Danmarks Tekniske Universitet

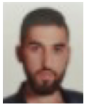
02312 02313 02315

62532 Version kontrol and test methods

Dice game

Gruppe - 20

s205457 - Assad Shamsi



s183033 - Ali Hassan

s092512 - Muhammad Ali

s184749 - Mahdi Muhammad

s164932 - Rivan Joba

s183729 - Thomas B. Frederiksen

02/10/2020

Versionsstyring og testmetoder

Terningespil i java

Testing af et software program

Timeregnskab

Timeregnskab (*hele antal timer*)							
	Mahdi	**Ali**	**Khan**	**Assad**	**Thomas**	**Mohamed**	**Rivan**
Analyse	5	6	5	6	4	5	5
Design	3	5	5	5	3	5	3
Implementering	4	3	4	3	4	3	4
Test	1	1	1	1	2	1	1
Dokumentation	1	1	1	1	1	2	1
I alt	**14**	**16**	**16**	**16**	**14**	**16**	**14**

Abstract

The purpose of this paper is to create a multiplayer game, where the player can throw two dice indicating which of the fields from 2 to 12 the player lands on, which in turn can have a positive or negative effect on the player. Furthermore, when the player reaches 3000 points the game ends. All of this is programmed with Java in IntelliJ, where version control (Git) has been applied to the process of making this game.

The game is working splendidly without any encountered problems. Furthermore the game has been tested on java version 1.8 on the Technical University of Denmark's (DTU's) computers and functions perfectly there as well.

Throughout this paper, there has been copious information retrieval, which is used while solving the issues. Throughout the information retrieval, we have remained source critical and only used the information if the source is credible.

Introduktion

I denne rapport vil et simpelt terningespil blive implementeret i java. Målet i spillet er for en spiller at opnå 3000 guld mønter. Hver runde kaster spilleren 2 terninger og værdien af det samlede øjne på terningerne, som angiver hvilket af de 11 mulige felter de er landet på. Hver felt har sin egen historie, og et antal guldmønter som enten bliver tilføjet eller trukket fra spillerens pengebeholdning. Af disse felter er felt nr. 10 specielt da det også giver spilleren en ekstra tur.
Igennem hele projektet vil VCS (Version Control) bruges til at styre projektet, danne overblik og muliggøre at hver enkelt udvikler kan arbejde på projektet individuelt. Strukturen af VCS er i vores tilfælde blevet opsat således at der findes en master branch, en development branch og et antal feature branches. Når en udvikler arbejder på en feature arbejder de på deres egen specifikke feature branch. Dette merges ind på development når denne feature er klar, og endelig ind på master når alting kører som det skal.

Formål

Formålet med denne opgave er at lave et program som en kunde ønsker. programmet er blevet beskrevet i ovenstående introduktions kapitel, samt er formålet at kunne versionsstyre under processen, vise de enkelte studerende kan brug klasser, relationer og GRASP.

Krav

Nedenstående kravliste, er de krav som er blevet stillet af kunden som skal være en del af det endelige produkt.

Kravliste:

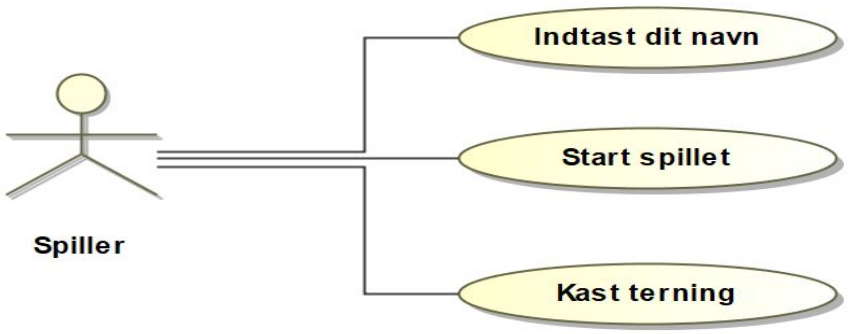
- Vi skal bruge to terninger, hvor hver en repræsenterer tal fra 1 til 6.
- Spilleren skal kaste terningerne.
- Spillet vælger terning værdi tilfældigt.
- Pengebeholdning med 1000 i starten.
- Regneprogram kan regne penge med ekstra tillæg fra feltliste.
- Spillet slutter når pengebeholdning rammer 3000 og så vinder spilleren.

Analyse

I denne del af rapporten beskrives programmets funktionalitet via forskellige diagrammer.

Use case diagram:

Diagrammet her viser hvilke funktioner spillet skal have. Og der er tre funktioner i vores spil.



Use case beskrivelse

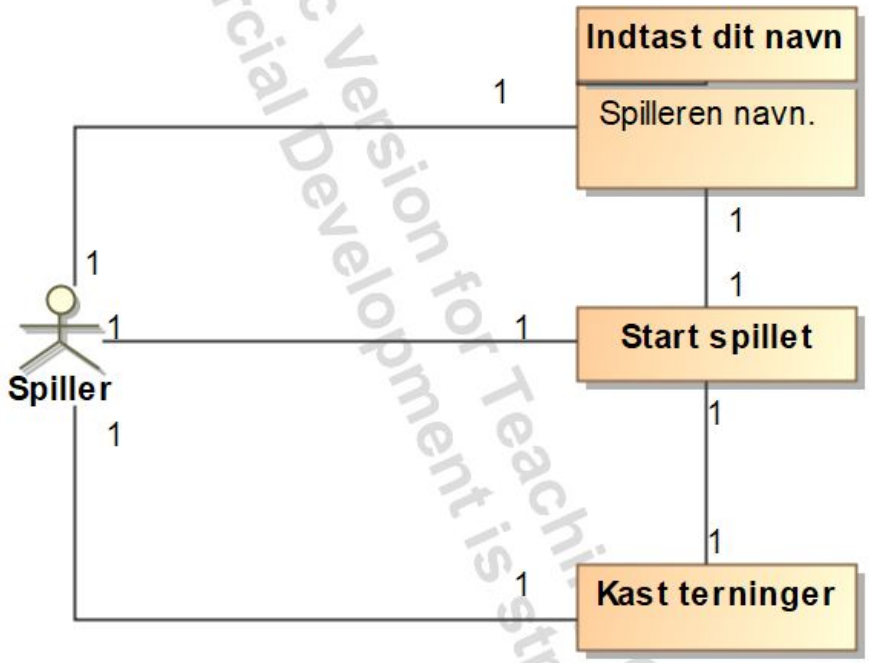
Indtast dit navn: denne funktion muliggør at spilleren kan indtaste sit navn.

Start spillet: denne funktion sætter spillet i gang.

Kast terning: denne funktion giver mulighed for at kaste terninger og se resultatet.

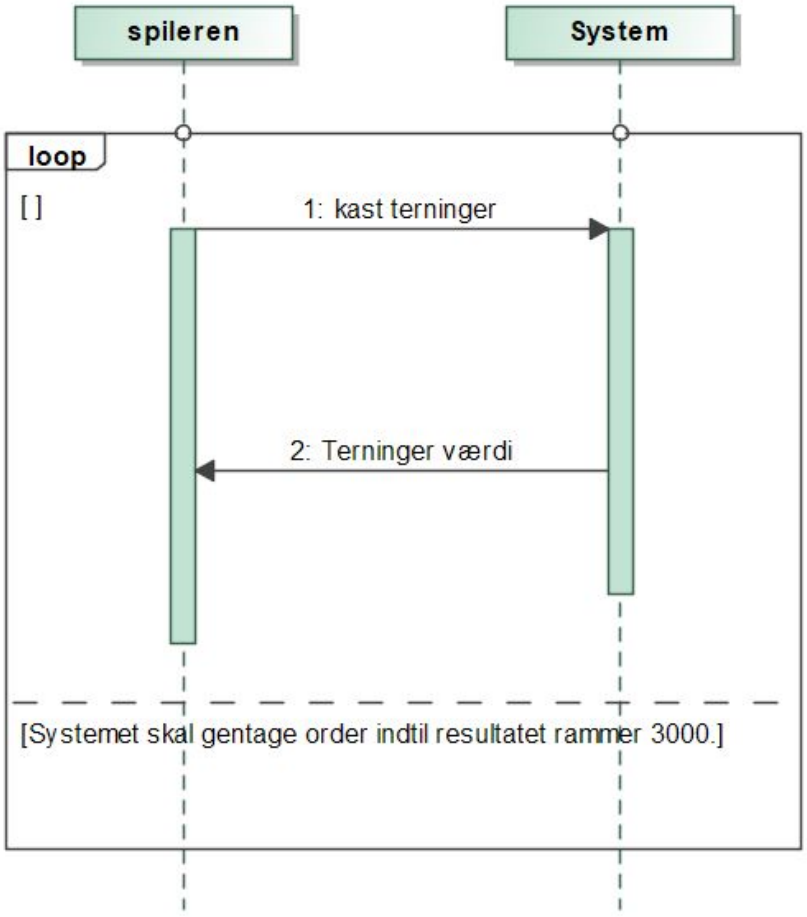
Domænemodel:

Domænemodellen viser relationer mellem forskellige funktioner i Use Case, og viser hvor meget afhængig af hinanden.



Systemsekvensdiagram:

Diagrammet viser en succes scenario med detaljerne om begivenheder, der genereres af spilleren uden for systemet.

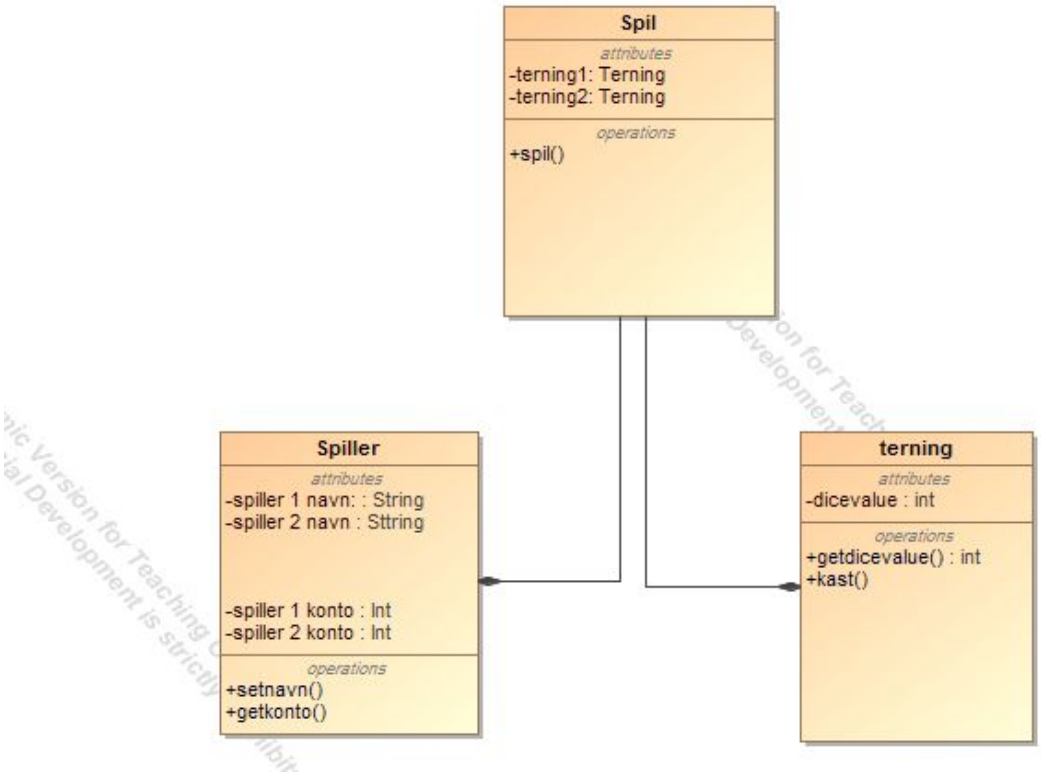


Design

I denne del af rapporten vil forskellige diagrammer præsenteres der viser baggrund tankerne for hvordan spillet blev designet, og hvorfor det blev designet på denne måde.

(Klassediagram)

Beskriver strukturen i systemet ved at vise systemets klasser,deres attributter, metoder og forholdet mellem objekter.



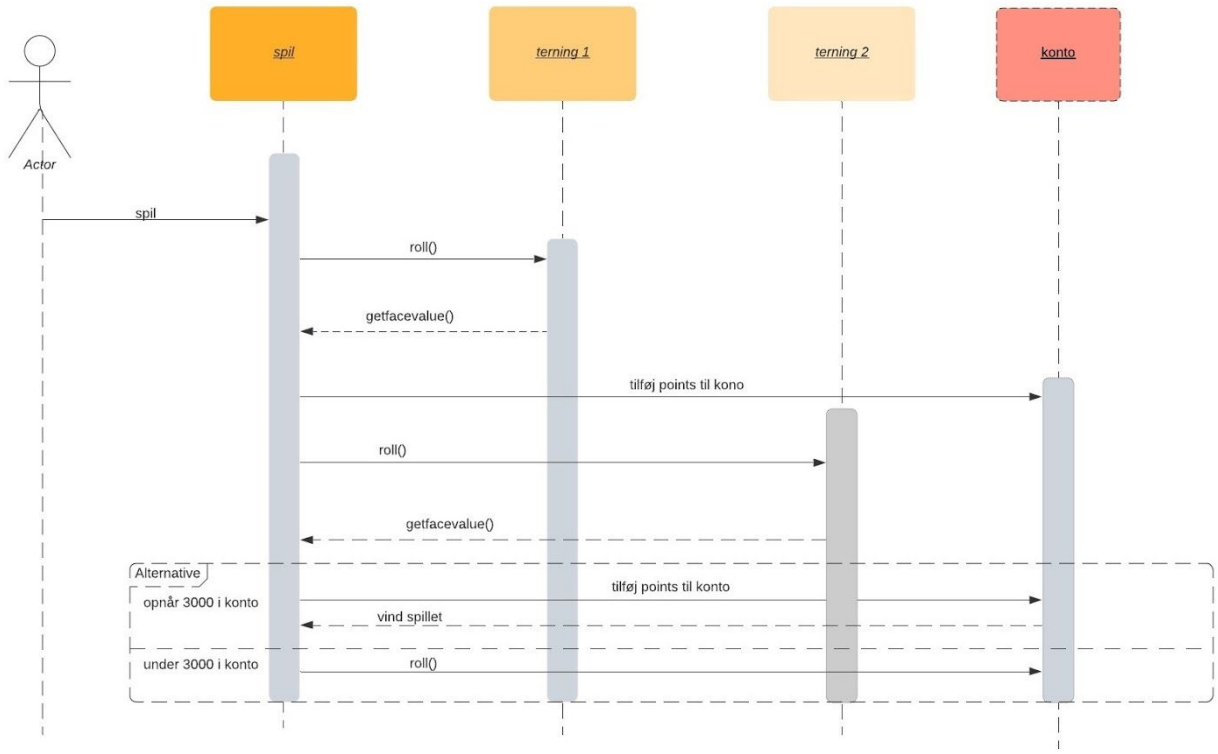
Spil klassen : Klassen har de metoder, der kan køre spillet og bruger de metoder, der er skrevet i de andre klasser

Spiller klassen : Denne klasse har en metode, hvor spilleren kan indtaste sit navn. systemet gemmer navnet og bruger det derefter til at vise spillernavnet med hans resultat.

Terningklassen : Klassen har de metoder, der er vigtige for at kontrollere, hvordan terningerne fungerer i spillet.

(Sekvensdiagram)

Viser interaktioner mellem skuespillere og genstande og mellem to objekter.



I dette diagram kan man se hvordan klasserne hænger sammen.

Man kan læse ud fra diagrammet hvad er de vigtigste regler i spillet, hvor spilleren skal nå 3000 point på sin konto for at vinde spillet.

Implementation

Denne implementations kapitel beskriver logikken for programmet heraf klasserne.

FeltListkeyValue

FeltListKeyValue klassen har til ansvar for at skabe den fundamentale felt heraf hvad der karakteriserer en felt såsom key, værdi og om brugeren får en ekstra tur, der er lavet fem metoder i klassen. 'getValue' metoden har til ansvar at returnere værdien samt har 'getKey' metoden til ansvar for at returnere key og 'GiverEkstraTur' returnere giverekstratur. Denne klasse kan ses i nedenstående bilag A - FeltListKeyValue.

FeltListe

FeltListe klassen har til ansvar at definerer de her felt så når den enkelte spiller lander på en felt, så er der enten en positiv eller negativ effekt. Dernæst er felterne navngivet i indeksene i et array. Denne klasse kan ses i nedenstående bilag A - FeltList.

Konto

Denne klasse har til ansvar for spillerens konto heraf hvor mange points man har i kontoen, opdatering af konto samt må spilleren ikke gå i minus som er blevet lavet vha. en IF sætning som tjekker om den spillerens opdateret konto er under 0. Denne klasse kan ses i nedenstående bilag A - Konto.

Main

Main klassen har til ansvar for at køre hele spillet, denne klasse kan ses i nedenstående bilag A - Main.

Print

Print klassen indeholder alle de metoder og felter der bruges til at printe til konsollen. Denne er opsat så der for et print Object først skal kaldes metoden “AskForLanguage” der beder brugeren vælge sprog. Herefter loades alt tekst der printes til konsollen for det givne sprog, ind i de forskellige felter af objektet, ved brug af “loadSingleLine” og “loadMultiLine” metoderne. Hertil kommer en række forskellige metoder til at få eller printe de forskellige felter af objektet. Denne opsætning er valgt da der i nogle tilfælde bare skal printes indholdet af de forskellige felter, mens der i andre sammenhænge skal printes sammen med andre objekter (Spillerens navn f.eks.). Koden for Print klassen kan ses i bilag A.

Spil

Her er der en ‘Start’ metode som har til ansvar for at spørge om sprog til den enkelte spiller heraf Dansk eller Engelsk. Dernæst bliver ‘StartGame’ metoden kaldt hvor spillet nu er begyndt. Der bliver tjekket om nuværende spiller har over eller lige med 3000 points for vinderen skal meddeles. Endvidere skal spillerens navn, account og penge printes ud så spillerne kan se det. Denne klasse kan ses i nedenstående bilag A - Spil.

Spiller

i denne klasse er der en string som er navn, samt konto. Det er her hvor spiller objekt bliver oprettet altså som nævnt navn og konto. Denne klasse kan ses i nedenstående bilag A - Spiller.

Terning

Terning klassen har til ansvar at lave objektet terning hvor maximum og minimum værdi af en terning defineres. metoden ‘generateRandomDiceValue’ skal returnerer værdi for terningerne. Denne klasse kan ses i nedenstående bilag A - Terning.

Test

Udover manuelt at køre spillet igennem for fejl, blev der også opsat en række automatiske test ved brug af JUnit. Klasserne FeltListe, Konto, Print, Spiller og Terning blev test for de metoder hvorpå det var set relevant. Disse fik derved hver deres Test klasse som hver især verificerede henholdsvis:

- At alle felter blev givet en rigtig værdi
- At man ikke kan have en negativ pengebeholdning
- At de forskellige print statements blev indlæst som forventet
- At spillerne bliver initialiseret med en begyndende pengebeholdning på 1000
- At terningen kun kan returnerer værdier mellem 1 og 6 (per terning) og at alle værdier ved 60000 kast forekommer lige mange gange, dvs. 10000 gange ± 400;

Derudover er det manuelt blevet testet at programmet kan kompileres og køres på DTU's computere (java 1.8, JDK 11). En Jar-fil blev genereret som programmet kan køres ud fra, ved kommandoen "java -jar 20_del2.jar" i cmd.

Projektplan

I nedenstående tabel ses det hvordan gruppe 20 har fordelt ansvar i projektet så alle har noget at lave.

Area of responsibility:	Main	Terning	Spil	Spiller	Print	Testing and Unit Testing	FeltList	FeltList KeyValue	Konto
Rivan									x
Mahdi	x							x	
Thomas					x	x			
Khan	x	x	x	x			x	x	x
Asaad		x							
Ali				x					
Tamim							x		

Konklusion

Der kan hermed konkluderes at programmet virker som det skal og er testet flere gange på både DTU's og de studerendes computer, hvor programmet kørte som det skulle uden forhindringer. Gruppen havde et rigtigt godt samarbejde hvor der blev kommunikeret godt. Dernæst blev gruppen klogere på versionsstyre under processen samt anvendelse af klasser, relationer og GRASP.

Bilag A: Kode

FeltListkeyValue

```java
package Classes;

public class FeltListKeyValue {
    private  int key;
    private int value;
    private boolean giverEkstraTur;

    public FeltListKeyValue(int key, int value, boolean giverEkstraTur){
        this.key = key;
        this.value = value;
        this.giverEkstraTur = giverEkstraTur;
    }

    public FeltListKeyValue(int key, int value) { this(key, value,  giverEkstraTur: false); }
    public int getKey() { return key; }

    public int getValue() { return value; }

    public boolean giverEkstraTur() { return giverEkstraTur; }



}
```

FeltListe

```java
package Classes;

public class FeltListe {

    private FeltListKeyValue tower = new FeltListKeyValue(2, 250);
    private FeltListKeyValue crater = new FeltListKeyValue(3, -100);
    private FeltListKeyValue palaceGates = new FeltListKeyValue(4, 100);
    private FeltListKeyValue coldDesert = new FeltListKeyValue(5, -20);
    private FeltListKeyValue walledCity = new FeltListKeyValue(6, 180);
    private FeltListKeyValue monastery = new FeltListKeyValue(7, 0);
    private FeltListKeyValue blackCave = new FeltListKeyValue(8, -70);
    private FeltListKeyValue hutsInMountain = new FeltListKeyValue(9, 60);
    private FeltListKeyValue werewall = new FeltListKeyValue( key: 10,  value: -80,  giverEkstraTur: true);
    private FeltListKeyValue pit = new FeltListKeyValue(11, -50);
    private FeltListKeyValue goldmine = new FeltListKeyValue(12, 650);

    private FeltListKeyValue[] feltListValues = new FeltListKeyValue[11];

    public FeltListe() {
        feltListValues[0] = tower;
        feltListValues[1] = crater;
        feltListValues[2] = palaceGates;
        feltListValues[3] = coldDesert;
        feltListValues[4] = walledCity;
        feltListValues[5] = monastery;
        feltListValues[6] = blackCave;
        feltListValues[7] = hutsInMountain;
        feltListValues[8] = werewall;
        feltListValues[9] = pit;
        feltListValues[10] = goldmine;
    }
    public FeltListKeyValue getValue(int feltKey) {
        for (int i = 0; i < feltListValues.length; i++) {
            FeltListKeyValue keyValue = feltListValues[i];
            if (keyValue.getKey() == feltKey) {
                return keyValue;
            }
        }
        return null;
    }
}
```

Konto

```java
package Classes;

public class Konto {
    private int pengeBeholdning;

    public Konto(int startPengeBeholdning) { this.pengeBeholdning = startPengeBeholdning; }

    public int getPengeBeholdning() { return pengeBeholdning; }

    public boolean updatePengeBeholdning(int fieldValue) {
        int nyPengeBeholdning = pengeBeholdning + fieldValue;
        if(nyPengeBeholdning < 0){
            return false;
        }
        pengeBeholdning = nyPengeBeholdning;
        return true;
    }
}
```

Main

```java
package Classes;

public class Main {
    public static void main(String[] args) {
        Spil spil = new Spil();
        spil.start();
    }
}
```

Print

```java
package Classes;

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

```
public class Print {

    private String[] Options;
    private String Turn;
    private String Language;
    private String[] PlayerNamePrompt;
    private String[] endGame;
    private String AccountError;
    private String Path;
    private String Account;
    private String AskForEnter;
    public Print() {
    }

    public void AskForLanguage() throws FileNotFoundException {
        Scanner s = new Scanner(System.in);
        System.out.println("Please choose a language \n");
        System.out.println("1. English \n");
        System.out.println("2. Danish \n");
        int choice = s.nextInt();
        if(choice==1) {
            Language = "English";
        }
        else {
            Language = "Danish";
        }

        setPrints();
    }

    public void setPrints() throws FileNotFoundException {
        if (Language.equals("English")) {
            Path = ".\\PrintStatements\\English\\";
        }
        else if (Language.equals("Danish")) {
            Path = ".\\PrintStatements\\Danish\\";
        }
        String filepath = Path + "Options.txt";
```

```
        Options = loadMultiLine(filepath,11);
        filepath = Path + "Turn.txt";
        Turn = loadSingleLine(filepath);
        filepath = Path + "PlayerNamePrompt.txt";
        PlayerNamePrompt = loadMultiLine(filepath,2);
        filepath = Path + "EndGame.txt";
        endGame = loadMultiLine(filepath,2);
        filepath = Path + "AccountError.txt";
        AccountError = loadSingleLine(filepath);
        filepath = Path + "Account.txt";
        Account = loadSingleLine(filepath);
        filepath = Path + "AskForEnter.txt";
        AskForEnter = loadSingleLine(filepath);
    }

    public String promptNames(int i) {
        Scanner s = new Scanner(System.in);
        System.out.println(PlayerNamePrompt[i]);
        System.out.println();
        return s.nextLine();
    }

    public void PrintField(int FieldNr) {
        System.out.println(Options[FieldNr-2]);
    }

    public String getAccount() {
        return Account;
    }
    public String AskForEnter() {
        return AskForEnter;
    }

    public void AccountError() {
        System.out.println(AccountError);
    }
    public void newRound() {
        System.out.println("--------------------------------");
```

```
        System.out.println();
    }
    public void endGame(String playerName) {
        newRound();
        System.out.println(playerName + endGame[0]);
        System.out.println(endGame[1]);
    }

    public String printTurn() {
        return Turn;
    }

    //Public to be used for unit testing
    public String[] loadMultiLine(String pathToFile, int lines) throws
FileNotFoundException {
        Scanner s = new Scanner(new File(pathToFile));
        String[] st = new String[lines];
        for(int i = 0;i<lines;i++){
            st[i] = s.nextLine();
        }
        s.close();
        return st;
    }

    static String loadSingleLine(String pathToFile) throws
FileNotFoundException {
        Scanner s = new Scanner(new File(pathToFile));
        String st = s.nextLine();
        s.close();
        return st;
    }
}
```

Spil

```java
package Classes;

import java.io.FileNotFoundException;
import java.util.Random;
import java.util.Scanner;

public class Spil {

    private Spiller player1;
    private Spiller player2;
    private Spiller currentPlayer;
    private FeltListe feltList = new FeltListe();
    private Terning terning = new Terning();
    private Scanner scanner = new Scanner(System.in);
    private Random random = new Random();

    public void start() {
        Print print = new Print();
        try {
            print.AskForLanguage();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }

        String player1Name = print.promptNames( i: 0);
        player1 = new Spiller(player1Name);
        String player2Name = print.promptNames( i: 1);
        player2 = new Spiller(player2Name);

        System.out.println();

        //player1 starts the game
        currentPlayer = player1;

        startGame(print);
    }

```

```
38      private void startGame(Print print) {
39          int winningPoints = 3000;
40          boolean gameIsRunning = true;
41          while (gameIsRunning) {
42
```

```
43              String PlayerName = currentPlayer.getNavn();
44              System.out.println(PlayerName + print.printTurn());
45              String waitForDiceThrow = scanner.nextLine();
46              if (waitForDiceThrow.equals("")) {
47                  int feltKey = terning.kast();
48
49                  FeltListKeyValue keyValue = feltList.getValue(feltKey);
50                  int value = keyValue.getValue();
51                  print.PrintField(feltKey);
52
53                  Konto konto = currentPlayer.getKonto();
54                  boolean Success = konto.updatePengeBeholdning(value);
55                  if (!Success) {
56                      print.AccountError();
57                  }
58
59                  System.out.println(PlayerName + print.getAccount() + konto.getPengeBeholdning());
60
61                  if (konto.getPengeBeholdning() >= winningPoints) {
62                      print.endGame(PlayerName);
63                      gameIsRunning = false;
64                      return; //exit the method
65                  } else {
66                      if (currentPlayer == player1) {
67                          currentPlayer = player2;
68                      } else {
69                          currentPlayer = player1;
70                      }
71                  }
72                  print.newRound();
73              }
74              else {
75                  System.out.println(print.AskForEnter());
76              }
77          }
78      }
79  }
```

Spiller

```java
package Classes;

public class Spiller {

    private String navn;
    private Konto konto = new Konto( startPengeBeholdning: 1000);

    public Spiller(String navn) { this.navn = navn; }

    public String getNavn() { return navn; }

    public Konto getKonto() { return konto; }
}
```

Terning

```java
package Classes;

import java.util.Random;

public class Terning {

    private Random random = new Random();

    public int generateRandomDiceValue(){
        int diceMinimumValue = 1;
        int diceMaximumValue = 6;
        return diceMinimumValue + random.nextInt((diceMaximumValue - diceMinimumValue) + 1);
    }

    public int kast() { return generateRandomDiceValue() + generateRandomDiceValue(); }
}
```