02314 62531 62532

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

CDIO 1

Søren Kjær
S235774

Hjalte Bischoff
S235770

Marcus Lemser
S230985

Gustav Mogensen
S235783

Bjørn Andersen
S231289

29 september 2023

Contents:

Project overview

Vision

We seek to bring joy and excitement to students and professors of DTU, with an easily accessible game, involving throwing dice with an abundance of fun features regarding the result of the thrown dice. Our Dungeon Dice Game is unparalleled in its easy usability, incredible response time, accurate modeling of dice probability. and the amount of fun it will bring to the player.

Requirements

URPS

Usability

- Two people can play a game of dice
- Rules must be stated in game

Reliability

- Has to roll at least 10000 times without failing to roll or failing to give correct results, and with a statistical accuracy of above 96 %.

Performance

- Time constraint of 333 millisekunder on die result *(sum or individual).

Supportability

- The program is to be usable on the computers in the data-bars (Windows) on DTU.
- The computer must support a Java 17 version or newer.

MoSCoW

Must have

- Allows 2 players to play dice game according to basic rules (stated in *x*)
- Rules are stated clearly in game

Should have

- Possibility of running test program
- Response time of less than 333 milliseconds

Could have

- Extra features as stated 1) - 4) in the CDIO1 exercise paper

Won't have

- Extra options for player such as “Restart” for instance

Actors and stakeholders

Stakeholders

Project Manager

- Wants a viable product to be delivered to the customer
- Wants the product delivered within the given deadline

Customer

- Wants a game with a defined response time
- Wants a game that can be used with no prior knowledge
- Wants a game which accurately mirrors dice probabilities

Player

- Wants a game that is easy to use
- Wants an entertaining game

Actors

Player *(primary)*

- Play a game of dice
- Be able to roll dices and get shown results

Customer *(secondary)*

- Must be able to run a test program

Use Case

Brief use cases

Test dice: Customer runs program "testProgram.java". The program asks for the desired number of throws to be tested. The program generates a simulation of how the game displays the dice roll, and the time it takes to display this in the game. The program displays a table of all the occurrences (in percentage) of the different dice sums, and compares it (in percent) to the theoretical statistical outcome of 2 6-sided dice rolls. The program displays the number of rolls and the accuracy of the total number of rolls.

Show points: The game displays the total points of both player 1 and player 2. The game then asks for another command.

Enter name: The game demands input from both player 1 and player 2, and sets the name to the respective player.

Exit game: The game displays a thank you message, and closes the program.

Verify win: The game checks the players points and the dice face value against win conditions, and triggers "*Win game*" use case, if conditions are met.
Alternate: Conditions are not met, and the game continues.

Win game: The game displays a "Player wins" message, and the program stops.

Add points: The game calculates the sum of dice values, and adds the amount to player points.

Roll cup: The game simulates two die rolls and adds the sum to the total amount of points of the player.
Alternate: Two 1's are rolled, and the player loses all points.
Alternate: Doubles are rolled between 2 and 5, the sum is added to total points, and the player receives another turn.
Alternate: Two 6's are rolled, the sum is added to total points, and the player receives another turn. The player rolls two 6's again, and wins the game.

Display points: The game displays the value of each die, and the total amount of points of the player.

Use case UC1: Dice Game

Scope: Playable game.
Level: User goal.
Primary actor: Players.
Stakeholder and interests:
Player(s)
Customer

Preconditions:

- Player(s) has access to DTU.
- Computer in "DATABAR" is fully functional.

Success Guarantee(Post conditions):

- Game is won by one of the players.

Main Success Scenario (Basic Flow):

1. Player launches DiceGame.
2. Rules are presented to Players.
3. Player enters name for Player 1.
4. Player 2 enters name for Player 2.
5. Player 1 Roll Dice cup.
6. Results of Roll is presented.
7. Roll value is added to Player 1's points.
8. Player 2 Rolls Dice cup.
9. Results are presented.
10. Roll value is added to Player 2's points.
11. Repeats until one Player reaches 40 points or more, and rolls a double.
12. Game displays winner.
13. Game ends.

Extensions or alternative flow

5/8a. Systems displays possible commands: “Roll”, “Show points”, “Exit”

5/8b. System receives invalid input as command
1. System display an error message indicating invalid command
2. Player enters new command

5/8c. Player makes a successful roll and rolls double 1’s.
1. Player loses all points and turn ends.

5/8d. Player makes a successful roll and rolls double between (including) 2 and 5.
1. Proceed to step 6/9 and 7/10.
2. Step 5 is repeated.

5/8e. Player makes a successful roll with double 6’s
1. Proceed to step 6/9 and 7/10.
2. Step 5 is repeated.
3. Player rolls double 6’s
4. Player wins the game
5. Proceed to step 12

5/8f. Player chooses “Show points” command
1. System displays points of both players
2. Player writes a new command

5/8g. Player chooses “Exit” command
1. System display thank you messages
2. Game and program closes down.

7/10h. Player reaches 40 points
1. System displays message that player has reached 40 point and now have to roll a double to win.

Requirements:
- Person playing must have DTU login to access computers in databar
- Result of dice roll must be displayed within 333 milliseconds
- All rules must be written in the game

Code conversion

We compiled the .java files in java-13, instead of java-20, to support the requirement of being able to run the game file in the Data-bar (which only support java 13 or below). This consisted of changing switch statements to if statements.

To make the game operate on DTU's Databar for all people, we made two .bat files. These files makes it easier for the user to execute the Dicegame.
The files are called:
runfullgame.bat
runtest.bat

And must be in same directory as the .class files.

GitHub Repo

https://github.com/SyndicaterSCI/05_del1

Workflow

Intro

We have been tasked with the development and inception of a Dice game for two players. Our first course of action was to interpret the wishes and requirements of the customer. We made a ‘translation’ of the requirements, and presented it to the project leader. The translation was approved.

Prototype

We set out to create a proof of concept. The concept was, could we make two dice, simulate a roll and calculate the sum of the dice. We made a simple dice program that could generate two numbers between 1 and 6 and. We also made a prototype of the game that could ‘call’ the dice program and calculate the sum.

Diagrams

We created ‘Use Case Diagram’, ‘Domain model’, ‘Basic Domain model’ and ‘Sequence Diagram’. Before we committed to the full development of the code, we wanted to show the team the overview of the flow.

Github

A github project was created. The project was separated in ‘Development’, ‘Main’, ‘Fullgame' and ‘Maingamefunctions’. The branching strategy was “Go with the flow”. The team has since learned that this strategy is not viable for bigger projects.

Coding

We wrote the full program in “Pseudo code”. Afterwards different parts of the program were assigned to different members of the team.
We set out to create the basic version of the game, without the extra rules. The program is separated into a ‘Dice’, a ‘Player’ and a ‘Game’ class. After completion of these three, the team set out to create the expanded game with all the extra features.

Requirements

Requirements specification

Questions and suggestions to customer:
Below are a series of questions along with our suggestions concerning aspects of the customer demands which are not clearly stated, or demands which are in conflict with each other.

Concerning game:
Q: Asks for a game with a raffle cup. Is there an expectation of animations?
S: Not in our capabilities. A simple window, with input and output in form of text.

Q: Which type of dice?
S: We suggest 6-sided dice

Q: When rafflin, does the die show an individual face or a sum of both die?
S: Show both sum and individual face to make it easier to implement further features with double face.

Q: Does the player win when reaching exactly 40 points or equal and/or above?
S: Easy solution, is “equal or above”. If “exactly”, then we need further rules for what happens when above.

Q: What is the specificity of “usable without manual”?
S: We suggest the game states the rules at the beginning of a new game, and asks the player clearly for input needed, so there is no need to read anything else before starting the game.

Q: What is meant by “terminology must be natural”?
S: We suggest writing documentation (in code and in diagrams/reports) which uses language understandable for the layman.

Concerning extra features:
Q: Extra feature “2.” conflicts with “1.”
S: Player loses all point with double 1’s. All other doubles result in an extra throw.

Q: Should extra throws be able to trigger indefinitely?
S: As long as a player throws doubles (except 1’s), this results in extra throw.

Q: Extra feature “3.”, states that player can win game if thrown double 6’s two times either because of extra throw or from previous turn. It seems it can only be from extra throw. Anything we missed? Please specify conditions
S: Player only wins if double 6’s are thrown 2 times in a row, in the same turn.

Q: Extra feature “4.”: will the player still win with double 1’s? Equal 40 or above and/or equal?
S: Double 1’s still results in resetting points. above and/or equal

Concerning setup:

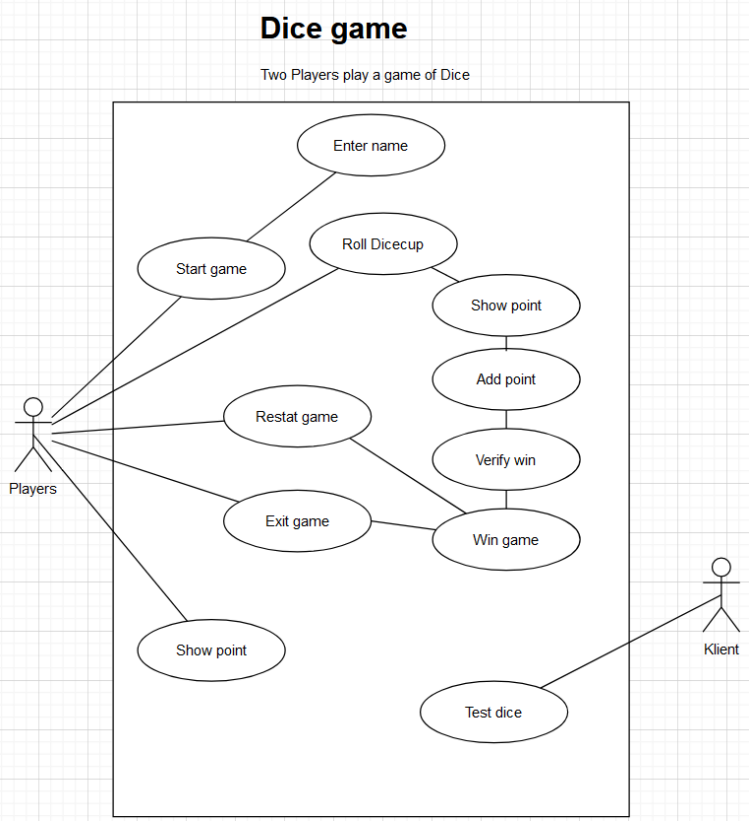
Q: Call project “05_del1”. Is this our main class? Or the folder?

Q: Package with game and test: is it separate folders, or .java files, you’re referring to?
Det skal ligge i samme folder

Q: Not good to use “default packages”: What is meant by this?

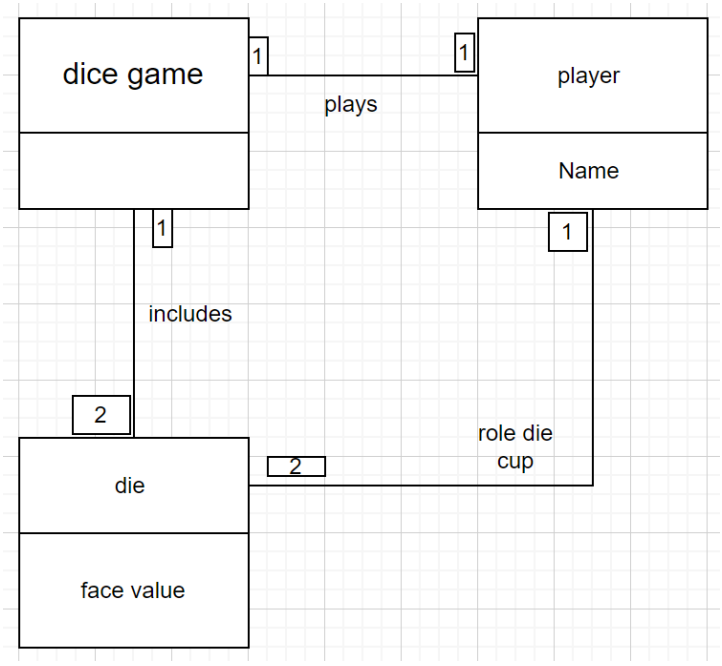
Documentation

Use Case diagram



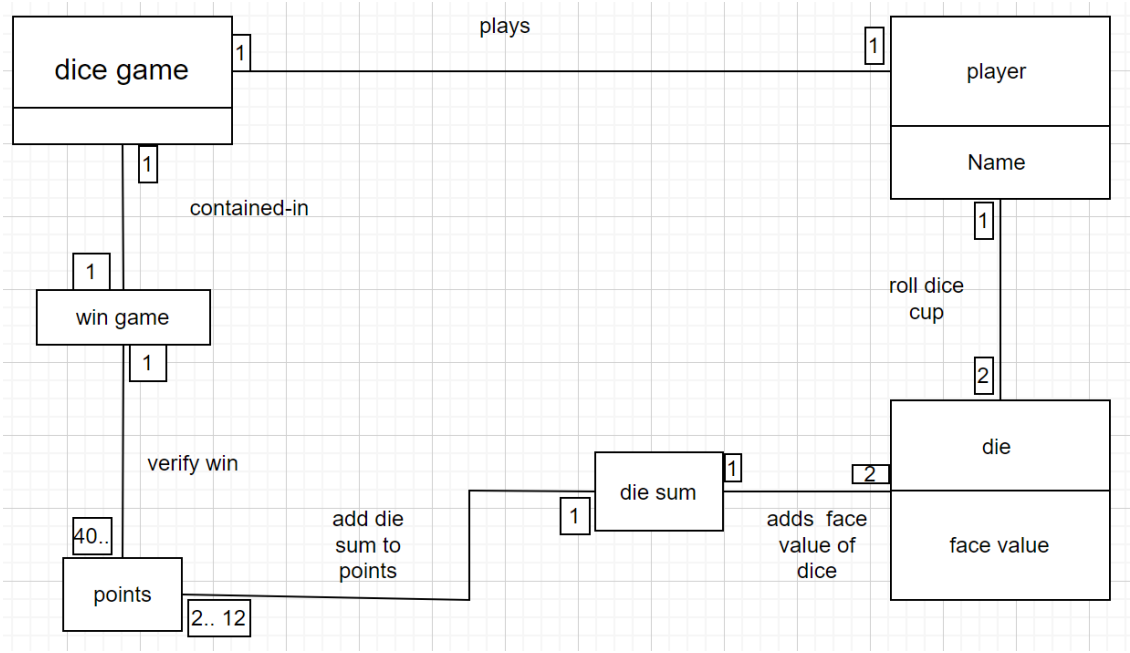
Above is the use case diagram for Dice Game, used to give the customer an overview of the different possibilities of the software. We see the player being associated to a variety of use cases, and the association between different use cases.
The (very important) test program, will be delivered as a separate file than the DiceGame, but make use of the same Die.java class.

Basic domain model



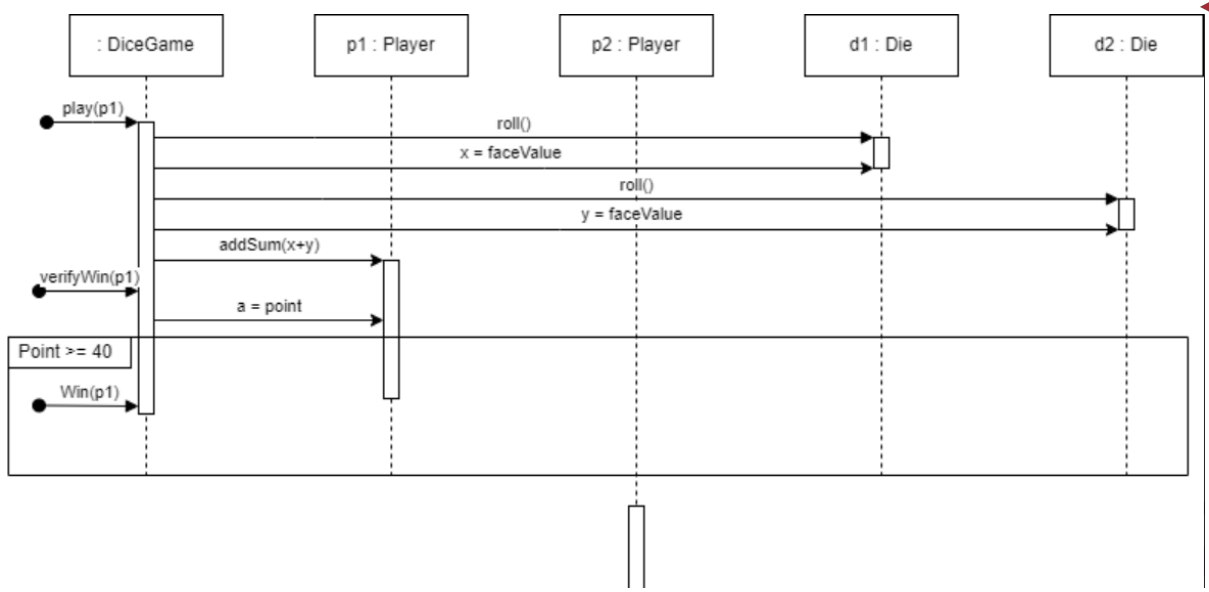
Here we see a basic version of the domain model, to give an initial idea of which classes should be created, and their interactions.

Domain model



Here we see a more detailed domain model, which gives a clear image of the architecture (classes and association) of the code in game, without extra features.

Sequence diagram (Roll cup)



Here we see the interactions between classes (DiceGame, Player, Die), and the methods called, as in a section of the Roll cup use case.

Code Comments

Our code has been documented with comments, concerning different uses and construction of classes and methods. The code has been commented, as well as possible, in layman terms, so it becomes accessible to the customer.

Proof of concept

Below are screen shots of "*Game.java*" and "*Die.java*", and an example of outputs in terminal of program running. This is used to provide the customer with a proof-of-concept, that it is possible to simulate a roll of 2 die, add the two sums, and display it immediately

```java
class Game {

    public static void main(String[] args) {

    var s = new java.util.Scanner(System.in);

    var die1 = new Die();
    var die2 = new Die();
    System.out.println("Welcome, player. Type 'roll' to make your move, mutherfuckar");
    while (s.hasNextLine()) {
        var roll = s.nextLine();
        switch(roll) {
            case "roll" -> {
                var value1 = die1.getValue();
                var value2 = die2.getValue();
                var sum = value1 + value2;
                System.out.println("Value of first die is " + value1 + "\nValue of second die is " + value2 + "\nThe sum of both dice is "
            }
            default -> {
                System.out.println("Are you stupid?...\nIf you wanna make another roll, type 'roll'");
            }
        }
    }

    s.close();

    }
}


```

```java
import java.util.Random;

class Die {
    public int faceValue;

    // Generate a random number between 1 and 6, and sets the variable "faceValue" to that number.
    // It simulates a dice roll.
    public void roll(){
        int min = 1;
        int max = 6;
        Random rand = new Random();
        faceValue = rand.nextInt((max - min) + 1) + min;
    }

    // "Rolls" the die and returns the number of the face value of the die, as an integer.
    public int getValue(){
        roll();
        return faceValue;
    }
}
```

```
lemse@LAPTOP-728N4TRC MINGW64 ~/Desktop/DTU/1 semester/CDIO/Del 1/05_del1/Protot
ype (main)
$ java Game
Welcome, player. Type 'roll' to make your move, mutherfuckar
roll
Value of first die is 4
Value of second die is 4
The sum of both dice is 8
roll
Value of first die is 4
Value of second die is 2
The sum of both dice is 6
roll
Value of first die is 3
Value of second die is 1
The sum of both dice is 4
sf
Are you stupid?...
If you wanna make another roll, type 'roll'
```