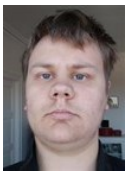
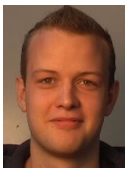
Rapport - CDIO1 Terningespil

Peter Revsbech (s183760)



Mathias Bærentzen (s176360)

Sulaiman Kasas (s195462)



Christian Kyed (s184210)

Bashar Bdewi (s183356)

Derar Amlosawe (s181553)

Link til GitHub-repo:
https://github.com/PeterRevsbech/CDIO-del-1-gruppe-11.git

Rapport i kurset Versionsstyring og Testmetoder 02315
Danmarks Tekniske Universitet
Afleveringsfrist: 04. Oktober 2019

Indhold

1 Abstract

This report describes the work done on the CDIO project part 1 by group 11, which is a software-project written in the Java language. The project is a simple dicegame played by two players. The report describes the requirements, that the group decided on after reading the project description, as well as an analysis of some central usecases of the final product. After the analysis section is a design section, describing the fundmental design of the software classes in the program. The last part of the report contains notes on the implementation of the design as well as the tests that were made to check the liability of the virtual dice. In the end it is concluded, that the project lives up to all the important requirements to the as far as the developers know.

2 Indledning

Vi er blevet stillet til opgave at udarbejde et terningespil-program som skal kunne spilles mellem to spillere. Programmet er udarbejdet på baggrund af projektbeskrivelsen "CDIO del 1.pdf". Vores GitHub-repo kan tilgås via dette link:
https://github.com/PeterRevsbech/CDIO-del-1-gruppe-11.git

3 Krav

Vi har udarbejdet en kravliste, som kan ses i tabel 1. Kravene er er blevet navngivet ud fra deres vigtighed. De er først og fremmest blevet inddelt i bogstaverne A, B, og C hvor A er de mest vigtige krav og C de mindst vigtige. Derudover er de blevet inddelt i funktionelle og ikke funktionelle(IF). De funktionelle krav er dem der beskriver en hel konkret funktion, som programmet skal udføre. Disse kunne f.eks beskrives ved en usecase som: "Brugeren slår med terningerne og får point svarene til summen af terningernes øjne". De ikke funktionelle krav er f.eks at programmet ikke må crashe.

I forhold til KA4 ved vi at der ikke findes en 100% random generator i en computer, men vi har valgt at bruge Math.random funktion til at generere vores terninger, som opfylder de statistiske forhold for en random generator.

Da vi læste i opgaveformuleringen om punkt KB1 i forhold til KB4, skabte det lidt forvirring. KB1 siger at man mister alle sine point hvis personen slår 2 ettere, hvor KB4 siger at hvis man slår 2 ens efter at opnå 40 point vinder man. Gruppen er blevet enig om at fortolke disse 2 krav på følgende måde:
Hvis en spiller har mere end 40 point og slår to ettere, vil vinde for han opfylder KB4.

4 Analyse

Med udgangspunkt i kravene, udarbejdede vi 3 korte usecases. Disse er som følger:

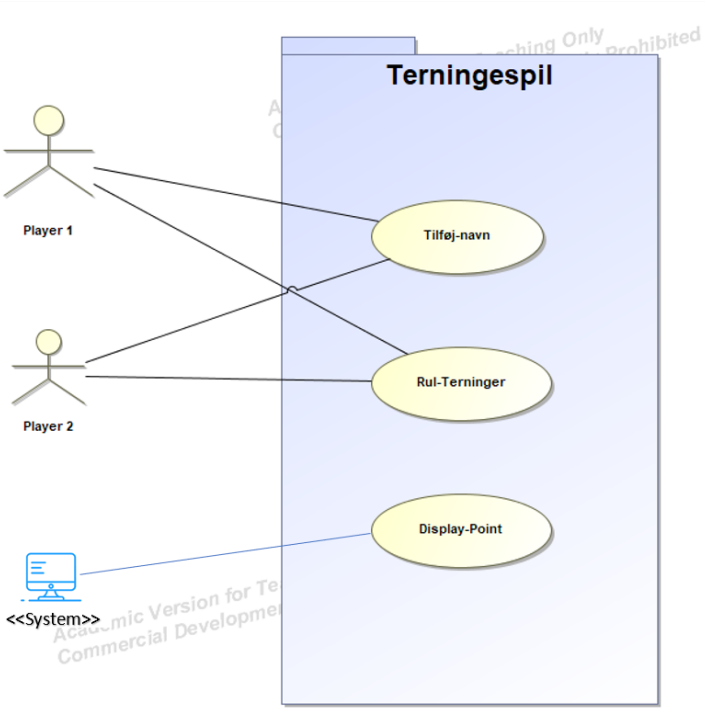
TilføjNavn: brugeren starter spillet. Spillet beder brugeren om et input til brugerens navn. Brugeren giver inputtet og fortsætter spillet.

RulTerninger: Spilleren vælger at rulle terningerne. Spilleren præsenteres med terningernes resultat. I tilfælde to ettere, nulstilles spillerens point. I alle andre tilfælde lægges terningernes øjne til spillerens point. Ved to ens, gives ekstra tur til spilleren. I tilfælde af 2 seksere to gange i træk vinder spilleren. I tilfælde af 2 ens efter 40 point vinder spilleren.

DisplayPoint: Spilleren er i et igangværende spil. Spilleren ønsker at se sit pointtal, og spillet viser pointtallet til spilleren.

Krav ID	Beskrivelse
KA1IF	UTF-8 skal bruges som tegnsæt
KA2IF	Der skal være public get- og set metoder, hvis attributter skal kunne tilgås udefra
KA3	Brugeren skal kunne slå med terningerne
KA4	Terningeslagene skal være tilfældigt genererede og svare statistisk til en rigtig terning
KA5IF	Det må ikke tage mere end 333 millisekunder fra brugeren slår med terningen til øjnene vises
KA6IF	Udviklingen af programmet skal laves med git via en github repo
KA6IF	Master-branchen i github-repoen skal være til testet kode. Udviklingen skal derfor ske i en eller flere developmentbranches.
KA7IF	Der skal udføres og dokumenteres tests af forekomsten af hver terningesum fra 2-12 samt antallet af gange, terningenre viser to ens ved 1000 slag. Resultaterne skal stemme overens med de teoretiske sandsynligheder.
KA8	Spillet er mellem to personer
KA9IF	Alle almindelige mennesker skal kunne spille spillet uden en brugsanvisning.
KA10IF	Koden skal være dokumenteret enten på engelsk eller på dansk.
KA11IF	Programmet må ikke crashe uforudset og skal først afsluttes, når brugerene beder om det.
KB1	Spilleren mister alle sine point, hvis spilleren slår to ettere.
KB2	Spilleren får en ekstra tur, hvis spilleren slår to ens.
KB3	Spilleren kan vinde spillet ved at slå to 6'ere, hvis spilleren også slog to seksere i sidste kast.
KB4	Spilleren skal slå to ens for at vinde spillet. Dette skal ske efter at spilleren har opnået 40 point. Hvis spilleren opnår 40 point og slår to ens i samme slag, vinder spilleren altså ikke. Dette gælder også ved to ettere.
KB5	Programmet skal vise billedet af terningerne når de er blevet kastet.
KC1IF	Den grafiske brugerflade skal være tydelig og overskuelig at se på

Tabel 1: Kravlisten. A angiver de vigtiste krav, som projektet skal leve op til. B angiver de næst-vigtigste, som projektet gerne skal leve op til. C er de mindst vigtige. IF angiver ikke-funktionelle krav.



Figur 1: Usecase-diagram over de vigtigste usecases og systemets aktører

Navn	**Beskrivelse**
rollDice()	Input: intet output: En arraylist med to integers tilfældigt genreret mellem 1 og 6
getSum(<ArrayList>)	input: En arraylist med to integers (fra rollDice()) output: summen af de to integers
getEns(<ArrayList>)	input: En arraylist med to integers (fra rollDice()) output: boolean der viser sand hvis de to integers er ens

Tabel 2: Oprindelig udkast til en oversigt over metoder i main-klassen

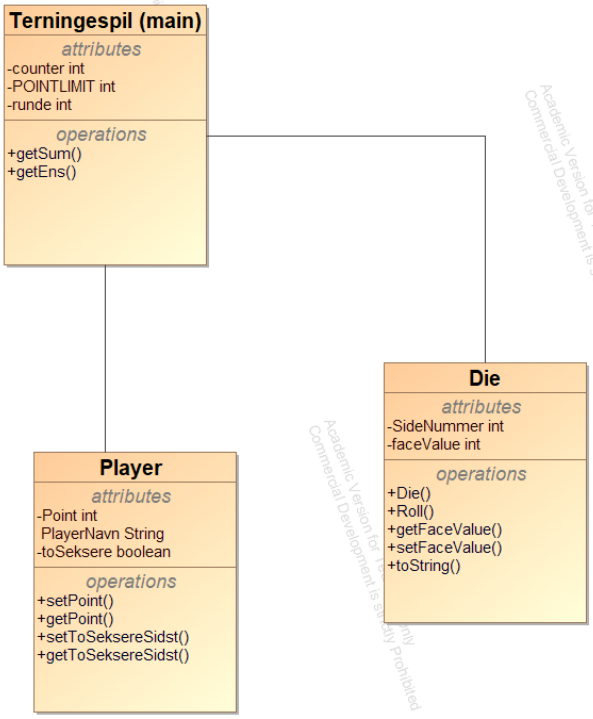
Ud fra disse korte usecase-tekster lavede vi et usecase-diagram, som viser hvordan aktørerne aktiveres i de tre usecases. Diagrammet kan ses i figur 1.

5 Design

Vi udarbejde en liste over de metoder, vi ville have med i programmet ud fra projektbeskrivelsen ”CDIO del 1.pdf”. Listen kan ses i tabel 2. Denne indeholder altså en del metoder i den dengang unavngivne main-klasse. Desuden var tanken at programmet skulle indeholde en Spiller-klasse, med variablene point og navn.

Dette klasse-design blev udarbejdet meget hurtigt og inden vi havde fået en egentlig introduktion til objektorienteret programmering. Det færdige program har derfor en noget anderledes struktur, og indeholder bl.a. også en Die-klasse, som indeholder nogle af de metoder, det oprindeligt var tanken at den unavngivne main-klasse skulle indeholde.

Dette førte til at vi senere udarbejdede et bedre klassediagram, der beskriver de klasser, variable og metoder, som vi er endt med at bruge i programmet. Det færdige klassediagram kan ses i figur 2.



Figur 2: Programmets færdige klassediagram

6 Implementering

Implementeringen af programmet foregik meget sideløbende med og ofte før designfasen, idet vores kendskab til objektorienteret programmering ikke var særlig stort i starten. Idet vi gennem projektet har lært en masse om objektorienteret programmering, ved vi nu, at vi en anden gang kunne lave programmet smartere på visse punkter. Vi har desuden redskaberne til i højere grad at kunne designe udkast vores klassestruktur på forhånd inden vi giver os i kast med at sammensætte hele programmet.

7 Test

Programmet indeholder overordnet to pakker, ”Program”og ”Test”. I Test-pakken blev et script lavet, som tæller antal gange hver sum fra 2 til 12 optræder ved 1000 forsøg med slag af to terninger. Test-programmet printer disse hyppigheder til os i terminalen. Herudfra beregner vi den observerede frekvens af hver sum på vores 1000 testslag. Disse ses i tabel 3, hvor de er sammenlignet med de teoretiske sandynligheder.
Sandsynlighederne for summen af to terningeslag kan beregnes ved at betragte, hvor mange kombinationer, der kan give hver sum. Antallet af kombinationer, n, for summen X følger fordelingen:

$$n = 6 - |7 - X| \ , \ n \in \mathbb{N} \ , \ n < 13 \tag{1}$$

Idet 7 er den sum, som optræder flest gange, og de andre summers sandsynlighed afhænger lineært at deres numeriske afstand fra 7. Vores testdata ses nedenfor i en tabel sammenlignet med de teoretiske sandsyngligheder.

Det ser altså ud til, at vores observerede frekvensser stemmer nogenlunde overens med de teoretiske sandsynligheder. Havde vi lavet en større test med mere end 1000 slag, havde de selvfølgelig stemt bedre overens.

Sum	Kombinationer	Sandsynlighed	Hyppighed	Frekvens
2	(1,1)	$1/36 \simeq 2{,}8\%$	30	3%
3	(1,2),(2,1)	$2/36 \simeq 5{,}6\%$	40	4%
4	(1,3),(2,2),(3,1)	$3/36 \simeq 8{,}3\%$	90	9%
5	(1,4),(2,3),(3,2),(4,1)	$4/36 \simeq 11{,}1\%$	115	11,5%
6	(1,5),(2,4),(3,3),(4,2),(5,1)	$5/36 \simeq 13{,}9\%$	148	14,8%
7	(1,6),(2,5),(3,4),(4,3),(5,2),(6,1)	$6/36 \simeq 16{,}7\%$	161	16,1%
8	(2,6),(3,5),(4,4),(5,3),(6,2)	$5/36 \simeq 13{,}9\%$	135	13,5%
9	(3,6),(4,5)(5,4),(6,3)	$4/36 \simeq 11{,}1\%$	115	11,5%
10	(4,6),(5,5),(6,4)	$3/36 \simeq 8{,}3\%$	81	8,1%
11	(5,6),(6,5)	$2/36 \simeq 5{,}6\%$	49	4,9%
12	(6,6)	$1/36 \simeq 2{,}8\%$	35	3,5%

Tabel 3: Overblik over den statistiske fordeling af summer ved slag med to terninger samt data fra vores test

I samme test blev også optalt hvor mange gange de to terninger vise samme antal øjne. Dette fik vi også programmet til at printe til os.
Resultatet var, at dette skete 168 gange på 1000 slag, hvilket svarer til en frekvens på 16,4%. Dette stemmer fint overens med den teoretiske sandsynlighed for at slå to ens med to seks-sidede terninger, som naturligvis er $1/6 = 16{,}7\%$.

8 Projektplanlægning

Vi begik os straks ud i at programmere efter vi havde udarbejdet vores kravliste og indledende klasse design. Det gjorde vi meget hurtigt og ikke med specielt meget planlægning, da vi fik at vide at projektet skulle afleveres 02/10-2019, hvilket kom noget bag på os. Vi blev senere informeret om, at afleveringsdatoen først var den 4/09-2019. På dette tidspunkt, var udarbejdelsen af programmet allerede begyndt dog uden ret meget analyse og design af f.eks. klassediagrammer. Derefter arbejdede gruppen sammen på projektet primært 30/09-2019 og 01/10-2010 hvor vi stort set satte hele programmet sammen, testede og skrev store dele af rapporten.

9 Konklusion

Programmet er tilsynneladende endt med at virke som det skal, og opfylder indenfor vores viden alle kravene på nær det mindst vigtige krav KC1IF, da den grafiske brugerflade viser et helt matadorbræt. Idet dette krav er et kategori-C-krav må det siges at være okay, at vi ikke nåede at gennemføre det.

10 Bilag

10.1 Timeregnskab

Dato	Peter	Mathias	Derar	Christian	Bashar	Sulaiman
26/09	2t 30m					
27/09		3t		2t		
28/09		2t				
29/09						
30/09	4t	4t	4t	4t	4t	4t
01/10	4t	4t	4t	4t	4t	4t
02/10						
03/10	1t		4t		4t	4t
04/10	2t			2t	2t	2t

Figur 3: Timeregnskab