Technical University of Denmark

02314, 62531 & 62532

INDLEDENDE PROGRAMMERING, UDVIKLINGSMETODER TIL IT-SYSTEMER OG VERSIONSSTYRING OG TESTMETODER

CDIO 1

GROUP 30

Alara-Melis Karatas - s235104

Turan Talayhan - s224746

Melih Kelkitli - s235114

Nikoonam Nabavi-Monfared - s235453

Elisa Esgici - s236164

Asma Maryam - s230716

September 29, 2023

1. Summary

To begin with, we created a Use Case to provide a visual representation of our Dice game and a brief Use Case description. Following that, we constructed a domain model for the game, which essentially outlines how the game commences. It involves two players competing to achieve the highest score, and the player with the lowest score loses. Subsequently, the game restarts, allowing the players to play again. We've also implemented the code for the Dice game and explained the testing process.

2. Hourly accounting

In the process of developing a software system that meets the specification of the requirements in the assignment, we have spent roughly 63 hours on the project in total. Each member of the group has contributed to the project, but in different ways. The hours that are used on it and listed down below, is an estimate on the total time spent on analyzing the project, making artifacts such as use case diagrams, domain models and descriptions, and the code itself.

Name of group member	Hours spent on project
Nikoonam	12 hours
Asma	15 hours
Melih	8 hours
Turan	8 hours
Elisa	10 hours
Alara	10 hours

3 Table of Contents

4. Introduction

In this CDIO 1 task, we have made a Dice game, that consists of 2 players in total. Each player must throw 2 dices, where the point is summed up each time by the program and is saved for each player every time the dices are thrown by the players. The player that reaches 40 points first wins the game.

5. Project planning

Our tasks include developing the core game mechanics, such as dice rolling, point calculation, win and loss conditions, and documenting the testing process and results. We will ensure that the game is user-friendly and does not necessitate an instruction manual.

6. Analysis

Our project, "Dice Game," is aimed at developing a two-player dice game for installation on Windows machines in DTU Learn's computer labs. The objective of this game is to reach 40 points by rolling 2 dice with a dice cup. The game includes additional features such as losing points for rolling two ones, earning an extra turn for rolling two identical dice, and winning the game by rolling two sixes, provided the player also rolled two sixes in the previous turn.

6.1 Requirements Specifications

The client requests that the Dice Game should have the following requirement specifications:

- 2 players
- 2 dices in a raffle cup
- The game should be able to work on Windows devices in the database at DTU
- It should not take more than 333 milliseconds for the results to come.

6.1.1 Noun analysis

- Player
- 2 dices
- One turn
- Last turns

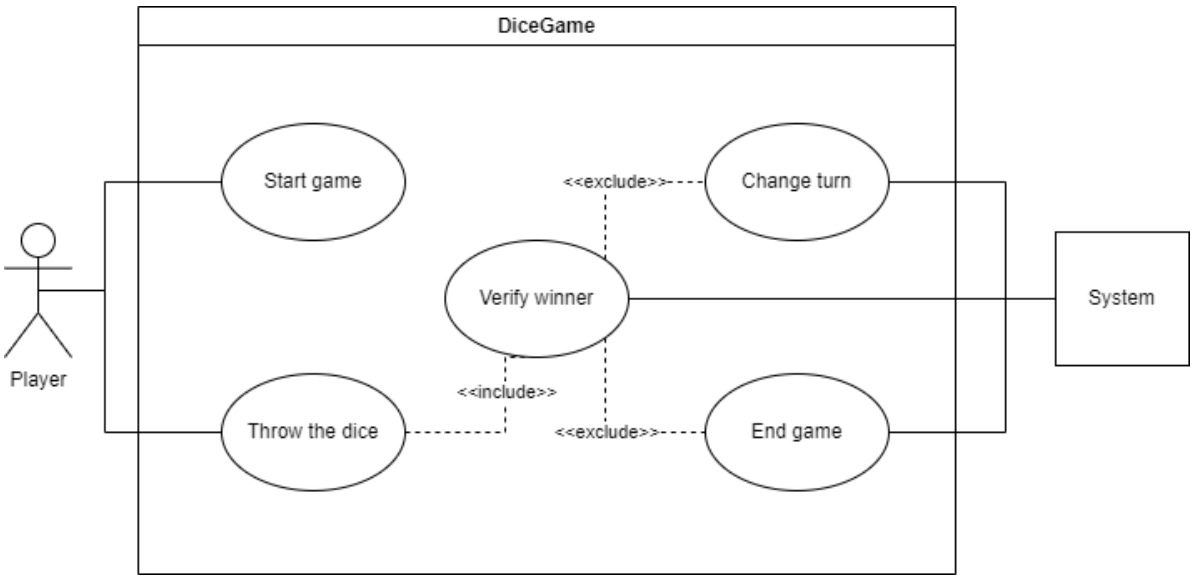
- One game
- One raffle cup
- One website (DTU Learn)

6.1.2 Verb analysis

- Start the game
- Roll the dice
- Lose the game
- Win the game
- Switch turners with players
- Update the game (when winner loses or wins)
- End the game (if winner or loses wins)
- Extra turns (if person get two dices landed on 6)

6.4 Use case

Below is the Use case of the game. We have a player (primary actor) on the left side and a system (Secondary actor) on the right side. When our game starts then one player will throw the dice and the system will always check the winner after every throw. Here we include (verify winner) and then there are two options, change turn or End game. We are using exclude because there should be only one possibility between change turn or end game. [1]



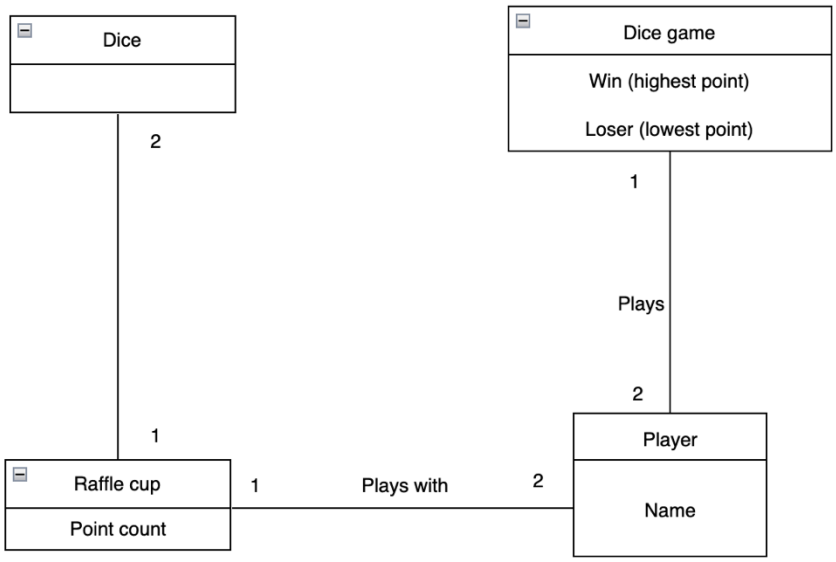
- [1] Larman, Craig (2004). Chapter 6

6.5 Use case description

Use case: Dice Game
Brief description: Prior to commencing the game, it is crucial for the two players to designate who will initiate the gameplay. Once this decision has been made, one of the two players takes the lead and initiates the game by rolling the dice, accomplished by pressing the <ENTER> key on the keyboard. Subsequently, the terminal will exhibit the total sum of the two dice and update the player's score. If a player accumulates a total of 40 points, the terminal will declare them as the winner, concluding the game. If this threshold has not been met, the players will switch turns, and the succeeding player will take their turn to roll the dice. This cycle continues in the same sequence as the preceding player until one of the participants attains 40 points.
Primary actors: Player 1 - 2
Secondary actors: The system
Preconditions: 1. System boots up the system. 2. There can only be two players. 3. When a player has reached 40 points, the player has won and the game ends.
Main flow: 1. Player 1 boots up the game and presses <ENTER> on the terminal. 2. The system rolls the dice and displays the sum of the two dice, which is automatically added to the players points. 3. System checks for the winner. 4. System turns to player 2, if player 1 has not won. 5. System detects if player one or two is the winner. 6. System ends the game when it displays the winner. 7. System detects the winner and informs the player about the winner.
Postconditions: The players have the ability to begin a new game, when they are finished with their current game.

6.6 Domain Model

Below is the domain model of the game, explaining how we begin our game called “Dice”, where we use the dice cup and count the points. We have 2 players who compete to achieve the highest points, and if not, the one with the lowest points loses. The game will restart, allowing the players to play again.[1]



7. Code Design

```
1. import java.util.Random;
```

//Here we are defining the class to a public class and its name to “Die”. At the beginning of the class, we create a private object called “number” that is the points of the dice, beginning from 0 to 6 (marked at the end, NextInt(7)).

```
2.
3. public class Die {
4.     private int number = 0;
5.
```

//Here we make a method of the “Number” called “GetNumber” which basically returns the value of the number with the code “return number”.

```
6.     public int getNumber() {
7.         return number;
8.     }
9.
10.     public void setNumber(int number) {
11.         this.number = number;
12.     }
13.
```

//Here we’re defining the variable random to the type Random which means it will give out a random value.

```
14.     Random random = new Random();
```

//A Public method is created and called “rollDie()” so whenever we run this code, variable “number” will carry a random number of type Intergers, because we use the random.nextInt() function and we set it to 7, because we don’t want the numbers of Intergers to exceed over 6 points. Remember an array counts from 0 to 6 so 0 is actually 1 and to get to 6 we can write 6+1.

```
15.
16.     public void rollDie() {
17.         number = random.nextInt(6) + 1;
18.     }
19. }
20.
```

```
1. import java.util.Scanner;
2.
3. public class RaffleCup {
4.     public static void main(String[] args) {
5.         Scanner scanner = new Scanner(System.in);
6.         Die die1 = new Die();
7.         Die die2 = new Die();
```

// In the “Die” class that we defined before, we created some objects that we are accessing now on line 6 and 7, and we have also created a scanner object on line 1.

```
8.
9.         int player1 = 0;
10.         int player2 = 0;
```

// Whenever the while loop is correct (whenever each player has below 40 points), it will allow the players to play the game. After pressing enter, it will roll both dice and tell the sum of the points of each dice for player 1. After player 1 has pressed enter, a similar code is made for player 2, where he/she presses enter and gets the value of each dice and the summed-up points.

```
11.
12.         while (player1 < 40 && player2 < 40) {
13.             System.out.println("Player 1:");
14.             System.out.println("Press enter to roll dice");
15.
16.             scanner.nextLine();
17.
18.             die1.rollDie();
19.             die2.rollDie();
20.
21.             System.out.println("Sum of rolled dice: " + (die1.getNumber() + die2.getNumber()));
22.
```

```
23.             player1 += die1.getNumber() + die2.getNumber();
24.             System.out.println("Points: " + player1);
25.
26.             if (player1 < 40 && player2 < 40) {
27.                 System.out.println();
28.                 System.out.println("------------------------");
29.                 System.out.println();
30.                 System.out.println("Player 2:");
31.                 System.out.println("Press enter to roll dice");
32.
33.                 scanner.nextLine();
34.
35.                 die1.rollDie();
36.                 die2.rollDie();
37.
38.                 System.out.println("Sum of rolled dice: " + (die1.getNumber() + die2.getNumber()));
39.                 player2 += die1.getNumber() + die2.getNumber();
40.                 System.out.println("Points: " + player2);
41.
42.                 System.out.println();
43.                 System.out.println("------------------------");
44.                 System.out.println();
45.             }
46.         }
47.
48.         if (player1 > player2) {
49.             System.out.println("Player 1 wins!");
50.         } else {
51.             System.out.println("Player 2 wins!");
52.         }
53.         scanner.close();
```

// line 48 to 52, our program checks which player has 40 points first, then it will be the winner.

```
54.     }
55. }
56.
```

7.1 Explanation

When we specify "Public class Die" we name a public class called Die, where the purpose of the class initially is to define a variable with the name "number" which therefore stores the results of the new rolled die.

In the getNumber() class, it returns the numbers that result from the roll of the dice.
When we write Random random, we define a variable "random" to the type Random that generates a random value. In the last piece, with the rollDie class, the variable "number" is defined to the values from the randomly thrown dice which are interposed in an array, where the maximum will bother interpersed from 1 to 6 points. We write 7, as arrays count 0 as the starting value, i.e. 1, therefore we write 7 and not 6.

8. Testing

It is essential in software development that the projects are evaluated and tested to ensure that the software is not broken or has significant flaws. In this project, we have been aware of bugs and potential mistakes from the very start, as when we were requested to communicate with the customer and project leader to narrow down the requirements:

"*A bug found at design time costs ten times less to fix than one in coding and a hundred times less than one found after launch*". Barry Boehm.[2]

By starting as early as possible, to eliminate bugs and defects, we have reduced a lot of time and effort being used on fixing bugs in the late development.

8.1 Probability

But a very relevant element in The DiceGame,is that we have two 6-sided dice that ultimately can determine which player will be the winner. As for this reason, the two dice must be fair or else the game is unplayable and breaks the rules. Therefore, we have also tested the two dice for bias and the probability of getting any of the possible outcomes for them to confirm that the two dice are unbiased.

In a standard singular die with 6 sides (1, 2, 3, 4, 5 and 6), the probability of a random number is equally likely. Each outcome is independent and that means they can occur once. To visually show the possible outcome of a die, we can create a sample space, which is a set of all the outcomes.

Number	1	2	3	4	5	6
Probability	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$

In the above sample set, probability was calculated by using this formula: $\frac{number\ occurence\ (1)}{sample\ size\ (6)}$.

In a situation of two dice, there are almost double as many possible results (2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12), but there are also 36 possible combinations that we must consider. This is simply calculated by multiplying $6 \cdot 6$ equals to 36. A sample set for two dice looks like this:

[2] Software Testing Primer v2, page 9

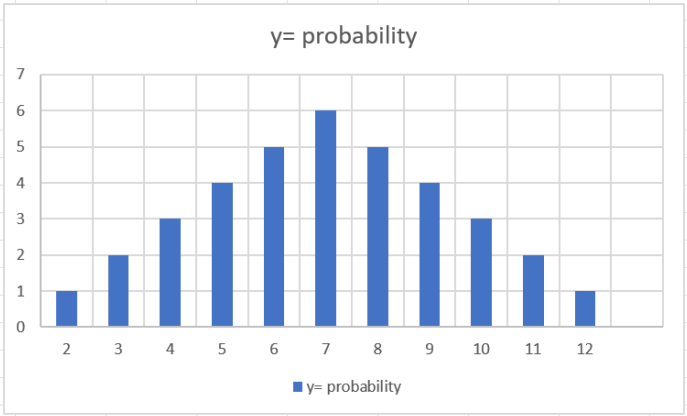
	1	**2**	**3**	**4**	**5**	**6**
1	(1, 1) = 2	(1, 2) = 3	(1, 3) = 4	(1, 4) = 5	(1, 5) = 6	(1, 6) = 7
2	(2, 1) = 3	(2, 2) = 4	(2, 3) = 5	(2, 4) = 6	(2, 5) = 7	(2, 6) = 8
3	(3, 1) = 4	(3, 2) = 5	(3, 3) = 6	(3, 4) = 7	(3, 5) = 8	(3, 6) = 9
4	(4, 1) = 5	(4,2) = 6	(4, 3) = 7	(4, 4) = 8	(4, 5) = 9	(4, 6) = 10
5	(5, 1) = 6	(5, 2) = 7	(5, 3) = 8	(5, 4) = 9	(5, 5) = 10	(5, 6) = 11
6	(6, 1) = 7	(6, 2) = 7	(6, 3) = 9	(6, 4) = 10	(6, 5) = 11	(6, 6) = 12

The number of possible outcomes here are 36, but the theoretical probability for each of the outcomes are different:

Outcome	2	3	4	5	6	7	8	9	10	11	12
Number of combinations	1	2	3	4	5	6	5	4	3	2	1
Probability	$\frac{1}{36}$	$\frac{2}{36}$	$\frac{3}{36}$	$\frac{4}{36}$	$\frac{5}{36}$	$\frac{6}{36}$	$\frac{5}{36}$	$\frac{4}{36}$	$\frac{3}{36}$	$\frac{2}{36}$	$\frac{1}{36}$

In the columns above, we have listed the probabilities of each possible sum and the number of combinations. This is a useful tool to compare and verify whether our two dice are fair or not, and see how closely it resembles to the graph below:

x= sum	y= probability
2	1
3	2
4	3
5	4
6	5
7	6
8	5
9	4
10	3
11	2
12	1



8.2 Our testing code and comparison

This is our Java testing code, and the code is a test class meant to simulate rolling two dice, 1000 times each, and counting how many times each possible sum (from 2 to 12) occurs.

```
1. public class Test {
2.     public static void main(String[] args) {
3.         Die die1 = new Die();
4.         Die die2 = new Die();
```

```
//Two objects die 1 and die2, of class die are created. These represent the two dice being rolled.
```

```
5.         int[] frequencyOfSums = new int[13];
```

```
//We are using frequencyOfSum array, with size 13. The array starts from zero that's why it will show all possible sums between 0-12. We know zero and one sum is not possible so array will not use 0 and 1. Because two dices sum starts from 2.
```

```
6.         for (int i = 1; i <= 1000; i++){
```

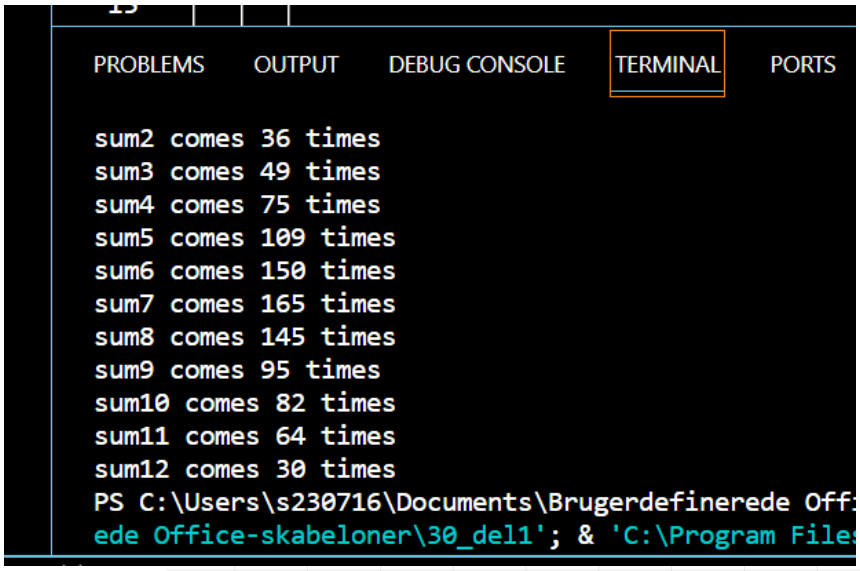
```
//We are using for loop, so it runs 1000 times, and simulating 1000 rolls of each die.
```

```
7.         die1.rollDie();
8.         die2.rollDie();
9.         var sum = die1.getNumber() + die2.getNumber();
10.
11.         frequencyOfSums[sum]++;
12.         }
13.         for (int i = 2; i <= 12; i++){
14.          System.out.println("sum" + i + " comes " + frequencyOfSums[i] + " times ");
```

```
//We have another for loop here, it will count the sum and then print how many times each sum has occurred.
```

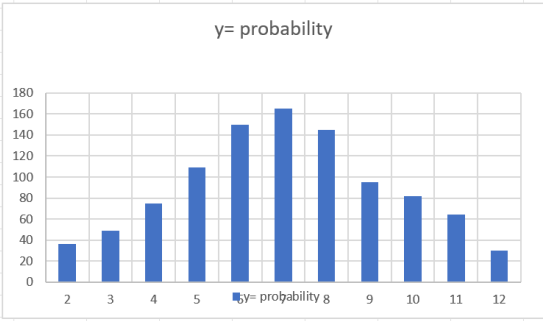
```
15.
16.     }
17.     }
18.
```

We ran the test code once and got the following results or output



And a graph of our testing code values:

x= sum	y= probability
2	36
3	49
4	75
5	109
6	150
7	165
8	145
9	95
10	82
11	64
12	30



To compare it with the theoretical probability, we can see it looks much alike and therefore our two dice must be fair and independent.

9. Conclusion

In conclusion, our journey in developing the “Dice Game” has been a comprehensive process that involved meticulous planning, analysis, and implementation. We began with a clear vision outlined in our Use Case and Domain Model, providing a visual representation and description of the game’s mechanics. Throughout the development, our primary goal was to create a user-friendly experience that required minimal instructions. The development process included coding the game and rigorously testing it to ensure its functionality and reliability. In addition, we made a graph that

shows the probability of the dice rolling a scattered number from 1-6. The results from the graph were close to the calculated graph values which can conclude the calculation of the dice possibility. Our dice game on java is constructed on java version 17, but the version should have been on java version 13. The dice game will still be expected driven on java 17.

10. Appendix

10.1 Literature

- Larman, Craig (2004). Applying UML and Patterns: "An Introduction to Object-Oriented Analysis and Design and Iterative Development, Third Edition." Chapter 6[1]

10.2 Git. repository

To track the progress of our project, we have utilized a Git repository. Our Git repository documents our commits, showcasing how our classes have evolved since the beginning of the project. Link to our Git-repository on GitHub: https://github.com/AsmaMaryam25/30_del1