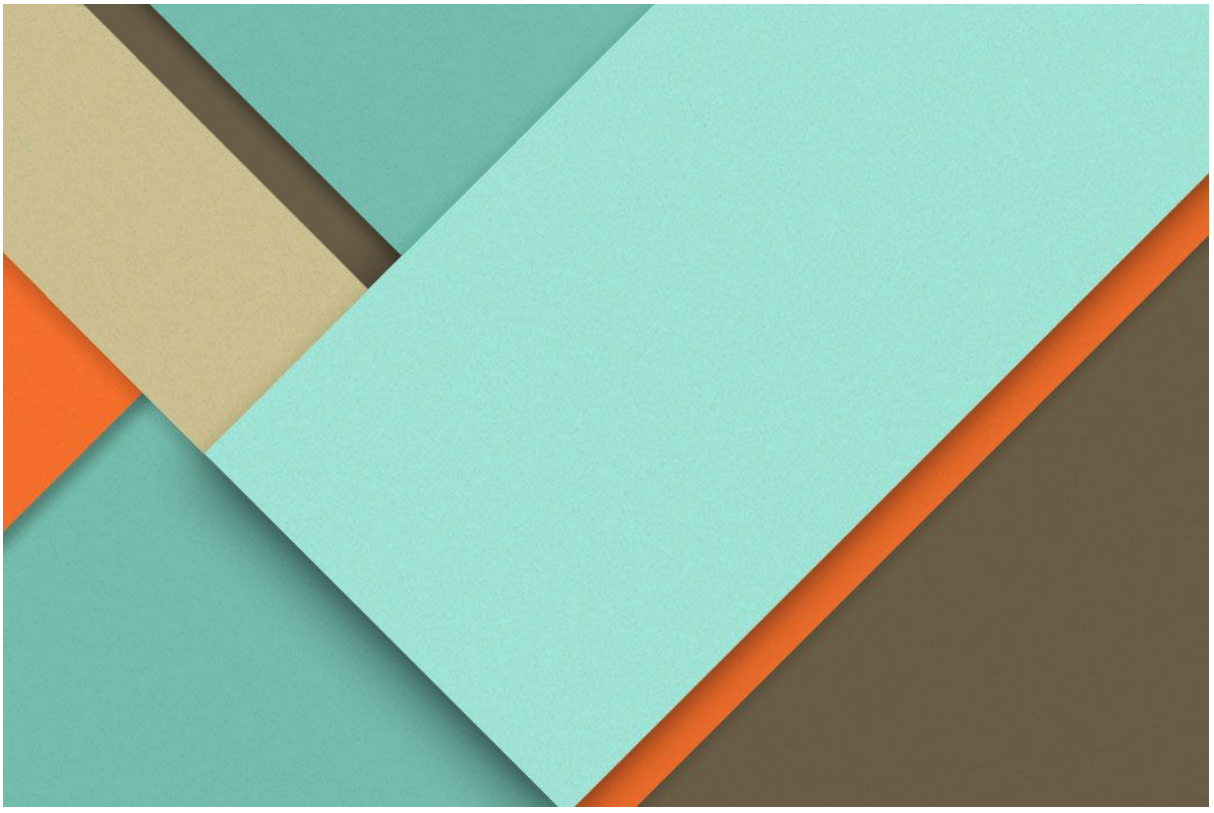


# CDIO delopgave 2

Kursus nr. 02313 - 02314 - 02315


Alexander Kjeldsen (s165477)

Ayat Abdel-Hadi Toma (s144245)

Elísa Kristín Sverrisdóttir (s175115)

Josephine Mellin (s163313)

Mads Jørgensen (s175184)

Oliver Uldall Schultz (s175113)


Gruppenr. 15

Afleveringsdato: 10. november 2017

# Time regnskab

| Deltager | Design | Impl. | Test | Dok. | Andet | I ALT |
|---|---|---|---|---|---|---|
| *Alexander* | 0 | 7 | 1 | 2 | 3 | 13 |
| *Ayat* | 0 | 0 | 0 | 7 | 1 | 8 |
| *Elísa* | 0 | 0 | 0 | 7 | 2 | 10 |
| *Josephine* | 1 | 1 | 0 | 7 | 2 | 12 |
| *Mads* | 2 | 7 | 2 | 1 | 1 | 13 |
| *Oliver* | 6 | 3 | 1 | 2 | 0 | 12 |
| **Sum** | | | | | | **68** |

## Abstract

Denne rapport omhandler 2. del af CDIO projektet. Opgaven handler om at lave et spil mellem to personer der kaster to terninger og lande på felt fra 2-12, hvert felt har enten en positiv eller negativ effekt på spillernes point beholdning. Der udvikles et Java projekt og der er gjort brug af GUI (Graphical User Interface). Vi har dokumenteret kundens krav ved brug af FURPS og beskrivelser af programmet med forskellige diagrammer, som er nemt forståelige, selv om man ikke har kendskab til programmeringssprog. Rapporten indeholder også beskrivelser af syntax, testning og en detaljeret projektplanlægning, hvor vi har gjort brug af Unified Process. Vi har et færdigt produkt, som opfylder kundens ønsker, som vi herved har illustreret med relevante artifacts som vedlægges i rapporten.

# Indholdsfortegnelse

**Time regnskab** 2

**Abstract** 3

**1.Indledning** 5

**2. Analyse** 5

**4. Implementering** 15

**5. Test** 16

**6. Projektplanlægning** 16

**7. Konklusion** 17

**8. Litteratur og kildeliste** 17

# 1. Indledning

I denne opgave skal der udvikles et spil, hvor man denne gang kaster med en terning og lander på et felt der har en positiv eller negativ resultat. Dermed får man et nyt pengebeholdning, afhængig af det installeret felt.

Som udgangspunkt klassificeres kravene som kunden stiller op gennem FURPS teknikken. Derudover gøre vi brug af UML diagrammer til at beskrive detaljerne i spillet og, hvordan det fungerer inde i programmet.

Koden vil også blive beskrevet undervejs og hvad den indeholder og hvordan den vil fungere i sidste resultat.

# 2. Analyse

**2.1 Klassificerede Krav - FURPS:**
Vi har valgt at bruge FURPS for at beskrive kundens krav, hvori der bådes beskrives de funktionelle krav, F eller 'Functional requirements', men videre også *ikke*-funktionelle krav, som indebærer U, R, P og S, som henholdvis står for 'Usability', 'Reliability', 'Performance' og 'Sustainability'[1].

**- Functional:**
**F1:** Vores kunde ønsker et spil, der skal kunne spilles af to person på DTU's databarer uden forsinkelser.
**F2:** Kunden efterspørger et program, hvori der indgår to terninger.
**F3:** Spillet skal indeholde felter fra 2-12, som skal have en negativ eller positiv effekt på dets spilleres pengebeholdning.
**F4:** Når en spiller lander på et felt, skal der udskrives en meddelelse som svarer til feltets effekt på spillerens pengebeholdning.

1. Tower +250 "250 guldmønter falder ud af tårnet og lander lige foran dig".
2. Crater -100 "Du falder ned og brækker dit ben - det koster dig 100 af få gips på".
3. Palace gates +100 "Du møder Bill Gates fætter, Palace Gates, som giver dig 100 guldmønter"
4. Cold Desert -20 "Du fryser og skal bruge 20 guldmønter på at købe et tæppe"
5. Walled city +180 "Du klatrer over muren og modtager 180 guldmønter"
6. Monastery 0 "Du bruger en tur på at meditere med nogle munke, du tjener ikke noget, men du er i et med universet"
7. Black cave -70 "Du farer vild og bruger 70 guldmønter på en GPS"
8. Huts in the mountain +60 "Du finder en hemmelig gruppe i bjergene og modtager 60 guldmønter"
9. The Werewall (werewolf-wall)-80, men spilleren får en ekstra tur. "Du bruger 80 guldmønter på billetter til Varulvevæggen, undervejs bliver du selv til en varulv og får en ekstra tur"

[1] Larman, 2005, side 57

10. The pit -50 “Du er i pitten til en Justice koncert og brækker benet, du bruger 50 guldmønter på vodka for at dulme smerten”
11. Goldmine +650 “Du stjal noget guld med og tjente 650 guldmønter”

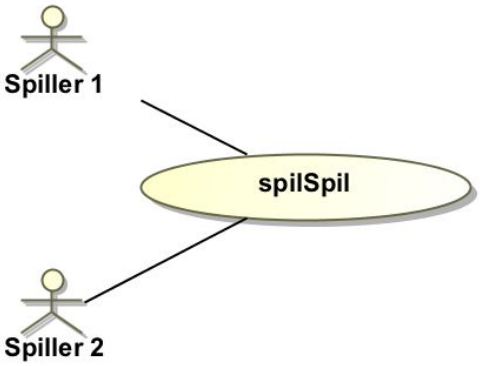
**F5:** Lander en spiller på felt nummer 10 gives en ekstra tur med spillets to terninger.
**F6:** Ved spillets start skal begge spillere have en pengebeholdning på 1000.
**F7:** Spillet slutter når en spiller når en pengebeholdning på 3000.
**F8:** Spillerenes pengebeholdning må ikke komme under 0.
**F9:** Programmet skal udprinte diverse meddelelser alt efter hvilket felt, en spiller lander på.
**- Usability:**
**U1:** En spiller skal med det samme kunne se, hvad han har slået.
**U2:** En spiller skal altid kunne se sin pengebeholdning.
**U3:** GUI’en skal være nem, overskuelig og responsiv.
**U4:** Hver spiller skal have en brik, der viser, hvor spilleren står på brættet.
**U5:** Spillerene skal selv kunne vælge, hvilke navne der bliver brugt i spillet.
**U6:** Når spillet er færdigt, skal spillerne få at vide, hvem der har vundet.
**U7:** Spillets regler skal fra starten være klart for spillerene, uden behov for en manual.
**U8:** Spillerne kan få information om felterne ved at bevæge musen hen over dem.
**U9:** Spiller 1 starter spillet.
**U10:** De meddelelser, der indgår i programmet, skal nemt kunne oversættes, af kunden, til andre sprog.
**- Reliability:**
**R1:** Kunden har ikke angivet nogen specifikke krav ift. responstiden af terningerne ved hvert slag. Vi har dog valgt at kravet til terningernes responstid i forbindelse med visning af deres værdi, skal ligge på 333 millisekunder.
**R2:** Det forventes videre, at spillet afsluttes, når den ene af spillerne har indsamlet en pengebeholdning på 3000.
**- Performance:**
**P1:** Den samlede værdi af terningernes øjne skal vises senest efter 333 millisekunder.
**P2:** Videre skal summen af terningernes værdi være korrekt.
**P3:** De meddelelser, som spillet skal give ift. de tilsvarende felter, skal ligeledes gives senest efter 333 millisekunder.
**- Supportability**
**S1:** Spillet skal kunne spilles på DTU’s databarer, som kører windows 10.

## 2.2 UML

### 2.2.1 Use case diagram:

Et use case diagram giver et godt billede af systemets kontekst, det viser forholdet mellem en use case og dens aktører. Som i dette tilfælde er spilSpil (use case) og Spiller 1 (primær aktør) og Spiller 2 (primær aktør). Det foretrækkes at use case diagrammet er simpelt og at man bruger use case beskrivelsen til at gå i detaljer.[2]

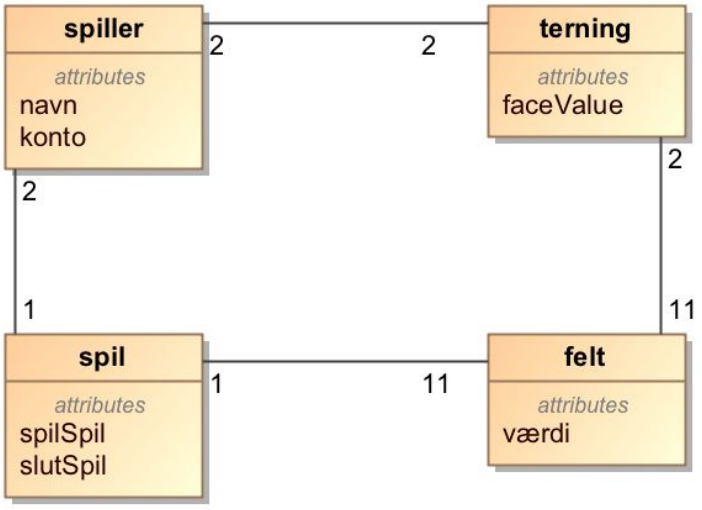
[2] Larman, 2005, side 158




**Figur 1**

### 2.2.2 Domæne model:

En domæne model (domain model) illustrerer vigtige begreber i et domæne.
For at nedbryde og overskueliggøre domænet må man identificere koncepterne, attributterne og forbindelser der er relevante. Her gives der blot et billede af koncepterne i "den virkelige verden" og ikke af selve software objekterne[3]. Dog kan den kan være en nyttig inspirationskilde for software designet og navngivning af klasser i design fasen. Vi har valgt objekterne spiller, terning, spil og felt til vores domæne model[4].



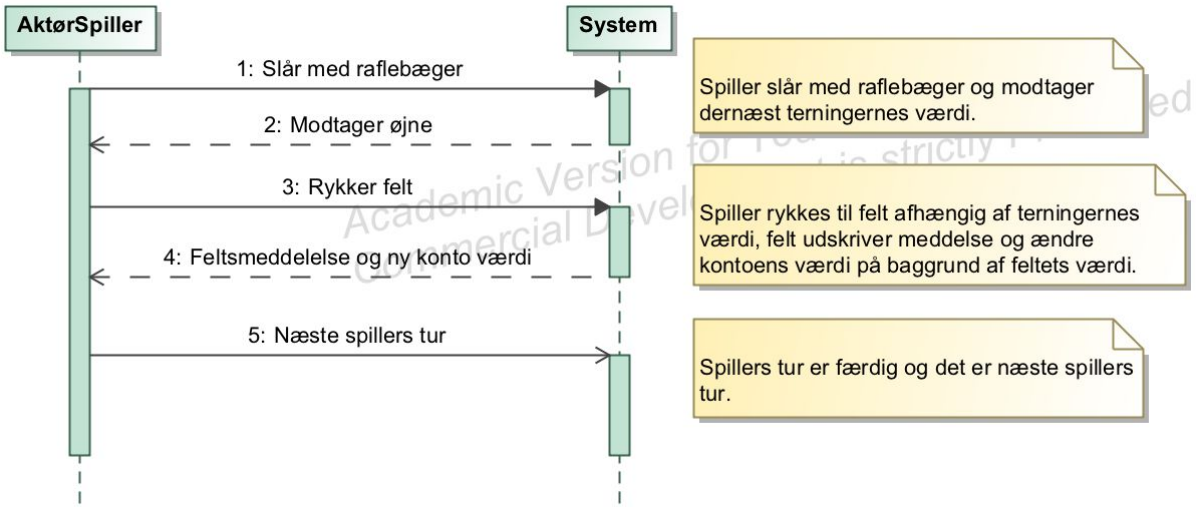

**Figur 2**

### 2.2.3 Sekvens Diagram:

Sequence diagrams eller et sekvens diagram er anvendeligt, når man ønsker at vise, de beskeder, der kæder softwarens objekter.
Altså skal sekvens diagrammet fremstår som et diagram over interaktionen i programmet, hvis styrker er at give et klart billede over en kronologisk liste over beskeder i programmet og videre har en simpel notation[5].

[3] Larman, 2001, side 8
[4] Larman, 2005, side 219-220
[5] Larman, 2001, side 118 + 198

Dette sekvens diagram viser hvordan Aktørspilleren interagerer med systemet, det udtrykkes, hvad der sker efter hvert skridt i spillet.




**Figur 3**

### 2.2.4 Use Case Specification:

For at beskrive vores use case, gør vi brug af en brief, casual og fully dressed description.
Vores brief description giver et overblik over det ønskede succes forløb af vores program.
Vores causal description afdækker forskellige scenarier for programmet, som en tilføjelse til programmet simple struktur - eksempelvis at en spiller gives ekstra tur, når der slås 10 og at deres pengebeholdning ikke kan komme under nul.
I vores fully dressed description, som er den mest udførlige og afdækkende for programmet, beskrives bl.a. De alternative flows, der gives af basic flowet. Eksempel basic flowets punkt 5, som har mange alternative flows, som bl.a. Fremgår som punkt 5a - dog her med beskrivelse af, hvordan et alternativ forløb kan foregå[6].

| **Use Case Specification:SpilSpil** |
|---|
| **ID:** 1 |
| **Brief description:**<br>Spillet kræver 2 spillere, spilleren slår terningerne, rykker og opnår en ny værdi afhængig af feltet. Summen beregnes og så er det den næste spillers tur. |
| **Primary actors:**<br>Spilleren |

[6] Larmann, 2001, side 49-50.

| |
|---|
| **Secondary actors:**<br>IOOuterActive. |
| **Preconditions:**<br>2 spillere og spilleren har ikke opnået 3000<br>De to spillere gives begge 1000 point fra start af. |
| |
| **Main flow:**<br>1. Spiller slår to terninger.<br>2. Terningen viser en værdi mellem 2-12.<br>3. Spillerens konto opdateres afhængig af feltets værdi.<br>4. Næste spillers tur.<br>5. Spillet slutter når en af spillerne opnår 3000. |
| **Postconditions:**<br>Spillet er færdig, når en spiller opnår 3000point. |
| **Alternative flows:**<br>1. Hvis en spiller slår 10, får man en ekstra tur. |

**Brief:**

**Use Case ID 1: SpilSpil:**

De to spillere, Player1 eller/og Player2 starter hver især med en pengebeholdning på 1000. De to spillere skiftes om at kaste med spillets to terninger via et raflebæger. Summen af terningerne vises, og spilleren må herefter rykke samme antal. Spillepladens felter kan både have en positiv og negativ effekt på spillernes pengebeholdning, som påvirkes ved at blive fratrukket eller tillagt point. Spillet spilles indtil at enten Spiller1 eller Spiller2 har indsamlet 3000 point i alt.

**Casual:**

**Use Case ID 1: SpilSpil:**

- De primære aktører (primary actors) for spillet Player1 eller Player2 begynder spillet ved at indtaster deres navne, hvor de herefter kan starte spillet ved at trykke på knappen 'roll'.
- Hver spiller, Player1 og Player2, starter med en pengebeholdning på 1000 hver.
- Der slås med to terninger via et raflebæger, og de to terningers samlede værdi vises, hvorefter den givne spiller må rykke til det felt på spillepladen, som det samme nummer, som terningernes samlede sum.
- Det er ikke muligt for spillerne at have en pengebeholdning på under nul.

- Lander den givne spiller på et felt, kan det have en negativ effekt på dets pengebeholdning, hvorved der fratrækkes en bestemt værdi fra dets pengebeholdning.
- Landet man derimod på et "positivt" felt tillægges en pointsamling til den enkeltes spillers pengebeholdning.
- Hver spiller gives en meddelelse
- Lander en spiller på felt nummer 10, gives der et ekstra slag med de to terninger.
- Spillet fortsætter indtil at enten Player1 eller Player2 har indsamlet i alt 3000 point.

**Fully dressed:**
**Use Case ID 1: SpilSpil:**
**Scope:** Felt Spil
**Level:** User goal
**Primary actors:** Spillets to spillere
**Secondary actor:** IOOuterActive
**Stakeholders and interests:**

- Vores sekundære aktør (secondary actor), IOOuterActives, ønsker et program, der skal indeholde en spilleplade, to terninger, et raflebæger.
- Kundens ønske til terningespillet er, at det skal kunne køre uden forsinkelser og returnere og beregne forskellige værdier i spillet rigtigt.
- IOOuterActives ønske til spillet er, at det skal kunne regne terningeværdier sammen, men også lægge til og trække fra de felter, som hver spiller lander på.
- IOOuterActives ønsker at spillet skal kunne spilles af to personer.
- Spiller1 (primary actor) ønsker at spille terningespillet og vinde.
- Spiller2 (primary actor) ønsker at spille terningespillet og vinde.
- Kunden ønsker at spillet skal indeholde felter fra 2-12.
- Disse felter skal videre kunne påvirke spille i en positiv og negativ retning, som enten trækker point fra eller lægger point til for hver af spillernes pengebeholdninger.
- Spiller1 ønsker at kunne trykke på en knap, for at de to terninger bliver slået.
- Spiller2 ønsker at kunne trykke på en knap, for at de to terninger bliver slået.
- IOOuterActive ønsker, at spillet lader dets spillere skiftes om at slå med terningerne af flere omgange.
- Player1 ønsker at kunne slå terningen af flere omgange.
- Player2 ønsker at kunne slå terningen af flere omgange.
- IOOuterActive ønsker, at programmet fortæller, forskellige meddelelser afhængigt af, hvilket felt en given spiller rammer.
- Kunden ønsker at spillet slutter, når enten Spiller1 eller Spiller2 har indsamlet en pengebeholdning på 3000.

**Preconditions:** Spillet er færdigt, når enten Spiller1 eller Spiller2 har indsamlet en pengebeholdning på 3000.
Både Spiller1 og Spiller2 starter med en pengebeholdning på 1000.
**Success guarantee (or Postconditions):** Spillet spilles indtil en vinder er fundet.
**Main Success Scenario (or Basic Flows):**

1. Spiller1 og Spiller2 starter programmet.
2. Spiller1 og Spiller2 indtaster hver især sit navn.
3. Spiller1 starter spillet.

4. Spiller1 'roll'-knappen og systemet responderer ved at vise summen af de to terningers værdi.
5. Systemet rykker Spiller1's brik til det felt, der er lignende summen er terningernes værdi.
6. Systemet aflæser feltets pointscore og udprinter en meddelelse til at tilhørende feltet.
7. Systemet lægger enten til eller trækker feltets pointscore til Spiller1's pengebeholdning.
8. Herefter er det Spiller2's tur, hvor samme spilbetingelser finder sted.

Spiller1 og Spiller2 repeterer punkt 1-8 indtil:

9. Spiller1 eller Spiller2 opnår 3000 point

**Extensions (or Alternative Flows):**

4a. For hver gang en Spiller1 eller Spiller2 slår:

1. Systemet responderer på valg af 'Roll''-knappen og returnerer terningernes værdi.
2. Viser terningernes sum 10, må den givne spiller slå igen.

5a. For hver gang en Spiller1 eller Spiller2 rammer felt nummer 2 kaldet 'Crater':

3. Trækkes der 100 point fra den givne spillers pengebeholdning.
4. Responderer systemet med meddelelsen "Du falder ned og brækker dit ben - det koster dig 100 af få gips på".

5b For hver gang en Spiller1 eller Spiller2 rammer felt nummer 3 kaldet 'Palace gates':

5. Belønnes den givne spiller 100 point.
6. Udprinter systemet meddelelsen "Du møder Bill Gates fætter, Palace Gates, som giver dig 100 guldmønter".

5c. For hver gang en Spiller1 eller Spiller2 rammer felt nummer 4 kaldet 'Cold Desert':

7. Trækker 20 point fra den givne spillers pengebeholdning.
8. Udprinter systemet meddelelsen "Du fryser og skal bruge 20 guldmønter på at købe et tæppe".

5d: For hver gang Spiller1 eller Spiller2 rammer felt nummer 5 kaldet 'Walled city':

9. Belønnes den givne spiller med 180 point.
10. Udprinter systemet meddelelsen "Du klatrer over muren og modtager 180 guldmønter".

5e: For hver gang Spiller1 eller Spiller2 rammer felt nummer 6 kaldet 'Monastery':

11. Trækkes der hverken point fra eller lægges point til.
12. Udprinter systemet meddelelsen "Du bruger en tur på at meditere med nogle munke, du tjener ikke noget, men du er i et med universet".

5f: For hver gang Spiller1 eller Spiller2 rammer felt nummer 8 kaldet 'Black cave':

13. Trækkes der 70 point fra den givne spillers pengebeholdning.
14. Udprinter systemet meddelelsen "Du farer vild og bruger 70 guldmønter på en GPS".

5g: For hver gang Spiller1 eller Spiller2 rammer felt nummer 9 kaldet 'Huts in the mountain'':

15. Belønnes den givne spiller der 60 point til sin pengebeholdning.
16. Udprinter systemet meddelelsen "Du finder en hemmelig gruppe i bjergene og modtager 60 guldmønter".

5h: For hver gang Spiller1 eller Spiller2 rammer felt nummer 10 kaldet 'The Werewall':

17. Fratrækkes 80 point fra den givne spillers pengebeholdning.
18. Gives den givne spiller en ekstra tur.
19. Udprinter systemet meddelelsen "Du bruger 80 guldmønter på billetter til Varulvevæggen, undervejs bliver du selv til en varulv og får en ekstra tur".

5i: For hver gang Spiller1 eller Spiller2 rammer felt nummer 11 kaldet 'The pit':

20. Fratrækkes 50 point fra den givne spillers pengebeholdning.
21. Udprinter systemet meddelelsen "Du er i pitten til en Justice koncert og brækker benet, du bruger 50 guldmønter på vodka for at dulme smerten".

5j: For hver gang Spiller1 eller Spiller2 rammer felt nummer 11 kaldet 'The pit':

22. Belønnes den givne spiller med 650 point.
23. Udprinter systemet meddelelsen "Goldmine +650 "Du stjal noget guld med og tjente 650 guldmønter".

**Special Requirements:**

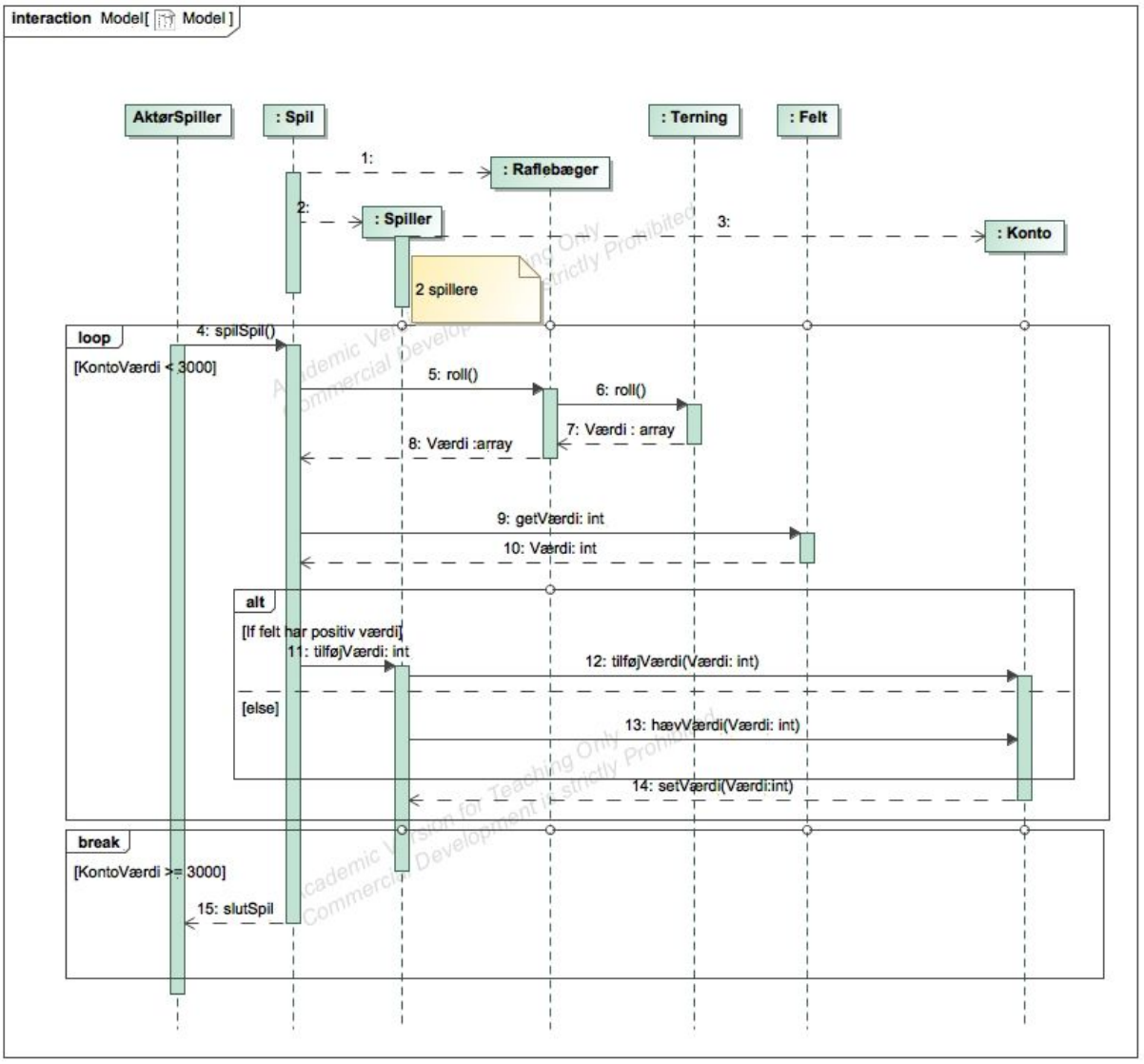
- Programmet skal efter kundens ønske fungerer uden bemærkelsesværdige forsinkelser.
- Dette har vi hertil valgt at konkretisere ved at programmets responstid ikke må gå over 333 millisekunder.
- Programmet skal være visuelt tilgængelig for spillerne, samt at spillerne skal kunne interagere programmets funktioner via GUI (graphical user interface).
- En pengebeholdning må ikke kunne blive negativ.
- Skal kunne spilles på DTU's databarer i Windows 10.
- Spillets skriftudtryk ift. Meddelelser skal nemt kunne oversættes til andre sprog.

**Technology and Data Variations List:** Java, Eclipse 8, GUI & JUnit testing, EclEmma og Windows 10.

# 3. Design

### 3.1 System Sequence Diagram:

For at visualisere vores programs in- og outputs har vi gjort brug af modellen System Sequence Diagram (SSD).
Der demonstreres en specifik hændelse, der udføres af den eksterne aktør, som interagerer med softwaren for vores system, og videre det system, som aktøreren igangsætter.
Dét, som aktøren igangsætter er systemet hændelser (events), som leder til en system operation, som ligeledes kan håndterer disse hændelser (events). [7]
Vores model er derfor lavet for at give et overblik over vores system:

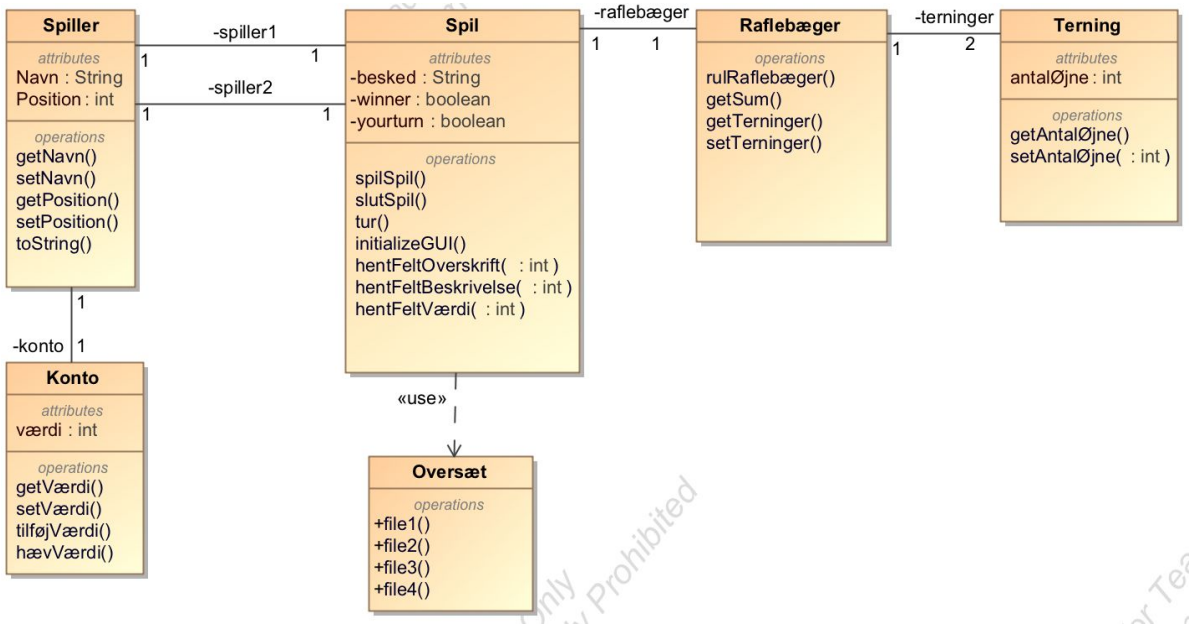



**Figur 4**

[7] Larmann, 2005, side 173, 175, 176

## 3.2 Design klasse diagram

Vores Design klasse diagram beskriver vores main metode spil og vores klasser. Som det kan ses på nedenstående figur, fortæller vores attributter hvad hvor klasser indeholder af variable, dernæst fortæller operationerne om hvilke metoder vi gør brug af i klasserne. Konto klassen kunne været i Spil klassen, men vi har knyttet Konto klassen til Spiller klassen, da vi synes der er sammenhæng i forhold til design og implementering.




**Figur 5**

### 3.3 GRASP (General Responsibility Assignment Software Patterns)

- Creator: *Spil → Felter*
- 
- Controller: Spiller trykker på en knap som er en del af UI layer. UI software skal så delegere opgaven videre til *Spil* (som er i domæne laget) som så får spillet til at starte. *Spil* er dermed vores Controller og styrer, hvad der sker senere.
- 
- Høj binding (High Cohesion): Samhørighed (binding) mellem objekter skal være høj og dermed støtte lav kobling. Objekterne skal være fokuserede og nemt forståelige.

- Information Expert: Vores Information Expert er *Spil,* fordi det har alle nødvendige informationer om *Felter*.

- Lav kobling (Low Coupling): Når vi skal ændre på noget i programmet, skal det helst ikke have en stor effekt på mange forskellige objekter. Hvis vores kobling er lav er det muligt at ændre i programmet et sted uden, at risikere at skulle ændre mange steder. Det sparer tid og arbejdsindsats og reducerer risikoen for defekter.[8] (CL s 429). Vi giver vores Information Expert *Spil* ansvaret og kobler *Spiller, Terning* og *Felt* ved den. (Se domæne model)

[8]Larman 2005, side 429

# 4. Implementering

Da vi startede med at programmere, tog vi udgangspunkt i de klasser, som vi havde lavet i vores Design diagrammer. Vi genbrugte dele af Terning og Spiller klasserne for at spare tid. Herefter skrev vi de metoder, som vi følte ville blive relevante. Vi valgte, at hver spiller skulle have en konto, som stod i sin egen klasse, og som ville blive oprettet, når man lavede et nyt Spiller-objekt.
Vores Terning-klasse blev ændret så man selv kunne vælge, hvor mange sider terningen skulle have.
Vi lavede en Raflebæger klasse som returnerede et array af seks-sidede terninger. Selvom vi som udgangspunkt altid kun skulle bruge 2 terninger til dette spil, kan man i raflebægeret selv vælge, hvor mange terninger raflebægeret skulle indeholde. Dette gjorde vi, så man kunne genbruge raflebægeret til eventuelle fremtidige projekter.

I vores main klasse har vi to metoder der styrer spillet, tur og slutSpil; slutSpil styrer, hvad der skal ske for hver spiller når de slår med terningen og slutSpil finder en vinder når en og spillerne når 3000 point.

Vores main klasse starter med at oprette relevante objekter og variable som bruges under spillet, herunder begge spillere, et raflebæger og en boolean som bliver sand, når der findes en vinder.

Vi gjorde stor brug af den lånte GUIs metoder til at visualisere spillerens position på pladen, samt at vise terningerne. GUI'en bliver også brugt til at vise beskeder til spilleren, der er relavante for spillets gang.

Vi har skrevet vores "String" beskeder i en txt fil, som bliver indlæst og sorteret i et array, så de kan blive læst af programmet og sættes ind i programmet, som beskeder. Dette er gjort, så det er nemmere at oversætte, hvis det skulle være tilfældet. Og så behøver man ikke at "recompile" programmet. Hvis programmet skal oversættes, er det derfor kun de 4 filer, som skal ændres hver gang. Programmet er dog skrevet til at skulle læse txt filerne, hver gang der er brug for en besked i programmet. Så det vil sige at, hvis der kun er brug for et ord i programmet, så skal hele txt filen læses igen.
Det tager ikke særlig lang tid og derfor er der ikke nogen bemærkelsesværdig forskel på om man laver et program som ikke skal indlæse en fil, eller om txt filen skal læses hver gang der er behov for en besked. Det ville dog være en forbedring af programmet hvis man sikrede sig at filerne kun bliver indlæst i starten, når programmet skal indlæses. På den måde kan man sikre sig at hvis man evt udviklede på programmet, og lavede en meget lang tekstfil som skulle indlæses, at der så stadig ikke ville være en bemærkelsesværdig forskel.

Vi havde lidt problemer i starten med hvilke referenced libraries der var blevet brugt på github. Og derfor ville det være en god ide at have et eksternt program til at ordne den slags ting for os. Også så der ikke ville komme forskellige stier til de ekstra libraries som vi har

brug for til vores program. Der kunne fx være en der har lagt en sti til vores GUI på et lokalt sted på computeren, og når den så bliver pushed til github, så er det ikke muligt for andre at bruge programmet, da der er forskel på stierne. Vi regner med at bruge et værktøj til hedder maven i fremtiden til at løse dette problem. Det er en online database med en masse libraries, som kan bruges til programmer. Da det kan være svært at holde styr på når man når til en masse, så kan det godt være en god ide at implementere allerede i programmet tidligt.

# 5. Test

Vores tests er lavet for at teste de simpleste komponenter i programmet, og sikre at de virker. Der er lagt vægt på klasser, der er lavet med lav kobling og høj sammenhørighed, som gør programmet lettere at teste og rette eventuelle fejl, uden at man behøver at omprogrammere det hele. Vi har taget nogle tests, som vi lavede i sidste opgave, og lavede dem om til at kunne passe i denne opgave. Det er bl.a. tests, som kunne være relevante for mht til fx at teste tilfældigheden af terningerne og tidsfaktoren, så vi sikre os et responsivt program.

Vi kunne evt have lavet en test der tester klassen med oversæt, og læser de filer, som skal indlæses i programmet, for at sikre at det ikke er de forkerte beskeder, der ender med at dukke op. Dette kan kun testes for specifikke sprog, der allerede på forhånd er kendt. Hvis filerne bliver lavet om så programmet er kompatibelt med andre sprog, er det ikke sikkert, at programmet kan sikres at have de korrekte ord de rigtige steder, men det er noget man ikke kan tage højde for.

Vi har haft nogle få problemer med vores libraries og det er også noget, der er blevet nævnt tidligere. Det burde også nævnes i test afsnittet, da det påvirker testene, som er nød til at være kompatibelt med vores libraries.

Programmet er designet til at være meget let at bruge, og det er derfor at der også er EN knap som man skal trykke på midt i vinduet. Og beskederne i vinduet viser hvad man skal gøre og hvad der er sket i spillet. Det er derfor meget let at bruge af slutbrugeren.
Vi burde have en ekstern person med begrænset kendskab til programmering til at teste programmet, det er selvfølgelig noget som vi ville kunne gøre bedre til en anden gang. Der´er alligevel taget meget hensyn til slutbrugeren i forbindelse med programmeringen.

# 6. Projektplanlægning

Ligesom i CDIO delopgave 1 har vi valgt at bruge Unified Process til at have overblik over vores arbejdsproces. Vi har gjort brug af de fire faser, Inception, Elaboration, Construction og Transition.

**Inception:**
I inceptionfasen har vi lavet en kort beskrivelse af kundens krav og har prøvet at klarlægge kundens vision og krav til programmet. Derudover vurderede vi, i denne fase, at det ønskede program godt kunne konstrueres på baggrund af de opgivende krav.

**Elaboration:**
I denne fase identificerede og opdelte vi kundens krav op efter FURPS-modellen. Vi blev dog enige om at tilføje ekstra krav til programmet såsom hastighed af responstid samt meddelelser spillet skulle returnere, som vi mente ville forbedre programmet.
Derefter gik vi i gang med de forskellige diagrammer kunden havde stillet krav til, use case diagram, use case beskrivelser (brief, casual og fully dressed), domain model og system sekvens diagram. Vi lavede også diagrammer, der kunne passe under design; Design klasse diagram, sekvensdiagram og til sidst en beskrivelse af GRASP mønstre.

**Construction:**
I constructions fasen udviklede vi dét program, som kunden ønskede med gruppens tilføjelser. Programudviklingen og implementeringen inddrager den største del af arbejdsprocessen.

**Transition:**
I denne fase videregives programmet til kunden, kunden tester programmet og vi opnår hermed feedback.

# 7. Konklusion

Formålet med opgaven har været at udvikle et terningespil af 11 felter, hvor man under hvert felt opnår et resultat der forstørre eller formindsker spillerens point. Spilleren er vinder når man opnår 3000 eller derover. Vores program virker som det skal, dog med nogle ændringer til at forbedre programmet. Vi har under projektet gjort brug af Unified Process for hver udviklingsfase, hvilket har gjort det mere overskueligt at færdiggøre alle elementer i projektet. Vi har benyttet flere UML diagrammer gennem faserne, såsom use case-, sekvens- og design klasse diagrammer. Diagrammerne giver en god visualisering af formålet til kunden og skaber overblik over programmets indhold.

# 8. Litteratur og kildeliste

- ***Larman, Craig. (2005)***: 'Applying UML and Patterns - An Introduction to Object-Oriented Analysis and Design and Iterative Development'. (3. udgave). Pearson Education Inc.

- ***Larman, Craig. (2001):*** 'Applying UML and Patterns - An Introduction to Object-Oriented Analysis and Design and Iterative Development'. (2. udgave). Pearson Education Inc. Fundet på siden: https://www.utdallas.edu/~chung/SP/applying-uml-and-patterns.pdf