CDIO – DEL 1

Dice game

GRUPPE 16

Afleveringsfrist: fredag 07/10-2016 kl.23:59

Christiansen, Morten René Tang - s162682
Dahl-Jensen, Jens Martin - s165159
Nielsen, Morten Enghausen - s165150
Skouenborg, Nick - s165233
Wienziers-Madsen, Aleksander - s114750

02312-14 Indledende programmering
02313 Udviklingsmetoder til IT-systemer
02315 Versionsstyring og testmetoder

Denne rapport indeholder **49** sider incl. denne side.

Morten N

Aleksander

Nick

Morten C

Martin

1 Resumé

Denne rapport vil vise vores produktion af et spil, der går ud på at slå med to terninger. Dette spil går helt basalt ud på at det gælder om at være den første spiller til at opnå en total terningssum på 40, og derefter slå to ens, for at vinde. Dertil har vi tilføjet nogle regler kendt som 'snake-eyes', 'equals og 'them sixes'. Rapporten vil vise vores arbejdsproces i form af følgende diagrammer og modeller: Use case diagram, Navneordsanalyse, System Sekvensdiagram, og Domænemodel bruges til den analyserende del, hvor problemstillingen i programmet udvindes. BCE Model, Design-sekvensdiagram samt et Design-klassediagram bruges til at løse de problemstillinger vi har fundet frem til.

Til sidst vil vi vise, at programmet kan køre, og udføre forskellige test for at vise at de oprettede terninger giver de forventede værdier, altså vil summen 7 være den mest regelmæssige, at der ca. er en 1/6 chance for at slå to ens og til sidst at summen kun kan være mellem 2-12. Vores tanker for udviklingen af dette projekt er som udgangspunkt at arbejde agilt og i iterationer. Vi ved dog godt, at dette ikke vil være helt muligt, da vi bliver nødt til at kunne planlægge noget konkret og have en vis idé om, hvad det egentlig er vi skal lave, men vi vil dog prøve at være så agile som overhovedet muligt.

2 Timeregnskab

Dato	Deltager	Analyse	Design	Impl.	Test	Dok.	Andet	Ialt
19-09-2016	Aleksander		2,5					**2,5**
19-09-2016	Martin		2,5					**2,5**
19-09-2016	Morten C		2,5					**2,5**
19-09-2016	Morten N		2,5					**2,5**
19-09-2016	Nick		2,5					**2,5**
19-09-2016	Aleksander	2,5						**2,5**
19-09-2016	Morten C		2,5					**2,5**
19-09-2016	Nick		2,5					**2,5**
21-09-2016	Aleksander		2					**2**
21-09-2016	Martin		2					**2**
21-09-2016	Morten C		2					**2**
21-09-2016	Morten N		2					**2**
21-09-2016	Aleksander	2						**2**
21-09-2016	Morten C		2					**2**
21-09-2016	Nick		2					**2**
26-09-2016	Aleksander		2,5					**2,5**
26-09-2016	Martin		2,5					**2,5**
26-09-2016	Morten N		2,5					**2,5**
26-09-2016	Nick		2,5					**2,5**
26-09-2016	Aleksander	2,5						**2,5**
26-09-2016	Morten C		2,5					**2,5**
26-09-2016	Nick		2,5					**2,5**
27-09-2016	Martin	3,5						**3,5**
27-09-2016	Aleksander	3						**3**
27-09-2016	Nick	1,5						**1,5**
28-09-2016	Martin	0,5						**0,5**
28-09-2016	Aleksander	2						**2**
28-09-2016	Morten C	1						**1**
29-09-2016	Aleksander	5			1			**6**
30-09-2016	Morten N	3						**3**
30-09-2016	Aleksander	5	2		1			**8**
01-10-2016	Morten N	6	2					**8**
01-10-2016	Aleksander	6	1,5		1			**8,5**
01-10-2016	Morten C		3					**3**
02-10-2016	Martin					2		**2**
02-10-2016	Morten N	3		5				**8**
02-10-2016	Aleksander		1,5	4	1			**6,5**

02-10-2016	Morten C	2						**2**
02-10-2016	Nick	1						**1**
03-10-2016	Martin		2,5					**2,5**
03-10-2016	Morten N			3				**3**
03-10-2016	Aleksander			2,5				**2,5**
03-10-2016	Morten C		2,5			1		**3,5**
03-10-2016	Nick		2,5					**2,5**
04-10-2016	Martin					2		**2**
04-10-2016	Morten N				3			**3**
04-10-2016	Aleksander					2		**2**
04-10-2016	Morten C					1		**1**
05-10-2016	Martin		2			1,5		**3,5**
05-10-2016	Morten N			2	3			**5**
05-10-2016	Aleksander		2					**2**
05-10-2016	Morten C		2			1		**3**
05-10-2016	Nick		2					**2**
06-10-2016	Martin					5		**5**
06-10-2016	Morten N					5		**5**
06-10-2016	Aleksander					3		**3**
06-10-2016	Nick					3		**3**
07-10-2016	Martin					6,5		**6,5**
07-10-2016	Morten N					6,5		**6,5**
07-10-2016	Aleksander					6,5		**6,5**
07-10-2016	Morten C					6,5		**6,5**
07-10-2016	Nick					6,5		**6,5**
Timer i alt		49,5	70	16,5	10	59	0	205

3 Indholdsfortegnelse

4 Indledning

Vi har i IOOuterActive fået tildelt en opgave om at lave et spil til DTU's maskiner. Dette spil skal være forbeholdt to spillere og indebærer et raflebæger med to terninger, samt spilleregler som giver spillet et lille twist. Spillet går ud på, at to spillere skal skiftes til at slå med to terninger, hvorefter summen af øjnene på terningerne lægges til spillerens score. Den spiller, der når 40 point har mulighed for at vinde spillet ved at slå to ens, for eksempel to 5'ere. Hvis en spiller slår to ens udløser dette en equals. Hvis en spiller slår to 1'ere, fremover refereret til som, snake-eyes, mister denne spiller alle sine point. Dette gælder også, hvis spilleren har 40 point. Hvis en spiller slår to 6'ere udløser dette en chance for at aktivere "ThemSixes". Hvis en spiller herefter slår to 6'ere igen har denne spiller automatisk vundet spillet uanset score.

Kunden er kommet med en række krav, som vi har udformet til nogle kravspecifikationer, for herefter at lave nogle use cases, som demonstrerer, hvordan programmet køres igennem.

Derudover forlanger kunden også, at programmet testes med minimum 1.000 terningkast. Dette gøres for at se, om terningerne slår mellem 2-12, at 7 er det hyppigste resultat, da der er flest kombinationer for netop dette resultat, og til sidst tjekke hvor ofte de to terningers øjne er ens.

Vi har planlagt at forløbet skal foregå efter UP-principper, altså skal vi planlægge agilt og iterativt.

Herunder vil vi vise system sekvens-diagrammer, domæne-modeller og andre modeller & diagrammer.

5 Hovedafsnit

5.1 Navneordsanalyse

Heri tog vi de navne og udsagnsord vi kunne finde i teksten fra kunden, for at få navngivet vores metoder og klasser i spillet ordentligt.

5.1.1 Navneord

System - System
Maskinerne – Machines
Databarene – Databar
Spil – Game
Personer – Players
Raflebæger – Cup
Terninger – Dice
Resultat – Result
Sum – Sum
Værdi – Value
Point – Points
Vinder – Win
Kast - Roll

5.1.2 Udsagnsord

Kaste – roll
Lægge (sammen) – add
Opnår – achieve
Se/få - get

5.1.3 Ektraopgaver

5.1.3.1 Navneord

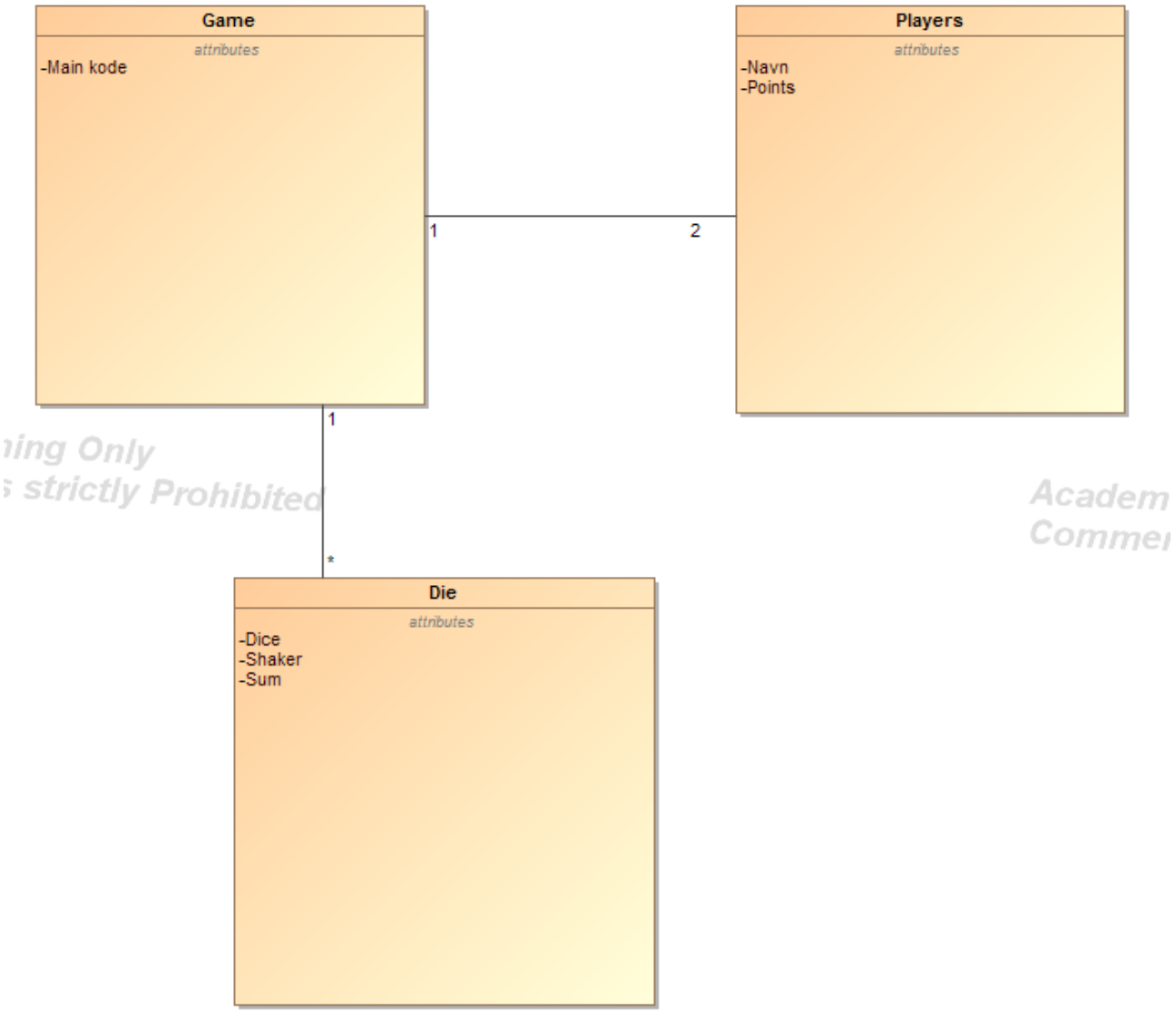
Ekstratur – ExtraTurn
Ektrakast - ExtraRoll

5.1.3.2 Udsagnsord

Miste - lose

5.2 Analyse

5.2.1 Domænemodel



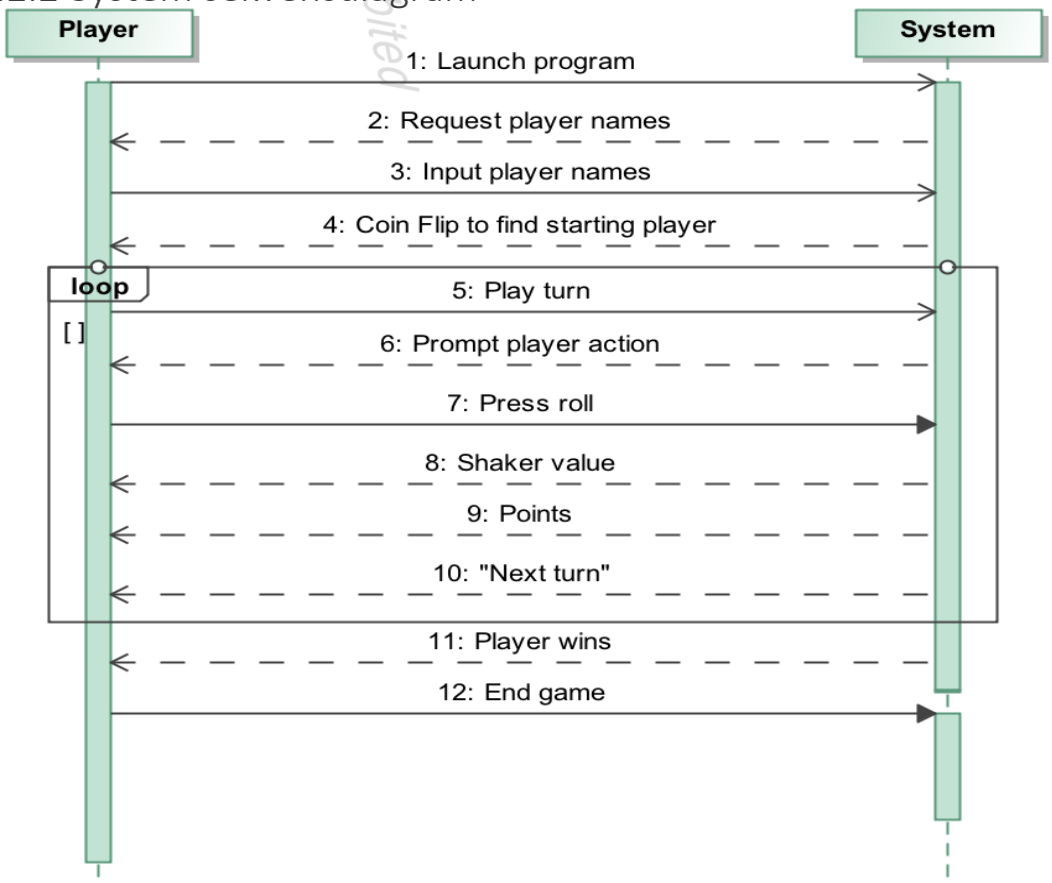
Dette diagram viser vores tanker til det program vi skal lave.

Player: Denne klasse får ansvaret for Navn på spilleren og deres nuværende point. Dette er så vi ikke skal gemme dem i vores Main kode, og vi kan forhåbentligt derfor korte denne ned, og dermed delegere ansvaret ordentligt.

Die: Denne klasse får ansvaret for selve funktionen at rulle en terning. Derudover at agere raflebæger og levere et resultat når terningerne bliver rystet. Den har altså ansvaret for terningerne og raflebægeret, og leverer så outputtet tilbage til vores main, som er Game.

Game: Denne klasse får ansvaret for at være vores main kode. Den skal derfor indeholde størstedelen af koden, og har ansvaret for at spillet kører rundt. Vi regner med denne bliver meget lang og at den får meget ansvar at holde styr på.

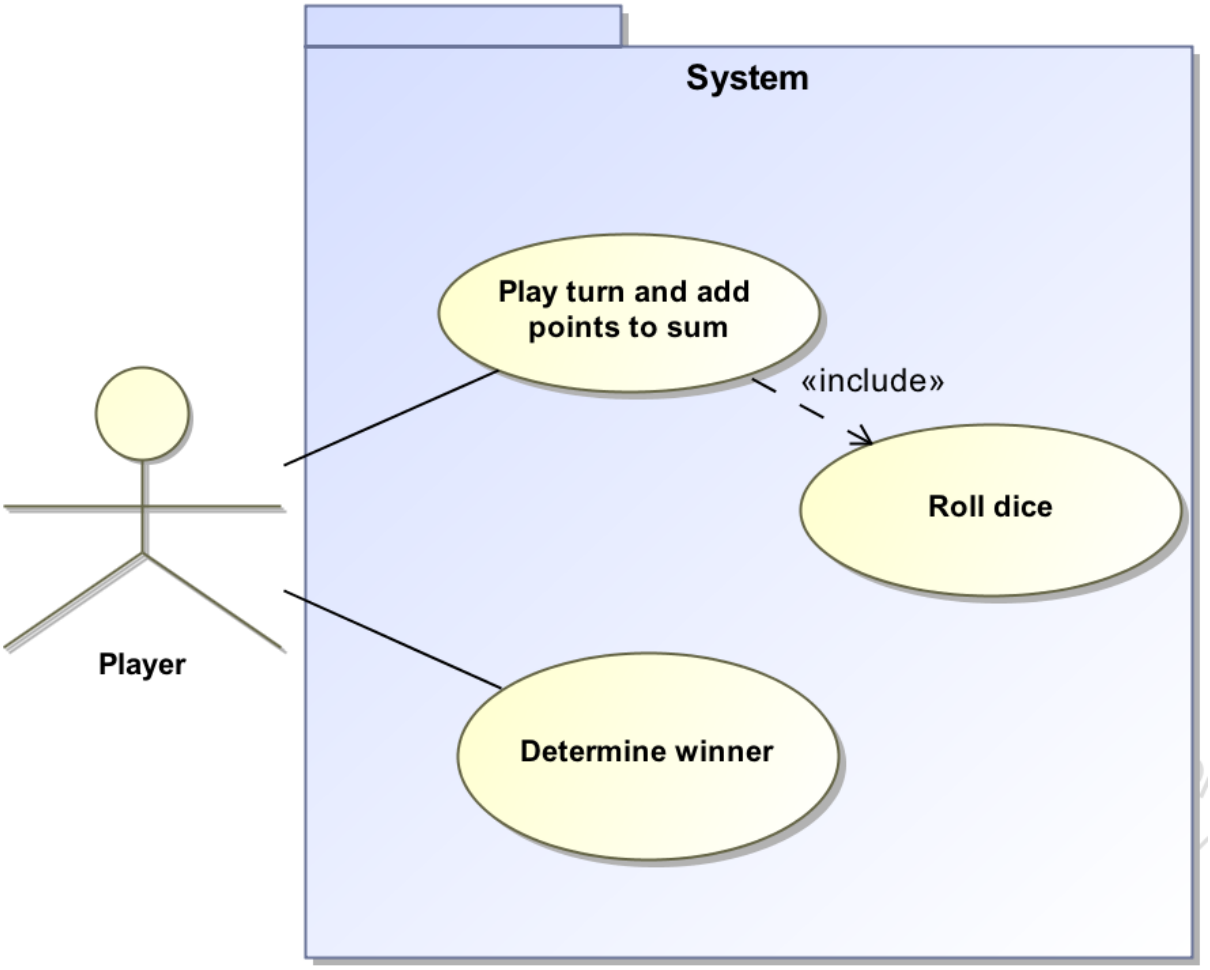
5.2.2 System sekvensdiagram



Dette diagram viser aktørens synsvinkel på vores program. Til venstre ses aktøren, Player, og til højre ses systemet. Da tiden forløber ud af y-aksen i nedadgående retning, kan man se, at programmet startes ved at aktøren starter programmer, hvorefter der foregår en udveksling af spillernavne. Herefter vælger programmet en tilfældig spiller, der skal starte for derefter at fortsætte i et loop. I dette loop er den første handling, at aktøren starter sin tur. Hertil vil programmet vente på, at aktøren kaster terningerne. Når dette er gjort returnerer programmet værdien af summen på de to terningers øjne, lægge dem til aktørens point og bagefter skifte til den næste aktør (spiller). Dette loop vil fortsætte indtil en aktør når en win-condition, hvilket kan opnås på flere forskellige måder jf. spillereglerne. Efter en vinder er fundet, vil aktøren slutte spillet ved at lukke programmet ned.

5.2.3 Use case

5.2.3.1 Use case diagram



Vores Use case diagram viser, i den simpleste form, sammenspillet mellem en aktør og vores program. Ud fra diagrammet ses det, at aktøren - der er en primær aktør, da der denne får sine brugermål opfyldt ved at interagere med systemet - interagerer med systemet. Først ved at starte sin tur, hvor programmet herefter kaster med terningerne, for derefter at vende tilbage til aktøren med det antal point denne har optjent. Dette vil fortsætte til en spiller - aktør - har vundet, hvorefter den næste interaktion mellem aktør og system sker ved, at systemet vælger en vinder.

5.2.3.2 Use cases

Use case: PlayersInGame	
ID:	K01 & K08
Brief Description	Skal kunne håndtere præcis 2 brugere
Primary actors	Brugere
Secondary actors	-
Preconditions	Starter ved computer, klar til at åbne programmet sammen med sin modspiller.
Main flow	1. Bruger starter programmet 2. Bruger får vist skærm hvor der står spillet er til 2 personer, samt de resterende regler. 3. Bruger 1 bliver bedt om at indtaste sit navn og trykke enter. 4. Bruger 2 bliver bedt om at indtaste sit navn og trykke enter. 5. Den bruger der vinder coin-toss bliver bedt om at starte spillet. 6. Bruger 1 trykker på enter og får vist en værdi, og afslutter sin tur 7. Programmet viser nu at det er spiller 2 tur. 8. Bruger 2 trykker på enter og får vist en værdi, og afslutter sin tur.
Postconditions	Inde i spillet og har set at der er 2 spillere.
Alternative flows	

Use case: RollDice	
ID:	K02
Brief Description	Programmet skal kunne slå med med to terninger.
Primary actors	Brugeren
Secondary actors	Ingen
Preconditions	Programmet er startet og spillet er i gang.
Main flow	1. Brugeren trykker på “Enter”. 2. Programmet udskriver to værdier, en for hver terning. Dette gøres ved brug af getDie1 og getDie2 metoden. 3. Programmet skal herefter være klar til endnu et kast.
Postconditions	Programmet har kastet to terninger, og vist resultatet.
Alternative flows	

Use case: getShake	
ID:	K03
Brief Description	Skal kunne vise summen fra seneste terningskast
Primary actors	Brugeren
Secondary actors	Ingen
Preconditions	At der er blevet kastet med terningerne
Main flow	1. Resultatet af terningerne bliver vist ved hjælp af getDie1 og getDie2 metoden. 2. Summen af terningerne bliver lagt sammen og vist via metoden getShake.
Postconditions	At summen af terningerne er vist
Alternative flows	

Use case: OpenGame	
ID:	K04
Brief Description	Skal kunne benyttes på Windows styresystem på DTU databars
Primary actors	Brugeren
Secondary actors	-
Preconditions	Brugeren har adgang til spillet. (jar. og .bat fil) og der er installeret java på computeren
Main flow	1. Brugeren åbner spillet ved at klikke på .bat filen 2. Brugeren kan nu spille spillet
Postconditions	Spillet er startet og klar til brug
Alternative flows	

Use case: setPlayerScore	
ID:	K05
Brief Description	Skal kunne tælle point op til 40 points
Primary actors	Bruger 1 og Bruger 2
Secondary actors	-
Preconditions	Programmet er startet og spillet er i gang.
Main flow	1. Bruger 1 kaster terningerne. 2. Summen af terningerne bliver lagt sammen 3. Summen af terningerne bliver lagt til “bruger 1” points, ved hjælp af metoden setPlayerScore 4. Punkt **1-3** gentages for “bruger 2” 5. Dette fortsætter op til en af brugerne har 40 points (Se K 07)
Postconditions	
Alternative flows	

Use case: getPlayerScore	
ID:	K06
Brief Description	Brugerens totale sum skal opdateres efter hvert kast.
Primary actors	Brugeren
Secondary actors	-
Preconditions	Brugeren har slået med to terninger
Main flow	1. Brugeren har ved sin tur fået en sum (fra K 03) 2. Turens sum lægges til brugerens samlede sum ved hjælp af setPlayerScore 3. Brugerens sum er nu opdateret og findes ved getPlayerScore
Postconditions	Brugeren har opnået en ny, forhøjet sum
Alternative flows	-

Use case: ruleForty	
ID:	K07
Brief Description	Brugeren der opnår 40 point skal vinde
Primary actors	Brugeren
Secondary actors	-
Preconditions	Brugeren har slået med to terninger
Main flow	1. Resultatet af brugerens totale sum (K 06) er 40 2. Bruger skal nu slå 2 ens for at vinde spillet. 3. Brugeren får besked om at have vundet 4. Spillet slutter
Postconditions	En vinder er fundet og spillet sluttes
Alternative flows	-

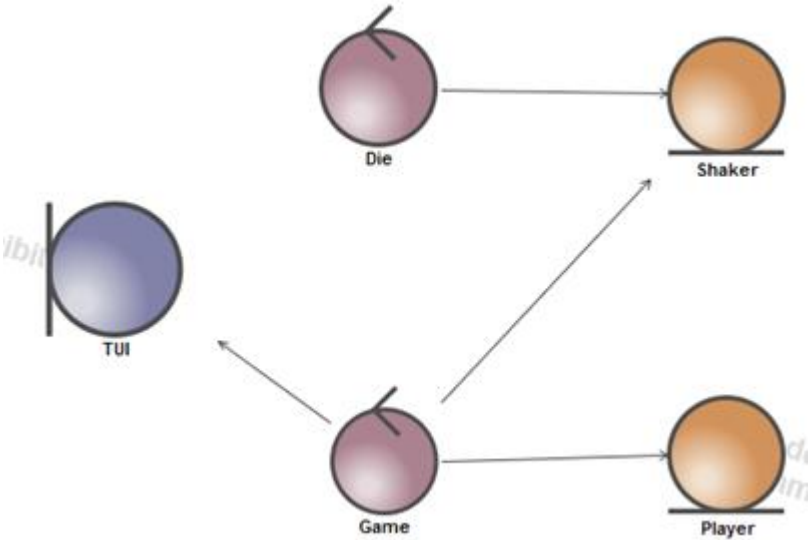
Use case: EasyUnderstanding	
ID:	K08
Brief Description	Brugeren skal kunne spille spillet uden brugsanvisning
Primary actors	Brugeren
Secondary actors	-
Preconditions	
Main flow	1. Brugeren starter spillet 2. Brugeren får vist et regelsæt og vejledning om hvordan man spiller spillet
Postconditions	
Alternative flows	

Fully Dressed	
ID:	K01 - K09.
Brief Description	Gennemgang af koden fra start til slut.
Primary actors	Brugerne.
Preconditions	Brugerne har en computer hvor systemet er tilgængeligt, og har åbnet det.
Main flow	1. Programmet startes. 2. Brugerne bedes indtaste navn. 2.1 Bruger et indtaster sit navn, og afslutter med enter. 2.2 Bruger to indtaster sit navn, og afslutter med enter. 3. Det bliver afgjort hvilken bruger som starter, hvor hver bruger har lige stor chance for at begynde ved et coin-toss. 3.1 Bruger et starter. 3.2 Bruger to starter. 4. Den bruger som starter, slår med raflebægeret 4.1 Brugeren opnår to forskellige værdier med terningerne, som lægges til brugerens samlede score. 4.2 Brugeren slår to ens og får en ekstra tur 4.2.1 Brugeren slår to 1'ere, hvormed brugerens samlede score nulstilles. Brugeren slår igen. 4.2.2 Brugeren slår to 2'ere, 3'ere, 4'ere eller 5'ere, vis samlede mængde point lægges til brugerens samlede score. Brugeren slår igen. 4.2.3 Brugeren slår to 6'ere, vis samlede mængde point lægges til brugerens samlede score og slår igen. 4.2.3.1 Slår brugeren to 6'ere igen, vinder brugeren spillet. Spillet afsluttes. 4.2.3.2 Reglerne fra punkt 4.1 - 4.2.2 går ellers igen. 5. Brugeren er færdig med sin tur, hvorpå den anden bruger gennemgår punkt 4. Punkt 4 og 5 gentages til en bruger har opnået 40 point. 6. Brugeren har opnået 40 point og slår med raflebægeret. 6.1 Brugeren slår to ens 6.1.1 Brugeren slår to 1'ere og får nulstillet sin score. Dermed skal brugeren tilbage til punkt 4 igen. 6.1.2 Brugeren slår to ens, udover to 1'ere og vinder spillet. Spillet afsluttes. 6.1.3 Brugeren slår to forskellige værdier, hvis samlede værdi lægges til brugerens samlede score. 7. Brugeren er færdig med sin tur, hvormed turen skiftes 7.1 Brugeren har ikke opnået 40 point, hvormed punkt 4 og 5

	gennemgås. 7.2 Brugeren har opnået 40 point, hvormed punkt 6 og 7 gennemgås.
Postconditions	En vinder er fundet mellem brugerne.

5.3 Design

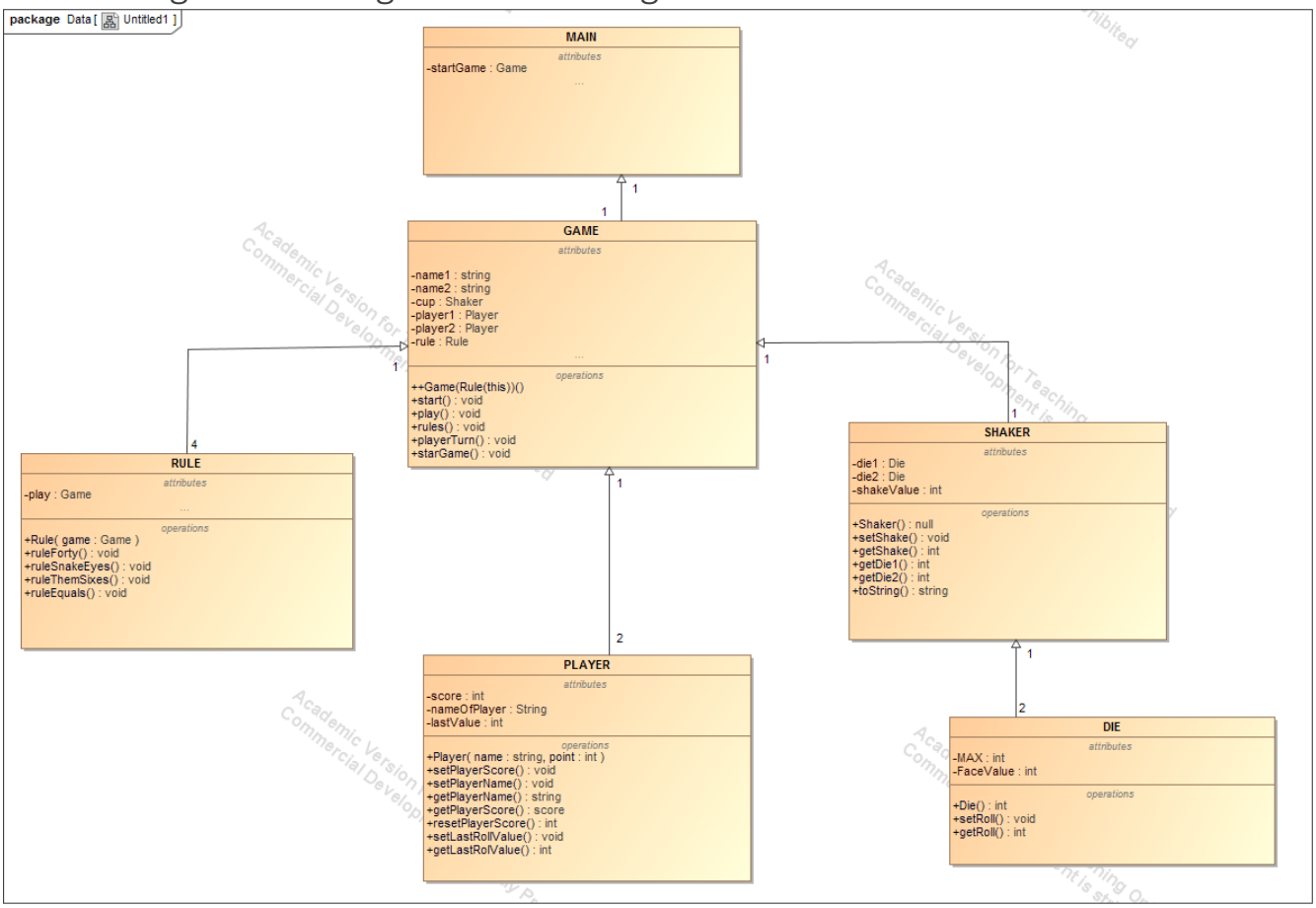
5.3.1 BCE model



Vores BCE model starter med to entities, som har til formål at modellere hvad systemet omhandler. Hver af disse klasser har hver deres persistens. Player-klassen har til formål at huske spillerens navn for at kunne adskille spillerne. Player-klassen reagerer kun med Game-klassen.

Game-klassen forbundet med TUI-klassen, der er en boundary. Altså en klasse, som sørger for at systemets omgivelser kan interageres med, værende spillerene i dette tilfælde. Game-klassen vil altså fortælle spilleren navn, for at adskille turene. Derudover hvad spillerene slår og ikke mindst hvad spillerens samlede score er. Game-klassen interagerer ikke med Die klassen, da de nødvendige informationer til TUI-klassen, opnås gennem Shaker-klassen. Hermed opnås også en højere kohæsion frem for lav kobling. Da den GUI-klasse vi fik tildelt på forhånd ikke havde de krav som vi fandt fyldestgørende, valgte vi kun at fokusere på en TUI-klasse, da vi ikke havde nok erfaring med at udvikle vores egen GUI-klasse.

5.3.2 Design-klassediagram med navngivne relationer



Vores klassediagram afspejler til dels vores domænemodel. Dog har vi her puttet en del flere klasser på for at få delt ansvaret ud.

Main: Har til ansvar at starte spillet. Der ligger ingen metoder eller andet i den, andet end at den kalder Game.

Game: Heri ligger hele strukturen i spillet, og den har til ansvar at kalde de andre klasser, og få deres output. Den styrer hvis tur det er, samt hvad der sker under hver enkelt tur. Derudover kalder den de andre klasser, når den f.eks skal have raflebægeret til at slå.

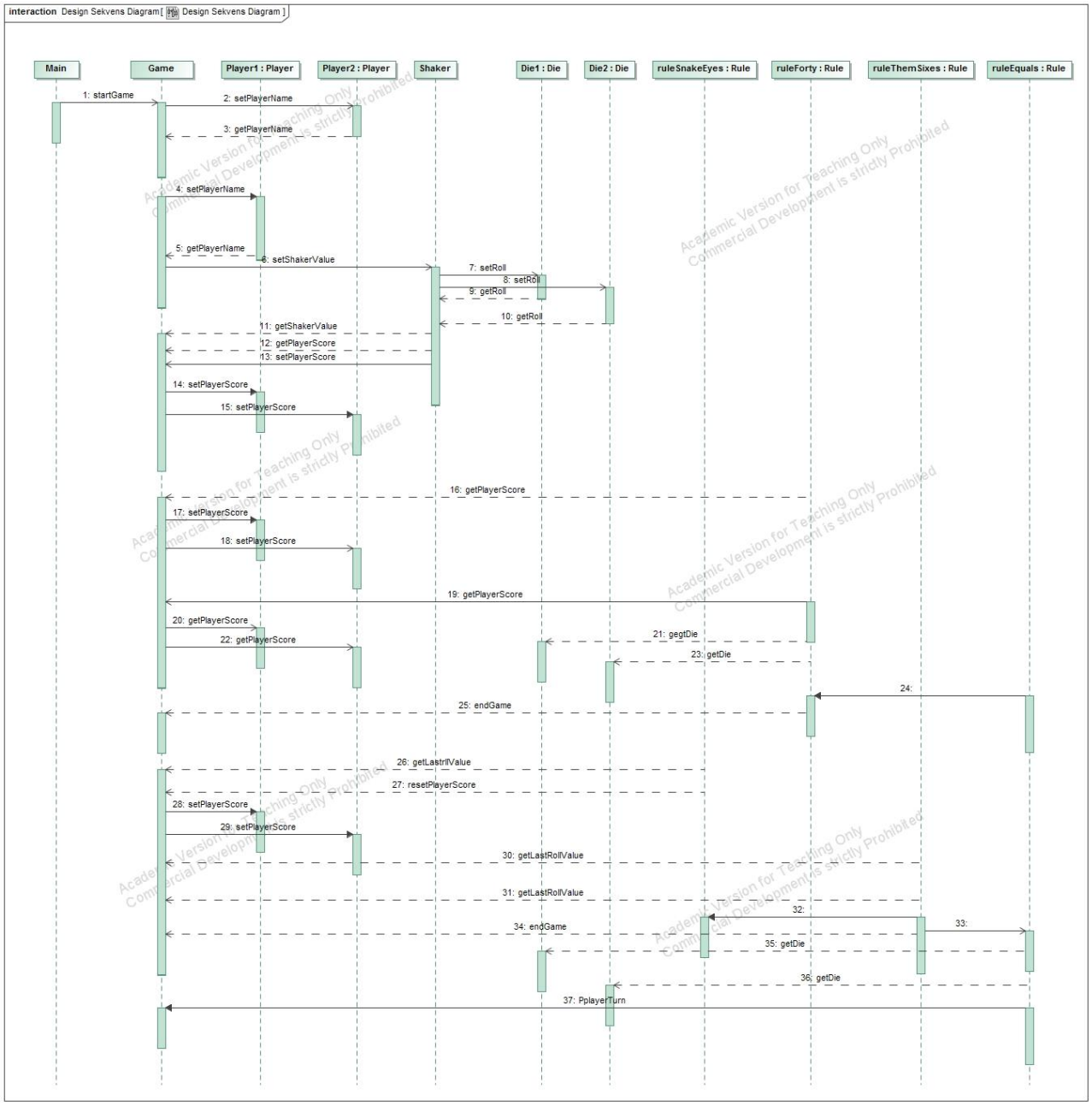
Player: Denne klasse har ansvaret for spillernavne, points og spillernes sidst slåede værdi.

Shaker: Denne klasse har ansvaret for at rulle terningerne inde i raflebægeret. Derudover skal klassen også holde styr på hvilken værdi de enkelte terninger har.

Die: Har ansvar for at rulle en enkelt terning og holde den værdi. Det er kun 1 enkelt terning der figurere i denne.

Rule: Denne klasse har ansvaret for at holde styr på de forskellige regler i spillet. De bliver således kaldt igennem Game. Dog blev vi nødt til at dele ansvaret fra Game klassen, over i Rule, for at kunne udføre nogle af vores funktioner. Dette gør vi da vi kalder nogle ting fra Game inde i Rule, hvor forudsætningen er at Rule kender til den instans af Game vi har kørende nu. Vi er opmærksomme på at dette ikke er best practice, og faktisk frarådes, og forstår grunden.

5.3.3 Design-sekvensdiagram



I Design-Sekvensdiagrammet ovenfor kan man se, hvordan objekterne I programmet interagerer med hinanden. Det hele starter med at Main beder Game om at starte et nyt spil. Game interagerer med Player og opretter derefter to spillere. Herefter tager Game fat i Shaker og Shaker tager fat i Die, Die1 og Die2, for at rulle terningerne. Die afleverer værdier for kastet tilbage til Shaker, som samlet giver en score til Game. Game sender summen for hver spiller til Player (hhv. Player1 og Player2). Nu tjekker Game i Rule for at se om nogle af reglerne er opfyldt. ruleForty ser først på den samlede score for en spiller, gennem Game. SnakeEyes tjekker efter hvert slag om det udkommet er par ét. I tilfælde af, at det er, nulstilles spillerens score. Hvis denne er 40 eller over tjekker den om det sidste kast er

et par som vil give en vinder. ThemSixes tjekker om der bliver slået par 6, og hvis der bliver slået par 6 igen. ruleEquals giver en ekstra tur.

5.4 Test

Kunden har stillet et krav om en test af terningerne. (K09) – Kravet lyder på 1000 kast, men for at give resultatet en større værdi, har vi valgt at teste på et langt højere antal slag. Dette giver os også muligheden for at sammenligne med de teoretiske sandsynligheder når vi arbejder med et slag hvor to terninger indgår.

Klassen Shaker vil være vores hovedfokus i unit testen. Denne klasse indeholder to instanser af klassen Die.

5.4.1 Hyppighedstest

Første test bliver udført i ShakerTestFrequency.java (Kan findes i bilag 8.1.7)

```
Frequency and result of 330000 shakes
Total of Two: 9300 - The frequency is 2.8181818% - Should be around 2.78%
Total of Three: 18143 - The frequency is 5.497879% - Should be around 5.56%
Total of Four: 27562 - The frequency is 8.352121% - Should be around 8.33%
Total of Five: 36595 - The frequency is 11.089395% - Should be around 11.11%
Total of Six: 45881 - The frequency is 13.903334% - Should be around 13.89%
Total of Seven: 54755 - The frequency is 16.592424% - Should be around 16.67%
Total of Eight: 45892 - The frequency is 13.906667% - Should be around 13.89%
Total of Nine: 36693 - The frequency is 11.119091% - Should be around 11.11%
Total of Ten: 27518 - The frequency is 8.338788% - Should be around 8.33%
Total of Eleven: 18424 - The frequency is 5.58303% - Should be around 5.56%
Total of Twelve: 9237 - The frequency is 2.7990909% - Should be around 2.78%
```

Som det fremgår, har vi simuleret 330.000 ”shakes”. Det er også muligt at se antallet af forekomster af hvert enkelt slag, både i antal og i procent. Dette giver os mulighed for at sammenligne med ”Should be around” som er den teoretiske sandsynlighed. Vores resultat er yderst tilfredsstillende da vores procent er næsten identiske med det teoretiske mulige.

5.4.2 Forekomster af par

Anden test bliver udført i ShakerPairs.java (Kan findes i bilag 8.1.8)

```
Distribution and result
Double One: 19931
Double Two: 19881
Double Three: 19968
Double Four: 20022
Double Five: 20134
Double Six: 20064
Total pairs: 120000
Total count of shakes: 718648
The frequency is 16.698023% - Should be around 16.7%
```

Her tester vi hvor mange shakes der skal til før vi har 120.000 par. I dette tilfælde har vi brugt 718.648 forsøg på at opnå netop dette. Til sidst beregner vi om dette stemmer overens med den teoretiske sandsynlighed. Vores resultat er yderst tilfredsstillende da vores procent er næsten identiske med det teoretiske mulige.

Efter disse to test kan vi bekræfte at de to objekter af terningerne vi benytter i vores shaker er yderst pålidelige og lever op til krav K09 i vores kravspecifikation.

5.4.3 Test cases

Test case ID	**TC01**
Summary	Tester om terningerne ikke er vægtet, ved at kigge på procentdelen af fremkomster af forskellige summe.
Requirements	K09
Preconditions	Instansen ShakerTestFrequency kører.
Postconditions	Instansen er stoppet.
Test Procedure	1. Test programmet startes. 2. Data kommer ud.
Test Data	Test data på 110.000 slag.
Expected Result	Bell kurve der har toppunkt i 7.

Actual Result	`Two: 3014`
	`Three: 6026`
	`Four: 9260`
	`Five: 12081`
	`Six: 15226`
	`Seven: 18347`
	`Eight: 15376`
	`Nine: 12200`
	`Ten: 9214`
	`Eleven: 6187`
	`Twelve: 3069`

Status	Passed
Tested by	Aleksander og Martin
Date	07-10-2016
Test Environment	Eclipse 4.6.0 on Windows

Test case ID	**TC02**
Summary	Tester hyppigheden af de forskellige par, på 120.000 par slag.
Requirements	K09
Preconditions	Instansen ShakerTestPairs kører.
Postconditions	Instansen er stoppet.
Test Procedure	1. Test programmet startes. 2. Data kommer ud.
Test Data	Test data på 120.000 par slag.
Expected Result	16,7% af alle slag er par slag.

Actual Result	`Distribution and result`
	`Double One: 19878`
	`Double Two: 19978`
	`Double Three: 20000`
	`Double Four: 19885`
	`Double Five: 20072`
	`Double Six: 20187`
	`Total pairs: 120000`
	`Total count of shakes: 721304`

	`The frequency is 16.636536% - Should be around 16.7%`
Status	Passed
Tested by	Aleksander og Martin
Date	07-10-2016
Test Environment	Eclipse 4.6.0 on Windows

Test case ID	**TC03**
Summary	Skal kunne håndtere præcis 2 brugere samt være brugervenligt
Requirements	K01, K04 & K08
Preconditions	Computeren er tændt og programmet ligger på denne.
Postconditions	Programmet er åbent og kører stadig.
Test Procedure	1. Start programmet. 2. Læs brugervejledning, der kommer frem. 3. Indtast navn på først spiller 1, derefter spiller 2. 4. Spillet startes.
Test Data	Test data om der er to spillere.
Expected Result	Programmet kan understøtte to spillere og er brugervenligt.
Actual Result	Programmet kunne understøtte to spillere og var brugervenligt.
Status	Passed
Tested by	Aleksander og Martin
Date	7-10-2016
Test Environment	Eclipse 4.6.0 on Windows

Test case ID	**TC04**
Summary	Programmet skal kunne slå med to terninger og vise resultatet heraf
Requirements	K02, K03 & K06
Preconditions	Programmet er startet og en spiller er valgt til at spille sin tur.
Postconditions	Programmet kører stadig og kan spilles videre.
Test Procedure	1. Tryk på enter for at slå med terningerne. 2. Programmet viser summen af terningernes øjne. 3. Programmet lægger denne sum til spillerens point.
Test Data	Se om programmet kan tælle øjnene på terningen og lægge dem til spillerens point.
Expected Result	Programmet kaster med terningerne og lægger summen til point.
Actual Result	Programmet kastede med terningerne og lagde summen til point.
Status	Passed
Tested by	Aleksander & Martin
Date	7-10-2016
Test Environment	Eclipse 4.6.0 on Windows

Test case ID	**TC05**
Summary	Skal kunne tælle op til 40 point og derefter, hvis spilleren slår 2 ens, afgøre en vinder.
Requirements	K05 & K07
Preconditions	Spillet er startet og kører.
Postconditions	Spillet er slut.
Test Procedure	1. En spiller har nået 40 point. 2. Spilleren skal slå 2 ens for at vinde. 1. Spilleren slår 2 ens og vinder spillet. 2. Spilleren slår ikke to ens og turen går videre.
Test Data	Se om spillet kan bestemme en vinder.
Expected Result	Spillet vælger vinderen ud fra hvem der først når 40 point og slår to ens.
Actual Result	Spillet valgte en vinder ud fra den person, der først nåede 40 point og slog to ens.
Status	Passed
Tested by	Aleksander & Martin
Date	07-10-2016
Test Environment	Eclipse 4.6.0 on Windows

5.4.4 Traceability matrix

Test Case	**Requirement number**								
	K01	K02	K03	K04	K05	K06	K07	K08	K09
TC01									x
TC02		x	x			x			x
TC03	x			x				x	
TC04		x	x			x			
TC05					x		x		

5.5 Versionsstyring

Vi har valgt at bruge GitHub som vores distribueret versionsstyringsværktøj. Dette har givet alle gruppens medlemmer mulighed for at kunne bidrage med kode.

GitHub indeholder også en lang række funktioner der er nødvendig i et udviklingsteam. Dette indebærer fx ”commits” som giver os muligheden for at udføre en ”roll back” til en tidligere version hvis dette skulle vise sig at blive nødvendigt. Derudover har det også givet os den uundværlige funktion at vi har kunne gennemgå hinandens ændringer undervejs og på den måde sikre en højere kvalitet af vores færdige produkt.

For en mere detaljeret oversigt over forløbet og løbende kodetilføjelser i forløbet henvises til vores GitHub.

GitHub link: https://github.com/s165150/CDIODel1

Liste over brugernavne på GitHub:

Brugernavn: s165150 - Morten Enghausen Nielsen (s165150)
Brugernavn: Kemikaze - Aleksander Wienziers-Madsen (s114750)
Brugernavn: DonPhister - Nick Skouenborg (s165233)
Brugernavn: mortentang - Morten René Tang Christiansen (s162682)
Brugernavn: MartinDahl96 - Jens Martin Dahl-Jensen (s165159)

6 Konklusion

Efter vi har afsluttet vores projekt, kan vi nu aflevere et færdigt produkt til kunden med alle kravene og endda også alle deres ekstra tilføjelser i form af særregler for spillet, samt kravet om at der skal være en hurtig responstid på programmet.

I starten af vores proces da vi skulle undfange ideen var planen at gøre det efter UP-principperne. Dog fandt vi hurtigt ud af, at alle gruppens medlemmer meget gerne ville kode, for at få en dybere forståelse for stoffet. Dette resulterede i, at efter vores første indledende møde, hvor vi fik tegnet nogle skitser for projektet, valgte at dele os op og sidde med koden individuelt. I dette forløb blev vi inspireret af hinandens kode, for til sidst at blive færdig med hele opgaven, dog kørte hele spillet i en enkelt klasse. For at opnå et mere tilfredsstillende resultat, valgte vi at give det et ekstra forsøg med at få det opdelt i 6 klasser, som interagerer med hinanden. Efter mange timer fik vi dette til at virke, og er derfor endt med denne opbygning. Vores idé var at starte med at lave basis-koden, altså selve spillet, for herefter at smide de "ekstra" regler ind i spillet. Dette havde dog en snert af vandfaldsmodel over sig, da vi på denne måde egentlig havde planlagt hele forløbet og derved satte os fast på kun at lave det på én enkelt måde. Vi besluttede os dog for at følge en iterativ og agil udviklingsmetode, da vi kunne se det ikke fungerede på den anden måde. Dette gjorde vi ved både at designe og implementere i programmet på én gang.

På grund af denne proces, har været nødt til at omskrive de tidligere diagrammer og use cases vi havde lavet, til at passe til det nye klasseopdelte program.

Til opgaven blev der stillet en GUI til rådighed. Vi forsøgte at implementeret denne, og efter lidt tid lykkedes det også. Dog var vi ikke tilfredse med resultatet. Dette skyldes, at GUI'en er til et matadorspil, og vi til vores spil faktisk kun bruger terningerne. Det efterlader en masse farverige felter i udkanten af GUI'en, som slet ikke bliver brugt. Vi besluttede os derfor for, bare at benytte os af en TUI i stedet, da denne er meget mere passende til vores program.

Det vi kan gøre bedre næste gang er at benytte os af iterativ og agil planlægning fra start af. Dette gør vi i håb om at kunne få det samlede antal timer til at falde, og derved ikke bruge lang tid på at rende rundt i ring.

7 Litteratur- og kildefortegnelse

7.1 Bøger

Larman, Craig. Applying UML and Patterns. 2004, 3. udgave. Addison Wesley Professional. 0-13-148906-2. 736 sider.

Lewis, John & Loftus, William. Java™ Software Solutions. 7. udgave. Addison Wesley. 0-13-214918-4. 806 sider.

7.2 Links

Bechmann, Henrik & Tange, Henrik. 02313 Udviklingsmetoder til IT-systemer: Lektion 5: Analyse, UML notation. https://www.campusnet.dtu.dk/cnnet/filesharing/SADownload.aspx?ElementId=522003&FileVersionId=5516667&FileId=4291632&mode=download_text Læst d. 03/10-2016. Hentet 26/09-2016

Sandsynligheder, terning.
http://alumnus.caltech.edu/~leif/FRP/probability.html Læst d. 06/10-2016

8 Bilag

8.1 Bilag 1 – Source code

8.1.1 Main.java

```
/**
 * @author
 * Aleksander wrote the code, and in doing so got help with the different parts, from the team. They are listed
under each class.
 * Proofreading done by Martin
 * Compressment of the code done by Morten N
 */
public class Main {

		public static void main(String[] args)
		{
			//Create an new instance of the Game class
			Game startGame = new Game();

			//Start game from the method startGame
			startGame.startGame();
		}
}
```

8.1.2 Game.java

```
/**
 * @author Aleksander and Morten N
 */
import java.util.Scanner;

public class Game {

	//Creation of attributes and new instance of the class Scanner, Player, Shaker and Rule
	private String namePlayer1 = "";
	private String namePlayer2 = "";
	private int turnCount = (int) (Math.random()*2);
	private Scanner scan = new Scanner(System.in);
	private Shaker cup = new Shaker ();
	private Player player1 = new Player (namePlayer1, 0);
	private Player player2 = new Player (namePlayer2, 0);
	private Rule rule;

		/**
		 * This make a new instance of the class Rule, and makes the current Game instance, follow,
for the rule class to access.
		 * This is NOT the optimal solution in our software! We need to work on this in another
version.
		 */
		public Game()
		{
			rule = new Rule(this);
		}

		/**
		 * Method we use to enter the name of the players and print the start score
		 */
		public void start()
		{
			System.out.println("Welcome to the best dice game in the world!");
			System.out.println("The game must be played by two players.");
			System.out.println("The person who gets to start is decided by a coin flip.");
			System.out.println("The game has four rules:");
			System.out.println("1) The winner will be the player who first reaches 40 points and
after that roll a pair");
```

```
                System.out.println("2) You will loose ALL of your points if you roll snake eyes (Pair
of ones).");
                System.out.println("3) You will get an extra turn if you hit any pair of the same
kind.");
                System.out.println("4) If you hit a pair of sixes two times in a row you win the
game!");
                System.out.println("Press ENTER to start the game");
                scan.nextLine();
                System.out.println("Enter name of Player 1:");
                player1.setPlayerName(scan.next());
                System.out.println("Enter name of Player 2:");
                player2.setPlayerName(scan.next());
                scan.nextLine();
                System.out.println("Press ENTER to take the coin flip!");
                scan.nextLine();
                if (turnCount == 0)
                {
                        System.out.println(player1.getPlayerName() + " won the coin flip!");
                }
                else
                        System.out.println(player2.getPlayerName() + " won the coin flip!");
        }

        /**
         * Method to play the game - This shake the cup, printout the score and make sure our rules
is complied
         * @param player Player1 or Player2
         * @param cup Cup we use to roll the dice
         * @param rule We use rule to make sure our rules is complied
         */
        public void play(Player player, Shaker cup, Rule rule)
        {
                playerTurn(player, cup);
                rules(player, cup, rule);
        }

        /**
         * Method for our rules
         * @param player Player1 or Player2
         * @param cup Cup we use to roll the dice
         * @param rule We use rule to make sure our rules is complied
         */
```

```
	public void rules(Player player, Shaker cup, Rule rule)
	{
		rule.ruleSnakeEyes(player);
		rule.ruleForty(player, cup);
		rule.ruleThemSixes(player, cup);
		rule.ruleEquals(player, cup);
	}

	/**
	 * Method we use every new turn - This shakes the cup, roll the dice, printout the score and
update the total score.
	 * @param player Player1 or Player2
	 * @param cup Cup we use to roll the dice
	 */
	public void playerTurn(Player player, Shaker cup)
	{
		System.out.println();
		System.out.println(player.getPlayerName() + " please press ENTER to roll the dice");
		scan.nextLine();
		cup.setShake();
		player.setPlayerScore(cup.getShake());
		player.setLastRollValue(cup.getShake());
		System.out.println(player.getPlayerName() + " you got:");
		System.out.println("Die One: " + cup.getDie1() + ", Die Two: " + cup.getDie2());
		System.out.println("Total score: " + player.getPlayerScore());
	}

	/**
	 * Method to start the game - The turnCount makes a "coin flip" and decides who will start
the game
	 */
	public void startGame()
	{
	start();

		while(turnCount == 0)
		{
			play(player1, cup, rule);
			play(player2, cup, rule);
		}

		while(turnCount == 1)
```

```
                {
                        play(player2, cup, rule);
                        play(player1, cup, rule);
                }
        }
}
```

8.1.3 Die.java

```
/**
 * This is the creation of our blueprint (class) Die.
 * Inside this class we've created a method called roll
 * If we want to call this method into another class we'll simply type die.setRoll() or die.getRoll.
 * This means that we're calling the class Die and then the method getRoll or setRoll.
 * This will in turn return the values seen below. What this means is that we now have a class
 * that will simulate a diceroll.
 * @author Aleksander and Martin
 */
public class Die {

	//Creation of attributes
	private final int MAX = 6;
	private int faceValue;

	/**
	 * Creates the constructor, Die.
	 */
	public Die()
	{
		faceValue = 1;
	}

	/**
	 * This method rolls the die for us.
	 */
	public void setRoll()
	{
		faceValue = (int) ((Math.random() * MAX) + 1);
	}

	/**
	 * This returns the die's value.
	 * @return
	 */
	public int getRoll()
	{
		return faceValue;
	}
}
```

8.1.4 Rule.java

```
/**
 * @author Aleksander and Morten N
 */
public class Rule {
	Game play;

	public Rule(Game game) {
		play = game;

	}

	/**
	 * Creates the rule that states that if a player gets more than 40 points and then hits 2 equals, he/she
wins.
	 * @param player Player1 or Player2
	 * @param cup Cup we use to roll the dice
	 */
	public void ruleForty(Player player, Shaker cup)
	{
		if (player.getPlayerScore()>40)
		{
			if(cup.getDie1() == cup.getDie2())
			{
			System.out.println(player.getPlayerName() + " Won!");
			System.exit(0);
			}
		}
	}

	/**
	 * Creates the rule for snake eyes. If the player hits double aces, his/her score gets reset.
	 * @param player Player1 or Player2
	 * @param cup Cup we use to roll the dice
	 */
	public void ruleSnakeEyes(Player player)
	{
		if (player.getLastRollValue() == 2)
		{
			System.out.println("Your score has been reset. Stay away from snakes!");
			player.resetPlayerScore();
```

```
		}
	}

	/**
	 * Sets the rule that when the player hits 2 6's he/she gets a chance to win the game. The extra WIN turn
is not counted into his/her points.
	 * @param player Player1 or Player2
	 * @param cup Cup we use to roll the dice
	 */
	public void ruleThemSixes(Player player, Shaker cup)
	{
		if (player.getLastRollValue() == 12)
		{
			System.out.println("You just hit them double 6's. Try again and win the game");
			play.playerTurn(player, cup);
			ruleSnakeEyes(player);
			ruleEquals(player, cup);

			if (player.getLastRollValue() == 12)
			{
				//Wins the game on two sixes in a row
				System.out.println(player.getPlayerName() + " Jackpot! You got two sixes in a row!");
				System.exit(0);
			}
			System.out.println("Better luck next time");

		}
	}

	/**
	 * Creates the rule that when the player hits 2 equal eyes, he/she gets another turn.
	 * @param player Player1 or Player2
	 * @param cup Cup we use to roll the dice
	 */
	public void ruleEquals(Player player, Shaker cup)
	{
		while (cup.getDie1() == cup.getDie2())
		{
			System.out.println("EXTRA TURN");
			play.playerTurn(player, cup);
			ruleSnakeEyes(player);
			ruleForty(player, cup);
```

```
                    ruleThemSixes(player, cup);
                }
            }
        }
```

8.1.5 Player.java

```
/**
 * Player class is created to keep track of the player's name, and their current score in the game.
 * The way this is done is explained below.
 * @author Aleksander and Morten N
 */
public class Player {

	//Creation of attributes
	private int score = 0;
	private String nameOfPlayer;
	private int lastValue = 0;

	/**
	 * Creates the constructor
	 * @param name Reserves space for the player's name.
	 * @param point Reserves space for the player's points.
	 */
	public Player (String name, int point)
	{
		nameOfPlayer = name;
		score = point;
	}

	/**
	 * Calculates the player's score. This is done by having the (int currentShakerValue) added to the score.
	 * @param currentShakerValue Adds the current shaker points to the total score.
	 * @return
	 */
	public int setPlayerScore(int currentShakerValue)
	{
		score = score + currentShakerValue;
		return score;
	}

	/**
	 * Sets the player's name
	 * @param name Sets the players name
	 * @return
	 */
	public void setPlayerName(String name)
```

```
	{
		nameOfPlayer = name;
	}

	/**
	 * Returns the player's name.
	 * @return
	 */
	public String getPlayerName()
	{
		return nameOfPlayer;
	}

	/**
	 * Returns the player's current score.
	 * @return
	 */
	public int getPlayerScore()
	{
		return score;
	}

	/**
	 * Resets a player's score to 0.
	 * @return
	 */
	public int resetPlayerScore()
	{
		score = 0;
		return score;
	}

	/**
	 * Stores the last roll a player has made. This is done in order to see if he/she scores 2 6's twice in a
row.
	 * @param currentShakerValue Puts currentsShakerValue to lastValue
	 */
	public void setLastRollValue(int currentShakerValue)
	{
		lastValue = currentShakerValue;
	}
```

```
	/**
	 * Returns the last roll the player made.
	 * @return
	 */
	public int getLastRollValue()
	{
		return lastValue;
	}
}
```

8.1.6 Shaker.java

```
/**
 * @author Aleksander and Morten N
 */
public class Shaker {

	//Creation of attributes and new instance
	private int shakeValue;
	Die die1 = new Die();
	Die die2 = new Die();

	/**
	 * Creates the constructor, Shaker.
	 */
	public Shaker()
	{
		shakeValue = 0;
	}

	/**
	 * Rolls the dice, but doesn't return a value.
	 */
	public void setShake()
	{
		die1.setRoll();
		die2.setRoll();
	}

	/**
	 * Returns the value of the roll.
	 * @return
	 */
	public int getShake()
	{
		shakeValue = die1.getRoll() + die2.getRoll();
		return shakeValue;
	}

	/**
	 * Returns the value of Die1 from the roll.
	 * @return
```

```
     */
    public int getDie1()
    {
        return die1.getRoll();
    }

    /**
     * Returns the value of Die2 from the roll.
     * @return
     */
    public int getDie2()
    {
        return die2.getRoll();
    }

    /**
     * Converts the shakeValue from an int to a string.
     */
    public String toString()
    {
        String result = Integer.toString(shakeValue);
        return result;

    }
}
```

8.1.7 ShakerTestFrequency.java

```
/**
 * @author Morten N
 */
import static org.junit.Assert.*;
import org.junit.Test;

public class ShakerTestFrequency
{

    @Test
    public void testShaker()
    {
        //Define attributes for test
        Shaker shaker = new Shaker();
        int two = 0, three = 0, four = 0, five = 0, six = 0, seven = 0, eight = 0, nine = 0, ten = 0,
eleven = 0, twelve = 0, shakeValue, controlCount = 0;
        float percentTwo = 0, percentThree = 0, percentFour = 0, percentFive = 0, percentSix = 0,
percentSeven = 0;
        float percentEight = 0, percentNine = 0, percentTen = 0, percentEleven = 0, percentTwelve = 0;

        //While loop for shaker test - Test for 330000 shakes
        while (controlCount < 330000)
        {
            //Shake the dice from the class shaker
            shaker.setShake();
            shakeValue = shaker.getShake();


            //Switch statement for control count and shake
            switch (shakeValue)
            {
            case 2: two++; controlCount++; break;
            case 3: three++; controlCount++; break;
            case 4: four++; controlCount++; break;
            case 5: five++; controlCount++; break;
            case 6: six++; controlCount++; break;
            case 7: seven++; controlCount++; break;
            case 8: eight++; controlCount++; break;
            case 9: nine++; controlCount++; break;
            case 10: ten++; controlCount++; break;
            case 11: eleven++; controlCount++; break;
```

```
            case 12: twelve++; controlCount++;
            }
        }

        //Calculate percent for every possible shake
        percentTwo = (float)two/controlCount*100;
        percentThree = (float)three/controlCount*100;
        percentFour = (float)four/controlCount*100;
        percentFive = (float)five/controlCount*100;
        percentSix = (float)six/controlCount*100;
        percentSeven = (float)seven/controlCount*100;
        percentEight = (float)eight/controlCount*100;
        percentNine = (float)nine/controlCount*100;
        percentTen = (float)ten/controlCount*100;
        percentEleven = (float)eleven/controlCount*100;
        percentTwelve = (float)twelve/controlCount*100;

        //Printout to confirm result
        System.out.println("Frequency and result of 330000 shakes");
        System.out.println("Total of Two: " + two + " - The frequency is " + percentTwo + "% - Should
be around 2.78%");
        System.out.println("Total of Three: " + three + " - The frequency is " + percentThree + "% -
Should be around 5.56%");
        System.out.println("Total of Four: " + four + " - The frequency is " + percentFour + "% -
Should be around 8.33%");
        System.out.println("Total of Five: " + five + " - The frequency is " + percentFive + "% -
Should be around 11.11%");
        System.out.println("Total of Six: " + six + " - The frequency is " + percentSix + "% - Should
be around 13.89%");
        System.out.println("Total of Seven: " + seven + " - The frequency is " + percentSeven + "% -
Should be around 16.67%");
        System.out.println("Total of Eight: " + eight + " - The frequency is " + percentEight + "% -
Should be around 13.89%");
        System.out.println("Total of Nine: " + nine + " - The frequency is " + percentNine + "% -
Should be around 11.11%");
        System.out.println("Total of Ten: " + ten + " - The frequency is " + percentTen + "% - Should
be around 8.33%");
        System.out.println("Total of Eleven: " + eleven + " - The frequency is " + percentEleven + "% -
Should be around 5.56%");
        System.out.println("Total of Twelve: " + twelve + " - The frequency is " + percentTwelve + "% -
Should be around 2.78%");

        //Test assertEquals on 330000 shakes
```

```
        assertEquals(9000,two,500);
        assertEquals(18000,three,600);
        assertEquals(27000,four,800);
        assertEquals(36000,five,1000);
        assertEquals(45000,six,1200);
        assertEquals(54000,seven,1400);
        assertEquals(45000,eight,1200);
        assertEquals(36000,nine,1000);
        assertEquals(27000,ten,800);
        assertEquals(18000,eleven,600);
        assertEquals(9000,twelve,500);
        assertEquals(330000,controlCount);
    }

}
```

8.1.8 ShakerTestPairs.java

```
/**
 * @author Morten N
 */
import static org.junit.Assert.*;
import org.junit.Test;

public class ShakerTestPairs
{

    @Test
    public void testShaker()
    {
        //Define attributes for test
        Shaker shaker = new Shaker();
        int pairOfOne = 0, pairOfTwo = 0, pairOfThree = 0, pairOfFour = 0, pairOfFive = 0, pairOfSix =
0, shakeValue, die1, die2, controlCount = 0, totalCountOfShakes = 0;
        float percent = 0;

        //While loop and for shaker test - Test for 120000 pairs
        while (controlCount < 120000)
        {
            //Shake and get the dice from the class shaker
            shaker.setShake();
            shakeValue = shaker.getShake();
            die1 = shaker.getDie1();
            die2 = shaker.getDie2();
            totalCountOfShakes++;

            //If statements count the resualt
            if (die1 == die2 && shakeValue == 2)
            {
                pairOfOne++; controlCount++;
            }
            if (die1 == die2 && shakeValue == 4)
            {
                pairOfTwo++; controlCount++;
            }
            if (die1 == die2 && shakeValue == 6)
            {
                pairOfThree++; controlCount++;
            }
```

```
            if (die1 == die2 && shakeValue == 8)
            {
                pairOfFour++; controlCount++;
            }
            if (die1 == die2 && shakeValue == 10)
            {
                pairOfFive++; controlCount++;
            }
            if (die1 == die2 && shakeValue == 12)
            {
                pairOfSix++; controlCount++;
            }
        }

        //Calculate percent
        percent = (float)controlCount/totalCountOfShakes*100;

        //Printout to confirm result
        System.out.println("Distribution and result”);
        System.out.println("Double One: " + pairOfOne);
        System.out.println("Double Two: " + pairOfTwo);
        System.out.println("Double Three: " + pairOfThree);
        System.out.println("Double Four: " + pairOfFour);
        System.out.println("Double Five: " + pairOfFive);
        System.out.println("Double Six: " + pairOfSix);
        System.out.println("Total pairs: " + controlCount);
        System.out.println("Total count of shakes: " + totalCountOfShakes);
        System.out.println("The frequency is " + percent + "% - Should be around 16.7%");

        //Test assertEquals on 110000 shakes
        assertEquals(20000,pairOfOne,300);
        assertEquals(20000,pairOfTwo,300);
        assertEquals(20000,pairOfThree,300);
        assertEquals(20000,pairOfFour,300);
        assertEquals(20000,pairOfFive,300);
        assertEquals(20000,pairOfSix,300);
        assertEquals(120000,controlCount);
    }

}
```