CMPT 276 – Group 2

Phase 2 - report

Sahba Hajihoseini, Dylan Kurath, David Ligocki, John Mitton

Overall approach to implement the game:

First, we implemented a plan based on our UML and each of us took responsibility for one part of the design and wrote the stub functions. After checking the stub functions, we started coding the classes and functions. Furthermore, all members made their own branches and we used them to keep our work organized. The process was simple: keep pulling from the origin master, commit and push in your branch and after approval of all members, merge your branch to the master.Moreover, for better management in coding process, first we tried implementing our entities:

- entity.java
- movingEnemy.java
- punishment.java
- regularReward.java
- userCharacter.java
- barrier.java
- bonusReward.java

After that we implemented the managers:

- entityManager.java
- stateManager.java

while we were working on entities and managers, we started implementing these classes:

- App.java
- Maps.java
- Renderer.java
- userInput.java

During this phase we kept updating and checking all classes to make sure they are suitable for the game.

Adjustments and modifications to the initial design

- Renaming game.java to App.java
- We added threading to create a tik timer to indicate when each entity moves and updates
- Added render to app.java
- Barrier was not a class in our UML design in phase 1, although we forgot about this, and added it in phase 2.
- Moving SetScore() and GetScore() from userCharacter.java to abstract class entity.java
- Adding CanMove(), Move(), SubtractScore(), Collision(), and render to entity.java
- Adding update(), render() in App.java
- Adding render() to map.java
- Adding pause(), unpause(), render() and, removeFromList() to entity manager and removing get and set remaining rewards
- Adding KeyReleased() and KeyPressed() to update the Boolean for each key in UserInput.java
- Adding CharacterCollision() and removing Spawn() from MovingEnemy.java

Management process of this phase and the division of roles and responsibilities

Most of our meetings were in person after the class hours however, we managed to have a few online meetings as well. We communicated through our Discord server and we used this server to plan our meetings and answer questions. In this phase, we put our focus on implementing the classes based on our UML and making some changes if necessary.

- David: Moving enemy, findhortestpath for movingEnemy, sprite design
- Sahba: Entity, Entity manager, Writing the report
- Dylan: Graphics, rendering, and spritesheets
- John: Assigned Roles, Stubbed all classes and methods except for graphics related files, Edited all files (excluding graphics related files) primarily App.java and methods shared by the entity class and update() in all classes, User Input, Managing game logic and flow, Bonus reward.

External libraries and imported classes

In this phase we used AWT (Abstract window toolkit) as our GUI for the following reasons:

`java.awt.*` (App.java)	For Graphics class
`java.awt.Canvas` (Render.java)	To make an area of the screen onto which the application can draw or to trap input events from the user
`java.awt.Dimension` (Render.java)	To encapsulates the width and height of the component
`java.awt.Graphics` (Render.java)	To abstract base class for all graphics contexts
`java.awt.Graphics2D` (Render.java)	Extending the Graphics class, coordinate transformations
`java.awt.image.BufferStrategy` (Render.java)	Organizing complex memory
`java.awt.image.BufferedImage` (Render.java)	Used to handle images
`java.awt.event.KeyAdapter` (UserInput.java)	Adaptor class to receive keyboard events from user
`java.awt.event.KeyEvent` (UserInput.java)	Receive keyboard events from user

And:

`javax.swing.JFrame` (Render.java)	Support for Swing component architecture

We also used the IO package:

`java.io.File` (Render.java)	Executing and reading files.
`java.io.IOException` (Render.java)	Used to interpret failed input or output

And we used ImageIO for:

`javax.imageio.ImageIO` (Render.java)	Reading and executing images from resources

Measures took to enhance the quality of the code

- We made more methods to inherit from entity.java.
- Analyzing codes in each branch before merging them to avoid making errors and conflicts.
- We also paid attention to high cohesion but low coupling function design.

Biggest challenges

- Time estimations and management: Our biggest challenge in this phase was time management. Coding this project took more time than what we estimated, so we had to work harder on the last days
- Installing and executing the code with Maven: At first we could not run our codes with Maven and it took some time to figure it out.
- Dealing with merge conflicts in the command prompt: We had difficulty in pulling from the origin master after merging our own branch because of merging conflict, but fortunately, we managed to solve it.