

Danmarks
Tekniske
Universitet



02314 | 62531 | 62532

Introductory Programming | Development methods for IT-systems | Version control and test
methods

CDIO 1 Group 7

Friday 29th September, 2023 18:00

1 Hourly Accounting

The Work during the project and development of the program has been split equally between all group members. Every member of the group has been a part of the entire project. There is nothing that has been made for the project without someone from the group having not been a part of it in some way. The group has always been working together and at the same time either over Discord or at DTU Campus. The group has spent approximately 6 hours each week on the project. This is 6 hours working together at the same time and place.

AUTHORS



Mustafa Baker - s235092



Adel Wisam - s236076



Elias Larsen - s235125



Emil Savino - s235096



Frederik Bastrup - s235117



Mads Hartwich - s235102

Contents

1	Hourly Accounting	2
2	Summary	2
3	Introduction	2
4	Hourly Accounting	3
5	Versionsstyring	3
6	Requirement/Analysis	4
6.1	Diagrams/Models	5
7	Implementation	9
7.1	Testing	10
8	Conclusion	10
9	Appendix	11
9.1	Litterature	11
9.2	Code Upload	11

2 Summary

Using Java code, Git and GitHub, we have developed a two-player dice game. The report includes everything from the project, both product and project wise. This means all the thoughts and processes the group has had during the development of the program, all the requirements for the program, and how they are met. This also includes all the different diagrams and models that have been used in the development process. The report also includes how the work process has been, how we tested the program, the code used for the program, and the repository on GitHub, that the group has used for the program development.

3 Introduction

Introducing our Java project, which is a dice game. The game is made for a customer who described a program they wanted made and the requirements for that program. The dice game is a two-player game, where you play 1 against 1 on the same computer. The players will roll two dice at once, one at a time. Depending on the player's dice roll result, they will either get points, win, or lose all their points. The player will also get an extra turn if he get two of a kind on a dice roll. You win the game if you either have at least 40 points and get two of a kind on a dice roll while having a least 40 points or if you on two dice rolls in a row get two 6's.

4 Hourly Accounting

The Work during the project and development of the program has been split equally between all group members. Every member of the group has been a part of the entire project. There is nothing that has been made for the project without someone from the group having not been a part of it in some way. The group has always been working together and at the same time either over Discord or at DTU Campus. The group has spent approximately 6 hours each week on the project. This is 6 hours working together at the same time and place.

5 Versionsstyring

There has been created a public repository, to which both the members of the group and the customer have access. The repository on GitHub contains two branches - the main branch and the development branch. The development branch is used to pull and push the versions where the code is still under development, and the main branch is used to push and have the code that is either finished or works, but still needs some work to it. In the repository, you can also see who, what, and when each group member has committed something to the repository. This was also one of the requirements from the customer. They wanted to be able to see who, what, and when someone from the group has committed something to the repository. Also the latest version of the code works. In the repository, there is also a README- file, which contains information about the program for both the customer and the user. For the customer, there is information about the program and how it works and for the user, there is information about how to start the program. The user will not get any instructions or information about how the program works, since this also was one of the requirements from the customer. The program has to be playable without having to give them any instructions.

GitHub repository: <https://github.com/adel051700/CDIO-1>

6 Requirement/Analysis

Requirement specification

- The program has to work on the "databarens" computers.
- The program has to be playable without any given instructions before playing.
- The result of the dice roll should be shown in under 1/3 of a second.
- The program has to be playable for two players at the same time.
- The dice in the program have to be a 6-sided dice with the numbers 1-6.
- The game is done as soon as one of the two players gets 40 points and roll two of a kind after having at least 40 points.
- The player can also win if they roll two 6's, two times in a row.
- The program has to pass a test, where the result of 1000 dice rolls must only have a deviation of 10

Usecase

A game is played where the primary objective is to obtain a minimum of 40 points and then roll two of a kind to win the game. The game starts when player 1 rolls the dice the first time. Player 1 then gets a result between 2 and 12, and this result is added to the player's total amount of points. After this the turn passes to player 2, unless player 1 has hit 2 of a kind. In this case player 1 gets another turn. If player 1 rolls a pair of 1's, they lose all of their points, but still gets another turn. On the other hand if player 1 rolls a pair 6's two rolls in a row then player 1 wins the game. When player 1's turn passes to player 2 everything that applied to player 1 applies to player 2. This continues until a player has won.

6.1 Diagrams/Models

Class diagram

The class diagram is used to describe and show all the classes there will be in the software program. Here you can see all the different classes, what they are called, what the classes contain, how they work together, and how they affect each other.

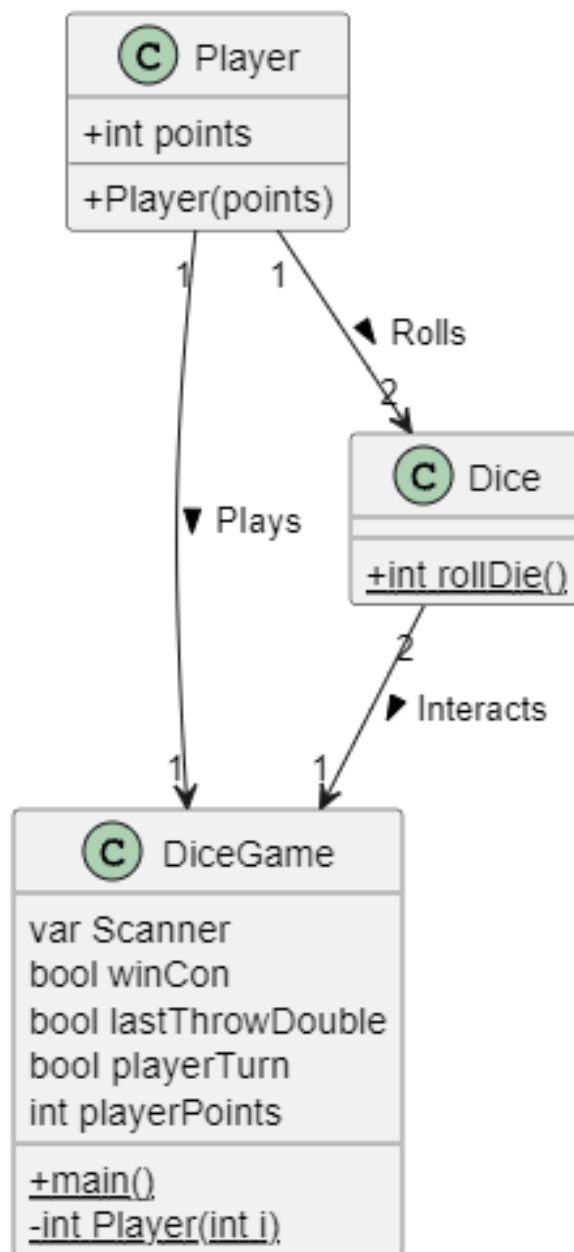


Figure 1: Class Diagram for DiceGame

Activity diagram

The activity diagram is used to show how the user interacts with the program. This means how the user gets from the start of the program, to the end of it. In this case, the group has chosen that the start of the program is when you press the play button and the end is when one of the players has won the game and the game ends. There has been made more than one activity program to show different ways the user can get from start to end in the program, since the player can win in multiple scenarios and therefore the user will not be in the same scenario every time the game ends. In other words, there can be different interactions between the user and the program to get from the start of the program to the end.

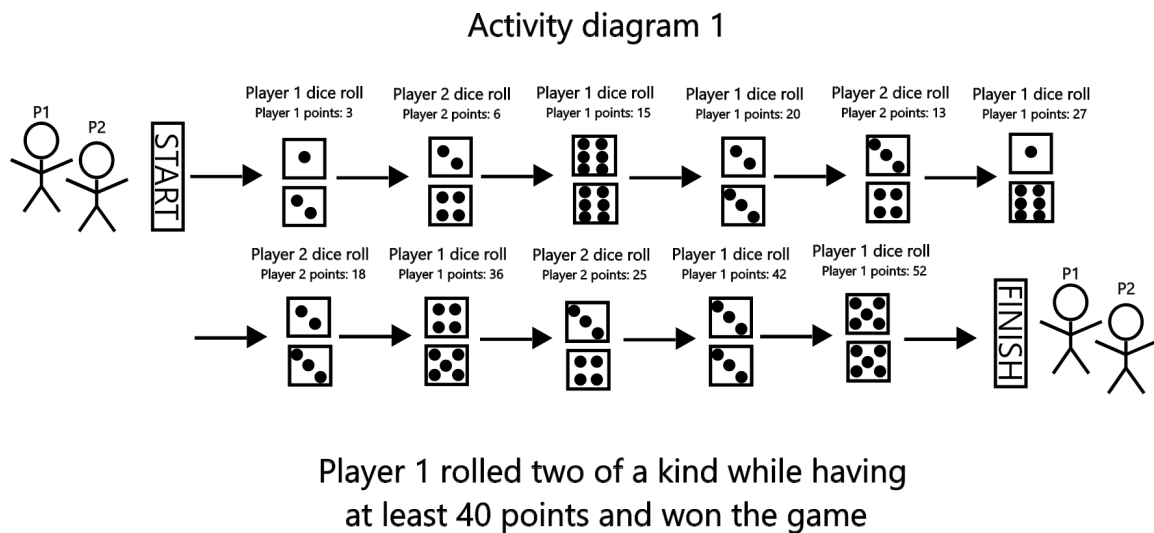
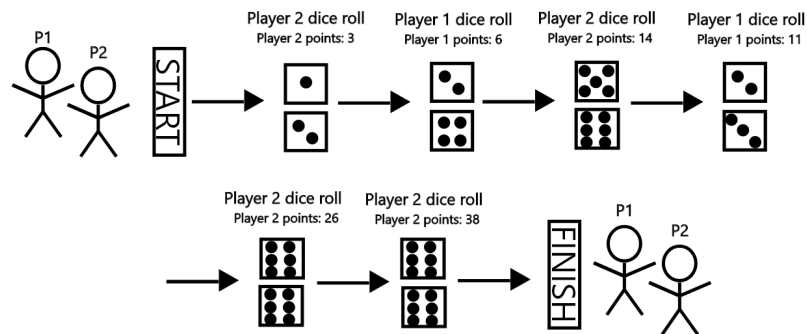


Figure 2: Activity diagram 1: Player 1 win by getting two of a kind while having at least 40 points.

Activity diagram 2



Player 2 rolled two 6's in a row and won the game

Figure 3: Activity diagram 2: Player 2 win by getting two 6's two times in a row.

Domain model

The domain model is very similar to the class diagram, but the class diagram shows all the classes in the program, how they interact with each other, and the code they contain. On the other hand, the domain model shows all the objects that are in the program and how they interact with each other. The domain model is usually more user-friendly and easy to understand for someone who knows nothing about the program or coding. The group has therefore also decided to construct a domain model for the program.

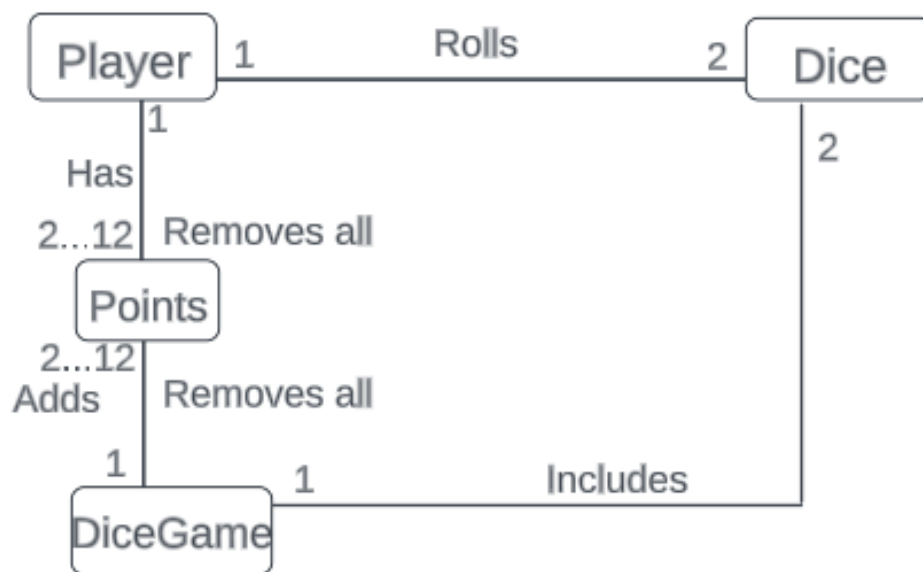


Figure 4: Domain model for the Dice Game

7 Implementation

When it comes to implementation of the project the group has had many thoughts and work, to see how different things would turn out and how we could make the best possible program and still stick to the requirements and expectations of the customer. The group has therefore taken use of FURPS ("functionality", "usability", "reliability", "performance" and "supportability").

FURPS

FURPS which stands for "functionality", "usability", "reliability", "performance" and "supportability" is a model used for classifying the software quality. In this project, it is therefore used to classify the software quality of the dice game program.

Functionality

The functionality is very simple. It is not a game with many requirements and therefore the functionality of the program is as said, very simple. It is always beneficial to have a program with a very simple functionality, if possible, as this means the program is easy for the users to use and get into.

Usability

The usability of the program is good. To improve usability we have created a .bat file that launches the game for the user. This means that you can launch the game with a simple double-click of the mouse and once you have opened the game, only one button is necessary to play.

Reliability

The reliability of the program is pretty high. The software should work almost perfectly and the chance of a program crash is close to zero. It is also a small program and there are not that many requirements for the software to work and not crash.

Performance

The performance is extremely good. The program requires almost nothing to run. You will be able to run the program on any modern computer no matter the specs of that device. The response time for the program is pretty good as well and is good enough to meet the requirement for the response time, which is no higher than 1/3 of a second.

Supportability

The supportability of the program is good since it is a small program. Therefore it is simple to both test, adjust, adapt, install, maintain, configure, and localize.

7.1 Testing

We have made iterative tests to confirm that our changes to the code have been effective and yielded the correct result. We have also constructed a probability test, that tests if the dice cup is giving the statistically correct results for the dice throws. The probability test analyses probability for sums and doubles over the course of 1000 hypothetical dice throws and outputs the statistical deviation from the theoretical probability.

We have also done a great amount of testing on the game's interaction with the user. Here we have tested the game to make sure that it performs as expected, depending on the value of the thrown dice. As we received wrong outputs, we have of course fixed our mistakes and used this testing to better understand our flaws in the code and the perspective of the user.

8 Conclusion

We have created a game, in which two players compete against each other to accumulate 40 points and the roll doubles to win, with a few extra rules on top of that. The customer's vision was a bit unclear, so we have clarified everything to give ourselves something concrete to work with. Diagrams and models have been constructed to showcase how the program works. To ensure maximum accessibility for all the members of the group to our project, we have used GitHub to push and pull our progress. A test was created too make sure our dice cup was as close to the statistical probabilities of rolling two dice as possible. At last we made sure that the customer does not need an IDE by converting the game into a bat-file.

9 Appendix

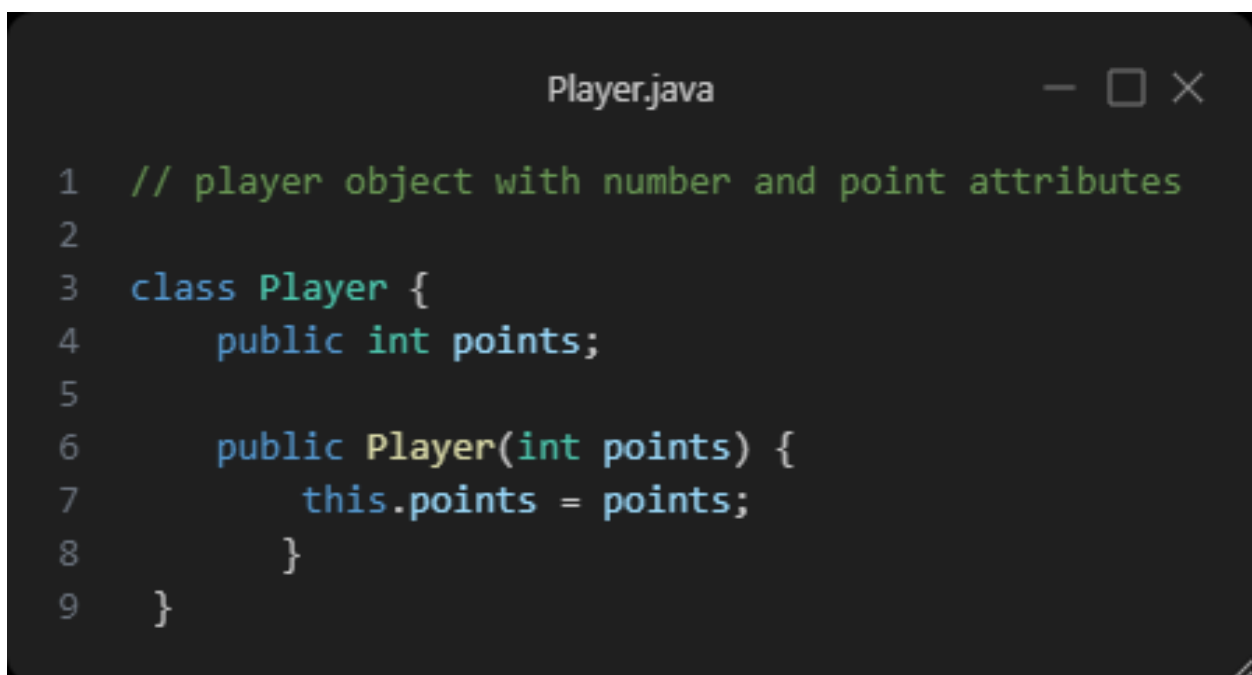
This is appendix

9.1 Litterature

We've made all the figures in this report

9.2 Code Upload

This is our code, for the CDIO 1 java project

A screenshot of a code editor window titled "Player.java". The window has standard window controls (minimize, maximize, close) in the top right corner. The code is written in Java and is as follows:

```
1 // player object with number and point attributes
2
3 class Player {
4     public int points;
5
6     public Player(int points) {
7         this.points = points;
8     }
9 }
```

Figure 5: Player Class

```
Dice.java

1  /**
2   * Class has method rollDie, that returns a random integer between 1 and 6.
3   */
4  import java.util.Random;
5
6  class Dice {
7
8      public static int rollDie() {
9          Random r = new Random();
10         int upBound = 6;
11         int rInt = r.nextInt(upBound);
12
13         return rInt + 1;
14     }
15 }
16
17 }
```

Figure 6: Dice Class


```
1  import java.util.Scanner;
2
3  class Dicegame {
4      public static void main(String[] args) {
5          var s = new Scanner(System.in);
6          var winCon = false;
7
8          int player1Pnt = Player(0);
9          int player2Pnt = Player(0);
10
11         boolean lastThrowDouble = false;
12         boolean playerOneTurn = true;
13
14         // The code snippet provided is a part of a dice game. It represents the main game loop where players
15         // take turns rolling two dice and accumulating points based on the sum of the dice rolls.
16         while (!winCon) {
17             int die1 = Dice.rollDie();
18             int die2 = Dice.rollDie();
19
20             if (playerOneTurn) {
21                 System.out.println("It's player 1's turn, press the enter button to throw the dice.");
22
23             } else {
24                 System.out.println("It's player 2's turn, press the enter button to throw the dice.");
25
26             }
27             s.nextLine();
28
29             if (lastThrowDouble && die1 != 6 && die2 != 6){
30                 lastThrowDouble = false;
31             }
32
33             // The code block you provided is checking if both dice are ones, and if not, adds points to the player object
34             if (die1 != 1 || die2 != 1) {
35                 if (playerOneTurn) {
36                     player1Pnt += die1 + die2;
37                     System.out.println("Player 1 rolls a " + die1 + " & a " + die2 + " and now has " + player1Pnt + "\n");
38                 } else {
39                     player2Pnt += die1 + die2;
40                     System.out.println("Player 2 rolls a " + die1 + " & a " + die2 + " and now has " + player2Pnt + "\n");
41                 }
42             }
43
44             // The code block you provided is checking if the two dice rolls are equal (i.e., if the player rolled
45             // a double). If the rolls are equal, it performs the following actions:
46             if (die1 == die2) {
47
48                 // This code block is checking if a player has accumulated 40 or more points and
49                 // has rolled a double (i.e., both dice have the same value). If these conditions
50                 // are met, it declares the player as the winner and sets the 'winCon' variable to
51                 // 'true' to exit the game loop.
52                 if (playerOneTurn && (player1Pnt - (die1 + die2)) >= 40) {
53
54                     System.out.println("Player 1 won, by having over 40 points and throwing identical die");
55                     winCon = true;
56
57                 }
58                 if (!playerOneTurn && (player2Pnt - (die1 + die2)) >= 40
59                 ) {
60
61                     System.out.println("Player 2 won, by having over 40 points and throwing identical die");
62                     winCon = true;
63                 }
64             }
65         }
66     }
67 }
```

Figure 7: First Part of DiceGame class

```
undefined - Dicegame.java

65 // The code block you provided is checking if the variable `lastThrowDouble` is `true` and if the value
66 // of `die1` (the first dice roll) is equal to 6.
67     if (lastThrowDouble && die1 == 6 && !winCon) {
68         if (playerOneTurn) {
69             System.out.println("Player 1 Wins");
70             winCon=true;
71         } else if (!playerOneTurn) {
72             System.out.println("Player 2 Wins");
73             winCon=true;
74         }
75     }
76 }
77
78 // This code block is checking if the value of `die1` (the first dice roll) is equal to 6. If it is, it
79 // sets the variable `lastThrowDouble` to `true`.
80     if (die1 == 6) {
81         lastThrowDouble = true;
82     }
83
84 // The code block you provided is toggling the value of the boolean variable `playerOneTurn`. If
85 // `playerOneTurn` is `true`, it sets it to `false`, and if it is `false`, it sets it to `true`. This
86 // is done to switch the turn between Player 1 and Player 2 in the game.
87     if (!playerOneTurn) {
88         playerOneTurn = true;
89     }
90     else {
91         playerOneTurn = false;
92     }
93 }
94
95 }
96 if (playerOneTurn) {
97     playerOneTurn = false;
98 }
99 else {
100     playerOneTurn = true;
101 }
102 }
103
104 } else {
105     if (playerOneTurn) {
106         System.out.println("Player 1 rolled two 1's and loses all points \n");
107         player1Pnt = 0;
108     }
109     else {
110         System.out.println("Player 2 rolled two 1's and loses all points \n");
111         player2Pnt = 0;
112     }
113 }
114 }
115 }
116 s.close();
117 }
118
119 private static int Player(int i) {
120     return 0;
121 }
122 }
123 }
```

Figure 8: Second Part of DiceGame class

```
undefined - TestDice.java

1  public class TestDice {
2      public static void main(String[] args){
3
4          // How many rolls you would like to test with
5          int amountOfTestRolls = 1000;
6
7          int sumis2 = 0;
8          int sumis3 = 0;
9          int sumis4 = 0;
10         int sumis5 = 0;
11         int sumis6 = 0;
12         int sumis7 = 0;
13         int sumis8 = 0;
14         int sumis9 = 0;
15         int sumis10 = 0;
16         int sumis11 = 0;
17         int sumis12 = 0;
18         int rollDouble = 0;
19         double deviation = 0.0;
20
21         for (int i = 0; i < amountOfTestRolls; i++){
22             int die1 = Dice.rollDie();
23             int die2 = Dice.rollDie();
24             int sum = die1 + die2;
25
26             if(die1 == die2){
27                 rollDouble++;
28             }
29
30             switch (sum){
31                 case 2 -> {
32                     sumis2++;
33                 }
34                 case 3 -> {
35                     sumis3++;
36                 }
37                 case 4 -> {
38                     sumis4++;
39                 }
40                 case 5 -> {
41                     sumis5++;
42                 }
```

```
undefined - TestDice.java

65
66     deviation += Math.sqrt(Math.pow((sum - 7),2));
67 }
68 double standardDeviation = deviation / amountOfTestRolls;
69 double afvigelse = (standardDeviation / Math.sqrt(amountOfTestRolls)) * 100;
70
71 System.out.println("You have rolled " + sumis2 + " 2's");
72 System.out.println("You have rolled " + sumis3 + " 3's");
73 System.out.println("You have rolled " + sumis4 + " 4's");
74 System.out.println("You have rolled " + sumis5 + " 5's");
75 System.out.println("You have rolled " + sumis6 + " 6's");
76 System.out.println("You have rolled " + sumis7 + " 7's");
77 System.out.println("You have rolled " + sumis8 + " 8's");
78 System.out.println("You have rolled " + sumis9 + " 9's");
79 System.out.println("You have rolled " + sumis10 + " 10's");
80 System.out.println("You have rolled " + sumis11 + " 11's");
81 System.out.println("You have rolled " + sumis12 + " 12's");
82 System.out.println("You have rolled " + rollDouble + " doubles");
83
84 System.out.println("This test deviates " + afvigelse + "% from the actual probabilities when rolling two dice");
85 System.out.println("This test has a standard deviation of " + standardDeviation + " from the actual probabilities when rolling two dice");
86
```

Figure 10: Second Part of TestDice class