Principles of Software Engineering Milestone Two

Report

Name: Bo Cao

Student Number: s5068270

Major: software development

Date: 19/09/2019

Contents

[Requirement 3](#_Toc19559596)

[Product Use Cases 4](#_Toc19559597)

[Software Architecture 5](#_Toc19559598)

[Summary of Design 5](#_Toc19559599)

[Design Goals 5](#_Toc19559600)

[Design Patterns 6](#_Toc19559601)

[Artifacts of the design 6](#_Toc19559602)

[Subsystem Decomposition 6](#_Toc19559603)

[Structural Modelling 7](#_Toc19559604)

[Dynamic Modelling 9](#_Toc19559605)

[User Interface 13](#_Toc19559606)

[Test Plan 14](#_Toc19559607)

[Historical analysis 16](#_Toc19559608)

# Requirement

The following table shows the functional requirements of milestone1 and the progress of each functional requirement.

|  |  |
| --- | --- |
| 1. The game shall provide a timer that player can check the time since them start the game until the game finish. | TimeController is created and properties and methods are defined |
| 2. The game shall let players know how many bomb left shows on the top of the screen. | The Class: GameBoard has been defined and the basic design of the game page has been completed. At the top of the game board will show how many bombs remain. |
| 3. The game shall provide a button that players are able to restart the game by click a single button when they meet difficulty in the game. | Have designed, development work is ongoing |
| 4. The game shall give the result immediately after players uncover a tile whatever it is a bomb or blank. | Defines the minisquare in the game panel. Defines the method of mouse monitoring. |
| 5. If the tile clicked by the player is not a bomb then a number will shows on the clicked tile to describe how many bombs around this tile | Defined the method, currently developing ongoing |
| 6. The game shall let the game stop immediately when player touch the bomb because they are failed. | Defined the countBomb method to calculate the number of surrounding bombs, ongoing |
| 7. The game shall provide players are able to use a flag to sign on a blank tile as a bomb so that they can know that how many bombs have already found. | Defines the mouseClicked method. |
| 8. The number of total bomb left should reduce after player place a flag on the blank tile. | Will display on the top of the game's interface, in the design phase |
| 9. The game shall allow player to place a question mark on the blank tile to sign it as a undetected tile | Defines the mouseClicked method. |
| 10. The game shall provide a selection after game over whether players want to play again or exit the game. | Defines the Clicked method. |

# Product Use Cases

In this game, the role of the game is a single user, user has the following types of use case:

1. Start Game

In this use case, the user launches the program and enters the game.

2. Action on Square

“Action on Square” is the user's mouse action. The user left-clicks the mouse to perform the mine-clearing operation.

3. Mark on Square

“Mark on Square” is the user's tagging operation. The user right-clicks or double-clicks the mouse to mark the attribute of the square

4. Restart Game

When the user encounters a difficulty and cannot play the game, he can choose to restart the game.

5. Exit Game

When the user finishes finding out all the bombs, or the user clicks on a bomb, the game ends.

6. Decide play or exit

After game over, the system will ask the user whether to repeat the game or opt out.

Use case implementation progress report:

|  |  |
| --- | --- |
| Use Case | Progress |
| Start Game | Have designed classes and methods, functions development are ongoing |
| Action on Square | Have designed and developed right-click operation |
| Mark on Square | Have done right-click mark on minisquare |
| Restart Game | Have designed the interface on the GameBoard |
| Exit Game | Have designed the game process |
| Decide play or exit | Have designed the game process and method |

# Software Architecture

The programming language used in this software engineering design is Java language, and the development tool used is Intellij IDEA.

Java is an object-oriented programming language with simplicity, object-oriented, distributed, robust, security, platform independence and portability, multi-threading, dynamics, etc. At the same time, simple GUI design is also very easy to develop. The development of small games.

The software architecture used in this software project is mainly the design of MVC (Model-View-Controller) separation. MVC is a software design paradigm that organizes code in a way that separates business logic, data, and interface. In this architecture design, the entire engineering module is divided into three parts:

|  |  |
| --- | --- |
| Model | The Model module mainly defines the entity classes and interfaces needed in the project. |
| View | The View module mainly defines the interface of the game and various operations of the user under the interface. |
| Controller | The class StartGame is defined in this module, and the main function is created to start the entire game. |

# Summary of Design

## Design Goals

Design goals:

Implement the functional and non-functional requirements defined by milestone1. There are some design principles which need to satisfy:

* Greater expressive power
* Solution orientation
* Very high modularity
* Easy to change or extend
* Code readability
* Simplification through abstraction

## Design Patterns

Object-oriented design focuses on "things" in the real world, which are part of the software engineering problem. Engineers need to define these things (we will call them objects) based on attributes and behaviors. In this project, the things correspond to the users of the game. The project needs to be designed and developed according to the various operations that the user can perform in the game and the various results obtained.

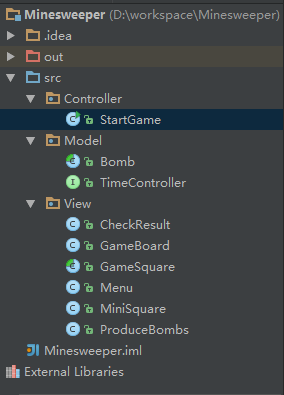
The whole design idea mainly adopts the top down design approach. The top-down design approach first identifies the main modules of the system, breaking them down into lower-level modules and iterating until the required level of detail is reached. This process is gradually improved. At the top, an abstract design is required, and then the design is refined to a more specific level in each step until the system reaches a level that no longer needs to be refined, and the design can be implemented directly.

Meanwhile, “Abstraction-Occurrence Pattern” and “The General-Hierarchy Pattern” are also used in this software engineering.

## Artifacts of the design

### Subsystem Decomposition

There are three packages in this project: Model, View and Controller.



1. Model Module

* Bomb Class: Defined the Bomb abstract class and created a way to create Bomb on the game panel.
* TimerController Interface: Calculate the amount of time the game spends running. The game panel will call this class and its methods, prompting the user for the total duration of the game after the game is over.

1. View Module

* Menu Class: As a startup menu for the game. In game extensions, display a rich game menu and define different levels of game difficulty.
* GameBoard Class: This class provides a graphical model of a board game. This class creates a rectangular panel with clickable squares.
* GameSquare Class: This class is abstract, in order to enhance the scalability of the code
* MiniSquare Class:
  + - * This class inherits from the GameSquare class.
      * This class implements methods in ActionListener and MouseListener to respond to different click events.
      * Each square has its own unique coordinates and attribute representation.
      * Once the user clicks on the square, this course will call the CheckResult class’s method to count the number of bombs around it.
      * This class displays pop-ups when the game fails or the user succeeds.
* CheckResult

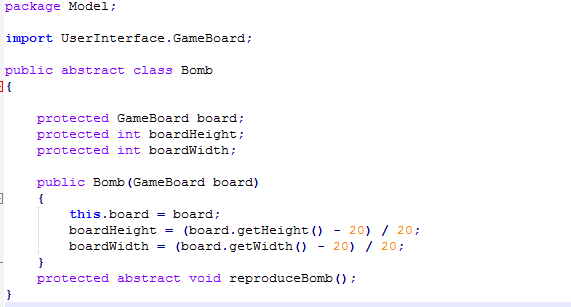
This class provides a way to calculate the total number of bombs around a given square. At the same time, the method checks if the square clicked by the user is a bomb. If it is a bomb, the bomb icon is displayed, and if it is not a bomb, the wrong icon is displayed.

### Structural Modelling

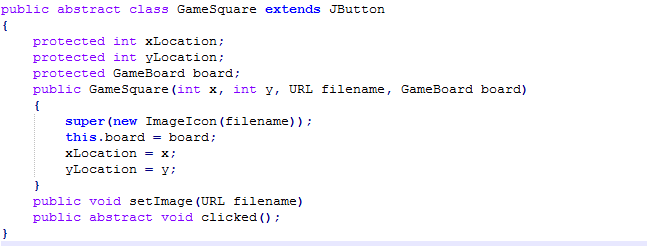
The following is a class diagram of the minesweeping game.



For example, in Model package, Bomb Class has been defined based on Class Diagram:



In the class diagram, there are extensions and implement relationships between different classes, such as MiniSquare extends the GameSquare class. GameSquare Class’s attributes and methods have been designed:



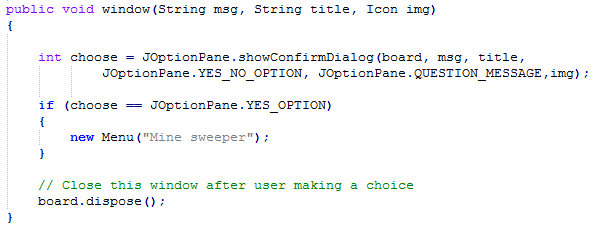
### Dynamic Modelling

1. Interaction diagrams describing the behavior between objects
2. **Sequence diagram**



The sequence diagram shows the changes in messages and states between classes in the game flow.

For example, when finishing this game, system will ask user whether again or not. Then after user chosen, the class will get result then change to menu class interface:



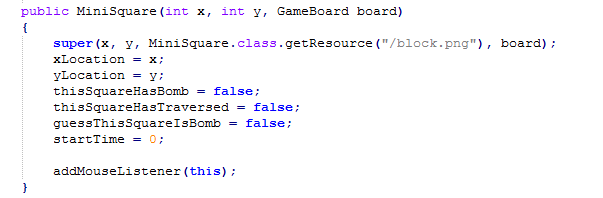
1. **Collaboration diagram:**



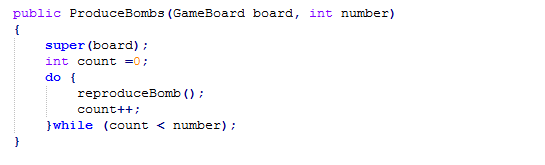
Collaboration diagrams primarily show call relationships and dependency relationships among classes.

For example as game panel, GameBoard needs to generate MiniSquare on top of it, and randomly generate a certain number of bombs, distributed in different minisquare positions.

Create MiniSquare instance, then placed on a GameBoard:



Generate bombs:

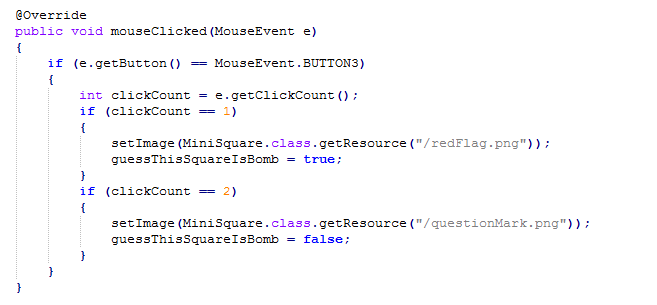


1. State-based diagrams.
2. **State Machine:**



In the state diagram, the user's actions will change the progress and results of the game. The user left click is minesweeping, right click two operations, on behalf of the user that this is a thunder, and the twin represents that the user is not sure what this is.

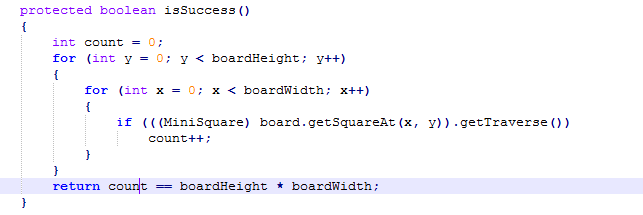
Here's how to monitor the user's right-click mouse action to change the background of the minisquare.



1. **Activity diagram:**



The activity diagram shows the activities of the game session in the form of a flowchart. For example, at the end of the game, check whether the user has dug all the mines through the CheckResult class.



# User Interface

This game is a stand-alone game that does not involve Persistent Data management and Access control and security. Therefore, this module discusses the UI design of the game.

The UI is the User Interface, which represents the user interface. The UE, User Experience, represents the user experience. These factors determine whether the usage process, habits, colors and proportions are reasonable, and whether users can be retained through a good interface and interactive experience. Therefore, UI, UE design is especially important for applications.

In this paper, good user interface and experience design are mainly reflected in the following points:

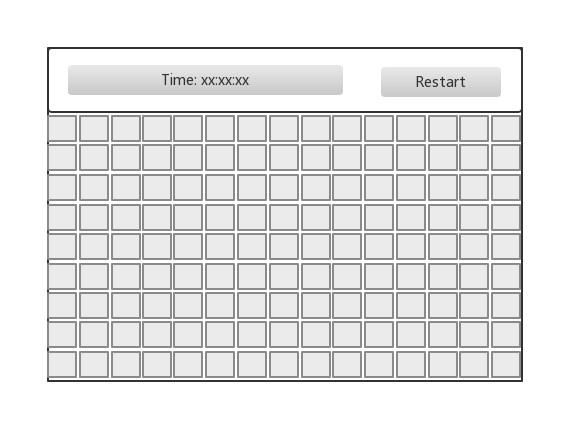
1) Clear navigation guides improve the ease of use of the system.

Design menu and top bar function to increase system usability

2) Use reasonable color and contrast.

The system mainly uses gray game panels, black bombs and numbers with different colors to distinguish the results obtained by the user operation. When the user clicks on a bomb, the bomb is red in color and is used for highlighting. Therefore, game panels and bombs can be clearly distinguished by color and contrast.

3) Maintain a single, flat layout plan.



# Test Plan

In the game system, the main function module is mainly divided into three major blocks: a game start module, a sweeping module, and a game end module. Therefore, functional test cases are written for these three modules and tested separately.

|  |  |  |
| --- | --- | --- |
| **Module** | **Test number** | **Use Case** |
| **Game Start Module** | ST-001 | Menu display test |
| ST-002 | Game start test |
| **Sweeping Module** | ST-003 | Game panel generation test |
| ST-004 | Left click test |
| ST-005 | Right click test |
| ST-006 | Count mine test |
| **Game End Module** | ST-007 | Successful mining test |
| ST-008 | Wrong Sweeping test |
| ST-009 | Again test |
| ST-010 | Time Control test |

In the test plan, by designing test cases for testing, the following are two design use cases for the design.

1. **Sweeping Module** ST-003: Game panel generation test：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Game panel generation test | | | |
| Number | ST-003 | Function Module | **Sweeping Module** | |
| Description | Test whether the game board and minisquare can be generated | | | |
| Preconditions | Enter the game through the menu | | | |
| Process | Process Description | Expected outcome | Test Results | Whether pass |
| 1 | Start menu to enter the game | Show game board and minisquare |  |  |

1. **Game End Module** ST-008: Wrong Sweeping test：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Wrong Sweeping test | | | |
| Number | ST-008 | Function Module | **Game End Module** | |
| Description | Test when the user clicks a mine, shows the red bomb sign, then game is over | | | |
| Preconditions | Enter the game through the menu | | | |
| Process | Process Description | Expected outcome | Test Results | Whether ] pass |
| 1 | Start menu to enter the game | Show game board and minisquare |  |  |
| 2 | User left click on minisquare for sweeping | The red bomb sign on this minisquare, then game is over |  |  |

In terms of performance testing, based on the non-functional requirements of milestone1 for performance testing, the main performance indicators are “Every action of this game must explicitly completely in less than 0.1 second”. Users can quickly start and close games with a good user experience in 0.1 second.

The overall operation of the game is simple and easy to understand, and it is very easy to play and easy to play. It also meets the requirements of "The game shall be easy to operate the player who is first time to play this game is able to skillful on it".

# Historical analysis