Danmarks Tekniske Universitet

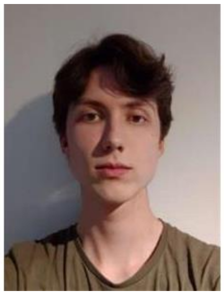
CDIO 2

(62531) Development Methods for IT-systems,
(62532) Version Control and Test Methods,
(02314) Introduction to Programming

Balder Jacobsen, s235094

Kasper Ehlert, s235073

Viktor Steffensen, s214707



Tobias N. Frederiksen, s235086

27. oktober 2023

1 Summary

In this report, we will argue why our way of coding and planning the code, lives up to our customers needs and the basics of good code etiquette. All the demands were not specific enough, and some contradicted eachother, but asking the customer the right questions and taking, the customer asks for specific classes and for tests that we consider relevant to our program. Through Object Oriented Design, we make sure our classes lives up to low coupling and high cohesion, and through tests we make sure the program runs as demanded by the costumer and expected by us.

2 Hourly accounting

Every member of the group has spent approximately 10 hours on the code and report combined. All groupmemebers have comitted to the git-repo, and have added relevant text and/or models to the report.

3 Table of Contents

4 Introduction

IOOuteractive needs our expertise to create another game for them. The game si for two players and is based around random computergenerated dice. Each player starts with 1000 points, and the goal is to get to 3000. The players take turns rolling dice, gets the sum of the dice, which then determines the field, that they land on. Based on the field the players will either lose or gain points.

5 Requirement analysis

We began our requirement analysis by finding and clarifying all the requirements we got from the costumer.

- The game is a dicegame.
- The game is played by 2 players who both take turns rolling dice.
- The dice are two 6-sides fair dice - it has to be easy to change the sides of the dice through our code.
- Both players have a pointpool - they start at 1000, and the game ends when a player reaches 3000 points - this player is the winner.
- The players and the points obtained must be able to interact with other programs.
- The sum of the diceroll determines what happens to the player.
- We have to use the same dice we used for the CDIO 1 assignment.

Demands for versioncontrol:

- IOOuteractive wants to be able to further develop the program and to be able to see who did what - they need out git-repo.
  - Commit often - atleast everytime a small portion of code is made or enhanced.
  - Every commit needs a small description of what has been made/changed.
  - Each commit has to only contain relevant changes. (If you fix a bug AND a spelling mistake - make two seperate commits with comments in both.)
  - Only code that works is to be commited into main.
- The time limit from input to output must not exceed .33 seconds.
- It has to be easily translated.

Demands for specific tests, methods and diagrams:

- Demandsspecifikation
- Use case diagram
- Use case despricitions (Brief and fully flexed)
- Domain model
- Systsem sequence diagram (only the most important operations.)
- A test thats shows that a players balance can never be negative, no matter the input.
- Other relevant tests.

Configuration:

IOOuteractive wants to know what demands there are to installed programs and OS. They need a description of minimumrequirements and a tutorial in how the sourcecode is compiled, installed and removed. This includes a description of how the code is imported from a git-repo.

6 Project planning

We began our project planning, after finishing our requirement analysis of our project. From this we could find our Actors and Stakeholders.
We chose to do this as one of the first things, to make sure we were focusing on the most important requirements first.

6.1 Actors

We have identified the actors in the program as:

- Player$_1$ (DTU Student)
- Player$_2$ (DTU Student)
- PC (Running)
- Keyboard (Input)
- Monitor (Visual output)
- Database/Savefile (Saving game data)

6.2 Stakeholders

We have identified the stakeholers in this project as:

- IOOuterActive
- Project leader
- Developers
- Costumer
- Players
- DTU

6.3 Usecases

Main succes scenario:
The players wirte their names. They choose which language they want to play in and then take turns to roll the dice. After each roll the players land on a field where they either get or lose points. Each player begins the game with 1000 points, and the winnder is the first player to get 3000 points.

Alternate scenarios: The players decide they no longer want to play the game, so they exit without saving the game.

The players deide they want to stop playing for now, and may want to resume the game at some other point, so they save the game. They end up starting a new game not using the saved data.

Use case: Save data and resume play

Scope	IOOuterActive dicegame
Level	User goal
Primary actor	Players
Stakeholders and Interests	- Players: Want to play the game. - IOOuterActive: Wants the game to fulfill requirements. - DTU: Wants students to have activities during breaks.
Preconditions	Players must have a computer capable of running the program.
Succes Guarantee	Game is saved and correctly loaded again.
Main Succes Scenario	1. The game is launched. 2. Players select a language. 3. Players enter their name. 4. Game displays whos turn it is. 5. Player types 'r' to roll the dice on his/her turn. 6. The field the player landed on is described, the consquence of the field shown, and the new balance is displayed. 7. Unless one consequence affects whos turn it is, the turn goes to the next player. 8. Players decide to quit and save the game. 9. Player types 's' to save the data. 10. Player types 'q' to close game. 11. Players open the game again and loads saved data. 12. Players resume playing taking turns. 13. When a player reaches 3000 points or above, this player wins, and the game ends.
Frequency of Occurance	Could be nearly continuous.

6.4 ////////////////////

With all of our actors and stakeholders identified we created a couple of Interest and Influence diagrams. We did this to clarify which of our actors we would need to work the cloest together with and who we just would need to keep informed. The diagrams we created are to get a better visual representation of this. In the first diagram, we have four different areas, "Monitor" "Keep Satisfied" "Keep informed" "Manage Closely", We then placed each of the stakeholders on this diagram the higher they are the more influence the stakeholders have, and the more to the right they are the more interrest they have. So if a stakeholder is placed in the top left corner, they will have a lot of influence over the project, but their interest is very low, so we will just need to keep them satisfied. Compared to if a stakeholder is placed in the bottom right corner, they hold a lot of interrest in the project but the don't have any influence over it.

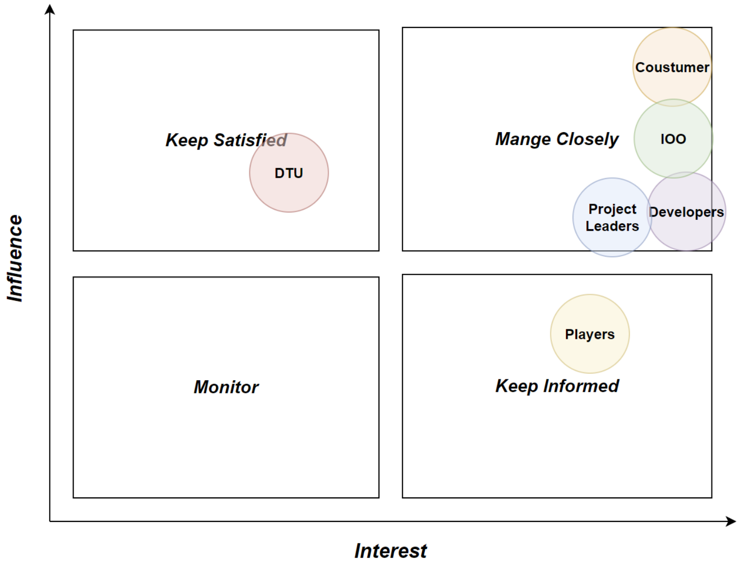


Figure 1: Influence Intrest Diagram no. 1

This second diagram helps us to show which of the stakeholders are in control of the project. So if a stakeholder is placed in the middle, they are in control, and the further out you go the less power over the project the stakeholders hold.

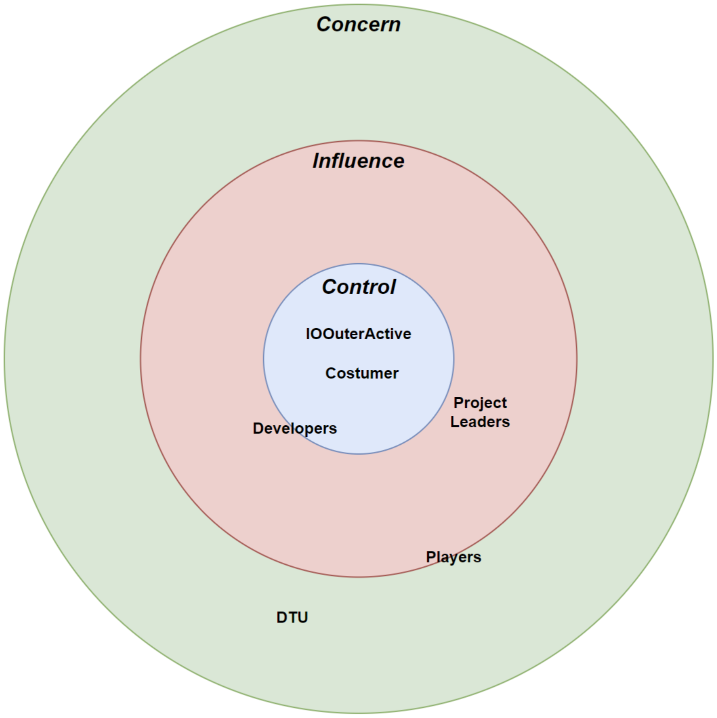


Figure 2: Influence Intrest Diagram no. 2

7 Design

7.1 Class diagram

The design of the program is displayed in the class diagram in the following. It shows the different classes, and makes it easy to see the relations between them. We made sure to make the classes with the least amount of coupling we could, while still having enough classes to maintain individual responsibility, which can be seen by how the classes are only dependent on one other class.

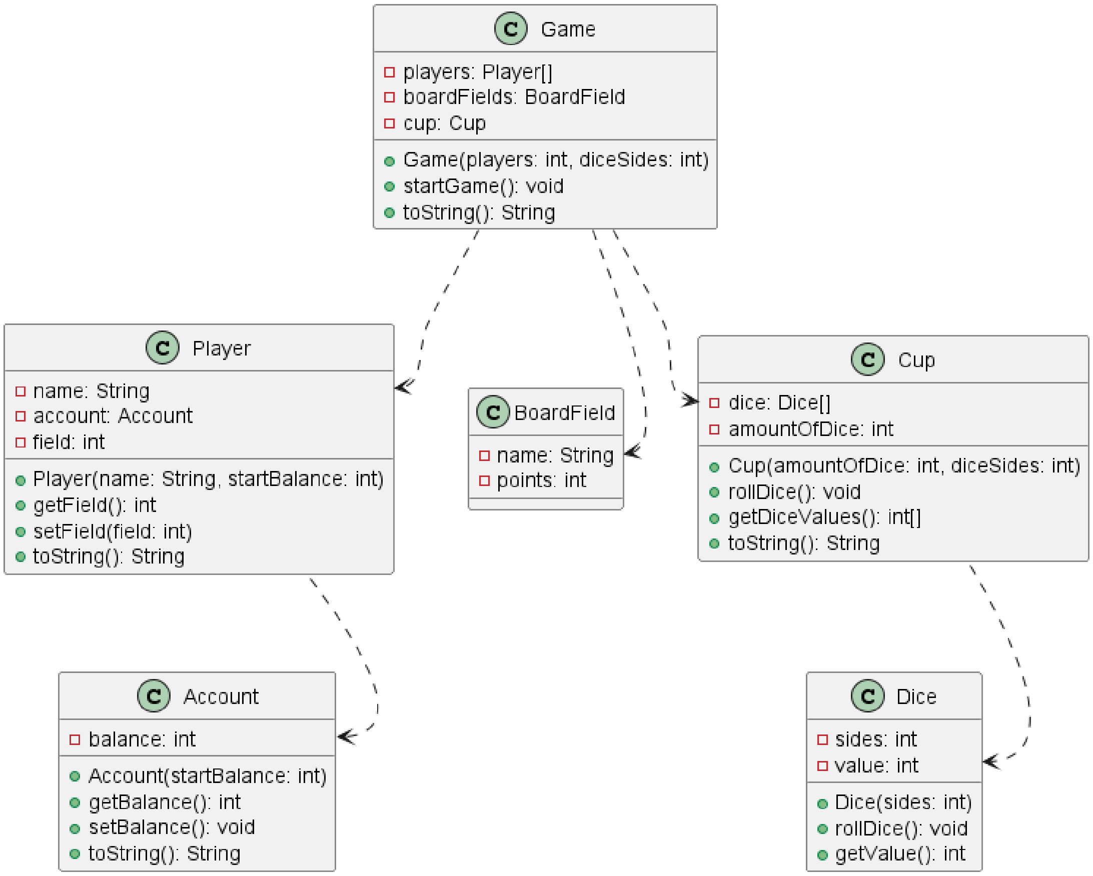


Figure 3: Class diagram

7.2 Domain model

Our domaain model shows are most important concepts and their attributes and the relationships between the concepts.

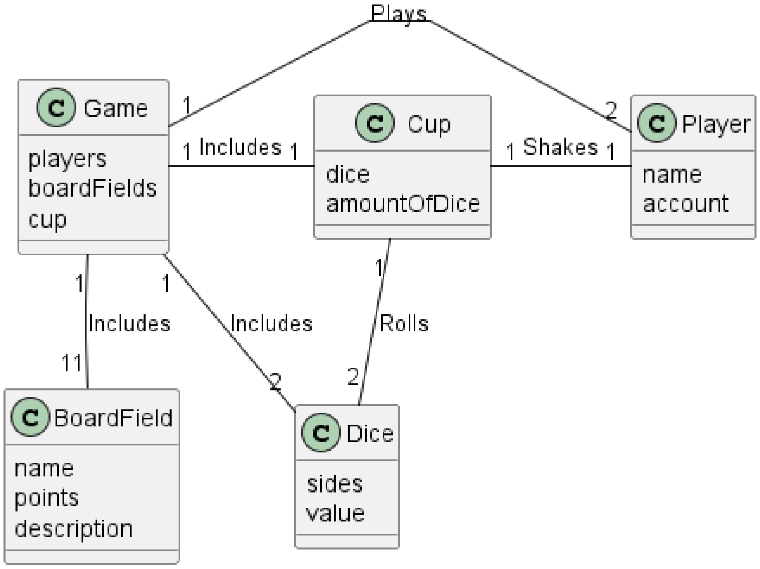


Figure 4: Domain model

7.3 System sequence diagram

The system sequence diagram shows the order of actions in the game. It can be seen which classes and actors do what at a given time during the game. Since the players take turns rolling the dice most of the actions are looped.

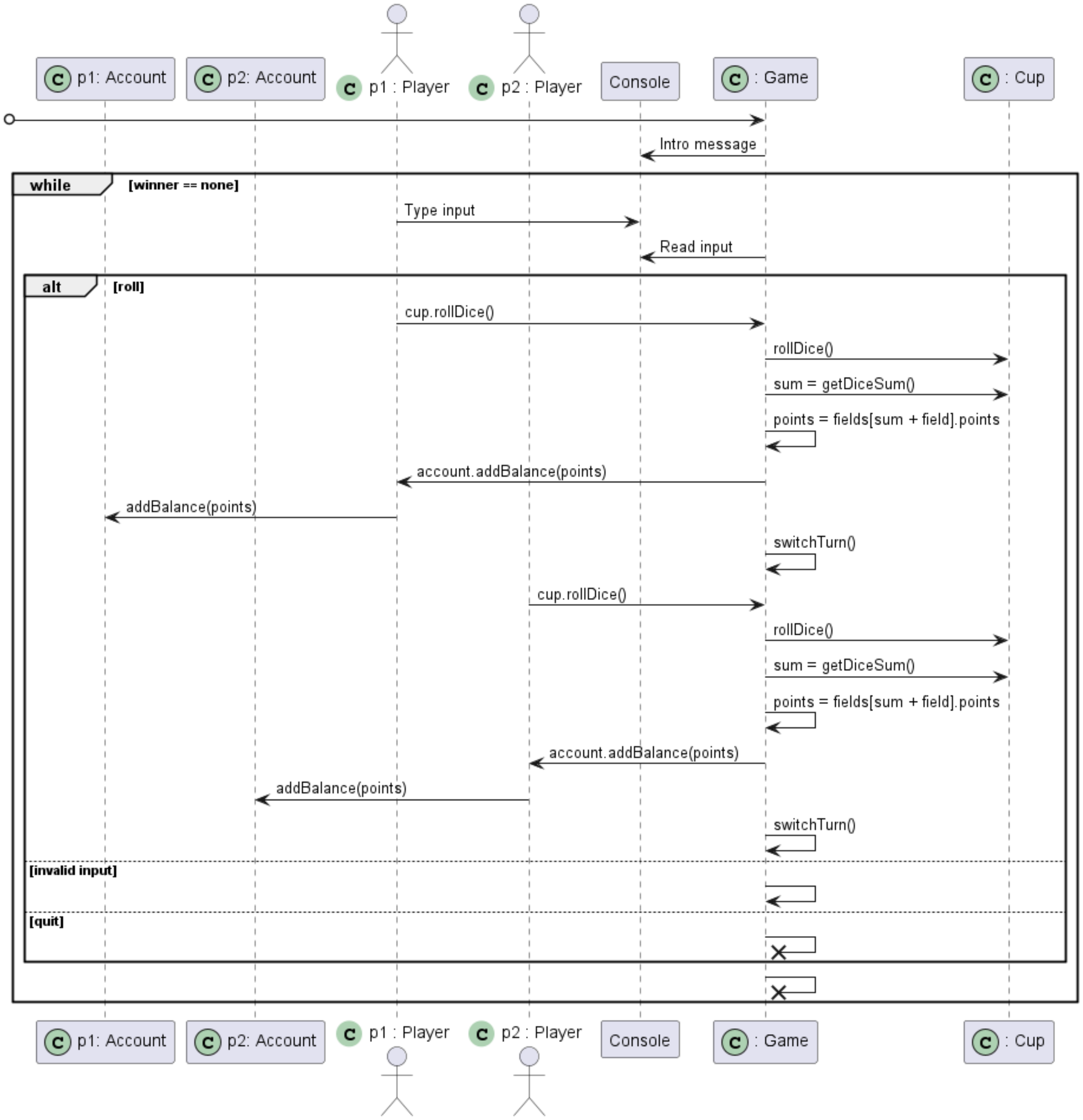
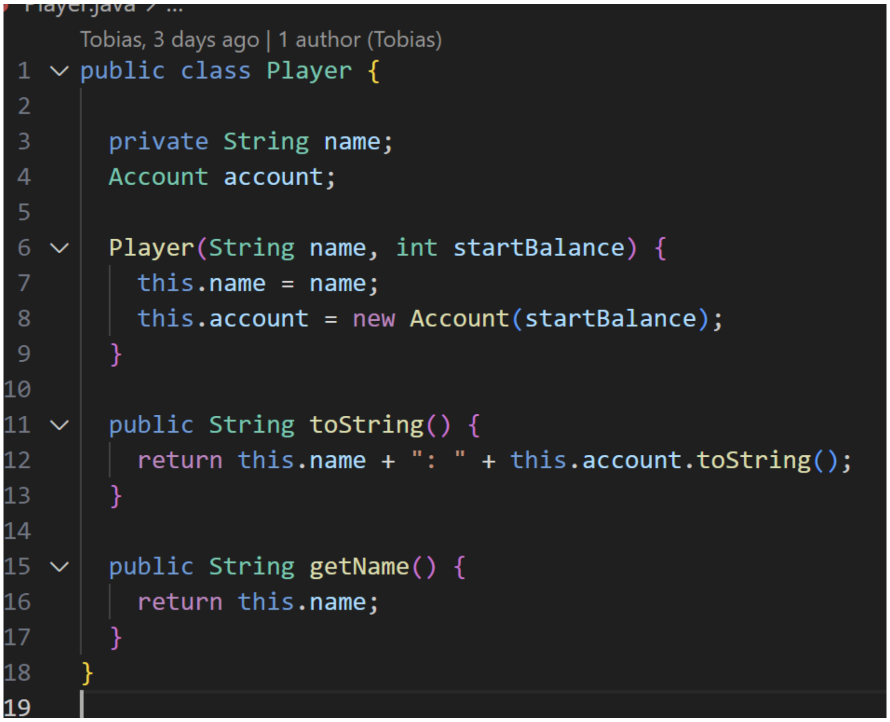


Figure 5: System sequence diagram

8 Implementation

As per IOOuteractives request, we designed our code to be agile. We did this so that, if the need should arrise IOOuteractive can add more; Players, dice, sides to the dice, board fields and even launguages. This can forexample be seen in the player class.



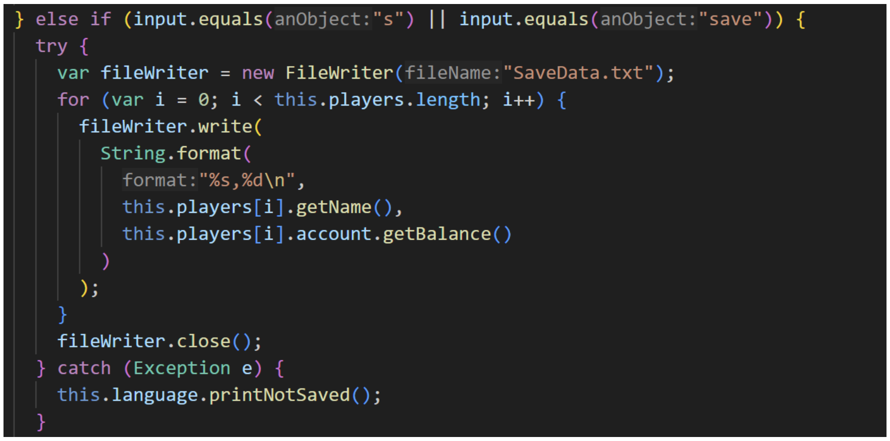
Player.java > ...

Tobias, 3 days ago | 1 author (Tobias)

```java
public class Player {

    private String name;
    Account account;

    Player(String name, int startBalance) {
        this.name = name;
        this.account = new Account(startBalance);
    }

    public String toString() {
        return this.name + ": " + this.account.toString();
    }

    public String getName() {
        return this.name;
    }
}

```

Figure 6: *Screencap of our code to o the player class*

To make the program easier for users to use, we also implemented a save and load system. During the game, the users of the code can choose to save the game data, this includes each of the players names and their ”gold”. The save system utilizes java’s fileWriter to save the data, to a .txt file.



```
} else if (input.equals(anObject:"s") || input.equals(anObject:"save")) {
  try {
    var fileWriter = new FileWriter(fileName:"SaveData.txt");
    for (var i = 0; i < this.players.length; i++) {
      fileWriter.write(
        String.format(
          format:"%s,%d\n",
          this.players[i].getName(),
          this.players[i].account.getBalance()
        )
      );
    }
    fileWriter.close();
  } catch (Exception e) {
    this.language.printNotSaved();
  }
```

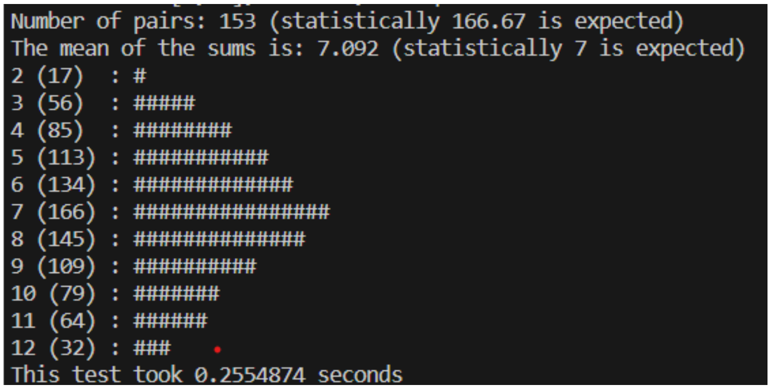
Figure 7: *Screencap of our code - This code snippet shows the main function of our save system.*

If the players save their gamedata, they can choose to load it if they start a new game, they can also choose not to do so, if a new game is started and game data is saved the old data will be overwritten.

9 Testing

9.1 Cuptest

In this test we simulated 1000 rolls with two dice to make sure the outcome corresponds with what is statistacally expexted. The results can be seen here. The code used for the CupTest was taken from our last project[1]

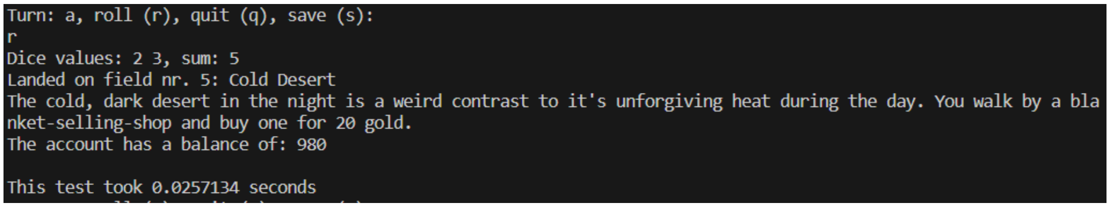


```
Number of pairs: 153 (statistically 166.67 is expected)
The mean of the sums is: 7.092 (statistically 7 is expected)
2 (17)  : #
3 (56)  : #####
4 (85)  : ########
5 (113) : ###########
6 (134) : #############
7 (166) : ################
8 (145) : ##############
9 (109) : ##########
10 (79) : #######
11 (64) : ######
12 (32) : ###
This test took 0.2554874 seconds
```

Figure 8: Cuptest results for 1000 rolls

9.2 Response time

We implemtented a timer in our code to make sure the response time doesn't take too long. We found that the time takes between 0.023 seconds to 0.037 seconds, which is very good.



```
Turn: a, roll (r), quit (q), save (s):
r
Dice values: 2 3, sum: 5
Landed on field nr. 5: Cold Desert
The cold, dark desert in the night is a weird contrast to it's unforgiving heat during the day. You walk by a bla
nket-selling-shop and buy one for 20 gold.
The account has a balance of: 980

This test took 0.0257134 seconds
```

Figure 9: Response time for rolling dice

[1] G13_del1

10 Conclusion

In this project we decided to go a bit out of our comfort zones, regarding the code. We decided early on that we wanted to be able to save the data from one instance of the game, so we would be able to use it in another instance of the game. And if there was time we would try and implement a GUI element to the game. We managed to get the save data feature working; we then decided to focus on the polish of the game, instead of implementing the GUI. We believe that this was the right decision.

Overall, this project was a success, we got all of the customers key features implemented, and working as intended. We had a great synergy in the group and we all lifted equally, and supported each other when needed. Our team was important in this project to get us to achieve our objectives and enhance the overall success of the project.

We can also confirm that the product was a success. We can do this by looking at the required features, and what we have implemented, these include: . This can also be backed up by some of the small tests we ran throughout the project. As requested we designed the code to be modular, so the functions in the code can be used in other projects.

11 Appendix

11.1 Literature

- **G13_del1**
  Dice-class and CupTest-class are copied and slightly modified from this assignment.

11.2 Code

We have sepereated our code into different files dependig on what part of the game they were used for. For instance we have a "Cup" file, which generates the dice and more, and we have a "Dice" file which uses the dice to get values.

11.2.1 Account.java

```java
public class Account {

  private int balance;

  Account(int startBalance) {
    this.balance = startBalance;
  }

  public int getBalance() {
    return balance;
  }

  public void setBalance(int balance) {
    if (balance < 0) {
      this.balance = 0;
    } else {
      this.balance = balance;
    }
  }

  public String toString(String message) {
    return message + this.balance;
  }
}
```

11.2.2 BoardField.java

```java
public class BoardField {

  private String name;
  private int points;
  private String description;

  public BoardField(String name, int points, String description) {
    this.name = name;
    this.points = points;
    this.description = description;
  }

  public String getName() {
    return this.name;
  }

  public int getPoints() {
    return this.points;
  }

  public String getDescription() {
    return this.description;
  }
}
```

11.2.3 Cup.java

```java
import java.util.Vector;

/**
 * The {@code Cup} class simulates the cup typically used in board games to roll
 * multiple dice.
 */
public class Cup {
    private Vector<Dice> dice = new Vector<Dice>();
    private int amountOfDice;

    /**
     * Constructs a {@code Cup} object that contains a specified amount of
     * {@code Dice} objects
     *
     * @param amountOfDice integar value of the number of dice in the cup
     */
    public Cup(int amountOfDice, int diceSides) {
        if (amountOfDice < 0 || diceSides < 0) throw new IllegalArgumentException("You cannot have
        this.amountOfDice = amountOfDice;
        for (var i = 0; i < this.amountOfDice; i++) {
            var dice = new Dice(diceSides);
            this.dice.add(dice);
        }
    }

    /**
     * Rolls all the dice in the cup
     */
    public void rollDice() {
        for (var i = 0; i < this.amountOfDice; i++) {
            this.dice.get(i).rollDice();
        }

    }

    /**
     * @return an array of integars that contain the value of each dice in the cup
     */
    public int[] getDiceValues() {
        int[] values = new int[this.amountOfDice];
        for (var i = 0; i < this.amountOfDice; i++) {
            values[i] = this.dice.get(i).getValue();
        }
        return values;
    }

    /**
     * @return the sum of all dice
     */
    public int getDiceSum() {
        int sum = 0;
        for (var i = 0; i < this.amountOfDice; i++) {
            sum += this.dice.get(i).getValue();
        }
        return sum;
    }

    /**
     * @return if the values of the dice are the same
     */
    public boolean isDiceEqual() {
        for (var i = 1; i < this.amountOfDice; i++) {
            if (this.dice.get(i - 1).getValue() != this.dice.get(i).getValue()) {
                return false;
            }
        }
        return true;
    }
}
```

11.2.4 CupTest.java

```java
import java.util.Arrays;

/**
 * Tests to make sure dice are fair and statistically accurate. Simulates 1000 rolls and counts pa
 */
public class CupTest {
    public static void main(String[] args) {
        var cup = new Cup(2, 6);
        var pairs = 0;
        int[] sums = new int[11];
        double sum = 0;
        double start = System.nanoTime();
        for (var i = 0; i < 1000; i++) {
            cup.rollDice();
            sum = sum + cup.getDiceSum();
            sums[cup.getDiceSum() - 2] += 1;
            if (cup.isDiceEqual()) {
                pairs = pairs + 1;
            }
            System.out.println(String.format("Dice values: %s, sum: %d, is equal: %s",
                    Arrays.toString(cup.getDiceValues()), cup.getDiceSum(), String.valueOf(cup.isD
        }
        double end = System.nanoTime();
        System.out.println("Number of pairs: " + pairs + " (statistically 166.67 is expected)");
        System.out.println("The mean of the sums is: " + sum / 1000 + " (statistically 7 is expect

        // Visualises the distribution of occurences of sums
        for (var i = 0; i < sums.length; i++) {
            System.out.print((i + 2) + " (" + sums[i] + ")\t: ");
            for (var j = 0; j < Math.round(sums[i] / 10); j++) {
                System.out.print(("#"));
            }
            System.out.println((""));
        }
        System.out.println("This test took "+ (end - start)/1000000000 + " seconds");
    }
}
```

11.2.5 Dice.java

```
import java.util.Random;

/**
 * The {@code Dice} class creates a rollable dice with a definable number of
 * sides
 */
public class Dice {
    private int sides;
    private int value;

    /**
     * @param sides of the dice
     */
    public Dice(int sides) {
        if (sides < 0) {
            throw new IllegalArgumentException("The number of sides on a dice cannot be negative")
        }
        this.sides = sides;
    }

    /**
     * Calculates a random number within the range of the number of sides on the
     * dice and assigns it to the value of the dice
     */
    public void rollDice() {
        Random rand = new Random();
        this.value = rand.nextInt(sides) + 1;
    }

    /**
     * @return the value of the dice
     */
    public int getValue() {
        return this.value;
    }
}
```

11.2.6 Game.java

```
import java.io.File;
import java.io.FileWriter;
import java.util.Scanner;

public class Game {

  private Player[] players;
  private Cup cup;
  private final int[] boardValues = {
    0,
    100,
    250,
    -100,
    100,
    -20,
    180,
    0,
    -70,
    60,
    -80,
    -50,
    650,
  };
  private BoardField[] boardFields;
  private Language language;
  private int turn = 0;
  Scanner scanner;

  Game(int players, int diceSides) {
    this.scanner = new Scanner(System.in);
    while (true) {
      System.out.println("Which language? (english, deutsch, dansk, nihongo)");
      var language = this.scanner.nextLine();
      if (
        language.equals("english") ||
        language.equals("deutsch") ||
        language.equals("dansk") ||
        language.equals("nihongo")
      ) {
        this.language = new Language(language);
        break;
      } else {
        System.out.println("Wrong input.");
      }
    }
    this.players = new Player[players];
    var boardNames = this.language.getBoardNames();
    this.boardFields = new BoardField[boardNames.length / 2];
    for (var i = 0; i < (boardNames.length / 2); i++) {
      this.boardFields[i] =
        new BoardField(
          boardNames[i * 2],
          this.boardValues[i],
          boardNames[i * 2 + 1]
        );
    }
    this.cup = new Cup(2, diceSides);
  }

  void startGame() {
    // double start;
    // double end;

    var loadSaveFile = false;
    try {
      var saveFile = new File("SaveData.txt");
      if (saveFile.exists()) {
        while (true) {
          this.language.printSaveQuestion();
          var input = this.scanner.nextLine();
          if (input.equals("y") || input.equals("yes")) {
            loadSaveFile = true;
            break;
          } else if (input.equals("n") || input.equals("no")) {
            break;
          } else {
            continue;
          }
        }
      }
      if (loadSaveFile) {
        var fileScanner = new Scanner(saveFile);
        for (var i = 0; fileScanner.hasNextLine(); i++) {
          var playerData = fileScanner.nextLine().split(",");
          var playerName = playerData[0];
          var playerBalance = Integer.parseInt(playerData[1]);
          this.players[i] = new Player(playerName, playerBalance);
        }
        fileScanner.close();
        for (var i = 0; i < this.players.length; i++) {
          System.out.println(
            this.players[i].getName() +
            ": " +
            this.language.getAccountMessage() +
            this.players[i].account.getBalance()
          );
        }
      }
    } catch (Exception e) {
      this.language.printNotLoaded();
    }
    if (!loadSaveFile) {
      for (var i = 0; i < this.players.length; i++) {
        this.language.printPlayerNameInput(i + 1);
        this.players[i] = new Player(this.scanner.nextLine(), 1000);
      }
    }

    int turn = 0;

    gameloop:while (true) {
      this.language.printPlayerOptions(this.players[turn].getName());
      var input = this.scanner.nextLine();
      if (input.equals("q") || input.equals("quit")) {
        break gameloop;
      }
      if (input.equals("r") || input.equals("roll")) {
        // start = System.nanoTime();
      } else if (input.equals("s") || input.equals("save")) {
        try {
          var fileWriter = new FileWriter("SaveData.txt");
          for (var i = 0; i < this.players.length; i++) {
```

```
      fileWriter.write(
        String.format(
          "%s,%d\n",
          this.players[i].getName(),
          this.players[i].account.getBalance()
        )
      );
    }
    fileWriter.close();
  } catch (Exception e) {
    this.language.printNotSaved();
  }
  continue;
} else {
  this.language.printWrongInput();
  continue;
}
this.cup.rollDice();
var sum = this.cup.getDiceSum();
var balance = this.players[turn].account.getBalance();
var boardPosition = sum % this.boardFields.length;
this.players[turn].account.setBalance(
    balance + this.boardFields[boardPosition].getPoints()
  );
var diceValues = this.cup.getDiceValues();
this.language.printDiceValues(diceValues);
this.language.printSum(cup.getDiceSum());

this.language.printLandedOn(
    boardPosition,
    this.boardFields[boardPosition].getName()
  );

System.out.println(this.boardFields[boardPosition].getDescription());

System.out.println(
  this.language.getAccountMessage() +
  this.players[turn].account.getBalance() +
  System.lineSeparator()
);
```

```
      // check for winner
      for (var i = 0; i < this.players.length; i++) {
        if (this.players[i].account.getBalance() >= 3000) {
          this.language.printWinner(this.players[i].getName());

          break gameloop;
        }
      }
      if (
        this.boardFields[boardPosition].getName().equals("The Werewall")
      ) continue;
      switchTurn();
      // end = System.nanoTime();
      // System.out.println(
      //   "This test took " + (end - start) / 1000000000 + " seconds"
      // );
    }
    this.scanner.close();
  }

  public void switchTurn() {
    this.turn = (this.turn + 1) % this.players.length;
  }
}
```

11.2.7 Language.java

```
import java.io.BufferedReader;
import java.io.FileReader;

public class Language {

  private String saveQuestion;
  private String playerNameInput;
  private String playerOptions;
  private String wrongInput;
  private String diceValues;
  private String sum;
  private String landedOn;
  private String winner;
  private String notSaved;
  private String notLoaded;
  private String accountMessage;

  private String[] boardFields;

  Language(String language) {
    try {
      var br = new BufferedReader(new FileReader("languages.csv"));
      var line = "";
      while ((line = br.readLine()) != null) {
        var data = line.split(";");
        if (language.equals(data[0])) {
          {
            this.saveQuestion = data[1];
            this.playerNameInput = data[2];
            this.playerOptions = data[3];
            this.wrongInput = data[4];
            this.diceValues = data[5];
            this.sum = data[6];
            this.landedOn = data[7];
            this.winner = data[8];
            this.notSaved = data[9];
            this.notLoaded = data[10];
            this.accountMessage = data[11];
            this.boardFields = new String[data.length - 12];
            for (var i = 12; i < data.length; i++) {
             this.boardFields[i-12] = data[i];
            }
            break;
          }
        }
      }
      br.close();
    } catch (Exception e) {}
  }

  public void printSaveQuestion() {
    System.out.println(this.saveQuestion);
  }

  public void printPlayerNameInput(int number) {
    System.out.println(String.format(this.playerNameInput, number));
  }

  public void printPlayerOptions(String name) {
    System.out.println(String.format(this.playerOptions, name));
  }

  public void printWrongInput() {
    System.out.println(this.wrongInput);
  }

  public void printDiceValues(int[] diceValues) {
    System.out.print(this.diceValues + ":");
    for (var j = 0; j < diceValues.length; j++) {
      System.out.print(" " + diceValues[j]);
    }
  }

  public void printSum(int sumValue) {
    System.out.println(String.format(", %s: %d", this.sum, sumValue));
  }

  public void printLandedOn(int boardPosition, String boardName) {
    System.out.println(
      String.format("%s %d: %s", this.landedOn, boardPosition, boardName)
    );
  }

  public void printWinner(String name) {
    System.out.println(String.format("%s: %s!", this.winner, name));
  }
  public void printNotSaved() {
    System.out.println(this.notSaved);
  }
  public void printNotLoaded() {
    System.out.println(this.notLoaded);
  }

  public String[] getBoardNames() {
      return boardFields;
  }

  public String getAccountMessage() {
    return this.accountMessage;
  }
}
```

11.2.8 main.java

```java
class Main {
    public static void main(String[] args) {
        var game = new Game(2, 6);
        game.startGame();
    }
}
```

11.2.9 Player.java

```
public class Player {

  private String name;
  Account account;

  Player(String name, int startBalance) {
    this.name = name;
    this.account = new Account(startBalance);
  }

  public String toString() {
    return this.name + ": " + this.account.toString();
  }

  public String getName() {
    return this.name;
  }
}
```

11.2.10 class_diagram.wsd

```
/'
https://plantuml.com/class-diagram
https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/
'/

@startuml class_diagram
'skinparam classAttributeIconSize 0

class Game {
    -players: Player[]
    -boardFields: BoardField
    -cup: Cup
    -language: Language
    +Game(players: int, diceSides: int, language: String)
    +startGame(): void
    +switchTurn(): void
    +toString(): String
}

class Language {
    - String saveQuestion
    - String playerNameInput
    - String playerOptions
    - String wrongInput
    - String diceValues
    - String sum
    - String landedOn
    - String winner
    - String notSaved
    - String notLoaded
    + printSaveQuestion(): void
    + printPlayerNameInput(number: int): void
    + printPlayerOptions(name: String): void
    + printWrongInput(): void
    + printDiceValues(diceValues: int[]): void
    + printSum(sumValue: int): void
    + printLandedOn(boardPosition: int, boardName: String): void
    + printWinner(name: String): void
    + printNotSaved(): void
    + printNotLoaded(): void
    + getBoardNames(): String[]
}

class Player {
    -name: String
    +account: Account
    +Player(name: String, startBalance: int)
    +getName(): String
    +toString(): String
}

class BoardField {
    -name: String
    -points: int
    -description: String
    +getPoints(): int
    +getName(): String
    +getDescription(): String
}

class Cup {
    -dice: Dice[]
    -amountOfDice: int
    +Cup(amountOfDice: int, diceSides: int)
    +rollDice(): void
    +getDiceValues(): int[]
    +getDiceSum(): int
    +isDiceEqual(): boolean
    +toString(): String
}

class Dice {
    -sides: int
    -value: int
    +Dice(sides: int)
    +rollDice(): void
    +getValue(): int
}

class Account {
    -balance: int
    +Account(startBalance: int)
    +getBalance(): int
    +addBalance(): void
    +toString(): String
}

Game::players ..> Player
Game::boardFields ..> BoardField
Game::cup ..> Cup
Game::language ..> Language
Cup::dice ..> Dice
Player::account ..> Account
@enduml
```

11.2.11 domain_model.wsd

```
@startuml domain_model
skinparam linetype polyline
class Player {
    name
    account

}

class Language {

}

class Game {
    players
    boardFields
    cup

}
class Cup {
    dice
    amountOfDice

}
class BoardField {
    name
    points
    description
}
class Dice {
    sides
    value
}

Game "1" - "1" Language: Includes
Game "1" - "1" Cup: Includes
Game "1" - "2" Player: Plays
Game "1" -- "11" BoardField: Includes
Game "1" -- "2" Dice: Includes
Cup "1" - "1" Player: Shakes
Cup "1" - "2" Dice: Rolls
```

```
@enduml
```

11.2.12 system_sequence_diagram.wsd

```
@startuml system_sequence_diagram

participant "p1: Account" <<(C,#ADD1B2)>>
participant "p2: Account" <<(C,#ADD1B2)>>
actor "p1 : Player" <<(C,#ADD1B2)>>
actor "p2 : Player" <<(C,#ADD1B2)>>
participant Console
participant ": Game" <<(C,#ADD1B2)>>
participant ": Cup" <<(C,#ADD1B2)>>

[o-> ": Game"
": Game" -> Console: Intro message

group while [winner == none]
        {roll} "p1 : Player" -> Console: Type input
        ": Game" -> Console: Read input
    alt roll
        "p1 : Player" -> ": Game": cup.rollDice()
        ": Game" -> ": Cup": rollDice()
        ": Game" -> ": Cup": sum = getDiceSum()
        ": Game" -> ": Game": points = fields[sum + field].points
        ": Game" -> "p1 : Player": account.addBalance(points)
        "p1 : Player" -> "p1: Account": addBalance(points)
        ": Game" -> ": Game": switchTurn()
        "p2 : Player" -> ": Game": cup.rollDice()
        ": Game" -> ": Cup": rollDice()
        ": Game" -> ": Cup": sum = getDiceSum()
        ": Game" -> ": Game": points = fields[sum + field].points
        ": Game" -> "p2 : Player": account.addBalance(points)
        "p2 : Player" -> "p2: Account": addBalance(points)
        ": Game" -> ": Game": switchTurn()
    else invalid input
        ": Game" -> ": Game"
    else quit
        ": Game" ->x ": Game"
    end
    ": Game" ->x ": Game"
end

@enduml
```