Danmarks Tekniske Universitet

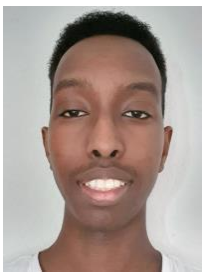
DTU

Gruppe 27 - CDIO 2 – Feltspil

Jasminder Singh Sahota
s235461

Emil Campos Hansen
s230995

Daud Arshad
s235072



Abaas Jama Bootaan
s216173

Rasoul Parmach-Nooristanil
s235118

Rapporten er udarbejdet I de følgende fag:
Versionsstyring og testmetoder - 62532
Udviklingsmetoder til IT-systemer - 62531
Indledende programmering - 02314

27. oktober, 2023

Indholdsfortegnelse

Indledning

I vores spilfirma, IOOuterActive, havde vi sidst et projekt hvor vi skulle udvikle et spil for en kunde, der ud fra deres ønsker og vision ville have et terningspil. Dette udarbejdes ud fra opstillede krav samt analyser.

Vi bruger i dette spil vores viden, samt erfaring fra følgende fag:

- Indledende programmering
- Udviklingsmetoder til IT-systemer
- Versionsstyring og testmetoder

Ud over ovenstående fag, bruges generel viden, bøger og kilder fra internettet.

Alt dette skal hjælpe os med at udvikle vores spil fra bunden, men samtidigt vise og vejlede hvordan og hvorfor de forskellige valg foretages, samt hvilken indflydelse dette har på vores endelige kode. Derudover skal der udføres tests for at verificere om koden virker som den skal, og indebære de givne samt opstillede krav.

Vi har tidligere udviklet et spil til denne kunde, der blev så imponeret over vores terningespil i det forrige projekt, at de nu ønsker at vi endnu udvikler et spil for dem. Denne gang skal vi udvikle et feltspil, som er et spil mellem 2 spillere, hvor de begge starter med et hvis beløb. De skal hver tur kaste terninger hvor de så lander på et felt som enten giver dem flere penge eller mister penge.

Herudover er det opstillet lidt flere og hårdere krav, som vi selvfølgelig bearbejder og implementerer i spillet, samt det der ikke er muligt, vil blive diskuteret og løst ved hjælp af kommunikation med kunden, samt supervisors.

Kundens vision

Kunden har konkluderet, at dette spil skal udføres mellem to personer. Derudover er det et krav, at det skal kunne spilles på DTU´s databarer, uden skavanker samt delays.

Derudover har kunden givet os nogle ekstra krav vi skal arbejde ud fra, samt har vi opstillet vores krav til koden/spillet.

Der skal slås på skift med to terninger, herefter skal der tjekkes hvilket felt de lander på. Felterne, der landes på, kan have både negative og positive effekter på spillernes pengepung, samt skal der udskrives på skærmen en tekst om det valgte felt.

Alle spillerne starter med 1000 point i deres navn.

Når 3000 point er nået for en af spillerne er spillet slut og den givne spiller med 3000 point har vundet.

Dette spil skal spilles af mennesker med forskellige sprog forståelser, og derfor skal spillet let og hurtigt kunne oversættes til andre sprog. Samt for at gøre spillet mere spændende skal terningerne på ethvert tidspunkt kunne udskiftes.

For at gøre det nemmere at udvikle ny spil til kunden i fremtiden, skal deres point/pengebeholdning kunne let overføres til et andet spil.

Herunder ses feltlisten:

Feltliste

1.	(Man kan ikke slå 1 med to terninger)		
2.	Tower	+250	
3.	Crater	-100	
4.	Palace gates	+100	
5.	Cold Desert	-20	
6.	Walled city	+180	
7.	Monastery	0	
8.	Black cave	-70	
9.	Huts in the mountain	+60	
10.	The Werewall (werewolf-wall)	-80,	men spilleren får en ekstra tur.
11.	The pit	-50	
12.	Goldmine	+650	

Kravspecificering

Derudover har vi fået nogle krav, som er ligesom nogle mål, vi skal opnå i dette projekt. Derfor hat vi lavet en kravliste over alle de ting som vi skal opnå.

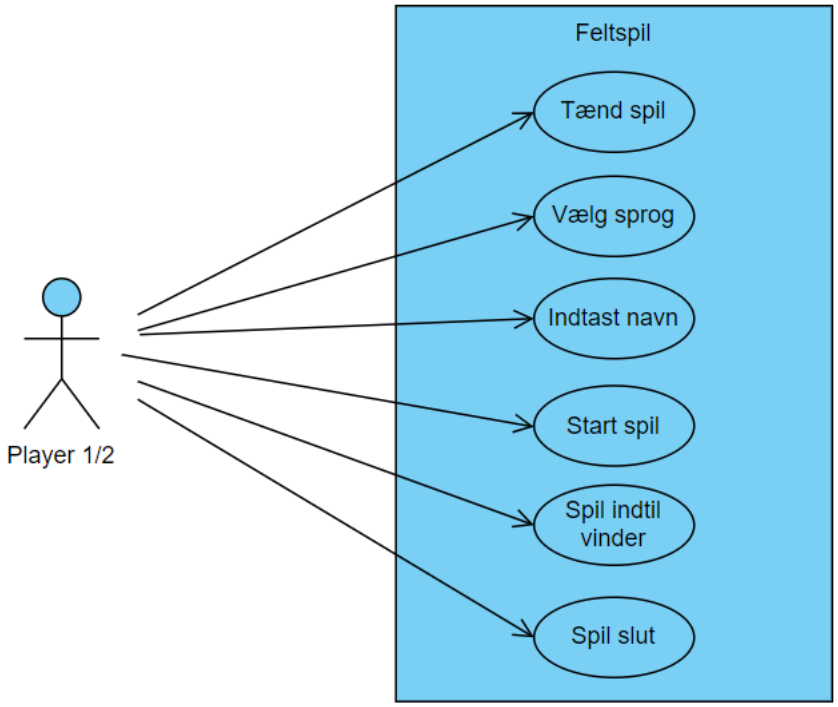
Kravliste:

- Design Klassediagram
- Use case diagram
- Domænemodel
- System-sekvensdiagram
- Grasp principper
- En klasse Spiller der indeholder spillernes attributter og funktioner.

- En klasse konto der beskriver spillernes pengebeholdning.
- Overvej hvilke typer attributterne i Spiller, samt i Konto skal have. Lav passende konstruktører.
- Lav passende get og set metoder.
- Lav passende toString metoder.
- Tilføj metoder til at indsætte og hæve penge på en konto.
- Ændr Konto-klassen således at der ikke kan komme en negativ balance, ligeledes skal metoderne fortælle om transaktionen er blevet gennemført

Use cases

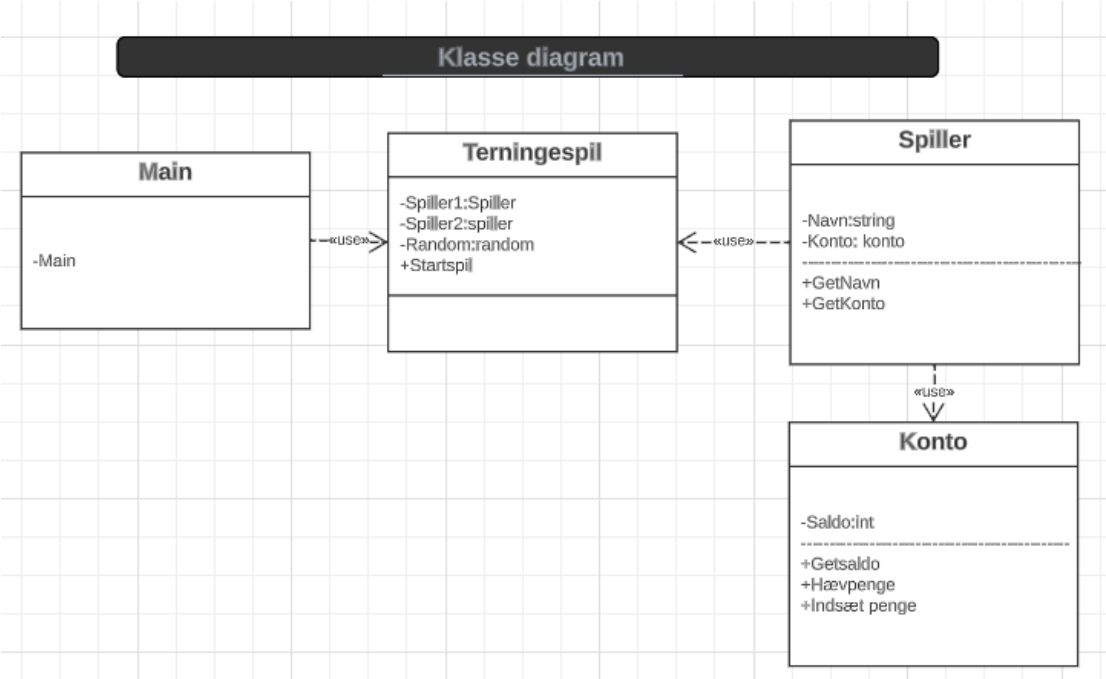
Actor	Goal
Spiller 1/2	Tænd spil
	Vælgsprog
	Indtast navn
	Start spil
	Spil indtil vinder fundet
	Afslut spil



Klassediagram

Klassediagrammer bruges til at give et strukturelt overblik over et system, hjælpe med at planlægge og designe softwareapplikationer, lette kommunikationen mellem interessenter og udviklere, og tjene som grundlag for kodegenerering og implementering af systemer.

I dette diagram ser vi hvordan alle faktorer og klasser arbejder sammen. Med dette menes at det viser hvordan de forskellige klasser arbejder frem og tilbage mellem hinanden. Derudover ser vi de forskellige ansvarsområder, som dermed giver overblik over, hvordan man skal gå til opgaven på bedst mulig måde.

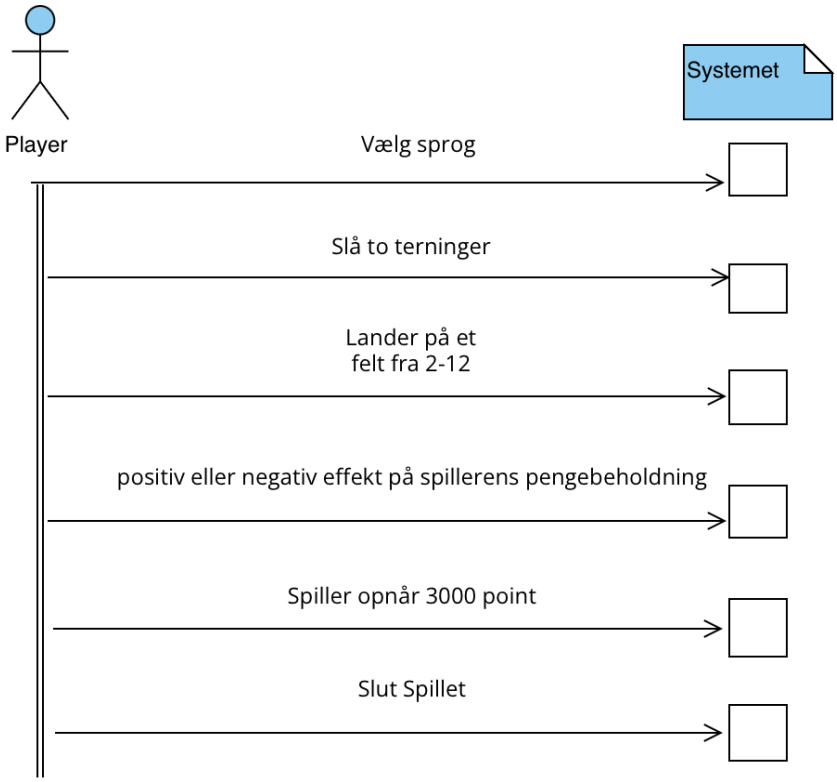


Sekvensdiagram

Vores sekvensdiagram opstillet bruges I bund og grund bruges til at give et detaljeret indblik i, hvordan objekter i et system samarbejder over tid, hvilket er afgørende for udvikling, vedligeholdelse og forståelse af komplekse systemer.

I vores diagram ser vi til venstre vores spiller, samt på linjerne under noteret hvordan de forskellige funktioner samarbejder med vores kode/system. Derudover er funktionerne opstillet i rækkefølge (oppe fra ned), i forhold til spillet.

Herunder ses vores sekvensdiagram (Unified Modeling Language):



Grasp principper

Information Expert

Konto-klassen har ansvaret for at opbevare saldoen og udføre operationerne: indsættelse og hævning af penge. Dette er i overensstemmelse med Information Expert-princippet, da det er den mest passende klasse at håndtere denne information.

Creator

Terningspil-klassen samler, indeholder, bruger og initialiserer data for spilleren.

Hvilket gør terningklassen til creator af, spiller-klassen

Low coupling

I koden er der lav kobling mellem klasserne for at gøre dem så uafhængige af hinanden som muligt. For eksempel indeholder Terningspil-klassen ikke kode om hvordan, Konto- og Spiller-klasserne.

Dette gør at klasserne er mere fleksible og øger uafhængigheden af klasserne. Derudover er klasserne kun associeret med de nødvendige klasser.

Det medfører at:

- Isolation of changes
- Easier to understand the design
- Easier to recycle

High cohesion

Der er høj ”cohesion” i koden. F.eks. har Konto-klassen ansvaret for penge og har operationer tilføjelse og hævning af penge. Spiller-klassen har ansvar for spillerens navn og konto.

Det medfører at

- Uniform area of responsibility
- Supports Low Coupling
- If there are several areas of responsibility in a class, several links quickly arise
- Better understanding
- Easier maintenance
- Easier Tests
- Easier recycling

Test

I denne rapport tester vi saldoerne for om de kan komme under nul.

I stedet for at køre programmet indtil vi landede på tilstrækkeligt nok minus felter i træk for at se om saldoen kunne komme under 0, har vi i stedet sat alle felterne til at give minus.

```
switch (sum) {
    case 2:
        feltEffekt = -250;
        feltBesked = "Du har fundet mønter og smykker i et tårn og sælger det: +250";
        break;
    case 3:
        feltEffekt = -100;
        feltBesked = "Du falder ned i et krater: -100";
        break;
    case 4:
        feltEffekt = -100;
        feltBesked = "Du indtræder en palads port og får penge ved indgangen: +100";
        break;
    case 5:
        feltEffekt = -20;
        feltBesked = "Du ankommer til en kold ørken og mister penge: -20";
        break;
    case 6:
        feltEffekt = -180;
        feltBesked = "Du befinder dig i en befæstet by og finder værdifulde genstande: +180";
        break;
    case 7:
        feltEffekt = 0;
        feltBesked = "Du indtræder et tempel, her er tomt: 0";
        break;
    case 8:
        feltEffekt = -70;
        feltBesked = "Du faret vild i den sorte grotte: -70";
        break;
    case 9:
        feltEffekt = -60;
        feltBesked = "Du fandt nogle forladte hytter i bjergene og fandt værdifulde genstande: +60";
        break;

    case 10:
        feltEffekt = -80;
        feltBesked = "Du stødte ind i en stor mur: -80 (Ekstra tur)";
```

Felteffekt sat til minus

med disse felteffekter, kunne vi hurtigt se om saldoen kunne komme under nul.
Vi er med testen kommet frem til at det er muligt at saldoen kan kommer under nul

```
Daud, tryk på Enter for at kaste terningerne.

Daud kastede 2 og 3 (sum: 5)
Du ankommer til en kold ørken og mister penge: -20
Saldo for Daud: 1050

Emil , tryk på Enter for at kaste terningerne.

Emil  slog 6 og 6 (sum: 12)
Du har fundet guld i minen og sælger det: +650
Saldo for Emil : -510
```

Viser at Emils saldo er kommet under nul

Vi har forsøgt at løse fejlen i koden, men ikke fået løst det. Vi har bla, forsøgt at ændre konto klassen til:

```java
class Konto {
    private int saldo;

    public Konto(int startSaldo) {
        this.saldo = startSaldo;
    }

    public int getSaldo() {
        return saldo;
    }

    public void hævPenge(int beløb) {
        if (beløb > 0) {
            if (saldo >= beløb) {
                saldo -= beløb;
            } else {
                saldo = 0;
            }
        }
    }

    public void indsætPenge(int beløb) {
        if (beløb > 0) {
            saldo += beløb;
        }
    }
}
```

Konklusion

I denne opgave har vi udviklet et terningspil, hvor spillerne slår på skift med 2 terninger og lander på et felt med numrene fra 2 - 12. Hvert af disse felter har en positiv eller negativ effekt på spillernes pengebeholdning. Vi har implementeret et felt oversigt, hvor hvert enkelt felt har en tilknyttet tekst, der beskriver den aktuelle effekt.

Vi udarbejdede en kravliste, hvor vi identificerede de vigtigste krav til spillet. Dette gav os en klar retning og et fælles mål for udviklingen.

Herefter udarbejdede vi et use case diagram, som hjalp os med at identificere de forskellige aktører og deres interaktioner med systemet. Dette gav os et overblik over de forskellige scenarier, der skulle implementeres i spillet.

For at visualisere interaktionen mellem aktørerne og systemet har vi udarbejdet en system sekvensmodel. Dette gav os et visuelt billede af, hvordan systemet skulle reagere på aktørernes handlinger og hvilke informationer der skulle udveksles. Klassediagrammet har også været et vigtigt værktøj for os, for at visualisere systemets design.

Samlet set har vi opnået vores mål, dog med nogle mangler. Vi har haft en klar struktur og planlægning i form af b.la. kravliste, use case diagram, use case beskrivelser og system sekvensmodel, hvilket har medvirket til at udvikle spillet på en systematisk og effektiv måde.

Bibliografi

DTU. (2021). *CDIO - delopgave 2.* Lyngby: DTU.