软件体系结构 作业18

22920212204392 黄勖

1　用GUI改写策略模式的例子

在本作业中，我使用JavaFX来绘制GUI

```
public class StrategyApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(StrategyApplication.class.getResource( name: "strategy.fxml"));
        Scene scene = new Scene(fxmlLoader.load(),  v: 1200,  v1: 800);
        stage.setTitle("猜拳游戏");
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) { launch(); }
}
```

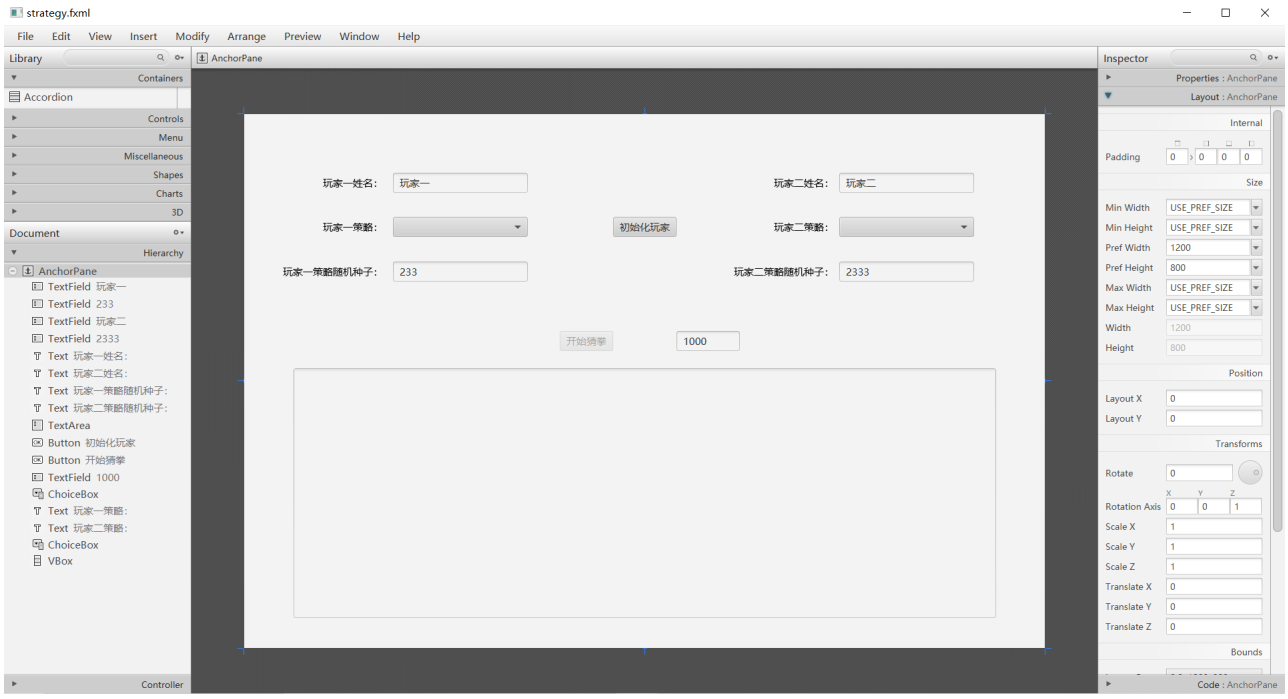
1.1　设置项目依赖

在Maven项目中添加JavaFX的相关依赖（本地运行环境为Java 17，若Java版本不同，pom.xml中JavaFX的版本可能需要更替）。

```
<dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-controls</artifactId>
    <version>17.0.1</version>
</dependency>
<dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-fxml</artifactId>
    <version>17.0.1</version>
</dependency>
```

1.2　绘制GUI界面

使用JavaFX进行简单的GUI界面绘制。



1.3 初始化下拉框

通过实现`Initializable`接口的`initialize`方法，初始化下拉框的可选值和初始值。

```
public class StrategyController implements Initializable {
```

1.4 初始化用户按钮功能

确保用户名和种子不为空，并确保种子是整数类型。

```
4 个用法
String[] allStrategy = new String[]{"ProbStrategy","WinningStrategy","DiscontinuousSameStrategy"};

@Override
public void initialize(URL url, ResourceBundle resourceBundle) {
    playerOneStrategy.getItems().addAll(allStrategy);
    playerTwoStrategy.getItems().addAll(allStrategy);
    playerOneStrategy.setValue(allStrategy[1]);
    playerTwoStrategy.setValue(allStrategy[0]);
}

@FXML
void OnInitPlayerButtonClicked(ActionEvent event) throws Exception {
    if(playerOneName.getText().isEmpty()||playerTwoName.getText().isEmpty()
            ||playerOneSeed.getText().isEmpty()||playerTwoSeed.getText().isEmpty()){
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("用户名及种子不可为空");
        alert.setHeaderText(null);
        alert.setContentText("用户名及种子不可为空，请重新输入！");
        alert.showAndWait();
        return;
    }
```

使用选定的策略初始化用户，并设置页面各个组件的可交互性。

```
player1 = new Player(playerOneName.getText(),
        (Strategy) Class.forName( className: StrategyController.class.getPackage().getName()+'.'+playerOneStra
player2 = new Player(playerTwoName.getText(),
        (Strategy) Class.forName( className: StrategyController.class.getPackage().getName()+'.'+playerTwoStra
playerOneName.setDisable(true);
playerTwoName.setDisable(true);
playerOneSeed.setDisable(true);
playerTwoSeed.setDisable(true);
initPlayerButton.setDisable(true);
playerOneStrategy.setDisable(true);
playerTwoStrategy.setDisable(true);
guessButton.setDisable(false);
```

1.5 实现猜拳功能

确保次数不为空，并判断次数是否为整数类型正数。

```
@FXML
void OnGuessButtonClicked(ActionEvent event) {
    if(guessNumber.getText().isEmpty()){
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("猜拳次数不可为空");
        alert.setHeaderText(null);
        alert.setContentText("猜拳次数不可为空，请重新输入！");
        alert.showAndWait();
        return;
    }
```

```
int num;
try{
    num = Integer.parseInt(guessNumber.getText());
    if(num<=0) throw new Exception();
}
catch (Exception e){
    Alert alert = new Alert(Alert.AlertType.ERROR);
    alert.setTitle("猜拳次数必须为int类型正数");
    alert.setHeaderText(null);
    alert.setContentText("猜拳次数必须为int类型正数，请重新输入！");
    alert.showAndWait();
    return;
}
```

进行指定次数的猜拳，我们将结果缓存在`StringBuilder`中，以避免大次数时实时输出导致的卡顿。为了避免内存溢出，在文本长度超出10000个字符时清空区域。

```
StringBuilder stringBuilder = new StringBuilder();
for (int i = 0; i < num; ++i) {
    if(stringBuilder.length() > 10000){
        stringBuilder.setLength(0);
    }
    Hand nextHand1 = player1.nextHand();
    Hand nextHand2 = player2.nextHand();
    if (nextHand1.isStrongerThan(nextHand2)) {
        stringBuilder.append("Winner:" + player1+'\n');
        player1.win();
        player2.lose();
    } else if (nextHand2.isStrongerThan(nextHand1)) {
        stringBuilder.append("Winner:" + player2+'\n');
        player1.lose();
        player2.win();
    } else {
        stringBuilder.append("Even..."+'\n');
        player1.even();
        player2.even();
    }
}
stringBuilder.append("Current result:\n");
stringBuilder.append("" + player1+'\n');
stringBuilder.append("" + player2+'\n');
showTextArea.appendText(stringBuilder.toString());
```

1.6 添加新策略

增加一种新策略，该策略确保玩家不会连续出相同的手势。

```
public class DiscontinuousSameStrategy implements Strategy{
    3 个用法
    private Random random;
    3 个用法
    private Hand prevHand;
```

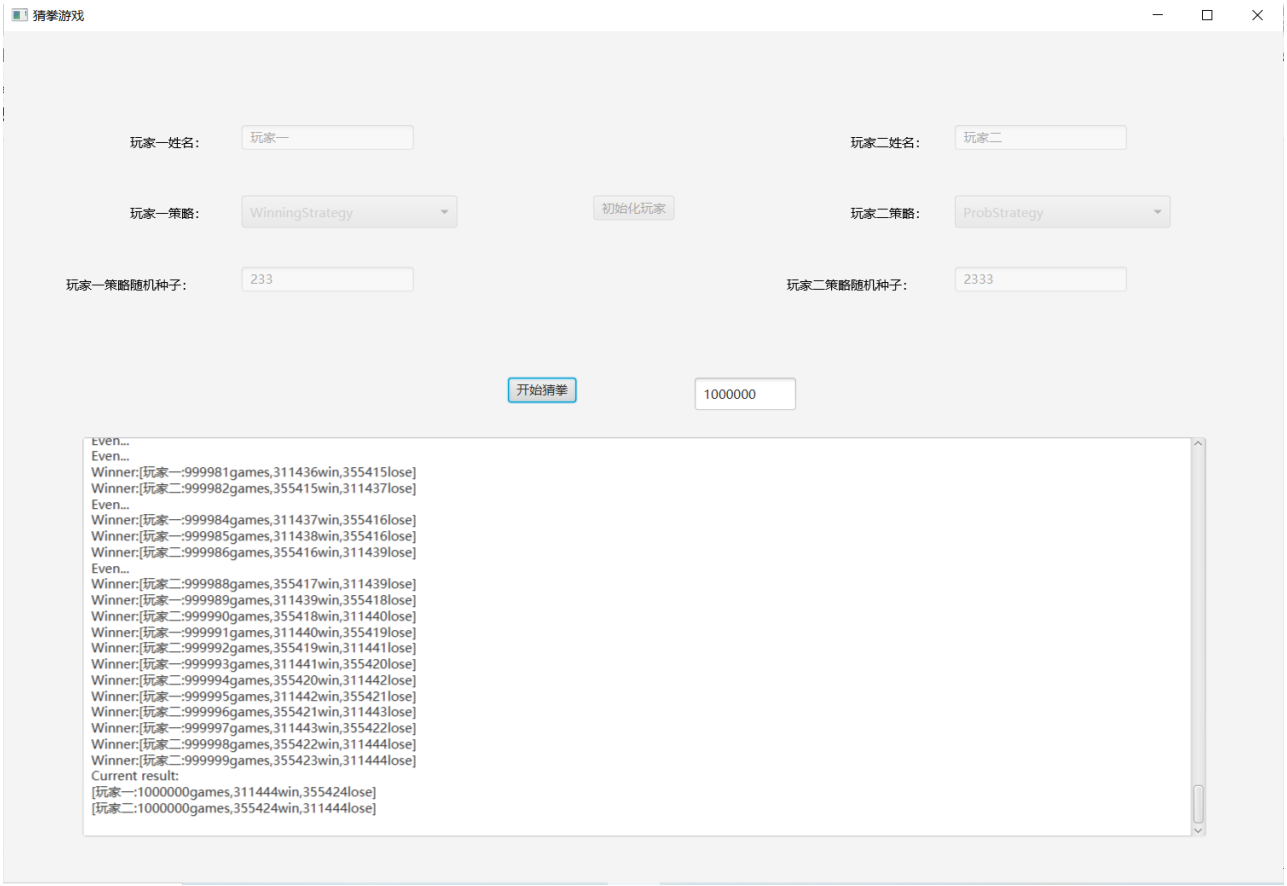
```
@Override
public Hand nextHand() {
    Hand curHand;
    while(prevHand==(curHand = Hand.getHand(random.nextInt( bound: 3)))){}
    return prevHand = curHand;
}
```

与`WinningStrategy`类似，这种策略不需要学习，因为每轮手势与是否获胜无关。

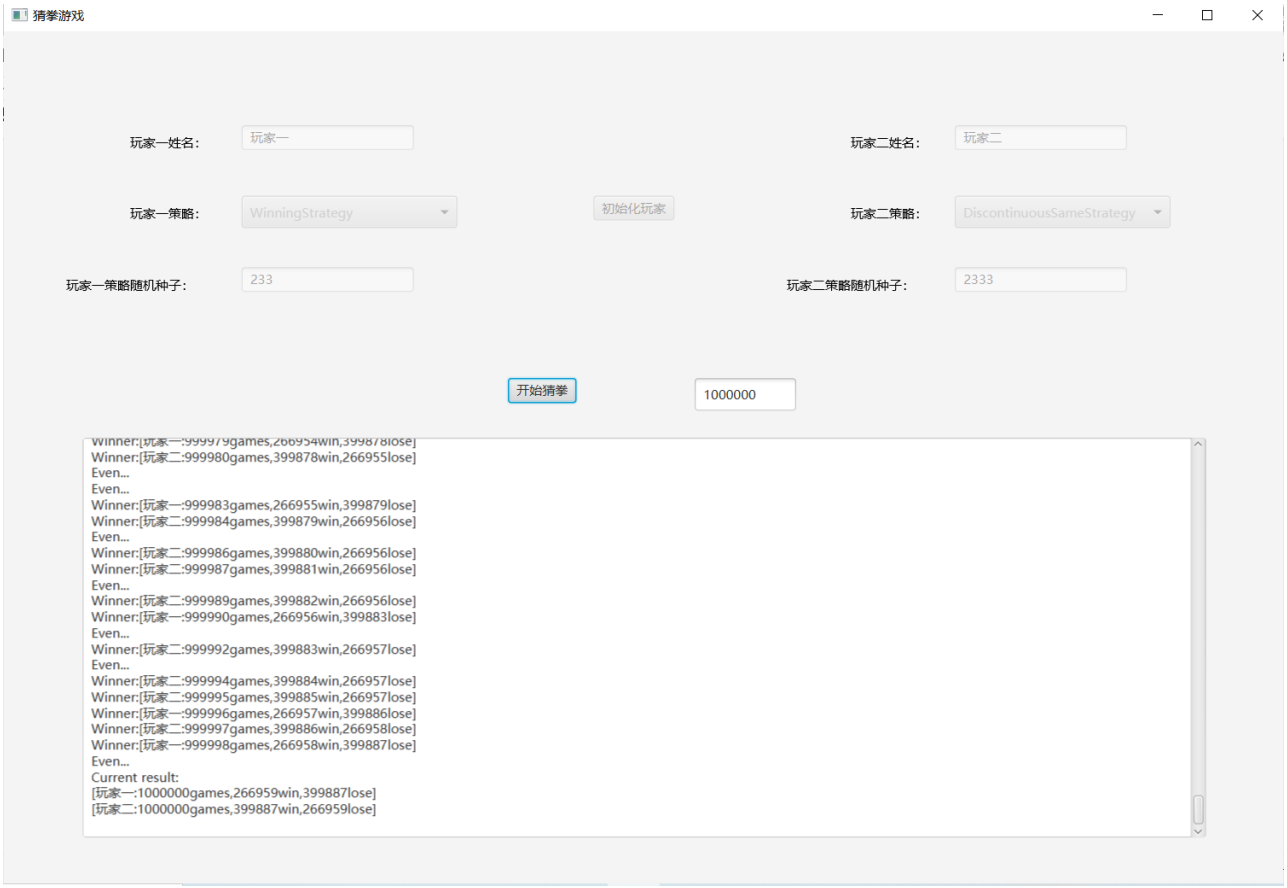
```
@Override
public void study(boolean win) {
    // this strategy don't need to study
}
```

1.7 策略对比实验

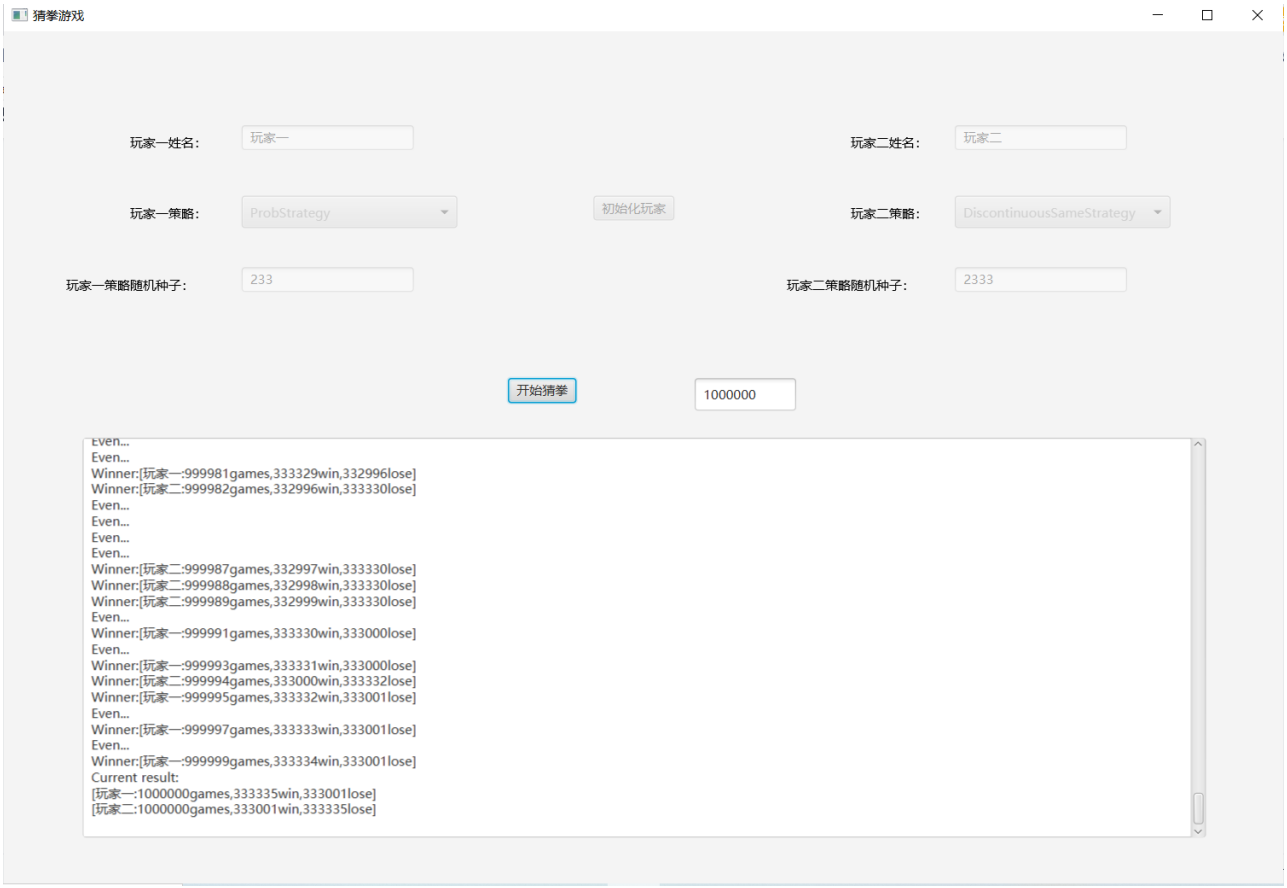
- **WinningStrategy vs ProbStrategy**：进行100万次猜拳，`ProbStrategy`明显克制`WinningStrategy`。（随机种子分别为233与2333，相同种子可复现结果）



- **WinningStrategy vs DiscontinuousSameStrategy**：进行100万次猜拳，`DiscontinuousSameStrategy`明显克制`WinningStrategy`，且比`ProbStrategy`更强。（随机种子分别为233与2333，相同种子可复现结果）



- **ProbStrategy vs DiscontinuousSameStrategy**：进行100万次猜拳，`ProbStrategy`与`DiscontinuousSameStrategy`相差无几，因为`ProbStrategy`预测对手手势的依据与`DiscontinuousSameStrategy`无关。（随机种子分别为233与2333，相同种子可复现结果）



1.8 注意事项

在本地运行时，进行1000万次猜拳时会出现约2~3秒的卡顿。如果希望一次进行更多的模拟，需要等待更长时间。