Report

We have partitioned the game into four states including menu, game, winning and losing states for encapsulating the behaviour of each particular state of the game.
Each state has its behaviour and functionalities. Thus, they need to be handled separately by using the state design pattern. As a result, it can help us to have a better idea of the requirements needed in each state and make the transitions among the states much smoother and easier.

Implementation of the game description:

The whole flow of the game runs based on the game loop that is responsible for controlling the flow and speed at which the game operates. Inside the game loop, it invokes the render and tick method sixty times per second. The render method is responsible for drawing all entities in the game into the screen such as Main characters, enemies, rewards, etc. The tick method is responsible for updating the game based on the processed inputs, which update the game states continuously. Each entity in the game has shared attributes such as position, width, and height that help to place and set the size of each entity on the screen. Moreover, we have implemented the collision detection between each entity in the game. After detecting the collision, it executes a certain task accordingly. Certainly, collision detection plays an important part in the game since it makes it playable and interesting.

Modification to the initial design of the project

Here are some of the modifications we made to the initial design of the project:

We modified the user interface package by using the state design pattern. To do so, we have added more states such as winning, losing, and game states since we realized that it is much easier to handle the behaviour of each state and transition between them. The game needs transit between each state class smoothly based on different scenarios. Therefore, the state pattern has helped us to solve the mentioned requirements by handling them much easier. Moreover, we have implemented an abstract class called state, which is responsible for encapsulating the its behaviour and giving us the ability to place the common functionalities of each one.

Added a new package called loading resources, which is contained with two classes named image manager and image loader. The image loader is responsible for loading an image in the resources folder while the image manager with the use of the loader class loads all the images needed to be used in the game. Having these two classes helps us to run the game more efficiently by easily loading all the images in one place.

Added a new package called Tile, which contains three classes named tile, wall, and dirt tile. This package contains the whole data of the map, for instance, each tile has its unique ID, width,

and height. Having this package with the mentioned classes is necessary to define each tile based on their ID. With the help of each tile's ID, we have made our desired map by creating the map.txt file, which contains the data of our map. Implementing the map based on the file is more dynamic because you can only change the IDs in the resources folder if you want to change the map. Therefore, it provides us with lots of flexibility with producing different types of maps.

A package called collision manager has been added to the structure of the project. Having this package, which holds the collision class, helps us with managing the collision between the entities that exist in our game.

Division of the role and responsibilities

Based on the packages that we came up with during phase one, we decided to divide the responsibilities. We divided the packages based on each person's abilities and comfort with the task. Therefore, by communicating with each other we planned out the steps we need to take in order to finish phase 2. During this phase, we had scheduled meetings three days a week for discussing the progress of the project and planning out the future steps to complete the project. Throughout our meetings, we helped each other if we had difficulties dealing with issues or bugs for being able to cope with problems along the way. We used social media platforms for discussion through text and being in touch with each other.

Libraries used for implementing the game

For implementing this game, we used java. awt library for painting and drawing graphics into the screen. Here are the classes we used from this library:

Canvas class:

It gives us an area for drawing into the screen. We made the screen class subclass of the Canvas for using its useful functionalities in order to draw and pain images into the screen.

Graphics:

Allows drawing images and graphical context onto the screen as we have used them mostly for rendering images.

BufferStrategy:

It gives us a mechanism for rendering images and showing them on the screen. The main purpose of using BufferStrategy is for users not to see any artifacts or flickering on the screen.

BufferedImage:

It is a subclass of the image class and is used for handling and controlling image data.

KeyListiner interface:

It is an interface used for receiving keyboard events from the user. We have used the KeyInput class for processing and validating the key inputs passed by the user for the character movement.

We also have used java. swing library for creating the window of our application. Below are the classes, which used from this library:

JFrame:

Used for creating the graphical user interface by providing a window where text fields, buttons, and labels can be added to it.

Timer:
Used for scheduling the tasks to be executed in the program based on the specific amount of time.

ArrayList:
Used for storing entities of the game such as rewards and punishments, which makes it easy to add and manipulate data.

List:
Provides the ability to add, remove, and update the game entities existing in the list.

How did we enhance the quality of the code?

We tried our best to make our code readable and understandable by using suitable naming conventions.

Using descriptive comments wherever was necessary for explaining the code in order to be clear and understandable. Having good comments can make code maintenance much easier, therefore, it is important to have them throughout the program.

For the main character and the game map, we used a singleton design pattern since both entities need to be instantiated only once. Both objects can be accessed globally since other objects in the program are dependent on them. Therefore, the singleton pattern gives us the flexibility to easily access map and main character objects.

The state pattern is used for each state of the game such as winning, losing, game, and menu states. Certainly, each state has its own set of unique behaviours and functionalities. Therefore, partitioning each behaviour with the help of state patterns improves the quality of the code by having more understandable classes that have their behaviour and avoiding the unnecessary long if/switch statements.

We implement the map data to be read from a txt file, which makes the design of the map more dynamic since you only need to change the IDs in order to have a different map. Therefore, reading the map data from a file helps in having a more flexible map, which can be easily changed.

Having appropriate JavaDocs, which can save a lot of time by making the code more maintainable and understandable that can help to improve the quality of the code and have an adequate coding style.

Biggest challenges faced during this phase

All the team members have never had any experience in developing games before. Therefore, any task done for the first time would have a set of challenges and difficulties. For instance, learning about the necessary libraries used for creating graphics can be challenging if you have never experienced working with them before. Moreover, detecting collisions between the entities of the game was also difficult and time-consuming. Certainly, collision detection plays a big role in improving the user experience playing the game. Therefore, it was important to make it as efficient and as real as possible for making the game enjoyable. However, never having experience in creating any software or game application before can make it time-consuming and challenging process to come up with effective logic for implementing the collision detection between the entities. With that said, it was an enjoyable experience despite the challenges we learned how to design graphics and create our first game application. At a same time, we learned many useful and interesting GUI libraries while understanding the process of developing a game, which can be a very useful experience for our future projects and creating other types of software applications.