

## CDIO 3 - Monopoly Junior

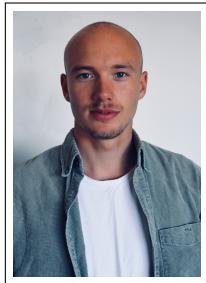
---

Group 15

Deadline: 30th of November 2018

This report contains 27 pages excluding front page and appendix

---



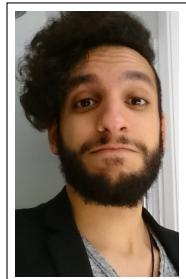
Karl Emil Jeppesen  
s180557



Søren Poulsen  
s180905



Alfred Röttger Rydahl  
s160107



Noah F. M. Hamza  
s185084



Rasmus Sander Larsen  
s185097

# Indhold

<b>1</b>	<b>Introduktion</b>	<b>2</b>
1.1	Timestyring . . . . .	3
<b>2</b>	<b>Kravspecifikation</b>	<b>4</b>
2.1	Spilkrav . . . . .	4
2.2	Fravalgte krav . . . . .	5
2.3	Implementeringskrav . . . . .	5
2.4	Test krav . . . . .	5
2.5	Feltliste . . . . .	6
<b>3</b>	<b>Analyse</b>	<b>7</b>
3.1	Interessentanalyse . . . . .	7
3.2	Use cases og kravsanalyse . . . . .	8
3.3	Domænemodel . . . . .	10
3.4	System sekvensdiagram . . . . .	11
3.5	Risikoanalyse . . . . .	12
<b>4</b>	<b>Design</b>	<b>13</b>
4.1	GRASP-mønstre . . . . .	13
4.2	Design klasse diagram . . . . .	14
4.3	Sekvensdiagram . . . . .	14
<b>5</b>	<b>Implementering</b>	<b>16</b>
5.1	Model package . . . . .	16
5.1.1	Player package . . . . .	16
5.1.2	Die Package . . . . .	16
5.1.3	Board Package . . . . .	17
5.1.4	Chancecard package . . . . .	18
5.1.5	Game Package . . . . .	18
5.2	Controller Package . . . . .	19
5.3	View Package . . . . .	19
<b>6</b>	<b>Verifikation</b>	<b>20</b>
6.1	Junit . . . . .	20
6.1.1	Valgte test cases . . . . .	20
6.1.2	Test om dækket er random . . . . .	20
6.1.3	Chancekortenes test . . . . .	21
6.1.4	Kan spillere modtage og betale penge? . . . . .	22
6.2	Code coverage . . . . .	22
6.2.1	Code coverage procenter: . . . . .	23
6.3	Bruger test . . . . .	23
<b>7</b>	<b>Documentation</b>	<b>23</b>

7.1	Arv	23
7.2	Abstract	24
7.3	LandOnField	24
7.4	GIT-import og kørsel af program	24
<b>8</b>	<b>Projektforløb</b>	<b>24</b>
<b>9</b>	<b>Konklusion</b>	<b>25</b>
<b>10</b>	<b>Bilag</b>	<b>26</b>
10.1	Chancekort	26

# 1 Introduktion

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

Spillet er en version af Monopoly Junior. To til fire spillere slår med en terning, og rykker rundt på en spilleplade med 24 felter, som hver har eget navn og effekt. I spillet er der inkluderet 10 forskellige chancekort.

Spillerne begynder ved "START" og flytter brikkerne med uret ifølge terningkast. Når en spillerbrik lander på et felt, der ikke allerede ejes af nogen anden spiller, skal spilleren købe det af banken og indkassere leje af modspillerne, når de lander på det pågældende felt.

Har en spiller ikke penge nok til at betale husleje, købe en ejendom, eller betale afgiften for et chancekort, så er spilleren gået fallit og spillet slutter dermed. De andre spillere tæller deres penge sammen, og den, der har flest, har vundet. Er det uafgjort, så lægges værdien af ens ejendomme oveni ens penge.



Figur 1: Monopoly Junior spilleplade med brikker, penge og chancekort

## 1.1 Timestyring

Herunder fremgår gruppemedlemmernes timeforbrug fordelt på specifikke opgavetyper. 2,1,7

Opgave/Navn	Alfred	Søren	Rasmus	Noah	Karl Emil
Design	0	1	0	2	0
- Navneord	0	0	0.5	1	0.5
- Designklasse	2	0	1	1	2
- System sekvens	0	0	0.5	3	0
- Sekvens	0	0	0.5	3	0
Implementering					
General opsætning	2	1	3	0	1
model					
- Die package	0	0	0	0	0
- Player package	0	0	1	0	0
- Board package	0	0	5	0	2
- Chancecard package	1	0	5	0	1
- Game package					
- - Game	0.5	0	3	0	5
- - Turn	1	0	15	0	3
controller	5	0	0	0	1.5
view	6	0	0	0	0
Test					
- Die	0	2	0	0	0
- Cup	0	1	0	0	0
- Player	0	2	0	0	0.5
- Account	0.5	2	0	0	0
- Chancekort	0	2	0	0	0
- Game	0	0.5	0	0	0
- Board	0	2.5	0	0	0
Rapport				8	
- Kravspecifikation	0	0	0.5	2	0.5
- Analyse	1	2	0	4	1
- Verifikation	0.5	6	0	0	0.5
- Implementering	3	0	4	0	1
Total	22.5	22	39	24	19.5
Nettoforbrug	127	timer			

## 2 Kravspecifikation

Kravsspecifikationen fremgår i dette afsnit, som den er stillet i opgaven. De krav, som ikke er blevet implementeret, forefindes i bunden af listen noteret med kursiv skrift for at differentiere disse.

### 2.1 Spilkrov

(S<sub>1</sub>) Spil mellem 2-4 personer.

(S<sub>2</sub>) Spillerne slår på skift med 1 terning.

- Det skal anvendes en 6-sidet terning.

(S<sub>3</sub>) Der skal være 24 felter.

- Felterne har numrene 1-24.
- Der skal være 16 felter med ejendomme.
- Der skal være 4 chance felter.
- Der skal være 1 "Start"felt.
- Der skal være 1 "På besøg i fængsel"felt.
- Der skal være 1 "Gratis Parkering"felt.
- Der skal være 1 "Gå i fængsel"felt.

(S<sub>4</sub>) Alle spillere skal starte på "Start feltet".

(S<sub>5</sub>) Spillerne skal kunne lande på et felt og så fortsætte derfra på næste slag. Man går i ring på braettet.

(S<sub>6</sub>) Spiller forsætter mod venstre. Man rykker altid frem.

(S<sub>7</sub>) Alle spillere starter med en vis mængde penge, baseret på antal spillere.

- For 2 spillere, så har hver spiller 20 sedler.
- For 3 spillere, så har hver spiller 18 sedler.
- For 4 spillere, så har hver spiller 16 sedler.

(S<sub>8</sub>) Chancekortene skal blandes før hvert spil.

- Der er 10 forskellige chancekort.

(S<sub>9</sub>) Spillet afsluttes når en spiller ikke kan betale husleje, købe en ejendom eller betale en udgift fra et chancekort.

(S<sub>10</sub>) Spilleren med flest penge er vinderen.

- Spilleren, som gik fallit, kan ikke vinde.
- Hvis det er uafgjort, så skal ejendommenes værdier lægges oveni i spillerens penge, og derefter tælles der på ny.

(S<sub>11</sub>) Spillerne vælger mellem 1 af 4 unikke spilbrikker: Bil, Racerbil, UFO og Traktor.

(S<sub>12</sub>) Når en spiller lander på en ledig ejendom, skal denne købes. Hvis den er ejet, skal spilleren betale husleje.

## **2.2 Fravalgte krav**

- (M<sub>1</sub>) *En spiller skal vælges som bankør i starten af hvert spil.*
- (M<sub>2</sub>) *Den yngste spiller starter først.*
- (M<sub>3</sub>) *Der medfølger 48 'solgt' kort for notation af ejerskab.*
- (M<sub>4</sub>) *Der medfølger 90 penge sedler.*

## **2.3 Implementeringskrav**

- (I<sub>1</sub>) Lav passende konstruktører.
- (I<sub>2</sub>) Lav passende get og set metoder.
- (I<sub>3</sub>) Lav passende toString metoder.
- (I<sub>4</sub>) Lav en klasse GameBoard der kan indeholde alle felterne i et array.
- (I<sub>5</sub>) Tilføj en toString metode der udskriver alle felterne i arrayet.
- (I<sub>6</sub>) Lav det spil kunden har bedt om med de klasser I nu har.
- (I<sub>7</sub>) Benyt GUI'en. Gui' skal importeres fra Maven: Maven repository

## **2.4 Test krav**

Herunder beskrives kundens tanker med hensyn til test af spillet.

- (T<sub>1</sub>) Lav tre testcases med tilhørende testscripts og testrapporter.
- (T<sub>2</sub>) Lav en Junit test til centrale metoder. Inkludér code coverage dokumentation.
- (T<sub>3</sub>) Lav mindst én brugertest. Husk at bruger'en skal være en der ikke kan kode.

## 2.5 Feltliste

Herunder fremgår de enkelte felters navn, pris og farve.

Feltnavn	Feltpriis/effekt	Feltfarve
1. Start	+2 penge når passeres	Ingen
2. Burgerbar	1 Penge	Brun
3. Pizzaria	1 Penge	Brun
4. Chance	Træk et chancekort	Ingen
5. Slikbutik	1 Penge	Lyseblå
6. Iskiosk	1 Penge	Lyseblå
7. På besøg	Gør intet	Ingen
8. Museum	2 Penge	Pink
9. Bibliotek	2 Penge	Pink
10. Chance	Træk et chancekort	Ingen
11. Skaterpark	2 Penge	Orange
12. Swimmingpool	2 Penge	Orange
13. Gratis parkering	Gør intet	Ingen
14. Spillehal	3 Penge	Rød
15. Biograf	3 Penge	Rød
16. Chance	Træk et chancekort	Ingen
17. Legetøjsbutik	3 Penge	Gul
18. Dyrehandel	3 Penge	Gul
19. Gå i fængsel	I fængsel (felt 7)	Ingen
20. Bowlinghal	4 Penge	Grøn
21. Zoo	4 Penge	Grøn
22. Chance	Træk et chancekort	Ingen
23. Vandland	5 Penge	Mørkeblå
24. Strandpromenade	5 Penge	Mørkeblå

### 3 Analyse

Der skal udvikles et Monopoly Junior spil, hvorfor det er nødvendigt at bestemme, hvilke klasser der skal interagere med hinanden.

1. Der er en 'die'-package som indeholder:

- En 'Die'-klasse, der ved kald returnerer et tal tilsvarende en ternings øjne. Det returnerede tal skal falde inden for den statistisk sandsynlighed.
- En 'Cup'-klasse, der sørger for at terningen bliver rullet og inputtet til spillet - og den kan styre hvor mange terninger, der skal slås med, samt hvor mange sider de terninger skal have.

2. Der er en 'player'-package som indeholder:

- En 'Player'-klasse, der holder styr på spillerens pengebeholdning, navn, placering på brættet og spilbrik.
- En 'Account'-klasse, som opdaterer kontoens pengebeholdning.

3. Der er en 'board'-package, som indeholder:

- En 'Board'-klasse, som sørger for at arrangere felterne i korrekt rækkefølge i et array.
- En 'Field'-klasse, som indeholder felternes nummer, navn, beskrivelse, ejer og farve. Field er en abstrakt klasse, som agere superklasse til henholdsvis ejendomsfelter, besøgsfelter, fængsel og start.

4. Der er en 'chancecard'-package, som indeholder:

- En 'Card'-klasse indeholder et chancekorts navn og beskrivelse. Cards er en abstrakt klasse, som er superklasse til klasser for kort med økonomisk konsekvens og konsekvens for spillerens position.
- En 'Deck'-klasse sørger for at antallet af chancekort er korrekt, at de bliver blandet og at en spiller kan trække et kort fra dækket.

5. Der er en 'game'-package som indeholder:

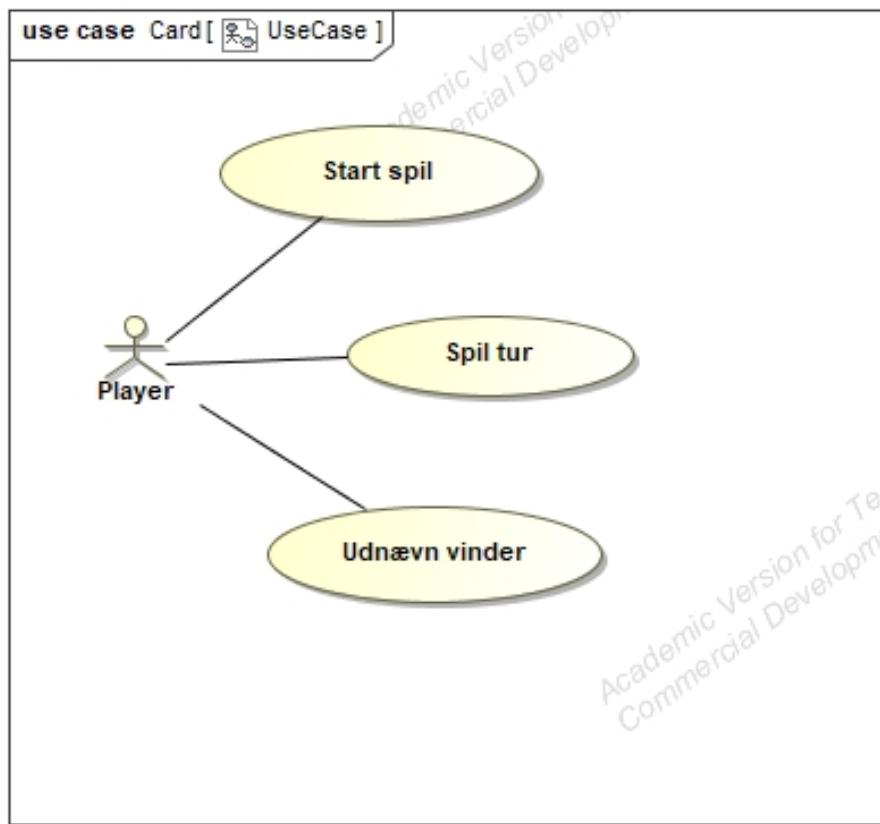
- En 'Turn'-klasse, som sørger for at en given spillers tur forløber hensigtsmæssigt, klasserne interagerer korrekt og at spilleren følger reglerne.
- En 'Game'-klasse, som indeholder spillets regler og interagerer med spillerne og terninger på en måde, der konstituerer spillet herunder alternering mellem spillere og kåring af en vinder.

#### 3.1 Interessentanalyse

Der er også i denne opgave interessenter 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 er afbilledet herunder:



Figur 2: Use Case Diagram

De viste use-cases er kortfattet beskrevet herunder, med 'under'-use cases inkluderet.

## Use case beskrivelser

Use case	Beskrivelse
<u>1: Start Spillet</u> - 1.1: Vælge spiller brik - 1.2: Vælg spiller navn - 1.3: Vælg terning	Spilleren skal kunne starte spillet Spilleren vælger/får tildet en af de 4 startbrikker Spilleren indtaster et navn Spillerne skal kunne vælge hvilke slags terninger (Hvis mulighed)
<u>2: Spil tur</u> - 2.1: Slå med en terning - 2.2: Bruge et chancekort - 2.3: Købe - 2.4: Leje/udleje - 2.5: Lande på et felt - 2.6: Ryge i fængsel - 2.7: Parkere gratis/på besøg  - 2.8: Lande på/passere start  - 2.9: Prøve chancen	Spilleren skal kunne spille spillet Spilleren slår med en terning og får en talværdi Spilleren skal kunne bruge et chancekort Spilleren skal kunne købe en ejendom Spilleren skal kunne udleje/leje en ejendom Spilleren skal rykke sig frem på brættet efter et slag Spilleren skal kunne ende i fængsel Spilleren skal have ture, hvor han ikke gør andet end at rykke sig Spilleren starter på start feltet og kan lande på eller passere det igen Spilleren skal kunne lande på et chance felt
<u>3: Udnævn vinder</u> - 3.1: En spiller kan gå fallit  - 3.2: Find vinder, hvis uafgjort	En spiller skal kunne vinde En spiller skal kunne gå fallit Hvis der er to spillere med samme pengeværdi efter en anden spiller er gået fallit, så skal deres ejendomsværdier tillæges deres pengebeholdning.

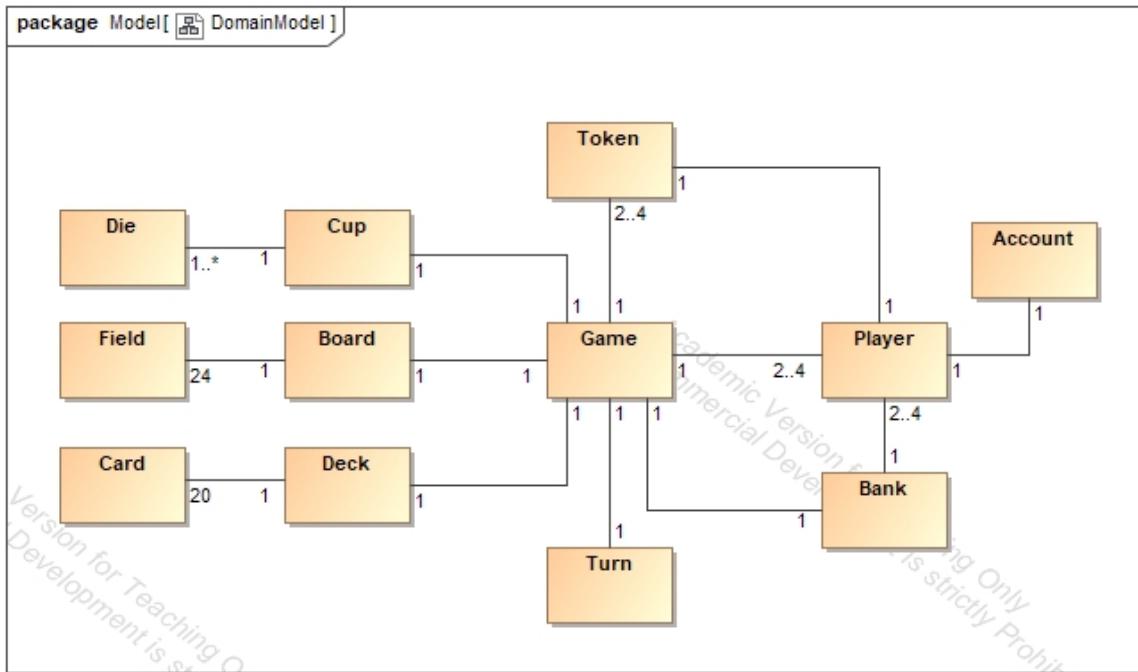
Som ønsket forefindes en fuld beskrivelse af en central use case i det følgende. Den valgte use case er "Spil tur".

Use case sektion	Beskrivelse
Navn	Spil tur
Scope	Monopoly spil
Level	Bruger
Primær aktør	Brugerne
Andre interesserter	IOOuterActive
Preconditions	Spiller er startet
Postconditions	En vinder udnævnes
Vigtigste success-scenarie	<ol style="list-style-type: none"> <li>1. Systemet viser spillets udfald, dvs. terningeslag og bevægelser på brættet.</li> <li>2. Spillere kan købe/sælge ejendomme</li> <li>3. Spillere kan leje/udleje ejendomme</li> <li>4. Ture gentages indtil spillet afsluttes når der er fundet en vinder</li> </ol>
Alternative scenarier	<ol style="list-style-type: none"> <li>1. En spiller kan ryge i fængsel</li> <li>2. En spiller kan "Gå på besøg" og "Gratis parkere"</li> <li>3. En spiller kan trække og bruge et chancekort</li> </ol>
Specielle krav	Ingen
Teknologi og dataliste	Ingen
Udvidelser	Ingen
Frekvens	Hver gang spillet startes

### 3.3 Domænemodel

En domænemodel er udarbejdet for at beskrive hvordan de vigtige klasser arbejder sammen. Det er ligeledes i den her model, hvor multipliciteten af en klasse ift. til en anden klasse ses, altså hvor mange gange et bestemt objekt forekommer i spillet. Der findes eksempelvis en terning til en cup, men spillets udformning muliggør tilføjelse af flere terninger, hvis det er behov for det. Der er 24 felter til et bræt, 10 chancekort til et dæk af kort og så fremdeles.

Undervejs i implementeringen, så blev "Token" og "Bank", som egen klasse overflødig, men diagrammet nedenfor er det originale diagram, beholdt for at vise hvordan designet blev udtænkt før selve programmeringen gik igang. Diagrammet er også let på programmeringstermer, f. eks står der ikke "string" under player. Ligeledes er der heller ikke attributer til de andre klasser, da diagrammet skal være overskueligt.



Figur 3: Domæne Model

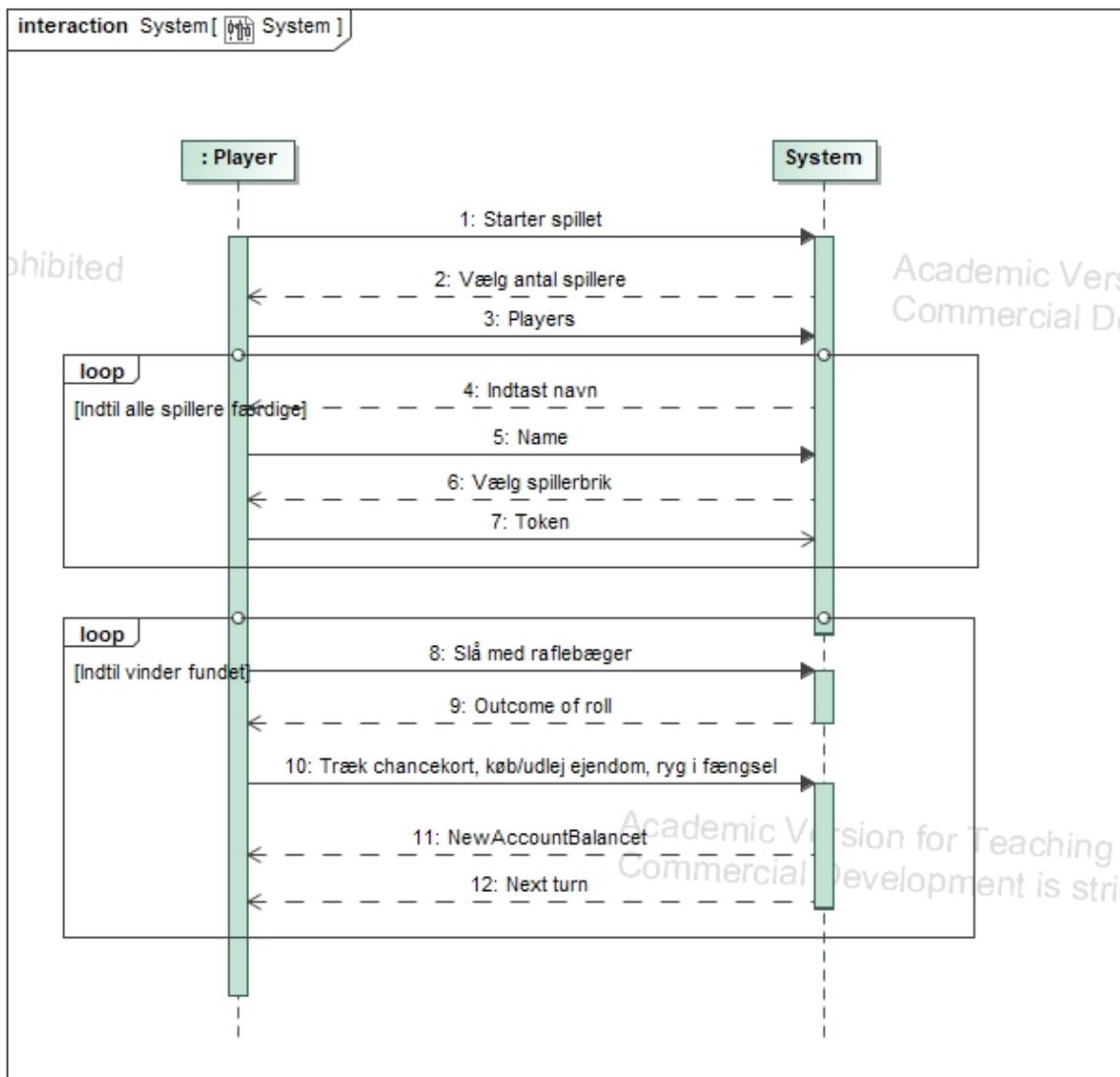
### 3.4 System sekvensdiagram

Nedenstående diagram dækker over nogle af elementerne i use case 1 (start spil) og 2 (spil spil), således at spillets forløb i sin helhed kan beskrives.

I diagrammet vises hvordan spillet startes med valg af antal spillere samt spilbrikker og navne til hver spiller (repeteret i form at et loop). Der er ikke taget højde for at et deck også bliver skabt og blandet på dette tidspunkt.

Derefter starter en spillers tur. Der slås med raflebægeret som giver et resultat, og spilleren rykkes hen på det korresponderende felt, hvorefter spilleren tager konsekvensen af fletet. I diagrammet er fremhævet "Træk chancekort, køb/udlej ejendom, ryg i fængsel" som muligheder efter at have landet på et felt, men der er flere muligheder - de bare ikke afbildet for at gøre diagrammet mere overskueligt. Hermed kan man se hvordan en normal runde for en spiller kan se ud, før spillet gentager det samme for den næste spiller. Dette loop gentages indtil en spiller går fallit og hvorefter en vinder findes.

Diagrammet kunne godt have været delt op i to, hvor det ene indholder pil 1-7 og det andet indholder pil 8-12, men det er samlet i et for at passe til opgavekravene.



Figur 4: System sekvensdiagram

### 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 af programmet ikke faciliteter implementering af ekstraopgaver.
2. Gruppemedlemmernes manglende versionsstyring besværliggør grupppearbejdet.
3. Projektledelsen og timestyringen skævvridter arbejdsbelastningen.
4. Den implementerede kode ikke er kompatibel med den udleverede GUI, og der derfor skal ændres markant på koden.

5. Gruppens mangel på test resulterer i fejl som ikke opdages, f. eks hvis terningerne ikke er testet til at være tilfældige.

Der er i brugen risiko for, at

1. Programmets vejledning ikke er tilstrækkelig til brug, især for en person der ikke har erfaring med programmering.
2. Fejlagtig brug kan beskadige eller manipulere programmet.
3. Spillet er ikke testet nok til at sørge for at alle tilfældigheder faktisk er tilfældige, og derfor ender spillet med at være ubalanceret.

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

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 en kort beskrivelse af anvendte GRASP-mønstre, et designklassediagram samt et design sekvensdiagram.

### 4.1 GRASP-mønstre

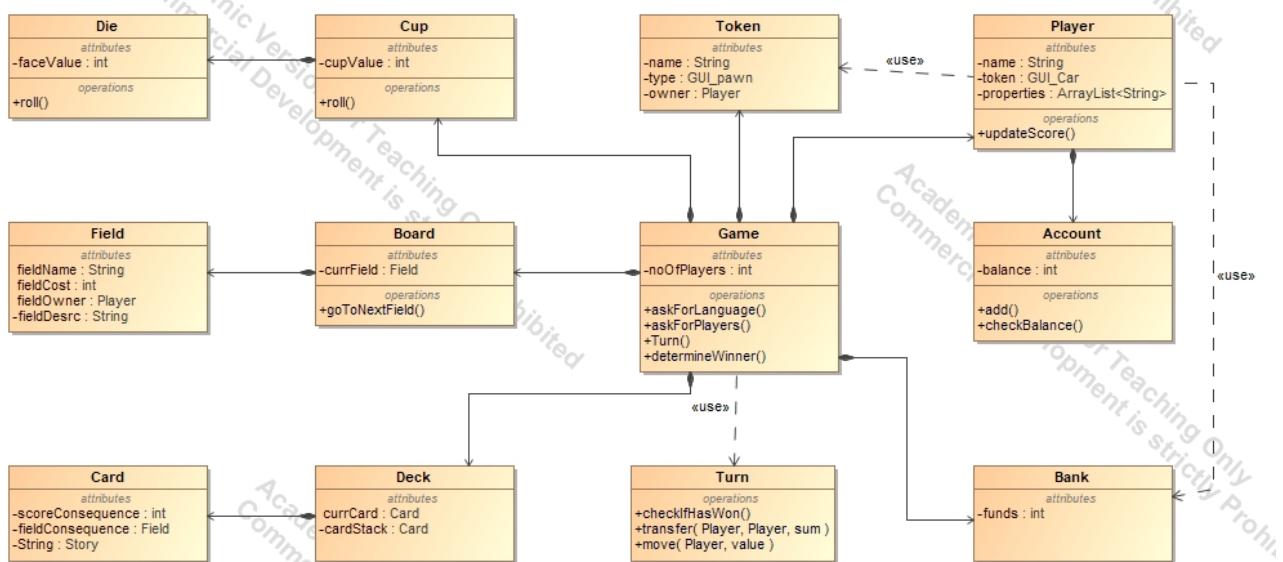
GRASP beskriver nogle retningslinjer i forhold til at fordele ansvar til klasser og objekter, altså en beskrivelse af, hvordan en problemtyppe kan løses. GRASP udgør forskellige mønstre og principper, herunder de fem mønstre, som der vil være fokus på i denne opgave og som ses herunder. Eksempler på brug af GRASP-metoder følger i afsnit 7.

1. **Creator:** En klasse A skal være ansvarlig for at oprette nye instanser af en anden klasse B i visse situationer.

2. **Information Expert:** Der er tildelt de forskellige klasser, der har den nødvendige information til at udføre en handling, ansvaret for denne handling.
3. **Controller:** En klasse der håndterer events i programmet, altså use cases. Controlleren er ikke UI men behandler brugerens input på en måde modellen kan bruge.
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 og genbruge kode.
5. **High Cohesion:** Den enkelte klasse tildeles et let, forståeligt og ensartet ansvarsområde, eller eventuelt flere ansvarsområder, der er tæt relaterede til hinanden.

## 4.2 Design klasse diagram

Herunder ses systemets design klasse diagram, hvor der fastlægges nedarvning (er-en-relationerne), referencer mellem objekter (har-relationer) samt de vigtigste variabler og metoder. Designklassediagrammet har gennemgået flere iterationer, idet der undervejs viste sig nye måder at strukturere koden på end først havde skitseret. Det nedenstående er et af de sidste klassediagrammer, inden "Token-klassen blev smidt ind i "Player-klassen.



Figur 5: Designklassediagram

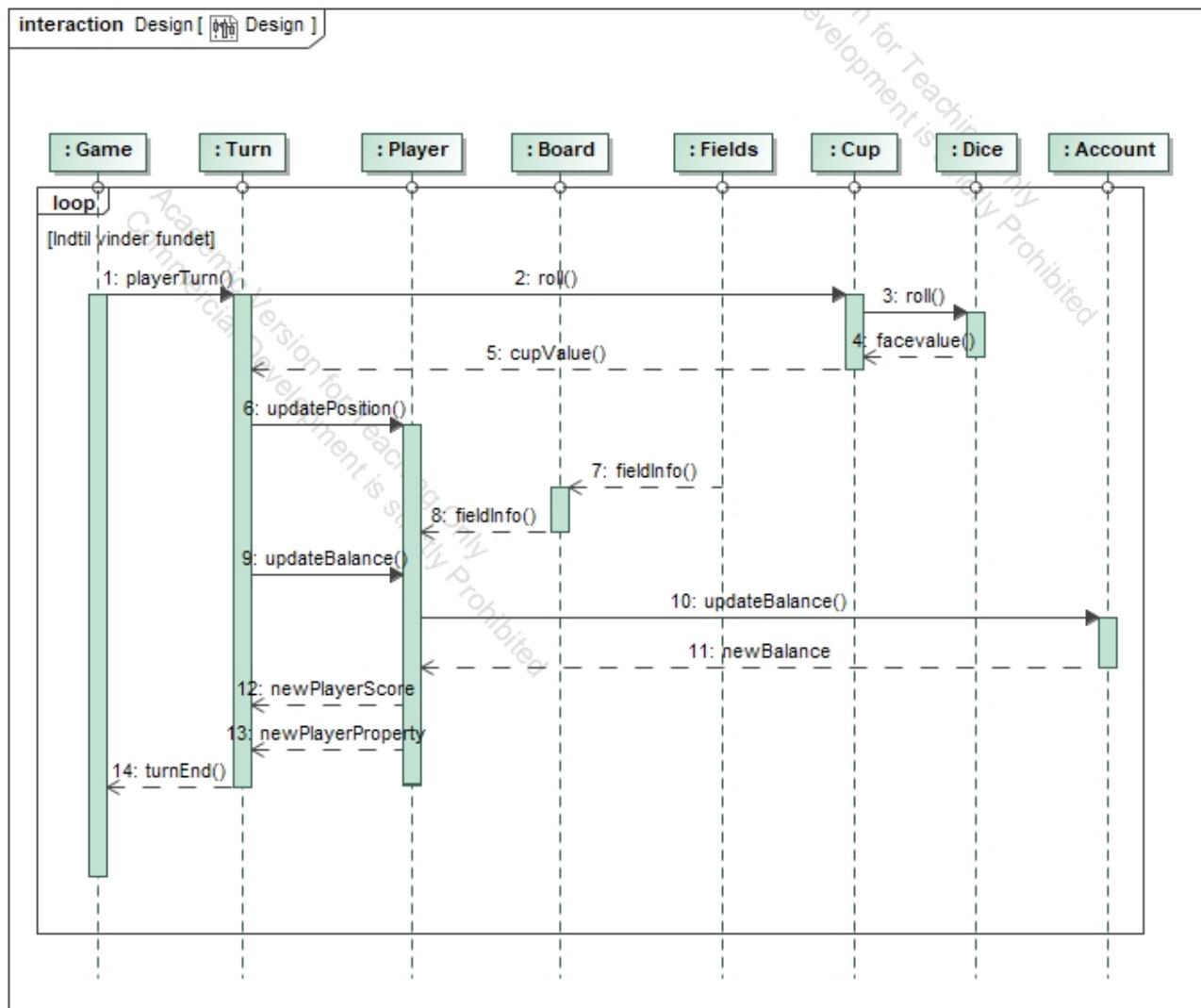
## 4.3 Sekvensdiagram

Sekvensdiagrammer viser en udveksling af meddelelser, altså 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. Tidsaksen er også lodret, og vokser nedad, så meddelelser sendes fra et objekt til et andet i form af pile med operationer og parameternavn.

Der er udarbejdet et design sekvensdiagram over use case 2 "Spil spil". Der er fremhævet de væsentlige aspekter af en tur for at gøre diagrammet overskueligt. GUI'en og 'Deck'-klassen samt 'Card'-klassen er derfor ikke afbilledet, og der er kun vist et par af de felt muligheder i spillet. En forklaring af diagrammet følger.

Spilleren slår med et raflebæger, hvis værdi bliver returneret. Spillerens position på brættet bliver opdateret og feltets information hentes fra "Fields". Derefter bliver spillerens pengebeholdning opdateret. Den næste spillers tur begynder nu, og det hele gentages i et loop indtil en vinder er fundet, altså indtil en spiller er gået fallit. Loopet er ikke afbilledet, da det gør diagrammet mere uoverskueligt.

Der er ikke indtastet parametre til hver pil, såsom 'int' inde i "cupValue()".



Figur 6: Design sekvensdiagram

## 5 Implementering

I dette afsnit vil brudstykker af den anvendte kode blive gennemgået, samt implementering forklaret. Programmet er opdelt overordnet opdelt i 3 packages; model, controller og view.gui. Det er gjort i et forsøg på at efterleve pricipperne fra Model, View, Controller princippet. De to packages, controller og view.gui indeholder hver én klasse, henholdsvis Controller klassen og GUI klassen. Model indeholder fem packages, hvilke sammen med hver klasse i controller og view gennemgåes nedenfor.

### 5.1 Model package

Model er den package, hvori al funktionalitet ligger og kan ses om motorrummet i programmet. Denne package indeholder 5 packages, som står for hver deres ansvarsområde. De 5 packages i model er player, die, board, game og chancecard. Herunder følger introduktion til hver enkelt pakke.

#### 5.1.1 Player package

Player indeholder to klasser: Player og Account. Denne Package har ansvaret for spilleren og vedkommandes score. *Denne package er genbrugt fra CDIO2 og tilrettet for at opnå funktionaliteter, som var nødvendige i dette program.*

**Player klassen** styrer navnet, pengebalancen (eller score), token (spillerbrikken), position og værdien af spillerens ejendomme, for en bestemt spiller. Balancen er håndteret af en instans af objektet Account, som styrer spillerens point.

Player har yderligere to booleans; hasLost og inPrison. hasLost fortæller om spilleren er gået fallit og har tabt, og inPrison fortæller om spilleren er blevet sendt i fængsel. Ved instansiering af en player er hasLost og inPrison "false", men hvis spilleren opfylder betingelserne for at have tabt eller bliver sendt i fængsel, bliver hasLost sat til *true*.

Player har 3 metoder; updateScore, update position og updateTotalPropertyValue. UpdateScore opdaterer spillerens score med værdien af en *int*. UpdatePosition opdatere spillerens position på brættet og sørger for at den ikke overstiger antallet af felter på brættet. UpdateTotalPropertyValue gennemgår priser på de ejendomme som spilleren ejer og summere dem og tildeler summen til variablen *totalPropertyValue*.

**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 modtager en *int*, som adderes til variablen *balance*.

#### 5.1.2 Die Package

Die indeholder to klasser, Die og Cup. Denne package står udelukkende for oprettelse og håndtering af terningerne og raflebægret, der bruges i spillet. *Denne packages er 100% genbrugt fra CDIO2, da koden herfra er tilstrækkelig fleksibel.*

**Die klassen** indeholder 2 konstruktører, som kan lave en klassisk 6 sidet terning eller terning med et vilkårligt antal sider alt efter input i konstruktøren. Der slås med terningen ved metoden "roll", som

returnerer et tilfældigt tal korresponderende til en af terningens sider.

**Cup klassen** fungerer som raflebæger og har også 2 konstruktører, som laver et bæger med én terning som brugt i spillet eller med et vilkårligt antal terninger alt efter input i konstruktøren. Bægeret bliver "raflet" ved at kalde metoden cupRoll, der kalder "roll" metoden på de oprettede die instanser. Summen af terningerne gemmes i cup instansen.

### 5.1.3 Board Package

Board indeholder to klasser, Field og Board, samt en package (Fields), som indeholder fem klasser, Prison, Visiting, Property, Start og Chancefield, som repræsenterer de fem typer felter brættet indeholder. Board package står for konstruktion af selve brættet, som spillet foregår på, dets felter og handlinger derpå.

**Field klassen** er abstrakt og danner grundlag for samtlig af de fem typer af felter, som brættet indeholder. Field indeholder den information, som alle felter har brug for og skal have til fælles, samt metoden "action", som udfører handlingen for det respektive felt. Et felt består af et feltnummer (*int*), typen af felt (*String*), en beskrivelse (*String*), en pris (*int*) og en farvekode (*Color*).

Som nævnt ovenfor indeholder programmets package board, en package som hedder "Fields", som indeholder klasserne for hver af de fem typer felter vi har i spillet. Alle disse fem klasser er nedarvet fra den abstrakte klasse Field.

**Start klassen** benyttes som startfelt på brættet og indeholder ingen yderligere funktioner.

**Prison klassen** benyttes som "fængselsfeltet" og "på besøg i fængslet". Feltets action metode skifter handlingen for "fængslet" og "besøg", ved at tjekke *fieldNumber (int)* og benytter "besøgs-handlingen", hvis feltet er placeret på felt nummer 7 og benytter "fængslet-handlingen" hvis feltet er placeret på felt nummer 19. "Fængslet-handlingen" rykker spilleren direkte til fængslet (felt 7) og sætter spillerens *hasLost (boolean)* til true.

**Visiting klassen** benyttes som parkeringsfelt på brættet og har ingen funktion i forhold til spillet.

**Property klassen** benyttes som ejendomsfelt på brættet. Feltets action metode har 2 funktion, køb ejendommen eller betal husleje til ejeren af ejendommen. Har feltet ingen *owner (Player objekt)* købes feltet, for feltets *cost (int)*, hvilket sker ved metoden *setOwner(player)*. Har feltet allerede en *owner*, betaler spilleren der landede på feltet, *cost* antal penge til ejeren.

**Chancefield klassen** benyttes som chancekortsfelt på brættet. Feltets action metode trækker det øverste kort i bunken med chancekort og udføre kortetshandling.

**Board klassen** består helt simpelt af et array af Field objekter, med en størrelse på 24 pladser, hvor hver plads er et felt, hvis specifikke egenskaber hardcodes i boardklassen. Med hardcoded menes at oplysninger som 'Burgerbar' og pris er feltspecifikke og derfor skrives ud når feltet instansieres. Udsnit af hardcoded af felter er vist nedenfor.

```
public void createBoard () {  
  
    addStart(1, "Start", "Du er landet p    start og har f et 2  
        ekstra penge", 0, Color.GREEN);  
    addProperty(2, "Burgerbar", "Du er landet p    burgerbaren", 1,
```

```
new Color(205,133,63));
```

#### 5.1.4 Chancecard package

Chancecard indeholder to klasser, Deck og Card, samt en package (chancecard), som indeholder 3 klasser, MovingRel, MovingAbs og Transfer, der repræsenterer de tre forskellige typer af chancekort, programmet indeholder. Chancecard package står for alt funktionalitet i forbindelse med chance kortene og stakken af chancekort.

**Card klassen** er abstrakt og danner grundlag for hver af de tre typer chancekort. Card indeholder en række variable, som alle tre korttyper, har til fælles, samt en metode *action*, som håndtere kortets handling. Et kort består af: en korttype *String*, en titel *String* og en beskrivelse *String*. Disse tre typer af chancekort er alle nedarvet fra den abstrakte klasse Card og har hver deres type af handling.

**MovingAbs klassen** benyttes som et chancekort der rykker spilleren til et specifik felt, altså en absolut placering. Kortets action metode sørger for at spillerens position bliver magen til den, som kortet foreskriver.

**MovingRel klassen** benyttes som et chancekort der rykker spilleren et relativt antal felter fremad eller bagud. Kortets action metode sørger for at spilleren position ændres, således det passer med den relativt flytning, som kortet forskriver. Ryk baglæns er en *int<0* og fremadrettet ryk er en *int>0*.

**Transfer klassen** benyttes som et chancekort med en økonomisk konsekvens for spilleren, altså enten tildeling eller inddragelse af sedler. Kortets action metode kalder updateScore på player, med kortets respektive pris *int>0* eller gevinst *int<0*.

Disse tre korttyper udgør de kort som ligger i spillets bunke af chancekort.

**Deck klassen** består af en *ArrayList* af Card objekter. Der tilføjes chancekort til bunken, ved at kalde *add*, på Arraylisten, efterfulgt af et Card objekt af en af de 3 korttyper, som hardcodeds. Nedfor vises et udsnit af hardcodingen, som skal forståes på samme måde som for felterne.

```
public void createDeck (int numberofSetsInDeck) {  
    for (int i =0; i<numberofSetsInDeck; i++) {  
        addMovingAbs("movingAbs","Start", "Ryk frem til Start", 0);  
        addMovingRel("movingRel","Forlomme", "Du får en forlomme i  
        k en", 1);  
    }  
}
```

Deck klassen har også to metoder, shuffleDeck og drawCard, som henholdsvis blander kortene i bunken og trækker det øverste kort i bunken, lægger det nederst og udfører kortets givne handling.

#### 5.1.5 Game Package

Game indeholder 2 klasser; Game og Turn. Game styre de ydre rammer for spillet, mens Turn styre én tur.

**Game klassen** indeholder alle de objekter som der er nødvendige for at spillet fungere, fx et Cup objekt, et Deck objekt, og et par variable, som bruges til spillets generelle retningslinjer, hvor mange penge spillerne starter med og hvor på brættet spillerne starter. Der oprettes det ønskede antal spiller

og hver spiller tildeles en tur (Turn klassen og dets turn metode), indtil der er en spiller der har tabt, hvorefter der udnævnes en vinder. Vinderen er spilleren med flest penge, når den første spiller er fået fallit. Har 2 eller flere spillere lige mange penge, optælles værdien af spillernes ejendomme og spilleren med størst ejendomsværdi er vinderen.

**Turn klassen** indeholder et par variable, som benyttes i logikken i en tur, samt metoden *turn*, der står for afviklingen af én tur. Først tjekkes det er spilleren er i fængsel *inPrison=true*, hvis det er tilfældet betaler spilleren for at komme ud af fængslet og ellers sker der intet. Så tjekkes det om spilleren er fået fallit *score<=0*, hvis det er tilfældet sættes spillerens *hasLost=true* og spilleren erklæres fallit, ellers sker der intet. Nu er spilleren klar til den reelle tur og slår med terningerne og spilleren rykker det antal felter, som vedkommende har slået. Passeres startfelt eller lander vedkommende på det, udbetales 2 pengesedler. Handlingen for det felt, som spilleren er landet på udføres, hvad end det er køb af ejendom, betal husleje, træk chancekort eller ryk i fængsel. Skulle spilleren lande på et chancefelt og trække et kort der rykker spilleren til et nyt felt, tjekkes det om spillet i dette ryk passere start, samt honoreres hvis det er tilfælges. Handlingen for det felt spilleren nu står på udføres, på samme måde som beskrevet ovenfor. GUI'en opdateres med spillernes nye pengebeholdning. Det tjekkes om spilleren, som resultat af turen er fået fallit, hvis det er tilfældet erklæres spilleren fallit, hvis ikke udskrives spilleren pengebeholdning.

Klasserne Game og Turn, styrer altså alt den logik der ligger bag spillet og det er her det reelle spil afvikles med hjælp fra de objekter, der er udviklet til spillet.

## 5.2 Controller Package

Pakken controller inkluderer kun én enkelt klasse, Controller.java, som har til opgave at håndterer brugerfladen og brugerinput. I denne implementering er Controller klassen lavet meget simpel, imod-sætning til den gængse MVC model.

## 5.3 View Package

Pakken view indeholder alle klasser, som har til ansvar at lave en brugerfalde og tage imod brugerinput. I denne implementering, indeholder pakken kun klassen Gui.java.

Gui.java har bl.a. ansvar for at linke objekterne Gui\_Player med Player, og GUI\_Field med Field. Det førstnævnte er hovedsagligt gjort med funktionen *createPlayers()*.

```
private ArrayList<GUI_Player> createPlayers (ArrayList<Player> players) {

    // Create the players arraylist as the right size
    ArrayList<GUI_Player> guiPlayers = new
        ArrayList<>(players.size());

    // Iterate over the players list and create a GUI_Player for each
    for ( int i=0 ; i < players.size() ; i++ ) {
        GUI_Player newPlayer = new GUI_Player(
            players.get(i).getName(),
            players.get(i).getAccount().getBalance(),
```

```

        new GUI_Car(Color.RED, Color.BLACK,
                     GUI_Car.Type.CAR, GUI_Car.Pattern.DOTTED)
    );

    // Add the created player to the player array and set the
    // player on the start field
    guiPlayers.add(newPlayer);
    fields[players.get(i).getPosition()].setCar(newPlayer, true);
}

// Return the GUI_Player array
return guiPlayers;
}

```

Funktionen `createPlayers()` har ansvaret for at oprette `GUI_Player` objekter ud fra en `ArrayList` af `Player` objekter. Gui har derudover også lignende metode til at oprette `GUI_Field`'s ud fra et Array af `Field` objekter.

Så helt generelt har denne klasse til ansvaret for at binde spillets objekter og objekterne, som ses i GUI'en.

## 6 Verifikation

Dette afsnit beskriver tilgangen til test af programmet, herunder deres udførelse og på hvilken måde de imødekommer de i opgaven beskrevne krav.

### 6.1 Junit

Testene er alle Junit tests, som kan køres automatisk og uden input. Der er filer som indholder testene henholdsvis "CupTest", "DieTest", "AccountTest", "PlayerTest", "DeckTest" og 3 "CardTest" til de forskellige typer af chancekort. Nedenunder gennemgås tre udvalgte test.

#### 6.1.1 Valgte test cases

- Er dækket random, og kan det blandes?
- Virker chance kortene som tiltænkt?
- Bliver spillerens account opdateret når der skal trækkes eller lægges penge til?

#### 6.1.2 Test om dækket er random

Når der testes om dækket er random, skal der tages forbehold for at man muligvis kan trække det samme kort 2 gange. Med 10 kort i decket, så er der nemlig en 10 procents chance for at kortet vil lande på samme placering som det havde før.

Derfor er test konstrueret således, at den først checker om det første kort der trækkes er ens med det andet kort, i hvilket tilfælde dækket igen blandes og der trækkes et kort på samme placering. Dette fremgår af nedenstående kodes 'hvis' betingelse.

```
if (cardIndex1.text.equals(cardIndex2.text)) {
    deck.shuffleDeck(1);
    cardIndex2 = deck.getChanceDeck().get(0);
}
assertNotEquals(cardIndex1.text, cardIndex2.text);
```

Hvis "if"koden bliver kørt, så får "cardIndex2"tildelt en ny værdi, inden det kører videre til "asserNotEquals", som kontrollerer af det første og andet kort ikke er det samme. Dermed kan vi se at decket er blevet blandet.

Testen har en lille chance for at fejle, hvis det skulle ske, at det første kort har den samme placering tre gange i træk, dvs. efter initialisering, efter første blanding og efter anden blanding.

### 6.1.3 Chancekortenes test

Chancekortene har mange mulige udfald, og derfor er chance kortene blevet opdelt i tre kategorier for at finde ud af om de virker om tiltænkt. De tre typer af chancekort er som følger:

- Ryk frem til et bestemt felt (fastlagt placering).
- Ryk x antal felter frem fra den nuværende placering.
- Ændre en spillerens balance, både negativt og positivt.

Det første vi checker efter er om man kan rykke spilleren hen til et specifik felt på brættet. Dette sker ved at spilleren, som står på felt Z, bliver rykket frem til felt X, hvorefter der testes om spilleren faktisk er placeret på felt X.

Herefter bliver spilleren rykket frem til et felt Y via et chancekort, hvor Y er placeret før X, og så testes der om spilleren er placeret på Y.

```
@Test
public void action() {
    //Move to a position above
    MovingAbs movingAbs = new MovingAbs("This field is on spot
        13", "and player land on 21", 21);
    Player player = new Player("jens", 100, 13);
    movingAbs.action(player);
    assertEquals(21, player.getPosition());

    //Move to a position below
    MovingAbs movingAbs1 = new MovingAbs("now you on 21, and get to
        7", "lets try", 7);
    movingAbs1.action(player);
    assertEquals(7, player.getPosition());
}
```

Testen, som er lavet for at checke om et chancekort kan flytte en spiller med udgangspunkt i der, hvor de står, sker ved at lave en spiller, som starter på feltet X. Derefter laves der et chancekort, som rykker spilleren fire felter frem, og så bliver der checket om spillerens nuværende placering er X+4. Hvis det er korrekt, så er testen lykkedes.

```
@Test
public void action() {
    MovingRel movingRel = new MovingRel("Does It move", "If it move
        the test is complete", 4);
    Player player = new Player("jens", 100, 3);
    movingRel.action(player);

    assertEquals(7, player.getPosition());
}
```

Den sidste chancekort test bliver lavet i det næste afsnit omkring transfer af penge.

#### 6.1.4 Kan spillere modtage og betale penge?

For at kontrollere om en spiller kan modtage og betale penge, så laves der to test. Den test er under account test og den anden er under chance.

Under account testen bliver der lagt et negativt og et positivt beløb ind for at se om spillerens account går op og ned. I chancekort testen bliver der kun checket om den kan få mere.

Chancekort testen i "Transfer" bliver brugt på den måde, at der generes et chancekort som skal give en spiller 200 valuta mere i sin account. Så bliver der skabt en spiller med 200 valuta til at starte med i accounten. Derefter bliver spilleren tildelt chancekortet, og der kontrolleres om spilleren har en account med 400 valuta.

```
@Test
public void action() {
    Player player = new Player("jens", 200, 1);
    int money = player.getAccount().getBalance();
    Transfer transer = new Transfer("hej", "lol", 200);

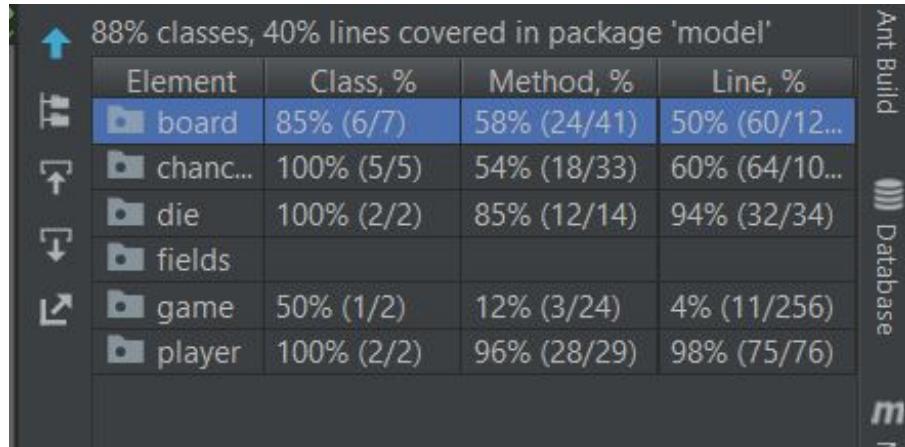
    transer.action(player);
    assertEquals(400, player.getAccount().getBalance());
}
```

## 6.2 Code coverage

For at sikre programmet virker som tiltænkt, så er der lavet nogle Junit tests som går igennem en stor del af koden samt de fleste klasser. Der er lavet code coverage for at sikre at metoder som programmet anvender virker på den tiltænkte måde. Med en code coverage på 60 procent af alle metoderne, kan man konkludere at programmet virker som ønsket.

### 6.2.1 Code coverage procenter:

- Klasser: 88 procent.
- metoder: 60 procent.
- linjer / kode: 40 procent



Figur 7: Code coverage

## 6.3 Bruger test

I brugeren testen bliver brugerne bedt om at spille et spil Monopoly junior på computeren. Brugerne bliver bedt sætte spillet til at indeholde 2 spillere, hvor spiller 1 skal være en Traktor, og spiller 2 skal være en UFO. Dette gør begge brugere individuelt. Herefter skal de to brugere spille spillet indtil dets konklusion.

Brugerne for startet spillet op som aftalt i ovenstående paragraf, og spiller spillet uden vanskeligheder. De anvender begge musen til at trykke "OK", og synes der manglede et overblik over hvilke grunde der var ejet af hvem. En vigtig kommentar fra brugerne er at de føler man skal klikke meget, og at teksten i GUI'en var alt for opbrudt.

Grundet tidspres har der ikke været tid til at ændre programmet, ud fra responsen fra brugeren testen, men havde vi haft tiden til det, havde vi forsøgt at tilpasse programmet til responsen.

## 7 Documentation

### 7.1 Arv

Arv er en af grundprincipperne i objektorienteret programmering. Objekter kan arve data og funktionalitet fra et andet objekt og udvide dem med ekstra data og funktionalitet, dvs, en eller flere klasser kan arve egenskaber (metoder og attributer) fra en anden klasse. Dette inddeltes strukturalt med "Superklasse" og en eller flere "Subklasser", hvilket kaldes nedarvning. Arv kan benyttes til at simplificere et design, når nogle klasser har en del til fælles, men stadig specialiserer sig på egen vis.

## 7.2 Abstract

Abstrakte klasser er en af grundprincipperne i objektorienteret programmering. Formålet er at håndtere kompleksiteten af at program ved at holde unødvendige detaljer fra brugeren, hvilket gør det muligt for brugeren af implementere endnu mere funktionalitet til en abstraction uden at skulle gengive al den gemte kompleksitet. Enhver klasse der indeholder abstrakte metoder kaldes abstrakte klasser, dette angives ved at skrive "abstract" foran klasse navnet. En abstrakt metode er bare et skelet for selve metoden. En abstrakt klasse kan ikke instantieres, dvs. alle instanser af abstrakte klasser er i virkeligheden instanser af subklasser til den abstrakte klasse. Abstrakte metoder SKAL implementeres i de nedarvede klasser.

## 7.3 LandOnField

At superklassen har en abstract metode (den behøver ikke være abstract, hvis subklasserne alle bruger samme parametre), som subklasser genbruger ved at have en metode i subklassen med samme navn som super klassen.

## 7.4 GIT-import og kørsel af program

For at køre dette program, så skal man have git, Java og et IDEA program installeret på sin computer. Det er rekommenderet at man bruger github til at hente Gruppe 15s CDIO3 Repository ned på sin computer, hvorefter man hiver det ind i et program såsom IntelliJ. Derefter burde maven blive opdateret pga. pom filen fra github, og så kan man køre Monopoly Junior spillet.

Nedenstående er en kort, punktlig opstilling at de krav, man skal opfylde for at kunne køre dette program.

1. Man skal have Java installeret.
2. Man skal have IntelliJ installed.
3. Man skal have github installeret og tilføjet sin IntelliJ.
4. Klon gruppe15 CDIO3 projekt ned i IntelliJ.
5. Opdater maven.
6. Kør projektet.

## 8 Projektforløb

Opgavens grundlæggende struktur følger Unified Process, jf. figur 7. Det er søgt at bruge GRASP-mønstre hele vejen igennem programmeringen. Det skal anerkendes, at Unified Process forløber over mange iterationer, som normalt hver især forløber over nogle uger - CDIO3 projektet har kun en varighed på i alt 3 uger. Derfor skal den nedenunder beskrevende process ses som en sammenstykning af CDIO1, CDIO2 og CDIO3, en process som alt i alt har været 8 uger lang.

Inception fasen er udført forud for projektets start idet opgaven og dennes business case er prædefineret.

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 af programmet, så arbejder gruppen 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 andre sideløbende branches. Der pushes kun til master branch når der er en ren, fungerende og sammenhængende kode.

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

## 9 Konklusion

Opgaven gik ud på at lave et Monopoly junior spil med en tilsvarende fyldestgørende rapport. Der er blevet udviklet et design til programmet med tilhørende diagrammer, som har været grundlag for implementationen. Programmet opfylder ca. 75 procent af spil kravene, hvor der har været fokus på de krav som har været nødvendige for at få spillet til at fungerer. Der er blevet lavet test for at sikre at programmet kører som tiltænkt. Der er koblet en GUI til programmet, som viser spillet til brugerne.

Der er lavet en github hvor programmet og rapporten ligger.

## 10 Bilag

### 10.1 Chancekort

Herunder beskrives de 20 chancekort, som skrevet i det originale monopoly junior spil.

- (C<sub>1</sub>) Giv dette til bilen, og tag endnu et chancekort. Bilen skal i sin næste tur drøne hen imod et selvvalgt ledigt felt og købe det. Hvis ikke der er ledige felter, så skal det købes fra en anden spiller.
- (C<sub>2</sub>) Ryk frem til start. Modtag 2 pengesedler.
- (C<sub>3</sub>) Ryk op til 5 felter frem.
- (C<sub>4</sub>) Gratis felt. Ryk frem til et orange felt. Er det ledigt, få det gratis. Ellers skal du betale leje til ejeren.
- (C<sub>5</sub>) Ryk 1 felt frem, eller tag et chancekort mere.
- (C<sub>6</sub>) Giv dette til bilen, og tag endnu et chancekort. Skibbet skal i sin næste tur sejle frem til et selvvalgt ledigt felt og købe det. Hvis ikke der er ledige felter, så skal det købes fra en anden spiller.
- (C<sub>7</sub>) Du har spist for meget slik. Betal 2 pengesedler til banken.
- (C<sub>8</sub>) Gratis felt. Ryk frem til et grønt eller orange felt. Er det ledigt, få det gratis. Ellers skal du betale leje til ejeren.
- (C<sub>9</sub>) Gratis felt. Ryk frem til et lyseblåt felt. Er det ledigt, få det gratis. Ellers skal du betale leje til ejeren.
- (C<sub>10</sub>) Du løslades fra fængsel uden omkostninger. Behold dette kort indtil du bruger det.
- (C<sub>11</sub>) Ryk frem til Strandpromenaden.
- (C<sub>12</sub>) Giv dette til katten, og tag endnu et chancekort. Katten skal i sin næste tur sejle frem til et selvvalgt ledigt felt og købe det. Hvis ikke der er ledige felter, så skal det købes fra en anden spiller.
- (C<sub>13</sub>) Giv dette til hunden, og tag endnu et chancekort. Hunden skal i sin næste tur sejle frem til et selvvalgt ledigt felt og købe det. Hvis ikke der er ledige felter, så skal det købes fra en anden spiller.
- (C<sub>14</sub>) Det er din b-dag. All giver dig 1 pengeseddels. Tillykke med dagen dawg!
- (C<sub>15</sub>) Gratis felt. Ryk frem til et pink eller mørkeblåt felt. Er det ledigt, få det gratis. Ellers skal du betale leje til ejeren.
- (C<sub>16</sub>) Du har lavet alle dine lektier. Modtag 2 pengesedler fra banken.
- (C<sub>17</sub>) Gratis felt. Ryk frem til et rødt felt. Er det ledigt, få det gratis. Ellers skal du betale leje til ejeren.
- (C<sub>18</sub>) Gratis felt. Ryk frem til skaterparken, Tony Hawk! Er det ledigt, få det gratis. Ellers skal du betale leje til ejeren.

- (C<sub>19</sub>) Gratis felt. Ryk frem til et lyseblåt eller rødt felt. Er det ledigt, få det gratis. Ellers skal du betale leje til ejeren.
- (C<sub>20</sub>) Gratis felt. Ryk frem til et brunt eller gult felt. Er det ledigt, få det gratis. Ellers skal du betale leje til ejeren.