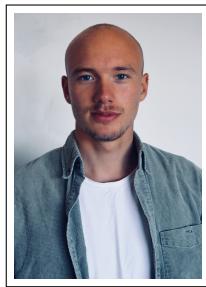


CDIO 2 - Goldminer

Group 15

Deadline: 9th of November 2018

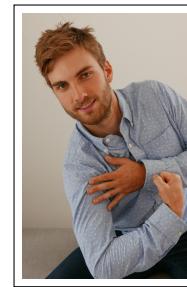
This report contains 17 pages excluding front page and appendix



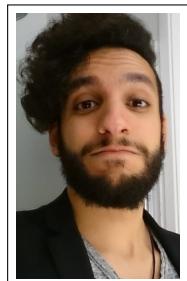
Karl Emil Jeppesen
s180557



Søren Poulsen
s180905



Alfred Röttger Rydahl
s160107



Noah F. M. Hamza
s185084



Rasmus Sander Larsen
s185097

Danmarks Tekniske Universitet

02313 Udviklingsmetoder til IT-systemer, 02312/14 - Indledende programmering
& 02315 - Versionsstyring og testmetoder.

Indhold

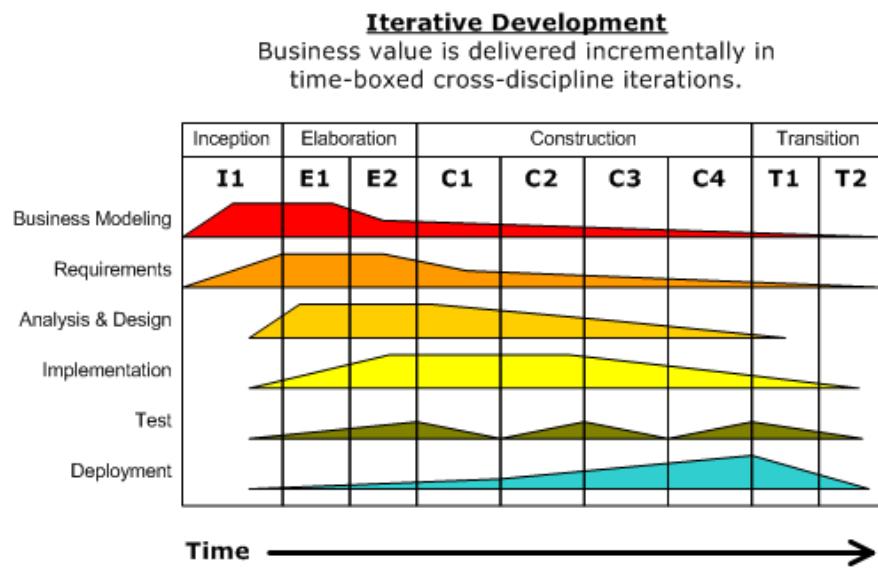
1	Introduktion	1
1.1	Timestyring	2
2	Kravspecifikation	3
2.1	Spilkrav	3
2.2	Feltliste	3
2.3	Test krav	4
3	Analyse	4
3.1	Interessentanalyse	5
3.2	Use cases og kravsanalyse	5
3.3	Domaenemodel	6
3.4	System sekvensdiagram	7
3.5	Risikoanalyse	9
4	Design	9
4.1	Designklassediagram	10
4.2	Sekvensdiagram	10
4.3	GRASP-mønstre	12
5	Implementering	12
5.1	Game package	12
5.2	Objects package	13
5.3	Player Package	13
5.4	Language package	14
5.5	Language file	15
6	Verifikation	15
6.1	Test	15
6.2	Account test	15
7	Projektforløb	16
8	Konklusion	17

1 Introduktion

Denne rapport er udarbejdet af gruppe 15 i "Indledende programmering", "Udviklingsmetoder til IT-systemer" og "Versionstyring og test metoder" i forbindelse med et projekt bestilt af "IOOuterActive".

Spillet går i alt sin enkelthed ud på at spillere rykker rundt mellem 11 felter, som alle har forskellig konsekvens for spillernes pengebeholdning. Spilleren lander på fletet tilsvarende de rullede terningers øjne.

Under udviklingen af spillet er ændringer løbende blevet 'committed' gennem GIT således at udviklingsprocessen kan følges. Udviklingen har foregået efter Unified Process strukturen (figur 1) og der er anvendt GRASP-mønstre.



Figur 1: Diagram over Unified Process

1.1 Timestyring

Herunder fremgår gruppemedlemmernes timeforbrug fordelt på specifikke opgavetyper.

Opgave/Navn	Alfred	Søren	Rasmus	Noah	Karl Emil	
Design	5	2	2	3	2	
- Navneord	0	1	0	1	0	
- Designklasse	1	1	1	1	4	
- System sekvens	0	1	0	3	0	
- Sekvens	0	0	0		0	
Implementering						
- Die	2	0.5	1	0	0	
- Cup	2.5	0	1	0	0	
- Account	2	0	0.5	0	0	
- Player	2	0	0.5	0	0	
- Opsætning	1	1	4	0	2	
- Sprog	1	2	5	0	0.5	
- Game	0	0	1	0	2	
- Turn	0	0	2	0	3	
Test						
- Die	2	0	0	0	0	
- Cup	2	0	0	0	0	
- Player	1	0	1	0	1	
- Account	0	2	0		0	
Rapport						
- Kravspecifikation	0	2		6	2	
- Analyse	0	2	0	1	3	
- Verifikation	0	2.5	0	0	0	
- Implementering	0	0	3	0	0.5	
Total Nettoforbrug	21.5 101.5	17	22	21	20	timer

2 Kravsspecifikation

Kravsspecifikationen fremgår i dette afsnit, som den er stillet i opgaven.

2.1 Spilkrav

- K1: Spil mellem 2 personer.
- K2: Skal kunne køres på DTU databars maskiner uden mærkbar forsinkelser.
- K3: Spillerne slår på skift med to terninger.
- K4: Der skal anvendes to 6-sidede terninger.
 - K4.1: Det skal være let at skifte terningerne.
- K5: Der skal være 11 felter.
 - K5.1: Felterne har numrene 2-12.
 - K5.2: Felterne har hver deres positive eller negative konsekvens for en spillers pengebeholdning.
 - K5.3: Der skal være en tekst tilknyttet hvert af felterne som beskriver feltet.
- K6: Spillerne starter med en pengebeholdning på 1000.
- K7: Spillerens balance må ikke kunne blive negativ.
- K8: Spillet er slut når en spiller opnår 3000 i sin pengebeholdning.
- K9: Spillet skal være let at oversætte og skifte til et andet sprog.
- K10: Klasserne "Player" og "Account" skal kunne anvendes i andre spil.

2.2 Feltliste

Herunder fremgår feltlisten. Denne inkluderer feltets nummer og navn, mængden af guld som feltet adderer eller subtrahere fra spillerens pengebeholdning, samt feltets handlings beskrivelse.

2. Tower +250 "*Du har fundet en skat på toppen af tårnet og sælger den for 250.*"
3. Crater -100 "*Du undgik lige akkurat selv at falde i krateret, men tabte 100 guld.*"
4. Palace gates +100 "*Du plyndrer et palads, og værdierne får du solgt for 100 guld.*"
5. Cold Desert -20 "*Din rejse har ført dig ud til en is-ørken, en ørkenboerne viser dig ud for 20 guld.*"
6. Walled City +180 "*Du kommer forbi en by med en kæmpe mur, du får 180 guld som gave og fortsætter din rejse.*"
7. Monastery 0 "*Du finder et kloster, men de var plyndret for nyligt og har ikke mere guld.*"

8. Black cave -70 "Du søger ly i en mørk grotte, men pludselig støder du på et spøgelse, og flygter så hurtigt du kan. Du taber 70 guld i hasten."
9. Huts in the mountain +60 "Du finder det originale bjergfolk, og plyndrer dem for alle deres værdier. Du går videre med 60 guld mere og et smil på læben."
10. The Werewall (werewolf-wall) -80, men spilleren får en ekstra tur. "Du går op til en kæmpe mur, men pludselig bliver du jagtet af fire hunde ligende skikkeler! Du taber 80 guld, men er allerede på vej et nyt sted hen."
11. The pit -50 "Du finder et hul, og vælger at lege på kanten af det. Desværre får du overbalance, og taber 50 guldmønter ned i hullet."
12. Goldmine +650 "Du fandt en masse guldgravere i en guldmine, og stjæler deres formue. Du forlader dem grædene, mens du fløjter og er 650 guld rigere."

2.3 Test krav

Herunder beskrives kundens tanker med hensyn til test af spillet.

- T1: Testene skal være mulige at gentage.
- T2: Der skal være afprøvnings koder i projektet.
- T3: Der skal testes at spillerens balance ikke kan blive negativ.

3 Analyse

Der skal igen udvikles et terningespil, hvorfor det er nødvendigt at bestemme, hvilke klasser der skal interagere med hinanden.

Aktøranalysen viser, at der er 'spillere', som interagerer med 'terninger' i et større 'spil', der har en række bestemte *regler*.

Der designes en 'die'-klasse, der ved kald returnerer et tal tilsvarende en ternings øjne. Det returnerede tal skal falde inden for den statistisk sandsynlighed. *Denne klasse genbruges i høj grad fra tidligere spil.* Der designes en 'player'-klasse, der holder styr på spillerens pengebeholdning og navn. Der designes en 'fields'-klasse, som indeholder felternes navne, placering, point og beskrivelse. Og afslutningsvis oprettes en 'game'-klasse, som indeholder spillets regler og interagerer med spillerne og terninger på en måde, der konstituerer spillet.

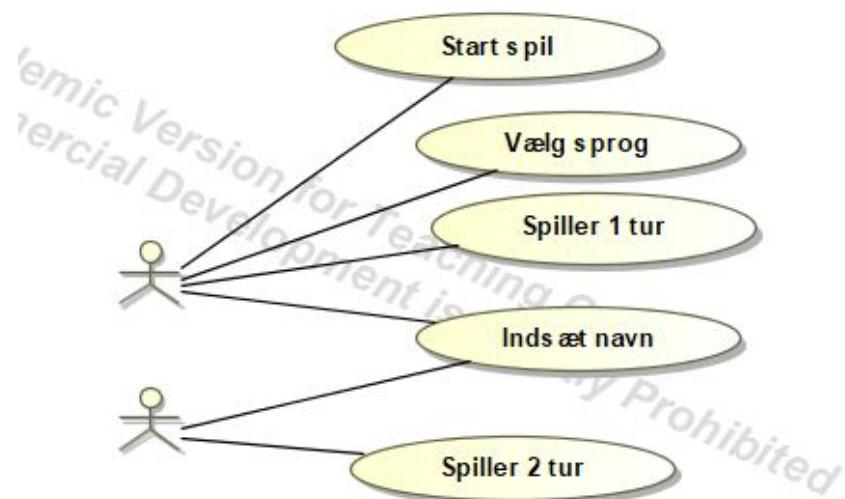
I forlængelse heraf oprettes en række støttende klasser. Der laves en 'turn'-klasse, som sørger for at en tur kører som den skal, at det bliver alterneret mellem spillere, og at ens terningers værdi bliver konverteret til et felt og det handlinger i spillet. Der laves en 'account'-klasse som hjælper med at holde styr på og opdatere spillerens pengebeholdning, samt sørger for at pengebeholdningen ikke kan være negativ. Der laves en 'cup'-klasse, som indeholder et antal terninger og sikrer at disse kan slås. Desuden oprettes en 'reader'-klasse som kan oversætte spillet mellem flere sprog i kombination med 'language'-klassen.

3.1 Interessentanalyse

Der er også i denne opgave interesserter i form af opgavestilleren, programmører og brugere. Opgavestilleren, 'IOOuterActive', har interesse i at programmet fungerer som ønsket, lever op til kravene og er let at bruge. Programmørerne er interesserede i, at kravene er klart definerede og er inden for en realistisk ramme. Brugeren er interesseret i, at programmet ikke har fejl og er intuitivt at bruge.

3.2 Use cases og kravsanalyse

Et simpelt use-case diagram for interaktionen mellem spillere og systemet er afbilledet herunder.



Figur 2: Diagram over Use Cases

De viste use-cases er kortfattet beskrevet herunder.

Use case beskrivelser

Use case	Beskrivelse	Krav
Start spil	En bruger åbner programmet	K2, K5
Indtast navn	Spilleren indtaster et navn	K1, K6
Vælg Sporg	Sprog vælges	K9
Tur: Spiller 1	Spilleren rafler med terninger og tildeles point	K7, K10, T3
tur: Spiller 2	Spilleren rafler med terninger og tildeles point	K7, K10, T3
Bestem vinder	Spiller med en score over 3000 udnævnes som vinder	K8

Som ønsket forefindes en fuld beskrivelse af central use case i det følgende.

Use casen 'Check if has won' tilgåes af systemet mellem hver slag. Efter at spilleren er tildelt point undersøges det, om dennes point er større end den mængde nødvendig for at vinde. Use casen bruges således ikke af spillerne, men melder tilbage til spillet, som returnerer informationen til spillerne.

Use case sektion	Beskrivelse
Navn	Vælg sprog
Scope	Terningspil
Level	Bruger
Primær aktør	Brugerne
Andre interesserter	IOOuterActive
Preconditions	Start program
Postconditions	Sproget ændres til det valgte
Vigtigste success-scenarie	Sproget bliver ændret
Specielle krav	Ingen
Teknologi og dataliste	Udskriver det valgte sprog
Udvidelser	Ingen
Frekvens	Hver gang programmet startes

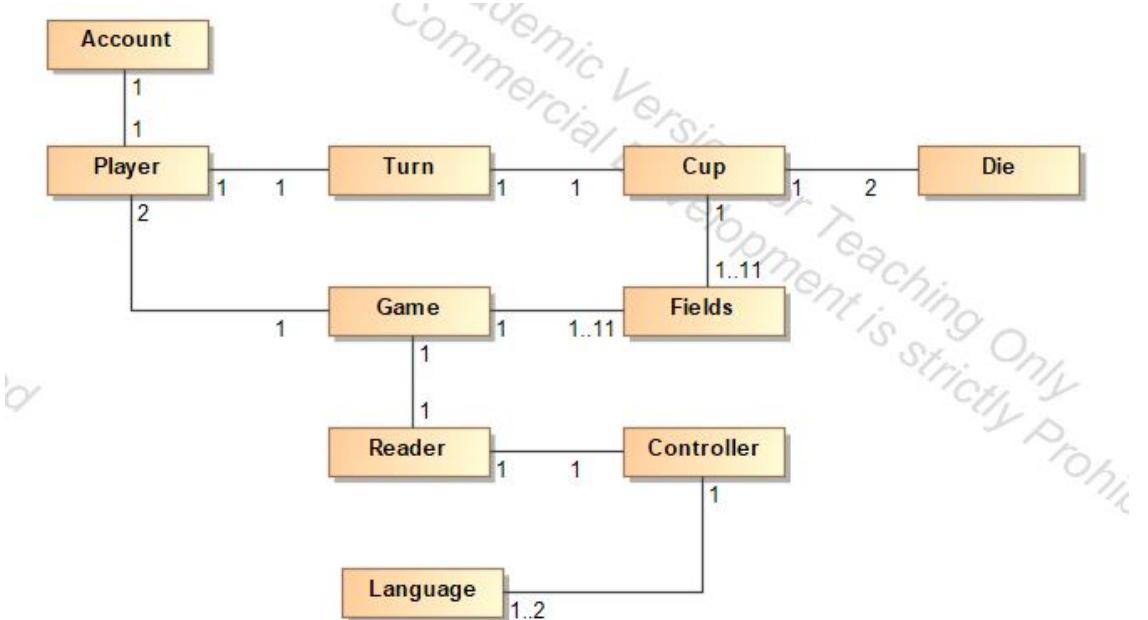
3.3 Domænemodel

En simpel **navneordsanalyse** giver et overblik over de nødvendige klasser og attributter. Disse er listet nedenfor.

Player, Game, Die, Account, Turn, Cup, Reader, Language, Controller and Fields.

Domænemodellen illustrerer spillets virkemåde.

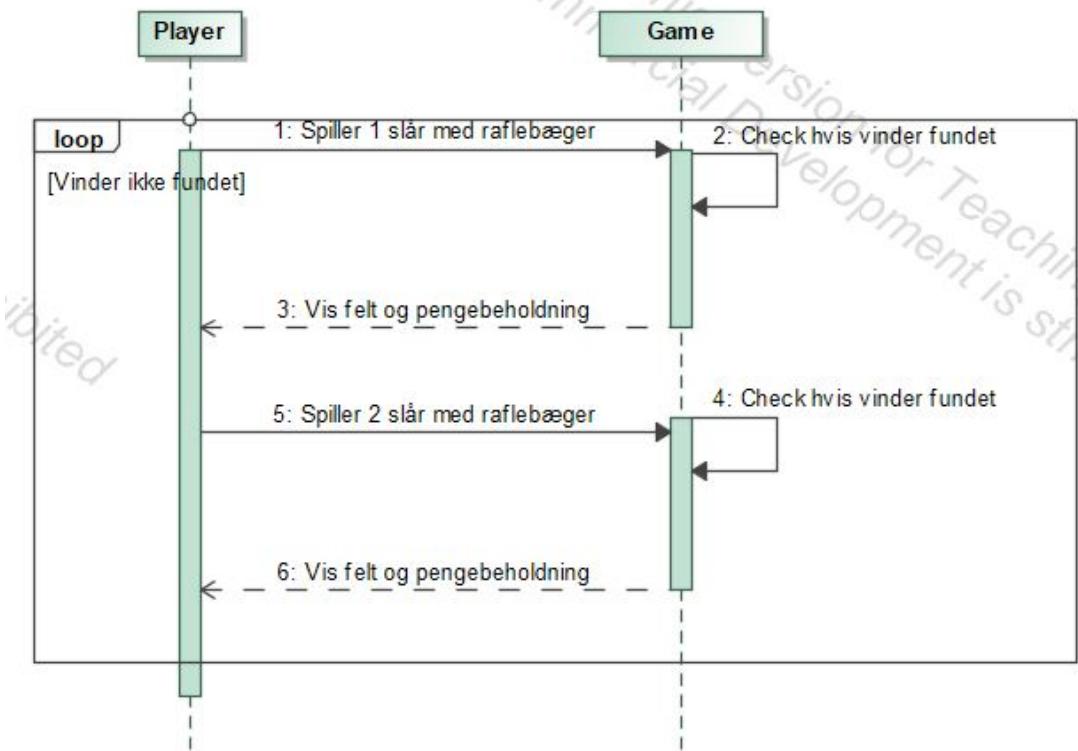
- Ved start af spillet åbnes spillet (Game). Dernæst vælges sprog og de to spillere navngives og vises en introducerende tekst med blandt andet startbalance.
- Den første tur (Turn) startes med spiller 1 (Player), som ved brug af raflebægeret (Cup) og derliggende terninger (Die) slår en talværdi. Denne talværdi bliver omdannet til et felt gennem 'Fields'-klassen, som enten adderer eller subtrahere en mængde point fra spillerens balance. En tekst tilhørende feltet udskrives samtidig.
- 'Turn' tjekker om 'player1' opfylder kriterierne for at vinde spillet (hasWon), og hvis ikke, så går turen videre til spiller 2 (player), som gennemgår samme trin som spiller 1.



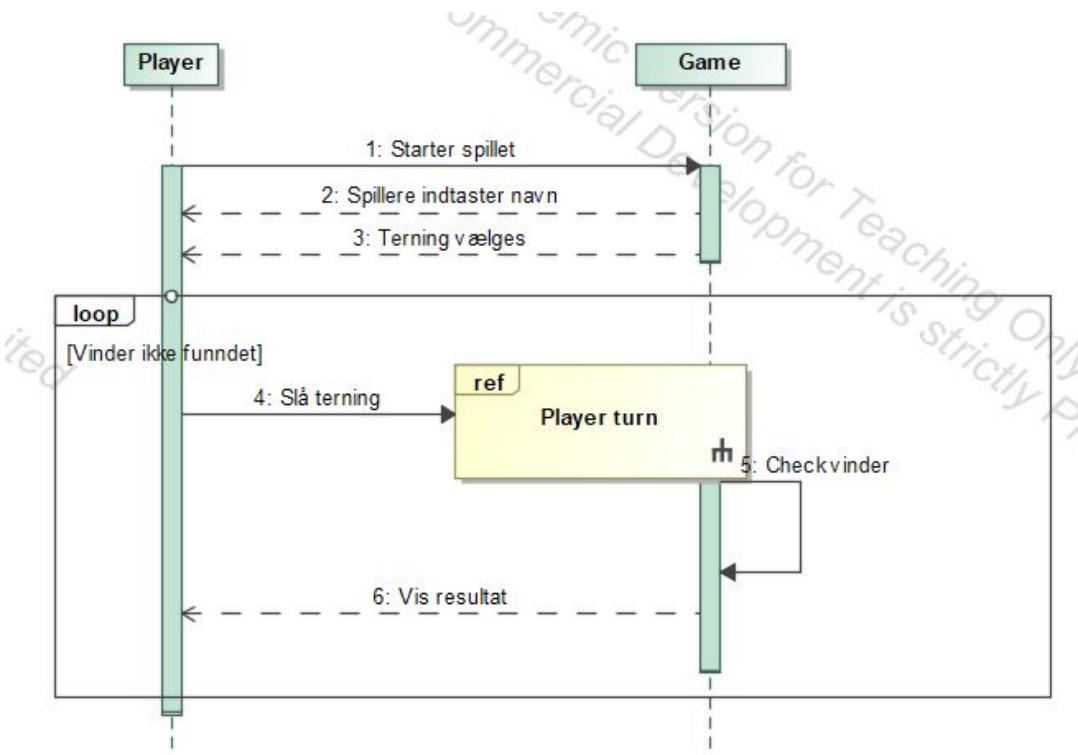
Figur 3: Domænemodel over spillet

3.4 System sekvensdiagram

Et system sekvensdiagram viser interaktionen mellem aktører og systemet under en use-case. Der bruges hverken returparametre eller metodenavne idet diagrammet er (programmerings-)sprogsuafhængigt. Det nedenstående systemsekvensdiagram giver et overblik over hele spillet, dvs. en samling af de forskellige systemsekvensdiagramer, der udgør dette terningspil.



Figur 4: Model over Player Turn



Figur 5: Model over Game

3.5 Risikoanalyse

Der er en række risici behæftet med udviklingen og brugen af spillet. Disse er ens med de i CDIO1 fundne.

Der er i udviklingen risiko for, at

1. det oprindelige design ikke faciliterer implementering af ekstraopgaver.
2. gruppens manglende versionsstyring besværliggør gruppearbejde.
3. projektledelsen og timestyringen skævvridter arbejdsbelastningen.
4. den implementerede kode ikke er kompatibel med den udleverede GUI.
5. gruppens mangel på test resulterer i fejl som ikke opdages da terningespil fundamentalt er tilfældigt.

Der er i brugen risiko for, at

1. programmets vejledning ikke er tilstrækkelig til brug.
2. fejlagtig brug kan beskade eller manipulere programmet.

Disse risici er kvantificeret i nedenstående tabel med risikofaktor 1-5 (lav til høj) og konsekvensfaktor 1-5 (lav til høj) til en samlet opmærksomhedsscore P/I.

Risiko	Riskofaktor	Konsekvens	P/I score
Implementering af ekstraopgaver	3	4	12
Versionsstyring	2	5	10
Projektledelse	1	4	4
GUI kompatibilitet	2	3	6
Fejl på grund af gruppens mangel på test	3	5	15
Utilstrækkelig vejledning	1	4	4
Fejlagtig brug	2	4	8

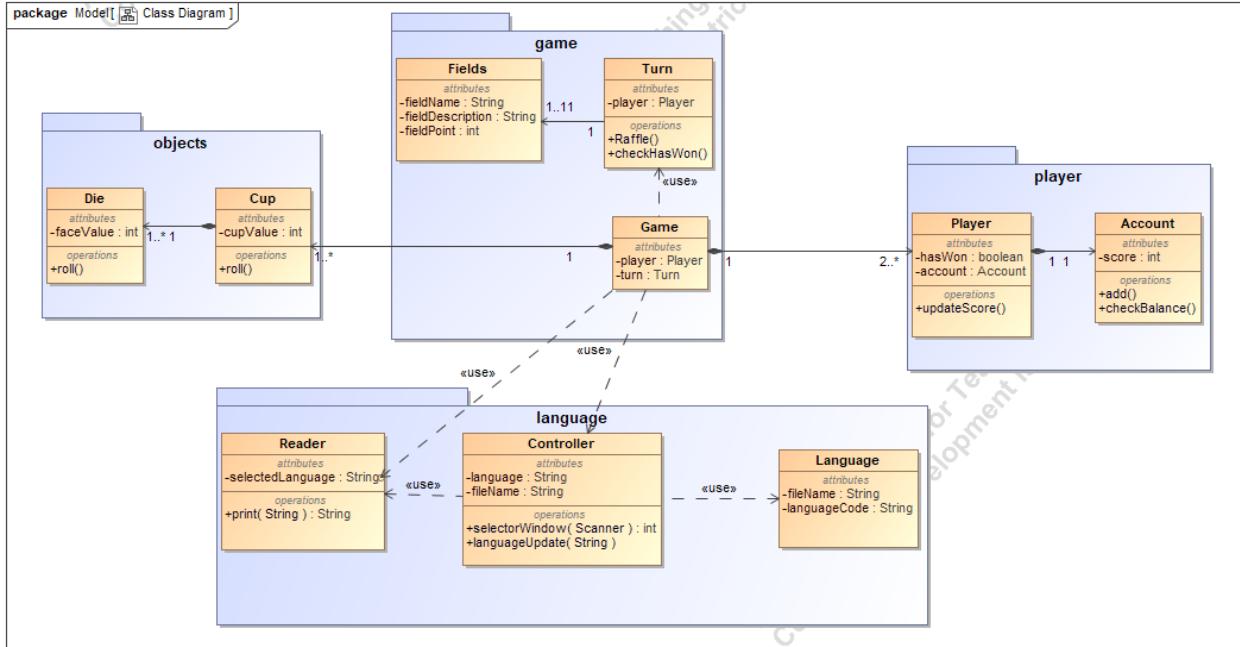
Tabel 1: Risikotabel

4 Design

For at opnå et design, der faciliterer en løsning af opgaven på en tidseffektiv og simpel vis, udarbejdes et udførligt design. Dette gøres gennem et designklassediagram, sekvensdiagramer af de mest centrale klasser, samt en beskrivelse af de anvendte GRASP-mønstre.

4.1 Designklassediagram

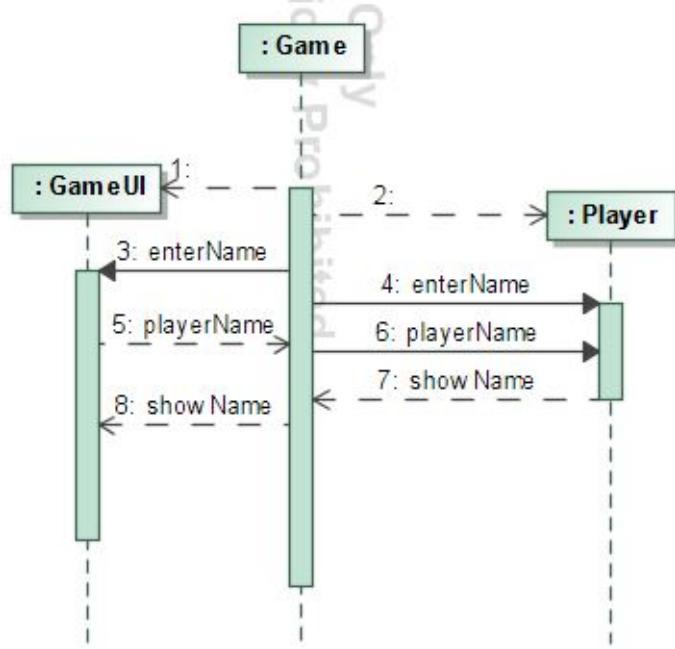
Her skitseres et designklassediagram, hvor der fastlægges nedarvning (er-en-relationerne), referencer mellem objekter (har-relationer), de vigtigste variabler og de vigtigste metoder. Designklassediagrammet har gennemgået flere iterationer, idet der undervejs viste sig nye måder at strukturere koden på end først havde skitseret. Det endelige klassediagram ses herunder.



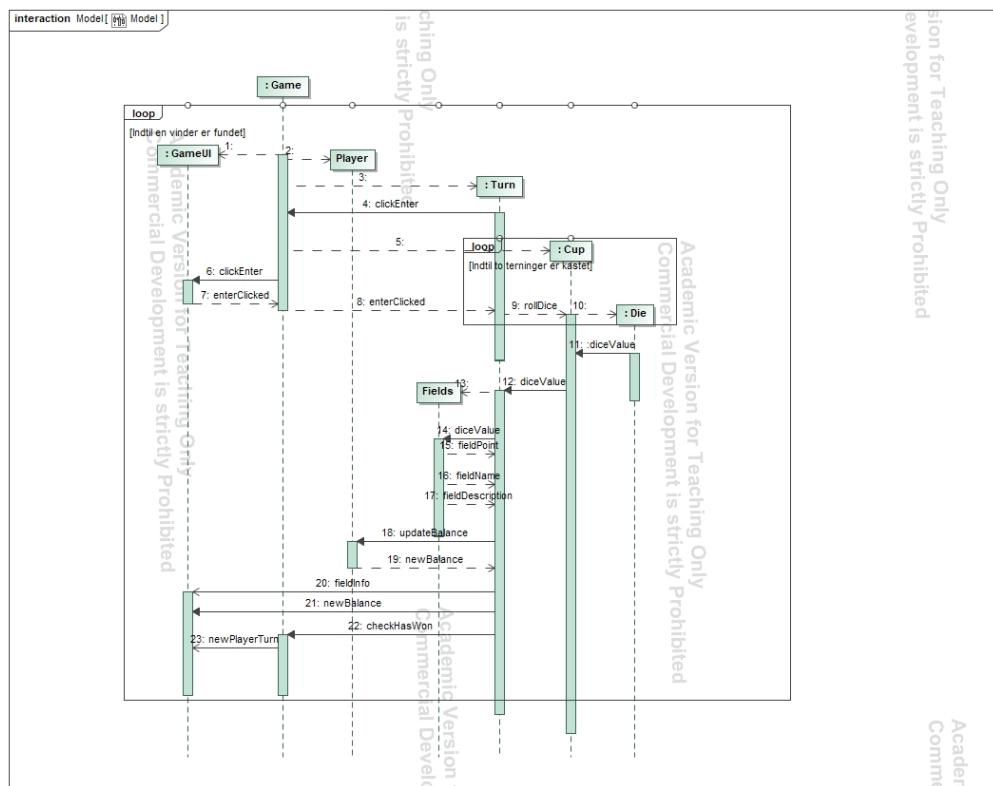
Figur 6: Design klasse diagram

4.2 Sekvensdiagram

Sekvensdiagrammer viser en udveksling af meddelelser (dvs. metodekald) mellem flere objekter, i en specifik, tidsbegrænset situation. Sekvensdiagrammer ligger særlig vægt på rækkefølgen og tiden når meddelelserne til objekter sendes. Objekter repræsenteres af lodrette stregede linjer i sekvensdiagrammer, med objektets navn øverst. Tidsakslen er også lodret, og vokser nedad, så meddelelser sendes fra et objekt til et andet i form af pile med operationer og parameternavn.



Figur 7: Sekvensdiagram Navn



Figur 8: Sekvensdiagram Spil

4.3 GRASP-mønstre

GRASP beskriver nogle retningslinjer i forhold til at fordele ansvar til klasser og objekter, altså en beskrivelse af, hvordan en problemtyppe kan løses. GRASP udgør forskellige mønstre og principper, herunder de fem mønstre, som der vil være fokus på i denne opgave.

1. **Creator:** En klasse A skal være ansvarlig for at oprette nye instanser af en anden klasse B i visse situationer. "Cup" er ansvarligt for at oprette "Die"-objekterne, idet et Cup indeholder og anvender dies.
2. **Information Expert:** Der er tildelt de forskellige klasser, der har den nødvendige information til at udføre en handling, ansvaret for denne handling. Terning-objektet har ansvar for de ting, der har med den enkelte terning at gøre f.eks. antal sider og det enkelte slag, mens "Cup" har ansvaret for at lægge værdierne sammen og få alle terningerne igennem Die.roll().
3. **Controller:** Der bruges en 'language controller', i starten af spillet til at vælge mellem Dansk eller Engelsk.
4. **Low Coupling:** Koblingen mellem klasser er et mål for, hvor afhængige klasserne er af hinanden. Lav kobling medfører, at klasser har så lille afhængighed af hinanden som muligt, hvilket øger overskueligheden af programmet, og gør det lettere at udskifte dele af programmet.
5. **High Cohesion:** Den enkelte klasse tildeles et let, forståeligt og ensartet ansvarsområde, eller eventuelt flere ansvarsområder, der er tæt relaterede til hinanden.

5 Implementering

I dette afsnit vil dele af den anvendte kode blive gennemgået og implementeringen forklaret. Overordnet er programmet opdelt i fire packages; game, language, objects og player. Hver package består af to eller flere klasser, som varetager forskellige funktionalitet for hver af de fire packages.

5.1 Game package

Game indeholder tre klasser; turn, fields og game. Denne package styrer al logik og struktur for spillet.

Turn klassen er ansvarlig for, at styre forløbet af en spillers tur. Slår med terningerne og fortæller spilleren om vedkommendes placering og handlinger, der sker på denne plads. Tildeler de respektive point og opdatere spillerens score, samt tjekker om vedkommende har vundet spillet.

Game klassen står for at styre det samlede spil. Klassen opretter det nødvendige antal spiller, giver dem deres tur (turn) og udnævner afslutningsvis vinderen af spillet.

Fields klassen bruges til at bestemme, hvilket felt spilleren er landet på, handlingen på feltet og pointene på feltet, ud fra vedkommendes slag med terningerne. I det store hele kan fields' ansvarsområde afgrænses til styring af storytelling, hvilket var et af kundens ønsker.

Håndteringen af resultatet af et terningslag er i Fields styret af en *switch*, med cases fra 2-12 (mulige udfald af øjne). Hver case indeholder navnet på feltet, beskrivelsen af handlingen på feltet og pointene på feltet.

Nedenfor vises udklip af switchen der styre beskrivelsen af handlingen på det pågældende felt.

```
public static void fieldsSwitch (int cupRoll) {  
  
    switch (cupRoll) {  
        case 2:  
            fieldName = "field2name";  
            fieldDescription = "field2description";  
            fieldPoint = (+250);  
            break;  
        case 3:  
            fieldName = "field3name";  
            fieldDescription = "field3description";  
            fieldPoint = (-100);  
            break;  
        .  
        .  
    }  
}
```

5.2 Objects package

Objects indeholder to klasser: Die og Cup. Denne package indeholder de "fysiske" objekter der er nødvendige for at spillet kan fungere, nemlig terninger og et raflebægre.

Die klassen repræsenterer en klassisk 6 sidet terning, med mindre andet defineres, og inkluderer metoden 'roll' der slår med terningen. Roll giver et tilfældigt tal fra 1 til antallet af terningens sider.

Cup klassen fungerer som raflebæger og indeholder to instanser af Die. Bægeret bliver "raflet" ved at kalde metoden cupRoll, som kalder roll metoden på begge die instanser. Summen af terningerne gemmes i cup instansen.

5.3 Player Package

Player indeholder to klasser: Player og Account. Denne Package har ansvaret for spilleren og vedkommendes score.

Player klassen styrer navnet og scoren på en bestemt spiller. Scoren er håndteret af en instans af account, som styre spillerens point. Player har yderligere en boolean, hasWon, som fortæller om spilleren har en score eller på anden vis er vinderen af spillet. Ved instansiering af en player er hasWon "false", men hvis spilleren opfylder betingelserne for at være en vinder af spillet, bliver hasWon sat til *true*. Player har to metoder, resetScore og updateScore, som henholdsvis sætter spillerens balance til 0 eller updaterer spillerens account.

Account klassen indeholder en talværdi i en *int*, hvilket i dette tilfælde bruges til at håndtere point. Opdateringen af point kaldes som metode i account, som også sørger for at spillerens score ikke

kan komme i minus.

5.4 Language package

Language indeholder tre klasser: Language, Controller og Reader. Denne package står for alt der er relateret til styringen af spilsproget, som er det sprog, som spilleren vælger at spilleret skal foregå på.

Language klassen består af en metode, languageSwitch, som bruges til at bestemme en sprogkode fx. DK eller ENG og den dertilhørende sprogfil fx. DK.txt og ENG.txt.

Reader klassen aflæser spillets sprogfiler og håndterer alt oversættelse fra tekstlabels til ren tekst. Spillet er indledningsvist indstillet til at være engelsk. Reader har metoden converter, som bruges til at konverterer labels om til tilhørende tekst. FileReader indlæser sprogfilen, der matcher det valgte sprog og BufferedReader bruges som midlertidig hukommelse for teksten. Gennem et while loop findes et tekstlabel og tildeler teksten på linjen under tekstlabellet til String result.

```
public static String converter(String field) {  
  
    String result = null;  
  
    try {  
        FileReader fr = new FileReader( "src\\main\\resources\\" +  
            selectedLanguage);  
        BufferedReader br = new BufferedReader(fr);  
  
        String str;  
        while ((str = br.readLine()) != null) {  
  
            if (str.equals(field)) {  
                result = br.readLine();  
            }  
        }  
    } catch (IOException e) {  
  
        System.out.println("fileError");  
    }  
    return result;  
}
```

Controller klassen benyttes til bestemmelsen og håndteringen af spillets sprog. Metoden selectorWindow præsentere brugeren for de sprog, som spillet kan fungere på og modtager brugens valg (gemt i int input). Brugens input omsættes ved hjælp af ovennævnte languageSwitch til de korrekte værdier og sætter ved hjælp fra Reader klassen sprogfilen der bruges i programmet. Afslutningsvis udskrives en linje, som beskriver hvilket sprog der er valgt.

5.5 Language file

Spillet er udstyret med tre sprogfiler, to som er reelle sprog, dansk og engelsk og en som er skabelon, der kan kan bruges til nye sprog.

Sprogfilen er en .txt, som består af tekstlabels og tekst. Et eksempel på dette ses nedenfor.

```
---- LANGUAGE ----
* FileName: ENG.txt
* Language: English

---- operations ---
selectLanguage
Select Language:
selectedLanguage
Your selected:
ENG
English
DK
Danish
languageError
The requested language is not support
fileError
Error: No language.txt file found
```

Ved at benytte Reader.print("tekstlabel") finder Reader.printer teksten til det tilhørende "tekstlabel" og indsætter i steder teksten der tilhører labellet. fx "player1" er et tekstlabel og næste linje er den tekst, som programmet vil udskrive, som i dette tilfælde er "player 1:". "enterName" bliver oversat til "Enter name for" og "hiPlayer" bliver til ", welcome to DutterLand. You have been granted a total of", og så fremdeles.

6 Verifikation

Dette afsnit beskriver diverse tests, hvordan de er blevet udført og hvordan de tilfredsstiller kravene.

6.1 Test

Testene er alle Junit tests, som kan køres automatisk og uden input. Der er 4 filer som indholder testene henholdsvis "CupTest", "DieTest", "AccountTest" og "PlayerTest". I næste afsnit gennemgås testen for at en spillers balance ikke kan blive negativ.

6.2 Account test

Måden der er sikret, at accounten ikke kan blive negativ er gennem player metoden, updateScore. Dette gøres ved, at der kontrolleres om balancen og det nye beløb ville blive en negativ værdi, i hvilket tilfælde at Accounten bliver genstartet med en score på nul.

```

    @Test
    public void noNegative() {

        Player player = new Player("jens");

        player.getAccount().setBalance(0);
        player.updateScore(-250);

        assertEquals(0, player.getAccount().getBalance());
    }

```

- 1 - En spiller oprettes.
- 2 - Spilleren account balance sættes til 0.
- 3 - Metoden updateScore "tilføjer-250 points til spilleren.
- 4 - Der checkes om spillerens account balance stadig er 0.

7 Projektforløb

Opgavens grundlæggende struktur følger Unified Process, jf. figur 1. Beskrivelsen i følgende afsnit baseres hovedsagligt på tidligere afleveret, da processen i høj grad er ens med den tidligere anvendte.

Inception fasen er udført forud for projektets start idet opgaven og dennes business case er pre-defineret.

Elaboration fasen indebærer analyse af den stillede opgave og kravsspecifikationen gennem use-case og risiko-analyse. Derefter udføres design af softwaren gennem en domænemodel, et systemsekvensdiagram, et sekvensdiagram og et klassediagram. Elaboration fasen fortsætter sideløbende med implementeringen, da designet påvirkes af erfaringer gjort i udviklingen.

Implementeringen af opgaven starter tidligt i forløbet, da det er bekendt at denne er en iterativ proces og implementering af én funktion muligvis leder til at designet af en anden må ændres. Under implementeringen arbejder gruppen intensivt med versionsstyring og test. Disse sikrer, at alle gruppens medlemmer kan arbejde sideløbende. Dette faciliteres gennem oprettelsen af et repository, hvor der primært arbejdes ud fra en development branch og udelukkende pushes til master branch når der er en ren, fungerende og sammenhængende kode. Gruppens medlemmer arbejder på hver deres 'function'-gren ud fra development grenen.

I forhold til sidste opgave var det tydeligt, at erfaringerne gjort tidligere hjalp til at opnå en bedre projektstruktur tidligt. Projektet bærer desuden præg af, at det er en iteration af det tidligere udviklede program og domæne strukturen derfor er forbedret mellem programmerne.

Transitionsfasen er i dette projektforløb meget kort, da udrulning i opgavens tilfælde betyder aflevering af opgaven på DTU Inside.

8 Konklusion

Det konkluderes kortfattet, at det er lykkedes at løse den stillede opgave ved at udvikle og teste et program for et 'brætspil' med terninger mellem to personer.