CDIO - 1

Et terningspil med to terninger, for to spillere.

Af gruppe 33

Simon Engquist
s143233

Arvid Langsø
s144265

Mikkel Lund
s165238

Jeppe Nielsen
s093905

Mads Stege
s165243

Danmarks Tekniske Universitet
DTU Compute

16. September 2016 - Kl. 23:59

Side antal (med bilag): 31

Kurser: 02312, 02313, 02314, 02315

CDIO - 1							
Time-regnsk	Ver. 2014-1	Tidsplan for CDIO del 1 - 2016					
Dato	Deltager	Design	Impl.	Test	Dok.	Andet	Ialt
29-09-2016	Mads S.	2	5	0,5	0,5		8
29-09-2016	Jeppe	2	5	0,5	0,5		8
29-09-2016	Simon	2	5	0,5	0,5		8
29-09-2016	Mikkel	1	4	0,5	0,5		6
29-09-2016	Arvid	3	5	0,5	0,5		9
30-09-2016	Mads S.	2,5			0,5		3
30-09-2016	Jeppe		4				4
30-09-2016	Simon	2	1				3
30-09-2016	Mikkel	2	2				4
30-09-2016	Arvid	3	1				4
30-09-2016	Mikkel			2			2
01-10-2016	Mads S.				1,5		1,5
02-10-2016	Arvid			2			2
02-10-2016	Jeppe			2			2
03-10-2016	Mads S.	2					2
03-10-2016	Jeppe	2					2
03-10-2016	Simon	2					2
03-10-2016	Mikkel	2					2
03-10-2016	Arvid	2					2
06-10-2016	Mads S.			1	6		7
06-10-2016	Jeppe	1		3	2		6
06-10-2016	Simon			4	2		6
06-10-2016	Mikkel	1		1	6		8
06-10-2016	Arvid			4,5	3,5		8
07-10-2016	Mads S.				2		1
07-10-2016	Jeppe				1		1
07-10-2016	Simon				1		1
07-10-2016	Mikkel				2		1
07-10-2016	Arvid				2		1
	Sum	31,5	32	20	31	0	114,5

Indholdfortegnelse

1 Abstract

This short report aims to describe and document the creation and implementation of a simple dice game.

Powered by nothing but a basic understanding of Java coding, we have been assigned with creating a small project capable of handling a semi-interactive text-based game with two opposing users. The game is built using a random number generator, in-code specific game rules, as well as the ability to handle and return user inputs.
The whole process began by creating UML diagrams. Through this simple approach, we ended up with a very comprehensive and thought-through solution.
We decided to split the base game up into four interconnected classes, with a main body.
The seperate parts are the:

- ”Game”(main) - the core component of our coding. It handles both the classes and their calculated outputs, reads the input from both of the users, handling the two separate users, and returns useable values to the separate TUI-class.
- ”TUI”(Textual User Interface) - The text frame handling the user input, and sending the required pieces of text back to the user. It also communicates back to the Main body, telling it when the user wishes to roll the dice.
- ”Player” - The class describing the attributes of each of the two players. Returns values to the Main, and starts the classes coming afterwards.
- ”DiceCup” - The virtual cup with the dice, is called upon by the Player-class, and begins rolling two objects of the Die-class.
- ”Die” - The smallest - but most important - piece of code in this project. The Die class is a simple number generator from 1 to 6. All the other classes and the main rely solely on this small piece of information for further actions.

The whole process, designing and discussing the implementation and limitations of the client-specified requirements proved a frustrating experience. Creating a valid product for this assignment was no meager task, since we have had to incorporate most, if not all of the so far taught material. Coordination between the five seperate pieces of coding describe above called for new levels of interaction between us.
In the end though, we managed to create a fully functional dice game entirely made through Eclipse v. 4.6.0.

2 Indledning

Spilindustrien har igennem de seneste tredive år taget stormskridt. Fra den spæde begyndelse med klassikere som Pong og Space Invaders med sit simple mål med blot at vinde spillet. Til de helt nye mesterværker som The Witcher-serien, Star Citizen og World of Warcraft hvor spillerne skaber historien. Spillene er gået fra blot at være et virtual tidsfordriv, til at blive til et helt nyt medie i det 21. århundrede.
Nye spil har udviklet sig til at blive små mini-universer, med personer, historier og scenarier som inddrager brugerne i langt højere grad end tidligere.

Når man skal skabe et spil, vælger stadigt flere udviklere at gøre deres software så livagtigt som muligt. En meget enkel måde at skabe både en følelse af autencitet og retfærdighed hos spilleren, er ved at bruge en simpel tilfældig tal-generator. På engelsk, en Random Number Generator (RNG).
Det er dog lidt snyd, at kalde generatoren for ”tilfældig”, da man i oftest i computere benytter sig af en pseudo-tilfældig tal generator (PRNG). Ved hjælp af algoritmer og beregninger baseret på computerens daværende clock-hastighed og andre parametre, frembringes et ”tilfældigt”tal.

Denne opgave, som består i at skabe et terninge spil, har et RNG-segment som en helt essentiel del af koden. At terningerne viser forskellige øjne efter hvert slag, og at sandsynligheden for at slå x-antal øjne er lige så stor som sandsynligheden for at slå y-antal øjne, er vigtig. Begge spillere skal nemlig have lige stor chance for at vinde, da et spil der favoriserer én af spillerne ikke er særligt sjovt for den anden spiller.

3 Problemformulering

Kunden har stillet os opgaven, at lave et terninge spil, for to spillere. I spillet skal man kaste et raflebæger med to terninger. Værdien af de to terninger skal ligges til spillerens score. Slår man to ens får man et kast til. Hvis man slår to 1’ere mister man sine point. Slår man et par seksere og efterfølgende endnu et par seksere, så vinder man spillet med det samme.
Det forventes af kunden, at spillet kan spilles uden en brugsanvising. Samtidig forventes det, at der er foretaget en test der viser at raflebægeret virker korrekt.
Vi skal derfor i dette projekt, lave et java program der emulere overstående terninge spil. Der skal derfor laves et stykke kode, som modtager kommandoer fra brugeren gennem konsollen. Disse kommandoer bruges til at fortælle programmet hvad spilleren ønsker at gøre. Når spillet er færdigt skal der er eklæres en vinder.

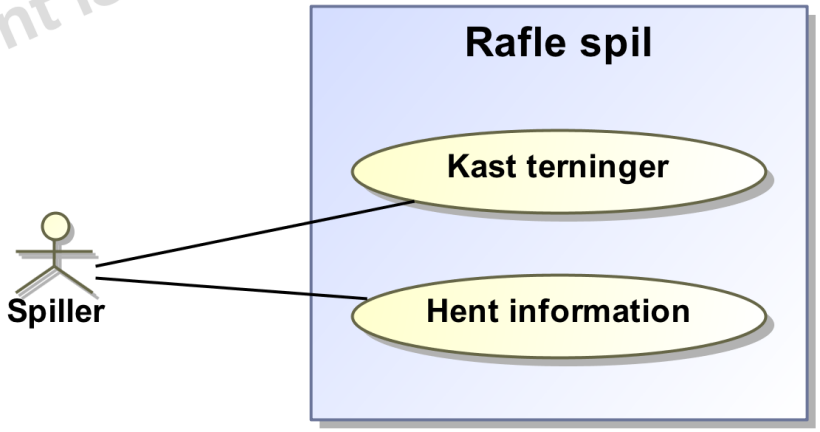
4 Krav specifikation

Vi vil nu opstille de krav som kunden specifikt har stillet, og de krav som vi har kunne udtrække af kundens beskrivelse af produktet. Kravene er som følger:

- K01: Terningespillet skal gå ud på at man slår med et raflebæger med to terninger og ser resultatet med det samme.
- K02: Spillet skal kunne køre på maskinerne (Windows) i databarene på DTU.
- K03: Spillet skal kun kunne spilles af to spillere.
- K04: Summen af terningernes værdier skal lægges til spillerens point.
- K05: En spiller skal vinde spillet, hvis én af følgende krav er opfyldt.
  - Spilleren slår to ens og har 40 point.
  - Spilleren slår to 6'ere to gange i træk.
- K06: En spiller skal have én ekstra tur, hvis spilleren slår to ens.
- K07: En spiller der slår to 1'ere skal miste alle sine point. Medmindre spilleren har 40 point.
- K08: Spillet skal kunne spilles af almindelige mennesker uden en brugsanvisning.
- K09: Terningerne skal være fair.
- K10: Spillet skal kunne spilles fra konsollen.
- K11: Programmet skal kunne håndtere fejl-input og give en passende fejlmeddelse.
- K12: Spillets responstid skal være mindre end 333 millisekunder per tur.
- K13: Spillerne skal have lige stor chance for at vinde, også over flere spil.
- K14: Terningerne i raflebægeret skal være sekssidet.

5 Design

Use case diagram



Figur 3: Use case diagram der viser de to mulige use cases.

Figur 3 illustrerer et use case diagram over vores terningespil, kaldet ”Rafle spil”. I diagrammet, danner raflespillet en grænse (boundary) for spillernes muligheder. De kan enten kaste med terningerne eller hente informationer omkring spillet.

Der er i realiteten 2 spillere, men da begge spillere har samme muligheder i spillet, er der kun indsat én aktør.

Navne- og udsagnords analyse

For at finde ud af hvilke klasser og metoder vi skal bruge i terningespillet, har vi udført en navne- og udsagnsordsanalyse. På baggrund af kundens vision, har vi opstillet de forskellige navneord og udsagnsord i en tabel.

Navneord	**Udsagnsord**
- System - Maskine - Databar - Spil - Person - Raflebæger - Terning - Resultat - Sum - Værdi - Point - Vinder - Tur - Kast - GUI	- Kaste terninger - Se resultat - Opnå point - Lægge point til score - Miste point - Få ekstra tur - Spille tur

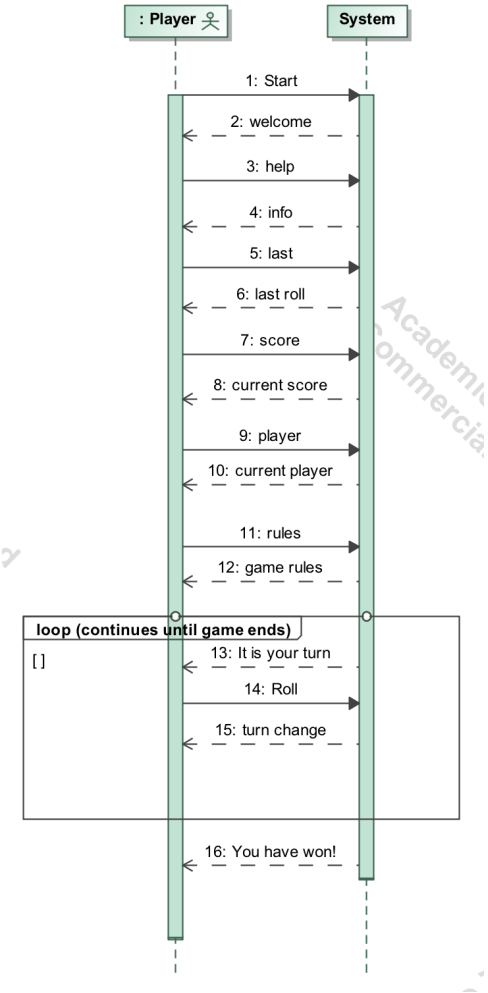
Tabel 1: Navne- og udsagnsord fra navne- og udsagnsords analysen.

I Tabel 1 ses de forskellige navneord og udsagnsord fra analysen, og vi har ud fra denne analyse valgt at oprette følgende klasser og metoder i programmet.

Klasser:
Spil (Game), Person (Player), Raflebæger (DiceCup), Terning (Die).
Derudover har vi oprettet en klasse ”TUI”, som håndterer brugerens indput og returnerer et tekst-output.

Metoder:
Kaste terning (rollDie), kaste terninger (rollDice), opnå point (setScore), se resultat (getScore), miste point (resetScore), spille tur (playTurn).
Vi har selvfølgelig flere metoder end de ovenstående.

System sekvensdiagram



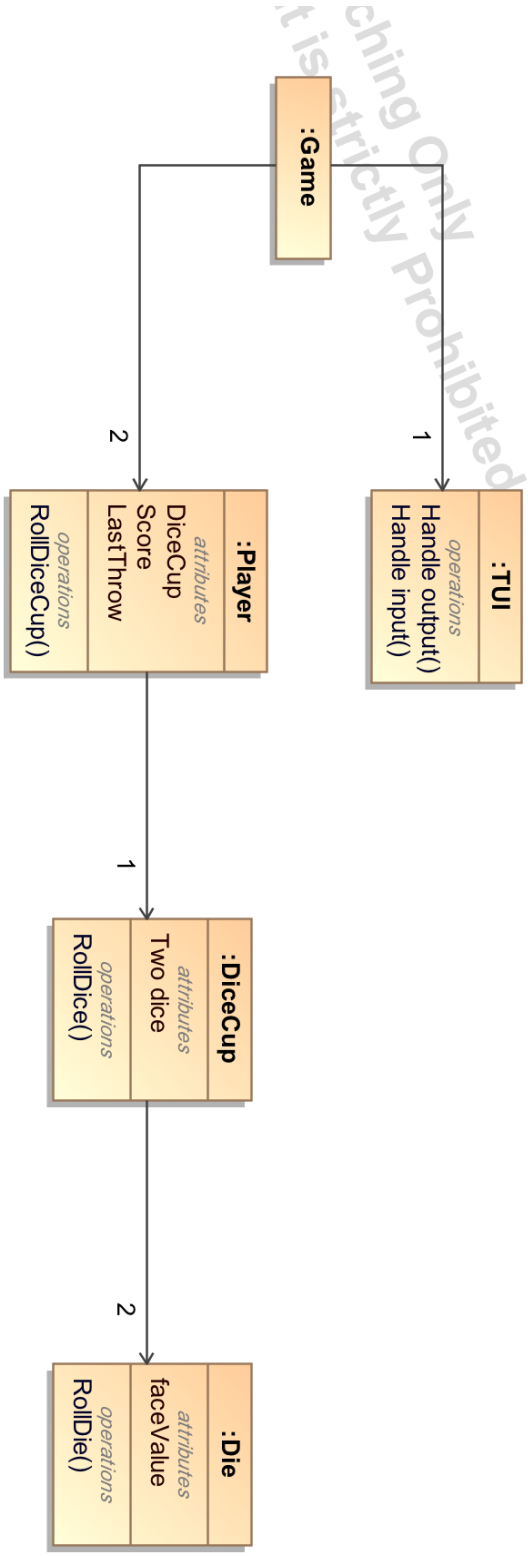
Figur 4: System sekvensdiagram der viser et eksempel på hvordan spilleren kan kommunikere med spillets konsol, og udtrække informationer derfra.

System sekvensdiagrammet beskriver de forskellige kommandoer spillerne kan give systemet. Herunder de fem hjælpe kommandoer. Help, last, score, player og rules. De returnerer så forskellige informationer.

- ”help” fortæller hvilke kommandoer der kan skrives.
- ”last” fortæller hvad du slog i sidste tur.
- ”score” fortæller hvad din score er.
- ”player” fortæller hvis tur det er.
- ”rules” fortæller spillets regler.

Loopet i diagrammet illustrerer hvordan spilleren kan bruge kommandoen roll til at foretage en tur. Når en spiller har vundet slutter programmet.

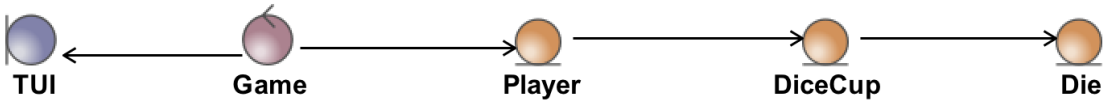
Domænemodel



Figur 5: Domænemodel diagram der viser de primære funktioner i klasserne og deres sammenhæng med hinanden.

Vi har valgt at have fem klasser. Game som har main metoden styrer spillet. TUI er en klasse der håndtere input og output fra konsollen, hvor game kommunikere med TUI. Så har vi lavet klassen Player som indeholder et raflebæger, en score, det forrige samt det nuværende slag spilleren har foretaget. Player kan desuden slå med raflebægeret. Vi har to klasser til at hjælpe Player-klassen. DiceCup der repræsenterer raflebægeret og Die som repræsenterer terningerne i raflebægeret.

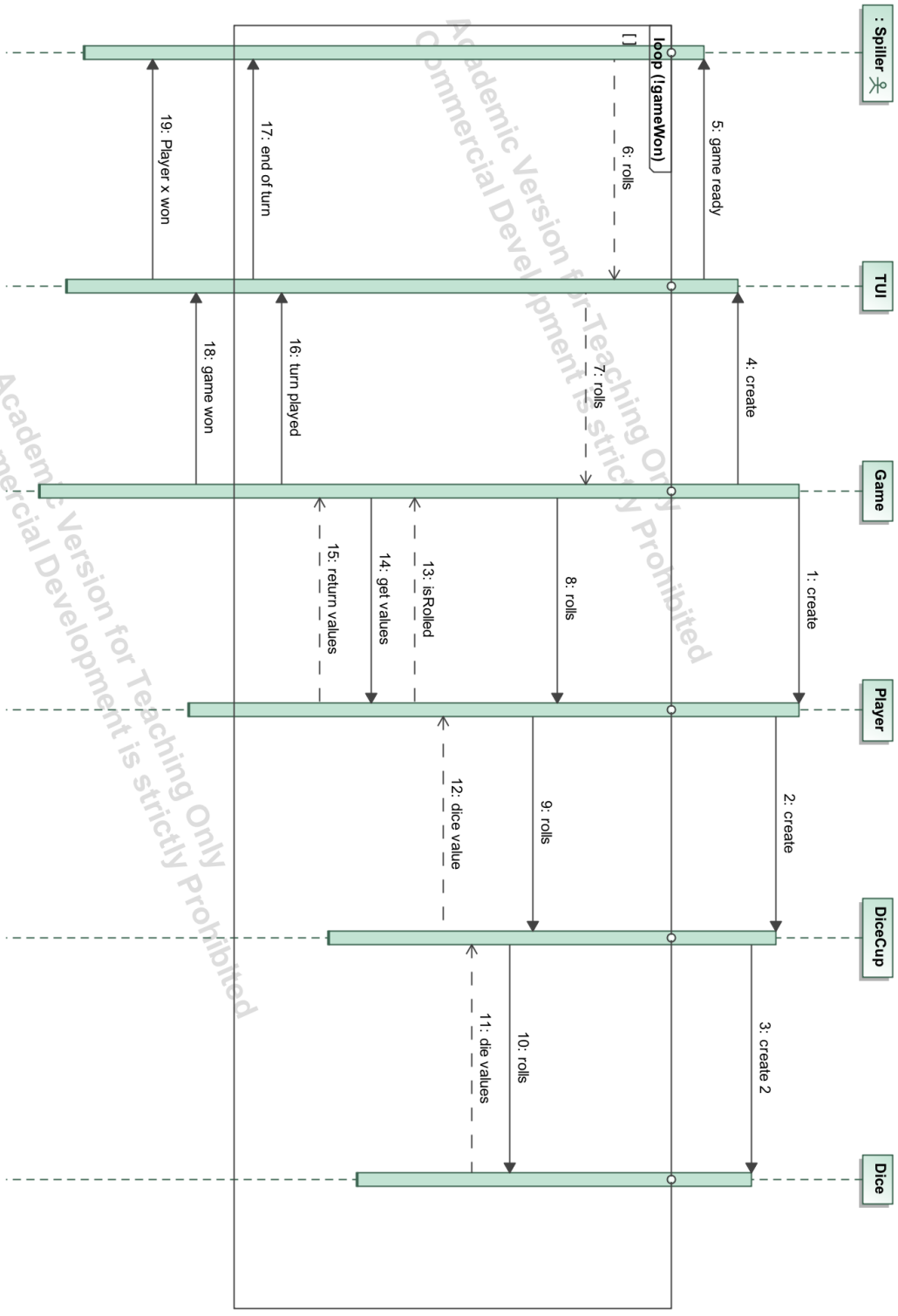
BCE model



Figur 6: BCE model der viser hvad de forskellige klasser gør.

Vores BCE-model (Boundary Control Entity-model) viser interaktionen mellem de forskellige segmenter i vores program. Game fungerer som controller der styrer interaktionen mellem brugerens input (håndteret af boundry klassen TUI), og Entity klassen Player. Player har adgang til DiceCup-klassen, som igen har adgang til en Die-klasse.

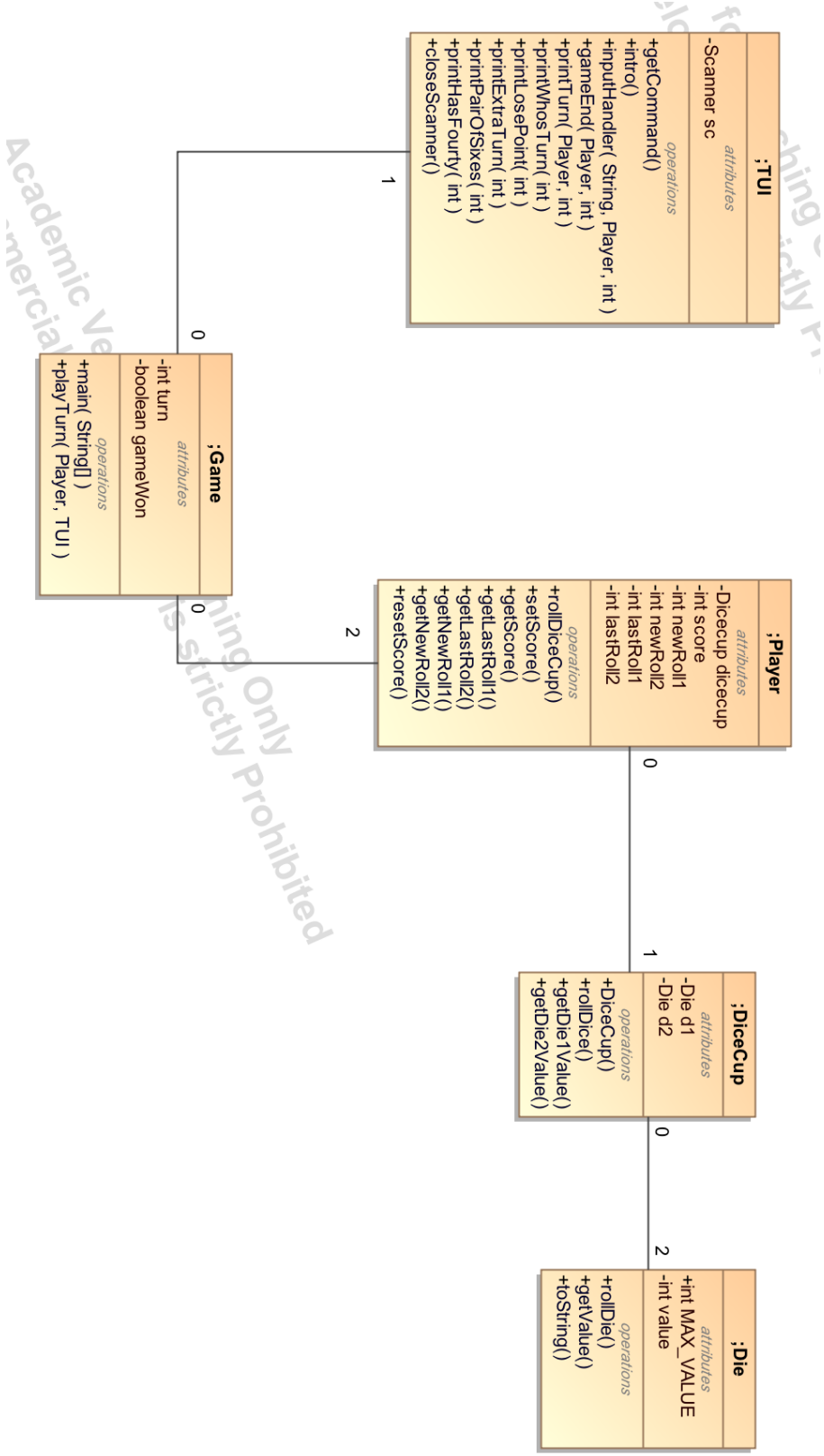
Design-sekvensdiagram



Figur 7: Design-sekvensdiagram der viser et eksempel på en spillers spil.

På sekvensdiagrammet kan det ses at spillet først opretter de forkellige klasser. Dernæst bliver spilleren bedt om, gennem TUI'en, at slå med terningerne. Dernæst fortæller TUI'en Game, at spilleren vil slå, og Game kalder derfor metoden i Player. Player slår så med DiceCup som slår de to Die objekter i DiceCup. Dette gentages indtil en spiller har vundet hvorefter vi forlader loopet. Der gives så en besked om at spilleren har vundet.

Design-Klasse diagram



Figur 8: Design-klassediagram med navngivne relationer.

Overstående klassediagram viser de fem klasser vi har i programmet. TUI, Game(main), Player, DiceCup og Die. Game indeholder Main metoden som styrer spillet. Den har en metode, playTurn, der modtager en spiller og TUI, og derefter spiller den en tur for den givne spiller. Den har en instans-variabel ”turn”som indeholder hvis tur det er. Derudover indeholder den en variabel ”gameWon”, der markerer om én af spillerne har vundet. TUI klassen er vores in- og out-put klasse. Den har metoden getCommand(), som tager imod kommandoer fra consollen. De resterende metoder printer information til spilleren i forskellige situationer. Det er Game der styrer hvornår TUI'en skal printe ting ud.
Vi har desuden en spiller klasse. Player har en score, en DiceCup og fire variable, der beskriver hvilke kast han har foretaget. Player har en metode der slår med DiceCup og derefter ændrer de gemte terninge værdier sig til de nye værdier. De fire variable er delt i to grupper: ”newRoll”som beskriver det seneste kast og ”lastRoll”som beskriver kastet før dette. Player har også metoder der kan hente de 4 variable. Player har desuden metoder der kan tilføje point til scoren, sætte score til 0 og hente scoren.
DiceCup metoden indeholder 2 terninger, og den kan slå med disse med rollDice metoden. Den har desuden get metoder til de to terninge værdier.
Tilslut har vi terningen som har en værdi mellem 1 og 6. Den har 2 metoder, rollDie som slår med terningen og getValuesom henter terningens værdi.

6 Implementering

Vi har implementeret koden i et while loop, der kører indtil en spiller har vundet. Inde i loopet sker to ting. Først tjekkes input fra konsollen i en loop, dernæst spilles en tur når spilleren har bedt om dette.

```
1                       // Lkken krer s lnge der ikke bliver skrevet "roll"
                           eller "" i
2                       // konsollen.
3                       do {
4                               // Srger for koden ikke kres i frste
                                   gennemkrsel.
5                               if (!consoleInput.equals("unassigned")) {
6                                       // Printer beskeder til spilleren
                                           udfra input
7                                       if (turn == 0)
8                                               console.inputHandler(
                                                   consoleInput, p1, turn);
9                                       else
10                                              console.inputHandler(
                                                   consoleInput, p2, turn);
11                              }
12                              // Henter en ny command fra brugeren.
13                              consoleInput = console.getCommand();
14                      } while (!(consoleInput.equals("roll") ||
                           consoleInput.equals("")));
```

Her kan det ses at vi har en string ”consoleInput”, som indholder det som sidst blev tastet fra consollen. Ved første gennemkørsel er denne sat til ”unassigned”. Det kan ses at når vi kører programmet, så tjekkes det først om consoleInput er ”unassigned”. Hvis den ikke er dette, så

sendes ”consoleInput”tilbage til TUI, som giver en passende besked til spillerne via konsollen. Dernæst bliver TUI bedt om at få en ny kommando fra spilleren. Hvis den nye kommando spiller giver er ”roll”eller en tom ”String”, vil spilleren gerne slå med terningerne, så vi forlader loopet og så spiller programmet nu en tur.

```
if (turn == 0)
        playTurn(p1, console);
else
        playTurn(p2, console);

// Stter hvis tur det er
turn = (turn + 1) % 2;
```

Vi kan se her at if statementet tjekker hvis tur det er og kalder metoden ”playTurn”i Game. Playturn er for stor til at vise i rapporten, men den slår med spillerens raflebæger, og tjekker om spilleren har vundet. Hvis han har vundet så sættes gameWon til true, så spillet slutter. Hvis han ikke har vundet så giver playturn ekstra slag eller fjerner point hvis det er relevant.

7 Test

Efter at have færdiggjort vores kodning, skulle vi nu teste hvorvidt vores kode indlæste, og behandlede inputs og outputs korrekt. Både fra brugerne af systemet, og systemets interne enheder.

Vi benytter os af Eclipses indbyggede JUnit-tester til at gennemgå vigtige segmenter af vores kode. Vi tester 18 individuelle use cases, arrangeret i denne kravmatrix (traceability):

		Use cases:																	
		UD01	UD02	UD03	UD04	IG01	IG02	IP01	IP02	IP03	IP04	IP05	IP06	IP07	IP08	IP09	IDC01	IDC02	IDC03
Requirements:	K01																		
	K02																		
	K03																		
	K04								■	■									
	K05					■													
	K06						■												
	K07						■				■								
	K08																		
	K09	■																	
	K10																		
	K11																		
	K12																		
	K13					■													
	K14	■	■																

Figur 9: Kravmatrix med Use casene arrangeret horisontalt, og de opsatte krav fra problemformuleringen vertikalt. Samtlige formelle krav kan findes i afsnit 10.1

Samlet liste over alle test cases:

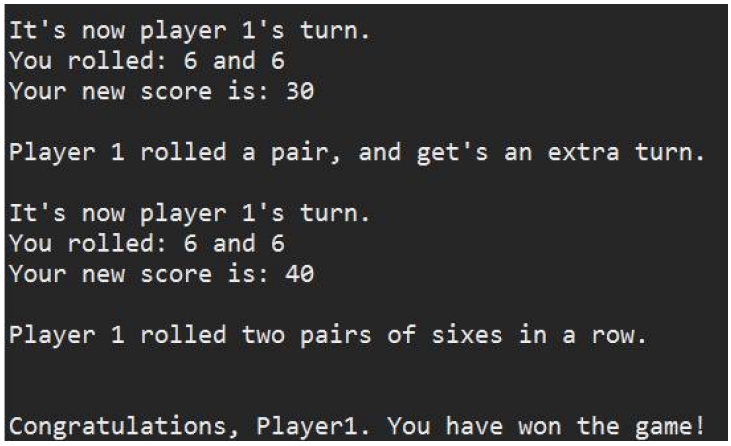
- Test casene UD01-04 tester Die-klassen:
  - UD01 tester hvorvidt udfaldschancen for at slå en af de 6 øjenværdier er en sjettedel.

  - UD02 tester hvorvidt at øjenværdierne for hver af de to terninger ligger imellem 1 og 6.
  - UD03 tester om kaldet ”getValue”returnerer de korrekte værdier.
  - UD04 tester om metoden ”toString”returnerer den korrekte ”String-repræsentation.
- Test casene IG01-02 tester Game (main) klassen, og ”playTurn”metoden:
  - IG01 lader spillet køre 10.000 gange, og undersøger sandsynligheden for at hver spiller vinder. Den undersøger hvor mange gange et spil bliver vundet ved at slå to gange to seksere. Der undersøges også om en af spillerne vinder selvom det ikke er berettiget (fejlagtig afslutning af spillet).
  - IG02 lader igen spiller køre 10.000 gange, og undersøger om spillerne som slår to ens korrekt får en ekstra tur, samt om de korrekt mister alle point hvis de slår to ettere.
- Test casene IP01-09 tester Player klassen, og dennes attributer.
  - IP01 tester om metoden ”rollDiceCup”sætter de korrekte værdier til den individuelle ternings forrige og nuværrende slag.
  - IP02 tester om terningernes værdi korrekt bliver adderet til spillerens score.
  - IP03 tester om det er muligt for en spiller at optjene en score over 40 point.
  - IP04 tester om metoden ”resetScore”korrekt nulstiller spillerens score til 0.
  - IP05 tester om metoden ”getScore”returnerer den korrekte værdi for den aktive spillers score.
  - IP06-IP09 tester om metoderne ”getLastRoll1”, ”getLastRoll2”, ”getNewRoll1”og ”getNewRoll2”returnerer de korrekte værdier for Die1 og Die2 i spillerens forrige og nuværrende tur.
- Test casene IDC01-03 tester DiceCup klassen, og dennes metoder.
  - IDC01 tester om metoden ”rollDice”giver en realistisk repræsentation af en sekssidet terning.
  - IDC02 tester om metoden ”getDieValue1”fungerer korrekt, og returnerer det korrekte svar fra Die-klassen.
  - IDC03 tester om metoden ”getDieValue2”fungerer korrekt, og returnerer det korrekte svar fra Die-klassen.

Alle disse formelle tests er med til at sikre at vores produkt virker som det skal, og er funktionsdygtigt. Næsten alle testene kom tilbage positivt bekræftende, at de returnerede de korrekte værdier.

En test der fejlede var IG01. Hvis en spiller slog et par seksere, og derefter slog et par seksere så han når en score på 40 i sit andet bonus slag, vinder han ikke spillet automatiskt. Dette selvom han ellers opfylder præmisset med at slå et par seksere to gange i træk. Denne fejl er nu blevet rettet, og vores spil fungerer igen som forventet.

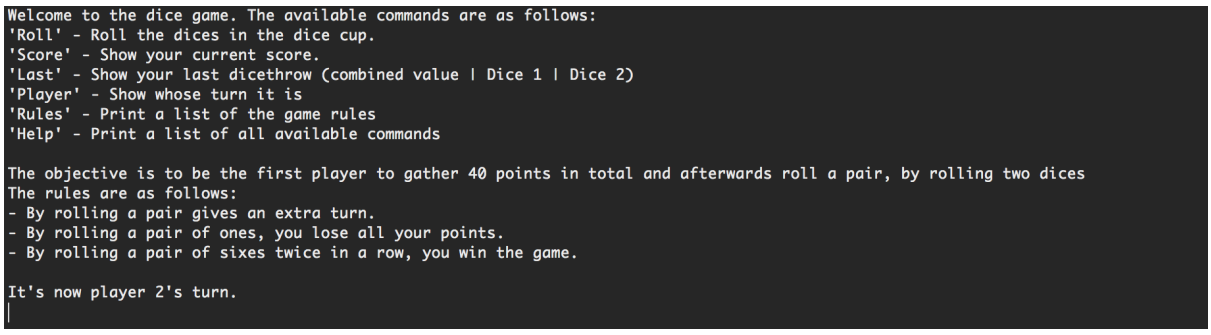
IG01 fandt også en anden fejl der gjorde at spillet favoriserede spiller 1. Spiller 1 startede altid spillet, og havde derfor bedre mulighed for at samle flere points sammen hurtigere end spiller 2. Vi valgte derfor at vælger en tilfældig spiller i starten af programmet, som giver hver spiller lige stor chance for at starte.



Figur 10: Bevis for at fejlen er blevet korrigeret i vores kode, og at spiller to nu korrekt vinder, med to gange to par seksere.

7.1 Acceptance test

I samarbejde med kunden køres programmet nogle gange på en maskine (Windows) i en databar på DTU og der tjekkes om programmet lever op til kravspecifikationen. Programmet starter, som forventet. Krav K02 er dermed opfyldt. Det første vi ser, når programmet åbnes, er en velkomstskærm til terninge spillet og hvilke kommandoer spillerne har mulighed for at bruge. Vi kan også ses spillets regler og hvem der starter spillet. Velkomstskærmen kan ses på figur 11.



Figur 11: Spillets velkomstskærm.

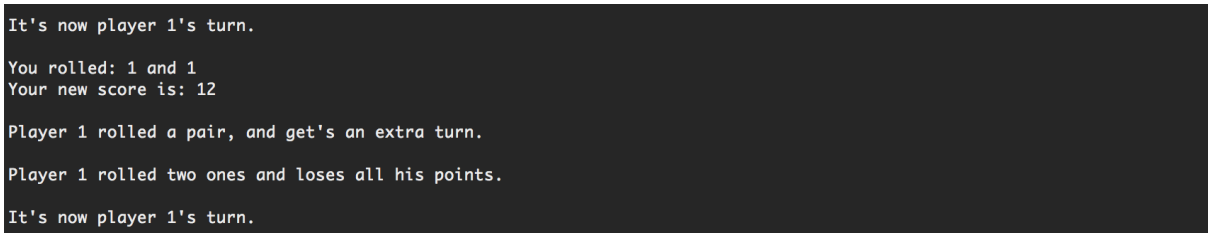
Kunden finder startskærmen overskuelig, imødekommende og med de rette informationer, men syntes det er uklart hvordan en kommando aktiveres (en kommando aktiveres ved at skrive kommandoen i konsollen, hvorefter der klikkes på ”Enter”for at aktivere den). Sælgeren argumenterer for at eftersom kunden bad om at spillet skulle kunne spilles i konsollen, forventes det, at kunden ved hvordan konsollen virker, og dermed at hvordan en kommando i konsollen aktiveres. Krav K08 og K10 er dermed opfyldt.

Kunden kaster med terningerne i spillet og ser umiddelbart resultatet kort efter. Derudover ses det at summen af terningernes øjne bliver lagt til kundens score umiddelbart efter kastet. Krav K01 og K04 er dermed opfyldt. Dette udsagn kan ses på figur 12.

Figur 12: Opfyldelse af krav K01 og K04.

Kunden ser at turen går videre til spiller 1, som forventet. Det er altså klart, at spillet er designet til kun 2 spillere. Krav K03 er dermed opfyldt.

Senere i spillet ser kunden, at når der bliver slået et par, så får spilleren, der har slået parret, et ekstra slag. Derudover ser han at når parret der blivet slået er et par ettere, så mister spilleren alle sine point. Begge dele som forventet. Krav K06 er dermed opfyldt og K07 er delvist opfyldt. Dette scenarie ses også i figur 13.



Figur 13: Opfyldelse af K06 og K07.

Derudover bemærker kunden, at hvis der bliver indtastet noget i konsollen, som spillet ikke genkender, som en kommando, så bliver der givet en fejlmeddelse, hvori man bliver bedt om at indtaste en ny kommando. Krav K11 er dermed opfyldt. Dette senarie kan ses på figur 14.

Figur 14: Opfyldelse af K11.

I slutningen af et spil ser kunden at når en spiller opnår 40 points, bliver spilleren bedt om at slå et par for at vinde spillet. Dette udsagn kan ses på figur 15.

```
It's now player 2's turn.

You rolled: 4 and 6
Your new score is: 37

It's now player 1's turn.

You rolled: 6 and 2
Your new score is: 32

It's now player 2's turn.

You rolled: 3 and 3
Your new score is: 40

Player 2 has achieved 40 points. Player 2 must now roll a pair to win.
```

Figur 15: En spiller har opnået 40 point.

Ydermere ser han, at selv om en spiller slår et højere antal øjne end der manglede for at opnå 40 points, så kan en spiller ikke få mere end 40 points.

Senere i samme spil slår en spiller et par ettere efter at have opnået 40 point og vinder spillet, som forventet. Krav K07 er derfor nu helt opfyldt og krav K05 delvist opfyldt. Dette kan ses på figur 16.

```
It's now player 2's turn.

You rolled: 1 and 1
Your new score is: 40

Congratulations, Player2. You have won the game!
```

Figur 16: En spiller vinder efter at have slået par 1 efter at have fået 40 point.

I en anden gennemkørsel af spillet slår en spiller et par seksere to gange i træk og vinder, som forventet. Krav K05 er nu helt opfyldt. Dette ses af figur 17.

```
It's now player 1's turn.
You rolled: 6 and 6
Your new score is: 30

Player 1 rolled a pair, and get's an extra turn.

It's now player 1's turn.
You rolled: 6 and 6
Your new score is: 40

Player 1 rolled two pairs of sixes in a row.


Congratulations, Player1. You have won the game!
```

Figur 17: En spiller vinder ved at slå par 6 to gange i træk.

Kunden finder spillets respons-tid ved gennemgangen af programmet med kunden acceptabel. Krav K12 er dermed opfyldt.

Sælgeren og kunden gennemgår kravspecifikationen og ser at krav K09, K13 og K14 indtil videre ikke er bevist opfyldt. Sælgeren gennemgår da de unit- og integrations-test med kunden der tester netop disse krav (De specifikke tests har id UD01, UD02, og IG1). Krav K09, K13 og K14 er dermed opfyldt.

Alle krav i kravspecifikationen er nu opfyldt, og kunden accepterer produktet.

8 Forbedringsforslag

I løbet af dette projekt er vi undervejs faldet over forskellige udfordringer, både fra opgavebeskrivelsen, og uforudsete konflikter imellem vores kodesegmenter. Dette er vigtigt, både for spillets forløb og dets videre udvikling.

Derudover faldt vi over at to af ekstraopgaverne modsagde hinanden. En af ekstraopgaverne går ud på at hvis en spiller slå to ettere, skal personen miste alle sine point. Samtidig skal en spiller dog slå to ens efter at have nået 40 point for at vinde spillet. Hvad skal programmet så gøre hvis en spiller, som har 40 point, slår to ettere, skal personen så vinde, eller miste alle sine point? Vi valgte at kode vores program til at en spiller med 40 point, som slår to ettere, skal vinde spillet. Vi vælger altså, i dette tilfælde, at se bort fra det tidligere præmis om at spilleren mister sine point.

Dette gør sig også gældende for ekstraopgaven om at et slag med et par seksere skulle give spilleren et ekstra slag. Vi har dog valgt et at en spiller med 40 point, som slår to seksere automatisk vinder spillet.

Vi har lavet vores spil uden brug af arrays, da vi ikke kendte til disse, da vi skrev programmet. Måden hvorpå programmet håndterer hvis tur det er, kan gøres nemmere ved at ligge spillerne i et array. Dernæst kan man blot kalde Array[spillerNr] for at vælge en tur. Det ville gøre koden simplere og lettere at forstå.

9 Konklusion

I denne rapport har vi beskrevet fremstillingen af et terninge spil ønsket af kunden. Først kiggede vi på kundensvision og opskrev de krav som var nødvendige for at tilfredsstille visionen. Vi opstillede desuden flere krav der også var nødvendige for at programmet opfyldte kundenskrav. Vi har i denne rapport også tilføjet diverse diagrammer, som illustrerer udviklingsprocessen af programmet.
Efter det oprindelige design er vi begyndt på implementering af koden og ændret diagrammerne der hvor det har været nødvendigt. Koden er delt op i fem klasser, en TUI som håndere in og ouput. Game som kører spillet, og har to Player objekter. Disse indeholder hver et Dicecup-objekt med to terninger.

Efter at have implementeret koden begyndte vi at teste. Vi fandt et par små stavefejl som vi rettede til. Vi fandt desuden en ret væsentlig fejl som gjorde at en spiller som burde have vundet ikke vandt i det specielle tilfælde. Vi har nu fikset fejlen.
Efter testene er vi noget frem til et par forbedringer vi kan indfører. Blandt andet er der uklarhed om når man slår to ettere og har 40 point. Skal man så miste sine point eller skal man vinde spillet. Vi har også konkluderet at koden kan effektiviseres ved brug af arrays. Da vi kan slippe for nogle if statements og dermed gøre koden lettere læselig.
Alt i alt er vi noget frem til et fungerende terningespil der opfylder kravene for kunden. Der er dog et par steder hvor der er uklarhed, og hvor der kan ske forbedringer.

10 Bilag

10.1 Formelle Test

Følgende er en pdf med alle de formelle test vi har foretaget. De er markeret med seperatte sidetal.

Test case ID	UD01
Summary	Tests that the die is fair, by measuring the probability of the different rolls to be equal to 1/6.
Requirements	Requirement specification K09 and K14
Precondtions	None
Postconditions	None
Test procedure	Roll the die 60000 times and see wheter the six facevalues are equal with a maximum deviation of 4%.
Test data	None
Expected result	None
Actual result	None
Status	Passed
Tested by	Mikkel Holmbo Lund
Date	5. oktober 2016
Test environment	Eclipse 4.6.0 on Windows 10 64-bit

Test case ID	UD02
Summary	Tests that the faceValue of the die is between one and six, both inclusive.
Requirements	Requirement specification K14
Precondtions	None
Postconditions	None
Test procedure	None
Test data	None
Expected result	None
Actual result	None
Status	Passed
Tested by	Mikkel Holmbo Lund
Date	5. oktober 2016
Test environment	Eclipse 4.6.0 on Windows 10 64-bit

Test case ID	UD03
Summary	Tests that the method getValue returns the correct value.
Requirements	None
Precondtions	None
Postconditions	None
Test procedure	None
Test data	None
Expected result	4
Actual result	4
Status	Passed
Tested by	Mikkel Holmbo Lund
Date	5. oktober 2016
Test environment	Eclipse 4.6.0 on Windows 10 64-bit

Test case ID	UD04
Summary	Tests if the method toString returns the correct string representation
Requirements	None
Precondtions	None
Postconditions	None
Test procedure	None
Test data	None
Expected result	"The value of the die is: 4"
Actual result	"The value of the die is: 4"
Status	Passed
Tested by	Mikkel Holmbo Lund
Date	5. oktober 2016
Test environment	Eclipse 4.6.0 on Windows 10 64-bit

Test case ID	IG01
Summary	This test runs the game 10000 times and test the probablity of winning. It also test that no one win by unseen circumstances.
Requirements	Requirement specification K05 and K13
Precondtions	None
Postconditions	None
Test procedure	Run the game 10.000 times and count how many times player 1 and player 2 win. Everytime a player wins, the test is checked to see if the win was legal.
Test data	None
Expected result	The game is fair within a statistical margin and no one won in unforseen cicurmstances.
Actual result	The game is fair within a statistical margin. After 10000 games the score was 5015 for player 1 and 4985 for player 2.
Status	Passed
Tested by	Arvid Langsø
Date	6. oktober 2016
Test environment	Eclipse 4.6.0 on Windows 10 64-bit

Test case ID	IG02
Summary	This test runs the game 10000 times. It testes if players get the correct extra throws and loose their points when they roll two ones.
Requirements	Requirement specification K06 and K07
Precondtions	None
Postconditions	None
Test procedure	Run the game 10.000 times and count how many times a player gets and extra trow and how many times they loose points.
Test data	None
Expected result	The test doesn't find any circumstances where players get unforseen extra throws or looses all point when they don't roll double ones.
Actual result	The test found no unforseen events.
Status	Passed
Tested by	Arvid Langsø
Date	6. oktober 2016
Test environment	Eclipse 4.6.0 on Windows 10 64-bit

Test case ID	IP01
Summary	Tests that the method rollDiceCup set's the correct values for the dice in the previous and current turn
Requirements	None
Precondtions	None
Postconditions	None
Test procedure	None
Test data	None
Expected result	None
Actual result	None
Status	Passed
Tested by	Jeppe Thougaard Nielsen
Date	06-10-2016
Test environment	Eclipse 4.6.0 on Windows 10 64-bit

Test case ID	IP02
Summary	Test that the dice value is added to the player's score.
Requirements	Requirement specification K04
Precondtions	None
Postconditions	None
Test procedure	1. Player rolls the dice 2. The values of the dice gets added to the player's score.
Test data	score = 0 dice1 = 2 dice2 = 4
Expected result	The current score plus the sum of the currently rolled dice.
Actual result	6
Status	Passed
Tested by	Jeppe Thougaard Nielsen
Date	06-10-2016
Test environment	Eclipse 4.6.0 on Windows 10 64-bit

Test case ID	IP03
Summary	Tests if a player can have a score above 40 points
Requirements	Requirement specification K04
Precondtions	None
Postconditions	None
Test procedure	1. Player rolls the dice 30 times 2. The values of the dice each turn gets added to the players score.
Test data	score = 0 dice1 = 1 to 6 times 30 dice2 = 1 to 6 times 30
Expected result	The score value should be max 40 points
Actual result	The score is 40
Status	Passed
Tested by	Jeppe Thougaard Nielsen
Date	06-10-2016
Test environment	Eclipse 4.6.0 on Windows 10 64-bit

Test case ID	IP04
Summary	Tests that the method resetScore resets the players score.
Requirements	Requirement specification K07
Precondtions	None
Postconditions	None
Test procedure	1. The player rolls the dice 5 times. 2. The players now have points added to his score. 3. The players score gets reset.
Test data	score > 0
Expected result	The value of score should be 0 after a score reset.
Actual result	0
Status	Passed
Tested by	Jeppe Thougaard Nielsen
Date	06-10-2016
Test environment	Eclipse 4.6.0 on Windows 10 64-bit

Test case ID	IP05
Summary	Tests if the method getScore returns the correct value of the players score.
Requirements	None
Precondtions	The player rolls the dice.
Postconditions	
Test procedure	1. Sets the value of the current rolled dice 2. Sets the players score with the sum of the rolled dice. 3. Return the players score.
Test data	score=0 dice1=2 dice2=2
Expected result	The value of score is a sum of the dice.
Actual result	4
Status	Passed
Tested by	Jeppe Thougaard Nielsen
Date	06-10-2016
Test environment	Eclipse 4.6.0 on Windows 10 64-bit

Test case ID	IP06
Summary	Tests if the method getLastRoll1 returns the correct value of the rolled die1 in the players previous turn.
Requirements	None
Precondtions	The player rolls the dice two times.
Postconditions	
Test procedure	1. Player rolls the dice in turn 1. 2. Player rolls the dice in turn 2. 2. The value of die1 from turn 1 gets returned.
Test data	The value of die1 gets set to newRoll1 in turn 1. The method getLastRoll1 gets called in turn 2.
Expected result	The value of lastRoll1 is equal to newRoll1 in turn 1.
Actual result	true
Status	Passed
Tested by	Jeppe Thougaard Nielsen
Date	06-10-2016
Test environment	Eclipse 4.6.0 on Windows 10 64-bit

Test case ID	IP07
Summary	Tests if the method getLastRoll2 returns the correct value of the rolled die2 in the players previous turn.
Requirements	None
Precondtions	None
Postconditions	
Test procedure	1. Player rolls the dice in turn 1. 2. Player rolls the dice in turn 2. 2. The value of die1 from turn 1 gets returned.
Test data	The value of die2 gets set to newRoll2 in turn 1. The method getLastRoll2 gets called in turn 2.
Expected result	The value of lastRoll2 is equal to newRoll2 in turn 1.
Actual result	true
Status	Passed
Tested by	Jeppe Thougaard Nielsen
Date	06-10-2016
Test environment	Eclipse 4.6.0 on Windows 10 64-bit

Test case ID	IP08
Summary	Tests if the method getNewRoll1 returns the correct value of the rolled die1 in the players current turn.
Requirements	None
Precondtions	None
Postconditions	
Test procedure	1. Sets the value of newRoll1. 2. Returns the value of newRoll1.
Test data	newRoll1 = 3
Expected result	The value of newRoll1 should be 3.
Actual result	3
Status	Passed
Tested by	Jeppe Thougaard Nielsen
Date	06-10-2016
Test environment	Eclipse 4.6.0 on Windows 10 64-bit

Test case ID	IP09
Summary	Tests if the method getNewRoll2 returns the correct value of the rolled die2 in the players current turn.
Requirements	None
Precondtions	None
Postconditions	
Test procedure	1. Sets the value of newRoll2. 2. Returns the value of newRoll2.
Test data	newRoll1 = 1
Expected result	The value of newRoll1 should be 1.
Actual result	1
Status	Passed
Tested by	Jeppe Thougaard Nielsen
Date	06-10-2016
Test environment	Eclipse 4.6.0 on Windows 10 64-bit

Test case ID	IDC01
Summary	Tests if method "rollDice" is a real dice
Requirements	None
Precondtions	None
Postconditions	None
Test procedure	rollDice
Test data	None
Expected result	Dice value is within 1..6
Actual result	Dice value is within 1..6
Status	Passed
Tested by	Simon Engquist
Date	06-10-2016
Test environment	Eclipse 4.6.0 on Windows 10 64-bit

Test case ID	IDC02
Summary	Tests if method "getDieValue1" works
Requirements	None
Precondtions	None
Postconditions	None
Test procedure	setDieValue1(5):
Test data	None
Expected result	5
Actual result	5
Status	Passed
Tested by	Simon Engquist
Date	06-10-2016
Test environment	Eclipse 4.6.0 on Windows 10 64-bit

Test case ID	IDC03
Summary	Tests if method "getDieValue1" works
Requirements	None
Precondtions	None
Postconditions	None
Test procedure	setDieValue2(3):
Test data	None
Expected result	3
Actual result	3
Status	Passed
Tested by	Simon Engquist
Date	06-10-2016
Test environment	Eclipse 4.6.0 on Windows 10 64-bit