Danmarks Tekniske Universitet

62531 udviklingsmetoder til it-system
62532 versionsstyring og testmetoder
02312 indledende programmering
Gruppe 23

Hans Christian Leth-Nissen
s205435

Henrik Lynggaard Skindhøj
s205464

Kasper Falch Skov
s205429

Christoffer Fink
s205449

Fillip Igor Meyer Clausen
s205468

Henrik Lynggaard Skindhøj, s205464
Hans Christian Leth-Nissen, s205435
Filip Igor Meyer Clausen, s205468
Christoffer Fink, s205449
Kasper Falch Skov, s205429

Timeregnskab

Kasper Falch Skov, s205429	9,5 timer
Henrik Lynggaard Skindhøj, s205464	8,5 timer
Christoffer Fink, s205449	6,5 timer
Filip Igor Meyer Clausen, s205468	8,5 timer
Hans Christian Leth-Nissen, s205435	8,5 timer
Ole Anker Vibe Lentz, s205475	0 timer

Abstract

The client for this project has completed a set of technical specifications to which we will explore and make some use cases. We will use these use cases to find and elaborate on the wanted classes, and from these classes we will code a dice game. We will prepare a test that proves that our program(dice game) fulfills the statistical laws and probabilities. We will conclude this assignment by reviewing the data from the test.

Henrik Lynggaard Skindhøj, s205464
Hans Christian Leth-Nissen, s205435
Filip Igor Meyer Clausen, s205468
Christoffer Fink, s205449
Kasper Falch Skov, s205429

1. Indledning

Terninger er en fast del af mange forskellige spil, og vi har fået til opgave at programmere et intuitivt nemt terningspil med to spillere. Vi vil lave en F.U.R.P.S på kravspecifikationerne fra kunden, finde ud af de relevante use cases, derefter vi vil definere forskellige relevante klasser, stemme klassernes egenskaber overens med kravspecifikationerne i programmeringsfasen, forberede en test af programmet og derefter udbedre bugs eller andre problemer i programmet.

2. Kravspecificering

- Spilleren mister alle sine point hvis spilleren slår to 1'ere.
- Spilleren får en ekstra tur hvis spilleren slår to ens
- Spilleren kan vinde spillet ved at slå to 6'ere, hvis spilleren også i forrige kast slog to uanset om det er på ekstra kast eller i forrige tur.
- Spilleren skal slå to ens for at vinde spillet, efter at man har opnået 40 point.
- Efter spilleren har lavet et kast, må der ikke gå mere end 333 millisekunder, før de modtager resultatet
- Alle skal kunne finde ud af spillet uden brugsanvisning
- Produktet skal kunne testes på 1000 kast
- Alle gruppemedlemmer skal have lavet et commit.
- Projektet skal indeholde versionsstyring, der bliver udført ved hjælp af et github repository fra master branch

2.1 F.U.R.P.S.

Functionality:

F1: Spilleren mister alle sine point hvis spilleren slår to 1'ere.
F2: Spilleren får en ekstra tur hvis spilleren slår to ens
F3: Spilleren kan vinde spillet ved at slå to 6'ere, hvis spilleren også i forrige kast slog to uanset om det er på ekstra kast eller i forrige tur.
F4: Spilleren skal slå to ens for at vinde spillet, efter at man har opnået 40 point.

Usability:

U1: Alle skal kunne finde ud af spillet uden brugsanvisning

Reliability:

R1: Produktet skal kunne testes på 1000 kast

Performance :

P1: Efter spilleren har lavet et kast, må der ikke gå mere end 333 millisekunder, før de modtager resultatet

Supportability:

S1: Der skal version styres ved hjælp af et github repository fra master branch
S2: Alle gruppemedlemmer skal have lavet et commit.

Henrik Lynggaard Skindhøj, s205464
Hans Christian Leth-Nissen, s205435
Filip Igor Meyer Clausen, s205468
Christoffer Fink, s205449
Kasper Falch Skov, s205429

3. Analyse

I opgavebeskrivelsen har vi fundet nogle svage beskrivelser i kundens vision, som vi har valgt at ændre til hvad vi tror de havde i tankerne da de lavede reglerne. Det bliver bestemt i F2, at man får en ekstra tur hvis man slår to ens. I F3 bliver det sagt, at hvis man slår to 6'ere, to gange i træk vinder man spillet, uanset om det er på et ekstra kast eller i forrige tur. Dette peger imod at kunden havde tænkt, at man kun havde mulighed for at få et enkelt ekstra kast, men vi har valgt at lave vores spil så man kan få uendelig mange ekstra kast.

F1 fortæller at man mister alle sine point ved at slå to 1'ere, og der opstår en konflikt med F4. I den fjerde regel står der, at hvis en spiller skal vinde spillet, skal spilleren enten slå to 6'ere to kast i træk eller hvis spilleren har 40+ point skal spilleren slå to ens.
Vi har valgt at sige, at man mister alle sine point og altså ikke vinder, hvis man slår to 1'ere og har 40+ point.

3.1 Use cases

Vi har i analysefasen opstillet vores use cases. I vores terningspil er der ikke særligt mange use case, da det eneste vores spil skal kunne, er at starte og kaste med en terning. Derfor kun de to use cases Start game, og Roll dice.

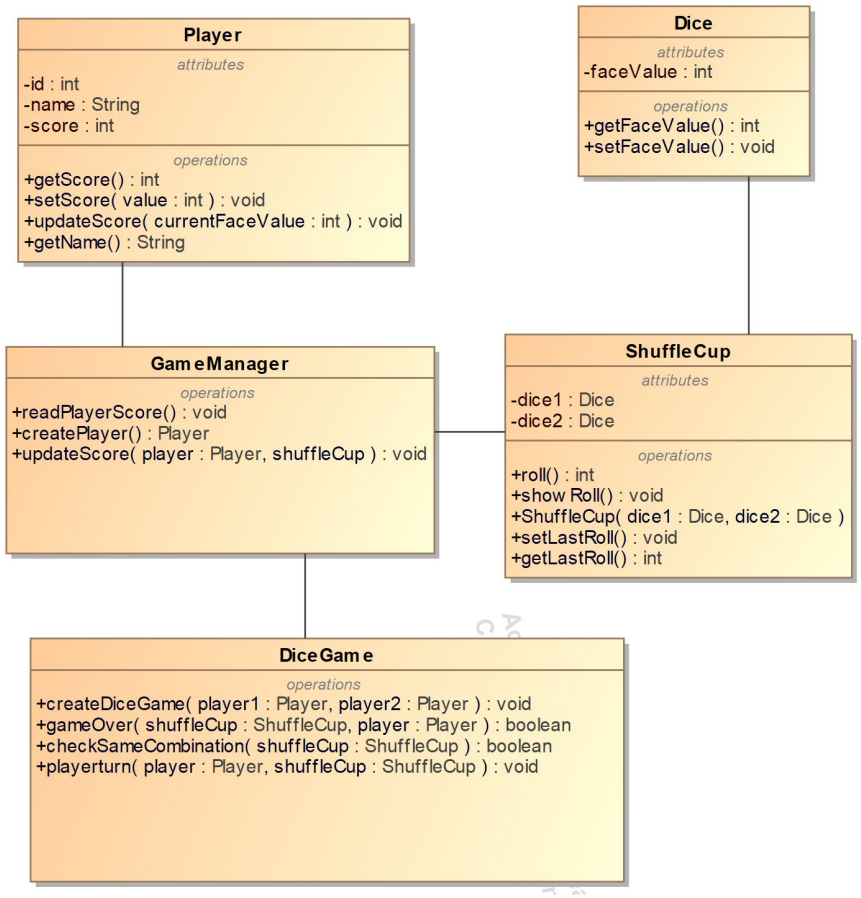
Use case section:	Beskrivelse:
Navn	UC 1.1 - Start game
Scope	System
Level	User Goal
Primær aktør	Player
Preconditions	Spillet er ikke allerede igang
Postconditions	Terningspil er sat op, og klar til at spille
Main Success Scenario	1. System: Opretter terningspil. 2. System: Opretter to spillere. 3. Player: Kan spille spillet.
Extensions	NaN
Frequency of occurence	Hver gang der skal startes et nyt spil

Henrik Lynggaard Skindhøj, s205464
Hans Christian Leth-Nissen, s205435
Filip Igor Meyer Clausen, s205468
Christoffer Fink, s205449
Kasper Falch Skov, s205429

Use case section:	Beskrivelse:
Navn	UC 1.2 - Roll dice
Scope	System
Level	User Goal
Primær aktør	Player
Preconditions	Terningspil er oprettet/startet
Postconditions	Der er rullet en terning
Main Success Scenario	1. System: Spørger efter inputtet “enter” fra spilleren. 2. Player: Trykker “enter” i programmer for at rulle med terningerne, og ruller terningerne. 3. System: Generer to tal fra 1-6, lægger dem sammen, og tilføjer dem til Player’s score.
Extensions	3.1.1. Player: Slår to 1’ere. 3.1.2. System: Sletter Player’s point. 3.2.1. Player: Slår to ens tal som ikke er 1’ere og får en ekstra tur. 3.2.2. Player: Kaster terning igen. 3.2.3. System: Generer to tal fra 1-6, lægger dem sammen, og tilføjer dem til Player’s score. 3.3.1. Player: Slår to 6’ere for anden tur i træk. 3.3.2. Player: Vinder spillet. 3.4.1. Player: Har 40 point og slår ikke 2 ens tal 3.4.2. System: Fortæller den anden spiller at han skal trykke “enter” for at kaste sine terninger 3.5.1. Player: Har 40 point og slår 2 ens tal 3.5.2. Player: Vinder spillet
Frequency of occurence	Hver gang en spiller skal rulle en terning

Henrik Lynggaard Skindhøj, s205464
Hans Christian Leth-Nissen, s205435
Filip Igor Meyer Clausen, s205468
Christoffer Fink, s205449
Kasper Falch Skov, s205429

4. Design



Figur 4.11: Klassediagram af vores program, og alle de parametre vores klasser indeholder.

I designfasen af dette program har vi i gruppen diskuteret frem og tilbage hvordan og hvorledes kundens krav og specifikationer først og fremmest skulle fortolkes, og dernæst inkorporeres på en intuitiv måde. Vi har defineret en række af klasser, og hvad de definerede klassers specifikationer er således, at den programmering der efterfølgende skulle laves, ville blive mere overskuelig. Designet at dette program har undergået et par iterationer, efter vi i gruppen løbende har udformet projektet, og der er derfor kommet flere klasser på, end hvad vi startede ud med.

4.1 Beskrivelse af klasser

Dice:
Dices funktion er at vise værdien af terningkastet genereret af *ShuffleCup* - i intervallet [1-6]

Player:
I vores spil skal der være 2 players, som hver har et id, et navn, og en score. Hver player skal holde på disse 3 værdier, indtil de modtager eller henter værdier fra de andre klasser.

Henrik Lynggaard Skindhøj, s205464
Hans Christian Leth-Nissen, s205435
Filip Igor Meyer Clausen, s205468
Christoffer Fink, s205449
Kasper Falch Skov, s205429

ShuffleCup:
ShuffleCup skal eksistere grundet - at imens terningkastet foretages skal terningerne skjules samt foretage et kast. Hvorefter den samlede værdi fra terningerne, samt den individuelle værdi offentliggøres og resultatet printes.

GameManager:
GameManager er vores controller som bruges til at forbinde vores andre klasser. Det vil altså sige at den tager resultatet af *showRoll* fra vores *ShuffleCup* og sender det til *DiceGame*, som tjekker om resultatet vedrører nogle af de spilleregler, som vi har lavet. Herefter bliver *Score* for den givne *Player* opdateret ved hjælp af *updateScore*.

DiceGame:
DiceGame fungerer som regler for selve spillet, og beskriver hvad spillet går ud på. Den checker så for om en player har slået to ens, altså F1, F2 eller F3, og opfylder reglerne for spillet, hvis nogle af udfaldene er hændt.

5. Implementering

DiceGame-klassen indeholder spillets gang, samt alle regler. Når vi kører spillet, så bruges metoden *createDiceGame*.

createDiceGame bærer “while” løkken, som kører konstant indtil *gameOver boolean* metoden bliver *true*, eller vores boolean *winWith2Pairs* bliver true. *gameOver* metoden er et af spillets regler. Denne *boolean* kan ændres ved to omstændigheder. Den ene er når spillerens score er over eller lig 40 og følges med to ens terning værdier. Den anden er når en spiller slår to 6’ere i streg. Ligeledes er vores boolean *winWith2Pairs* en af spillets regler, som kun bliver true, hvis en spiller har slået to ens af 6’ere to gange i træk.

6. Test

I opgavebeskrivelsen fik vi opgaven, at teste terningspillet ved at kaste to terninger 1000 gange, og se resultatet. Resultaterne af testen skulle indeholde hvor mange gange de forskellige tal fra 2-12 forekom, og hvor mange af de 100.000 kast, der resultere i to ens.

Vi har valgt at foretage en test på 100 gange så meget som kunden forlanger for at være sikre på at vores terninger slår tilfældige værdier. Vi vil analysere udfaldet af testen for at tjekke om testens resultater stemmer overens med sandsynligheds principperne.

Henrik Lynggaard Skindhøj, s205464
Hans Christian Leth-Nissen, s205435
Filip Igor Meyer Clausen, s205468
Christoffer Fink, s205449
Kasper Falch Skov, s205429

6.1 Test med 100.000 slag:

Vi indfører vores procentvise fordeling af resultatet ind i en tabel, og sammenligner den med en tabel med teoretiske procenter, for at se om de stemmer overens. Da det er sandsynlighedsregning, kan vi ikke regne med at resultaterne stemmer fuldstændigt overens, men så længe at fordelingerne ligner hinanden hen over 100.000 kast, kan vi med overvejende sikkerhed sige at vores raflebæger fungerer som det skal.

Test = 100000	%	Teoretisk %
2	2783*100/100000=**2.78%**	1/36*100=**2.78%**
3	5551*100/100000=**5.55%**	1/18*100=**5.56%**
4	8329*100/100000=**8.33%**	1/12*100=**8.33%**
5	11174*100/100000=**11.17%**	1/9*100=**11.11%**
6	13887*100/100000=**13.89%**	5/36*100=**13.89%**
7	16652*100/100000=**16.65%**	1/6*100=**16.67%**
8	13920*100/100000=**13.92%**	5/36*100=**13.89%**
9	10905*100/100000=**10.9%**	1/9*100=**11.11%**
10	8471*100/100000=**8.47%**	1/12*100=**8.33%**
11	5498*100/100000=**5.5%**	1/18*100=**5.56%**
12	2830*100/100000=**2.83%**	1/36*100=**2.78%**
To ens	16660*100/100000=**16.66%**	1/6*100=**16.67%**

Vores skema viser, at resultaterne af testen med 100000 kast, procentvis stemmer meget godt overens med de teoretiske procentsatser for udfaldet af terningkastet. Altså kan vi konkludere at vores raflebæger virker som det skal, samt genererer to tilfældige tal mellem 1 og 6 og lægger dem sammen.

Henrik Lynggaard Skindhøj, s205464
Hans Christian Leth-Nissen, s205435
Filip Igor Meyer Clausen, s205468
Christoffer Fink, s205449
Kasper Falch Skov, s205429

7. Projektplanlægning

Vi har i gruppen aftalt hvordan de forskellige dele af CDIO 1, projektet skulle laves. Vi gjorde dette for at give et overblik til alle gruppemedlemmer om, hvornår vi skulle lave hvad. Vi delte CDIO 1 op i rapport, og kode. Dette gjorde at vi kunne bestemme hvilket dage vi ville lave på rapporten, og hvilket dage vi ville arbejde på koden.

Under selve projektet blev det dog nødvendigt at sætte nogle i gang med at skrive kode, på de dage hvor vi havde aftalt at lave rapport, da der var flere dele i rapporten, der blev nemmere af at have et indblik i, hvordan vores kode ville se ud.

Vi har prøvet at bruge git til at organisere vores udvikling af vores kode. Til at starte med havde vi aftalt at bruge 3 branches, nemlig master, development og bugfixes. Dette er ikke helt gået som forventet, da vi kun har opdateret master branchen 3 gange under hele projektet, hvilket var i starten af projektet, en enkelt gang undervejs, og en gang til sidst i projektet. Derudover har vi endt med at fikse de fleste bugs vi stødte på undervejs og inde i development-branchen frem for i vores bugfixes-branch. Dette gør vores git projekt en smule uoverskueligt at finde hovede og hale i, og det er helt sikkert noget vi skal have mere fokus på til næste gang.
Vi har også bestræbt os efter at committe tit, men det er også blevet glemt et par gange undervejs, hvilket resulterer i nogle store ændringer fra enkelte commits. Dette er også noget vi har italesat i gruppen, og som vi skal være bedre til næste gang.

8. Konklusion

Ud fra vores test kan vi konkludere at den teoretiske sandsynlighed for vores terninger i vores terningspil stemmer overens med sandsynligheden for en realistisk terning. Terningen afviger altså ikke fra den realistiske sandsynlighed. Dermed kan vi konkludere at vores produkt virker inden for fejlmarginen. Vi har ligeledes brugt vores terninger til at lave et terningspil, som arbejder inden for kundens vision.