Goldmine

Gruppe 14
Malte Bjørklund - s205789
Mathilde Elia - s215811
Millard Barakzai - s215780
Peter Juul - s215781
Elias Rajabi - s215817

29-10-2021

Resumé

Den rapport omhandler terningespillet ”Guldminen”, der går ud på at slå terninger og efterfølgende lande på et felt. Rapporten følger processen af udviklingen, hvor der startes ud med kravene, systemsekvensdiagram og domænemodel. En kunde stillede nogle krav, som vil blive analyseret ved brug af use case, hvorefter der opstilles et design klassediagram og et sekvensdiagram. Efter analyse vil der blive skrevet om brug af GRASP, som fører videre til Implementering, test og konklusion

Indholdsfortegnelse

Indledning

Denne rapport vil gennemgå processen i udviklingen af terningespillet Guldmine.. Derudover er der gjort brug af diagrammer og modeller for at kunne implementere det i koden.

Krav

ID	**Funktionelle krav**	**Beskrivelse**
K1	2 spillere	Spillet skal kunne spilles mellem 2 spillere
K2	2 terninger	Der bliver kastet 2 terninger ad gangen
K3	2-12 felter	Der skal være 2-12 felter med hvert deres nummer
K4	Ekstra tur	Spilleren får en ekstra tur hvis de lander på felt 10.
K5	Pengebeholdning	Spillernes point gemmes i deres pengebeholder.
K6	Konto-balance	Konto-balancen må ikke blive negativ.
K7	1000 points start	Spillerne starter med 1000 point
K8	Til 3000 point	Spilleren vinder, når vedkommende når 3000 point.

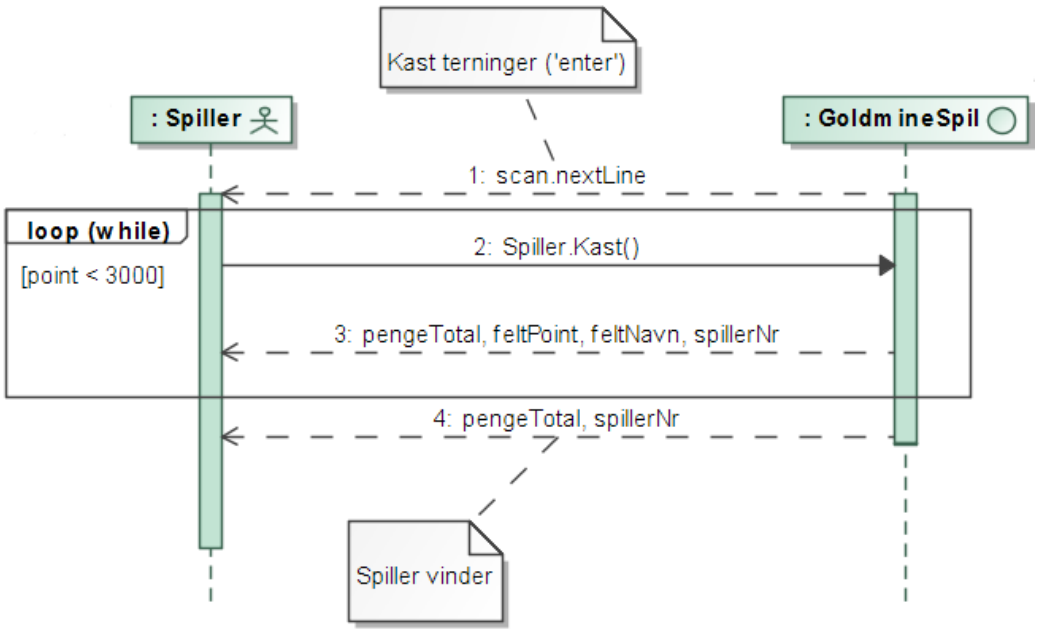
Tabel 1: Overblik over funktionelle krav.

ID	**Ikke-funktionelle krav**	**Beskrivelse**
IK1	Windows	Skal kunne bruges på Windows styresystem
IK2	Sprog	Skal kunne oversættes til andre sprog
IK3	Skift terninger	Koden skal definere antallet af terninger og antallet af terningeflader et enkelt sted.

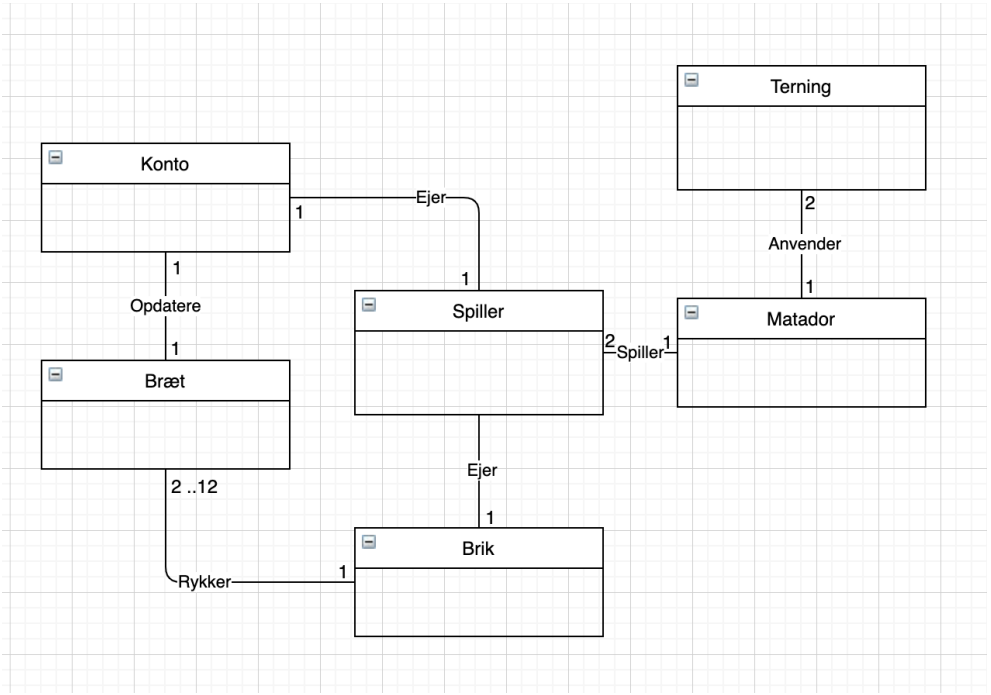
Tabel 2: Overblik over ikke-funktionelle krav.

Efterfølgende blev der benyttet et systemsekvensdiagram og en domænemodel.

Systemsekvensdiagram



Domænemodel



Analyse (Use case)

Use case: Spil Goldmine
Main success scenario:

1. Spiller kører systemet.
2. Systemet starter et nyt spil.
3. Systemet beder spilleren om at trykke Enter for kaste terningerne.
4. Spilleren trykker på Enter.
5. Spilleren lander på et felt, der svarer til summen af terningernes øjne.

6a. Spilleren lander på et felt, hvor spilleren modtager point.

1. Antallet point bliver lagt til spillerens pointsum.

6b. Spilleren lander på et felt (som ikke er felt 10), hvor spilleren fratages point.

1. Antallet point bliver trukket fra spillerens pointsum.

6c. Spilleren lander på felt 10.

1. Antallet point bliver trukket fra spillerens pointsum.
2. Systemet fortæller spilleren de har fået en ekstra tur.

   Scenariet fortsætter fra trin 3.

7. En spiller opnår 3000 point.

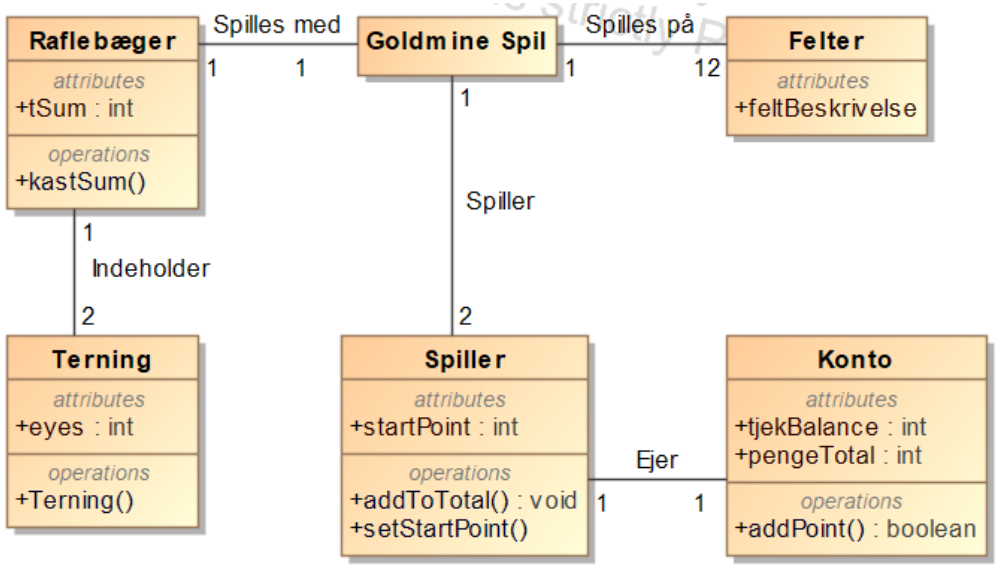
1. Systemet viser at pågældende spiller har vundet.
2. Spillet afsluttes.

8. Systemet giver turen til den anden spiller.

Scenariet fortsætter fra trin 3.

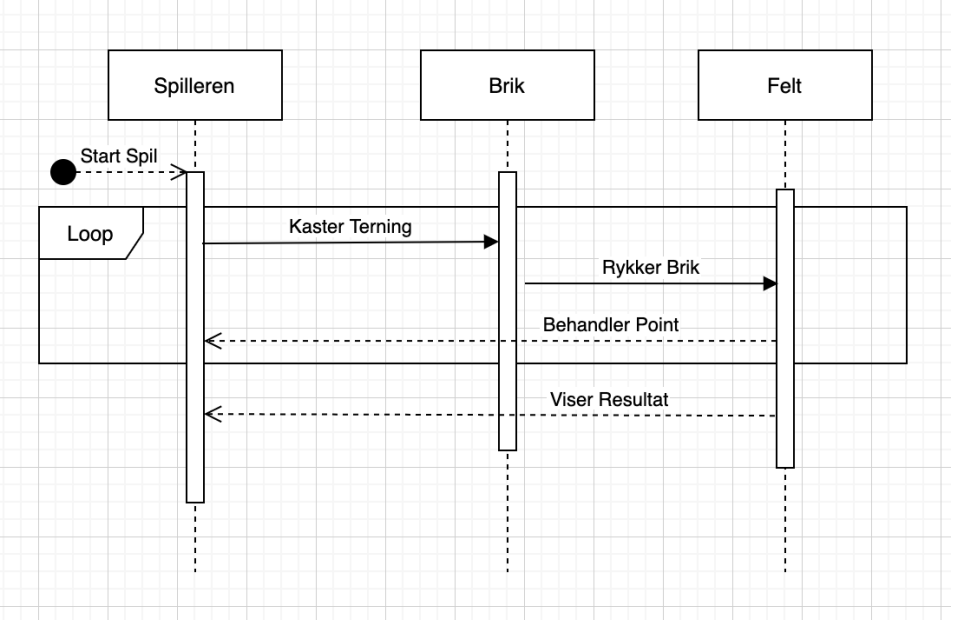
Design

Design klassediagram:



“Goldmine Spil”-klassen repræsenterer main klassen som indeholder spillets gang. Ud fra DCD’et ses det at Goldmine spilles spillet af to spillere, som hver især ejer deres egen konto. Goldmine består 12 felter som hver har deres felt beskrivelse. Derudover spilles spillet med et raflebæger som indeholder to terninger.

Sekvensdiagram:



I forhold til vores usecase repræsenterer vi sekvensdiagrammet med to forskellige objekter, der interagerer med hinanden i systemet under sekvensen. Hvert objekt har en kolonne, og de meddelelser, der udveksles mellem dem, er repræsenteret med pillerne. Pillerne viser at spillerne kaster terninger, og sender en massage til felt objekt. Felt objekt kontrollerer massagen, behandler point og penge, og returneres en massage tilbage til spillerne. Dette foregår mange gang til penge samles op til 3000. det vises med en loop i diagrammet.

Derudover begynder to tidslinjer øverst og falder gradvist for at markere rækkefølgen af interaktioner. På tidslinjer står der activation bars, der viser at objektet er aktivt under en interaktion mellem to objekter. Længden af rektanglet giver varigheden af objekterne forbliver aktive.

Brug af GRASP mønstre
Man kan se eksempler på flere af GRASP mønstrene i vores design.
Vi har generelt forsøgt at følge “high cohesion” og “low coupling” mønstrene, ved at danne klasse med mindre og præcise fokusområder og gøre disse klasser så uafhængige af hinanden som muligt.

Dette ses bl.a. i terning-klassen hvis eneste fokusområde at lave et 6-sidet terninge objekt. Grunden til at terninge-klassens fokusområde kunne holdes så smått, er bl.a. pga. inddragelsen af raflebæger-klassen. Terningerne fungerer som information expert, da de indeholder informationen om en terning (at den har en tilfældig værdi fra 1-6) som raflebægeret skal bruge til at kaste den. Man kunne argumentere for at raflebægeret egentlig er "pure fabrication" i og med at kundens vision ikke beskrev spillet med et raflebæger. Klassen er udelukkende blevet tilføjet for at støtte op omkring high cohesion og low coupling mønstrene.

Raflebægerets fokusområde er at kaste to terninger, vise deres værdi hver især og vise øjensummen. Et af raflebægerets formål er at afkoble terningerne fra selve spillets forløbet, og det følger dermed indirection mønstret. Raflebægeret instantierer terningerne ligesom spiller-klassen instantierer konto-klassen. De kan derfor begge ses som creators.

Implementering

Terning-klassen
Terninge klassen definere en terning, så denne klasse skaber et tilfældigt heltal, indenfor intervallet 1 til 6. Dette bliver gjort med *Math.random()* metoden.
For at opfylde IK3 kravet

Raflebager-klassen
Raflebaeger-klassen består af en KastSum metode, som instantierer terningerne fra terninge-klassen. Der er blevet defineret to terninger (tern1 og tern2) og en heltals sum (tsum). Denne sum består af de to terningeværdier, lagt sammen.

Konto-klassen
Konto-klassen fungerer som spillerens pengebeholdning, hvor spillerens konto balance er givet ved variablen "pengeTotal".

Klassen indeholder metoden "addPoint" hvor man kan lægge penge til kontoen. Er penge-inputtet negativt, fungerer metoden også til at hæve penge fra kontoen.
Under metoden bliver der lavet en midlertidig balance under variablen "tjekBalance". Denne er blevet tilføjet for at "pengeTotal", som er spillerens balance, aldrig bliver negativ.

Derudover returnerer metoden *addPoint* en boolean, som har til formål at fortælle om transaktionen er blevet gennemført. Den returnerer *true* når tjekBalance er over eller lig 0, og ellers returnerer den *false*. Vi har gjort så balancen aldrig bliver negativ ved at sætte pengeTotal lig nul i tilfælde hvor tjekBalance er under 0. I dette spil fremkommer der altså ikke en fejl ved at miste flere penge end man har. Men hvis konto-klassen skulle blive genbrugt i et scenarie hvor man ikke må bruge flere penge end man har, ville boolean metoden blive nyttig.

Spiller- klassen
Spiller klassen har til ansvar at instantiere konto-klassen, så hver spiller er tilknyttet en konto, når spilleren bliver kaldt. I forhold til kravene har en spiller 1000 point som startpoint som er angivet ved variablen “Startpoint”. Konto er blevet gjort private inde i spilleren da spiller-klassen er den eneste som skal bruge kontoen.

Klassen indeholder metoden ”Startpoint” der samarbejde med ”addPoint” fra Konto-klassen hvor angiver noget point til spilleren. Den indeholder også metoden “addToTotal” som forbinder “addPoint” metoden fra konto til spilleren.

Felter-klassen
Felter-klassen er lavet som en constructor for hver spiller. Selve “brættet” er gjort med java’s *switch*, hvor summen af terningerne i RafleBæger() bestemmer hvilken *case* der skal benyttes. Hver *case* i *switch*-udtrykket har objektet “feltBeskrivelse”, som er en *String* der indeholder forklaring af relevante felt.

Main-klassen
Main-klassen er blevet brugt til at forbinde klasserne og printe spillets gang i terminalen. Dette er blevet gjort ved at lave Spiller-objekterne (*new Spiller*) i main, hvorefter objektet *braet* bliver defineret med *new Felter(spiller)*. Derefter er der benyttet *if*-sætninger i en *while*-løkke, for gøre spillet turbaseret mellem to personer.

Test

ID	Test Case
TC1	Test af terninger
TC2	System Test

Test ID	Krav ID							
	K1	K2	K3	K4	K5	K6	K7	K8
TC1		x						
TC2		x	x	x	x	x	x	x

Som det kan ses ud fra matricen er der ikke nogle der opfylder K1, at det er spillere. Dog blev det kørende program testet ved manuel afprøvning af spillet, hvor det kunne ses at det opfyldte kravene.

Test af terninger

```
Antal af de forskellige sum:
2'ere = 30
3'ere = 61
4'ere = 73
5'ere = 106
6'ere = 133
7'ere = 171
8'ere = 142
9'ere = 97
10'ere = 91
11'ere = 63
12'ere = 33
```

Antal af de forskellige sum fra terningerne på 1000 kast.

Værdi	2	3	4	5	6	7	8	9	10	11	12
Sandsynlighed (%)	2,8	5,6	8,3	11,1	13,9	16,7	13,9	11,1	8,3	5,6	2,8

Tabel 3: Den teoretiske sandsynlighed for de forskellige sum

Vi har efterprøvet om de summer, vores program simulere, fordeles korrekt i forhold til den teoretiske sandsynlighed. Det lader til at programmet har bestået prøven, for som der vises i billedet af antallet af vores summer, fordeles tilfredsstillende tæt på vores teoretiske sanlynlighedsprocenter.

Systemtest

Vi har skrevet en testklasse, der skal efterprøve om vores spil nu fungere som vi havde til hensigt. Den klasse definere en spiller, der starter med at spille spillet med 1000 point og derefter bliver ved med at spille spillet, indtil denne spiller ender med at have opnået 3000 point. Testresultaterne bliver skrevet ud, i løbet af hver gentagelse af spillets forløb indtil denne spiller vinder og afslutter spillet.

```java
import org.junit.jupiter.api.Test;

class SystemTest {

    Spiller testSpiller = new Spiller();
    int testSaldo;

    // Test af systemet og spillets gang
    @Test
    void main() {
        testSpiller.setStartPoint(1000);

        while(testSaldo < 3000) {
            Felter braet = new Felter(testSpiller);
            testSaldo = testSpiller.getKonto().pengeTotal;

            System.out.println("Spiller " + "slog_____" + braet.felt);
            System.out.println(braet.feltBeskrivelse);
            System.out.println("Saldo_____" + testSaldo + " Point");
            System.out.println();

        }
        System.out.print("Spiller vinder med " + testSaldo + " point");

    }
}
```

Konklusion

Det kan konkluderes at programmet løser den opgave, der er blevet stillet og alle krav er blevet opfyldt. Programmet virker som det skal, det er blevet testet og har bestået alle prøver. Koden er denne gang bedre dokumenteret, og lettere at læse og forstå, i forhold til sidste gang.

Litteraturliste

Billede til forside:
https://wherethewindsblow.com/wp-content/uploads/2015/07/JUMBO-Six-sided-white-dice.jpg