Proejct 3: The “Target Game” with GUI

an Java Application with a Simple GUI with JavaFX

CISC 3120
Design and Implementation of Software Applications I

The project is to help students who have already had experience in programming in `C++` to achieve the following learning objectives in addition to the learning objectives defined in Projects 1 and 2,

- to be able to design simple Graphical User Interface (GUI),
- and to be able to implement the GUI design using JavaFX.

1 Project Description

We are now converting the text-based “Target Game” application to a GUI application. The Target Game is a single player game. First, the game sets a target on a two-dimensional space *randomly*, and asks the player to guess where the target is. The player wins the game when the player’s guess is correct.

The instructor wrote the start-up code for the project, and the start-up code is in a Maven project in the `sampleprograms` repository. You may browse the code from the following URL,

`https://github.com/CISC3120/sampleprograms/tree/master/TargetGameFX`

1.1 Project Requirement

This is a team project. You and your team collectively must complete the following requirements by making revision to the start-up code.

If you and your team complete *all* of the requirements in this section satisfactorily, the instructor considers the quality of the project as “A”. Be aware whether you as an individual will receive the “A” grade for the project depending also on a project peer evaluation to be conducted anonymously after the submission of the project.

1.1.1 Maximum Guesses

The start-up code permits the player to guesses any number of times before the player guesses correctly. In your project, you will set up a maximum guess threshold in the application. When the player exceeds the threshold without entering a correct guess, the player loses the game. This is in effect one of the requirement in Project 1.

1.1.2 Add Shooting Pane

When you run the start-up code, you can guess the location of the target and enter the guess only using the two `TextField` controls. In the project, you will add a shooting pane to the interface as illustrated in Figure 1 where the shooting pane at at the top-right portion of the interface. Using

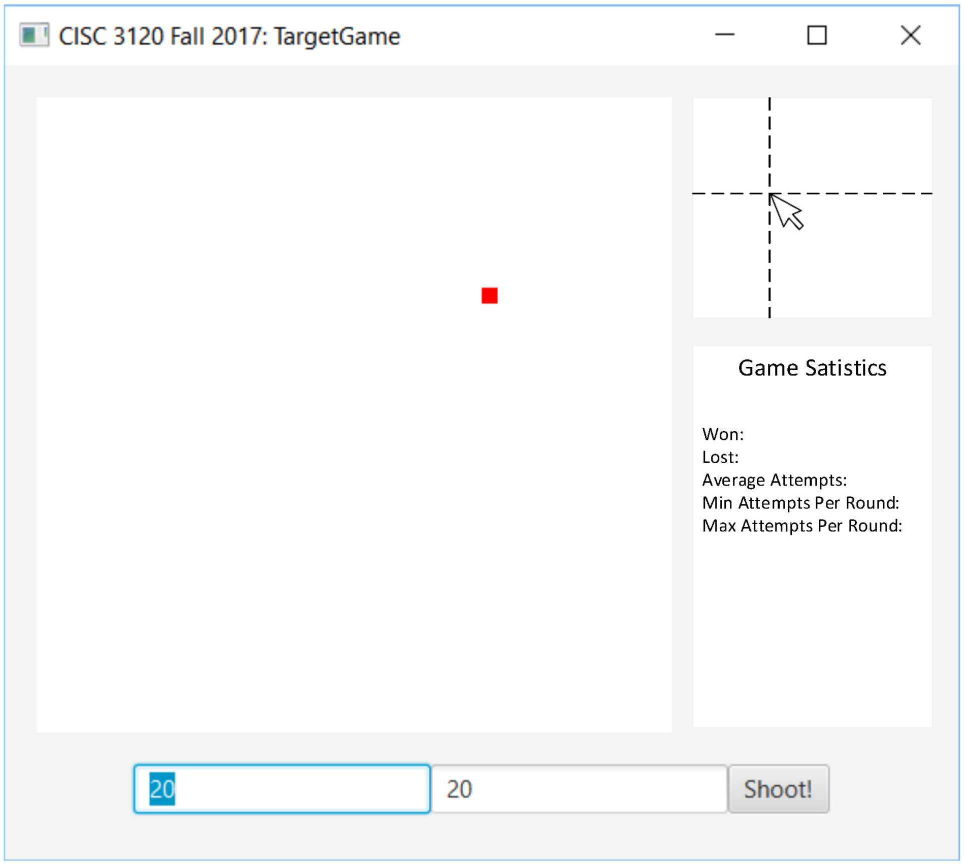

Figure 1: Final GUI of the “Target Game”.

the shooting pane, you can use mouse to guess where the target is. As shown in Figure 1, when the mouse enters the shooting pane, you will display a cross that follows the mouse pointer. When you click the mouse button, a guess is entered and you will update the values in the two `TextField` controls and determine whether the guess is correct.

1.1.3 Add Game Statistics Pane

As illustrated in Figure 1, you will add a game statistics pane that records game statistics. In Figure 1, the game statistics pane is below the shooting pane. We call it a *round* when a player wins by guessing correctly or loses by entering too many guesses. The game statistics must include at least the following,

- the numbers of rounds that the player has won,
- the numbers of rounds that the player has lost,
- the average attempts per round that the player attempted,
- the minimum attempts that the player made among all the rounds,
- and the maximum attempts that the player made among all the rounds.

1.1.4 Add Simple Style using CSS

The start-up code does not apply any style to the user interface. You will add a style to the user interface using one or more Cascading Style Sheets (CSS).

1.2 Bonus Project Requirement

If you and your team complete any one of the following two *bonus* requirements *in addition to* the requirements above, the instructor considers the quality of the project as "A+".

1.2.1 Add Target of Different Shape and Size

This is in effect a requirement in Project 2. You make a *target* a hierarchy of different shapes. A *target* can also take different size.

The target shapes and sizes must satisfy the following requirements.

- You will implement at least three shapes of targets, the Point shape, the Rectangle shape, and the Triangle shape.
- When the game runs, the type of shape, the size of the shape, and the position of the shape is randomly generated at *every round*.

1.2.2 Display Shot when Target is Missed

The start-up code uses an `Alert` window to display whether a guess is correct or wrong. To meet this requirement, you will display a "shot" in the main canvas (where the target is) when the player guesses wrongly, i.e., misses the target. You may choose any shape or size of your "shot", for instance, you may display the shot as a little box similar to the target, but with a different color, or better yet, you can use an image that resembles a smashed bullet.

2 Project Invitation

Each team shall elects project coordinator for this project. The coordinator has the following responsibility,

- to accept the assignment invitation via the Github classroom;
- to clone the team project repository;
- to copy and add the start-up project to your own project repository
- to commit and push the project, and to inform team members that the project set-up is ready

The team members shall accept the invitation and clone the project repository. The collaboration and project development continues.

The project assignment invitation is at

`https://classroom.github.com/g/hvb8ScZl`

3 Submission

Submit your project by pushing your project to Github by 11:59PM, Wednesday, November 1.

4 Appendix: From Anonymous Class to Lambda Expression

Java 8 starts to support *Lambda expression*, a concept ubiquitous in functional programming languages, and now widely supported in many programming languages. In the start-up code of Project 3, you will see a few lines of code similar to,

```
xGuessedTextField.setOnMouseClicked(
    (MouseEvent e) -> {xGuessedTextField.selectAll();}
);
```

where Line 2 that may look strange to you is a Lambda expression. Lambda expression is a more concise way to write an *anonymous class* that implements a single abstract method. We can rewrite the above example using *anonymous class* and the code would look as follows,

```
xGuessedTextField.setOnMouseClicked(
    new EventHandler<MouseEvent> () {
        @Override
        public void handle(MouseEvent e) {
            xGuessedTextField.selectAll();
        }
    }
);
```

which is much more cumbersome. Most writings in lines 2 – 7 are boilerplate-like code. First, the `setOnMouseClicked(MouseEvent<? super MouseEvent> value)` method of the `TextField` tells us that we must implement the interface below to invoke the method,

```
public interface MouseEvent<? super MouseEvent> {
     public void handle(MouseEvent e);
}
```

Second, when we implement the interface in a class, we must override the abstract method `public void handle(MouseEvent e)` with a concrete implementation. That is to say, every single time when we invoke the `setOnMouseClicked(MouseEvent<? super MouseEvent> value)` method, we will write the following every single time when we write it as an *anonymous class*,

```
    new EventHandler<MouseEvent> () {
        @Override
        public void handle(MouseEvent e) {
            .....
        }
    }
```

Using an *Lambda expression*, we can write it more concisely. First, we write the parameter list of the abstract method we must implement. In the above example, since the parameter list of the abstract method `public void handle(MouseEvent e)` is `(MouseEvent e)`, we write

```
    (MouseEvent e)
```

Second, we write the "mapping" operator `->`,

```
    ->
```

where "mapping" has in effect the same meaning as "mapping", denoted as $\to$ when you define a mathematical function, such as, $f : x \to x^2$ and $f : (x, y) \to x^2 + y^2$.

Third, we write the statements inside the method. In the above example, we write,

```
    {
            xGuessedTextField.selectAll();

    }
```

Putting the three parts together, we arrive at,

```
    (MouseEvent e) -> {xGuessedTextField.selectAll();}
```

4.1 Discussion

4.1.1 Writing Lambda Expression More Concisely

If the statements in the abstraction method has only one statement, we do not have to write the { and }. The above example can be simplified to

```
    (MouseEvent e) -> xGuessedTextField.selectAll()
```

If the arguments are of the same type of the parameters, we can forgo the type in the parameter list. If the parameter list has only one parameter, we can leave out the `(` and `)`. That is to say, for the above example, we simply write,

```
    e -> { xGuessedTextField.selectAll(); }
```

Combining the above two, we can write the *Lambda expression* in the most concise form,

```
    e -> xGuessedTextField.selectAll()
```

which means the code snippet in the beginning of this appendix can be rewritten as,

```
xGuessedTextField.setOnMouseClicked(e -> xGuessedTextField.selectAll());
```

4.1.2 Functional Interface

Interfaces like the `EventHandler` interface as shown below has only one abstract method. In Java, we call an interface like this a *functional interface.*

```
public interface MouseEvent<? super MouseEvent> {
    public void handle(MouseEvent e);
}
```