SOFTWARE MAINTENANCE (COMP2042)

Maintaining and evolving existing software

Student name	Student ID
Loh Jin Xian	016763
Tan Kheng Hau	014719
Bryan Foong Jia Ming	017622
Sharon Pawa Denys	017033

Table of Contents

1. Introduction

In the context of Software Engineering, software maintenance is one of the many technical process that is important in order to modify and maintain an existing software. It is needed in order to ensure satisfaction of customer and user. One of the many ways to maintain a software is by implementing new or changed user requirement. In this coursework, we focus on adding new features to Graphical User Interface of Diamond Hunter. It is important as it provides better interaction between the machine and the user.

Diamond Hunter is a simple 2D maze game where the main character (user) has to collect diamonds by going through several obstacle. To further enhance the gaming experience additional feature such as a map viewer is added. Map viewer allows the user to set the location of entities (Axe and Ship) in the map.

2.0 Prototypes

2.1 Design philosophy

Diamond Hunter is a Java game that has an old-school 2D pixel art interface. The task given for this coursework is to design a graphical user interface (GUI) for the newly built map viewer for this game. Therefore, the design philosophy for the GUI has been decided to incorporate both minimalism and 16-bits pixel art style.

Firstly, the GUI is designed to be user-friendly for the user. Every buttons or sprite serve an essential purpose in the GUI, making the whole experience of navigating the GUI simple yet elegant. Other than that, the colours used in the GUI are hand-picked carefully to encourage the user to interact with our buttons. By using colour theory, buttons that have a brighter colour will communicate a sense of interactivity with the user. Finally, the GUI is also presented with an easy-to-understand message box that contains an important guide on how to navigate the GUI for the first-time-user.

To achieve the proposed design, the standard industry practice to prepare a low-fidelity, mid-fidelity and high-fidelity prototypes are made and tested. After multiple trials and errors, the best design is implemented into the game using JavaFX scene builder.

2.2 Low-fidelity

A rough sketch on a paper about the basic layout for the GUI. The following low-fidelity prototype serves as a backbone for the project.

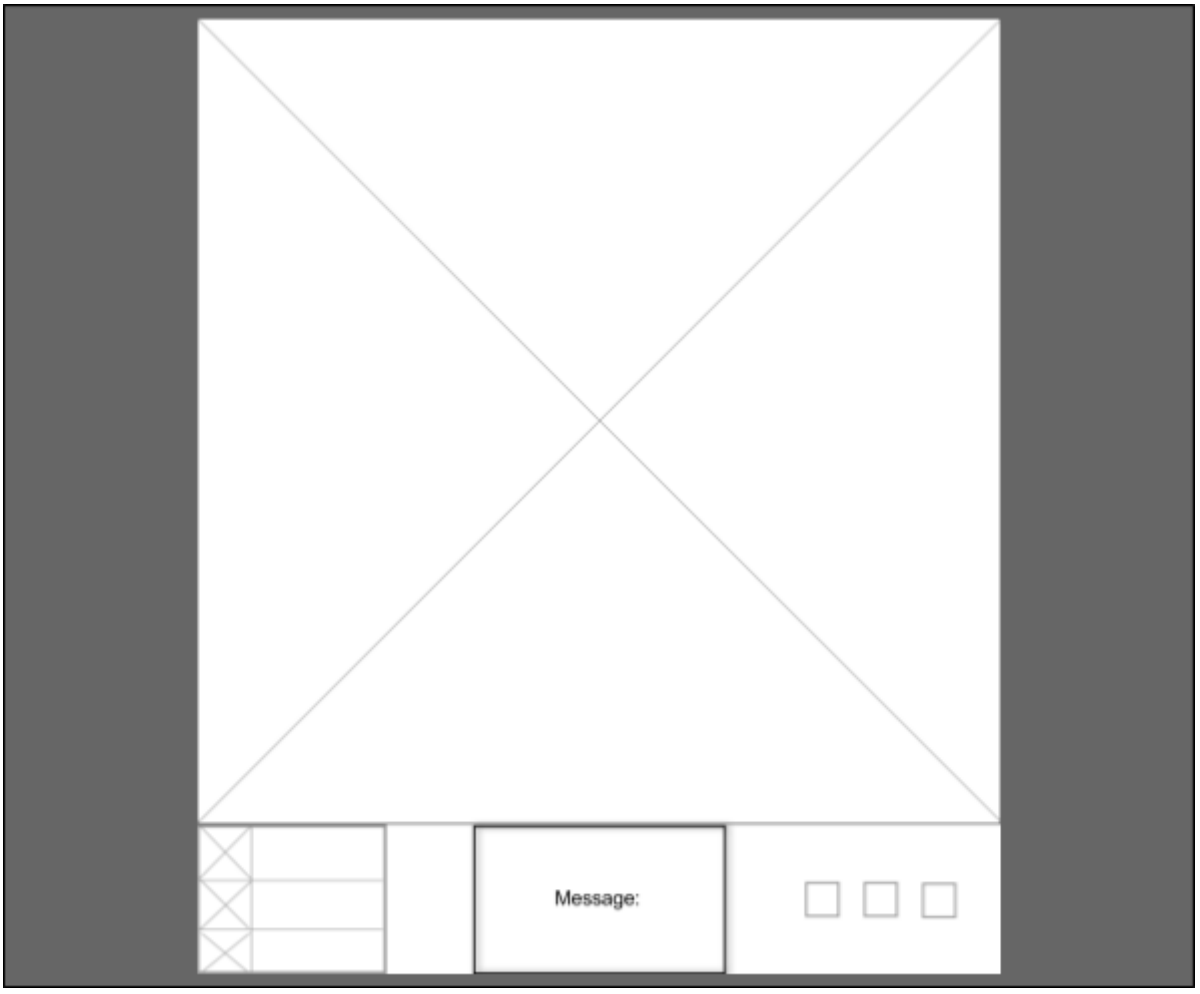

Figure 1.0: Low Fidelity Prototype

2.3 Mid-fidelity

Mid-fidelity prototype contains a little bit more details compared to low-fidelity prototype. Images, icons and texts will be placed in the layout for better visual enhancement. The following mid-fidelity prototype shows the improvement made to the GUI.

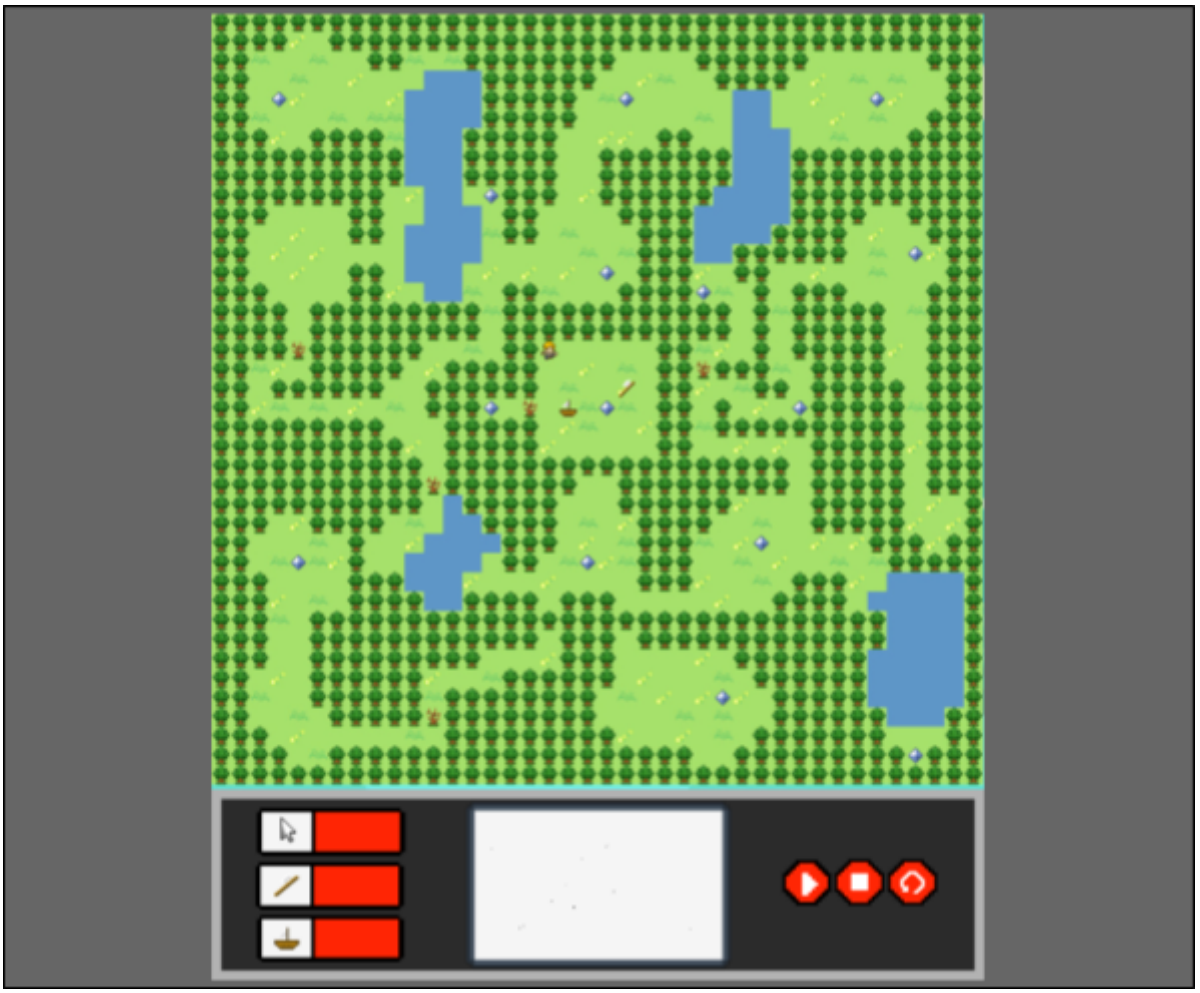

Figure 1.1: Mid fidelity Prototype

2.4 High-fidelity

High-fidelity prototype will look like a completed design of the project. Everything from UI design, images, texts etc have been added to the prototype. The following high-fidelity prototype is the targeted design for the project.

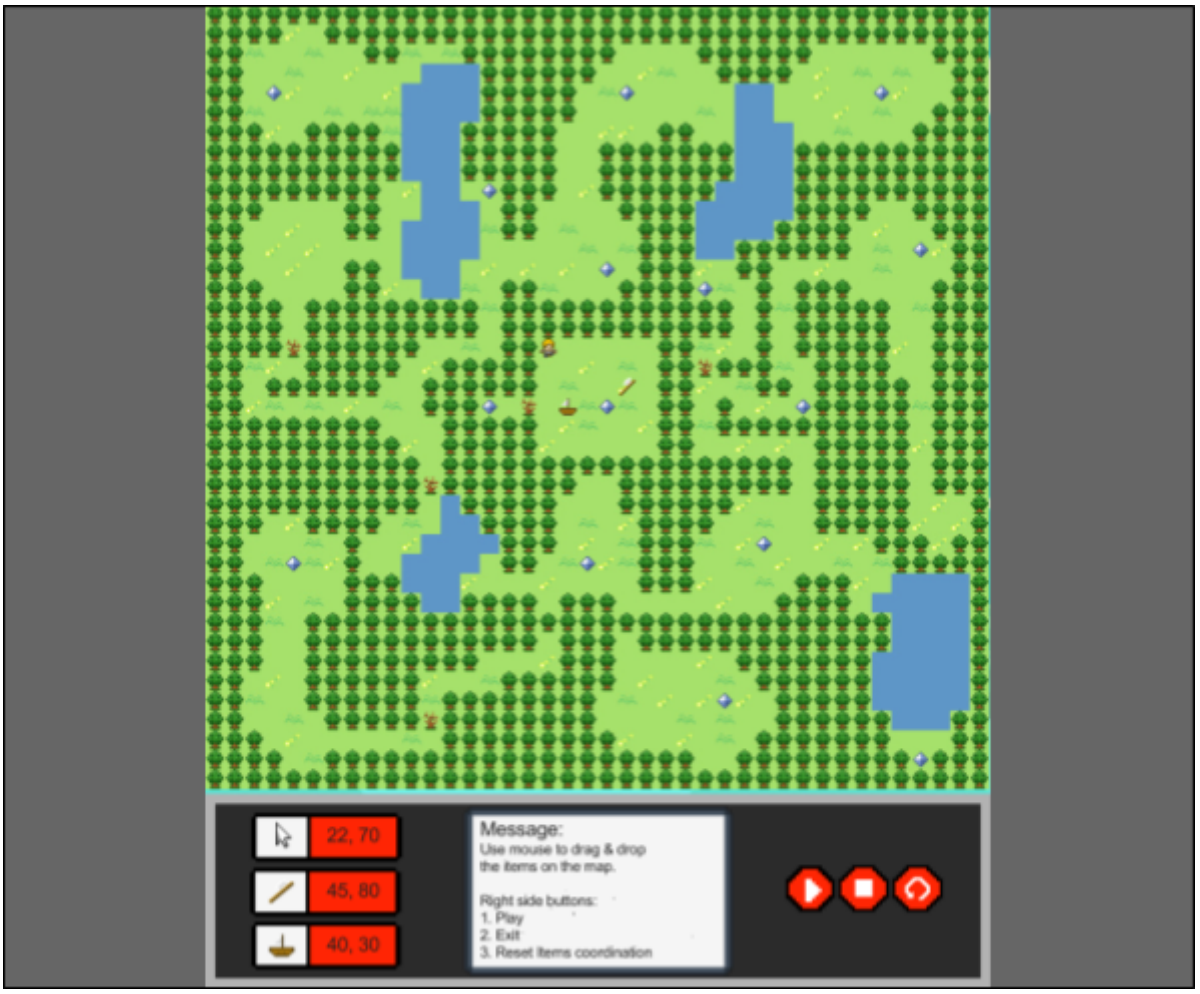

Figure 1.2 : High Fidelity Prototype

3.0 Storyboard

Initial Drafting

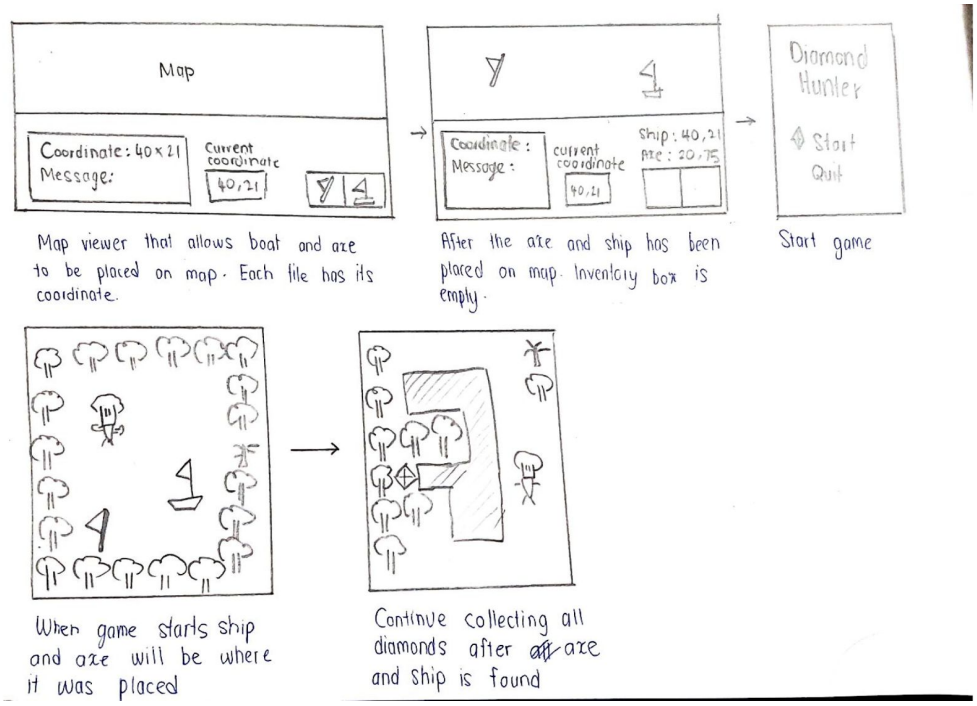

Figure 2.0 : Storyboard draft of Map Viewer

Finalised user guide

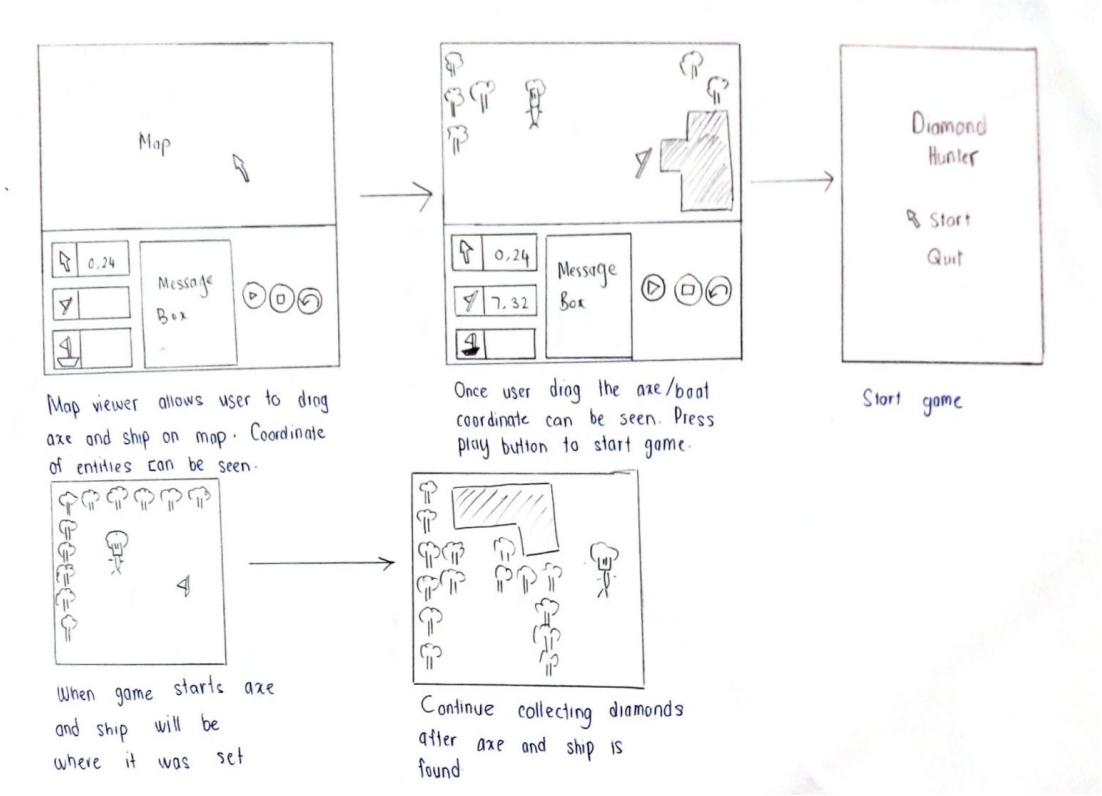

Figure 2.1: Finalised storyboard of Map Viewer

4.0 Code Analysis

Map Viewer

To create the map viewer, the first step is to create a launcher application class, *MapLauncher.java*. The application is linked to a graphics user interface (GUI), which is built by SceneBuilder. The codes for GUI are then converted to *MapViewer.fxml*. The innermost container : *anchorPane* and *mapPane*, is linked to the application controller *MapController.java*. The canvas *mapCanvas* is used for loading map.

Map is loaded during initialization of application. The class contains boolean of game launch, setting canvas size, and load map into the application. The map's functions are separated to class *MapPane.java*. The functions are similar to *TileMap.java*, which is a part of the game itself. The map's procedures are as follows:

1. Setting up tiles based on fixed tile size of 16 pixels
2. Saving individual sprite images of *testtileset.gif* into respective tile number.
3. Load the map by tiles by parsing array number in *testmap.map*.
4. Buffer the drawing canvas into graphic context.
5. Draw the rendered image into *mapCanvas* by setting up a snapshot parameter.

The challenge of this task is utilising the java library to print the map. Java implements various libraries, to convert from pixels into respective rendered images. Codes are used to print the map instead of any design, as tiling games uses few tiles repeated throughout the canvas.

Map Details

In order to have coordination on the Map , a *GridPane* is build on top of the Canvas. The *Gridpane* act as a grid to separate the whole map into boxes of rows and columns. A *TileInformation* class is created to so that each tile will have their own tile information based on their coordinate in the map. In the *MapController* class, *initTileMapping* method create an *TileInformation* object, *tileInfo* that will create a row and column constraints with size of 16 pixels in each box in the map and load each box with image based on the tiletype provided in the *map.txt* file.

Mouse Coordinate

AddTile method responsible for printing out the coordinate of the tile when mouse hover on it . When *initTileMapping* created tiles for the whole map , *addtile* method is called so that for each tile coordinate of x-axis and y-axis so that it will have their coordinate information on the tile itself, therefore when the mouse hover to any tiles, it will print the current coordinate which the mouse stopped on.

Reset Button

Creating an resetting method which place both axe and ship to the default position by hard coding the coordinate of the axe and ship respectively inside the reset button method. When the user click on the reset button , both axe and ship coordinate will be on default position.

5.0 Conclusion

To make the GUI functional, large amount of background coding is needed. Some aspects throughout the development are:

1. Understanding various java libraries, and the syntax format of its constructors.
2. Consistent variable naming and method operations, important for other developers to reuse and retain software structure.
3. Retaining the structure of the original source code, while relating methods to the new GUI.
4. Clear and important planning of using git, such as clear commit statement, pushing and pulling scheduling and file selection.

One of the challenges of creating standalone applications is maintaining the original source code of existing software. Understanding original codes, designing user interface, and linking new application to software takes precise implementation and consistency.