CMPT 276: Introduction to Software Engineering

REPORT PHASE 2 - GROUP 09

Minh Phat Tran	301297286
Chun Hei Yiu	301329448
Steven Xia	301444281
Miles Burkholder	301405683

I. Overall Approach

In this phase, our group designed and developed a game called Easter Bunny Hunt by following Agile methodology in planning and managing this game project. This game is created in the Java language. This project is divided into sub-tasks assigned to each member of the group. Firstly, each member set up the Java JDK as well as the compatible IDE to write and compile Java code. Git is also used to manage and update new features for our group project.

The idea for this game is to have one main character - an Easter Bunny, which is controlled by the player through the keyboard to make them move up, down, left or right. Movable enemies are free-roaming the maze, which threatens the main character since they can kill them in an instant. To win the game the main character must collect all the eggs to open and go through the exit portal. Moreover, there are some bonus eggs that reward the main character with more points and power up to increase their odds of winning. We also implemented some punishment traps that are created to increase the level of challenge. If the main character moves to a cell that contains a punishment, they will be penalized by deducting their score. The player loses if the punishment causes the overall score to become negative. The map of this game is designed by the 2D array of integers (each number represents a different class in the map), which is read from a text file and later translated to a 2D object array that will be constantly updated during the game. Winning screen, losing screen, pausing screen are shown when the player wins, loses or pauses the game respectively. The interfaces include the main character which is an easter bunny, enemies, trees, eggs, traps and an exit portal. Adobe Illustrator was used by our graphic designer to create all the images for the pixel characters and environment. The animated gifs are specially designed for the main character, enemies, traps and port.

The Agile methodology Scrum is applied to process and manage the project. Scrum is a framework used in software development for designing, developing and delivering products in the most efficient way. Each member in the group is assigned work and assists other members in the group if they have any problems. After that, the weekly meeting is scheduled to discuss which feature in the game needs to be added or ignored. As a result of this, our game is developed and improved quickly after every week. Meanwhile, all members are always testing the project to see whether it works properly as planned before, such as the movement of the main character or enemies as well as the time counter. All features of the game are always updated via GitLab that helps each member easily observe and manage their work.

At this time when the deadline of phase 2 is nearly coming, our group project has satisfied all requirements, though there are some features we need to improve such as the complexity of the map or adding more traps and bonus eggs to the map. Overall, our game project is playable and can be updated and improved according to the clients’ requirements.

II. Adjustments and Modifications

In our original UML Diagram, we were going to use Abstract factory as our creation pattern, but after testing the implementation, we decided it is not suitable for this game. While Abstract Factory provides a great interface for creating families of related or dependent objects, it was simply not utilized enough to be effective.

We also changed the implementation of the game map, instead of directly writing a map inside the code, we made a function call readMap(), which reads from a text file from resources and produces a 2D int Array call levelData. levelData is a 12x24 2D array that represents the initial state of the map, it is an int array where every number in the array represents a class. levelData will then be translated to a 2D Object array called screenData, the function is called when the player initializes the game. By avoiding hard coding the map in the code, it is much more flexible to change and design the map, the users can even design the map themself, they don't need to modify the source code, just the map.txt in the resources directory. We designed multiple maps that can be randomly chosen by the game, so the map of the game may not be the same every time you play.

Another design change we make is removing the throw trap function of the Hunter class. We initially wanted to give Hunter a special function where he will occasionally throw a trap in an empty space to increase the difficulty of the game. However, due to time constraints, we did not make it in time. Moreover, throwing random traps in the map can greatly impair the fairness of the game, as the trap can be put in a dead-end narrow passage and effectively enclose the player with 4 walls and unable to move out without touching the trap. While we understand this problem can be solved by designing a more advanced enemy AI, we simply have class and fiction with higher priority to implement. As a result, we later removed the hunter trow trap function.

The biggest modification to the UML diagram is the modification of the checkCollision function and the addition of the KeyAdaptor. Initially, checkCollision was implemented inside the movable character class and to be called every tick to check. However, we decided to move it to the maze class where instead of the 2D object array being passed through the function, the character itself passed through the function and be compared with the 2D Environment object array, which is more efficient. As mentioned above, we implemented an inner class within the maze class called KeyListener, which is essential for the game to take the keyboard input from the user.

Overall, we fail to deliver some function that was promised in phase one due to time constraints, but we also improve the flexibility and performance of the game by experimenting with new ideas.

III. Roles and responsibilities

Minh Phat Tran
1. Assisted to create UML Diagram 2. Implemented punishment classes for the game, scores for the main character 3. Designed random maps for the game 4. Optimized Java coding style
Chun Hei Yiu
1. Designed all game images (Characters, backgrounds) using Adobe Illustrator 2. Implemented the logic flow of the game 3. Assisted to implement the GUI and Punishment Classes 4. Wrote UML Diagram 5. Assisted to implement maven build automation
Steven Xia
1. Designed player/enemy/environment collision logic 2. Designed different enemy movement algorithms for each enemy type 3. Many small polishes (eg. game restart logic, using character animation images) 4. Implemented stack-based input handler algorithm for smooth player controls.
Miles Burkholder
1. Implemented game timers. 2. Designed/implemented bonus eggs (Freeze, Score, Speed) and their generation. 3. Implemented map initialization from txt files. 4. Assisted to implement maven build automation.

Management Process:

We had our initial meeting in October, we discussed the basic logic flow and requirements of the game, we also designed a simple maze and only 4 walls to serve as a base code. We then divided our responsibility and created our own separate branches in GitHub for different classes(Hero, Trap, Reward, etc), and later merged with the base code after peer review and discussion. We utilized Discord and Zoom to conduct multiple online meetings to discuss the complications we faced and provide advice and solutions to each other. We had occasional in-person meetings, since member availability differs from week to week, some in-person meetings are just 2-3 people to discuss topics relevant to their responsibility.

Overall, members are given distinct roles and branches in the GitLab to implement different parts of the game and are free to implement in their own way as long as the behaviour of their Classes matches with the requirement and are reviewed by a teammate upon requesting a merge. Most meetings are online and message-based, and in-person meetings are occasional and mostly for important discussion and implementation.

IV. External Libraries

The library used to handle the GUI of our game was Java's Swing library (java.swing). We chose to use the Swing library because it is platform-independent, well documented, and some members had previous experience using the library. From the library, JFrame is used to open a new window frame for the game, and the entire game is shown on a JPannel (a GUI component that can be resized and have other GUI components appended to it). We also used a Swing Timer (which triggers an event at regular specified intervals) to represent each "tick" of our game. Finally, at each "tick" we also used the Swing repaint method where we could specify what should be shown on our games panel/screen, as well as the exact locations of the object images.

The Swing library is an extension of Java's AWT (Abstract Window Toolkit) library. Thanks to this, we also used many of the AWT libraries methods to assist with graphic components. These include but are not limited to Font, Colour (pick RGB colour), Dimension (for setting the size of components), Image (convert image file into usable GUI component), and Graphics which is used to draw and display graphics on the game panel such as images or text. Furthermore, we also used AWT to handle keyboard input from the user by utilizing functions of KeyAdapter (takes the key event and performs desired logic), KeyEvent (listens for events on keyboard), and ActionListner.

Lastly, we used java.util library for data structures such as an ArrayList and Stack which were used for various purposes such as our map of Environment objects, finding our enemies' next move, and deciding the current direction of our main character. For the purpose of reading our game's map matrix from a file, we took advantage of java.util Scanner to read from our map files, and finally the java.io library for File and file exceptions (FileNotFoundException).

V. Quality Enhancement Measure

As mentioned before, all members created their own branches in GitLab and implemented their designated classes and functionality. Upon finishing their part. Members requested a merge and a peer review of the code to ensure the quality and format of the code are standardized.

To ensure the program is maintainable and of high quality, we try our best to keep the programming to be low in coupling and high in cohesion, proper encapsulation is applied to all classes to ensure the prevention of direct accessing.

Finally, frequent communication is key to a high-quality code, although all members are in charge of different classes and functions, we encourage communication and ask for help and clarification if needed, every modification and merges are notified on our own Discord server to ensure there is no miscommunication.

VI. Challenges

One of the biggest challenges with the implementation phase of our project that we faced was getting started, we found it difficult to add components individually when there was no framework originally. Once we developed the base code for the maze and some characters that could move, it was much easier to work independently, then assemble each component for testing. Although, sometimes we will run into merge conflicts where at times a member would need to greatly change their implementation to mediate the conflict. Especially for the biggest class “Maze.java”, where one teammate implementation may have a huge impact on another teammate implementation. Proper communication is the key to solving this problem

Designing pixel charter and the background image is also a challenge, while one of our members has some graphic design experience, designing character animation from scratch is indeed a difficult challenge. While graphics may be seen as less important than other components of the code and may seem as mere presentation, graphics is what capture the player’s attention and provide important instruction to the player, it is definitely an integral part of game development. After all, no one wants to play a game where a bunch of colourless squares and dots chase each other.

Finally, it was challenging to get started using Maven and creating the proper pom.xml file, we struggled to create a proper jar file that ran our game as we did not include some information in the pom, we also run into some problems like for some IDE like IntelliJ IDEA, relative paths does not work for getting resources like game images, we struggled with this until we change the IDE working directory to the target directory.