settings of an object, either by putting up a GUI to obtain values from a user or by other means
java.util.LinkedList	Linked List data structure with methods similar to ArrayList. This means that you can add items, change items, remove items and clear the list in the same way.
javax.swing.JFrame	Works like the main window where components like labels, buttons, text fields are added to create a GUI.
java.awt.Dimension	Many functions of Java AWT return dimension objects. It will create a new Object with height and width set to zero.

Challenges and Measures taken to enhance Code Quality:

We created a Handler Class that runs through a loop to update all game objects to avoid duplicate inputs, which through rigorous testing, we saw that a need for a handler class was necessary. The Handler is initialized in the Game class and calls the tick and render method once to process all game objects. Moreover, we implemented the handler class to process all game objects add and remove functions to avoid latency. We created an enumeration save for distinguishing which game object we were processing. The Handler is passed to the KeyInput and player class; then, we loop through and distinguish if the game object is the player. Since the player is the only object that reacts to key inputs, we assign the up, down, left, and correct variables in the Handler class instead of the GameObject class to avoid other game objects having unused variables.

The game map with pixel portable network graphics(PNG) had to be designed using photo editing web applications from scratch. Then we implemented a BufferedImageLoader Class to load the image and call it in the Game class. We

matched the corresponding colour on the image and created relative game objects with the Handler class.

In the Game class, the render method will render everything in our game, and the tick method will update everything. We make these two separate because the tick method gets updated 60 times a second while the render method gets updated a couple of thousand times a second.

A Sprite sheet was created for the game images to have all images in a single portable network graphics(PNG) file, which avoids processing multiple image files at a time.

The biggest challenge we faced was implementing the moving enemy class during this phase. Since we do not have relative knowledge to process the enemy moving towards the player, after some long hours and extensive research, we successfully implemented the enemies to chase the player.