CDIO nr. 1

Mathias Tværmose Gleerup, s153120,
Mikkel Geleff Rosenstrøm s124363,
Sofie Freja Christensen s153932
Morten V. Christensen s147300
Simon Lundorf s154008
Jonas Larsen, s136335,

Group nr. 15 Classes: 02312, 02313, 02315

Deadline : October 7th 2016

October 7, 2016

Danmarks Tekniske Universitet


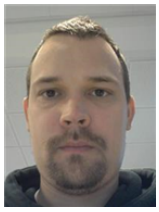



Table 1: Hours registration

CDIO1	Jonas	Mathias	Mikkel	Morten	Simon	Sofie	Sum
Conceive	0,5	1,5	1,5	1,5	0	0	5
Design	3	3	1,5	3	3	3	16,5
Impl.	3	3	3	3	1	4,5	17,5
Test	2	4	0	0	0	2	8
Dok.	2	2,5	3,5	2	6,5	2	18,5
Andet.	0	0	0	0	0	0	0
Sum	10,5	14	9,5	9,5	10,5	11,5	65,5

Contents

1 Abstract

A program has been compiled in java. The program is designed, to be a dice game, in which the two players competes to reach 40 points. There has been added bonus features to the game, to make it harder to win, which include reset of points and special demands to win.

2 Foreword

This report takes its foundation in the program (CDIO 1), drawn from the following courses Indledende programmering (02312), Udviklingsmetoder til IT-systemer (02313) og Versionsstyring og testmetoder (02315). All of which takes place at DTU campus Lyngby, fall of 2016. The project is made in collaboration by the members of group 15 :

- Jonas Larsen - SWT
- Mathias Tværmose Gleerup - SWT
- Mikkel Geleff Rosenstrøm - ITØ
- Morten Velin Christensen - SWT
- Simon Lundorf - SWT
- Sofie Freja Christensen - SWT

3 Requirements

3.1 **Has to be useable on computers in DTU's databars.**

(a) As the program is written in Java-code, it is automatically compatible with Windows.

3.2 **The game has to take place between two players.**

3.3 **There has to be two dice in one cup.**

(a) The value of any given roll has to be between the value of 2 and 12.

3.4 **The game is played with two six sided dice.**

3.5 **Each player has one roll per turn.**

3.6 **The result of a roll has to be displayed immediately.**.

3.7 **The sum of the resulting roll has to be added to the points of the player whose turn it is.**

3.8 **The game ends when a player reaches 40 points.**

(a) The player to first reach 40 points win.

(b) The game has to announce the winner.

(c) The program terminates upon declaring a winner.

3.9 **Useability: The game has to be playable without a manual.**

(a) A short explanation of rolling doubles and of the win conditions have been added.

3.10 **Documentation has to be in Danish or English, technical terms has to be used naturally.**

3.11 **The cup has to be tested to 1000 rolls.**

(a) After this test the expected value has to be statistically equivelant to rolling an actual fair six sided die.

3.12 **There has to be a way to terminate the program early.**

4 Requirements, addendum.

1. On a double 1, player looses all their points.
2. Player gains another turn on any double of a kind roll.
3. If two double sixes are rolled in succession the player wins.
4. once a player reaches 40 points, a two of a kind roll is required to win.

(a) There is no mention of what will happen should a player reach over 40 points and then roll a double 1. We've implemented that rolling a double 1 over 40 will still make you loose your points.

5 Use Case and Sequence Diagram

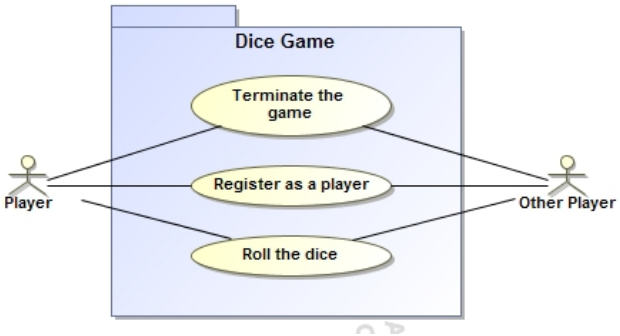

Figure 1: Use case
Use case diagram detailing briefly how the diffent users relate to the program.

Use case: Cup Game
ID: 1
Brief description: The game is to be played by two players. Each player takes a turn to roll the two die. The number of eyes are the points acquired. The game is won when you: (1) Have over 40 points and roll a double 2, 3, 4, 5 or 6 (double 1 resets your score). or (2) Roll two sixes in a row.
Primary actors: Players.
Secondary actors: Points.
Preconditions: Two players enters the game.
Main flow: 1. Player one throws the die. 2. The number of eyes on the die, represents the amount of points earned. 3. The points are saved and added to pre-existing points. 4. This is repeated for each player. 5. If a player rolls a double 2, 3, 4, 5 or 6 the player gets another roll. 6. If a player rolls a double 1, that players score is reset. 7. The game terminates when a player has over 40 points and rolls a double 2, 3, 4, 5 or 6.
Postconditions: The winner is announced.
Alternative flows: 1. If a player rolls two double 6 in a row, that player wins the game. 2. If a player refuses to roll, the game ends, with no winner.

Figure 2: Cupgame
Use case diagram detailing briefly how Cup game works. We used it to give us a reference when creating the SSD and eventually the code.

Use case: Use cup
ID: 2
Brief description: The players activates the cup, and two, righteous six sided - dies are thrown. Then the result is revealed.
Primary actors: The cup.
Secondary actors: The two players.
Preconditions: None.
Main flow: 1. Player 1 activates the cup. 2. Result is shown.
Postconditions: The dies "eyes"are added together and the point score is updated. The cup is given to Player 2.
Alternative flows: None.

Figure 3: Usecup

Use case diagram detailing briefly how Use cup works. We used it to give us a reference when creating the SSD and eventually the code.

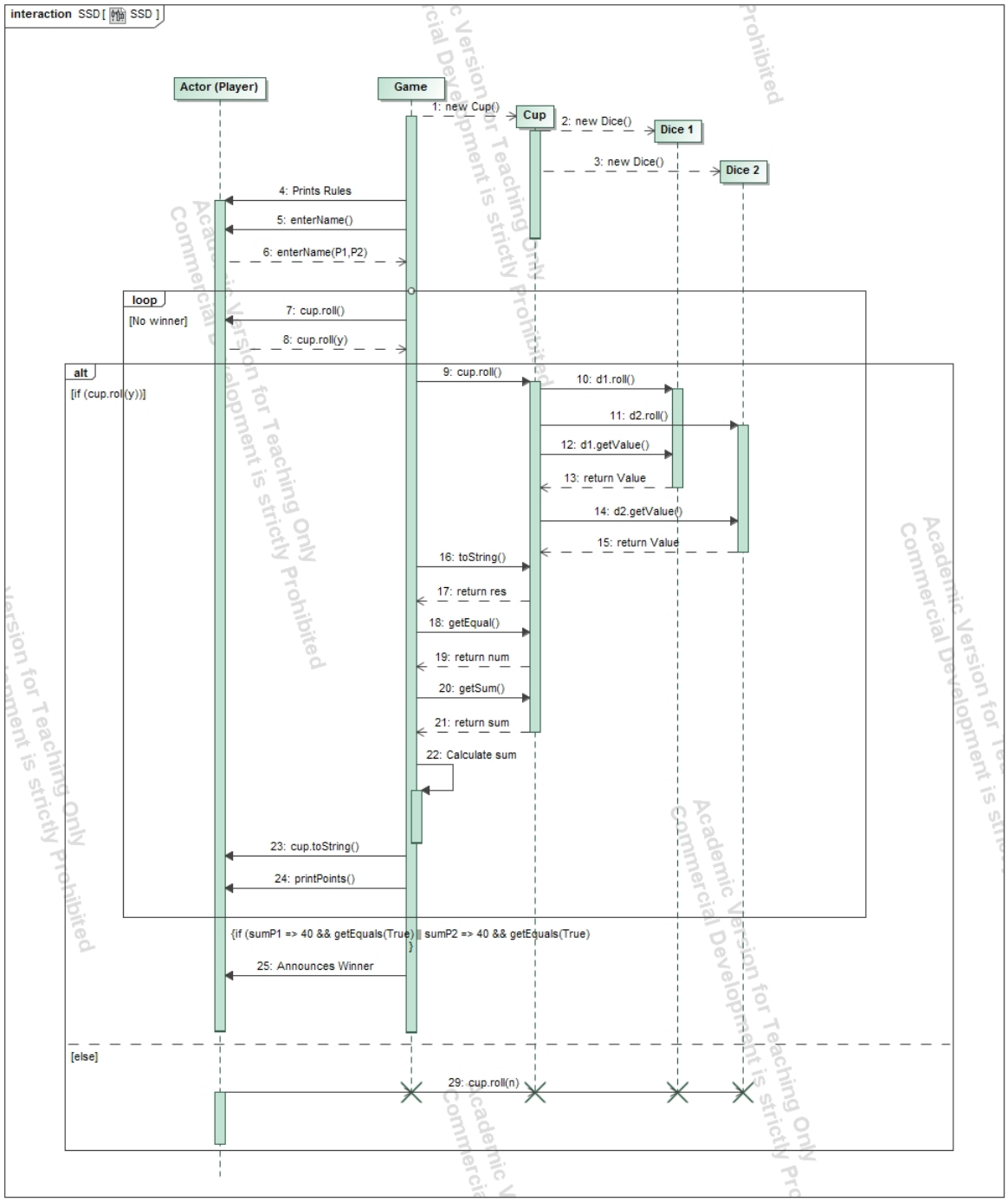

Figure 4: Sequence Diagram

Detatiled diagram depicting the different actions taken during a game of dice. Due to the relative simplicity of the program we decided to merge the SSD of Cup game and Use Cup. Resulting in an overview.

6 Procedure

1. Started with basic system, without extra assignments or GUI.
   (a) Discussed system boundaries, as well as actors and created use cases.
   (b) Used casual domain modelling to identify and conceive conceptual classes.
   (c) Began programming classes and main.
   (d) Discussed the possiblity to use the GUI at hand.
      i. It was decided that the GUI feature would not be implemented.
   (e) Tested the basic system.
2. Revisited SRS to discuss implentation of extra assignments.
   (a) Updated use case diagrams to implement alternative flows.
   (b) Changed main to catch alternative flows.
   (c) Tested system.
   (d) System finished.

7 Conclusion

The creation of the program was concluded as expected and on time to specifications.
We looked at the relations between the different classes and concluded through testing, both JUnit and bruteforce methods, that the program worked as outlined by the specifications detailed in the requiments section.
Despite scheduling conflicts there has been little, to no friction between memebers and the cooperation has been fruitful and enjoyable. We clearly made progress in accomodating different skill levels and pre-existing knowledge. The work process was hampered by our inability to predict the workflow, that deficiency has been remedied through discussions of expectations and the implimentation of new programs and functions into our work flow. We have planned how to accommodate these challenges for the next assignment.

8 Test

8.1 testPrintTime

This test was made to ensure that the results of a Cup.roll() was printed fast enough, e.g. The Project Leaders comments.
We have had input that determines this test as obsolete because we don't use the GUI, but you can never be to sure.
The test case used System.nanoTime(); to calculate the time spent on the lines of code in question.

8.2 DiceMeanTest

This test was made to check that the class Dice creates objects that are fair (having a mean of 3.5).
In testing this we established an upper and lower threshold, beacuse it would not, over a hundred thousand rolls, get a mean of exactly 3.5.
Test passed.

8.3 Brute force testing

We tested main by running the game many many times. Also tested by inserting certain numbers.
For example, we inserted six instead of the .getEqual() method in a switch statement to test wether the

8.4 DiceTest

Prints how the rolls are distibuted over the possible values of the dice.
Tester evaluates the printed results to assert passed/failed.

8.5 CupTest

Prints how the rolls are distibuted over the possible results of rolling an object of the type Cup.
Tester evaluates the printed results to assert passed/failed.

9 Litteraturliste

References

[1] Craig Larman, Applying UML and Patterns 2004.

[2] Lewis and Loftus Java Software solutions 7th ed.