Bui Nguyen Kim Hai
QMIBHU

Programming Technology
Assignment 2: Problem 2

Metadata

This assignment was made with:

- Code: Java (NetBeans)
- Document: LaTeX (Overleaf)
- UML: Draw.io

1 Problem

Pebble is a two-player game, played on a board consists of $n \times n$ fields. Initially, n white and n black pebbles are placed on the board randomly. Each color belongs to only one player. The players take turns choosing one of its pebble, and then move it horizontally or vertically. The movement also affects the neighbouring pebbles in the direction (the pebble on the edge falls off). The objective of the game is to push out as much pebbles of the opponent from the board as we can, within a given number of turns ($5n$). A player wins, if he has more pebbles on the board at the end than his opponent. The game is draw, if they have the same number of pebbles on the board.

Implement this game, and let the board size be selectable (3×3, 4×4, $6 \times 6 \rightarrow$ turns are 15, 20, 30). The game should recognize if it is ended, and it has to show the name of the winner in a message box (if the game is not ended with draw), and automatically begin a new game.

2 TL;DR

- We have to create a 2 player game named Pebble.
- The game has 3 option: 3×3, 4×4, 6×6.
- The window must fit for all game options.
- The game must know it is end after 15, 20, 30 turns or when some one is out of pebble.
- The game will show a message when it is end, and then automatically create a new one.

3 Solution Plan

- **UI**:
  - There will be a base window for menu window and in-game window.
  - Players will move their pebbles through a pop-up frame (controller).
- **Logic**:
  - There will be a model to store and control the game state (table and players).
  - The model will check whether the game is finished every time a player make his move.
  - When new game is created, players will stay remains (passing to new game state).
  - General methods will be placed in the class named Utility.

4 UML Class

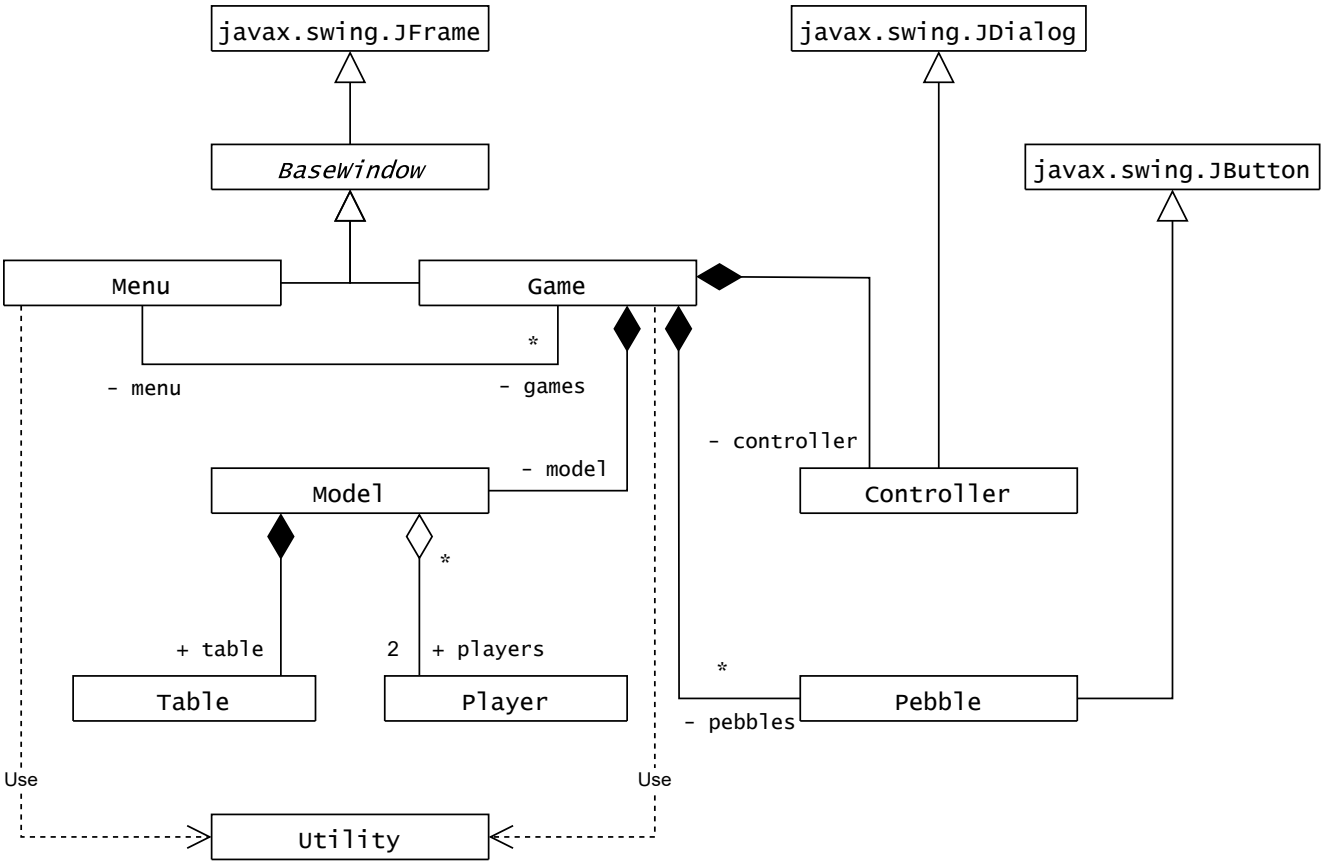

Figure 1: Designed UML Class (Omit methods, attributes).

5 Classes Details

5.1 BaseWindow, Player, Table, Pebble, and Controller

Figure 2 shows the classes: ***BaseWindow***, **Table**, **Player**, **Pebble**, **Controller**.

- ***BaseWindow***: This is an abstract class for **Game** and **Menu**.
  - Creates a centered squared window which is half size of the screen.
  - Ask for confirmation when user want to close it.
- **Table**: This is what a game is stored in the background.
  - Count how many pebbles in a given color are in the table.
  - Set/get the color at a given position.
- **Player**: This class describes what is a player in the game.
- **Pebble**: The UI of the pebble. Self-aware of its position.
- **Controller**: Allow player to move their pebble horizontally or vertically.

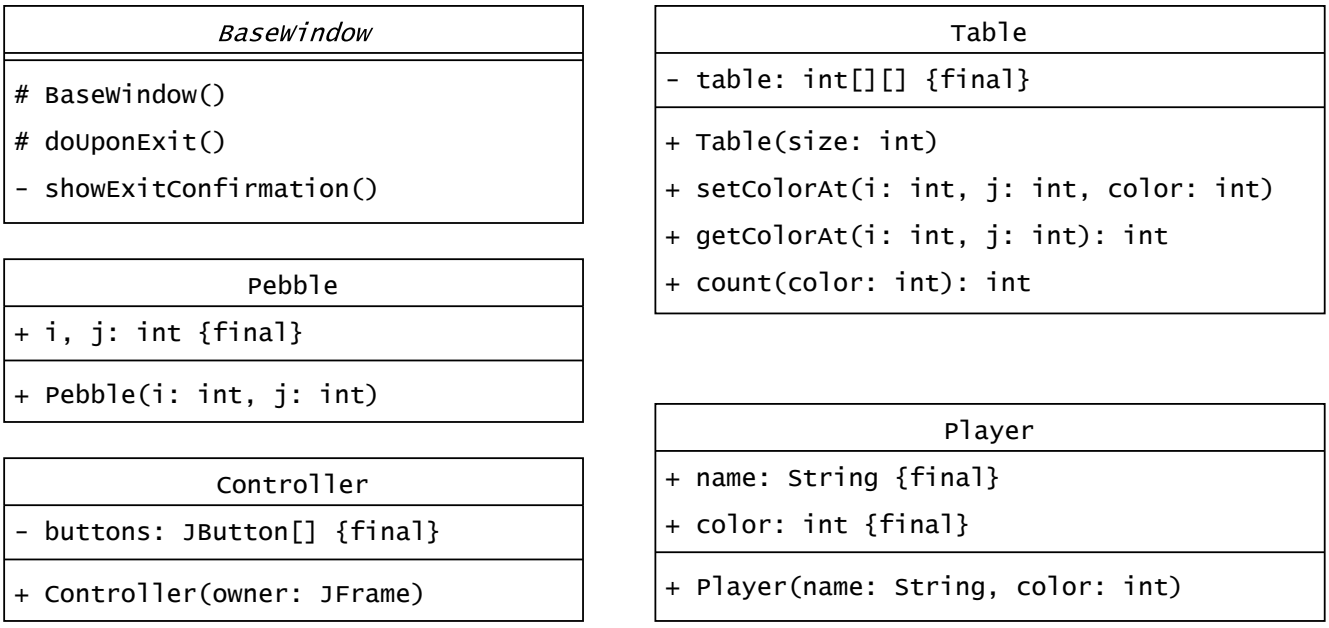

Figure 2: Designed of ***BaseWindow***, **Player**, **Table**, and **Controller**.

5.2 Utility

Figure 3 shows the **Utility** class. This is a class contains static helper methods. Utility helps me organize the code in a cleaner way, avoiding duplicated codes.

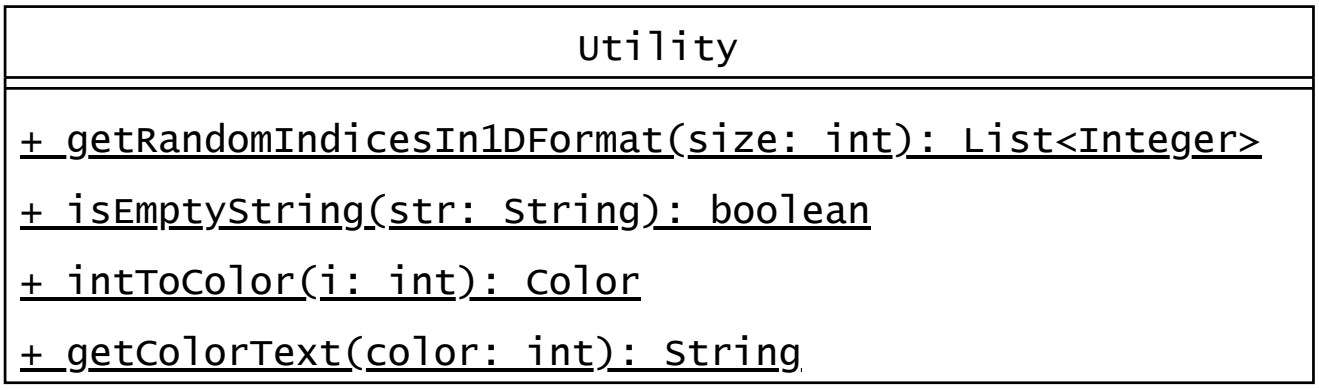

Figure 3: Designed of **Utility**.

5.3 Model

Figure 4 shows the **Model** class. Let's talk about some interesting fields and methods.

- Fields
  - **currentPlayerID**: 0 for player1, 1 for player2.
  - **size**: The size of the table (3, 4, 6).
- Methods
  - **Model**: Create a new game state with given size and players.
  - **nextTurn**: Move to next turn.
  - **getCurrentPlayer**: Get the player in current turn.
  - **getNumTurns**: Get number of turns left.
  - **getWinner**: Get the winner.
  - **isGameFinished**: Answer whether the game is finished.

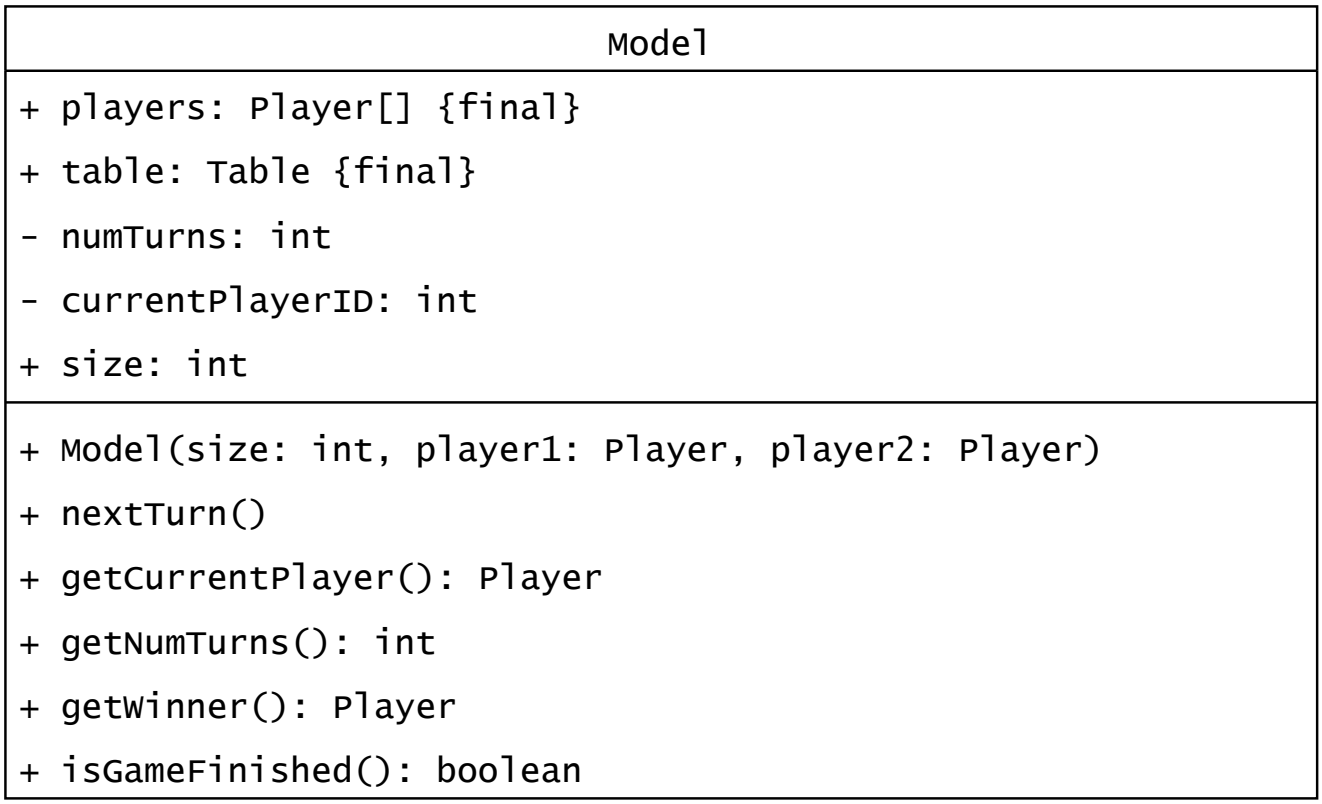

Figure 4: Designed of **Model**.

5.4 Menu

Figure 5 shows the **Menu** class. Let's walk through some interesting fields and methods.

- Fields
  - **games**: List of opened games.
- Methods
  - **Menu**: Create a menu window.
  - **isGoodInput**: Check if the username is not empty and not exceed the length of 6

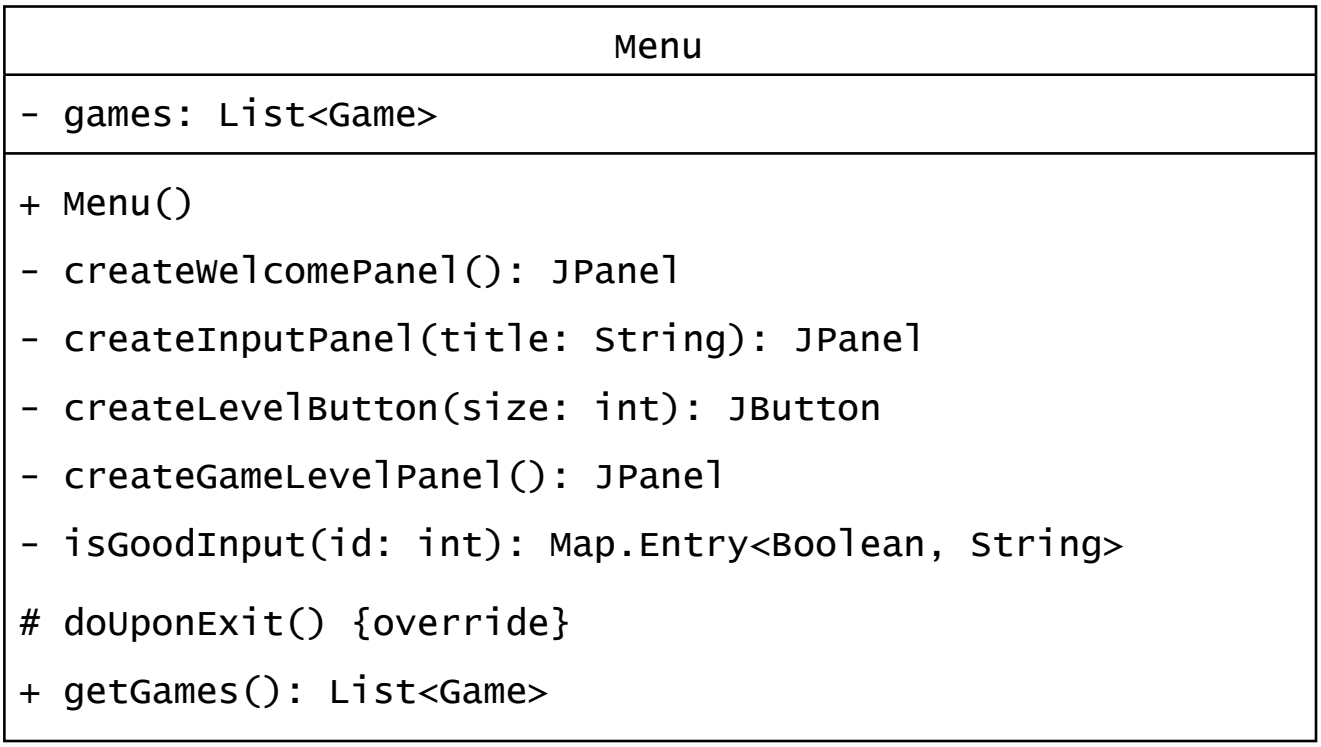

Figure 5: Designed of **Menu**.

5.5 Game

Figure 6 shows the **Game** class. Let's walk through some interesting fields and methods.

- Fields
  - **pebbles**: The UI of the Table.
- Methods
  - **Game**: Constructor, create a game window.
  - **startNewGame**: Start a new game, same level, same players.
  - **updateLabelText**:Update the game information after each turn.
  - **pushForward**: Logic to push a pebble in a given direction.

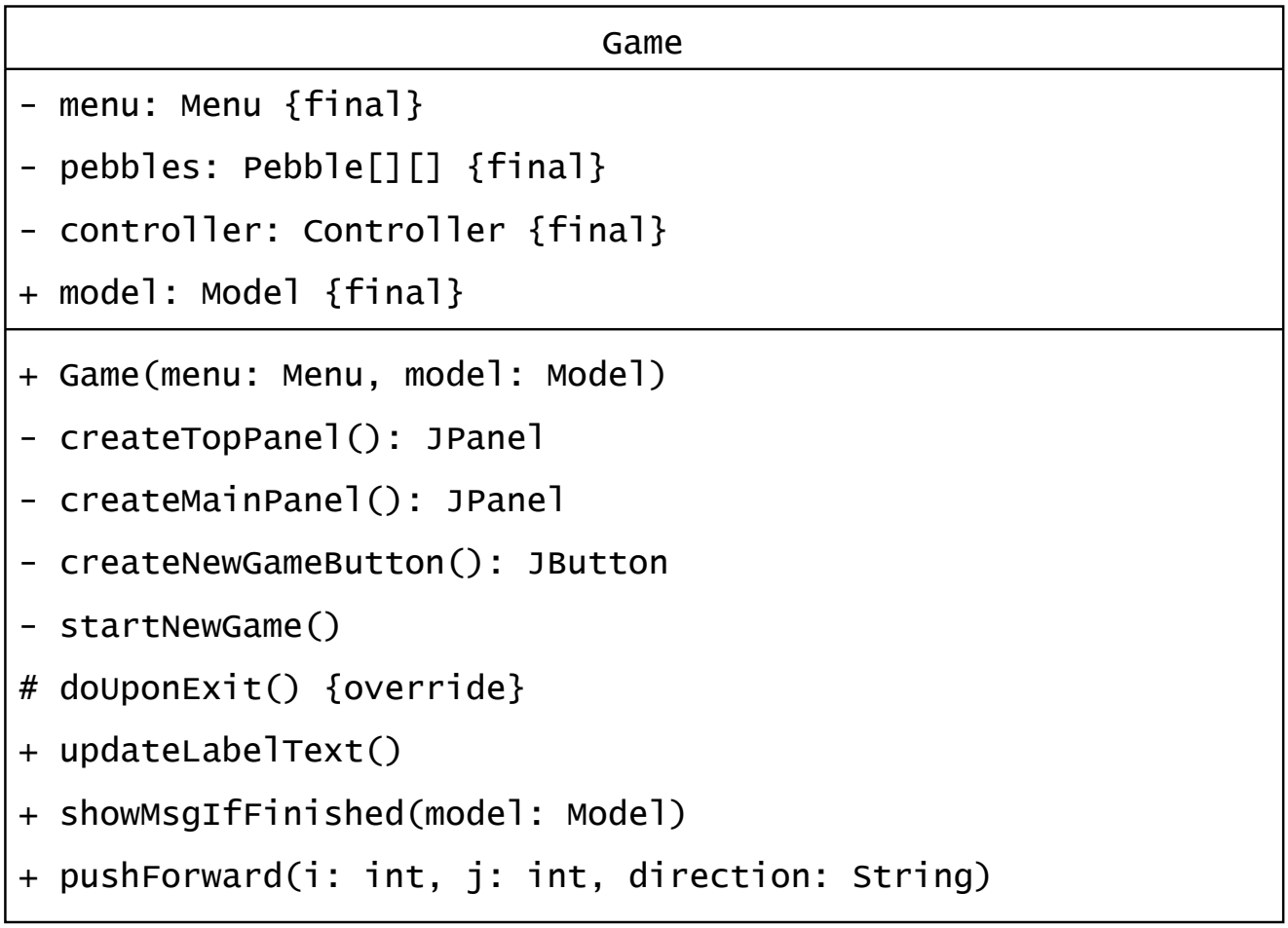

Figure 6: Designed of **Game**.

6 Connections

Each game option (**JButton**) is connected to a **ActionListener** that will create a game with selected size when the condition is met.

```
private JButton createLevelButton(int size) {
    JButton button = new JButton();
    button.setText(text:"%d x %d".formatted(args:size, args:size));
    button.setPreferredSize(new Dimension(width: 150, height: 50));

    button.addActionListener((ActionEvent e) → {
        Map.Entry<Boolean, String> result1 = isGoodInput(id: 1);
        Map.Entry<Boolean, String> result2 = isGoodInput(id: 2);

        if (!result1.getKey() || !result2.getKey()) return;

        Game window = new Game(
            menu:Menu.this,
            new Model(
                size,
                new Player(name:result1.getValue(), color: 1),
                new Player(name:result2.getValue(), color: -1)
            )
        );

        window.setVisible(b: true);
        games.add(e: window);
    });
    return button;
}
```

Figure 7: The connection between game option and its **ActionListener**.

Each **Pebble** is connected to a **ActionListener** that will trigger the **Controller** if the condition is met.

```
// Event listener
pebbles[i][j].addActionListener(e → {
    Pebble pebble = (Pebble) e.getSource();
    if (model.table.getColorAt(i: pebble.i, j: pebble.j) == model.getCurrentPlayer().color) {
        controller.setIJ(i: pebble.i, j: pebble.j);
        controller.setVisible(b: true);
    }
});
```

Figure 8: The connection between **Pebble** and its **ActionListener**.

The **Controller** has 5 options (**JButton**). Each option is connected to a **ActionListener** to allow **Player** to move.

```
for (JButton button : buttons) {
    button.addActionListener(e → {
        JButton btn = (JButton) e.getSource();

        if (!btn.getText().equals(anObject: "X")) {
            game.pushForward(i, j, direction: btn.getText());

            // Move to next player turn
            game.model.nextTurn();
            game.updateLabelText();

            // Check if game is Finished
            game.showMsgIfFinished(model: game.model);
        }

        // Close the controller window
        dispose();
    });
}
```

Figure 9: The connection between controller option and its **ActionListener**.

7 White Box Testing

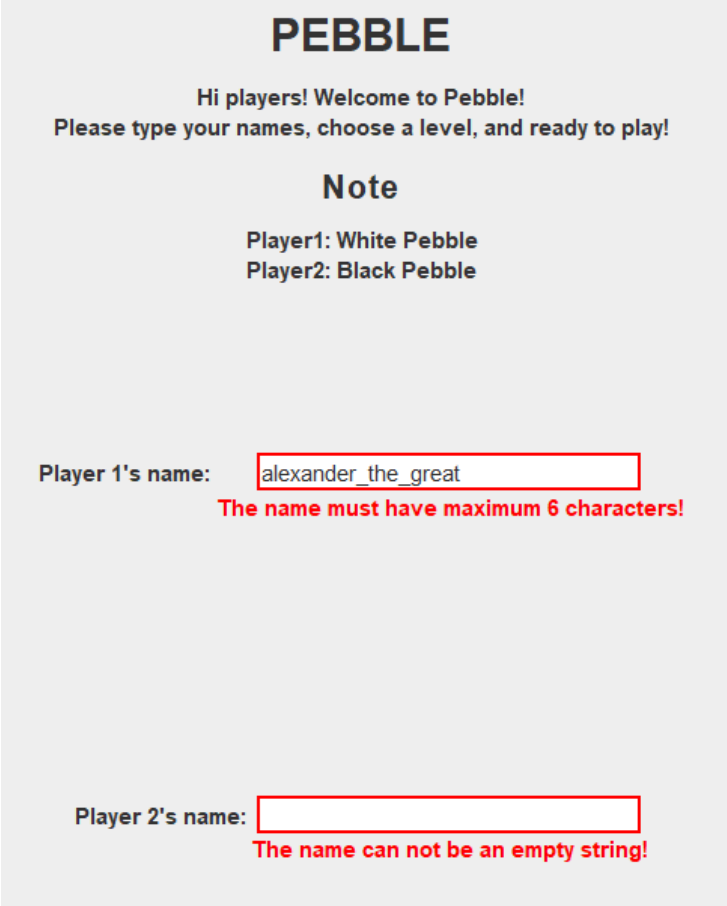

Figure 10: Player(s) type wrong name format!

Figure 10 user stories:

AS A user
I WANT TO run the program
GIVEN the menu is visible and name is longer than 6 characters
WHEN the button level is clicked
THEN the program display **The name must have maximum 6 characters!**.

AS A user
I WANT TO run the program
GIVEN the menu is visible and name is empty
WHEN the button level is clicked
THEN the program display **The name can not be an empty string!**.

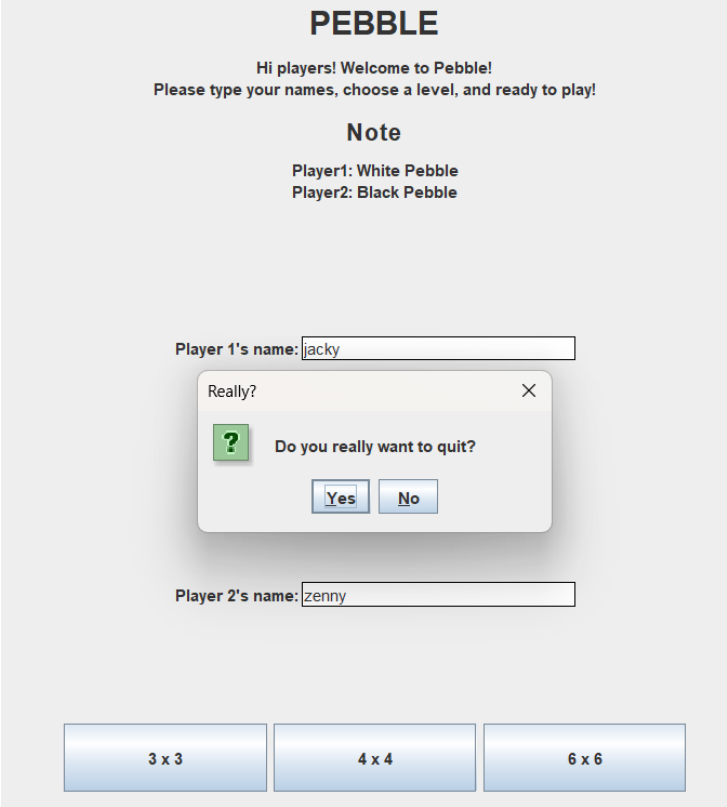

Figure 11: Player(s) click "X" icon.

Figure 11 user story:

AS A user
I WANT TO close the program
GIVEN the menu is visible
WHEN the "X" icon is clicked
THEN the program ask for a confirmation.

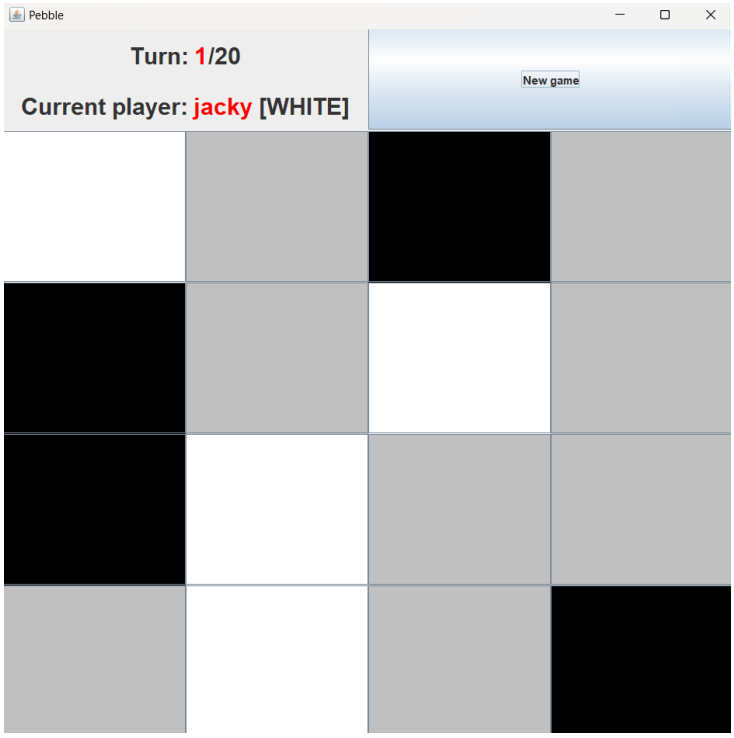

Figure 12: Player(s) click level button.

Figure 12 user story:

AS A user
I WANT TO start a game
GIVEN the menu is visible and the names are valid
WHEN the level button is clicked
THEN the program create a new game in that level.

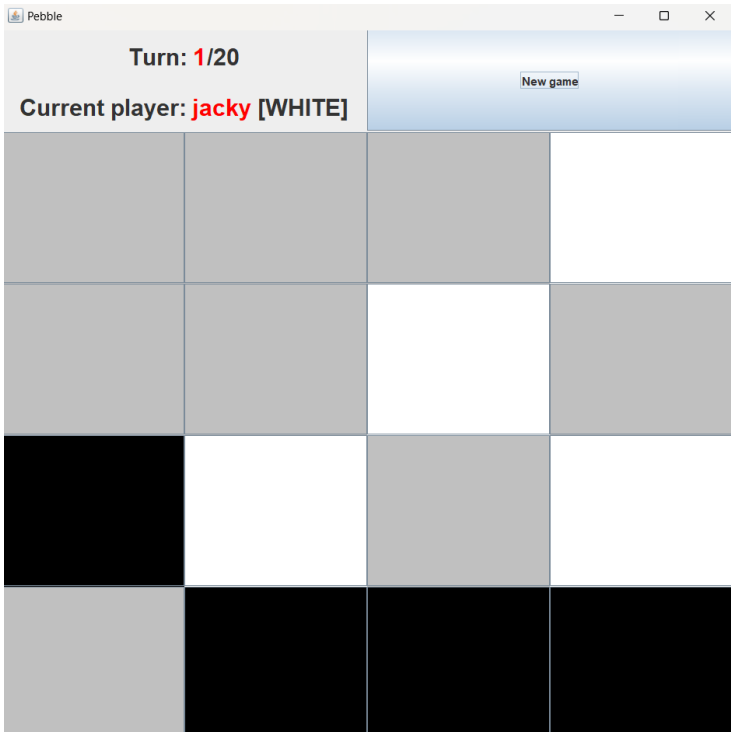

Figure 13: Player(s) click New game button.

Figure 13 user story:

AS A user
I WANT TO start a new game
GIVEN the current game is visible
WHEN the New game button is clicked
THEN the program create a new game in that level and delete the current game.

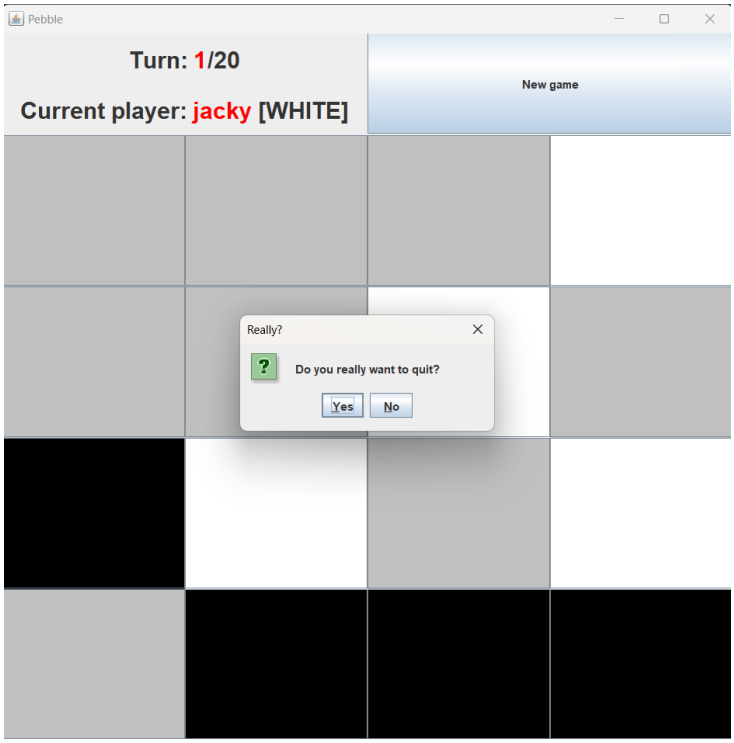

Figure 14: Player(s) click "X" icon in the game window.

Figure 14 user story:

AS A user
I WANT TO close current game
GIVEN the current game is visible
WHEN the "X" icon is clicked
THEN the program ask for a confirmation.

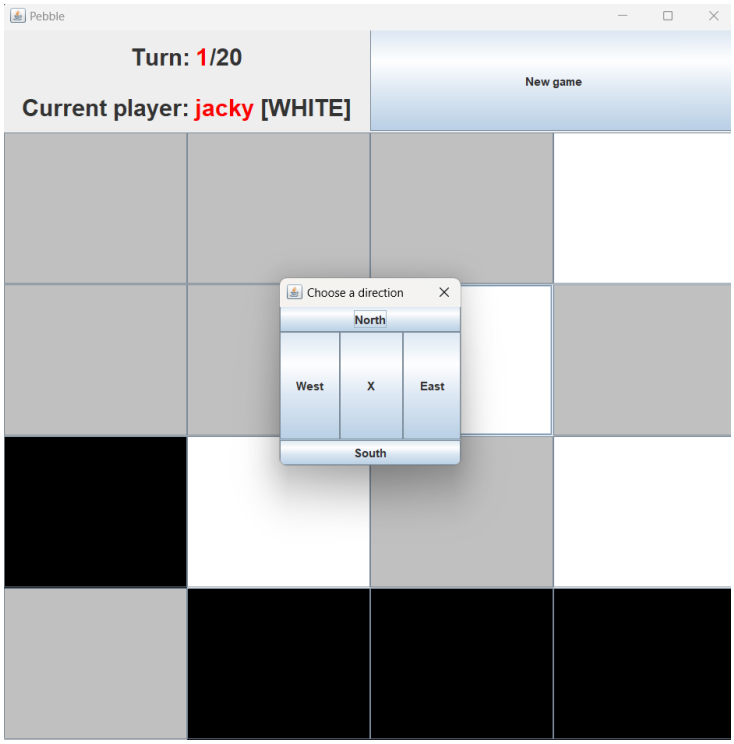

Figure 15: Player(s) click to the Pebble.

Figure 15 user stories:

AS A user
I WANT TO move my pebble
GIVEN the current game is visible
WHEN the pebble is clicked
THEN the direction controller is visible.

AS A user
I WANT TO move another pebble
GIVEN the direction controller is already showed
WHEN the "X" button is clicked
THEN the direction controller is closed, and I can choose another pebble to move.

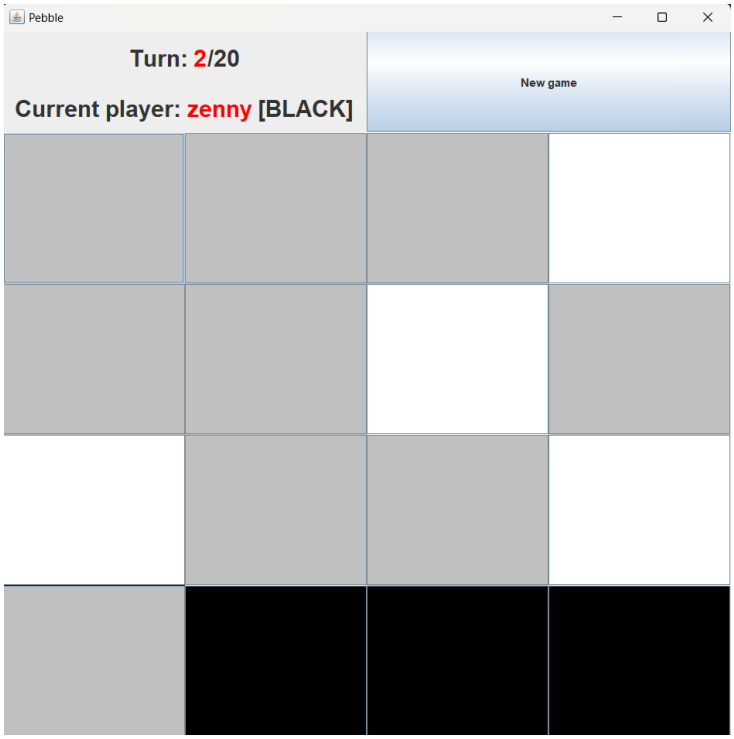

Figure 16: Player(s) click to the Pebble.

Figure 16 user story:

AS A user
I WANT TO move my pebble in a direction
GIVEN the direction controller is visible
WHEN direction (not "X") is clicked
THEN pebble is move to that direction and push other consecutive pebbles in front of it by 1 step, pebbles are in the edge will fall off the game and switch to next player turn.

8 Black Box Testing

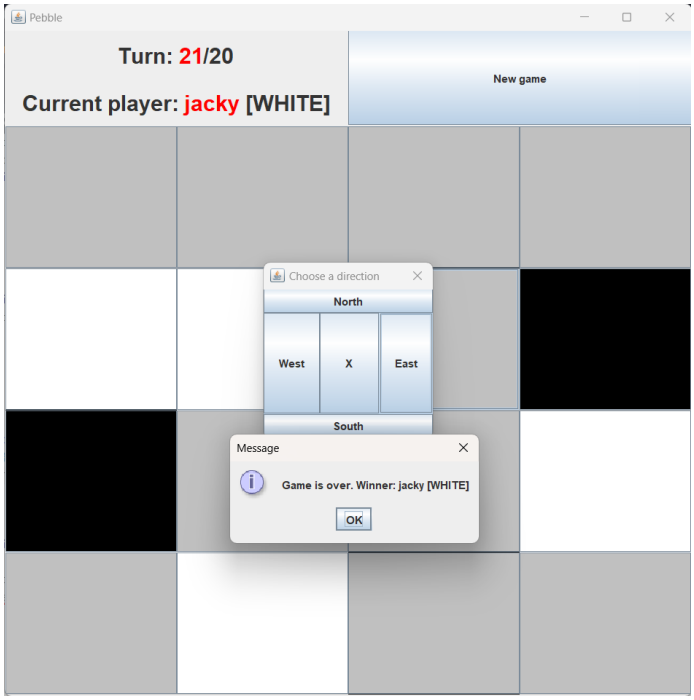

Figure 17: Player(s) win game.

Figure 17 user stories:

AS A user (BLACK)
I WANT TO move my pebble
GIVEN this is the last turn
WHEN direction (not "X") is clicked
THEN pebble moves and I wins if I have more pebbles than WHITE.

AS A user (WHITE)
I WANT TO move my pebble
GIVEN this is the last turn
WHEN direction (not "X") is clicked
THEN pebble moves and I wins if I have more pebbles than BLACK.

AS A user
I WANT TO move my pebble
GIVEN this is the last turn
WHEN direction (not "X") is clicked
THEN pebble moves and DRAW if our pebbles are equals.

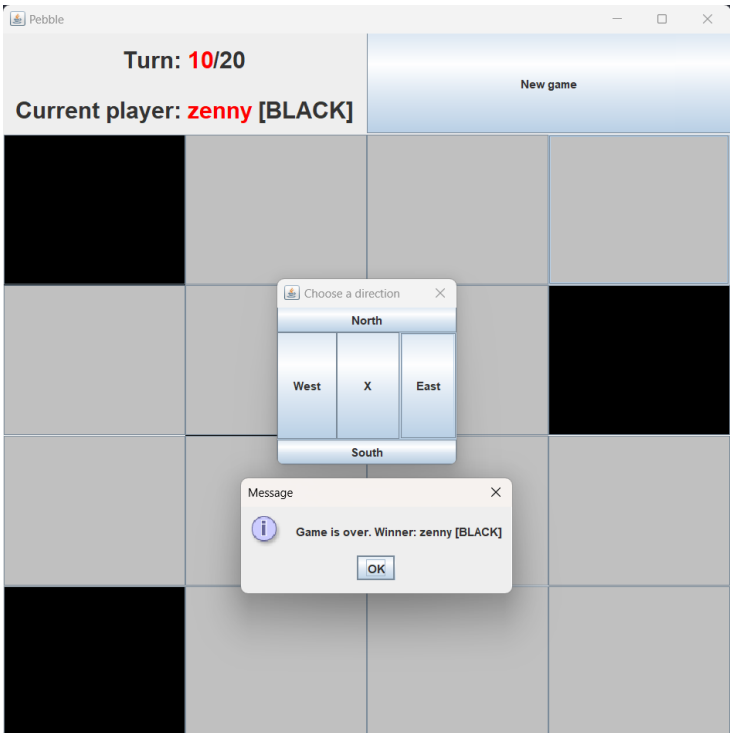

Figure 18: Player(s) win game.

Figure 18 user stories:

AS A user (BLACK)
I WANT TO move my pebble
GIVEN this is not the last turn
WHEN direction (not "X") is clicked
THEN pebble moves and I wins if WHITE does not have any pebbles left.

AS A user (WHITE)
I WANT TO move my pebble
GIVEN this is not the last turn
WHEN direction (not "X") is clicked
THEN pebble moves and I wins if BLACK does not have any pebbles left.

Figure 19: Player(s) click New game button after game is finished.

Figure 19 user story:

AS A user
I WANT TO start new game
GIVEN the game is ended
WHEN "OK" is clicked
THEN the new game is created with the same level and same players.

the end