Danmarks Tekniske Universitet

# CDIO 1

**Project by**

**Søren Nielsen (s205820), Noah Nissen (s235441), Christian Worsøe (s235111), Andreas Jensen (s235455), Alex Lundberg (s235442),**

**Development Methods for IT Systems Fall 23, 62531**

**Technical University of Denmark**

# Contents

# 1 Summary

To summarize the whole project, this project has involved designing and implementing a dice game, for a customer. The customer gave some requirements, and those had to be parsed into something useful, so the whole game could be a success. The project utilizes agile development, to iterate through the whole process multiple times, and is version controlled with GitHub. Tests are done at the end, where quality is assured, and to make sure all the customers requirements are satisfied. The game itself is about throwing 2 dice, and then a plethora of things happen, according to what eyes each die is showing. Getting two of the same amount of eyes, means another turn, and its even possible to win by getting 2 sixes after getting 2 sixes the turn before. You also receive points according to the sum of eyes shown, but if you get two 1's your score gets reset. After getting 40 points, you can also win by simply getting 2 of the same amount of eyes in one throw. Other than that, the game is turn bases and switches between each player for each throw, unless someone unlocked a second turn.

# 2 Hourly accounting

| Hour count | Inception | Elaboration | Construction | Report writing |
|---|---|---|---|---|
| Søren | 3 | 3 | 2 | 2 |
| Noah | 3 | 2,5 | 2 | 2 |
| Christian | 3 | 3 | 2 | 2 |
| Andreas | 2 | 2 | 2 | 3 |
| Alex | 3 | 3 | 2 | 2 |

Table 1: The estimates of work hours put into various tasks.

# 3 Introduction

The purpose of the game is to satisfy the customer's needs and requirements. This is done by reading the customer's vision, and revising what is written, for the project many times and being in close contact with the project leader to determine the best solutions for the customer's needs.

The entire project was split up into phases to better focus on specific things, but also to better parallelize, the process, allowing for multiple people to be working on different tasks at once, allowing for faster development, and to catch any mistakes on the way that could cause larger issues down the road if not taken care of.
All of this is done by using UML methods such as:

Inception:
Inception is finding the key features of the project – how does it benefit a customer and what are the potential risks involved while making the project? This was done with UML tools such as Use case diagrams and domain models.

Elaboration:
The elaboration phase is the actual building of the product. For this, GitHub was used to parallelize the work.

Construction:
In the Construction phase, testing was implemented to verify the functionality of the game, including the dice. The non-essential features were also implemented in this phase. The game was also tested on one of the data bars at DTU, as the ability for it to run here was a requirement.

# 4 Project planning

| | | | |
|---|---|---|---|
| CDIO1 | Start: | | |
| | Week_1 | | |
| | Monday | Tuesday | Friday |
| On the day: | Github | Rapport | |
| | Class model | Usecases | |
| | Plan out project | Recuirements | |
| | | | |
| | | | |
| | | | |
| For next time: | | | |
| | | Code the 5 base features | |
| | | | Deadline |
| | Week_2 | | |
| | Monday | Tuesday | Friday |
| On the day: | Implementation | Conclusion | |
| | Testing | Summary | |
| | | Hourly accunting | |
| | | Table of contents | |
| | | Introduction | |
| | | Appendix | |
| For next time: | | | |
| | Code 4 extra features | Finish report | |

Table 2: The project plan.

Here is the project plan, where every day is set up in what will be done on the day, and what will be done for the next time. The subjects on the day are not in order, but more so a plan for everything that needs to be done on the given day.
As seen in the table, there are some days with alot of tasks and days with very few, and the project has been planned, so all the heavy/hard tasks would be done at first, especially those who require extra groundwork or group discussion. After those are done, all the mundane and easy to do task are listed, and that is why the table looks the way it does.

# 5 Requirements/Analysis

To start the project of we had the following statement from the customer.

"Vi vil gerne have et system, der kan bruges på maskinerne (Windows) i databarerne på DTU. Det skal være et spil mellem 2 personer.

Spillet skal gå ud på at man slår med et raflebæger med to terninger og ser resultatet med det samme. Summen af terningernes værdier lægges til ens point. Vinderen er den, der opnår 40 point. Hvis der er ressourcer til det, er der følgende ekstraopgaver:

Spilleren mister alle sine point hvis spilleren slår to 1'ere.
Spilleren får en ekstra tur hvis spilleren slår to ens.
Spilleren kan vinde spillet ved at slå to 6'ere, hvis spilleren også i forrige kast slog to 6'ere uanset om det er på ekstrakast eller i forrige tur.
Spilleren skal slå to ens for at vinde spillet, efter at man har opnået 40 point.

Vi forventer at alle almindelige mennesker kan spille det uden en brugsanvisning.

Vi vil gerne se en test, der beviser at raflebægeret virker korrekt, hen over 1000 kast.

Det er op til jer om dokumentation og kode skal være på dansk eller engelsk, dog skal fagudtryk være naturlige."

From this we can determine what actors and use cases we need.

We need to make dice game which has 1-2 actors (the players). First the players will need to start the game locally on 1 computer (one of the computers at DTU). The players will need to coordinate with each other who is player 1 and who is player 2. After initiating the game, player 1 will be prompted with the ability to roll 2 dice. After rolling the 2 dice, the sum of the face values of the dice will be displayed to them immediately after (within 333 ms,) and the sum is added to their total points. After player 1's turn, the same process will happen for player 2. The game ends when one of the 2 players reaches 40 points, which is the condition for winning. This process needs to be completed by the player without any prior knowledge or guidance.

Use cases:

- The user presses something to start the game.
- The user then rolls the die by pressing a button.
- The user sees their roll and it gets added to their points.
- Alternates turn to the other player.
- Repeat process.
- A player wins the game at 40 points.

Actors:

- Player

By looking at the use cases we can determine the following requirements for our game

Requirements:

- Game should run locally on the PC.
- The turn switches between two players.
- They will be throwing two six sided dice, with the number 1-6 on the sides, for a total of 12 different outcomes.
- The result of the dice should be shown within 333 ms.
- The players start with 0 points.
- You win by achieving 40 points or more.
- The game should be playable by the average DTU student.
- The game should be playable without a tutorial, by intuition or simple on-screen instructions.
- The dice should be random, and always produce a random number between 1-6.

# 6 Design

By looking at the figure, it's clear that there will be 5 different features in the base game. There will also be an additional 4 extra features. This makes a total of 9 different feature branches. The 4 extra features will, however, only be implemented once the base features are implemented and confirmed to be working as intended.

After having made the use case diagram 1 we made a system with our player as a primary actor and the game controller as a supporting actor. Where the only thing that the player does not need to have a hand in is swapping turns between players.

And then on the basis of this as well as the requirements set out by the client we then developed a first draft at the system by making a class diagram. 2 where we here created objects to help fulfill the use cases.

after having made the initial product, consisting of the base requirements which we had modelled in the class diagram 2, and implementing them, see next chapter, we then reiterated the class diagram to include everything we found we needed after the implementation as well as some extra attributes and methods needed to fulfill the extra requirements. See the extended Class diagram 3.

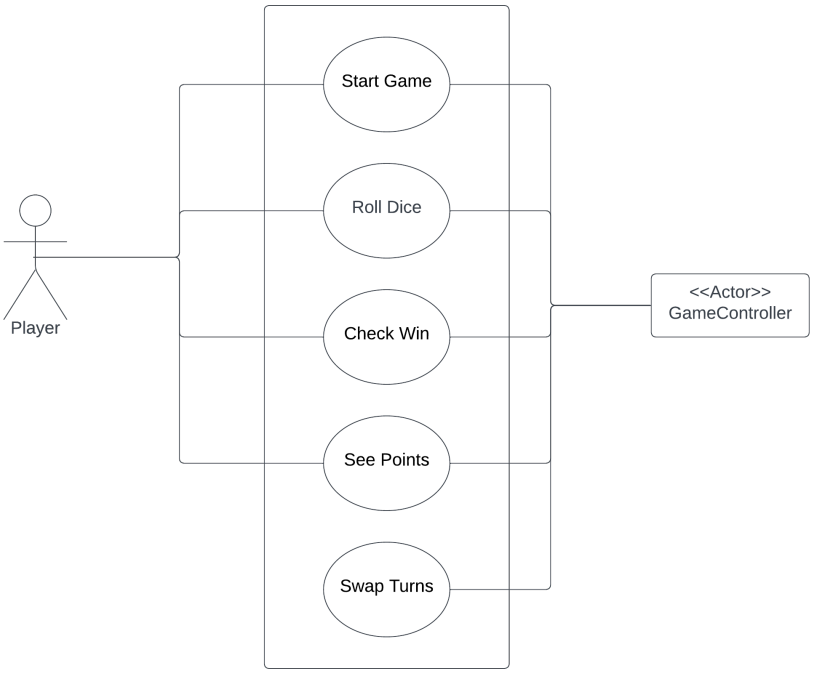


Figure 1: Use case diagram

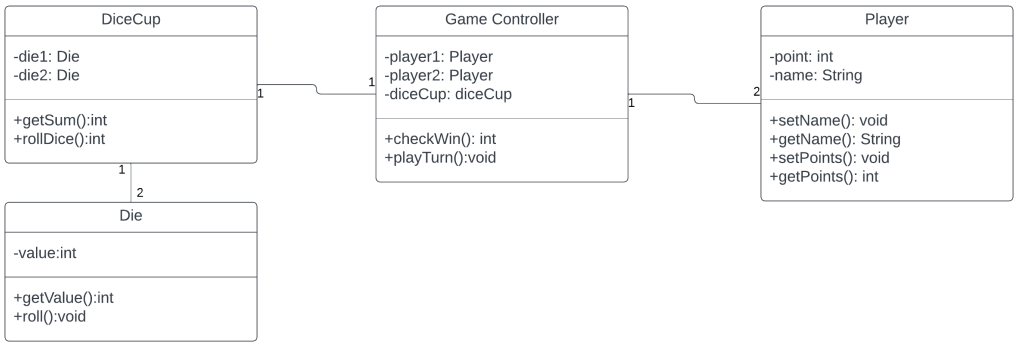


Figure 2: Base version of class diagram

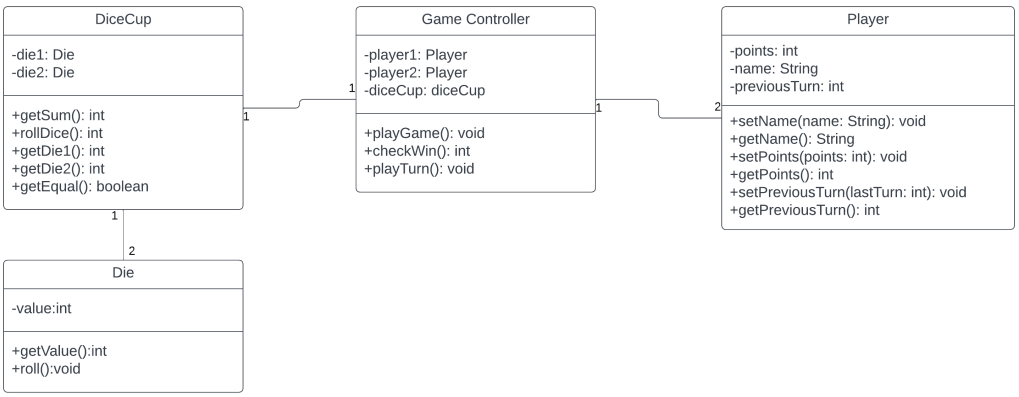


Figure 3: extended version of class diagram

# 7 Implementation

The product was implemented by following the original class diagram that was created early on. The empty master branch was branched off to development, which then was branched off for every base feature. Within

each feature branch, the individual classes were written following the class diagram, although with additions, as the necessities of the individual classes were redefined throughout the process. For example, an `addPoints` method was added to the `Player` class. This method was not a part of the original class diagram for the `Player` class, but during implementation, we quickly realised that this was needed.

When a specific feature was fully implemented and confirmed to be working, the corresponding branch was merged into development and then deleted. After all the base features had been implemented, we could merge the development branch into the master branch, as we now had an initial fully working version of the game. The extra features were then implemented using the same procedure with feature branches.

### 7.1 Code structure

The code is sectioned into four different files, each containing a class, and then the associated methods and variables. The main function is located in the `GameControl` file. It simply creates an instance of `GameControl` and calls `playGame()` on it:

```
1 public static void main(String[] args) {
2
3     var game = new GameControl();
4     game.playGame();
5
6 }
```

## 8 Testing

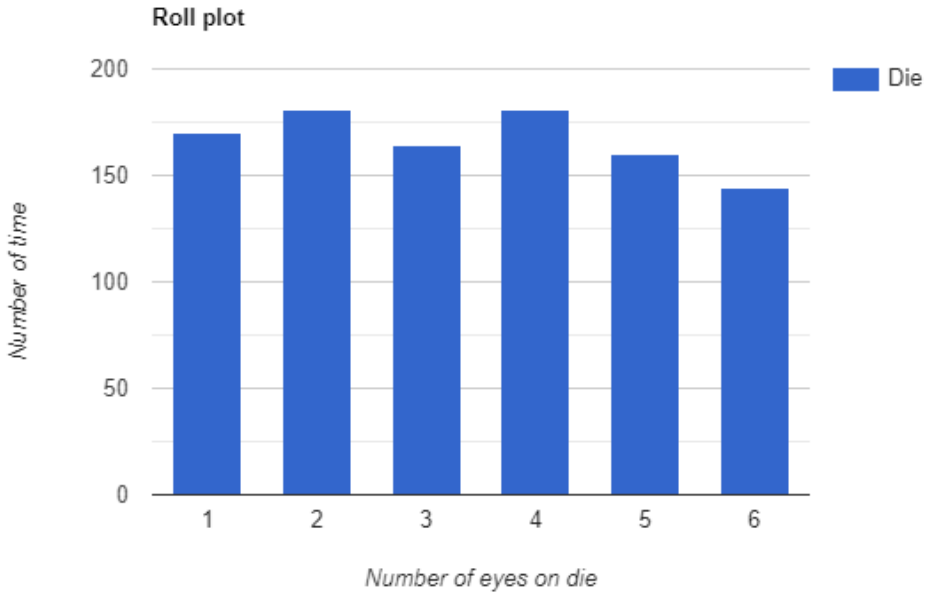


Figure 4: Bar plot of die out come

In figure 4 it is shown how often a dice outcome is rolled over 1000 rolls. This test is done to see if the function used for random numbers in Java is actually random. This is necessary because the game needs the dice to be random, as described by the customers in the product description.
For this project Math.random() was used over java.util.random.nextInt() because Math.random() only can return a double value wheres as java.util.random.nextInt() can return untegers, longs, float, double and boolean. All of that is not needed and therefor Math.random() was choosen.

# 9 Conclusion

After completing this task, its possible to conclude that developing a game for a customer is possible with the guidelines received from the customer. Of course some things had to be specified during the process, and the customers vision had to be laid our clear and concise, but after analysing and finding the requirements it was quite easy to start the design. Utilizing GitHub to manage version control and by following the project plan, that was developed upon receiving the request from the customer, it was possible to design and implement the game, and the extra features the customer had requested. After finishing the game without the extra features it was tested and found to be working, with dice that would be random on each throw. After the testing of the base game, the extra features were added and confirmed to be working, before they were merged into the main branch of the games development tree. Each feature was coded on a separate branch, before being merged to development, were they received hotfixes and so on, before going to the master branch as a working part of the game.

# 10 Appendix

## 10.1 Code

```
1  public class Die {
2
3      public int value;
4
5      public void roll() {
6          var val = (Math.random() * 6) + 1;
7          value = (int) val;
8      }
9
10     public int getValue() {
11         return value;
12     }
13
14 }
15 public class DiceCup {
16
17     Die die1 = new Die();
18     Die die2 = new Die();
19
20     public void rollDice() {
21         die1.roll();
22         die2.roll();
23     }
24
25     public int getSum() {
26         return die1.getValue() + die2.getValue();
27     }
28
29     public boolean getEqual() {
30         return die1.getValue() == die2.getValue();
31     }
32
33 }
34 class Player {
35
36     private String name;
37     private int points;
38
39     // Constructor
40     public Player() {
```

```
41          this.name = "";
42          this.points = 0;
43      }
44
45      // Constructor with name overload
46      public Player(String givenName) {
47          this.name = givenName;
48          this.points = 0;
49      }
50
51      // Constructor with name and point overload
52      public Player(String givenName, int startPoints) {
53          this.name = givenName;
54          this.points = startPoints;
55      }
56
57      public String getName() {
58          return this.name;
59      }
60
61      public void setName(String nameToSet) {
62          this.name = nameToSet;
63      }
64
65      public int getPoints() {
66          return this.points;
67      }
68
69      public void setPoints(int score) {
70          this.points = score;
71      }
72
73      public void addPoints(int toBeAdded) {
74          this.points += toBeAdded;
75      }
76
77  }
78  class GameControl {
79
80      Player player1 = new Player();
81      Player player2 = new Player();
82
83      DiceCup diceCup = new DiceCup();
84
85      public void playGame() {
86          // This function makes use of ANSI escape codes to move the
87          // cursor around, clear lines, and to do a bit of formatting
88
89          boolean isPlayerTwo = false;
90          Player currentPlayer;
91          var scanner = new java.util.Scanner(System.in);
92
93          try {
94              // Ask the user for the player names
95              String input;
96
97              System.out.print("Player 1 name: ");
98              input = scanner.nextLine();
99              player1.setName(input.isBlank() ? "Player 1" : input);
```

```
100
101             System.out.print("Player 2 name: ");
102             input = scanner.nextLine();
103             player2.setName(input.isBlank() ? "Player 2" : input);
104
105             // Clear the previous two lines and move the cursor back up
106             System.out.print("\033[A\033[K\033[A\033[K");
107
108             // The format string used to print player info
109             // The first part clears the line to remove any previous text
110             var formatString = "\033[K\033[%sm%-" + Math.max(
111                     player1.getName().length(),
112                     player2.getName().length()) + "s\033[m    %3d%s";
113
114             // Write the text at the bottom telling the player how to roll
115             System.out.print(System.lineSeparator().repeat(6));
116             System.out.print("Press [Enter] to roll\033[5F");
117             while (true) {
118                 currentPlayer = isPlayerTwo ? player2 : player1;
119
120                 // Write out the player info (name, points, whether it's their turn)
121                 System.out.format(formatString + "%n",
122                         isPlayerTwo ? "35" : "95",
123                         player1.getName(), player1.getPoints(),
124                         isPlayerTwo ? "" : "    Your turn!");
125                 System.out.format(formatString + "%n%n",
126                         !isPlayerTwo ? "34" : "94",
127                         player2.getName(), player2.getPoints(),
128                         !isPlayerTwo ? "" : "    Your turn!");
129
130                 // Wait for the player to press Enter
131                 scanner.nextLine();
132                 // Make sure to clear any text they may have written
133                 System.out.print("\033[A\033[K");
134                 // Colour the name in the text saying what the player just rolled
135                 System.out.print("\033[" + (!isPlayerTwo ? "35" : "34") + "m");
136
137                 playTurn(currentPlayer);
138
139                 // Move back up to the first line of player info
140                 System.out.print("\033[3F");
141
142                 if (checkWin(currentPlayer)) {
143                     // Rewrite the player info to show what the final points are
144                     // Also write a "You won!" instead of "Your turn!"
145                     var won = "    \033[32mYou won!\033[m";
146                     var lost = "    \033[31mYou lost!\033[m";
147                     System.out.format(formatString + "%n", "35",
148                             player1.getName(), player1.getPoints(),
149                             isPlayerTwo ? lost : won);
150                     System.out.format(formatString + "%n%n", "34",
151                             player2.getName(), player2.getPoints(),
152                             !isPlayerTwo ? lost : won);
153                     // Move down to the "Press [Enter] to roll" line and clear it,
154                     // allowing the shell prompt, if present, to appear at that line
155                     System.out.print("\033[2E\033[K");
156                     break;
157                 }
158                 // If the die dont have the same value the turn switches
```

```
159                 if (!diceCup.getEqual()) {
160                     isPlayerTwo = !isPlayerTwo;
161                     System.out.println("\033[A\033[K");
162                 } else {
163                     System.out.println("\033[A\033[32mCongratz you've gotten another turn\033[m");
164                 }
165             }
166         } catch (java.util.NoSuchElementException e) {
167             // This happens when you press Ctrl+C
168             // Simply ignore the exception in this case
169         } catch (Exception e) {
170             System.err.println("\033[J\033[31mSomething went wrong\033[m");
171             e.printStackTrace();
172         } finally {
173             scanner.close();
174         }
175 
176     }
177 
178     public void playTurn(Player player) {
179 
180         diceCup.rollDice();
181         var sum = diceCup.getSum();
182 
183         System.out.format("%s\033[m rolled a %d and %d (=%d)",
184                 player.getName(), diceCup.die1.getValue(), diceCup.die2.getValue(), sum);
185 
186         player.addPoints(sum);
187 
188     }
189 
190     public boolean checkWin(Player player) {
191 
192         return player.getPoints() >= 40 && diceCup.getEqual();
193 
194     }
195 
196     public static void main(String[] args) {
197 
198         var game = new GameControl();
199         game.playGame();
200 
201     }
202 
203 }
204 
```