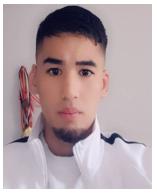
CDIO Del 1 - Gruppe 18

Anders Lønborg - s185100 

Rahmat Kazimi - s185012 

Mohamed Anshur - s185115 

Rasmus Leiberg - s185090 

Nikolaj Wassmann - s185082 

Johannes Piil - s185114 

Kurser: 02313, 02314/02312 og 02315

Afleveringsfrist: 12-10-2018, kl. 18:00

Timeregnskab

Nikolaj Wassmann:

- 70 min programmering (Main program)
- 45 min Implementeringsafsnit
- 10 min Litteraturafsnit
- 15 min Projektforløb
- 10 min Testafsnit
- 20 min Designklasse diagram
- 20 min Designsekvens diagram

Johannes Piil:

- 220 min Programmering (Bugfix + Features).
- 30+ min test

Anders Lønborg:

- 160 min Programmering (Bugfix + Features + JUnit test)
- 30 min Implementering
- 10 min Unit test

Mohamed Anshur:

- 45 min Indledning og Abstract
- 150 min Krav (funktionelle krav, ikke funktionelle krav, domænemodel)
- 120 min Analyse (Use-case diagrammer, use-case beskrivelser)
- 20 min Konklusion

Rasmus Leiberg:

- 60 min test.

Rahmat Kazimi:

- 45 min Indledning og Abstract
- 150 min Krav (funktionelle krav, ikke funktionelle krav, domænemodel)
- 120 min Analyse
- 20 min Konklusion

Abstract

In this CDIO project, we intend to develop a dice game meant for two players, where the customer's wishes have been granted. The system fulfills a list of requirements set by the customer, which we then modify throughout the use of use cases and diagrams, which will illustrate the system's functionality. We intend to test the program's functionality and come to a conclusion that will satisfy both the programmers and the customer.

Indholdsfortegnelse

Indledning

I dette CDIO-projekt vil vi udvikle et terningspil, som skal kunne anvendes på DTU's databarer. Terningspillet bliver udviklet på basis af tre kursusfag: "Udviklingsmetoder til IT-systemer", "Indledende Programmering" og "Versionsstyring og Testmetoder". Vi optræder som programmører for spilfirmaet IOOuterActive, hvor vi står for at udvikle spillet. Vi har udviklet terningespillet baseret på kundens vision, hvor vi har set på de forskellige krav og modeller. Vi har benyttet os af UP (Unified Process) til at analysere og producere programmet. Til slut tester vi programmet i detalje for at sikre kundens krav opfyldes.

Krav

Kundens vision for dette projekt er at terningspillet skal kunne bruges på maskinerne på DTU's databarer, hvilket vil sige at det skal kunne køre på windows computerer. Kunden vil gerne have at terningespillet skal kunne spilles af 2 personer, som skal kunne slå et raflebæger og dermed få summen af de to terninger med det samme. Derudover har kunden stillet nogle krav, som kun skal løses hvis der er ressourcer til det, hvilket vi har gjort. Vi ved, hvad kundens vision er og hvordan personen gerne vil have dette program til at fungere, dermed kan vi udlede kravene alt efter hvad kunden søger i forhold til spillets funktioner. Ud fra den korte beskrivelse vi får, har vi lavet en kravspecifikation[1].

Kravspecifikationer

Ud fra kundens vision har vi så udledt denne kravspecifikation:

K1: Spillet skal kunne bruges på DTU's maskiner.
K2: Spillet skal kunne spilles mellem to personer.
K3: Terningerne skal have 6 øjne i alt.
K4: Man kaster to terninger.

[1] Lektion 2, Udviklingsmetoder til IT-systemer

K5: Resultatet skal kunne ses med det samme.

K6: Spilleren skal slå to ens for at vinde spillet, efter at man har opnået 40 point.

K7: Spillet skal være egnet til alle normale mennesker.

K8: Raflebægeret skal testes for at se, om det virker hen over 1000 kast.

K9: Man skiftes om at kaste hver runde.

K10: Antal point spillerne opnår hvert kast, bliver gemt.

K11: Spilleren mister alle sine point, hvis spilleren slår to 1'ere.

K12: Spilleren får en ekstra tur hvis spilleren slår to ens.

K13: Spilleren kan vinde spillet ved at slå to 6'ere, hvis spilleren også i forrige kast slog to 6'ere uanset om det er på ekstra kast eller i forrige tur.

Korrektion af kravene:

Vi har lavet nogle ændringer i vores kravspecifikation i forhold til kundens vision. Da kundens vision og beskrivelse var ufuldstændig, bliver vi nødt til at lave nogle ændringer i forhold til kravspecifikationen, så vores krav bliver målbare. Her er bl.a. nogle de ting, som vi mente kundens vision manglede, for at sikre en høj kvalitet.

1. Vi fik ikke at vide om, at man skulle skifte tur efter hvert kast.
2. Vi fik ikke at vide om pointene blev gemt.

Funktionelle krav

De funktionelle krav er de krav, som er nødvendig for systemet og spillets funktionalitet. De er essentielle for programmets funktionalitet og af den grund kan ikke undværes uden at påvirke spillets "gameplay" markant. Her er vores funktionelle krav[2]:

K2: Spillet skal kunne spilles mellem to personer.

K3: Terningerne skal have 6 øjne i alt.

K4: Man kaster to terninger.

[2] Slides fra Lektion 2, Udviklingsmetoder til IT-systemer

K6: Spilleren skal slå to ens for at vinde spillet, efter at man har opnået 40 point.
K9: Man skiftes om at kaste hver runde.
K10: Antal point spillerne opnår hver runde, bliver gemt.
K11: Spilleren mister alle sine point, hvis spilleren slår to 1'ere.
K12: Spilleren får en ekstra tur hvis spilleren slår to ens.
K13: Spilleren kan vinde spillet ved at slå to 6'ere, hvis spilleren også i forrige kast slog to 6'ere uanset om det er på ekstra kast eller i forrige tur.

Ikke-funktionelle krav

De ikke funktionelle krav er ikke så essentielle for spillets gameplay, men bare mindre krav som involverer ting såsom brugervenlighed, layout, svartider osv.. I dette tilfælde ville det være K1, K5, K7 og K8, da de ikke direkte påvirker spillet, men er mindre detaljer som er nødvendige for kundens tilfredsstillelse[3].

K1:Spillet skal kunne bruges på DTU's maskiner.
K5: Resultatet skal ses med det samme.
K7: Spillet skal være egnet til alle mennesker.
K8: Raflebægeret skal testes for at se, om det virker hen over 1000 kast.

Det er ikke altid med sikkerhed at ikke-funktionelle krav kan løses, f.eks. hvis kunden ønskede at summen af terningekastet vises endnu hurtigere end hvad det kan lade sig at gøre for programmet, derfor er ikke-funktionelle krav ikke så vigtige for selve programmet, men mere for kundens tilfredsstillelse, da de ikke altid kan løses også selvom man har ressourcerne til det.

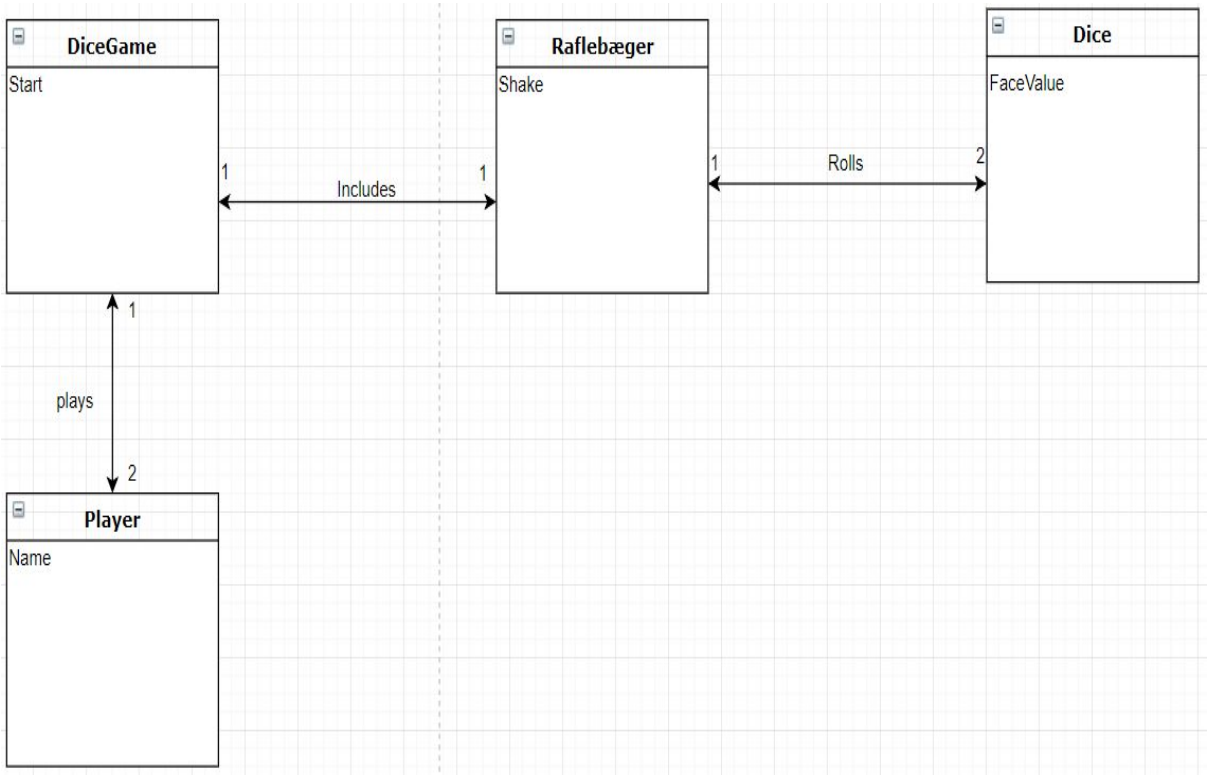
[3] Slides fra Lektion 2, Udviklingsmetoder til IT-systemer

Analyse

Vores analysedel består bl.a et use case diagram, domænemodel og en analyse af vores centrale use-case, hvor vi anvender brief, casual og fully dressed beskrivelser for at kigge dybere på vores use-cases.

Domænemodel

En domænemodel er en model af spillet set med virkelighedens "briller", dvs. at man finder objekter i virkeligheden og ser hvordan de forskellige objekter hænger sammen.

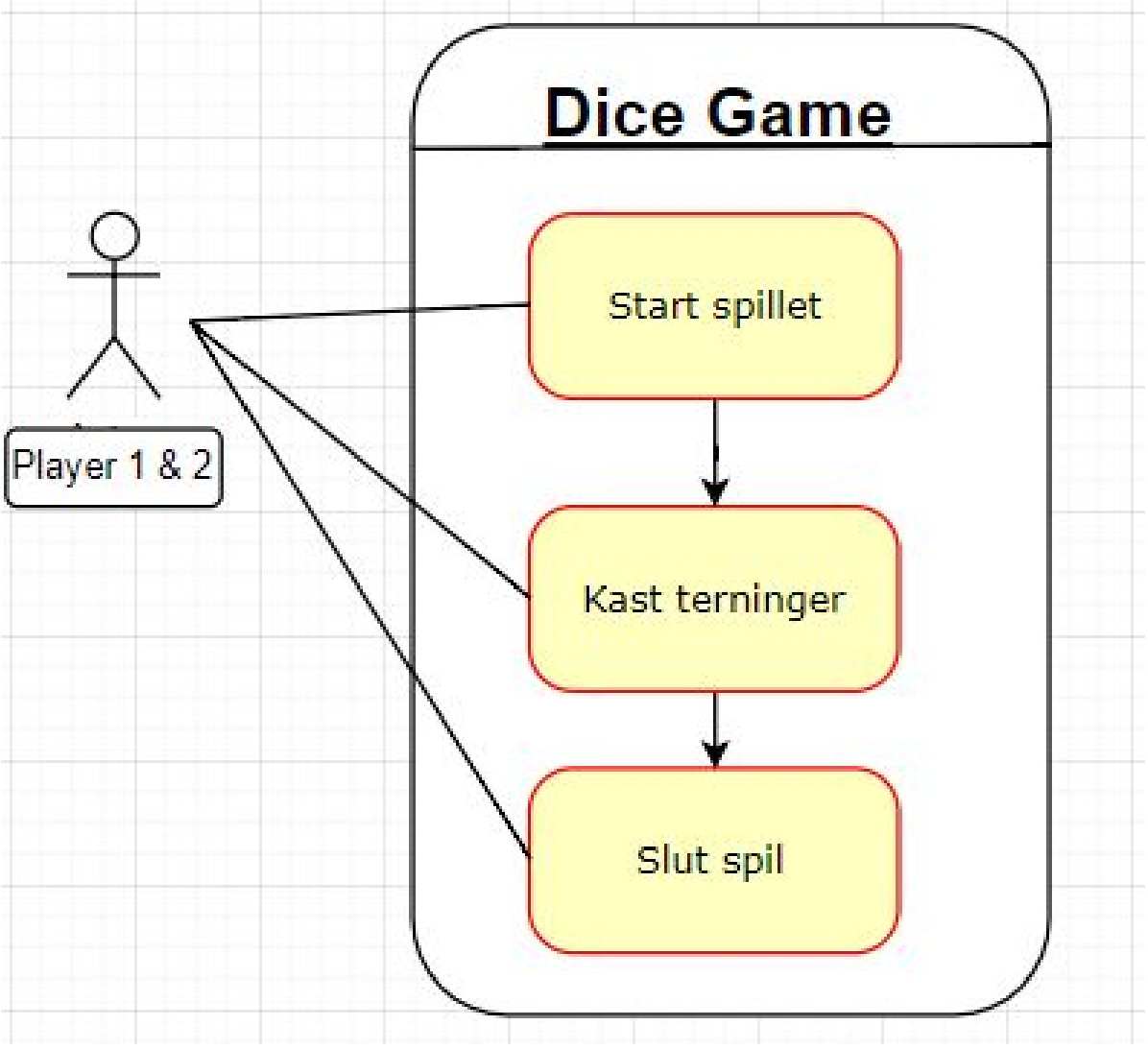


Her har vi lavet en Domænemodel, hvor vi overordnet forklarer hvordan vores system virker uden at være alt for detaljeret. Den viser vores begrebsklasser, sammenhænge og evt. meget vigtige attributter[4].

[4] Slides fra Lektion 1, Udviklingsmetoder til IT-systemer

Use-case diagram

Et use-case diagram er et diagram, hvor vi illustrere samspillet mellem aktørerne og use-cases, og hvordan de interagerer med hinanden. Det er med til at danne et billede af, hvordan de forskellige use-cases hænger sammen og hvordan spillets funktioner virker i et overordnet billede[5].



Her har vi et use-case diagram, hvor følgende handlinger bliver foretaget af systemet alt efter aktørens dvs. spillernes interaktioner med det. Spilleren starter spillet ---> Spilleren kaster terningen -----> Spillet slutter når spilleren har fået den ønskede værdi og har opfyldt kravene for at vinde spillet, hvilket afslutter terningspillet.

[5] Slides fra Lektion 3, Udviklingsmetoder og IT-systemer

Use-case beskrivelser

Her bliver vores use cases forklaret via brief description, hvor vi kommer med et enkelt tekstafsnit om det primære scenarie[6].

- Start spillet

Brief: Spilleren starter spillet, hvorefter det hele fungere som det skal.

- Kast terninger

Brief: Spilleren kaster to terninger, hvorefter de får to resultater lægges sammen.

- Slut spillet

Brief: Spilleren har vundet spillet, og slukker så får spillet.

Central use case:

Vi har valgt en central use case og beskriver den casual. I denne situation vælger vi ”kast terningerne”, hvilket vi mener vil være oplagt angående en casual beskrivelse af en central use case[7].

  - Main success scenario: Spilleren kaster terningerne og får to værdier, hvor summen af dem bliver til point.

  - Alternative scenario 1: Der bliver kun kastet med en terning.
  - Alternative scenario 2: Den ene spiller forlader midt i spillet.
  - Alternative scenario 3: Spillet crasher.

Vi vil nu lave en fully dressed description af den centrale use case ”Kast terningerne”:

- Use case name:
  - “Kast terningerne”

[6] Slides fra Lektion 3, Udviklingsmetoder til IT-systemer
[7] Slides fra Lektion 3, Udviklingsmetoder til IT-systemer

- Level
    - Målet er at kunne kaste to terninger, og få en værdi.

- Primær aktør:
    - Spillerne

- Interessenter:
    - Kunden: Vil gerne have et terningspil, som kan bruges på DTU's databarer.
    - IOOuterActive: Skal lave et terningespil, som opfylde kundens krav.
    - Spillerne: Vil gerne spille spillet.

- Preconditions:
    - For at use casen kan udføres, skal spillet kunne bruge et ræflebæger, som kaster to terninger.

- Postconditions:
    - Spilleren skal kunne kaste to terninger, som giver en værdi.

- Succes Scenarie/Basic flow:
    - Spillerne starter spillet.
    - Spillet beder 1.spiller om at kaste terningerne.
    - Systemet genererer et tilfældigt tal mellem 1 og 6, for begge terninger.
    - Summen af de to terninger, omdannes til point.
    - Spiller beder 2.spiller om at kaste terningerne.
    - Systemet genererer et tilfældigt tal mellem 1 og 6, for begge terninger.
    - Summen af de to terninger, omdannes til point.
    - Systemet gentager de ovenstående punkter indtil en vinder er fundet.

- Alternativ flow:
    - Der bliver kun kastet en terning.
        - Spilleren genstarter spillet, men der er stadig kun en terning.
        - Spilleren ringer til udviklerne, og de retter fejlen.
        - Spillet opdateres.

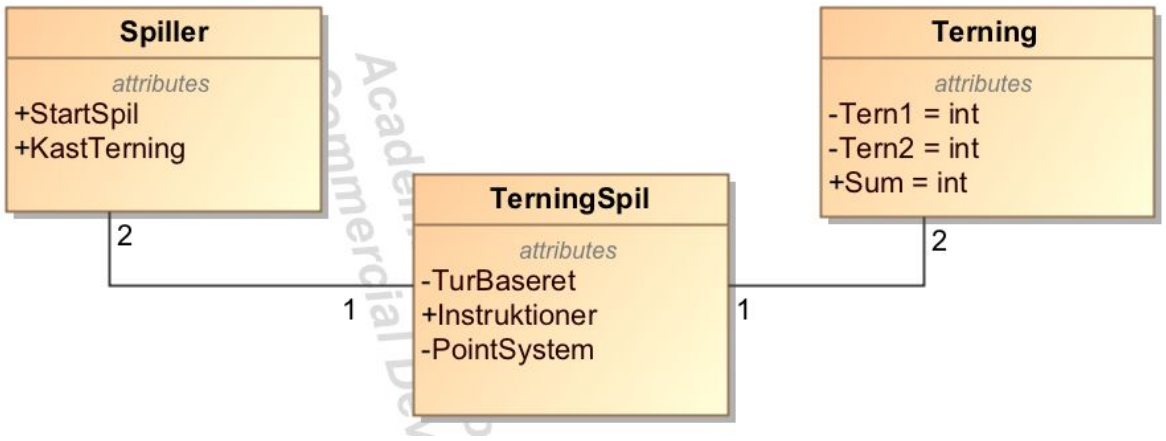
    - Den ene spiller forlader spillet, men spillet er i gang.
        - Spilleren prøver at finde en ny medspiller.
        - Spilleren spiller alene.
        - Spilleren slukker spillet.

- Spillet går i stå.
  - Spillet genstarter spillet, men spillet virker stadig ikke.
  - Spilleren ringer til udviklerne, og de retter fejlen.
  - Spillet opdateres.

Implementering

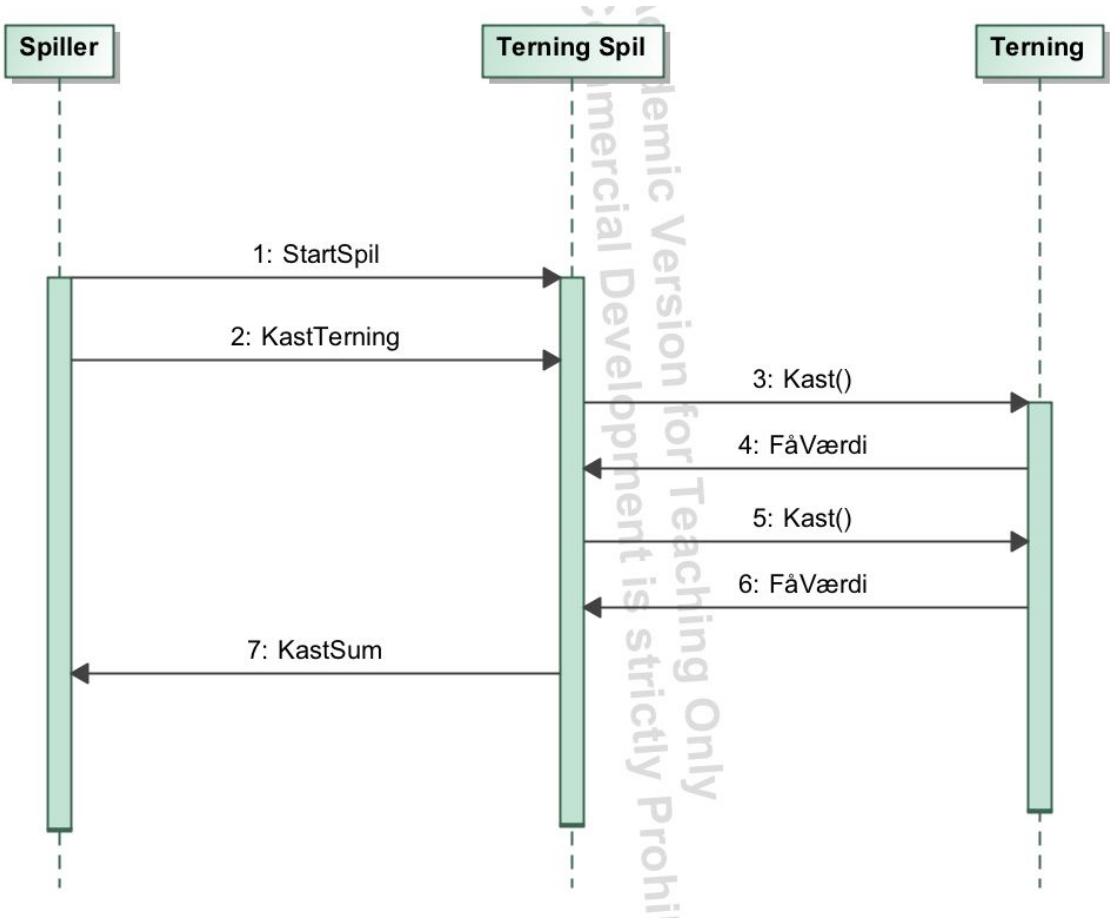
Designklasse diagram

Under implementeringen af javakoden, lavede vi et designklasse diagram.



Ovenfor ses et designklasse diagram udtrykt ved UML (Unified Modelling Language), med andre ord programmeringssprog. Designklasse diagrammet viser hvordan de forskellige “Klasser” benytter de implementerede funktioner, til at få systemet til at køre som det skal ifl. De funktionelle krav. Som man kan se så har “Spiller” klassen en multiplicitet på 2, dvs. at der skal 2 spillere til at spille ét Terningspil. Det samme gælder for “Terning” klassen. Der skal 2 terninger til ét TerningSpil.
Udover at lave et designklasse diagram, har vi også lavet et designsekvens diagram. Et designsekvens diagram viser hvordan de forskellige funktioner henter information fra systemet, dvs. hvordan Spiller funktionen henter information fra f.eks. TerningSpil.

Sekvensdiagram



Her ses det at “Spiller” sender information til “TerningSpil” om at kaste terningen. “TerningSpil” henter informationen fra klassen “Terning” og der kastet en tilfældig værdi mellem 1-6. Når dette er gjort, giver “TerningSpil” summen af kastet til “Spiller”.

Under implementeringen af javakoden, udvalgte vi nogle centrale punkter af software. Disse eksempler ses nedenfor:

```java
public class Terning {

    // Kaster terning
    public int kast() {
        float t1 = (float) Math.random();
        float t2 = t1 * 5;
        int t3 = Math.round(t2);
        return t3 + 1;
    }
```

Dette eksempel viser vores “Terning” klasse. Denne klasse er central for koden, da den definerer vores terning. Uden denne klasse vil programmet ikke kaste terningen. Det ses at der er defineret en “public int” dvs. en offentlig variabel “kast”. Når “kast” skrives i Main/TerningSpil, så vil den følgende kode blive eksekveret, som udvælger en tilfældig værdi mellem 1-6.

Nu hvor vi har vores klasse “Terning”, skal vi have en måde at benytte den på. Det gør vi ved at gøre følgende:

```java
//Selve spillet
while ((k1 != k2 && k3 != k4 || (point1 - (kastsum1)) < MAX && (point2 - (kastsum2)) < MAX) && !to6_1 && !to6_2)
    System.out.println();
    System.out.println("Spiller nr.1 kaster");
    do {
        forkert = false;
        System.out.println("Tast 2 for at kaste terningen: ");
        k1 = tern.kast();
        k2 = tern.kast();
        String kast = Spil.nextLine();
        if (kast.equals("2")) {  //tjekker for korrekt input
            kastsum1 = k1 + k2;
            point1 += kastsum1;
            System.out.println(k1 + " " + k2);
            if (k1 + k2 == 2) { //tjekker om slaget var 2 ettere
                System.out.println("Spiller 1. mister alle sine point.");
                point1 = 0;
            } else {
                System.out.println("Summen af kastet er: " + kastsum1);
            }

            System.out.println("Spiller nr.1 har " + point1 + " point");
```

Ovenfor vises loopet for spiller 1. Dette fortæller os at spillet bliver ved så længe ingen af spillerne har vundet. Her kommer der også information ud til brugerne om hvem der skal kaste terningerne, hvad de skal gøre for at kaste terningerne, og hvor mange points hver spiller har. Det er derfor, at det her er centralt for vores kode at

have klassen “Terning” ellers ville vi jo ikke have en måde at kaste terninger eller tælle antal points man får.

Test

Unit-tests

For at tjekke om programmet fungere over mange forskellige kast, har vi lavet en JUnit test (en automatiseret test), hvor vi tester om at vi kan kaste en terning 1000 gange, uden at terningens lander på en værdi der ikke ligger inden for 1 til 6.

```
import org.junit.Test;

import static org.junit.Assert.assertTrue;

public class TerningTest {

    Terning T = new Terning();

    @Test
    public void kast() {
        for (int i = 0; i < 1000; i++) {
            int tal = T.kast();
            assertTrue( condition: tal > 0 && tal < 7);
        }

    }
}
```

Dette er et udprint af vores kildekode fra selve programmet, som viser en JUnit test af funktionen “kast”.

Vi har valgt at lave en JUnit test der tester resultater af 12000 terningekast (6000 terningpar), og ser om sandsynligheden for at slå 2-12 er lige stor.

```
2 Øjne = 115
3 Øjne = 510
4 Øjne = 886
5 Øjne = 1461
6 Øjne = 1895
7 Øjne = 2172
8 Øjne = 1967
9 Øjne = 1385
10 Øjne = 974
11 Øjne = 493
12 Øjne = 142
Ens antal øjne = 2170
```

Denne JUnit test fortæller os at sandsynligheden for at slå 2-12 ikke er lige stor, da Math.random i klassen “Terning” ikke er 100% random, da der bliver rundet op og ned, dermed er sandsynligheden for at slå 1 og 6, med en terning mindre og derfor er sandsynligheden for at slå 2-12 også mindre. Her ses det at sandsynligheden for at slå 2,3,4,10,11 og 12 væsentlig mindre en de mest hyppige værdier nemlig 5,6,7,8 og 9. Dog generelt skal sandsynligheden for at slå f.eks. 7 være større end 2, da der er 6 forskellige måder at slå 7, hvor der ved 2 kun er 1 mulighed. Det passer dog godt sammen, at der er lige stor hyppighed for 7 øjne og 2 ens, eftersom begge burde have chance på 1/6 for at blive slået. Nedenfor ses en tabel med de forskellige måder at slå værdier på.

	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

Test-cases

For at teste om kravene opfyldes af programmet, har vi valgt at lave Test-cases på alle de funktionelle krav dvs. vi har manuelt testet alle krav for at se om de bliver opfyldt og udtrykt det med “positiv” og “negativ” test.

K1. Spillet skal kunne bruges på DTU’s maskiner.

Positiv test:

Spillet kan godt bruges på DTU’s maskiner.

Negativ test:

Der er ingen negativ test af dette krav.

K2. Spillet skal kunne spilles mellem 2 personer.

Positiv test:

Som kravet beskriver, kan spillet spilles mellem 2 personer.

Negativ test:

Spillet kan derimod ikke spilles af en enkelt person.

K3. Terninger skal have 6 øjne i alt.

Positiv test:

Terningerne har 6 øjne, altså 6 forskellige udfald.

Negativ test:

At slå 1 eller 6 sker mindre hyppigt end andre udfald, pga Math.random.

K4. Man kaster to terninger.

Positiv test:

Programmet finder 2 værdier mellem 1-6 som den skal.

Negativ test:

Intet fundet ved negativ test.

K5. Resultatet skal ses med det samme.

Positiv test:

Resultatet bliver vist lige efter et kast er blevet initieret.

Negativ test:

Intet fundet ved negativ test.

K6. Spilleren skal slå to ens for at vinde spillet, efter at man har opnået 40 point.

Positiv test:

Man vinder første spillet hvis man slår 2 ens efter at have nået 40 point.

Negativ test:

Et potentielt negativ er at hvis efter en spiller er nået 40 point, og slår to 1'ere, er at spilleren så mister alle sine point siden af vinder erklæringen først står til sidste i koden. Så alt efter om det er intentionen at det skal virke sådan her kan det være et negativ.

K7. Spillet skal være egnet til alle normale mennesker.

Positiv test:

Hvis spillets regler bliver overholdt, fungere spillet som det skal, og ved forkert input printes en fejlmeddelelse.

Negativ test:

Intet fundet ved negativ test.

K8. Raflebægeret skal testes for at se, om det virker hen over 1000 kast.

Det her krav bliver dækket i unit tests, som vises i afsnittet før.

K9. Man skiftes om at kaste hver runde.

Positiv test:

Spillerne skiftes om at kaste hver runde.

Negativ test:

Intet fundet ved negativ test.

K10. Antal point spillerne opnår hver kast, bliver gemt.

Positiv test:

Summen af hvert kast bliver gemt.

Negativ test:

Intet fundet ved negativ test.

K11. Spilleren mister alle sine point, hvis spilleren slår to 1'ere.

Positiv test:

Alle point går tabt hvis spilleren slår to 1'ere.

Negativ test:

Intet fundet ved negativ test.

K12. Spilleren får en ekstra tur hvis spilleren slår to ens.

Positiv test:

Spilleren får en ekstra tur hvis der bliver slået to ens.

Negativ test:

Intet fundet ved negativ test.

K13. Spilleren kan vinde spillet ved at slå to 6'ere, hvis spilleren også i forrige kast slog to 6'ere uanset om det er på ekstra kast eller i forrige tur.

Positiv test:

Spillet slutter og en vinder bliver erklæret hvis der bliver slået to 6'ere to gange i træk.

Negativ test:

Intet fundet ved negativ test.

Projektforløb

Under projektforløbet blev vi enige om at opdele projektet i to dele, nemlig analysedelen og design/implementerings-delen. Vores gruppe blev inddelt i to grupper, hvor hver gruppe stod for at få lavet sin del til tiden, og dette blev overholdt under hele projektet. Det vil sige at vores projektforløb blev udført som forventet/planlagt.

Konklusion

Vi har indset mange ting i løbet af dette projekt ift. programmering, design og krav som er essentielle værdier som kræves af en softwareudvikler. Vores job i denne opgave handlede primært om at udvikle et terningsspilsprogram, hvilket vores kunde havde diverse ønsker til. Disse ønsker havde vi forvandlet til relevante krav, som vi dannede som et grundlag for vores endelige produkt - nemlig terningspillet. I vores bestræbelse om at udvikle et kompetent program, fremstillede vi mange diagrammer, use-cases og test-cases i håbet om, at vores program ville være til kundens tilfredsstillelse, hvilket vi ender med. Vi anvender viden som vi har tilegnet os fra tre forskellige kurser ved navn Versionsstyring og testmetoder, Udviklingsmetoder til IT-systemer og Indledende Programmering. Vi har under udviklingen af programmet opfyldt kundens funktionelle såvel som ikke-funktionelle krav.

Dette projekt minder også meget om vores ugeopgaveprojekt som vi b.la. har i vores kursus Udviklingsmetoder til IT-systemer, da vi designer vores opgaves analysedel på baggrund af det. Vi gør brug af forskellige former for use-case beskrivelser, som vi også bruger på vores ugeopgaveprojekt såsom casual og brief beskrivelser, og fully dressed. Her kan vi se at det er essentielt for softwareprojekter at benytte dybdegående analysering for at sikre et højkvalitets softwareprogram.

Litteratur

“Java Software Solutions - Foundations of Program Design” af John Lewis og William Loftus
“Applying UML and Patterns - An introduction to Object-Oriented Analysis and Design and Iterative Development” af Craig Larman
Forelæsningsslides fra 02313 (Udviklingsmetoder til IT-Systemer) af Ian Bridgwood
Forelæsningsslides fra 02315 (Versionsstyring og Testmetoder) af Sune Nielsen

Bilag

Nedenfor ses relevante bilag for opgaven

Bilag 1

Kildekode: TerningSpil.java, Terning.java & TerningTest.java.

```
import java.util.Scanner;

public class TerningSpil {
   public static void main(String[] args) {

       // Scanner, metode og variable defineret.
       Terning tern = new Terning();
       Scanner Spil = new Scanner(System.in);
       final int MAX = 40;
       int kastsum1 = 0, point1 = 0, k1 = 0, k2 = 0, gemk1 = 0, gemk2 = 0; //Spiller 1,
k = kast
       int kastsum2 = 0, point2 = 0, k3 = 0, k4 = 0, gemk3 = 0, gemk4 = 0; //Spiller 2,
gemk = forrige kast
       boolean to6_1 = false, to6_2 = false; //2 6'ere i træk
       boolean forkert; //fejlinput

       // Hvis spiller 1 taster 2, så vil programmet kaste terningen, og lægge de to
slået værdier sammen

       //Instukser til Spillerne
       System.out.println("Dette er et terning spil mellem 2 personer");
       System.out.println("Hver runde lægges summen af ens slag til ens point");
       System.out.println("Man mister alle sine point, hvis man slår 2*1, men slår man 2
ens, får man et ekstra slag");
       System.out.println("Den som først slår 2 ens, efter at have 40 point vinner");
       System.out.println("Ellers kan man også vinde, hvis man slår 2 6'ere, 2 gange i
træk");

       //Selve spillet
       while ((k1 != k2 && k3 != k4 || (point1 - (kastsum1)) < MAX && (point2 -
(kastsum2)) < MAX) && !to6_1 && !to6_2) {
           System.out.println();
           System.out.println("Spiller nr.1 kaster");
           do {
               forkert = false;
               System.out.println("Tast 2 for at kaste terningen: ");
               k1 = tern.kast();
               k2 = tern.kast();
               String kast = Spil.nextLine();
               if (kast.equals("2")) {  //tjekker for korrekt input
                   kastsum1 = k1 + k2;
                   point1 += kastsum1;
```

```
                    System.out.println(k1 + " " + k2);
                    if (k1 + k2 == 2) { //tjekker om slaget var 2 ettere
                        System.out.println("Spiller 1. mister alle sine point.");
                        point1 = 0;
                    } else {
                        System.out.println("Summen af kastet er: " + kastsum1);
                    }

                    System.out.println("Spiller nr.1 har " + point1 + " point");

                    //Tjekker for 4 6'ere inden for 2 kast af samme spiller
                    if (gemk1 + gemk2 == 12 && k1 + k2 == gemk1 + gemk2) {
                        to6_1 = true;
                    }
                    gemk1 = k1;
                    gemk2 = k2;

                    if (k1 == k2 && MAX > point1 && !to6_1) { //specialt udprint ved
ekstra slag
                        System.out.println();
                        System.out.println("Spiller 1. får et ekstra slag");
                    }
                } else {
                    System.out.println("Forkert input...");
                    forkert = true;
                }
            } while (k1 == k2 && MAX > point1 && !to6_1 || forkert); //tjekker for 2 ens

            //Hvis spiller 1 ikke har vundet, er det spiller 2's tur.
            if ((MAX > point1 || k1 != k2) && !to6_1) {
                System.out.println();
                System.out.println("Spiller nr.2 kaster");
                do {
                    forkert = false;
                    System.out.println("Tast 2 for at kaste terningen: ");
                    k3 = tern.kast();
                    k4 = tern.kast();
                    String kast2 = Spil.nextLine();
                    if (kast2.equals("2")) {
                        kastsum2 = k3 + k4;
                        point2 += kastsum2;
                        System.out.println(k3 + " " + k4);
                        if (k3 + k4 == 2) {
                            System.out.println("Spiller 1. mister alle sine point.");
                            point2 = 0;
                        } else {
                            System.out.println("Summen af kastet er: " + kastsum2);
                        }

                        System.out.println("Spiller nr.2 har " + point2 + " point");

                        if (gemk3 + gemk4 == 12 && k3 + k4 == gemk3 + gemk4) {
                            to6_2 = true;
```

```
                    }
                    gemk3 = k3;
                    gemk4 = k4;

                    if (k3 == k4 && MAX > point2 && !to6_2) {
                        System.out.println();
                        System.out.println("Spiller 2. får et ekstra slag");
                    }
                } else {
                    System.out.println("Forkert input...");
                    forkert = true;
                }
            } while (k3 == k4 && MAX > point2 && !to6_2 || forkert);
        }
    }
    Spil.close();
    //spillet er slut og vinderen findes.
    if (MAX <= point1 - kastsum1 && k1 == k2 || to6_1) {
        System.out.println();
        System.out.println("Spiller nr.1 vandt");
    } else {
        System.out.println();
        System.out.println("Spiller nr.2 vandt");
    }
  }
}
```

Terning klasse

```
public class Terning {

   // Kaster terning
   public int kast() {
       float t1 = (float) Math.random();
       float t2 = t1 * 5;
       int t3 = Math.round(t2);
       return t3 + 1;
   }

}
```

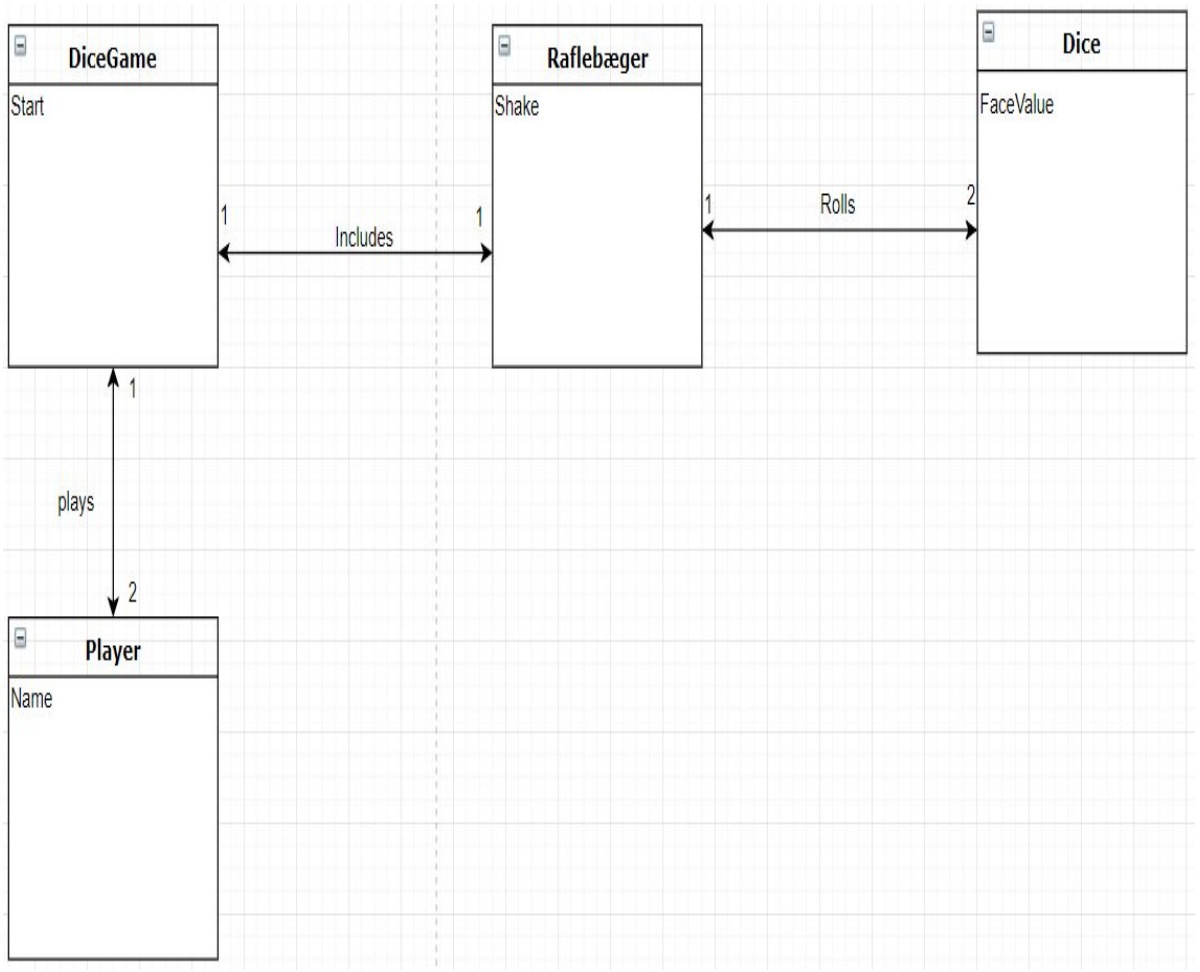
JUnit test

```
import org.junit.Test;

import static org.junit.Assert.assertTrue;

public class TerningTest {

   Terning T = new Terning();

   @Test
   public void kast() {
```

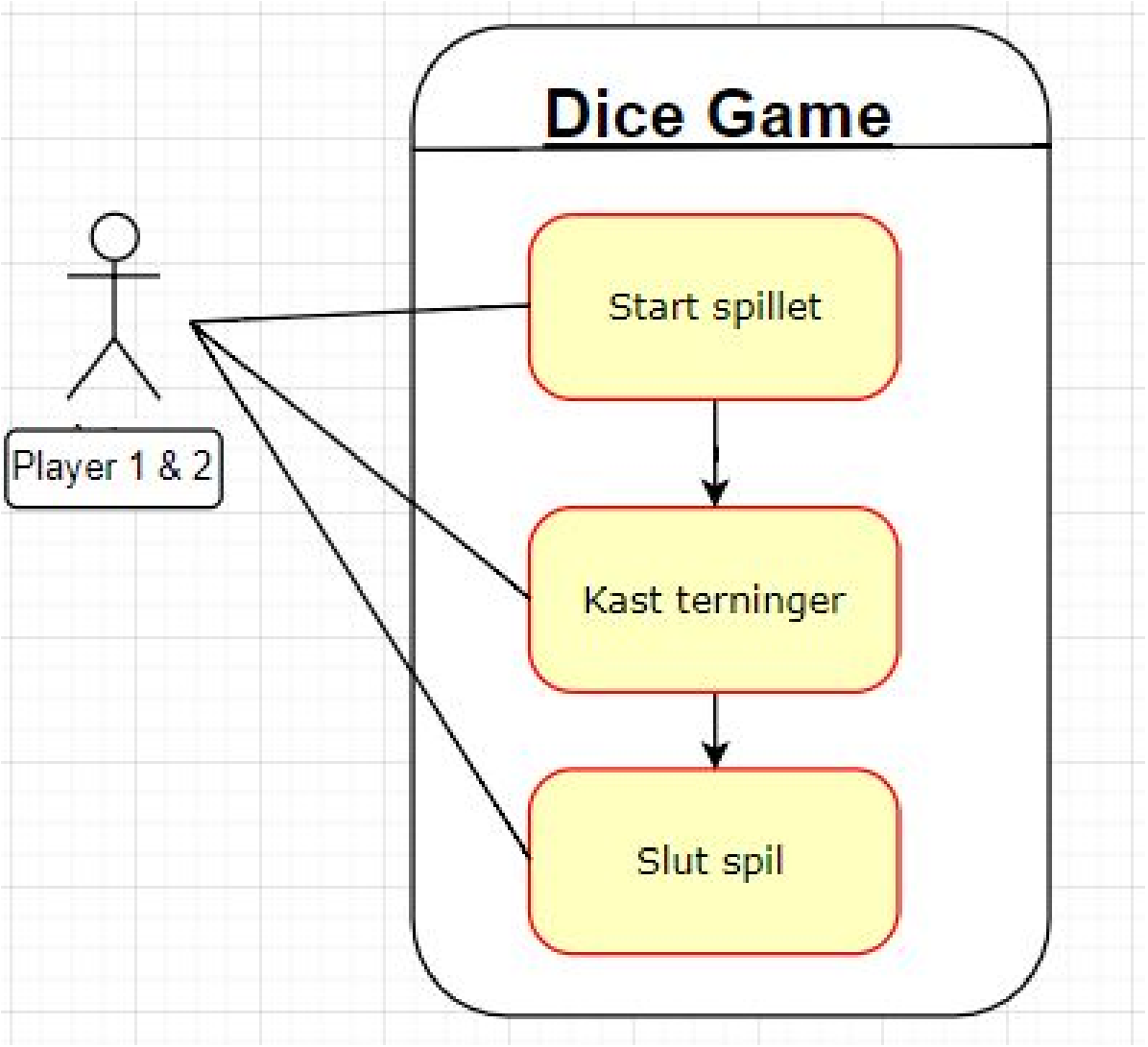
```
for (int i = 0; i < 1000; i++) {
    int tal = T.kast();
    assertTrue(tal > 0 && tal < 7);
}
int SammeSum=0;
int a=0,b=0,c=0,d=0,e=0,f=0,g=0,h=0,i_1=0,j=0,k=0;
for (int i=0; i<12000; i++){
    int kast_1=T.kast();
    int kast_2=T.kast();
    int tal = kast_1+kast_2;

    if(kast_1==kast_2){
        ++SammeSum;
    }
    if (tal==2){
        a++;
    }
    if (tal==3){
        b++;
    }
    if (tal==4){
        c++;
    }
    if (tal==5){
        d++;
    }
    if (tal==6){
        e++;
    }
    if (tal==7){
        f++;
    }
    if (tal==8){
        g++;
    }
    if (tal==9){
        h++;
    }
    if (tal==10){
        i_1++;
    }
    if (tal==11){
        j++;
    }
    if (tal==12){
        k++;
    }

}
System.out.println("2 Øjne = " +a);
System.out.println("3 Øjne = " +b);
System.out.println("4 Øjne = " +c);
```

```
        System.out.println("5 Øjne = " +d);
        System.out.println("6 Øjne = " +e);
        System.out.println("7 Øjne = " +f);
        System.out.println("8 Øjne = " +g);
        System.out.println("9 Øjne = " +h);
        System.out.println("10 Øjne = " +i_1);
        System.out.println("11 Øjne = " +j);
        System.out.println("12 Øjne = " +k);
        System.out.println("Ens antal øjne = " +SammeSum);
    }
    }
```

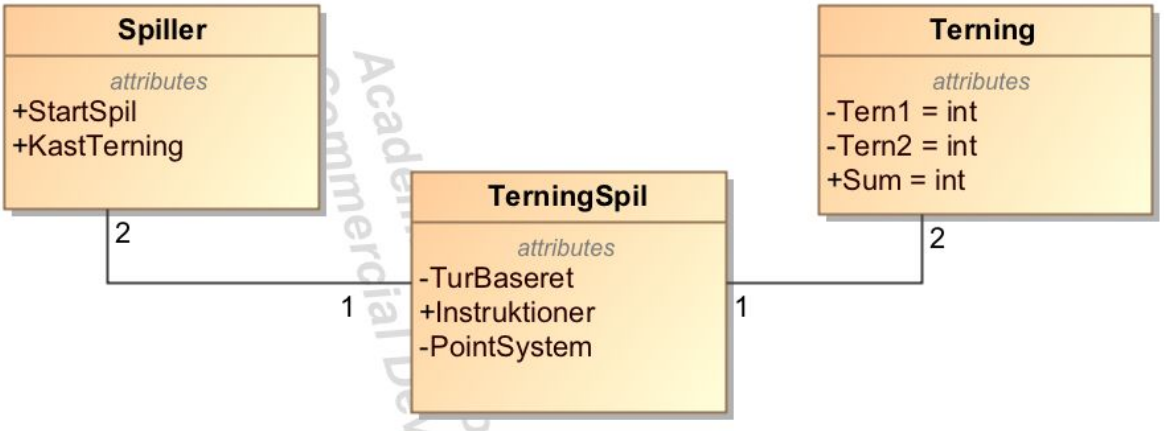
Bilag 2



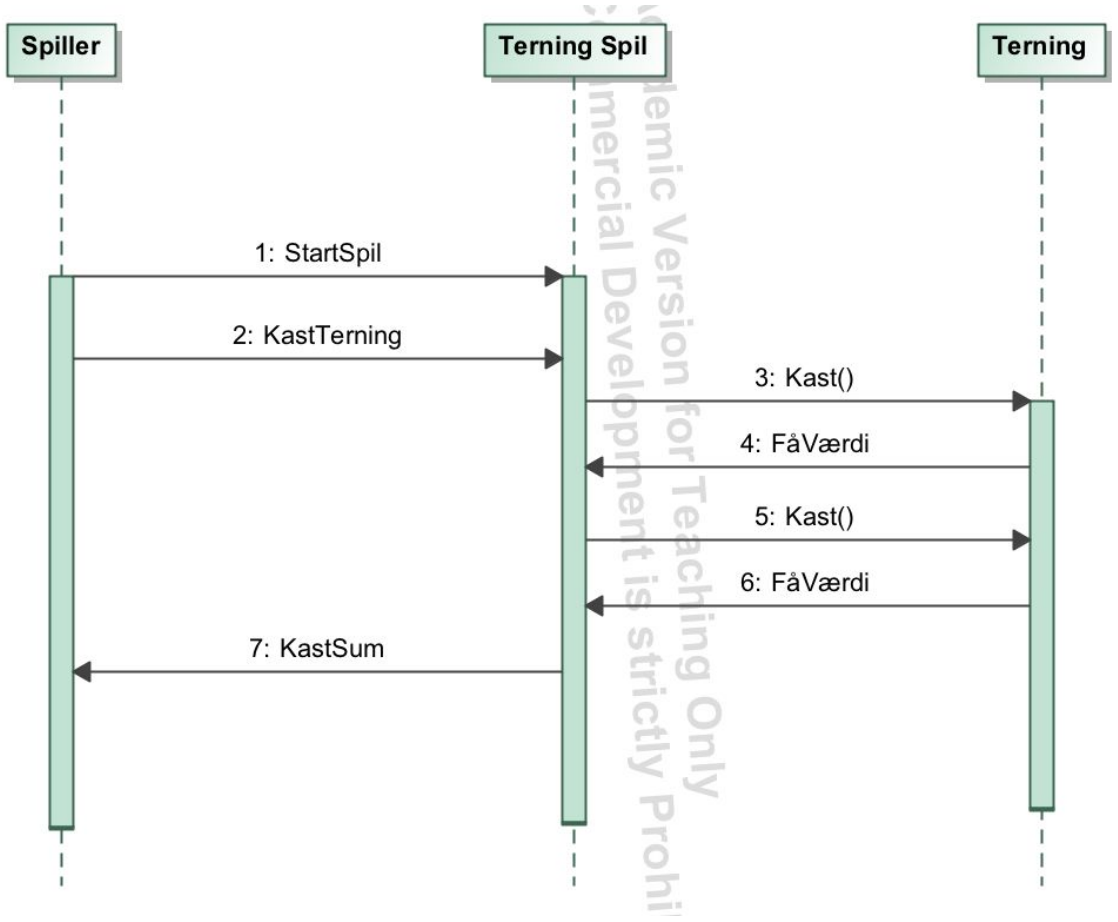
Bilag 3



Bilag 4



Bilag 5



Bilag 6

```
public class Terning {

    // Kaster terning
    public int kast() {
        float t1 = (float) Math.random();
        float t2 = t1 * 5;
        int t3 = Math.round(t2);
        return t3 + 1;
    }
```

Bilag 7

	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8

3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

Bilag 8

```
//Selve spillet
while ((k1 != k2 && k3 != k4 || (point1 - (kastsum1)) < MAX && (point2 - (kastsum2)) < MAX) && !to6_1 && !to6_2)
    System.out.println();
    System.out.println("Spiller nr.1 kaster");
    do {
        forkert = false;
        System.out.println("Tast 2 for at kaste terningen: ");
        k1 = tern.kast();
        k2 = tern.kast();
        String kast = Spil.nextLine();
        if (kast.equals("2")) {  //tjekker for korrekt input
            kastsum1 = k1 + k2;
            point1 += kastsum1;
            System.out.println(k1 + " " + k2);
            if (k1 + k2 == 2) { //tjekker om slaget var 2 ettere
                System.out.println("Spiller 1. mister alle sine point.");
                point1 = 0;
            } else {
                System.out.println("Summen af kastet er: " + kastsum1);
            }

            System.out.println("Spiller nr.1 har " + point1 + " point");
```

Bilag 9

```
2 Øjne = 115
3 Øjne = 510
4 Øjne = 886
5 Øjne = 1461
6 Øjne = 1895
7 Øjne = 2172
8 Øjne = 1967
9 Øjne = 1385
10 Øjne = 974
11 Øjne = 493
12 Øjne = 142
Ens antal øjne = 2170
```

Bilag 10

```java
import org.junit.Test;

import static org.junit.Assert.assertTrue;

public class TerningTest {

    Terning T = new Terning();

    @Test
    public void kast() {
        for (int i = 0; i < 1000; i++) {
            int tal = T.kast();
            assertTrue( condition: tal > 0 && tal < 7);
        }

    }
}
```