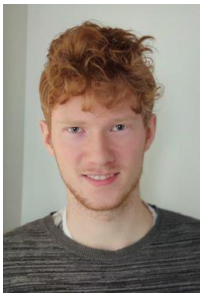
Technical University of Denmark

DTU Compute
Department of Applied Mathematics and Computer Science

DTU Diplom
Center for Bachelor of Engineering Studies

65532 Versionsstyring og testmetoder
62531 Udviklingsmetoder til IT-systemer
02312 Indledende programmering

Isak Risager
S205452



Simon Junker Eriksen
S205453

Daniel Terrell Sutton
S205451

Victor Daugaard Cebula
S205867

Louis Monty-Krohn
S205424

Lucas Loft Skals
S205444

CDIO DELOPGAVE 2 - 30/10-2020
GRUPPE 24
SOFTWARETEKNOLOGI, DIPLOM

Timeregnskab

Timeregnskab for gruppe 24 CDIO del 2 i timer	
Isak	**17**
Louis	**23**
Daniel	**12**
Lucas	**10,5**
Victor	**6,5**
Simon	**2**

Ansvarsfordeling (ansvar for rapportskrivning fremgår i indholdsfortegnelsen via paranteser):

Programmering: Louis, Lucas, Victor & Simon (fratrådt)
Rapportskrivning: Louis, Isak & Daniel

Abstract (Isak)

Denne rapport beskriver processen i at lave et brætspil med terningerne mellem to spillere.
Selve projektet er blevet udført med Unified Process, hvor der er blevet arbejdet med Elaboration og Construction faserne over 4 iterationer. Dertil er der blevet arbejdet ud fra en kravliste der er baseret på krav fra kunden og projektlederen. I analysen har vi brugt diagrammer; use case diagram, system sekvens diagram, sekvensdiagram og design klasse diagram.
Vi har følgende klasser: Main, GameLoop, Dice, Player, Account, Gameboard, Rules. Al koden er kodet i Java, via IDE'en IntelliJ i sproget UTF-8 og versioneret med github. Der er blevet lavet ad hoc test til projektet for at sikre sig at metoderne virkede korrekt.

Indholdsfortegnelse

1 Indledning (Isak)

Spilfirmaet IOOuteractive har endnu en gang ønsket, at vi laver et spil til dem. Denne gang er der tale om et brætspil der spilles med terninger mellem to personer. Spillet skal kunne spilles på tur, og afhængigt af terningekastet vil spillerens konto ændres. Der vindes ved at opnå mindst 3000 point, og spillet skal være nemt at ændre til både andre sprog og andre terninger.

Til spillet har kunden opstillet en række krav, og projektleder givet bemærkninger i forhold til hvordan software skal udarbejdes. Dette bliver beskrevet i rapporten.

I løbet af projektet, var der desværre et gruppemedlem der besluttede sig for, at revurdere sin beslutning om at være på studiet to uger inde i projektet, og dermed ikke udførte det arbejde han var tildelt fra dag 1. Derudover var der et andet gruppemedlem, der havde et familiemedlem der blev meget syg, og dermed ikke havde hverken tid eller overskud til dette projekt.

2 Krav (Daniel)

Til denne opgave er vi blevet stillet nogle krav som skal følge kundens vision af systemet. Tabellen nedenfor viser disse krav:

Krav	Beskrivelse
1	To spillere med hver deres point-konto
2	Skal kunne spilles på DTU's databarer
3	Kode skal køre på under 333 ms
4	Skal spilles med to terninger med seks sider hver
5	Felterne skal kunne ændre en spillers point-konto
6	Spiller kan lande på et felt mellem 2-12
7	En spiller starter med 1000 point
8	Man vinder ved at have mindst 3000 point
9	Der anvendes UTF-8
10	Skal kunne skifte til andre terninger
11	Man starter forfra på felterne når sidste felt er nået

3 Analyse (Louis)

For at forberede os til designarbejdet i dette projekt, har vi lavet 3 forskellige analyser:
(1) use-case analyse
(2) domæne analyse
(3) analyse af system sekvens.

Vi arbejder på analysen i elaborations fase af projektet med inspiration draget fra kravspecifikationerne.

3.1 Use case

Vi har udarbejdet en liste over use cases og har derefter inddelt dem i, hvilke aktører der skal bruge dem.

Aktører	Mål
Spilleren	vil kunne slå terninger vil kunne se balance vil kunne vinde/tabe
Terninger	vil kunne generere tilfældige terningkast
Spilleplade	vil kunne opholde spilleregler vil kunne give information om terning vil kunne +/- balance

Tabel 3.1.1

Vi har valgt den mest centrale use case og har så skrevet en fully dressed use case beskrivelse

Use Case UC1: RollDice	Kommentarer
Scope:	Brætspil
Level:	User-goal
Stakeholders and conditions	- Spilleren: vil kunne slå terninger. Vil kunne se balance. Vil kunne vinde/tabe -Projektledere vil kunne teste terningerne sandsynlighed
Precondition:	Spillerne har en tilgængelig til en jdk/jvm
Postcondition:	Terningen har genereret en total. Spilleren får et output baseret på, hvad spilleren slår. Spillerens balance bliver opdateret. Der skriftes tur til næste spiller
Main success scenario:	1. Spilleren indtaster kommando for at slå med terninger 2. Terningerne generer to tilfældige tal fra 1-6 3. De to tilfældige tal udskrives af konsollen. 4. Systemet finder en aktion, som passer til spillerens position

	5. Systemet tilføjer eller fjerner balance 6. Systemet tjekker om reglerne for spillet er opfyldt 7. Systemet skifter tur til en anden spiller 8. En spiller vinder 9. Spillet slutter
Extension:	1a. Spilleren indtaster en forkert kommando. 1. Spillet printer en error statement
Special requirements:	TBA
Technology and Data Variations list:	N/A
Frequency of occurrence:	Spillet skal kunne spilles indtil der findes en vinder

Tabel 3.1.2

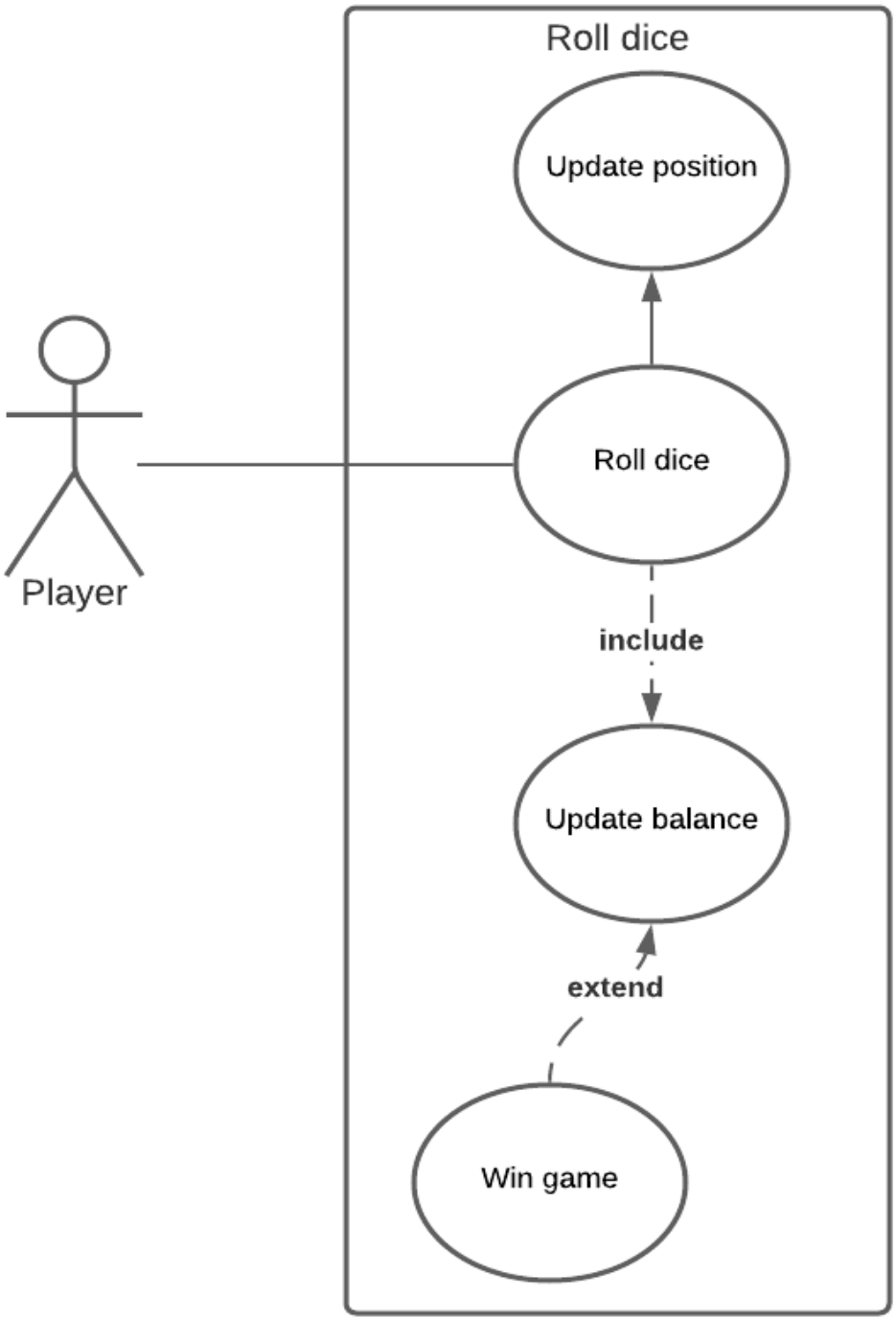
Vi har 5 ekstra use cases som ikke var centrale, men nødvendige. Disse er beskrevet briefly:

Use case	Brief use case description
StartGame	Starter spillet
GetPoosition	Giver positionen på spilleren
UpdateBalance	Opdatere balance for at tilføje eller fjerne penge fra spilleren
GetDice	Værdien af terningerne
WinGame	Spilleren vinde spillet

Tabel 3.1.3

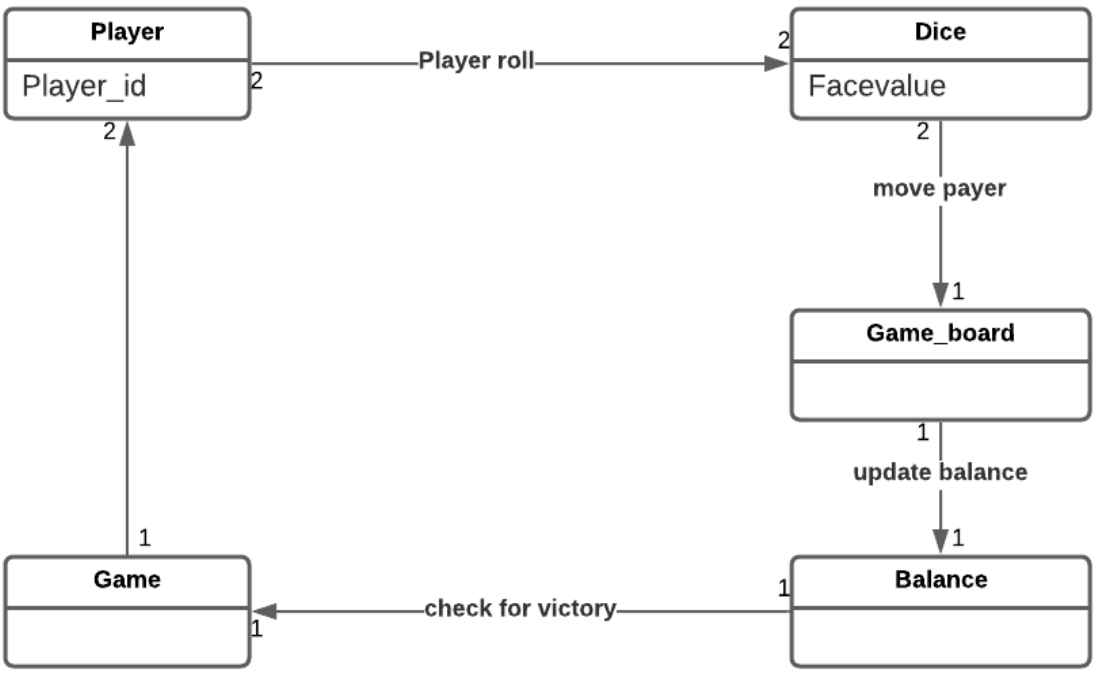
3.2 Use Case diagram

Vi har ud fra vores centrale use case lavet et use case diagram



Figur 3.2.1: Et billede af use case for RollDice.
Spilleren starter med at slå med to terninger, som opdatere spillerens placering, men samtidig også ændrer spillerens score. Der bliver også tjekket op på om spilleren har vundet.

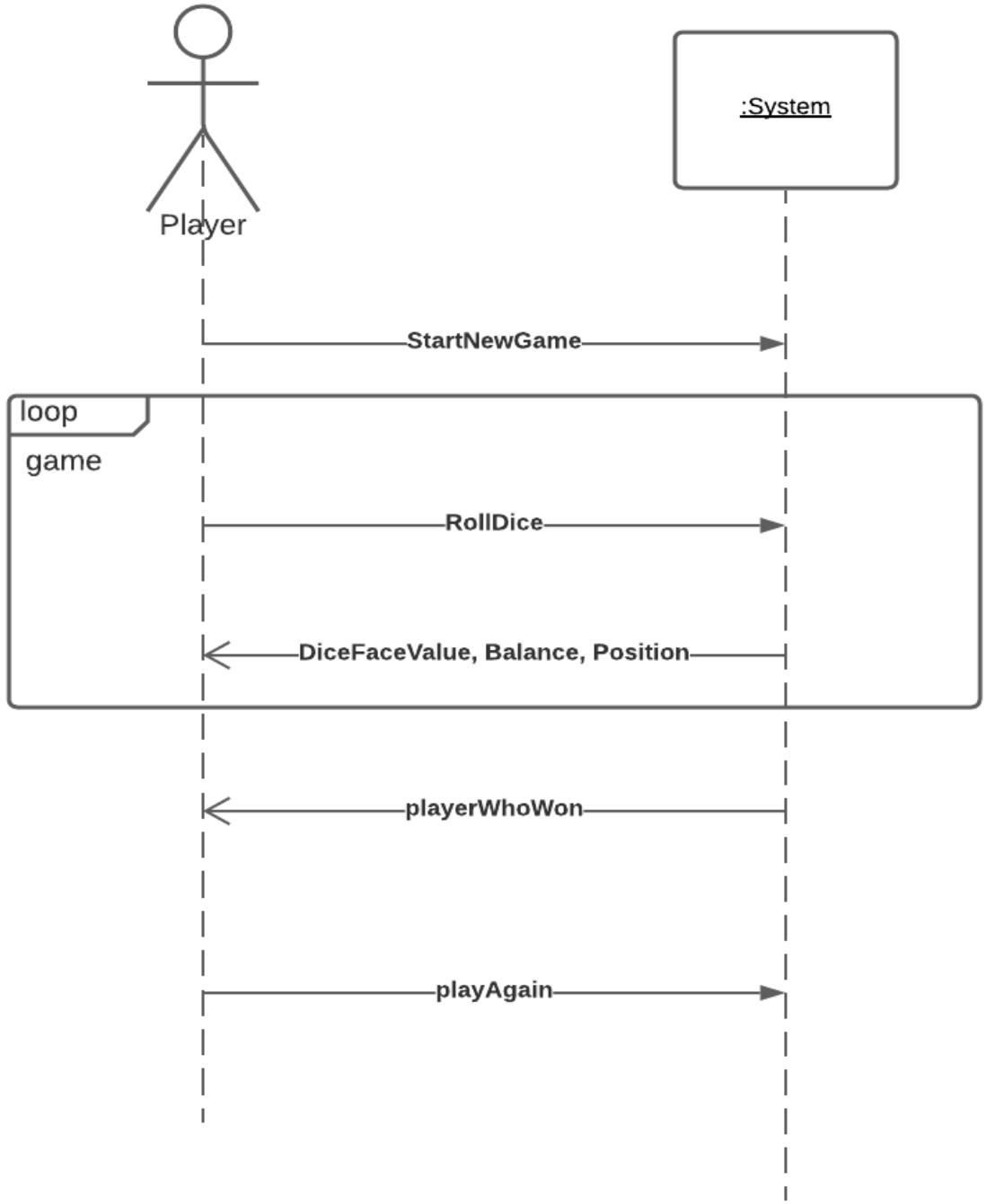
3.3 Domæne model



Figur 3.3.1: Domæne model for spillet

Her starter spilleren med at ruller 2 terninger, som rykker spilleren. Der bliver så tjekket, hvad der skal ske med player i gameboard. Der sker herefter en ændring i spillerens points og der bliver tjekket om spilleren har vundet. Til sidst bliver der skiftet spiller og sådan køre spillet i ring.

3.4 System Sekvensdiagram



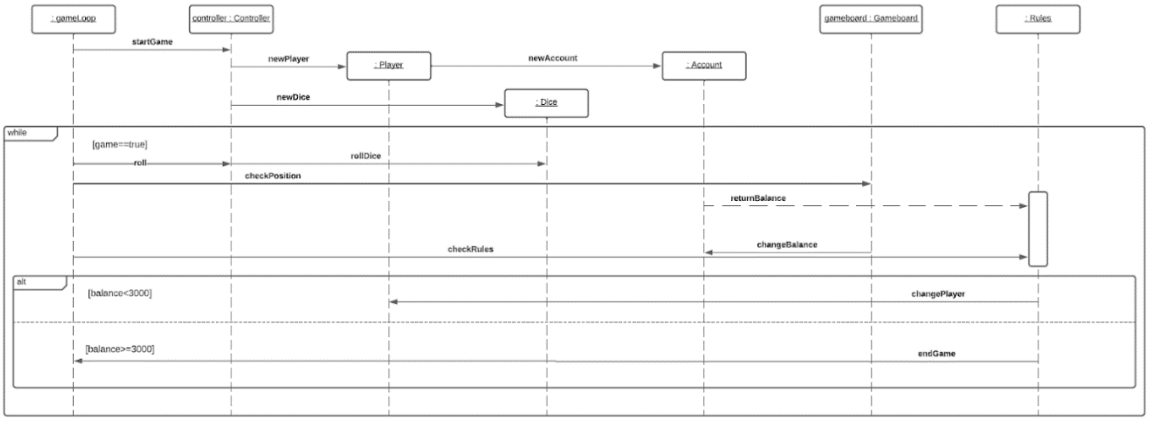
Figur 3.4.1: System sekvens diagram viser overordnet hvordan spillet fungerer.
I system sekvensdiagrammet ser vi, hvordan programmet skal forløbe. Spilleren starter spillet og slår med terningen, hvorefter systemet kører. Spilleren får at vide, hvad spilleren slog, spillerens score og spillerens position på spillepladen. Når loopet det er færdigt, skriver systemet hvem der vandt, og man kan herefter vælge at spille igen.

4 Design (Isak)

Til dette spil, er vi desværre blevet mødt af en række uheldige tilfælde og hændelser. Derfor er spillet udarbejdet efter et minimumskrav om blot at kunne køre. Vi har været igennem en række design ideer vi gerne ville have implementeret, men desværre ikke har kunne gøre (se indledningen). Derfor vil der i dette afsnit være mere fokus på hvordan vi ville have designet vores program hvis vi havde tiden og ressourcerne til det.

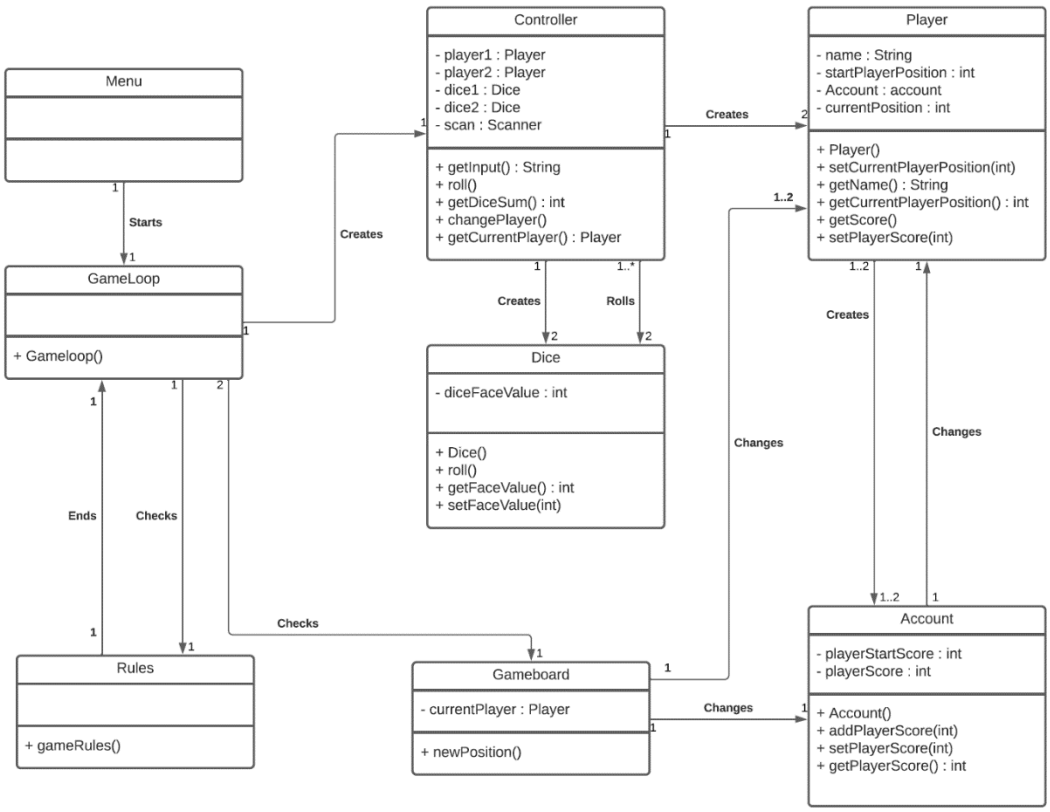
Vi ville have stræbt efter at lave en kode der har lav kobling og høj samhørighed. Dette ville vi opnå ved, at designe koden efter GRASP(General Responsibility Assignment Software Principle). Ved at designe klasser til at arbejde indenfor et bestemt område, og ikke brede sig ud, men i stedet kunne kalde metoder fra andre klasser, og skabe et ansvarsområde for hver klasse, der kun arbejder med information relevant til klassen og har dermed høj samhørighed.
I stedet for at controller-klassen laver objekter af alle objekter der skal laves når spillet starter, ville vi i stedet bruge en controller primært at styre bruger-inputs og arbejde efter GRASP med "creator", så objekterne ville blive instantieret hvor det er relevant. Koden skulle også designes til, at man let kunne skifte sprog ved brug af klasser til andre sprog med nedarvning og dermed brug af nedarvning. Samme er gældende for terningerne.

4.1 Sekvensdiagram



Figur 4.1.1: Her ser vi et sekvensdiagram af hvordan vores program fungerer. Her ses det, at controlleren er creator for alle objekter.

4.2 Design klasse diagram



Figur 4.2.1: Klassediagrammet uddyber hvad der ses i figur 4.1.1.

Her kan vi se variablerne og metoderne ikke arbejder efter lav kobling eller høj samhørighed, hvilket vi gerne ville hvis vi kunne starte forfra efter GRASP.

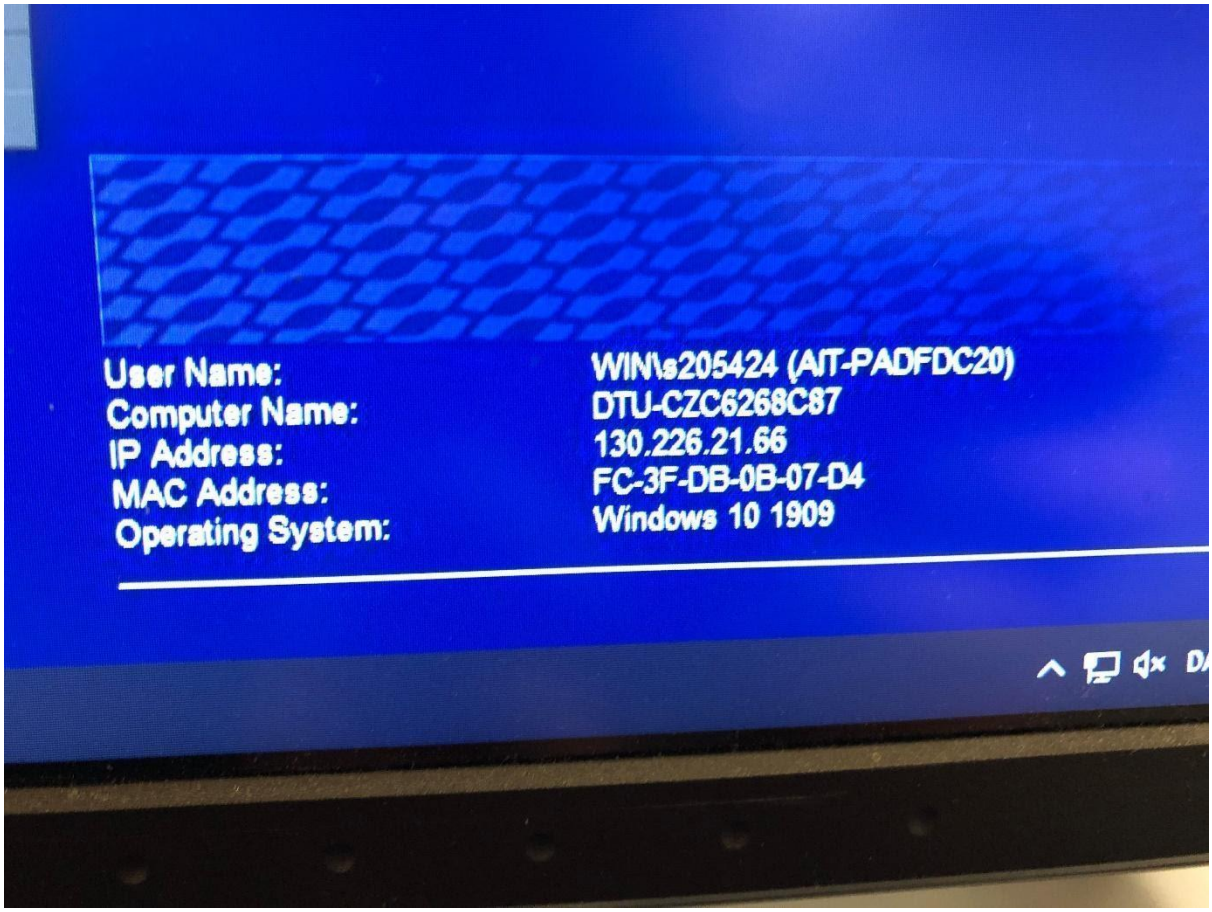
5 Implementering (Lucas)

Når Main.java filen eksekveres kaldes der på gameloop metoden i Gameloop klassen (figur 1.1). Her køres først et while loop, så længe der ikke er fundet en vinder af spillet. Dette while loop indeholder en if-sætning, som kører, hvis scanneren registrerer ”R” eller ”r” som input (som køres som tastaturinput). Herefter printes der en besked med den nuværende spiller via System.out.println(Controller.getCurrentPlayer().getName()); metoden. Herefter rulles de to terninger via Controller.roll();, som er blevet instantieret i Controller klassen. Dette producerer så en udskrift med, hvad der er blevet slået og lægger de to terningekast sammen. Herefter køres Gameboard.newPosition(); metoden som sætter den nuværende spillers position på gameboardet til noget nyt, hvorefter spillerens konto opdateres og der udskrives en tekst med den nuværende spillers point og hvilket felt, spilleren er landet på. Næstsidst tjekker Rules.gameRules(); metoden, hvorvidt en spilleren har opnået 3000 eller flere point eller, om der skal gives en ekstra tur hvis spilleren er landet på felt 10. Til sidst, hvis gameRules ikke er opfyldt, så skiftes der til den anden spiller og if-sætningen kører i loop.

```
1  package Game;
2
3  import com.sun.tools.javac.Main;
4
5  public class Gameloop {
6      static boolean game_run = true;
7      public static void gameloop(){
8          while(game_run)
9          if(Controller.getInput().equals("r")){
10             System.out.println(Controller.getCurrentPlayer().getName());
11             Controller.roll();
12             Gameboard.newPosition();
13             Rules.gameRules();
14             Controller.changePlayer();
15
16
17         }
18
19     }
20
21
```

Figur 5.1: Udskrift fra gameloop klassen, der viser den overordnede kørsel af programmet.

Ydermere er programmet testet på DTU’s databarer, hvor JUnit er tilføjet. Databarens specifikationer kan ses på figur 5.2.



Figur 5.2. Oversigt af DTU's databar specifikationer.

5.1 Snapshot af kode

Her har vi valgt at kigge på koden fra Gameboard klassen.

```
public class Gameboard {
    private static Player currentplayer;

    public static void newPosition() {
        currentplayer = Controller.getCurrentPlayer();
        currentplayer.setCurrentPlayerPosition(((currentplayer.getCurrentPlayerPosition() + Controller.GetDiceSum())%10)+1);
        int score = currentplayer.getScore();


        switch(currentplayer.getCurrentPlayerPosition())

        {
        case 1:
            System.out.println("Tile 2: Tower");
            System.out.println("Your balance is now: " + (score + 250));
            currentplayer.setPlayerScore(currentplayer.getScore() + 250);
            break;
```

Figur 5.1.2: Snapshot af Gameboard klasse og metode, som opdaterer en spillers score.

På figur 5.1.2 ses det, at newPosition() metoden modtager det nuværende spiller objekt og opdaterer spillerens position via spillers nuværende position adderet med terningernes sum modulo 10+1.

Herefter bruger vi en switch statement til at udskrive, hvilket felt spilleren nu er landet på og opdaterer spillerens score. Dette switch statement kører fra case 1 til 11, hvor der er forskellige printbeskeder og pointopdatering for hver switch case.

6 Test (Lucas)

Til det pågældende program har vi to tilsvarende test packages, der består af JUnit tests for dice klassen og account klassen:

6.1 Dice test package

Void roll() metoden:

I dice test void roll(); tester vi, hvorvidt de to terninger genererer et tal indenfor intervallet]2, 12] over 1000 kast uden afvigelser.

```
13      @Test
14      void roll() {
15          for(int i = 0; i<1000 ;i++){
16              dice1.roll();
17              dice2.roll();
18              if(Dice.getdiceSum() >2 || Dice.getdiceSum()<12 ){
19                  //do nopthing
20              }
21              else{
22                  assertEquals(Dice.getdiceSum(),10000);
23              }
24          }
25
```

Figur 6.1.1: Oversigt over void roll() JUnit test koden.

setFaceValue() metoden:

Her testes der, hvorvidt terningerne returnerer det tal, som man overfører til hvert terning objekt. F.eks. overføres 1 til dice1 og 2 til dice2, hvor de hhv. returnerer 1 og 2.

```
28      @Test
29      void setFaceValue() {
30          dice1.setFaceValue(1);
31          assertEquals(dice1.getFaceValue(),1);
32          dice2.setFaceValue(2);
33          assertEquals(dice2.getFaceValue(),2);
34      }
```

Figur 6.1.2: Oversigt over setFaceValue() JUnit test koden.

getdiceSum() metoden:

Her testes der, hvorvidt metoden for at lægge de to terningeobjekters værdi sammen er overholdt. Der testes derved, at man giver dice1 heltallet 6 og dice2 heltallet 6, hvorefter getdiceSum() returnerer tallet 12.

```
35      @Test
36      void diceSum() {
37          //Dice.dicesum();
38      }
39
40      @Test
41      void getdiceSum() {
42          dice1.setFaceValue(6);
43          dice2.setFaceValue(6);
44          assertEquals(Dice.getdiceSum(),12);
45      }
```

Figur 6.1.3: Oversigt over getDiceSum() JUnit test koden.

6.2 Account test package

addPlayerScore() metoden:

Her testes, hvorvidt playerScore bliver opdateret med den korrekte værdi udregnet af GameBoard. Her tilføjes 100 til playerScore, hvorefter, der testes om 1100 returneres fra getPlayerScore(), da startPlayerScore er lig 1000. Herefter addes -200, hvor der testes om der getPlayerScore() returnerer 900.

```
6       Game.Account test = new Account();
7       @org.junit.jupiter.api.Test
8       void addPlayerScore() {
9           test.addPlayerScore( incomingGameBoardScore: 100);
10          assertEquals(test.getPlayerScore(),1100);
11          test.addPlayerScore( incomingGameBoardScore: -200);
12          assertEquals(test.getPlayerScore(),900);
13      }
14
```

Figur 6.2.1: Oversigt over addPlayerScore() JUnit test koden.

setPlayerScore() metoden:

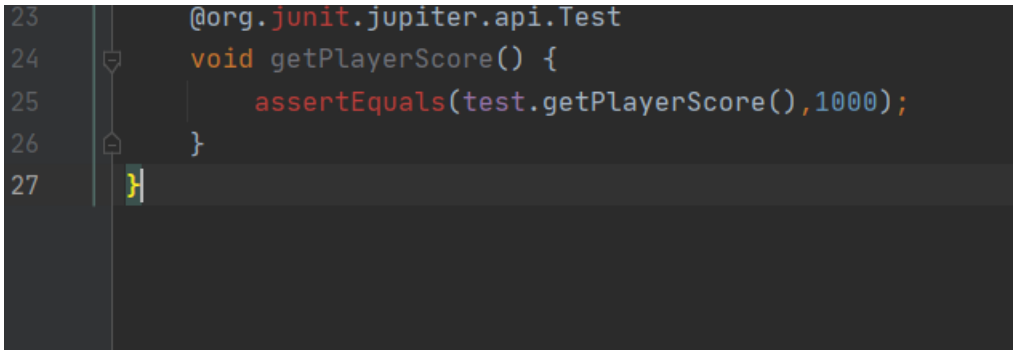
Her testes, hvorvidt en player's score kan tage både positive og negative tal ved at indsætte enten 10 eller -10 som værdi for setPlayerScore().

```
16      void setPlayerScore() {
17          test.setPlayerScore(10);
18          assertEquals(test.getPlayerScore(),10);
19          test.setPlayerScore(-10);
20          assertEquals(test.getPlayerScore(),-10);
21      }
```

Figur 6.2.2: Oversigt over setPlayerScore() JUnit test koden.

getPlayerScore() metoden:

Her testes, hvorvidt en spiller starter ud med scoren 1000 ved at sammenligne getPlayerScore() med værdien 1000.



Figur 6.2.3: Oversigt over getPlayerScore() JUnit test metoden.

7 Projektplanlægning (Isak)

Figur 7.1: En oversigt af vores Unified Process arbejdsplanlægning.

8 Konklusion (Louis)

Vi fik stillet en opgave, hvor vi skulle lave et spil, hvor to eller flere spillere kunne rykke rundt på en spilleplade ved at kaste 2 terninger efter tur. Alt afhængig af hvor spillerne lander på pladen kan der fjernes eller tilføjes points for den enkelte spiller. Vinderen er spilleren, der har 3000 points først.

Opgaven blev desværre ikke løst efter planen, da vi havde problemer undervejs som beskrevet i indledningen. Vi nåede ikke alle vores mål, men prioriterede at fokusere på det, som vi tænkte vi lærte mest ved at lave. Vi prøvede at nå så mange mål som muligt og kom ud med et fungerende spil, men vi ville gerne have gjort bedre og opfyldt alle kravene. Vi ved nu, hvordan vi skal ændre vores proces til næste gang for at få et bedre resultat, bl.a. ved at overholde tidsplanen for de enkelte delelementer og have mere kommunikation omkring, hvor langt de enkelte gruppemedlemmer er nået.

9 Ordforklaring (Daniel)

Ord eller tegn	**Forklaring**
GIT	Et værktøj til at versionsstyre projekter, der består primært af simple tekstfiler.
GRASP	"General Responsibility Assignment Software Patterns" Er nogle principper man bruge til at at tildele Responsibility til Classes og objekter i objekt orienteret design (OOD).
IntelliJ IDE	Betyder IntelliJ Integrated Development Environment. Et værktøj, der bruges til at udarbejde softwareprojekter understøttet af en grafisk brugergrænseflade.
Java	Programmeringssprog.
Unified Process	En itterativ design metode til software systemer.
Use Case	En Use Case er en analyse af hvordan brugeren kommer til at bruge programmet.
UTF-8	Betyder Unicode Transformation Format 8-bit. Et encoding system, der tillader op til 1.112.064 forskellige symboler og tegn.