

# INTRODUCTION

The purpose of this assignment is to develop a dice game called Yahtzee by applying Object Oriented Programming (OOP). In this game players try to achieve the highest score by rolling five dices. The game consists of thirteen rounds for each player. In each round a player rolls dice and then decides whether to re roll dice/dices or to score from the given thirteen categories. However a player can only roll the dice a total of three times. In the end the player with the higher score wins. The game supports both single player and multiplayer play. Java platform was used to develop this game.

Moreover the assignment documents the testing of the application incorporating all its functionalities; furthermore a fully detailed UML diagram has also been added.

# TASK 1

This application is a Yahtzee Game. I have used Object Oriented Programming to develop this game. I have constructed the Graphical User Interface (GUI), implemented the dice rolling and point scoring mechanism. I have also included single player and multiplayer gameplay. I have used various tools during the development of this application such as functions, loops, classes, objects etc. Below are screenshots of my application interface.

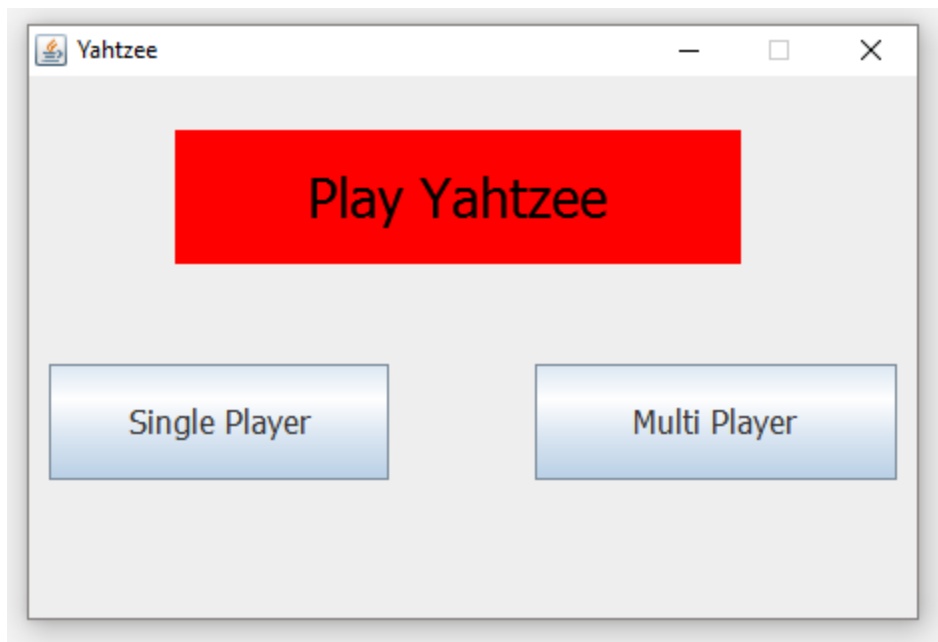


Figure 1 Starting Frame of my application

Single Player Yahtzee

Roll Dice

	SinglePlayer
Ones	
Twos	
Threes	
Fours	
Fives	
Sixes	
Three Of a Kind	
Four Of a Kind	
Small Straight	
Large Straight	
Full House	
Chance	
Yahtzee	
TOTAL SCORE	0

Times Dice Rolled : 0

Figure 2 Single Player



### Figure 3 Multiplayer

# TASK 2

## *Testing Data:*

### *Data Selection and execution:*

For Testing I have decided to include -

- Single player and Multiplayer button to make sure that my game supports both single player and multiplayer gameplay
- The Roll button to check whether the button rolls dices successfully or not
- The Dice Panel to check whether I can select which dices that I want to re roll.
- The Reroll button to test whether the dices selected are rerolled successfully or not.
- All the thirteen custom JLabels to check that the user can click on them to confirm their score for each round, to test that the scoring is correct according to given condition, to test that total of all the scores is correct and check the game flow in both single player and multiplayer.

# Black box Testing:

**Test Plan** – For this testing I have decided to check all the functionalities of my application through a general user's perspective. In this test all the functionalities of the application are tested thoroughly to check whether they are functioning properly or not. Screenshots of successful test results are taken and failure of any test has been mentioned. My test plan for black box testing –

1. Test the Single Player button to test whether my application can handle single player flow.
2. Test the Multi Player button to test whether my application can handle multi player flow.
3. Test the Roll Dice Button to check that my application can roll dices or not.
4. Test the Dice panel to test whether I can select dice or dices that I want to re roll.
5. Test the Re Roll Dice to check that whether I can reroll the dice or dices that I have selected earlier in the dice panel.
6. Test my thirteen custom JLabels, check that they are correctly representing score from the thirteen different categories.
7. Test that the user can click on my Custom JLabel to confirm their score, check the addition of the total score is correct and inspect the flow of the game in both single player and multiplayer.
8. Test that correct scores are shown including the total and assess how the game ends Example (showing the final score, declaring the winner etc.).
9. Finally check that all the required functionalities mentioned in the assignment are present and that they work correctly.

Starting Black Box Testing:

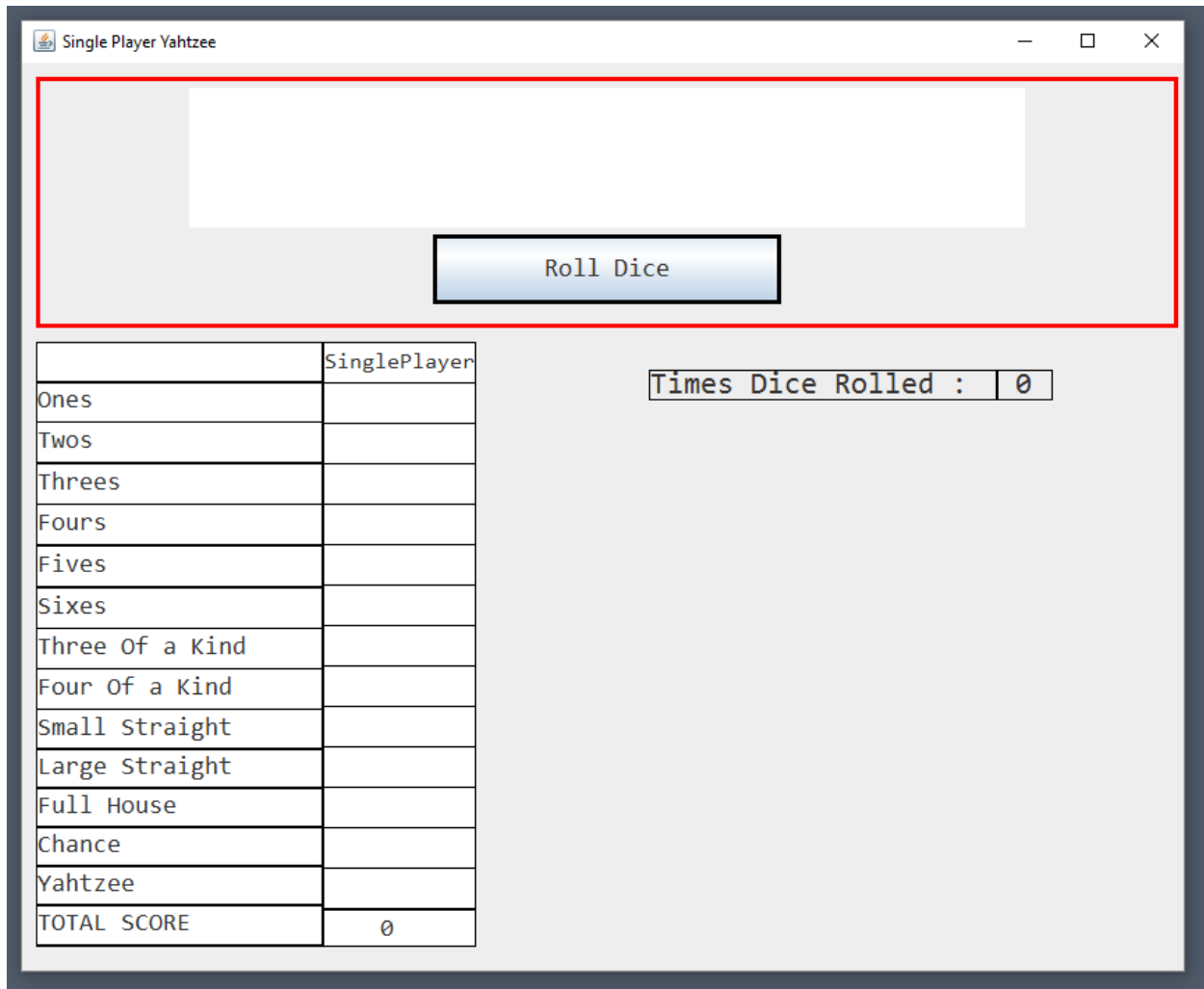


Figure 4 Single Player

Clicking on the single player button on the starting frame starts single player gameplay.

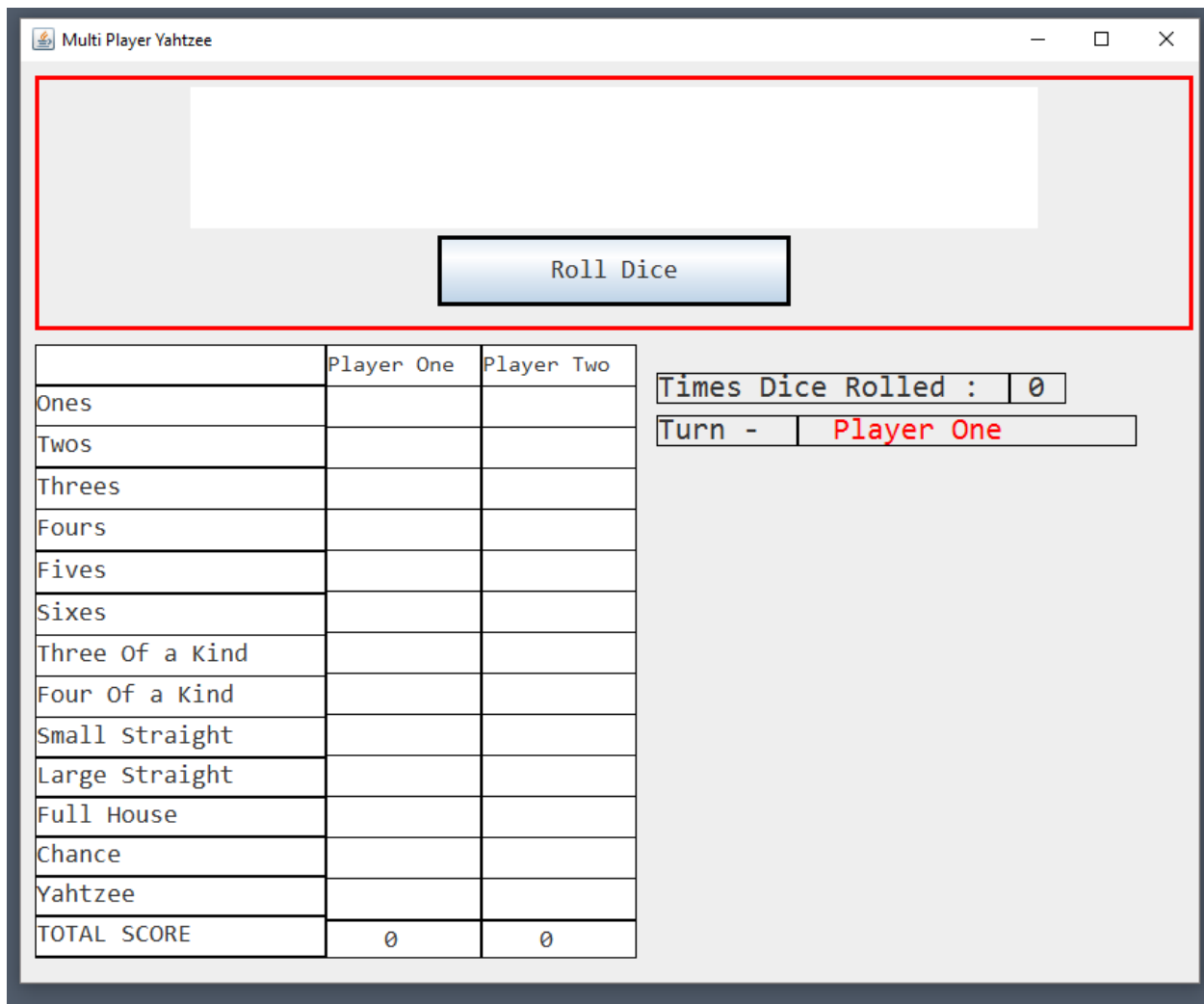


Figure 5 Multi Player

Clicking on the multiplayer on the starting frame starts multiplayer gameplay



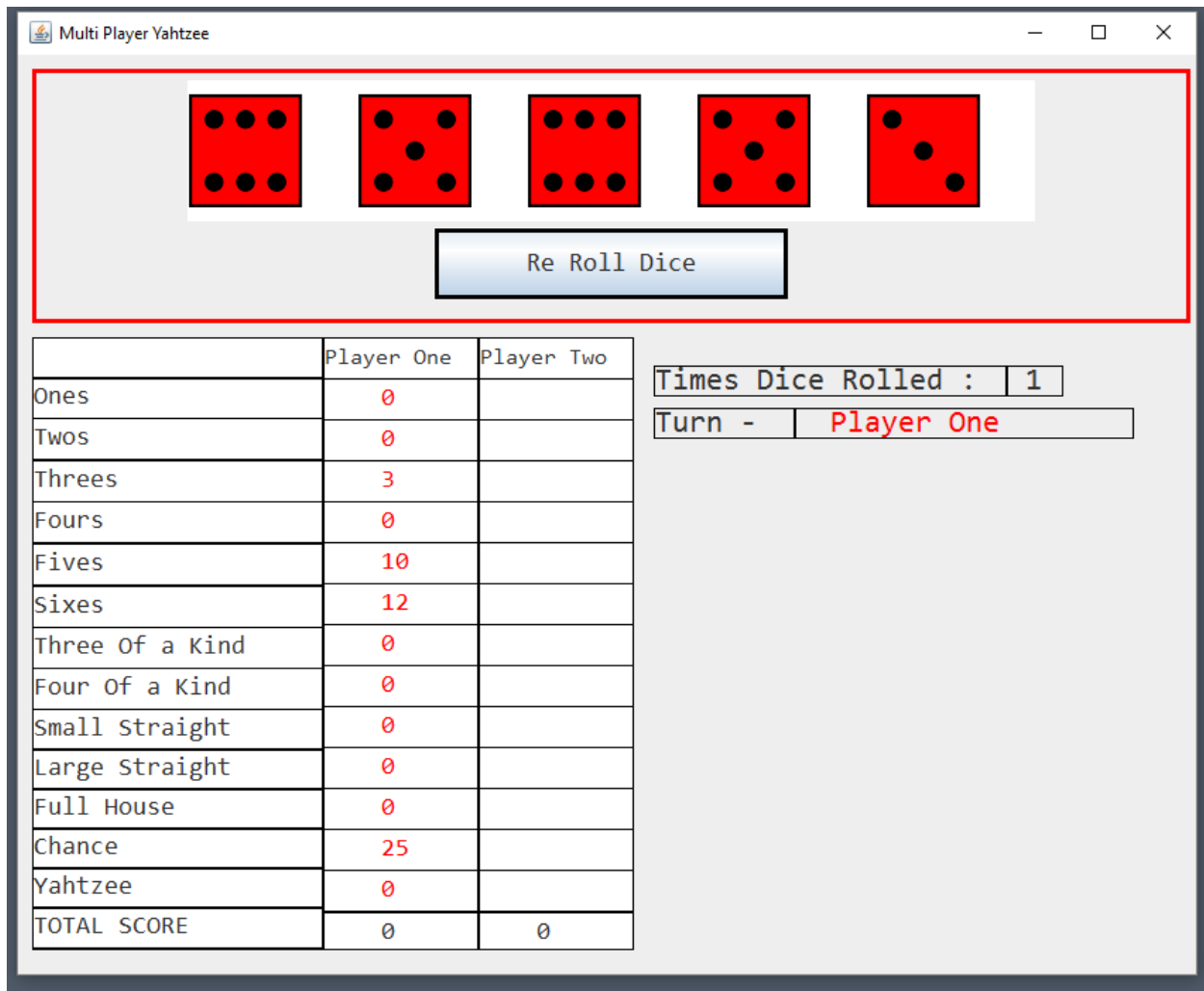


Figure 6 Roll Dice

Clicking on the Roll Dice buttons rolls dice and displays them. The button the hides itself from the interface and Re Roll Dice button appears.

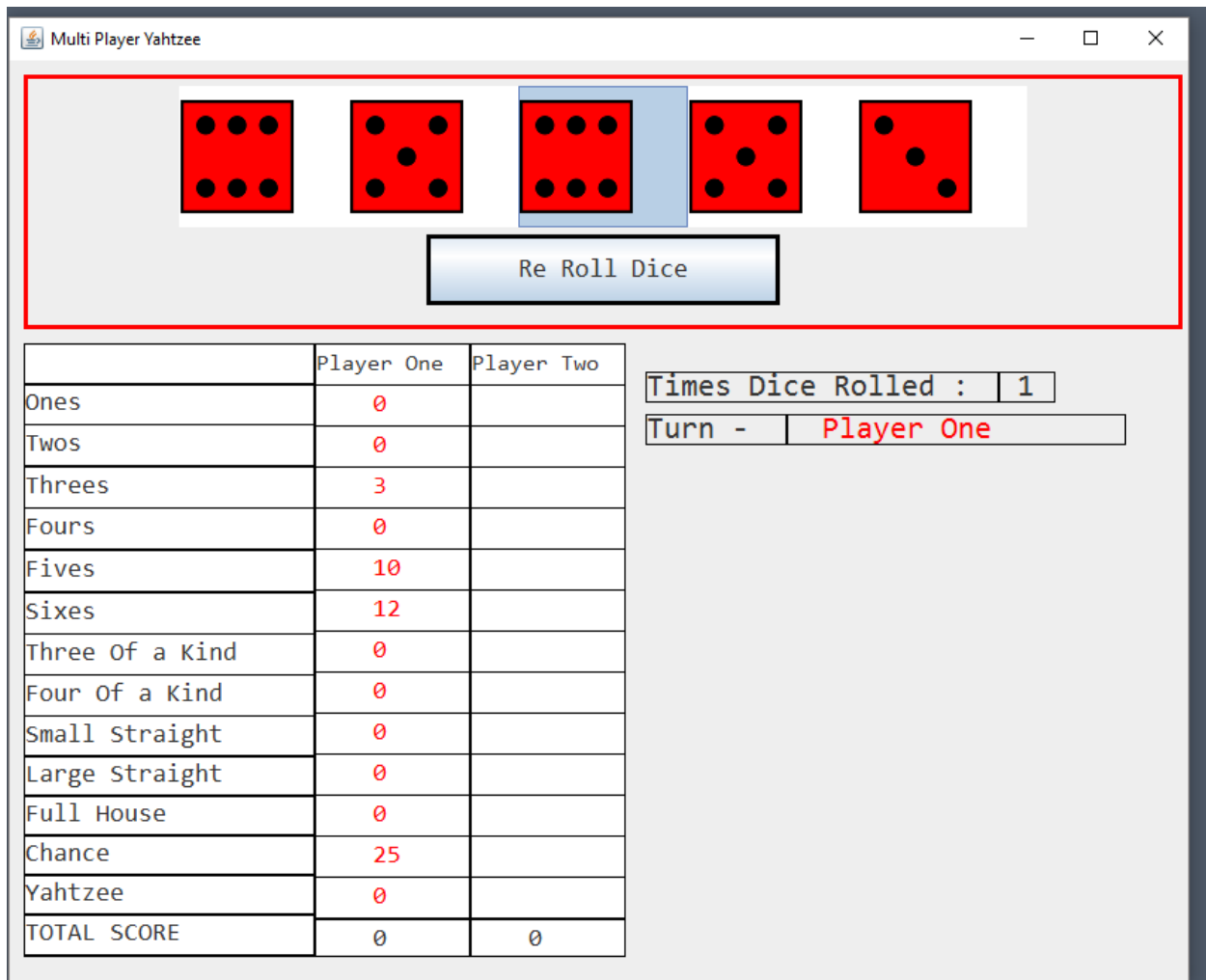


Figure 7 Single Dice Selection

The interface allows me to select a dice by clicking the left mouse button on it.

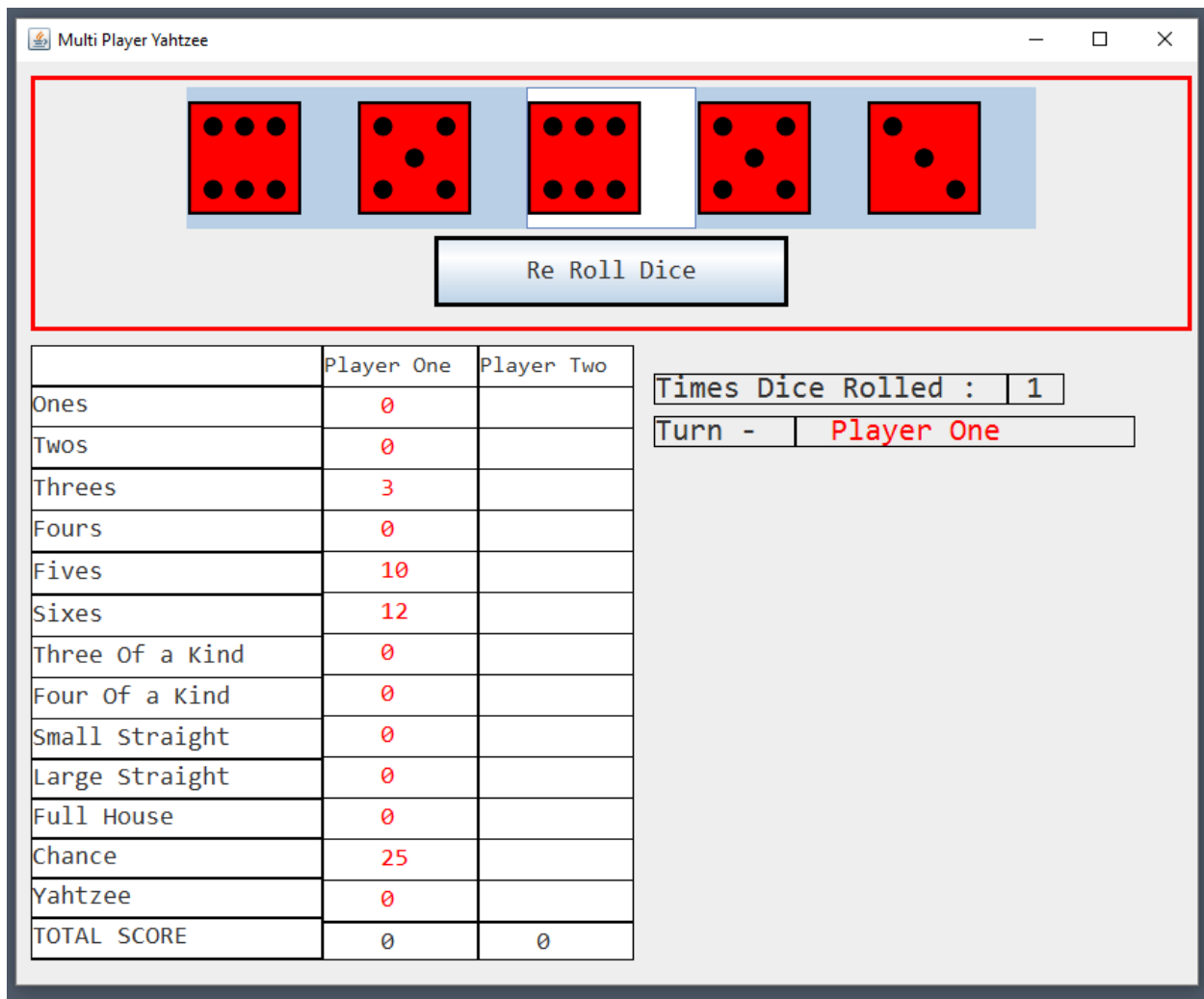


Figure 8 Multiple Dice Selection

The interface allows me select multiple dices by using (Ctrl + Left Mouse Button) to make my selection.

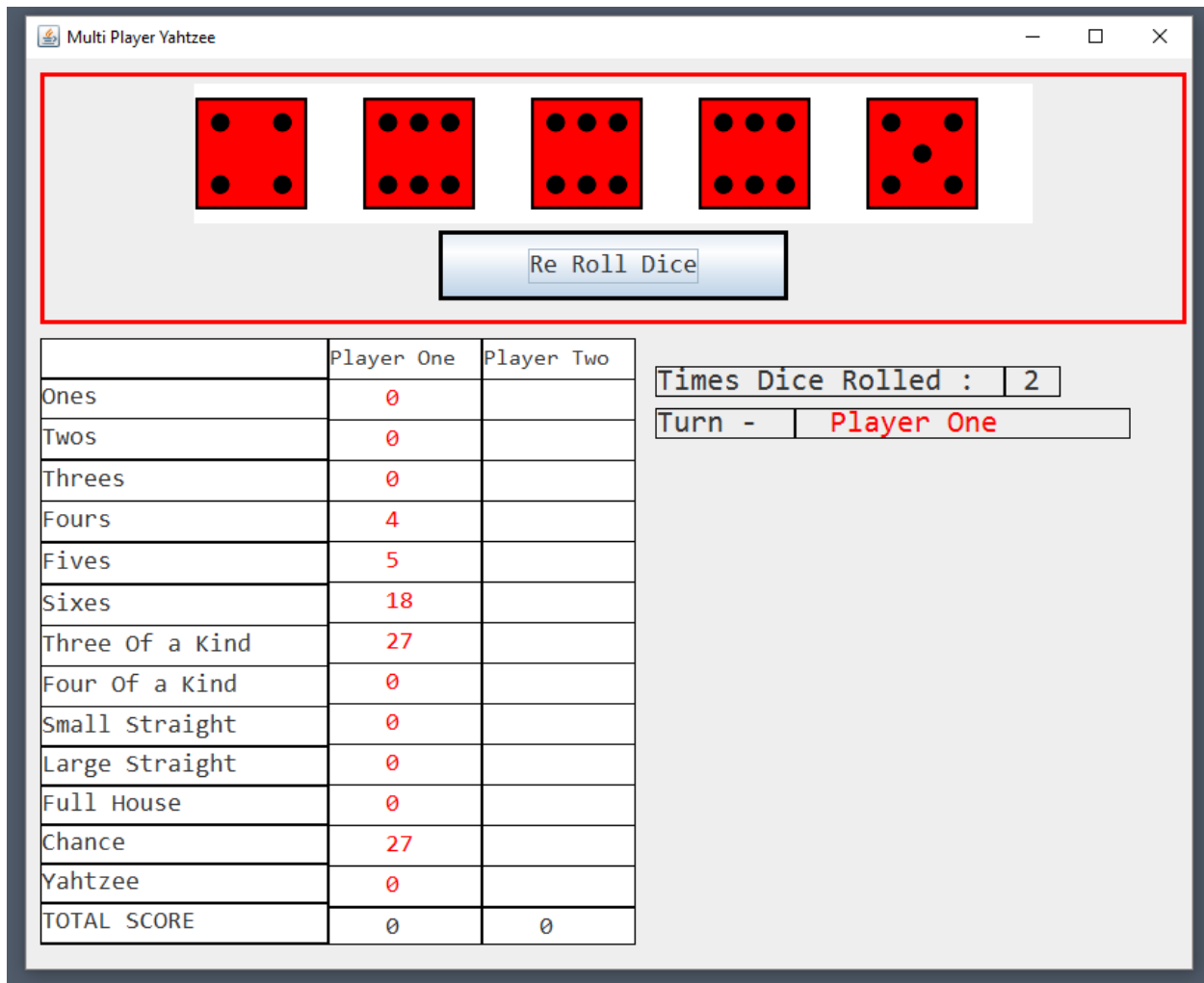


Figure 9 Re Rolling Dices

Clicking on the Re Roll Dice Button make the dices re roll.

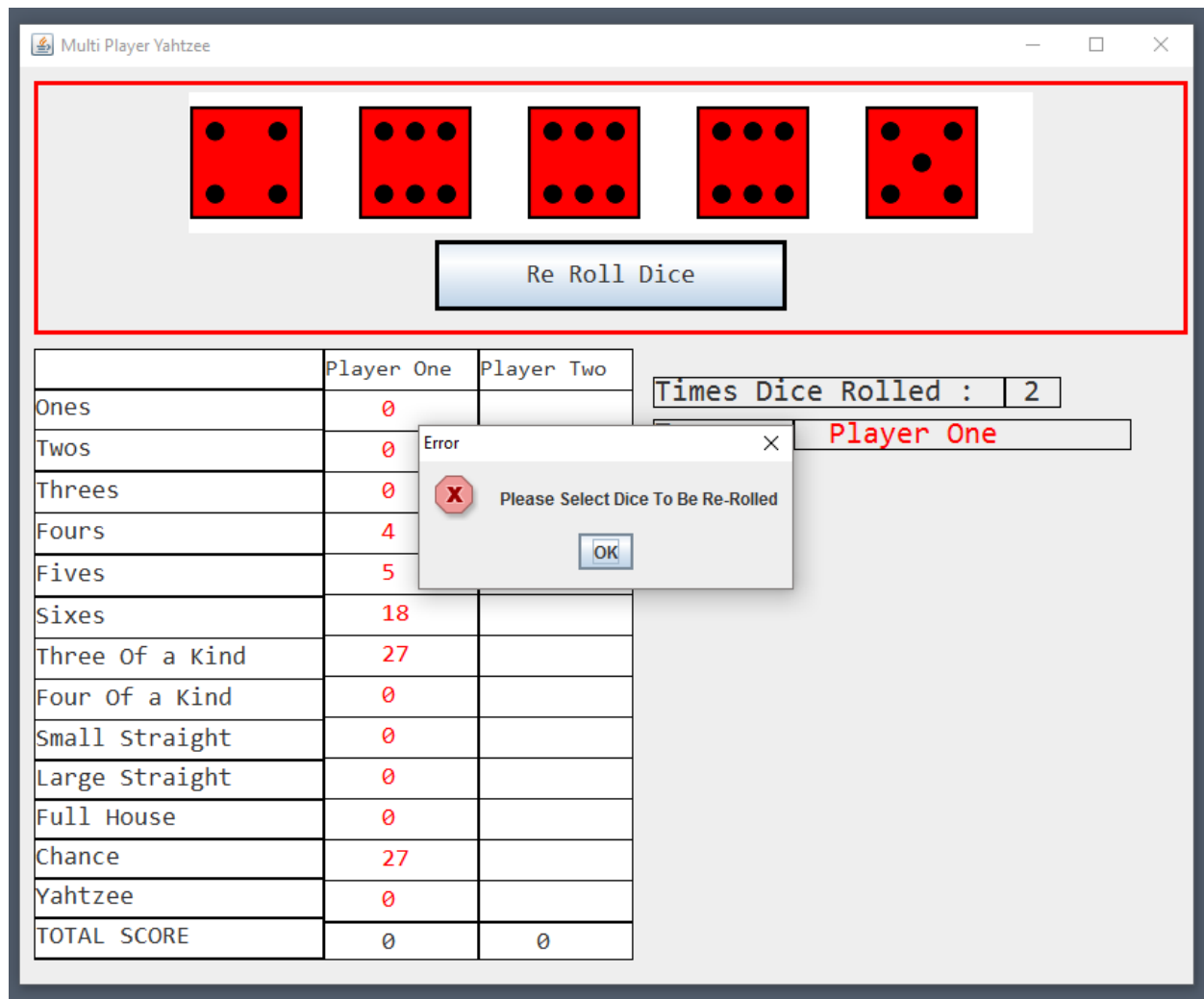


Figure 10 must select dice before rolling them again

The interface makes sure that the user i.e. me must select dice/dices before clicking on the reroll button.

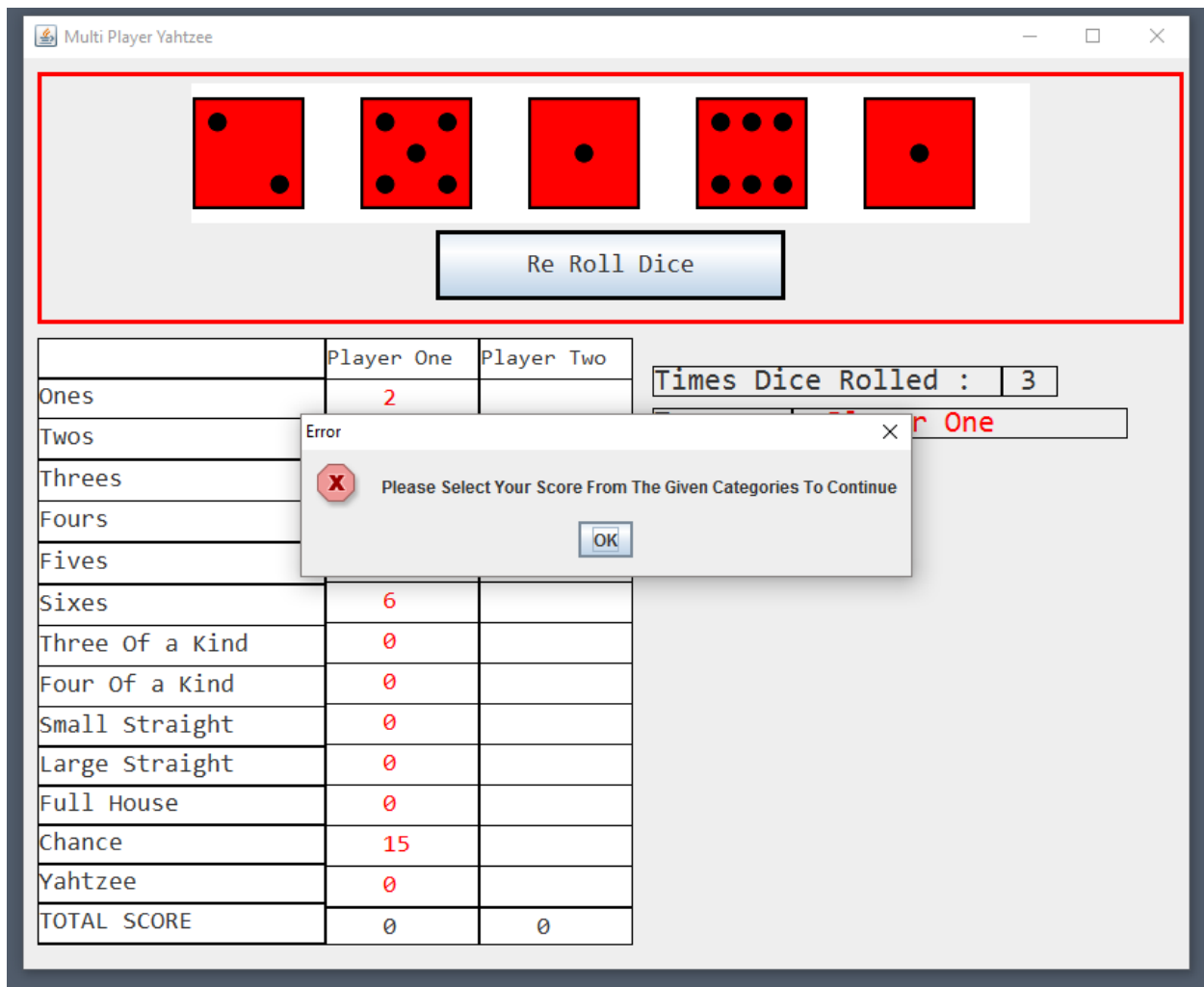


Figure 11 Rolling Dice More than three times???

The interface only allows the user to roll the dices a total of three times. The interface also shows the number of times dice rolled which in the screenshot above shows "3".

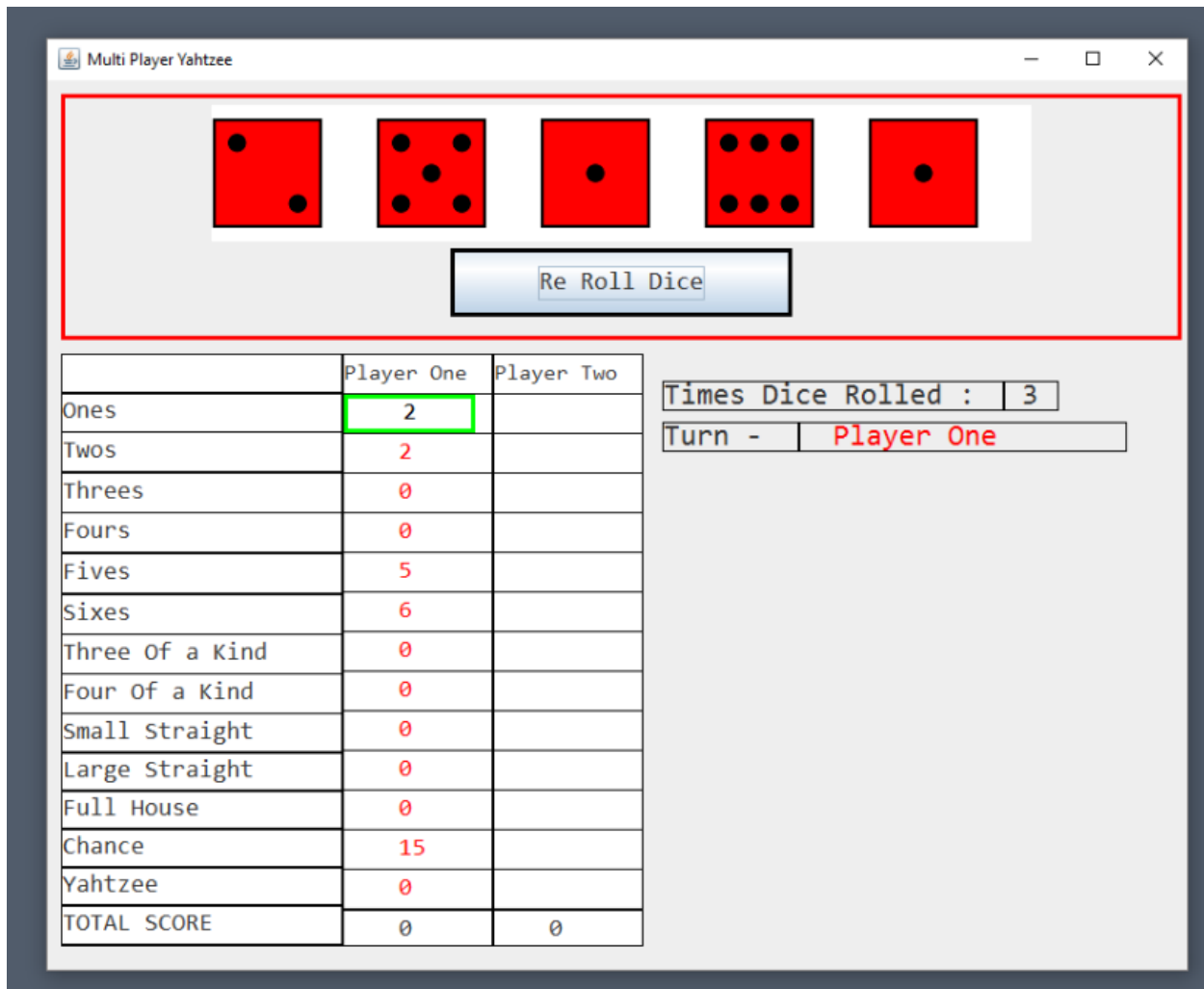
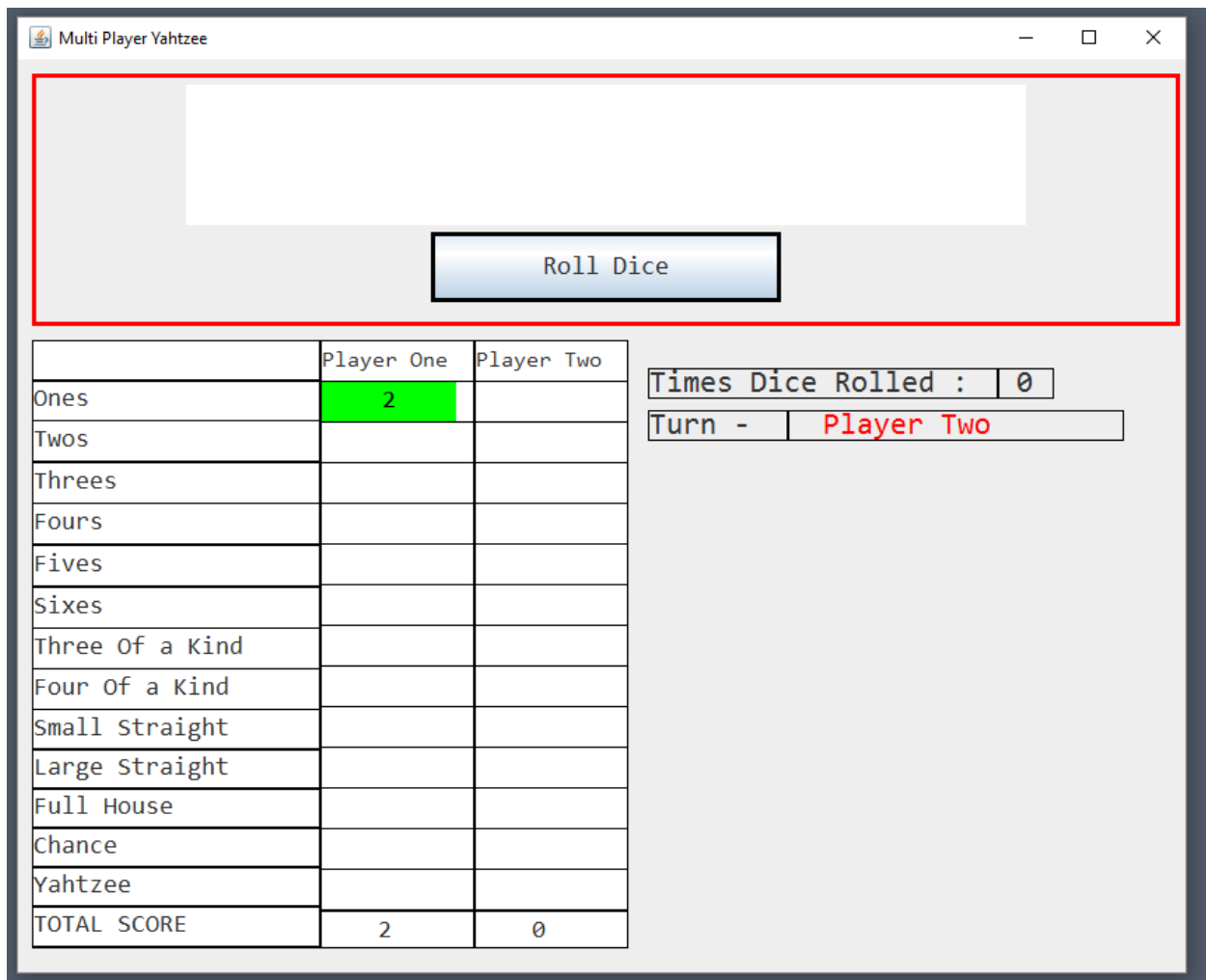


Figure 12 Potential Score Selection (Example a)

The score with foreground color of red represents the points that the user can score for this round. When the user points the mouse cursor over the score the label makes a green border to aware user that they are hovering over the score with their mouse cursor and can select it to confirm score.

**Figure 13 Potential Score Selection (Example b)**





### Figure 14 Score Confirmation (Example a)

When the user clicks on the score the background of the score is turned to green and all previously unconfirmed scores are removed. The total points for the player are also displayed.

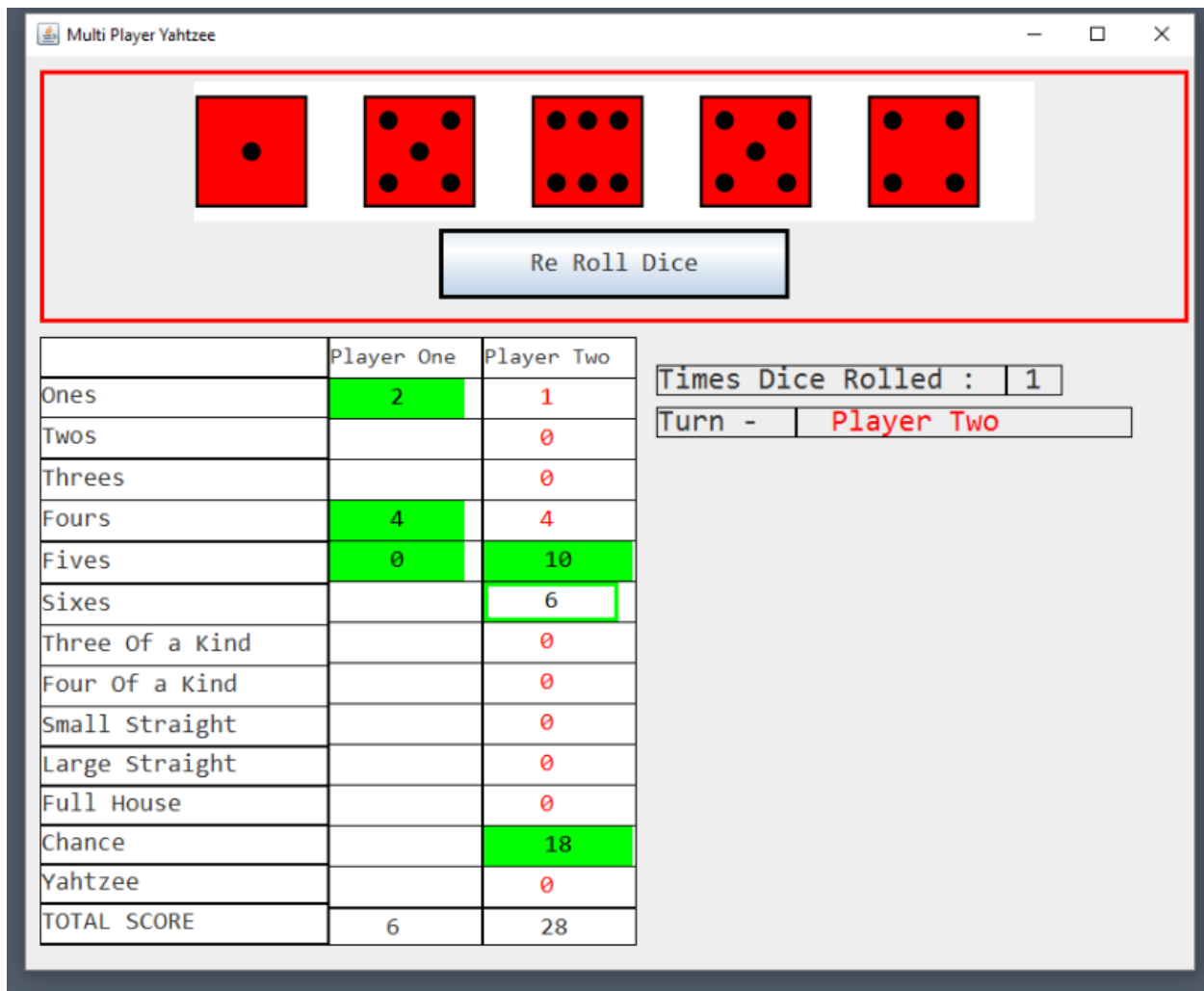


Figure 15 Score Confirmation (Example b)

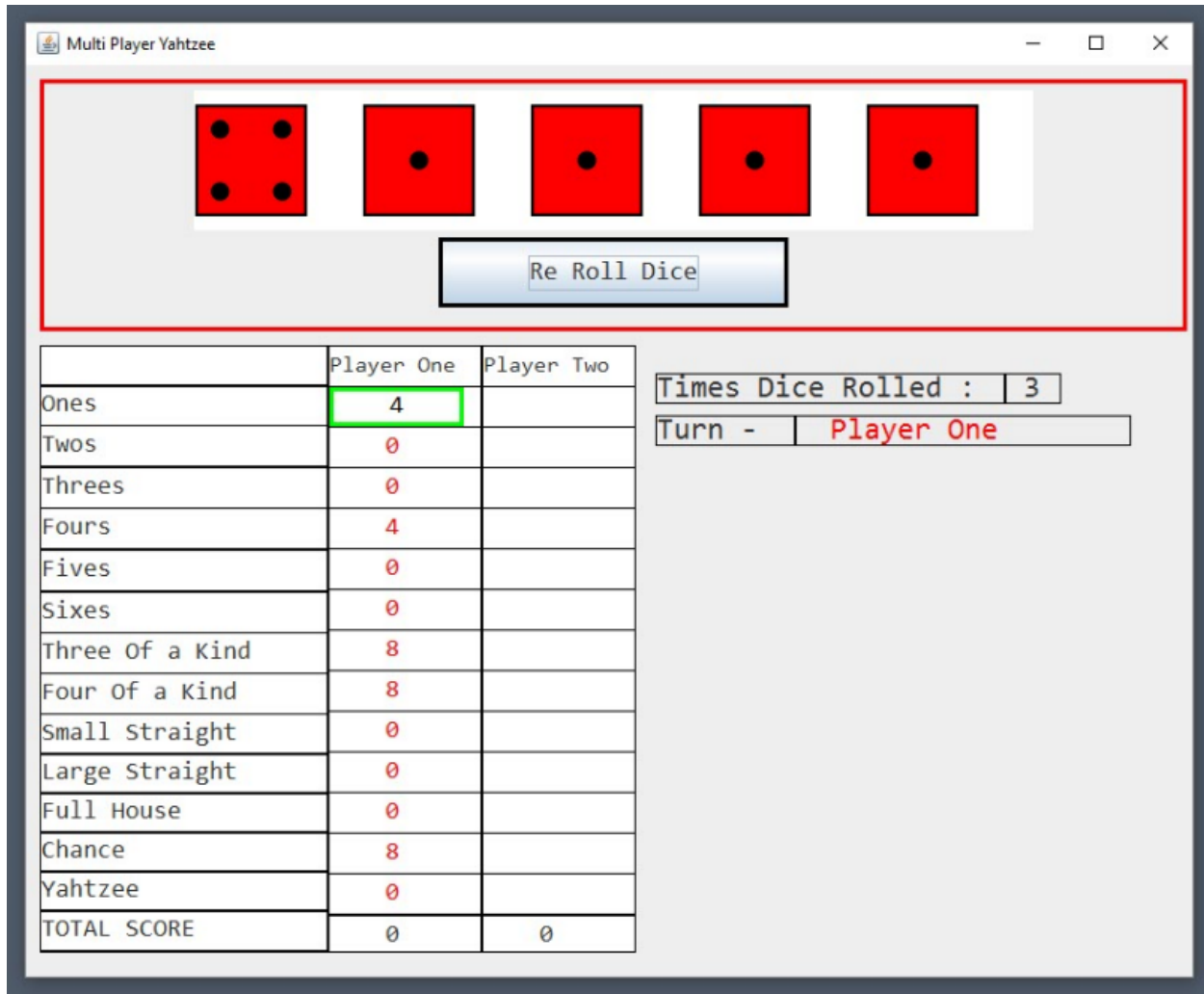
The points with green background show the categories that the user has already scored and the previously selected scoring categories cannot be further selected in the next round.

The points with red foreground color are the categories that the user can score for this round.

And When the user points the mouse cursor over the score with red foreground the label makes a green border to aware user that they are hovering over the score with their mouse cursor and can select it to confirm score.

Therefore the interface implements correct scoring routine .

## Point Calculation for Each Category



The screenshot shows a window titled "Multi Player Yahtzee". At the top, five red dice are displayed: the first shows four dots, and the other four show one dot. Below the dice is a button labeled "Re Roll Dice".

Below the dice area is a table for tracking scores for two players:

	Player One	Player Two
Ones	4	
Twos	0	
Threes	0	
Fours	4	
Fives	0	
Sixes	0	
Three Of a Kind	8	
Four Of a Kind	8	
Small Straight	0	
Large Straight	0	
Full House	0	
Chance	8	
Yahtzee	0	
TOTAL SCORE	0	0

To the right of the table, there are two input fields: "Times Dice Rolled :" with the value "3", and "Turn -" with the value "Player One".

Figure 16 Correct Point Calculation for Category: Aces

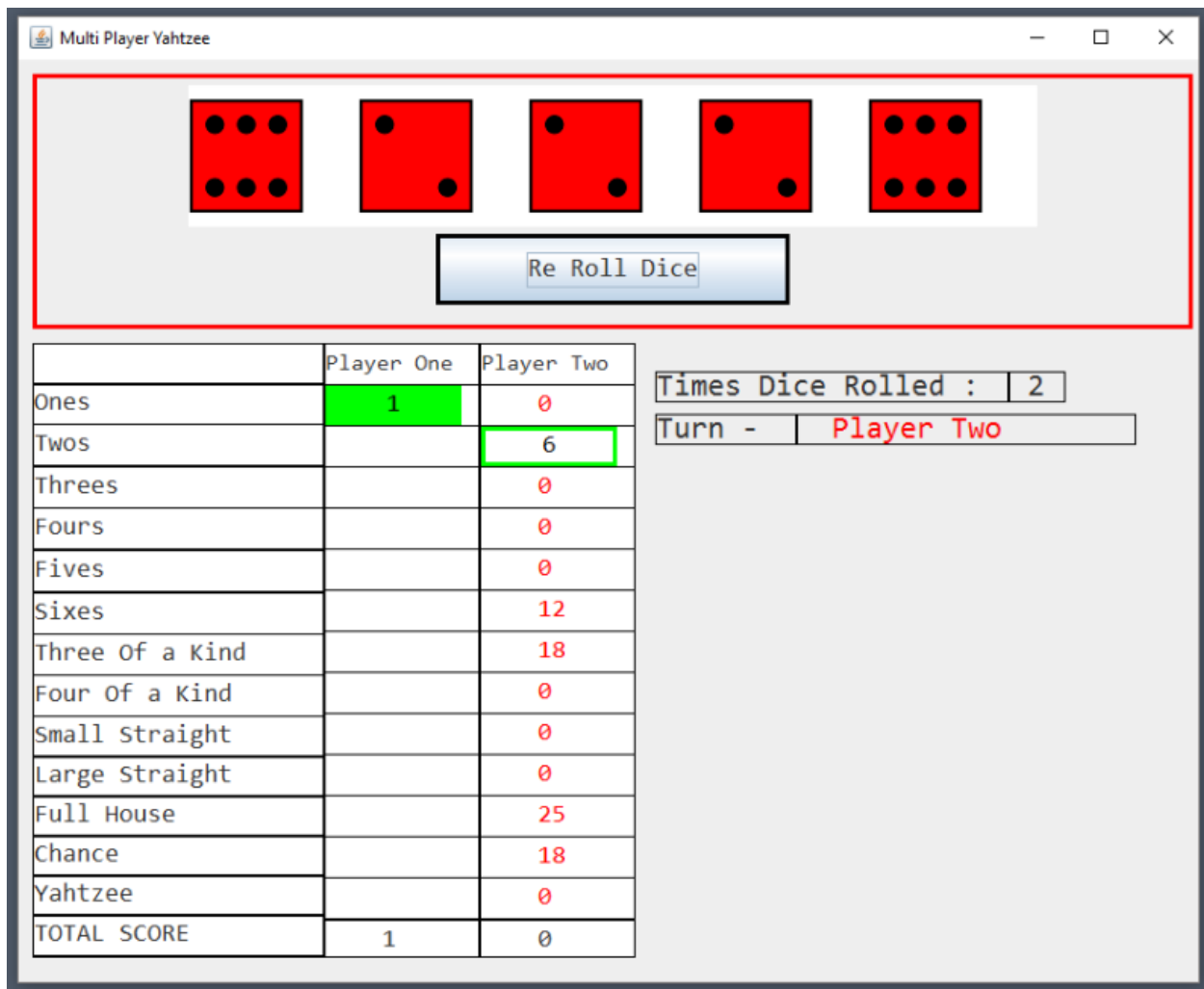


Figure 17 Correct Point Calculation for Category: Twos







### Figure 20 Correct Point Calculation for Category: Fives

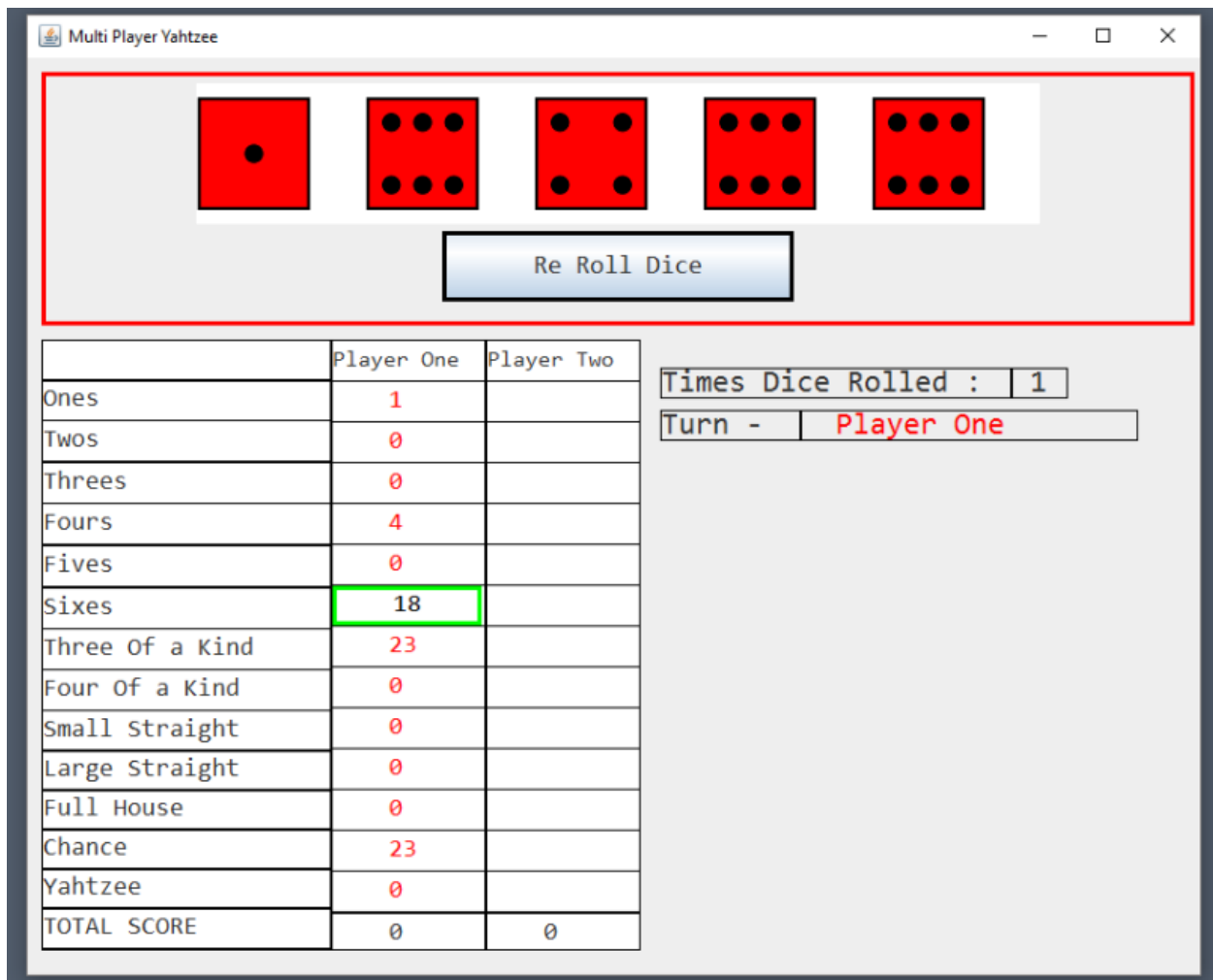


Figure 21 Correct Point Calculation for Category: Sixes



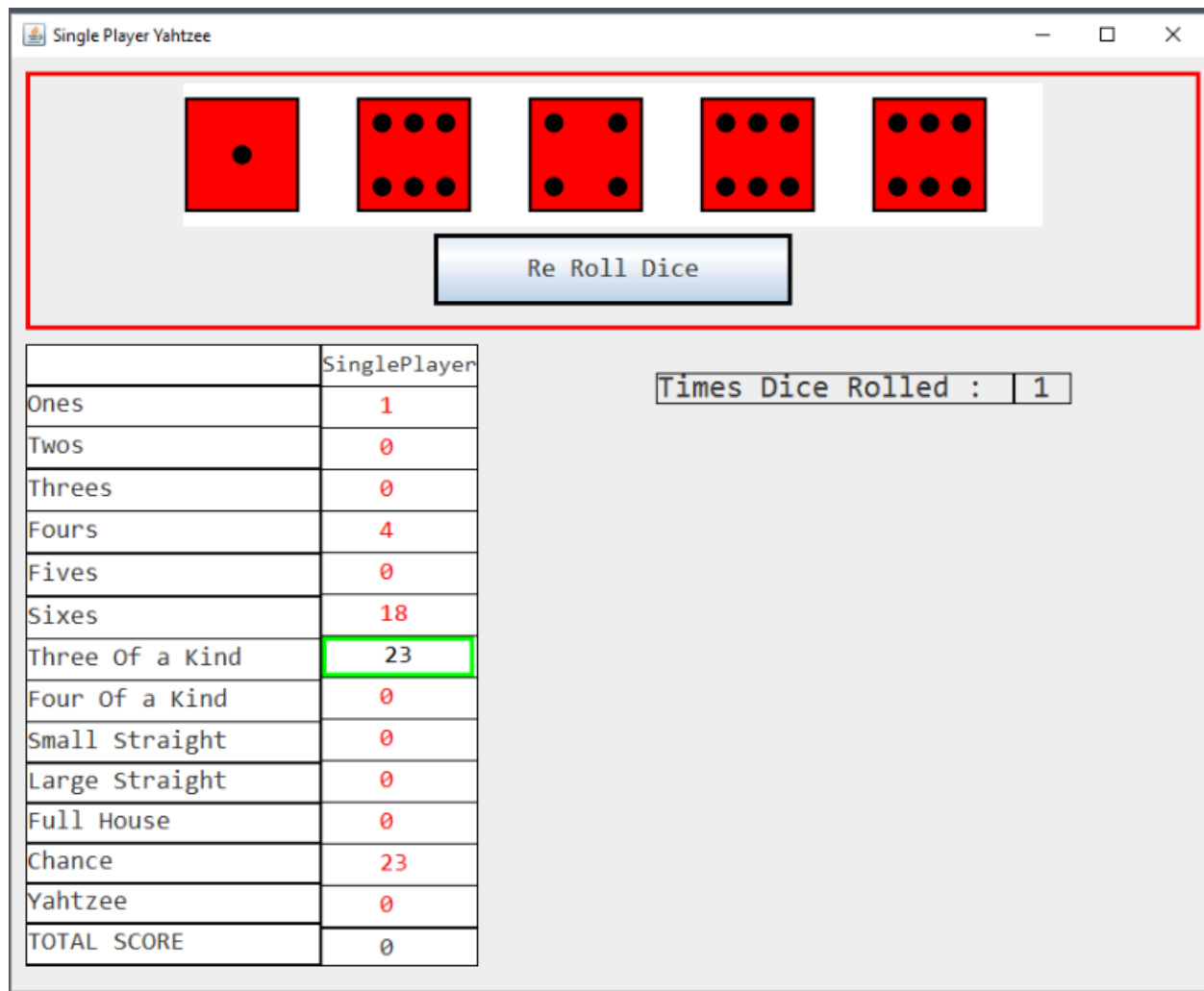



Figure 22 Correct Point Calculation for Category: Three of a Kind

Multi Player Yahtzee
—
□
×



Re Roll Dice

	Player One	Player Two
Ones	0	
Twos	2	
Threes	0	
Fours	0	
Fives	20	
Sixes	0	
Three Of a Kind	22	
Four Of a Kind	22	
Small Straight	0	
Large Straight	0	
Full House	0	
Chance	22	
Yahtzee	0	
TOTAL SCORE	0	0

Times Dice Rolled : 1

Turn - Player One

Figure 23 Correct Point Calculation for Category: Four of a Kind



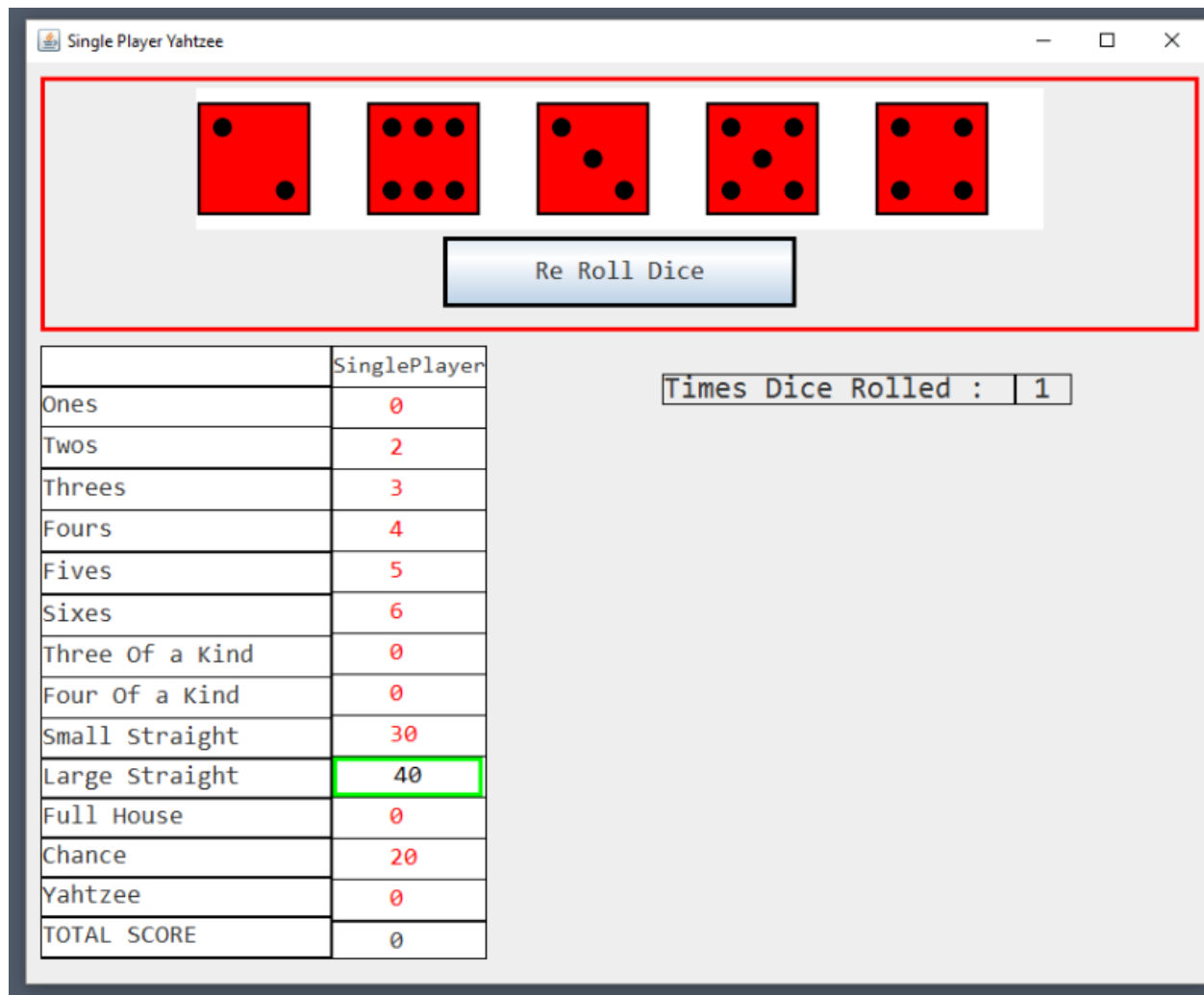


Figure 25 Correct Point Calculation for Category: Large Straight

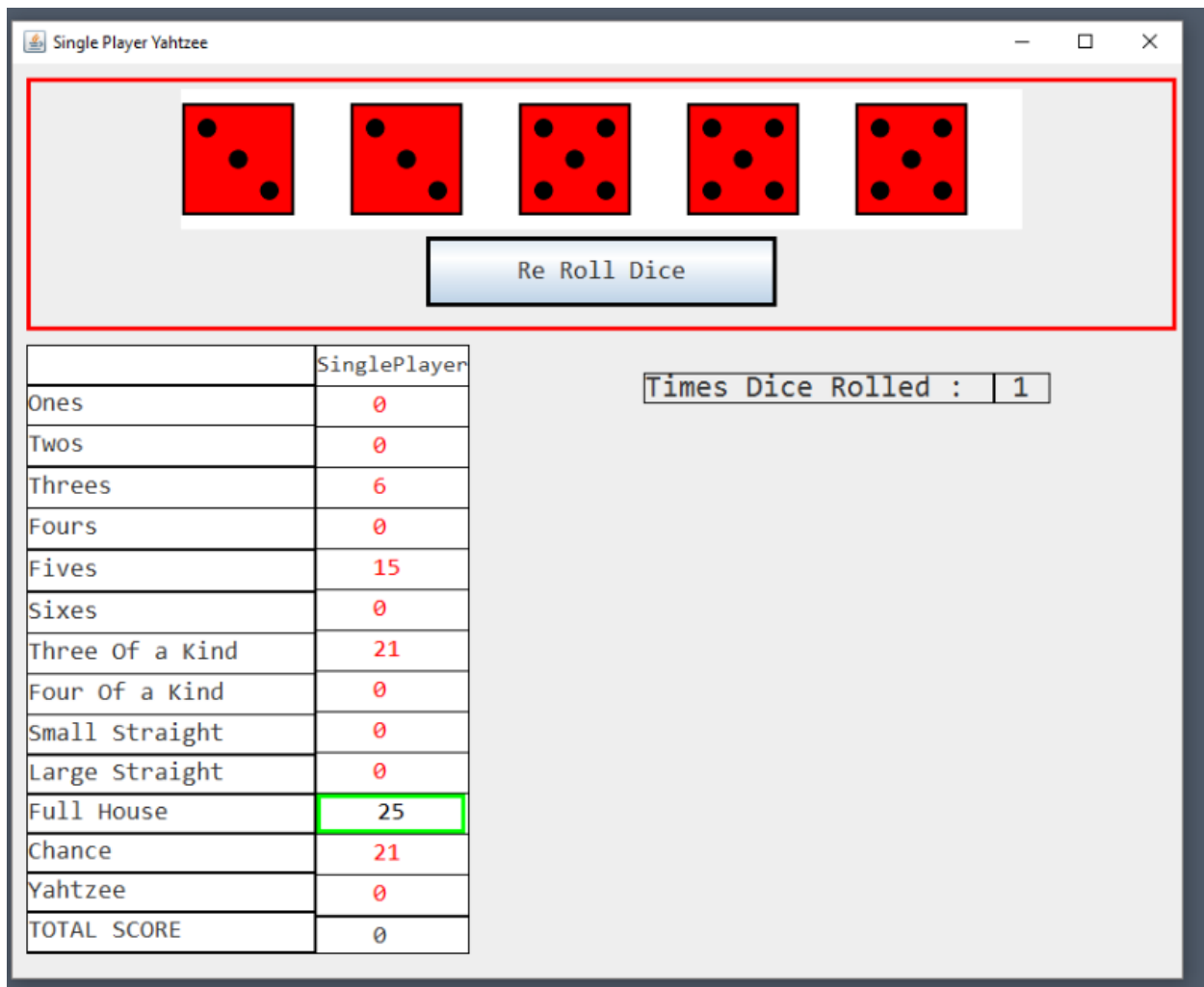


Figure 26 Correct Point Calculation for Category: Full House

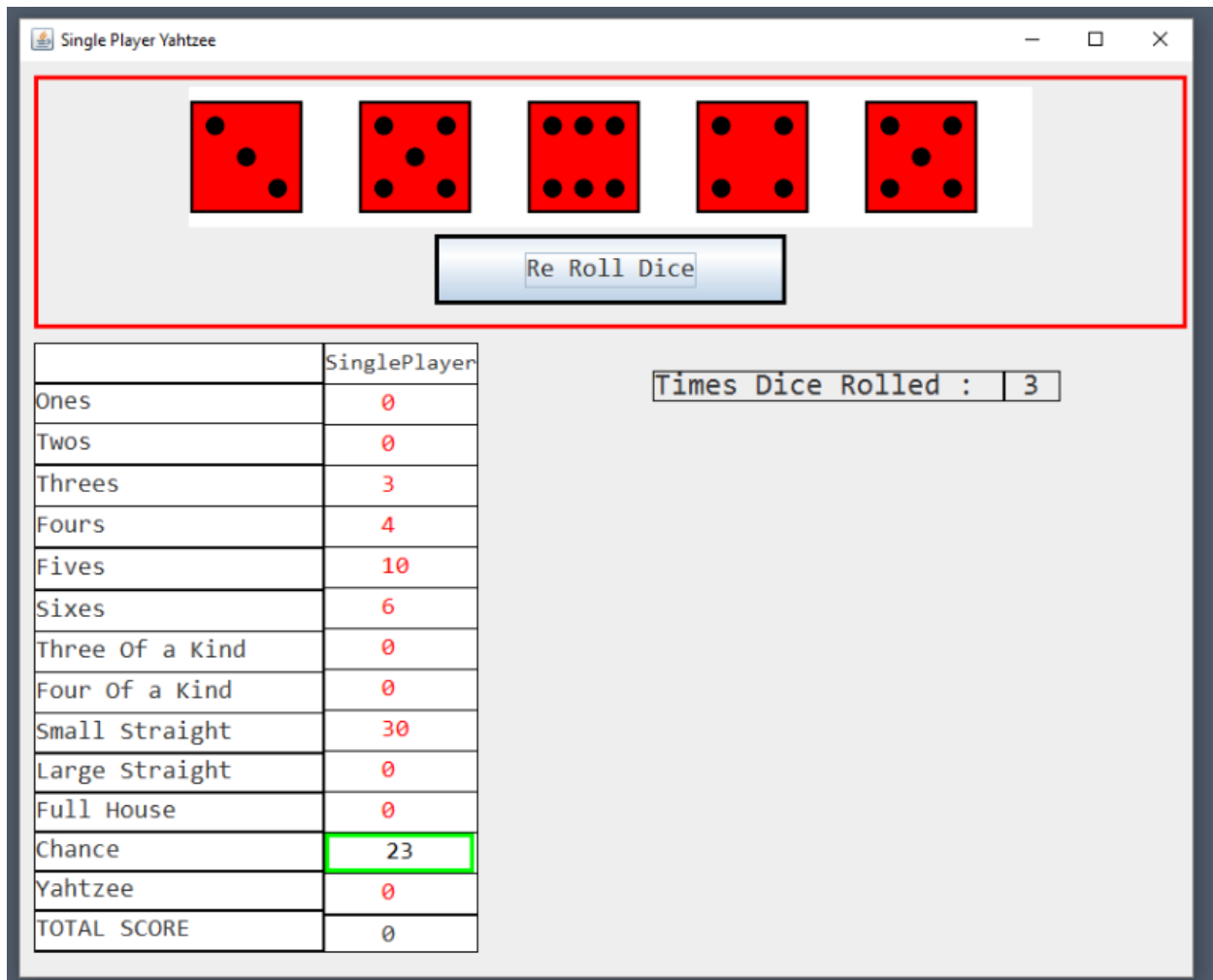


Figure 27 Correct Point Calculation for Category: Chance

### Figure 28 Correct Point Calculation for Category: Yahtzee

**Figure 28 Correct Point Calculation for Category: Yahtzee**

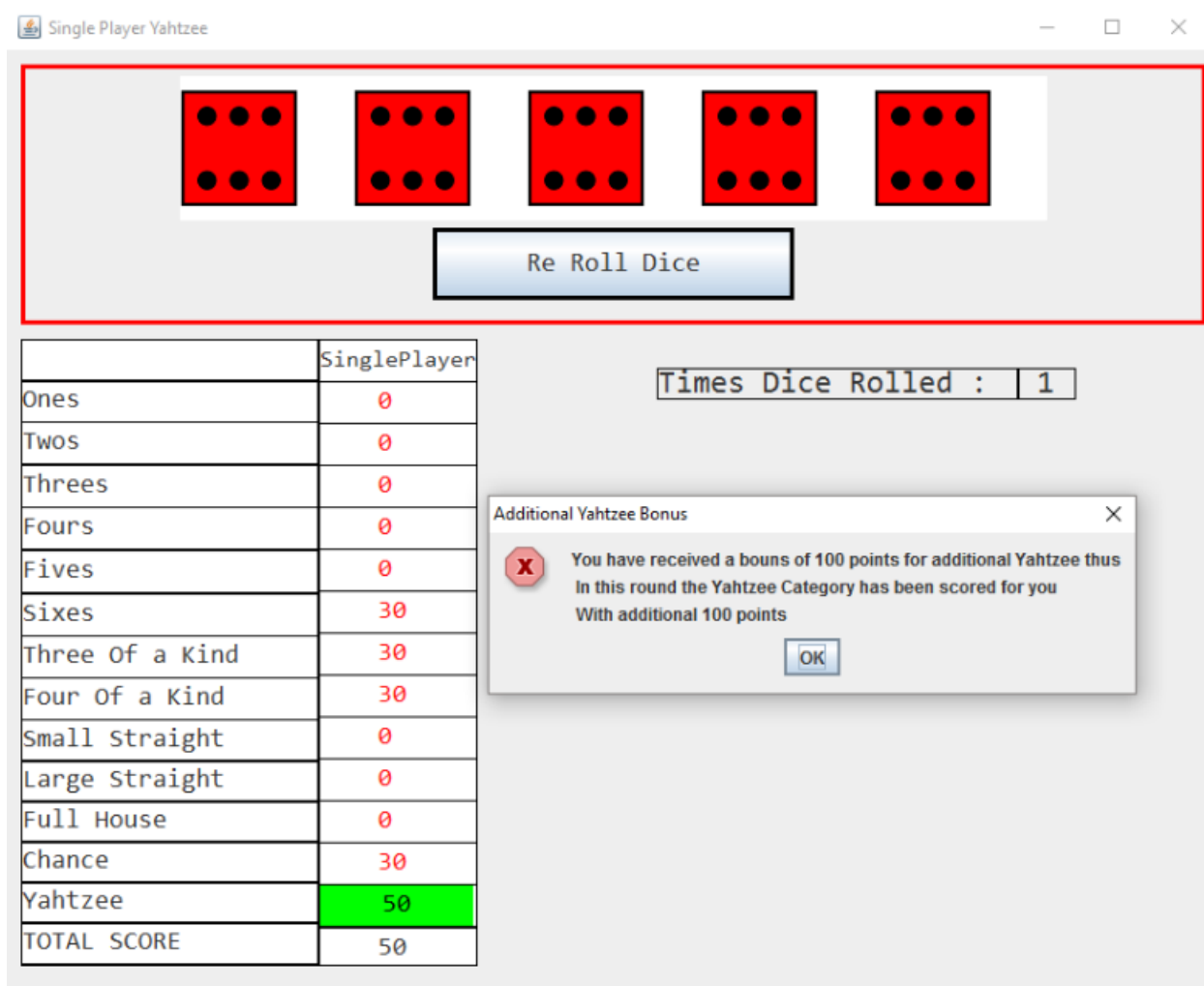


Figure 29 Bonus Yahtzee Alert



Single Player Yahtzee

Roll Dice

Times Dice Rolled : 0

	SinglePlayer
Ones	
Twos	
Threes	
Fours	
Fives	
Sixes	
Three Of a Kind	
Four Of a Kind	
Small Straight	
Large Straight	
Full House	
Chance	
Yahtzee	Bonus +100 150
TOTAL SCORE	200

Figure 30 Yahtzee Bonus Confirmed and Displayed after clicking okay

## Single Player Total Score Calculation and Ending

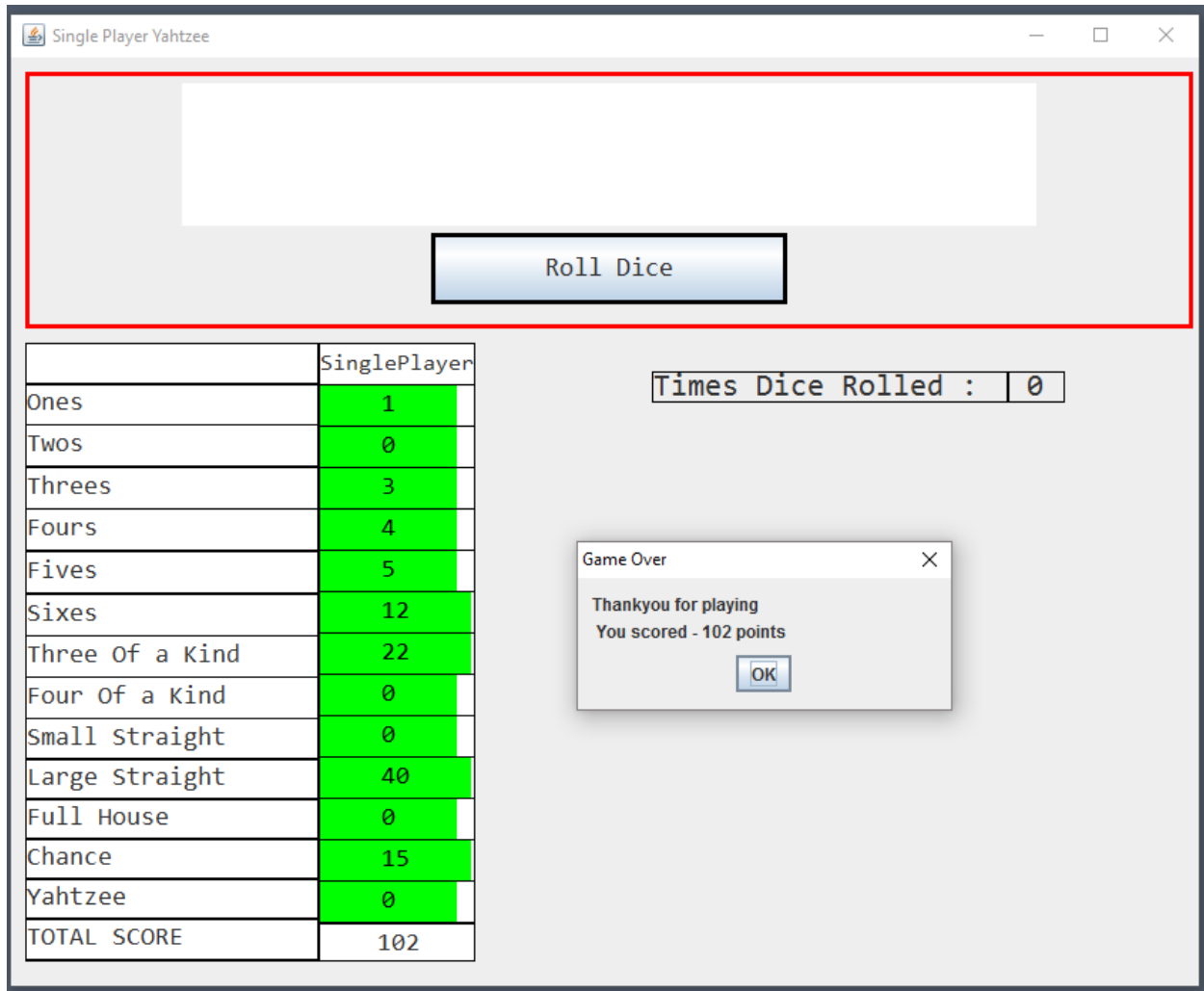


Figure 31 Calculation of total score is correct and game ending in Single Player

In Single Player after all the categories are scored the total score displayed is absolutely correct. At this point the interface notifies the player his/her score and the game exits after user clicks ok.

## MultiPlayer Gameplay, Total Score Calculation and Ending

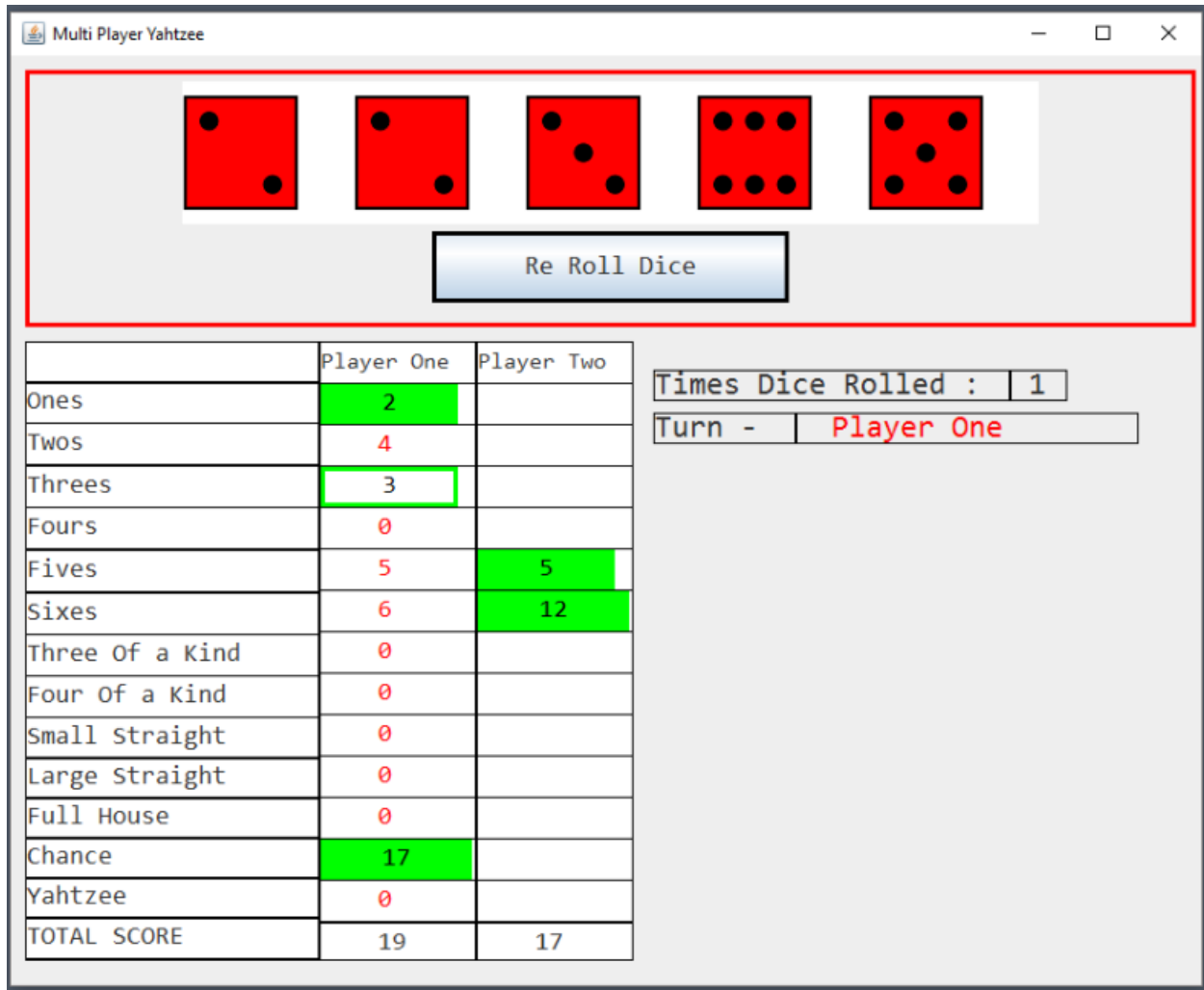


Figure 32 Multiplayer Flow

The interface shows that it's Player one's turn now. When player one selects and confirms score, the turn moves on to player two (further screenshot on next page).



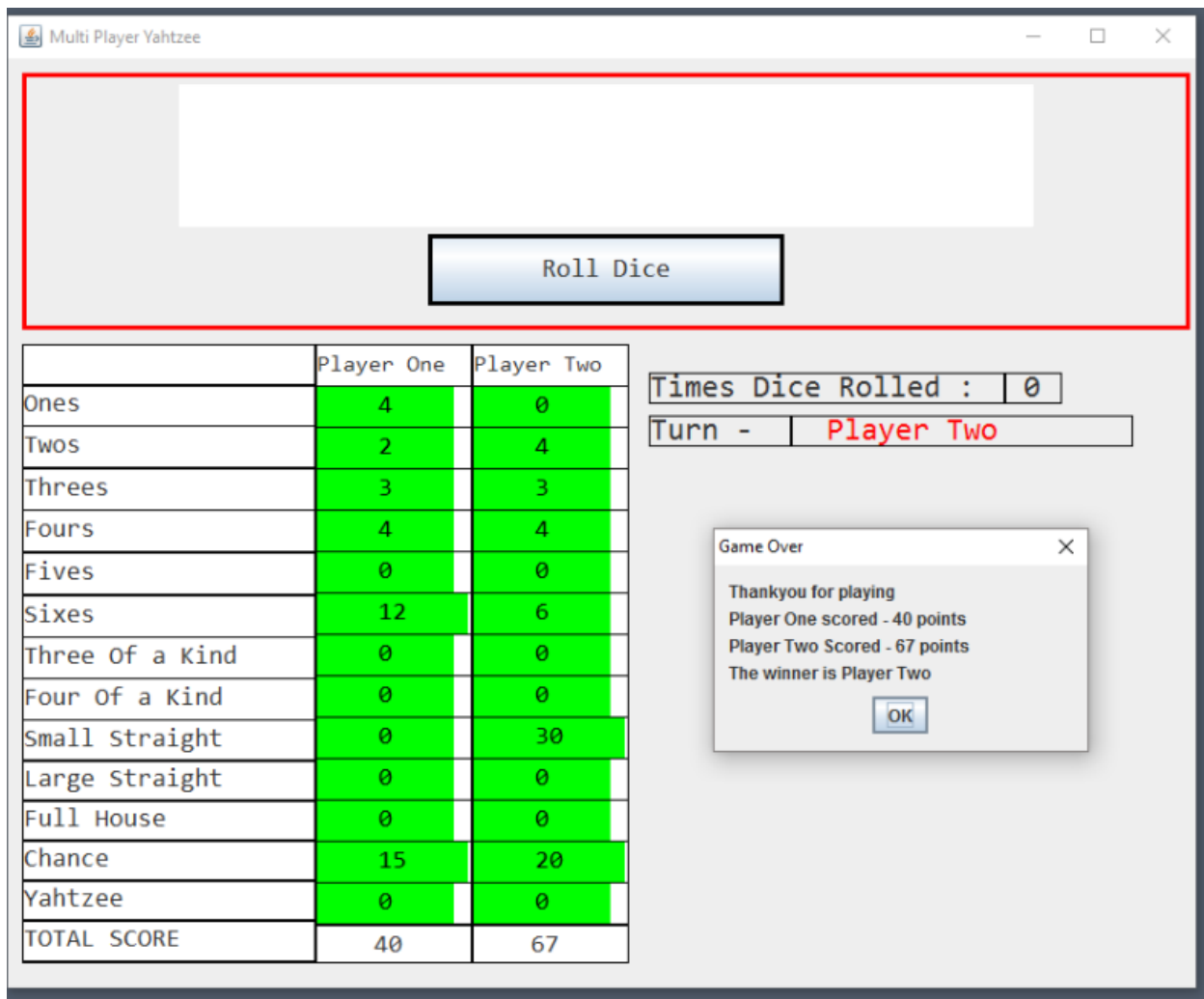


Figure 34 Multiplayer Gameplay Score Calculation and Ending

After the multiplayer gameplay ends the application shows correct total score and displays the final result i.e. winner. In this case the winner is Player Two.

The application can successfully handle both single player and multiplayer gameplay.

## Functionality that I failed to add to my application

Not all the functionalities and conditions mentioned in the assignment could be fulfilled. I failed to assign random special powers to the players of this game, thus the testing of special powers for my application failed to take place.

The screenshot shows a window titled "Multi Player Yahtzee". Inside the window, there is a large white rectangular area at the top, which is highlighted with a red border. Below this area is a blue button labeled "Roll Dice". Below the button is a table with two columns, "Player One" and "Player Two", and rows for various Yahtzee categories. To the right of the table, there are two text boxes: "Times Dice Rolled : 0" and "Turn - Player One".

	Player One	Player Two
Ones		
Twos		
Threes		
Fours		
Fives		
Sixes		
Three Of a Kind		
Four Of a Kind		
Small Straight		
Large Straight		
Full House		
Chance		
Yahtzee		
TOTAL SCORE	0	0

Times Dice Rolled : 0

Turn - Player One

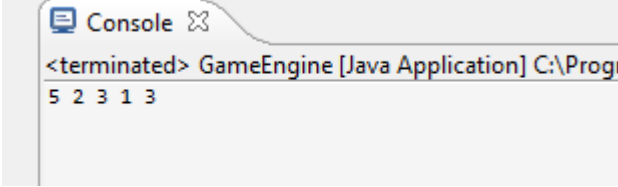
Figure 35 Random Special Powers for Players missing from the application

# White box Testing:

**Test Plan** –In this testing I will examine whether my code works as expected or not and show the outcomes where possible via console. Furthermore for the testing I will focus on the logic of the code and on the core logical operation of my application.

And thus I will test GameEngine Class which is the backbone of my application; it generates die faces and performs score calculation. Therefore White Box Testing will be related to GameEngine Class and therefore the test plan includes–

1. Test my Game Engine's functionality to generate six random numbers from one to six which become my die faces.
2. Test Game Engine's functionality of generating upper category scores.
3. Test Game Engine's functionality of generating three of a kind and four of a kind category score.
4. Test Game Engine's functionality of generating small straight and large straight category score.
5. Test Game Engine's functionality of generating full house category score.
6. Test Game Engine's functionality of generating full Yahtzee category score.
7. Test Game Engine's functionality of generating chance category score.

<u>White Box Testing</u> ⋮ <u>Test Plan</u> <u>Number-</u>	<u>White Box Testing</u> ⋮ <u>Test case</u>	<u>White Box Testing</u> ⋮ <u>Expected results</u>	<u>White Box Testing :</u>  <u>Code</u>	<u>White Box Testing</u> ⋮ <u>Actual results</u>
1	Game Engine : Generating Dice Numbers	Generate 6 random numbers	<pre>public static Random dice = new Random();  public static int[] diceResults = { dice.nextInt(6) + 1, dice.nextInt(6) + 1, dice.nextInt(6) + 1,     dice.nextInt(6) + 1, dice.nextInt(6) + 1 };  public static void willRollAllDices() {      for (int i = 0; i &lt; 5; i++) {          diceResults[i] = dice.nextInt(6) + 1;      }  }</pre> <p>Test</p> <pre>public static void main(String[] args) {      willRollAllDices();      for (int i : diceResults) {          System.out.print(i + " ");      }  }</pre> <p>Outcome</p> 	Result as Expected



2	Game Engine : Generating Upper Scores	Give correct upper score	<pre>public static int upperScores(int diceFace, int[] diceResults) {     int count = 0;     for (int i : diceResults) {         if (i == diceFace) {             count++;         }     }     return (diceFace * count); }</pre> <p>Test</p> <pre>public static void main(String[] args) {     willRollAllDices();     for (int i : diceResults) {         System.out.print(i + " ");     }     System.out.println("");     System.out.println(upperScores(1, diceResults));     System.out.println(upperScores(3, diceResults)); }</pre> <p>Outcome</p> 	Result as Expected
---	--	-----------------------------	---	-----------------------

3	Game Engine : Generating three of a kind and four of a kind score	Give correct three of a kind and four of a kind score	<pre>// Logic set for 3 &amp; 4 public static int ofAKind(int kindOf, int[] diceResults) {      int x = 0;     int count = 0;     int number = 0;      for (int i = 0; i &lt;= (diceResults.length - kindOf); i++) {          x = diceResults[i];         count = 0;          for (int k = i; k &lt;= (diceResults.length - 1); k++) {              // Increasing count if number matched             if (x == diceResults[k]) {                  count++;              }              // Checking if count is 3 or 4 of a kind             if (count == kindOf) {                  // Adding all up and exit                 for (int m : diceResults) {                      number = number + m;                  }                  return number;              }          } // second loop end      } // first loop end      return 0;  }</pre> <p>Test</p> <pre>public static void main(String[] args) {      for (int i : diceResults) {          System.out.print(i + " ");      }      System.out.println("");      System.out.println(ofAKind(3, diceResults));     System.out.println(ofAKind(4, diceResults));  }</pre> <p>Outcome</p>	Result as Expected
---	--	---	--	--------------------

			<div><div>Console</div><div>&lt;terminated&gt; GameEngine [Java Application] 4 4 4 4 3 19 19</div></div>	
4	Game Engine : Generating small straight and large straight score	Give correct small straight score and large straight score	<pre>public static int straightScore(int consecutiveDieFaces, int[] diceResults) {     // consecutiveDieFaces should either 4 or 5     StringBuilder text = new StringBuilder();     for (int i = 0; i &lt; diceResults.Length; i++) {         if (diceResults[i] == 1) {             text.append("1");         } else if (diceResults[i] == 2) {             text.append("2");         } else if (diceResults[i] == 3) {             text.append("3");         } else if (diceResults[i] == 4) {             text.append("4");         } else if (diceResults[i] == 5) {             text.append("5");         } else if (diceResults[i] == 6) {             text.append("6");         }     } }</pre>	Result as Expected

```
// Text.Added.to.String.Builder.Next.Check
///////////////////////////////////////////////////

if (consecutiveDieFaces == 4) {

    if (text.toString().contains("1" && text.toString().contains("2" && text.toString().contains("3"
        && text.toString().contains("4")) {

        return 30;

    } else if (text.toString().contains("2" && text.toString().contains("3" && text.toString().contains("4"
        && text.toString().contains("5")) {

        return 30;

    } else if (text.toString().contains("3" && text.toString().contains("4" && text.toString().contains("5"
        && text.toString().contains("6")) {

        return 30;
    } else
        return 0;

} // consecutiveDieFaces = 4-check closing Bracket {

else if (consecutiveDieFaces == 5) {

    if (text.toString().contains("1" && text.toString().contains("2" && text.toString().contains("3"
        && text.toString().contains("4" && text.toString().contains("5")) {

        return 40;

    } else if (text.toString().contains("2" && text.toString().contains("3" && text.toString().contains("4"
        && text.toString().contains("5" && text.toString().contains("6")) {

        return 40;

    } else
        return 0;

} // consecutiveDieFaces = 5-check closing Bracket }

else {

    System.out.println("You are using Straight method in incorrect way");
    return 0;

}

}

}
```

Test

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

    for (int i : diceResults) {

        System.out.print(i + " ");

    }

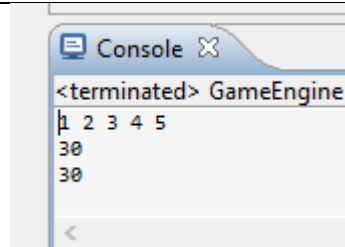
    System.out.println("");

    System.out.println(straightScore(4, diceResults));
    System.out.println(straightScore(4, diceResults));

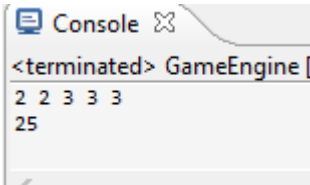
}

}
```

Outcome



5	Game Engine : Generating full house score	Give Correct full house score	<pre>public static int fullHouse(int[] diceResults) {     if (ofAKind(3, diceResults) == 0) { // means if three of a kind does not exist then return 0         return 0;     } else { // means three of a kind exists. Now-Check if TwoOfAKind Exists         int searchForThreeOfAKindNumber = 0; // First Have to searchForThreeOfAKindNumber and then exclude this to         // search for two of a kind         for (int i = 0; i &lt;= 2; i++) { // 0,1,2             int x = diceResults[i];             int count = 0;             for (int k = i; k &lt; diceResults.Length; k++) {                 if (x == diceResults[k]) {                     count++;                 }             } // second for loop end             if (count == 3) {                 searchForThreeOfAKindNumber = x;                 break;             }         } // first for loop end         //////////////////////////////////////         // Now the search for two of a kind will begin and will exclude         ////////////////////////////////////// searchForThreeOfAKindNumber         for (int i = 0; i &lt;= 3; i++) { // 0,1,2             int x = diceResults[i];             int count = 0;             for (int k = i; k &lt; diceResults.Length; k++) {                 if ((x == diceResults[k]) &amp;&amp; (x != searchForThreeOfAKindNumber)) {                     count++;                 }             } // second for loop end             if (count == 2) {                 return 25;             }         } // first for loop end     }     return 0; }</pre>	Result as Expected
Test				

			<pre>public static void main(String[] args) {     for (int i : diceResults) {         System.out.print(i + " ");     }     System.out.println("");     System.out.println(fullHouse(diceResults)); }</pre> <p>Outcome</p> 	
6	Game Engine : Generating Yahtzee score	Give correct Yahtzee score	<pre>public static int yahtzee(int[] diceResults) {     int yahtzee = diceResults[0];     int count = 0;     for (int i = 0; i &lt; diceResults.length; i++) {         if (yahtzee == diceResults[i]) {             count++;         }     }     if (count == 5) {         return 50;     } else {         return 0;     } }</pre> <p>Test</p> <pre>public static void main(String[] args) {     for (int i : diceResults) {         System.out.print(i + " ");     }     System.out.println("");     System.out.println(yahtzee(diceResults)); }</pre> <p>Outcome</p>	Result as Expected

			<div><div>Console</div><div>&lt;terminated&gt; GameEngine [Java Application] 2 2 2 2 2 50</div></div>	
7	Game Engine : Generating chance score	Give correct chance score	<div><div><pre>public static int chance(int[] diceResults) {     int sum = 0;     for (int i = 0; i &lt; diceResults.length; i++) {         sum = sum + diceResults[i];     }     return sum; }</pre></div><div>Test</div><div><pre>public static void main(String[] args) {     for (int i : diceResults) {         System.out.print(i + " ");     }     System.out.println("");     System.out.println(chance(diceResults)); }</pre></div><div>Outcome</div><div><div>Console</div><div>&lt;terminated&gt; GameEngine [Java Application] 2 2 2 2 2 10</div></div></div>	Result as Expected



## Exception Handling:

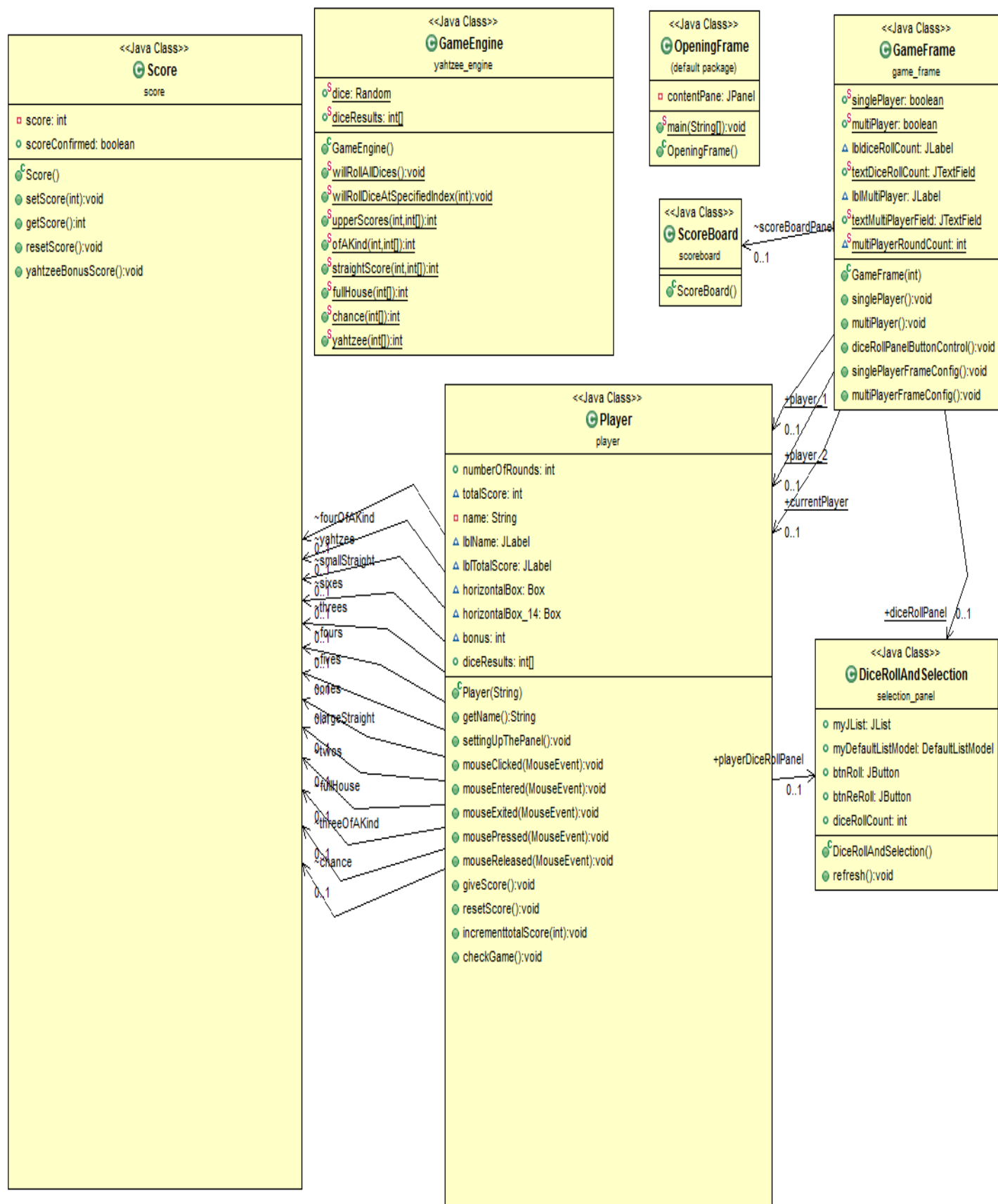
```
public static void main(String[] args) {  
    EventQueue.invokeLater(new Runnable() {  
        public void run() {  
            try {  
                OpeningFrame frame = new OpeningFrame();  
                frame.setVisible(true);  
            } catch (Exception e) {  
                e.printStackTrace();  
            }  
        }  
    });  
}
```

Figure 36 Effective Exception Handling

Exception were handed appropriately

# TASK 3

A detailed class diagram of my application has been presented in this task. I have outlined the relationship between my classes and as well have included the attributes and method of each class. The fully detailed UML diagram is on the next page -



# CONCLUSION

The Yahtzee game was successfully designed and developed using object oriented programming (OOP). The program algorithm made effective use of various tools. As per the assignment requirements the application was tested thoroughly. Test plan was designed and successfully implemented. A detailed UML Diagram was also presented.