CDIO DEL 1

1/10/2021

GRUPPEMEDLEMMMER

Ali Houssein Yousef Shanoof

s215716

Anisa Riaz

s216237

Berfin Flora Turan

s215709

Feroz Ali

s215724

Payam Madani

s192077

Musavvar Azimi

s216238

GRUPPE 9

Indholdsfortegnelse

Indledning

Vi har i denne opgave arbejdet tværfagligt med fagene Indledende programmering og Udviklingsmetoder til IT- systemer. Vi har analyseret, designet, implementeret og testet vores udviklet system, ud fra kundens vision og projektlederens krav samt bemærkninger.

Kravspecifikation

Ude fra den udleverede opgavebeskrivelse, blev der udført en analyse af de diverse krav til systemet. Nedenstående ses en liste af de fundne krav til hvad systemet skal kunne. Da der blev fundet en række af krav, blev de sorteret ud fra FURPS+ modellen samt prioriteret ud fra MoSCoW teknikken.

I opgavebeskrivelsen blev der givet en minimumsopgave samt en række af bonusopgaver. Da der vil blive prioriteret i kravene ud fra MoSCoW teknikken, vil kravlisten bestå af krav fra hhv. minimumsopgaven samt bonusopgaverne.

Kravliste

Id	**Kravbeskrivelse**
K01	Spillet skal kunne anvendes på Windows maskinerne i databarerne på DTU.
K02	Spillet skal kunne spilles mellem 2 spillere.
K03	Resultatet af terningeslagene må ikke tage mere end 333 millisekunder.
K04	Summen af terningeslagenes værdier skal kunne gemmes i de enkelte spilleres point.
K05	En spiller skal kunne vinde ved at opnå 40 point.
K06	En spiller skal miste alle point, hvis nævnte spiller slår to 1'ere.
K07	En spiller skal få en ekstra tur, hvis nævnte spiller slår to ens.
K08	En spiller skal vinde spillet, hvis nævnte spiller slår to 6'ere to gange.
K09	En spiller skal vinde spillet, hvis nævnte spiller slår to ens efter at have opnået 40 point.

Tabel 1: Kravliste

Krav K01 til og med K05 beskriver kravene for minimumsopgaven og de resterende beskriver hver især en bonusopgave.

FURPS+

FURPS+	Functionality	Usability	Reliability	Perfomance	Supportability	+
	K04	K02	K10	K01		
	K05			K03		
	K06					
	K07					
	K08					
	K09					

Tabel 2: Krav sorteret ud fra FURPS+

Ovenstående kan en tabel over de forskellige krav sorteret ud fra FURPS+ modellen ses. Størstedelen af kravene tilhører funktionalitets krav, som beskriver hvad spillet skal kunne. Der er ingen krav om at systemet skal kunne virke over en bestemt periode af tid i minimumsopgaven. Dog bliver der bedt om en udførelse af en test, hvor systemet bliver testet over 1000 terningekast, for at sikre, at raflebægeret fungerer korrekt. Af denne årsag er der blevet tilføjet endnu et krav til listen:

K10	Raflebægeret skal virke korrekt over 1000 terningekast

Herefter bliver de diverse krav sorteret ud fra MoSCoW metoden. Der bliver primært taget udgangspunkt i minimumsopgave kravene samt testkravet.

Id	MoSCoW
K01	M
K02	M
K03	M
K04	M
K05	M
K10	M
K06	S
K07	C
K08	C
K09	C

Tabel 3: Krav i prioriteringsrækkefølge ud fra MoSCoW

Der blev i gruppen vurderet, at en løsning af minimumsopgaven var af højeste prioritet. Herefter blev K06 vurderet til at være en 'should have', da gruppemedlemmerne fandt denne især interessant. De resterende krav blev vurderet som værende 'could have's, da de kunne få spillet op på et højere niveau, men der på tidspunktet af prioritering af kravene ikke kunne vurderes om hvorvidt de kunne løses grundet ressourcemangel.

Use-case

For at skabe et bedre overblik over systemets behov samt et bedre indblik i hvordan softwaredesignet kunne se ud, blev der udarbejdet en detaljeret beskrivelse af systemets main use-case, som kan ses på nedenstående tabel.

Use case: Spil Terninge-spillet
ID: U01
Kort beskrivelse: Hvad skal det være? To spillere skal spille terningespillet.
Primær aktør: • Spiller
Sekundær aktør: • Systemet
Stakeholders og interests: • Kunde
Preconditions: 1. Spilleren har tændt sin computer og kan finde programmet enten på DTU's databaser
Main flow: 2. 2 spillere deltager i én lobby 3. Systemet registrerer begge spillere. Spillet starter. 4. Spiller 1 starter med at kaste med begge terninger 5. Systemet viser antallet af tegn der er blevet slået til spilleren 6. Spiller 2 gentager *Step 3-5 bliver gentaget indtil der er blevet opnået 40 points* 6. Spilleren med 40 point prøver at vinde spillet ved at slå 2 ens terninger

Den anden spiller gør det samme indtil der er blevet fundet en vinder

7. *Systemet meddeler nu om spiller 1 eller 2 har vundet*
8. *Spillerne kan nu vælge om de vil afslutte spillet eller spille igen*

Postconditions:

7. Spillet er nu afsluttet og derfor kan spillerne lukke computeren

Alternative flows:

2a.

- 2a.1 Hvis en af spillerne, slår 2 ens terninger, to gange i træk, har spilleren vundet
- 2a.2 I så fald springes der ned til step 7

6a.

- *6a.1Hvis begge spillere vælger at spille igen, vil spillet starte igen*
- *6a.2 Hvis en af spillerne vælger at spille igen, mens den anden går ud, prøver systemet at finde en ny spiller, som kan deltage.*
- *6a 3 Efter den anden spiller er fundet, startes der fra step 2*

Specielle krav:

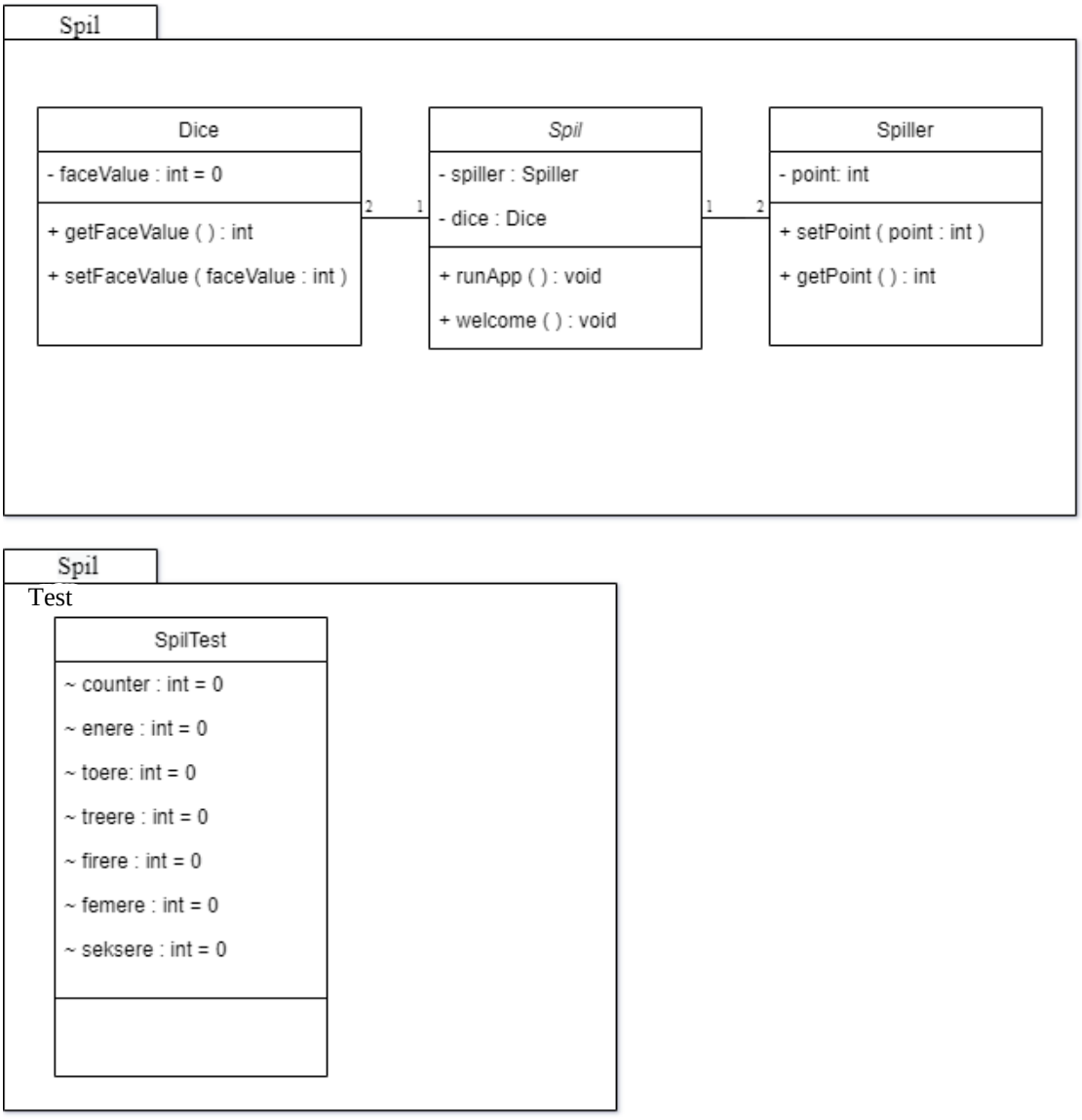
- Netværksforbindelse

Analyse og design

Under opstart af projektet blev der udarbejdet hhv. struktur-, samt adfærdsdiagrammer, for at skabe et overblik over mulige nødvendige packages samt klasser.

Pakkediagram

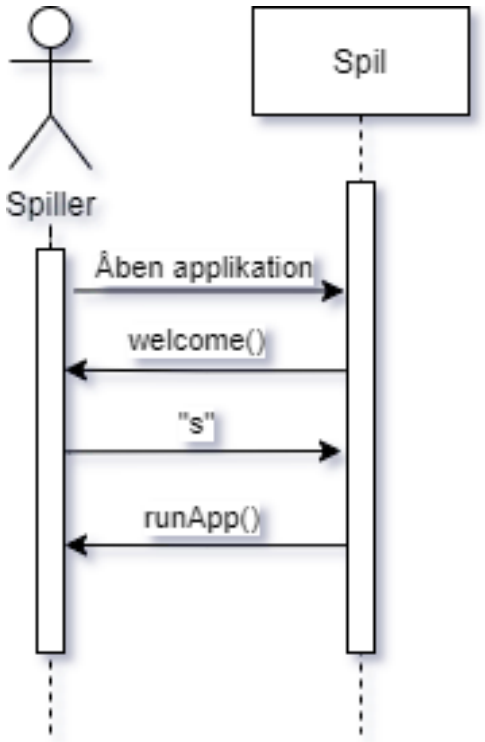
Først blev der udarbejdet en domænemodel for at se på de forskellige associationer, som der kunne være mellem de forskellige klasser i projektet. Dette hørte under projektets analysefase. Den udarbejdede domænemodel blev løbende opdateret under analyse af de diverse krav til projektet samt use-cases. Dette resulterede i et pakkediagram, som kan ses på nedenstående figur. Pakkediagrammet beskriver programmets design nøjere end domænemodellen. Af denne årsag vises pakkediagrammet fremfor domænemodellen.



Figur 1: Pakke-/domænediagram over projektet

Sekvensdiagram

Da pakkediagrammet beskriver projektets struktur, blev der udarbejdet et sekvensdiagram for det udvalgte main use-case, U01, som beskriver en del af programmets adfærd samt de forskellige funktioners sammenhæng. Sekvensdiagrammet kan ses på nedenstående figur.



Figur 2: Sekvensdiagram over main use case

Implementering

Det udviklede spil kan kort beskrives som værende et multiplayer-spil, bestående af to spillere. Hver spiller skal kunne slå med to terninger, som får tildelt en tilfældig værdi mellem et og seks. Dette er blevet løst ved at lave en klasse dedikeret til hhv. en spiller og en terning. Klassen *Dice* består af en attribut, som er terningens faceValue, altså terningens øjne. Derudover består den af en get-metode samt en set-metode. Disse bruges ifb. med faceValue attributten.

Få linjer af *Dice* klassen kan ses nedenstående.

```
private int faceValue = 0;

public int getFaceValue() {
    return faceValue;
}
public void setFaceValue(int faceValue) {
    this.faceValue = faceValue;
}
```

Spiller klassen minder om *Dice* klassen. *Spiller* består af en point-attribut som også er benyttet gennem bl.a. en get-metode samt en set-metode.

Den tredje og sidste klasse i 'Spil' packagen er *Spil*. Denne består af tre metoder. Den første metode er main-metoden, hvori spillet bliver kørt. Herefter er der blevet udviklet følgende to metoder; `runApp()` og `welcome()`.

I main-metoden bliver `welcome()` kørt. Denne metode giver brugeren af spillet mulighed for at taste 's' for at køre `runApp()`. Dette er blevet udviklet ved at implementere et Scanner objekt hvorefter inputtet fra brugeren blev scannet for at tjekke om et 's' er blevet indtastet. Et uddrag af koden kan ses nedenstående.

```
void welcome() {
    System.out.println("Velkommen til terninge-spillet. For at starte spillet:
tast 's' :)");
    Scanner scanner = new Scanner(System.in);
    String tast = scanner.next();
    if (tast.equalsIgnoreCase("s")) {
        runApp();
    }
}
```

`runApp()` er den vigtigste metode i dette projekt, da det er i denne metode det reelle spil er. Denne metode opretter først 4 objekter – hhv. to terning objekter og to spiller objekter. Herefter bliver der initialiseret en variabel kaldet sum og de to spiller objekters point bliver sat til nul ved brug af `.setPoint()` metoden. Resten af metoden virker ved brug af forskellige typer af loops. Der er gjort brug af et for-loop samt nogle if-loops.

Test

Under udviklingen af dette program blev der udviklet tests, hvor der udelukkende blev beskæftiget med en terning. Årsagen til testen kun er implementeret i én terning er som følgende, fordi at terningen for hver runde udvælger tilfældige tal, som vil ende ud i at resultatet altid vil være det samme.

Ud af tusind slag	**Antal slået**	**Antal slået i procenter (%)**	**Statistisk afvigelse i procentpoint**
Antal enere (1)	183	18,3	1,63
Antal toere (2)	154	15,4	-1,27
Antal treere (3)	145	14,5	-2,17
Antal firere (4)	185	18,5	1,83
Antal femere (5)	175	17,5	0,83
Antal seksere (6)	158	15,8	-0,87
Middelværdi	166,6666667	16,67	

Testen er blevet udført ved først at oprette et terning-objekt i klassen *SpilTest* under packagen ’Test’. Dette objekt er blevet sat ind i et while loop, som kører nul til tusind gange. Dette holdes der styr på ved brug af et af de mange initialiseret variabler. Den brugte variabel er blevet kaldt counter. Der er desuden gjort brug af switch-case samt inkrementering, for at optælle forekomsten af de forskellige tal, som terningen kan slå.

Det kan ses på ovenstående tabel, at der er en meget lille statistisk afvigelse, og der kan derfor konkluderes at systemet kan klare 1000 terningeslag samt at brugen af tilfældighedsmetoden for terningernes værdi er blevet brugt korrekt.

Konklusion

Ud fra den udførte test samt sammenligning af det endelige produkt og kravlisten, kan der konkluderes, at alle højest prioriteret krav er opfyldt. Der er blevet taget højde for den udleverede vision samt detaljerede beskrivelser for at udvikle et terningspil mellem to spillere.
Følgende link fører til gruppens GitHub: https://github.com/Berfin20/09_del1

Procesevaluering

I dette projekt har vi hver især brugt 7 timer; hvor i 4 timer blev der sat fokus på implementeringen, og 3 timer på den resterende rapport. Det som gik i den positive retning for gruppen: Fin samarbejde og rimelig fair fordeling af arbejdsopgaver. Det som virkede aftagende for gruppen, var en undervurdering samt hurtig bedømmelse af det individuelle medlem af gruppens kompetence og fokus. Dette har resulteret fremadrettet i en bevidst tanke om dannelsen af et bedre overblik over arkitekturen i selve implementeringskode inden selve processen starter.

Der er blevet udarbejdet et Gantt diagram, som kan ses på Bilag 1 på side 12. Dette diagram afdækker projektets tidsplan, start-, og slut dato for projektets opgavefordeling, over gruppens indsats.

Bilag

Bilag 1

TASK TITLE		START DATE	DUE DATE	DURATION	PCT OF TASK COMPLETE	PHASE ONE													
						WEEK 1					WEEK 2					WEEK 3			
						M	T	W	R	F	M	T	W	R	F	M	T	W	R	F
CDIO 1																			
Projekt start		9/24/21	9/24/21	0	100%														
Kravspecifikationer		9/24/21	9/24/21	0	100%														
Analyse & design		9/25/21	9/26/21	1	100%														
Inplementering		9/26/21	9/28/21	2	100%														
Test		9/28/2021	9/28/21	0	100%														
Diskussion & konklusion		9/29/21	10/1/21	2	100%														