

30/09/2022

Indledende Programmering

032112

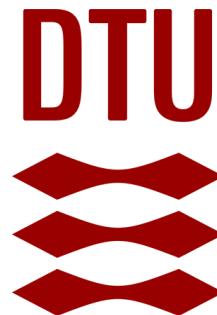
Versionsstyring og Testmetoder

62532

Udviklingsmetoder for IT

62531

CDIO 1



Undervisere og Vejledere

Thorbjørn Konstantinovitz

Christian J. L. Andersen
Frederik M. S. Frost
Frithjof P. Sletten
Gustav W. Kinch
Jakob S. Jacobsen
Kristian Knudsen

Daniel Rubin-Grøn

Anton K. Ludvigsen
Frederik N. Helsø
Mathilde S. Elia
Pi E. B. Bøgeskov-Nielsen

Christian Budtz

Berfin Flora
Emil Iversen
Mikkel Johansen
Sander Albeck Johansen

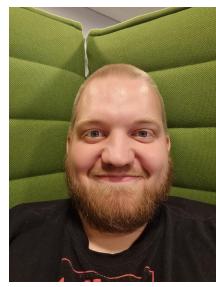
Gruppemedlemmer - Gruppe 14



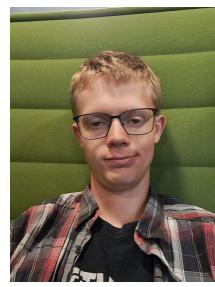
Andreas W. Gansler
s221094



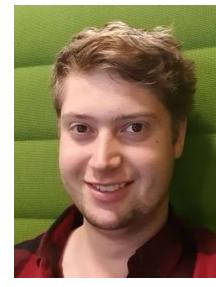
Kenneth Kaiser
s226410



Mikkel Jürs
s224279



Nicklas T. Christensen
s224314



Tobias P. Gørlyk
s224271

1. Resumé

I dette projekt udarbejder vi et terningespil ud fra vores kundes vision. Vi har analyseret den information vi har fået fra kunden og lavet en kravspecifikation ud fra dette og de spørgsmål vi stillede til kunde og projektleder for at afklare reglerne for spillet. Vi har analyseret hvilke interesser og aktører der er for vores terningspil og hvilke use-cases vi kunne finde til det. Hertil har vi udarbejdet vores casual use-case og lavet et use-case diagram samt en fully dressed use-case analyse. Vi har udarbejdet en domænemodel og et systemsekvensdiagram. I vores design afsnit har vi så lavet et design klasse-diagram og et design sekvensdiagram, og dermed gennemgået hvordan vi vil designe vores endelige kode. En dokumentation og forklaring af dele af koden efterfølger så i implementering. Vi har til sidst testet den metode vi bruger til at rulle terninger, for at sørge for at spillet fungere med oprigtig tilfældighed. Til sidst har vi vores konklusion af projektet, samt bilag, hvori vi har linket til vores GitHub repository.

2. Timeregnskab

Timeregnskab:	Mikkel	Nicklas	Tobias	Andreas	Kenneth
Tid (minutter):	820	840	700	710	820

3. Indholdsfortegnelse

1. Resumé	2
2. Timeregrnskab	2
3. Indholdsfortegnelse	3
4. Indledning	4
5. Kravspecifikation	5
5.1 Kundens vision:	5
5.2 Spørgsmål til kunden	5
5.3 Spørgsmål til Projektleder	6
6. Analyse	8
6.1 Interessenter	8
6.2 Aktører	9
6.3 Use cases	9
6.3.1 Casual Use-case	9
6.3.2 Use Case-Diagram	10
6.3.3 Fully dressed Use-case	10
6.4 Domæne model	11
7. Design	12
7.1 Design Klasse-diagram	12
7.2 Design Sekvens Diagram	13
8. Implementering	14
8.1 Main	14
8.2 Player	15
8.3 Players	16
8.4 Conditions	17
9. Test	20
9.1 Beskrivelse af test	20
9.2 Udførelse af test	21
10. Konklusion	23
11. Bilag	24
11.1 Github og Kode	24

4. Indledning

I denne opgave antog vi rollen som en gruppe software ingeniører hyret af Spilfirmaet IOOuterActive til at lave et terningspil. For at lave det ønskede terningspil, har vi måttet arbejde på en virkelighedsnær og iterativ måde.

Opgaven blev gennemført ved inception, hvor gennemgang af krav og analyse har ført til idér og elaboration for hvordan vores terningspil skulle laves, sågar noget tidlig kode blev skrevet. Efterfølgende og samtidig har spørgsmål rejst sig og der har været gennemgang med både kunde og projektleder for hvordan systemet skulle se ud. Efterfølgende har inception og elaboration kørt igen, således kørte arbejde og planer indtil at den endelige kode så kunne skrives færdigt og systemet havde taget sin form.

Systemet blev testet og krav efterset. Således stod vi med noget som kunne bruges som det endelige terningspil. Her er vores arbejde så gennemgået igen og skrevet efter, for at skabe en rapport til vores terningspil.

5. Kravspecifikation

Kunden har en vision for programmet, som vi som udviklere bliver nødt til at forstå bedst muligt for at kunden får et produkt tilbage som de er glade for. Vi får derfor en beskrivelse af kunden som vi analysere og kan stille spørgsmål til, langt før vores design og analyse projekt er startet.

5.1 Kundens vision:

"Vi vil gerne have et system, der kan bruges på maskinerne (Windows) i databarerne på DTU. Det skal være et spil mellem 2 personer. Spillet skal gå ud på at man slår med et raflebæger med to terninger og ser resultatet med det samme. Summen af terningernes værdier lægges til ens point. Vinderen er den, der opnår 40 point. Hvis der er ressourcer til det, er der følgende ekstraopgaver:

1. *Spilleren mister alle sine point hvis spilleren slår to 1'ere.*
2. *Spilleren får en ekstra tur hvis spilleren slår to ens.*
3. *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 ekstrakast eller i forrige tur.*
4. *Spilleren skal slå to ens for at vinde spillet, efter at man har opnået 40 point.*

Vi forventer at alle almindelige mennesker kan spille det uden en brugsanvisning.

Vi vil gerne se en test, der beviser at raflebægeret virker korrekt, hen over 1000 kast.

Det er op til jer om dokumentation og kode skal være på dansk eller engelsk, dog skal fagudtryk være naturlige."

Ud fra den beskrivelse af spillet kom vi op med disse spørgsmål til kunden og projektlederen, som skulle sørge for at der var enighed om udkommet. Her er hvert punkt et spørgsmål og har et underpunkt som beskriver det svar vi fik.

5.2 Spørgsmål til kunden

- Går turen på skift?
 - Ja mellem spiller 1 og 2.
- Kan man blive ved med at slå dobbelt og få ekstra ture?
 - Ja
- Gælder det, at man vinder hvis man slår to seksere i en runde, slår så ikke dobbelt og runden efter slår to seksere igen? Eller skal det forstås i speciering 3, at man aldrig kan slå dobbelt efter sidste runde og kun efter sidste kast.
 - Ekstra kast kun!
- Vinder man hvis man når 40 point præcist selvom man ikke slog dobbelt?
 - Man skal slå dobbelt efter man har opnået 40 point.

- Hvis man får mere end 40 point, kan man så stadig vinde spillet?
 - Ja, hvis man slår dobbelt.
- Hvordan ønskes det at det bliver indikeret at en spiller har vundet.
 - print en besked om at spiller 1 eller spiller 2 har vundet, slut spillet eller spørg om man vil spille igen.
- Hvad er et almindelig menneske i vores målgruppe? evt. PEG-rating
 - Folk der ikke kan kode, men har arme, øjne og kan læse.
- Hvis man slår dobbelt et, efter at være nået over 40 point, vinder man så, eller mister man stadig alle sine point.
 - Man mister alle sine point og skal starte forfra.

5.3 Spørgsmål til Projektleder

- Er der en compiler installeret på maskinerne?
 - I Så Fald Hvilken compiler?
Java 17 er installeret
 - Hvis ikke, har vi mulighed for at installere en?
- Se ovenstående.
- Hvilke input devices er der? Keyboard, mus?
 - Keyboard og mus.
- Hvilke output devices? Er der skærm tilgengængelig og hvad er størrelsen/opløsningen?
 - 1080p, ja der er skærm.
- Hvad beskriver "virker korrekt", når det gælder test af 1000 kast.
 - Indenfor 5% af variation af normalfordeling.

Vi har derfor udarbejdet at programmet skal have følgende krav, for at åbne kundens ønsker. Disse krav vil blive brugt fremadrettet til at designe programmet.

Det skal køre på en pc, i DTU's databar.

- Det skal derfor kunne køre på denne version af Windows og java
- Vi skal kunne bruge disse skærme, og skal køre med den oplosning
- Vi bruger deres input-devices keyboards og mus.
- Det skal kunne uploades til maskinen via ekstern device eller køres fra online websted.

Der skal værer to spillere

- Det skal spilles af to spillere, no more, no less.
- Det skal være muligt at kende forskel på de to spillere
- Der skal tælles point individuelt for de to spillere
- Man skal sørge for at spillerne kan se hvis tur det er.
- Evt en måde at finde hvem der starter.

Der skal være visse spilleregler

- Begge spillere skal kunne slå med to terninger.
- Terningerne skal slå inden for en variation af 5% fra total tilfældig efter 1000 rul.
- Det terningerne viser gives af point til spilleren der slog terningerne
- Begge spillere skal vinde hvis spilleren opnår over 40 point og slår dobbelt med terningerne
- Hvis spillere slår to ens med terningerne kan de slå igen.
- Hvis spillere slår terningerne og begge viser en, mister de alle deres point, selv hvis de havde opnået målet med 40 point.
- Hvis en spiller slår to seksere, to gange i træk, vinder de automatisk spillet, selvom de måske ikke har opnået de 40 point.

Man skal se når man vinder spillet

- Hvis en spiller vinder, skal det vises til spillerne
- Et valg skal gives:
 - a. vil i spille igen
 - b. vil i afslutte spillet

Vi har derudover fået et krav for kunden at de skal kunne følge med i designprocessen, her har vi brugt GitHub, hvor man kan se alle commits og kommentarer til koden.

6. Analyse

6.1 Interessenter

- Spillere (spiller 1 og spiller 2): Ønsker at have en intuitivt måde at spille terninger på. (intuitiv betyder her at spillet ikke kræver nogen manual og kan spilles på computeren, uden fysiske terninger). Deres indflydelse er lav.
- Spilfirmaet IOOuterActive: Ønsker et spil som de kan udgive og sælge til en indkomst. Deres indflydelse er høj.
- Os som udviklere - Ønsker at holde budgethåndtering, samt have en kognitiv og iterativ udvikling. Vores indflydelse er medium.

	Spillere	Spilfirmaet IOOuterActive	Os som udviklere
Interesse	Ønsker at kunne spille spillet	Ønsker at have et spil, der skal kunne spilles i på DTU's databarer	Ønsker at kunne kontrollere og styre projektets budget, samt kognitiv og iterativ udvikling
Indflydelse	Lav	Høj	Medium

6.1.1 - Interessenter - Interesse og indflydelse

Key-Stakeholders:

Os som gruppe:

Vi som gruppe, er ultimativ beslutningstager for projektets afvikling og udvikling.

IOOuterActive:

De besidder muligheden for at kunne tildele flere midler til projektet, men samtidig også dem der kan lukke projektet ned.

IOOuterActive:

Ønsker at kunne udbyde et terningespil, som de kan få profit på.

Spillere:

Ønsker at kunne spille et spil, hvor der skal slås med 2 terninger.

6.2 Aktører

Primær:

- Spillere (spiller 1 og spiller 2)

Sekundær:

6.3 Use cases

6.3.1 Casual Use-case

Spiller terningspil

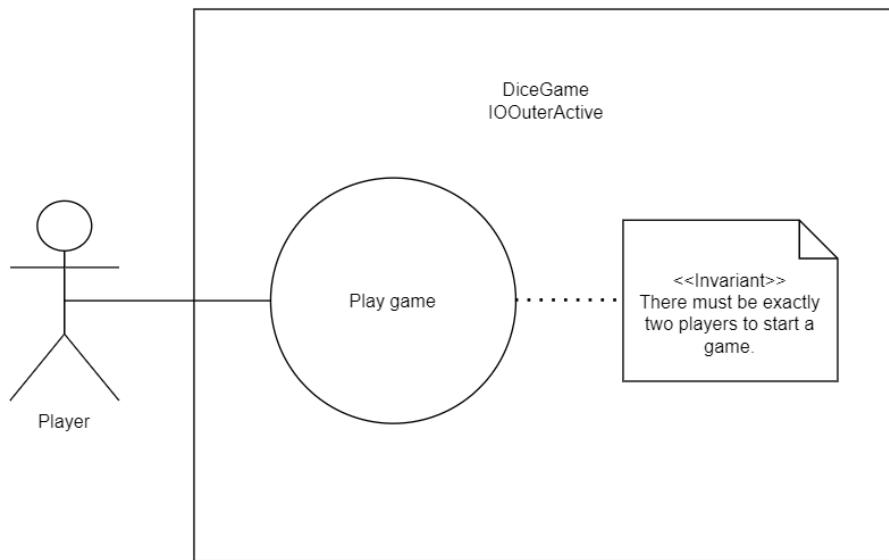
actor: Spillere (spiller 1 og spiller 2)

Spiller 1 og spiller 2 starter systemet op for at tage et terningespil. Spiller 1 starter og beder systemet om at kaste to terninger for sig. Systemet viser to tilfældige tal mellem 1 og 6 som terningernes øjne, i dette tilfælde får spiller 1 en 1'er og en 6'er. De to terningers samlet værdi ligger til spiller 1's sum som nu er 7.

Spiller 2 har nu sin tur og beder også systemet om at kaste to terninger, systemet giver spiller 2 to 5'ere og ud over at få 10 i sin sum, får spiller 2 også lov til at kaste terninger igen, da terningerne var to ens. Spiller 2 slår nu 6 og 4 og ender sin runde med en sum på 20.

Spiller 1 slår 3 og 4 og slutter med 14 som sum. Spiller 2 slår 5 og 2 og ender sin runde med 27. Spiller 1 slår 2 og 4 og ender sin tur med 20 som sum. Spiller 2 slår to 6'ere efterfulgt af 6 og 5, hvilket bringer spiller 2 til en sum på 50 og ender sin tur. Spiller 1 slår 2 og 3 og ender sin tur med 25 som sum, hvorefter spiller 2 slår to 3'ere og vinder spillet, da spiller 2 har en sum på 40 eller derover og slog 2 ens efterfølgende. Spillet er nu slut.

6.3.2 Use Case-Diagram



Figur 6.3.2.1 - Use-case diagram af main-usecase

6.3.3 Fully dressed Use-case

Primær actor: Spillere (spiller 1 og spiller 2)

Stakeholders and Interests: IOOuterActive, DTU IT support og udviklerne

Preconditions: Computeren har installeret og kører systemet

Succeskriterier: Systemet kan følge spillets regler og udpege ved enden af spillet en vinder.

Basic flow:

1. Spiller 1 starter og kaster to terninger i systemet.
2. Systemet viser de tilfældige værdier af de to terninger mellem 1 og 6.
3. Systemet ligger værdien af de to terninger til spiller 1's sum og spiller 1 har færdiggjort sin runde
4. Spiller 2 gentage trin 1 til 3, hvorefter deres tur er færdig og spiller 1 gentager trin 1 til 3.
5. trin 1 til 4 gentager sig indtil spiller 1, spiller 2 eller begge har en sum på eller over 40

6. Trin 1 til 4 fortsætter indtil, spilleren med en sum, på eller over 40, slår to ens værdier som ikke er 1.
7. Spilleren med en sum på eller over 40 som også slog to ens, som ikke er 1, er nu vinderen og spillet er slut

Extension:

- a. En spiller slår to 1'ere
 1. Spilleren som slog to 1'ere mister nu alle sine point i sum.
- b. En spiller slår to ens
 1. Spilleren som slog to ens får lov til at slå igen inden den anden spiller får sin tur.
 2. Spilleren slår til sidst noget som ikke er to ens og det er den anden spillerens tur igen.
- c. En spiller slår to seksere to gange i træk
 1. Spilleren som slog to seksere to gange i træk er nu, uanset hvad summen er, kåret til vinderen og spillet er slut.

Special Requirements:

1. Systemet skal kunne køre på Windows, med Java 17, over en pc fra DTU's databar.

6.4 Domæne model

En model til at visualisere objekter og situationer i virkeligheden, for så bedre at kunne repræsentere dem i software.

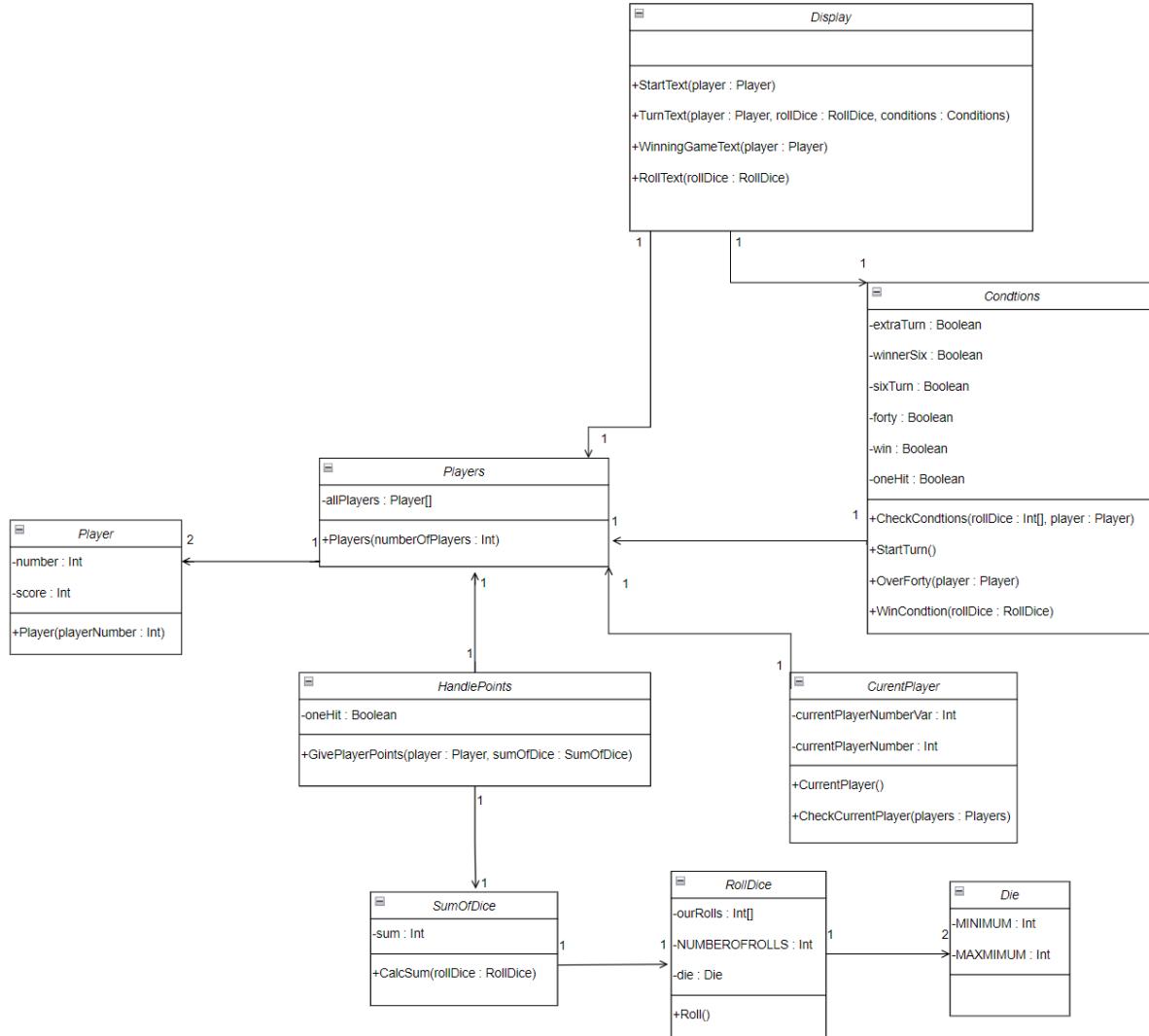


6.4.1 - Domænemodel

En simpel model til et simpelt spil. Der er to spillere som har en rækkefølge når de spiller et terningspil. Terningspillet spilles med to terninger som har et antal øjne.

7. Design

7.1 Design Klasse-diagram

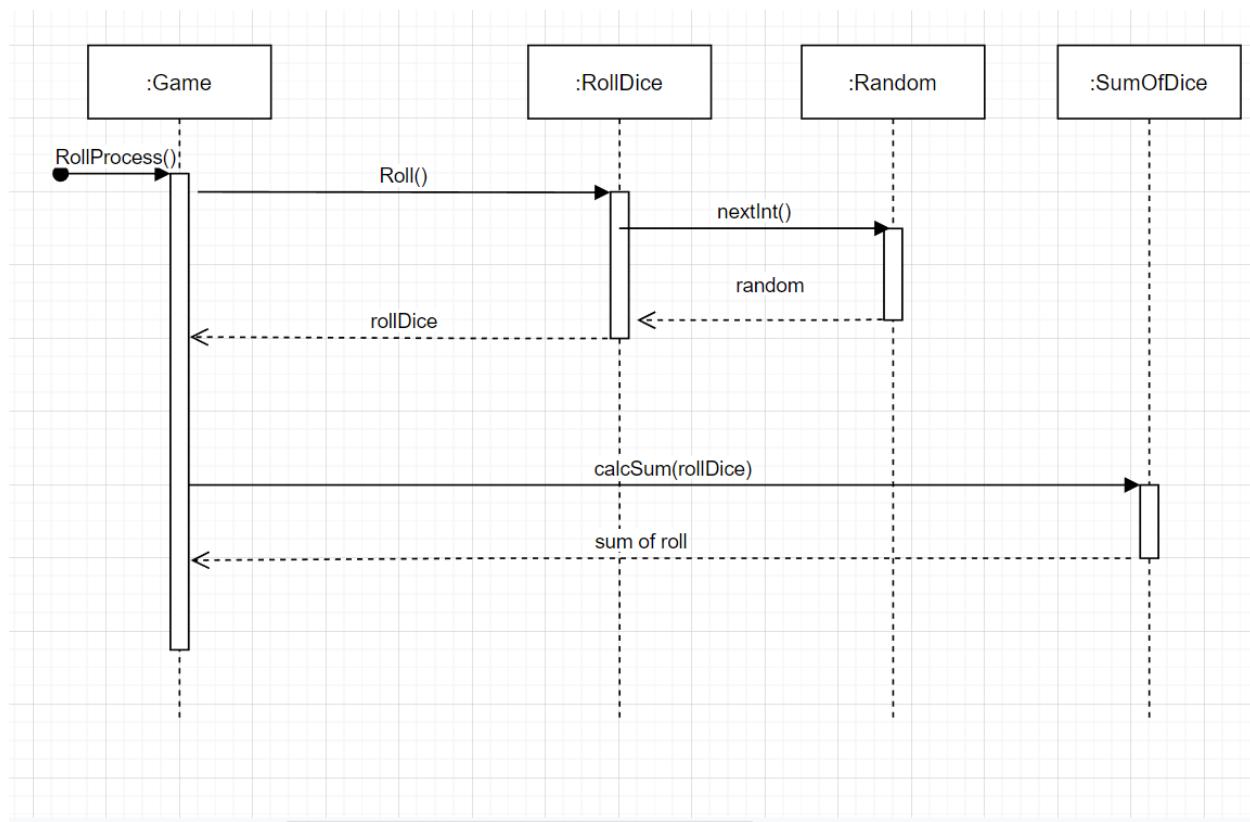


Figur 7.1.1 - Design klasse-diagram

I dette projekt, har vi forsøgt at arbejde så iterativt som muligt. Da vi havde fundet de første 3 metoder vi skulle bruge for at kunne lave spillet, startede vi med at lave disse i forskellige klasser. Vi startede med at lave: Player, Players, Die. Derefter kom hvilke metoder vi skulle bruge naturligt, og vores klasse-diagram blev udvidet som projektet udfoldede sig.

Vi har forsøgt at gå efter "High cohesion, low coupling", ved at lade de forskellige klasser bestå af enkelte eller få metoder. Conditions har flere metoder, da denne indeholder samtlige ekstraopgaver i projektet. Foruden dette, har næsten alle klasserne kun 1-2 associationer, hvor "Players" klassen har flere associationer. Players klassen er dog vedligeholdelse venlig, da associationerne primært handler om at andre klasser får "numberOfPlayers" fra Players klassen. Vi har bygget spillet, således at man let kan ændre følgende: Antal spillere, Hvor mange sider terningen skal have, antal point for at vinde.

7.2 Design Sekvens Diagram



Figur 7.2.1 - Design sekvens-diagram

Vi har lavet et design sekvens diagram for at vise hvordan de forskellige software objekter interagere med hinanden og i hvilken rækkefølge objekterne bliver initieret i. Vi prøvede at designe den funktion som ruller de 2 terninger og derefter lægger summen af de 2 terninger sammen. På den måde fik vi et overblik over, hvordan det var vi skulle kode interationen mellem disse objekter.

8. Implementering

I dette afsnit gennemgår vi nogle af de klasser som vi bruger i terningkast spillet. Vi forklare hvordan de arbejder på tværs af hinanden og hvad der skal til for at starte spillet, spille spillet og vinde spillet.

8.1 Main

```
public class Main {  
    public static void main(String[] args) {  
        Game game = new Game( howManyPlayers: 2); //Creates new instance of a game  
    }  
}
```

Snippet 8.1.1 - Main - main

I vores ‘Main’ class hvor programmet starter med at køre har vi kun vores første void ‘main’ som kører en enkelt linje kode. Den bruger klassen ‘Game’ og laver et nyt instance af game, og beskriver at der skal være 2 spillere i dette.

8.2 Player

```
public class Player {  
    3 usages  
    private int number;  
    2 usages  
    private int score;  
    1 usage  
    ↴ public Player(int playerNumber){  
        number = playerNumber;  
    }  
    9 usages  
    ↳ public int getNumber() { return number; }  
  
    ↳ public void setNumber(int number) { this.number = number; }  
    6 usages  
    ↳ public int getScore() { return score; }  
    2 usages  
    ↳ public void setScore(int score) { this.score = score; }  
}
```

Snippet 8.2.1 - Player - Player, getters & setters

I 'Player' klassen laver vi to integeres, den ene beskriver spillerens nummer og den anden spillerens score. Vi bruger dette til at kunne holde score individuelt for spillere. Vi har så en metode der sætter spillerens nummer fra 'Players' klassen. Derudover er der en del 'Getters & Setters'.

8.3 Players

```
public class Players {  
    4 usages  
    private Player[] allPlayers;  
    1 usage  
}    public Players(int numberOfPlayers){  
        allPlayers = new Player[numberOfPlayers];  
  
    }        for(int i =0; i<numberOfPlayers; i++){  
  
        allPlayers[i] = new Player( playerNumber: i+1);  
    }  
}  
}  
3 usages  
}    public Player[] getAllPlayers() { return allPlayers; }  
}    public void setAllPlayers(Player[] allPlayers) { this.allPlayers = allPlayers; }  
}
```

Snippet 8.3.1 - Players - Players, getAllPlayers, setAllPlayers

I 'Players' klassen laver vi et array af 'Player's.

Vi har så en metode som tilføjer spillere og sætter spillerens nummer ud fra indexet i det array med et for-loop. Det tilføjer et antal spillere ud fra hvor mange man har valgt skal være med. Derudover er der 'Getter' og 'Setter' for det array.

8.4 Conditions

```
private boolean extraTurn, doubleSix, sixWin, forty, win, oneHit;  
//Method one to check both double turn, snake eyes and double 6. (Condition 1, 2 and 3)  
1 usage  
public void CheckConditions(int[] rollDice, Player player){  
    sixWin = false;  
    if (rollDice[0] == rollDice[1]){  
  
        //Extra assignment 1.  
        if (rollDice[0] == 1){  
            player.setScore(0);  
            oneHit = true;  
            // put 1 here maybe  
        }  
        else if (rollDice[0] == 6){  
            if (doubleSix){  
                sixWin = true;  
            }  
            doubleSix = true;  
            extraTurn = true;  
        }  
        else {  
            extraTurn = true;  
            doubleSix = false;  
        }  
    }  
    else{  
        doubleSix = false;  
        extraTurn = false;  
    }  
}
```

Snippet 8.4.1 - Conditions - booleans og CheckConditions

'Conditions' klassen bruges til at beskrive de forskellige betingelser for spillet, altså hvad det er der skal resultere i. Det er en længere beskrivelse så her på kodestykke 9.4 kan man se at vi laver seks booleans, for henholdsvis ekstra tur, vindere for dobbelt seks, en tur man har slået to sekser, hvis man har fyrré point, man har vundet og hvis man har slået dobbelt 1. Her har man en metode der skal modtage et integer array som svarer til de forskellige slag der er blevet slået, samt en 'Player' som skal bruges så man kan se hvilken spiller det var der slog.

Booleanen sixWin sættes til at være false som udgangspunkt.

Der er da et if-statement som spørger om det første rul viser samme øjne som andet rul, hvis de ikke gør vil der ikke være en ekstra tur og doubleSix variablen sættes til false, så næste kast tror at der blev slået dobbelt seks i denne runde.

Derimod hvis de to terninger viser samme øjne, vil der komme endnu et if-statement som spørger om øjnene på den ene terning viser 1, man behøver ikke tjekke for begge, da vi allerede har bevist at de må være ens for at nå dette statement. Hvis de viser 1 sættes den givne spillers point til at være 0 og variablen der beskriver om man slog dobbelt et sættes til at være sand.

Hvis terningen derimod ikke viser 1, kommer der et else-if-statement som spørger om terningerne viser 6, hvis de gør de, ses der først om der lige er blevet slået dobbelt seks, hvis ja, sættes variablen for at du vandt med dobbelt, dobbelt seks til at være sand. Hvis ikke sættes variablerne for ekstra tur og at sidste slag var dobbelt seks til at være sande.

Men hvis terningen ikke vist 6 kommer der et else, som siger du istedet bare får en ekstra tur og det sættes til at du ikke lige har slået dobbelt seks.

```
public void StartTurn(){
    extraTurn = false;
    oneHit = false;
}
1 usage
public void OverForty(Player player){
    if(player.getScore() >= 40){
        forty = true;
    }
    else{
        forty = false;
    }
}
1 usage
public void WinCondition(RollDice rollDice){
    if(forty && extraTurn){
        win = true;
    }
    else if(doubleSix && sixWin){
        win = true;
    }
}
```

Snippet 8.4.2 - Conditions - StartTurn, OverForty, WinCondition

Udover det, er der en del flere metoder i Conditions.

Her ses StartTurn som kører en enkelt gang til den spiller som starter i spillet. Den sætter bare variabler til false.

Der er metoden overForty der modtager en Player, den checker spillerens score og sætter en boolean forty til at være henholdsvis true eller false ud fra om scoren er over 40 eller ej.

Så er der vores WinCondition, der kaldes for tjekke om en spiller har vundet. Der er to forskellige måder at vinde på, først tjekker den om du har over 40 point samt har en ekstra tur, fordi du slog dobbelt, og hvis det er sandt vinder du. Ellers tjekkes der om du har slæbt dobbelt seks to gange i streg, hvorfra du også vinder.

```
2 usages
public boolean isExtraTurn() { return extraTurn; }

2 usages
public void setExtraTurn(boolean extraTurn) { this.extraTurn = extraTurn; }

public boolean isWinnerSix() { return doubleSix; }

public void setWinnerSix(boolean winnerSix) { this.doubleSix = winnerSix; }

public boolean isSixWin() { return sixWin; }

public void setSixWin(boolean sixWin) { this.sixWin = sixWin; }

public boolean isForty() { return forty; }

public void setForty(boolean forty) { this.forty = forty; }

3 usages
public boolean isWin() { return win; }

public void setWin(boolean win) { this.win = win; }

1 usage
public boolean isOneHit() { return oneHit; }

public void setOneHit(boolean oneHit) { this.oneHit = oneHit; }
```

Snippet 8.4.3 - Conditions - Getters & Setters

Udover det er der en masse getters og setters for vores booleans hvis vi skal bruge dem.

9. Test

9.1 Beskrivelse af test

Vi har fået til krav at kunne bevise, at vores raflebæger virker korrekt hen over 1000 kast.

Vi har valgt at gøre det an ved at lave et kode eksempel, hvor vi slår med en terning 1000 gange.

Vi gemmer dermed den samlede sum af de 1000 terningekast.

Derefter regner vi normalfordelingen som er $\pm 5\%$ af 7000, hvor 7000 er gennemsnittet er summen ved 1000 terningkast

Dermed sammenligner vi om den samlede sum ligger indenfor eller udenfor normalfordelingen.

Hvis den samlede sum ligger indenfor normalfordelingen printer vi ud på skærmen at værdien er indenfor normalfordelingen og hvis den samlede sum ligger udenfor normalværdien printer vi at værdien ligger udenfor normalværdien.

På den måde vil man hurtigt kunne se om terningerne slår tilfældige tal.

Hvis man kører programmet igennem et par gange og den viser at den samlede sum ligger indenfor normalværdien og den samlede ændrer sig for hver gang, vil man kunne se at terningerne slår tilfældigt hver gang og at raflebægeret virker korrekt.

Vi har her to kode-eksempler fra testen.

Det første kode eksempel kører inde i et for loop 1000 gange, hvor den bruger metoden Random.nextInt() to gange og lægger det til en samlet sum.

```
//For loopet regner forekomsten af alle tilfælde to til tolv sammen i en int (to - tolv) ved 1000 terningekast.  
for (int i=0; i<1000; i++) {  
    slag1 = test.nextInt( bound: (6-1)+1)+1;  
    slag2 = test.nextInt( bound: (6-1)+1)+1;  
    sum = slag1 + slag2 + sum;
```

Snippet 9.1.1 - For-loop og nextInt metoden

I det andet kode eksempel kontrollere vi om den samlede sum ligger indenfor eller udenfor normalfordelingen i en if/else statement.

```
if (sum < 7000+(7000*0.05) && sum > 7000-(7000*0.05)) {  
    System.out.println("Værdien er indenfor normalfordelingen");  
}  
else {  
    System.out.println("Værdien er udenfor normalfordelingen");  
}
```

Snippet 9.1.2 - if-statement: er output er indenfor normalfordeling

9.2 Udførelse af test

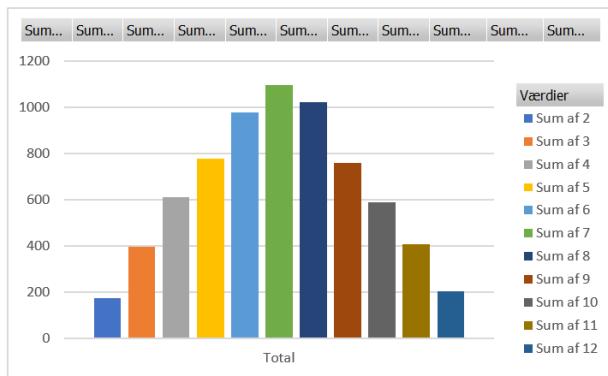
Vores forventning er at vi får en graf der følger nedenstående kombinatorik, når vi kører vores loop 1000 gange.

Kombinatorik for alle mulige slag med to terninger mellem 1-6						sum	probability
{1,1}	{1,2}	{1,3}	{1,4}	{1,5}	{1,6}	2 & 12	1/36
{2,1}	{2,2}	{2,3}	{2,4}	{2,5}	{2,6}	3 & 11	1/18
{3,1}	{3,2}	{3,3}	{3,4}	{3,5}	{3,6}	4 & 10	1/12
{4,1}	{4,2}	{4,3}	{4,4}	{4,5}	{4,6}	5 & 9	1/9
{5,1}	{5,2}	{5,3}	{5,4}	{5,5}	{5,6}	6 & 8	5/36
{6,1}	{6,2}	{6,3}	{6,4}	{6,5}	{6,6}	7.	1/6

Figur 9.2.1 - Sandsynligt resultat af slag med 2, 6 sided terninger.

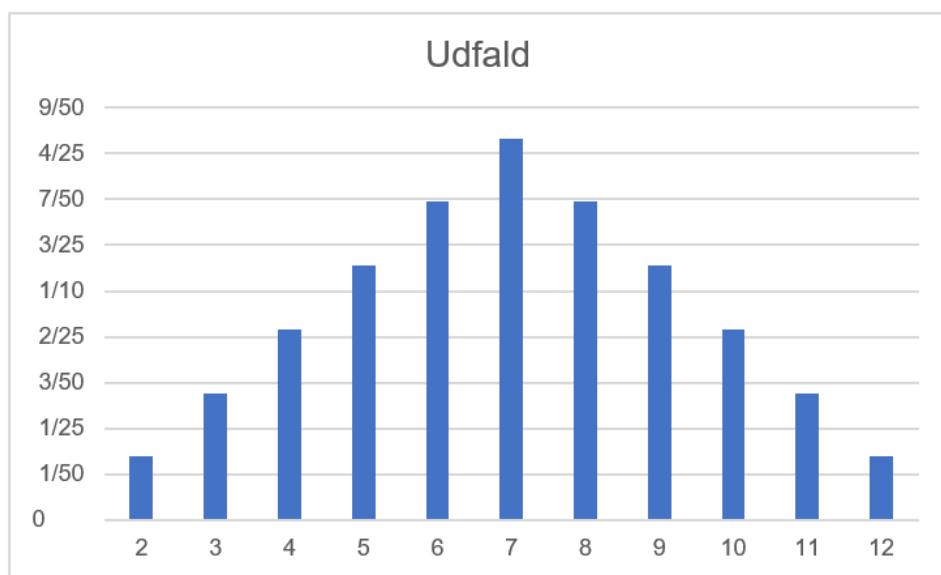
Herunder er et diagram ved testen 7 gange. Fra tallene kan ses at vi bevæger os tættere på sandsynligheden for alle udfaldene jo flere gange testen køre.

	2	3	4	5	6	7	8	9	10	11	12
Test1	25	64	82	133	144	140	124	104	99	61	24
Test2	27	56	78	110	134	165	148	109	74	65	34
Test3	26	56	85	93	137	175	142	123	82	60	21
Test4	22	62	103	99	138	174	144	115	76	36	31
Test5	25	50	99	122	132	156	150	93	80	53	40
Test6	23	62	75	106	147	133	172	105	87	68	22
Test7	26	44	89	115	143	151	139	108	91	62	32



Figur 9.2.2 - Testen af 7000 terningkast (testen 7 gange)

2	3	4	5	6	7	8	9	10	11	12
1/36	1/18	1/12	1/9	5/36	1/6	5/36	1/9	1/12	1/18	1/36



Figur 9.2.3 - Sandsynligheden for alle udfald af to seks sidede terninger

10. Konklusion

Vi kan konkludere at vi ved hjælp af flere samtaler med kunden og det dannede feedback loop, har opnået kundens vision efter den positive tilbagemelding vi har fået. Vi har igennem projektet arbejdet iterativt, og konstant vendt tilbage for at sikre vores planer holdte trin med kundens vision. Vi har udarbejdet diagrammer igennem processen for at planlægge og designe koden før den er skrevet, og længere fremme i projektet som vi fik skrevet kode, vendte vi tilbage til vores diagrammer og udviklede nye mere uddybede dele for selv at få en bedre forståelse for hvordan vi fortsatte projektet. Vi har kigget på design patterns som 'high cohesion' og udarbejdet vores kode til at have individuelle klasser for forskellige fokuserede funktioner og gået målrettet efter low coupling for at vi måske kan bruge vores klasser igen i senere projekter. Vi kan til sidst konkludere resultaterne af vores tests, som stemmer overens med vores forventede målinger, vi tester ved 1000 terningkast og vi har i alle vores tests ramt indenfor de 5% af normalfordeling i gennemsnit, da det er utrolig usandsynligt.

11. Bilag

11.1 Github og Kode

Dette er vores GitHub repository:

https://github.com/Kraminius/_14_del01

GitHub Brugernavne - af en eller anden grund viser den ikke Kenneth "Your Name" på github, som contributers, men kun i git loggen.

Gruppemedlem	Kenneth	Tobias	Andreas	Nicklas	Mikkel
GitHub Brugernavn	Your Name	Toby	AWG96	Shjorn	mikju
GitHub Brugernavn	Kenneth Kaiser	TwoFacedTo by			mikkelJurs

Grundet at Kenneth havde indtastet forkert email i intellij, fremgår det at Kenneth kun har commitet en gang. Men alle commits fra user Your Name, er lavet af Kenneth.