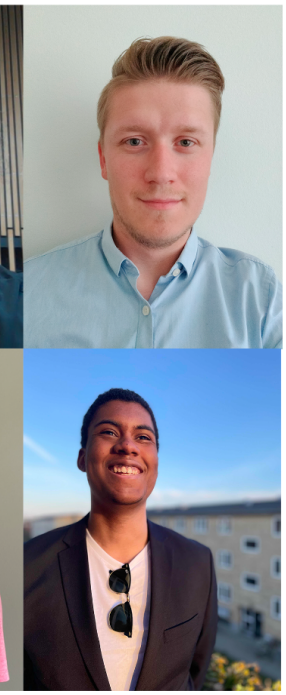


02314 62531 62532

INTRODUCTORY PROGRAMMING, DEVELOPMENT METHODS FOR IT SYSTEMS, VERSION CONTROL AND TEST METHODS

# CDIO 1


Mathias J. Hidan
s215874

Nicolai D. Madsen
s213364

Sofus J. Brandt
s214972

Adrian A. C. Macauley
s225733

# 1. Summary

Our assignment is to create a game for the Windows PC's you can find around DTU Campus' data bars and have it run smoothly. The game itself needs to have 2 active "dice" which needs to be simulated through the computer, and it needs to be able to have 2 players at the very least. There are 4 bonus parts to the game which can be added as you wish, so it could be more fun to play. We have decided to call them "expansion packs". They are as follows:

1. If you roll 2 1's in one dice throw, you lose all your points.
2. If a player rolls two of the same number in a dice throw they get to throw the dice anew.
3. If a player over 2 consecutive throws manages to hit a double 6 on both instances, they automatically win the game.
4. If a player has 40 or more points they need to throw and have two of the same number to win the game.

With a group working together striving for success and hungry for perfection we tried to make a game which had all the versatility both be played only as the base game, and with 0-4 expansion packs active. Even with us having endured one of our colleagues leaving the project due to personal reasons we managed to create a very substantial and great product. We managed to create a better than desired product for the customer in question, with all the requirements met and the scope being unchanged for the time being.

# 2. Hourly accounting

All members have used 10 hours outside of University classes to complete and provide value to the group.

# 3. Table of contents

# 4. Introduction

Our assignment this time is to create a game for the windows PC's you can find around DTU Campus' and have it run properly. The game itself needs to have 2 active "dice" which needs to be simulated through the computer, and it needs to be able to have 2 players at the very least. There are also a bunch of extra bonus parts to the game which can be added as you wish, so it could be a little more fun to play than just the base game. We have decided to call them expansion packs and they will also be introduced later with certain functions and funky interactions in the game. The plan is to provide a finished document by the date: 29/09/23.

# 5. Project Planning

## 5.1. Project Objectives

We were provided with an opportunity by DTU's administration to make a game for their pc's in the databar which are of middling quality. First and foremost we wanted a program which could withstand running on a Windows computer of middling quality, so we decided to make our game in Java with a CLI since it is easy on the computer. The goal of this is to make sure we can run the game in a good state and make fulfilling DTU's aspiration of a simple game which all pc's can play a reality. (Look in "Requirements/Analysis" for a more detailed explanation of our requirements and decisions in this project.)

## 5.2. Scope

The scope of our product is to have 1 base game, which is easily accessed through one file, and having 4 expansion packs with base game altering rules, to make sure the experience is enhanced when playing the game. The base game consists of a summation function that counts up towards 40, and the 2 six sided dice you need to roll. The amount of points you gain is based upon your roll with said dice we have created with a math function. You win when you reach 40 points. This includes one

pack that has a function which resets points and another pack for one bonus turn if you hit the same number on both dice. two packs which alter the rules and change the way you can win the game. This being, hit a roll with 2 of the same number and win for the one pack and for the other we decided to add a function which makes you win if you roll 2 6's twice in a row. The packs are easy to incorporate in the pregame phase. The game should be easily runnable with and without the expansions, and make it so easy to understand that even someone without the rules can play and follow along for fun.

## 5.3. Schedule

With our Schedule, finishing up the primary part of the programming on the 25th of September 2023, it became very apparent that it might not be a possibility to do so, therefore we pushed it a bit to the 26th being a more realistic date. Having all the programming done we could fish up our use cases which were also being worked on by the group from the 18th (start of project) so we could make sure they were all adequate representations of how our program flowed throughout its execution.

## 5.4. Resource allocation

We have all put in a lot of work to the project but Nikolai stood for most of the programming, Adrian programmed and managed the team, Sofus stood for testing and programming, Mathias programmed and stood for implementation of our program on DTU. We are 4 people who are working on this project and have all committed at least 26 hours a week to completing this project. Adrian, responsible for managing, creating and upholding plans, he programmed expansion pack 1 and the dice function also. Nicolai, stood for programming the bulk of the program and has throughout the process been our main programmer, he also assisted with the testing kit and did expansion pack 2, Player class and Gamemanager class. Sofus primarily stood for test kit programmed expansion pack 4, was our primary use-case writer and analysis provider. Mathias did expansion pack 3, assisted with the programming. We have programmed in Java and used most of the basic packages that come in with the VsCode extensions which provide a very good base for our use-case. Hereby also creating many methods and classes which can do the repetitive tasks inside our program.

## 5.5. Risk Management

The risks in this project are not limited to Computers crashing, the code not working as desired, and many more things (look at our matrix for our concerns.) We have of course taken measures to try to prevent the most likely scenarios, such as rigorously testing our code and making sure that our dice fill out the basic requirement of "randomness". In the case of someone leaving the rest of us would take up the mantle such as when our former programmer left, we took over his duties and completed the 3rd expansion pack. We have taken measures to make sure we are ahead of schedule by being done with most of our tasks approximately 3 days beforehand. In the case of someone getting kicked off for different reasons, we are able to complete their task within this period of 3 days.

| Risk Matrix (CDIO -1) | | Severity | | | | |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Likelihood | 1 | 1 | 2 | 3 | 4<br>- Data lost and cant be recovered. (Code, Documentation) | 5<br>- DTU går konkurs<br>- Software tool not work<br>- Data leak |
| | 2 | 2 | 4 | 6<br>- Dice game crashing | 8 | 10 |
| | 3 | 3 | 6<br>- People leaving project pre release<br>- Management change | 9<br>- Program lagging in the CLI | 12 | 15 |
| | 4 | 4 | 8<br>- Requirements change | 12 | 16 | 20 |
| | 5 | 5 | 10 | 15 | 20 | 25 |

## 5.6. Quality assurance

We created a test kit for the dice and have rigorously throughout the period of programming tested the code so we are sure that it works and has the desired outcome. For example if the packs do not mingle together we need to change the order in which the code has been structured so we get the desired outcome, if expansion 1 and 4 are active together as an example. (expansion 1 and 4 can both be seen in our use-case UC1 as Alternate Flow, Extra Rules 1 and 4.) Everytime we push and commit a change in GitHub, we make sure to have tested and run the code and that it has done the intended action as per that version. Examples of our criteria would be that the expansions work both individually and together with the other expansion packs and with the base game, such as expansion 4 and base game.

## 5.7. Budget

The only budget for this project would be our own time due to there being no direct monetary payment, other than the experience we gain from programming and learning to work together more efficiently towards creating a good product in the two weeks time we have had for the dice game.

## 5.8. Communication Plan

The plan for communication would be 2-3 weekly meetings and a debrief Monday morning after the first lecture so we have an idea of what progress there has been, and how we go forward towards the assignment date due on the 27th. of september. We used google docs as our report creator, and discord as our primary communication platform so we had less to save on our laptop, and more freedom in when/where we communicated.

### 5.9. Change Management:

In the case of a major change in scope or schedule, a reallocation of resources will be needed and has been done when our 5th team member decided it was not the way he would like to go. Hereby having us cover up for what he was assigned to do. That means we pushed back the schedule of finishing programming to the 27th of September hereby giving us an extra day on programming though a day less for quality control of our report. The changes have been discussed and approved by the group before committing to anything at a meeting we had on the 26th of September in Uni.

### 5.10 Monitoring and Control

Our monitoring and control will all happen over GitHub with the purpose of seeing who did what at what time and why they did it with a message that explains what thoughts and ideas that have been going through their mind as well as our 2-3 weekly meetings which make elaborating upon the thought process much easier since it was recent they made a change to the code or anything else for that matter. The tools used to upload and change code would be GIT BASH or GitHub desktop which uses GIT commands as well, just in a more friendly manner.

### 5.11. Closure Plan

Towards the end of the development process we will make sure to have 2 meetings one on Thursday the 28th of September and Friday the 29th of September on the due date before we hand in the code and report. Our final testing will be one for each situation, so one where we have the base game alone, one for each of the expansion packs, one for each combination of expansion packs, and one with all of the expansions active.

# 6. Requirements

The requirements for the system are in this section described in the form of a use case UC1: Play Game, as well as a list of additional requirements. These requirements come from a combination of the customer's written vision and conversation with the customer/project leader.

### 6.1. Use case UC1: Play Game

**Primary Actors:** Players

**Stakeholders and Interests:**

- Player: Wants a fun and smooth gaming experience without bugs or errors.
- Customer: Wants a system that fulfills their vision.
- IOOuterActive: Wants a fun and successful game for reputation and business reasons.

**Preconditions:** The game is successfully installed on the user's computer.

**Postconditions:** The game has concluded. Players had a smooth gaming experience. The players had no need to read an external guide to play the game.

**Basic Flow:**

1. Two Players open the game file.
2. A player throws two 6-sided dice.
3. The System displays the rolled dice within 333ms.
4. The sum of the dice is added to the player's score.

*Players take turns repeating 2-3 until one player's score reaches 40 or above.*

5. The player whose score first reaches 40 or above has won the game.
6. The game is closed.

**Alternative Flow:**

5a. The player whose score first reaches 40 or above had their first turn before the other player.

1. The player whose first turn came second gets a new turn, such that the two players end the game with the same total amount of turns taken.

1a. After the new turn, the game ends with both players having the same score.

1. The stalemate is resolved.

**Alternative flow, Extra rule 1:**

4a. The player's dice have rolled two 1's.

1. The player's score is set to 0 instead.

**Alternative Flow, Extra rule 2:**

4b. The player's dice have rolled two of the same number.

1. The current player gets an extra turn before the next player.

**Alternative Flow, Extra rule 3:**

4c. The player's dice have rolled two 6's on both this turn and their own previous turn.

1. The player wins the game.

**Alternative Flow, Extra rule 4:**

5b. The player reaches a score of 40 or above.

1. From now on, if the player's dice roll two of the same number, the player wins.

**Special Requirements:**

- Game must be playable on the Windows machines in the DTU Data bar.

### 6.1. Other Requirements

- The customer wants to see a test which shows that the results of the game's dice are consistent with the theoretical probabilities over a simulation of 1000 dice rolls.
- The customer wants the report and code comments to be in the same language of either Danish or English. If Danish is chosen, certain pieces of English terminology are still to be written in English.
- The customer has specified additional rules to optionally be implemented into the game. These are described as additional Alternative Flows in UC1. These rules, if implemented, must be done so separately to the version of the game without these rules.

# 7. Design

A dice game played in the CLI, on minimal hardware, 2 players can battle each other til one of the players reach 40 points and win.

### 7.1. Architecture

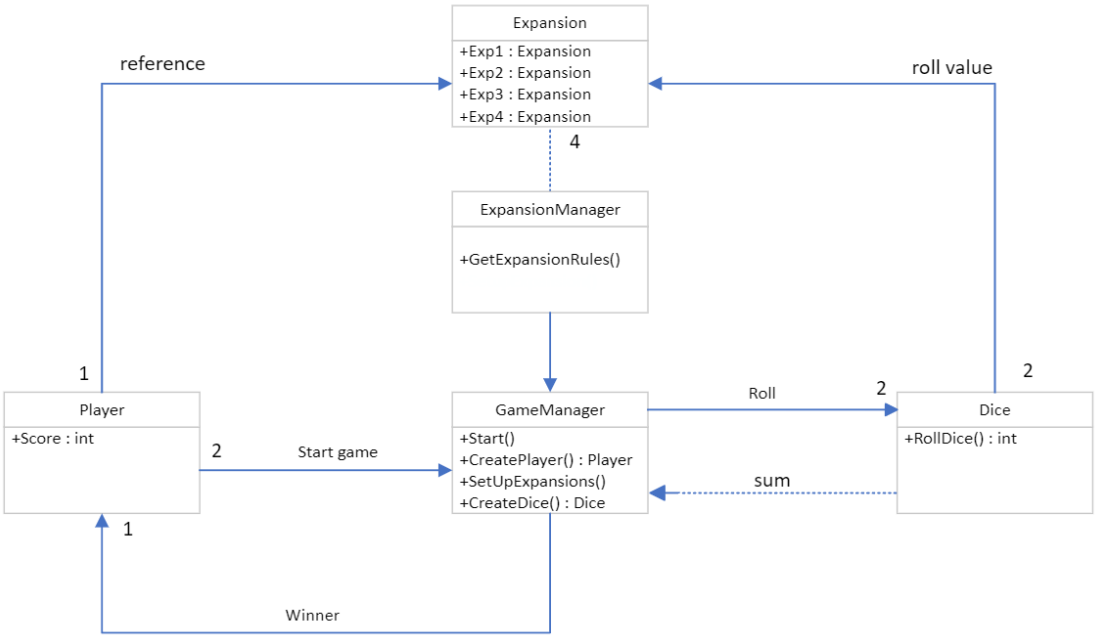
The core architecture of the game includes player interaction, dice rolling and rules. Players take turns to roll dice, scores get tracked by the sum of the 2 dice and rules(Expansions) can be toggled on and off before starting the game.

### 7.2. Rules/Expansions

The base game is the standard turn based dice roll game, but players have the opportunity to enable 1-4 new rules called Expansions. This gives the players control over the game, and interactive gameplay.

### 7.3. The Code

The game state is handled in the GameManager Class. Thus Players and Dice are created, expansions are chosen, and the game initializes. Each Player takes turns in the CLI calling the method "RollDice()" in the Dice class. First one to reach 40 points win.




# 8. Implementation

The implementation phase of our project involved translating our design concepts into actual code. We started out discussing the game structure and rules, in line with the vision that was presented, as well as interviewing our project leader to get a clearer picture of the requirements of the project.

### 8.1. Programming Languages and Technologies

We used Java as our programming language. With Java we could create an object oriented program, with freedom to create classes and objects to meet the vision and requirements. With the CLI we could utilize the Scanner class for output and input data.

### 8.2. Version Control

We utilized Git for version control, enabling collaborative development and code tracking. This way we could all do our part on the project and manage changes, branches. updates and merges.

### 8.3. Testing and Quality Assurance

We tested the dice game with a class that ran the project 1000 times during development, to see if the project behaved accordingly. This helped us identify issues early and throughout the development process.

# 9. Testing

### 9.1. Dice test

For this project, we were tasked with supplying a test of our digital dice, in order to make sure they were sufficiently random as to feel like real dice. This test was to consist of 1000 dice rolls, where the results were to be recorded and analyzed. It was specified that the percentage difference between the digital dice in our game and the statistically expected average results were to be no more than 1%.

#### 9.1.1. DiceTest.java

In our project folder, there is a file called DiceTest.java. This is the file we have used to make and execute tests of the dice functionality. When run, the file simulates 1000 rolls and shows a results page with which one may analyze the data. The results page looks something like this:

```
Test results from 1000 rolls:
1's rolled: 157 (5.973223% off expected result)
2's rolled: 172 (3.149606% off expected result)
3's rolled: 148 (11.864407% off expected result)
4's rolled: 163 (2.224469% off expected result)
5's rolled: 176 (5.447471% off expected result)
6's rolled: 184 (9.885932% off expected result)

Test failed
```

*Fig. 9.1.1.: Results page of DiceTest.java*

This page states the amount of rolls the test consisted of, as well as the frequency and percentage difference from the expected result of each face of the dice. At the end, the file decides whether the test was successful or not, based on whether any of the percentage differences exceed 1%. You may notice that they do, and the test in fig. 9.1.1. has concluded itself to have failed on that basis. This is explained in section **(9.1.2.)**.

### 9.1.2. The 1% threshold

As stated in the first paragraph of **(9.1. Dice test)** as well as in section **(6. Requirements)**, the test of 1000 rolls was to conclude that the results were within 1% of the expected results. In our tests of 1000 rolls, this is *very rare*. This, however, is not because our dice are faulty.

The 1% threshold does not give a great margin of error. The expected frequency of each face of the dice is:

$$\frac{1000\,rolls}{6\,faces} = 166.67\,\frac{rolls}{face}$$

1% of this is:

$$\frac{166.67\,rolls}{100} = 1.67\,rolls$$

What this means is this:

| **Frequency of a particular face** | **Percentage difference from 166.67** |
|---|---|
| 164 | 1.61% |
| 165 | 1.01% |
| 166 | 0.40% |
| 167 | 0.19% |
| 168 | 0.79% |
| 169 | 1.39% |

For the test to succeed, **every** face's frequency needs to be within 166 and 168. For a test of this scale, this is far from realistic. It is still randomness we're dealing with - every face being within 3 rolls of each other is almost more suspicious than if they weren't. Compare this to testing over 6 rolls, and having to have one of each face each test, every time. If it was truly random, this would be an exceptional occurrence.

We believe our dice is accurate despite failing the required test. This is substantiated by different tests. We are able to do the exact same test, but simply over a greater amount of rolls than the stated 1000. Doing this shows that the percentage difference between ours and the expected frequencies drops as the amount of rolls in the test, *n*, increases.
With *n=1,000*, seeing all faces within 1% of the expected frequencies was very rare.
With *n=100,000*, it happened every now and then
With *n=1,000,000*, the results were within 1% every time. The results page looked something like this:

```
Test results from 1000000 rolls:
1's rolled: 166420 (0.148110% off expected result)
2's rolled: 166729 (0.037393% off expected result)
3's rolled: 166873 (0.123723% off expected result)
4's rolled: 166664 (0.001600% off expected result)
5's rolled: 166535 (0.079031% off expected result)
6's rolled: 166779 (0.067377% off expected result)

Test succeeded
```

*Fig. 9.1.2.: Successful test of 1,000,000 rolls*

In conclusion, despite failing the required test of 1000 rolls, we believe our digital dice is sufficiently random, and that if any other digital dice consistently shows more evenly distributed results, we suspect less randomness is going on than in our dice.

### 9.2. Interactions between expansion packs

- Conflict between expansion 1 (player can lose his/hers points if they roll two 1's) and expansion 4 (player have to roll a pair to win after reaching 40 points)
    - **Solution:** We came to the conclusion that expansion 1 should prevail and it would be more interesting, if a player had a risk of losing all their points in the winning moment.
- Conflict between expansion 2 (player gets an extra turn if rolling a pair) and expansion 4 (player have to roll a pair to win after reaching 40 points)
    - **Solution:** Early tests showed that expansion 2 kept firing and the game was stuck in a loop with no winner after reaching 40 points. We solved the problem by ignoring expansion 2 after a player has reached 40 points.
- There were no conflicts with expansion 3.
- Conflict between Score management and Exp 4. The problem occurred because the player score was added before the rules were checked. This would result in the player winning instantly on a pair, if the sum of the pair is higher then score requirement"“end game” statement. Due to the structure of the code, we were unable to add the score after the expansion rules were checked, as this would cause issues with the rest of the expansions.
    - **Solution:** The solution was to add a new boolean ‘scoreMetRequirement’, that was set to true, the next time the player rolled the dice. This means that the rule will always be applied at the next turn of the player after he has met the score requirement.

# 10. Conclusion

Throughout the requirement phase, we reached a sufficient level of understanding, how we should structure and proceed with the project. We sat down together and discussed the use-cases, objects, methods, and associations while sketching a diagram that would later be the model we would follow to create the game. Going into the elaboration phase we created our repository utilizing Git. Thus we could establish a work environment where each of us could contribute to the project with our own branches. The core architecture was created, including a GameManger, Player, and Dice class. A DiceTest class was set in place for testing each iteration. The iterations were structured in 3-4 branches, that was based on the rules presented in the vision. We wanted to give freedom and interaction for the players, to easily activate and deactivate rules as they wish. This also gave us a better testing environment, and control. The Expansion class was created, and implemented into the GameManager class as well as the ColorUtils class to signal the choices the players made.

We are satisfied with the product and we believe we have met the requirements.

Thanks for playing.