


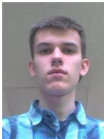



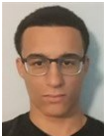


02312

INDLEDENDE PROGRAMMERING

# Final - Matador

## Hold A - Gruppe 17

s185118 Aleksander Lægsgaard Jørgensen			s175561 Andreas Østergaard Schliemann
s185096 Jonatan Amtoft Dahl			s164177 Josephine Weirøse
s185103 Søren Hother Rasmussen			s185121 Theodor Peter Guttesen

21. januar 2019

**DTU Compute**  
Institut for Matematik og Computer Science

**DTU Diplom**  
Center for Diplomingeniøruddannelse

Kode og Git: [https://github.com/theogutt/17\\_final?fbclid=IwAR2JyHDbw3X\\_xycIFZBxoZ3Mg8Vavisahl7rYh3FAX77LGILfQtRYw5tHsg](https://github.com/theogutt/17_final?fbclid=IwAR2JyHDbw3X_xycIFZBxoZ3Mg8Vavisahl7rYh3FAX77LGILfQtRYw5tHsg)

## Abstract

In this report we describe a Matador game, that we planned and programmed. Our plan was to create a digital version of the boardgame as close to the original as possible, though some features were left out in the end. This means we have a game board with 40 squares, that is illustrated with an GUI. In the finished game it is possible to trade ownables, build houses and hotels, and lots of other features. There are different square types such as Prison, Parking, Streets, Tax, Chance and so on.

The report showcases how the program works, in the form of diagrams and snapshots of the program. Furthermore this report contains the analysis from before the coding began, how we have designed the program and how it was implemented and the tests. The program is supposed to be object oriented, and follow the principles of GRASP. The Program is written in Java using IntelliJ.

# Indhold

<b>1</b>	<b>Indledning</b>	<b>3</b>
<b>2</b>	<b>Timeregnskab</b>	<b>4</b>
<b>3</b>	<b>Krav</b>	<b>5</b>
3.1	Vision . . . . .	5
3.2	Kravliste . . . . .	5
3.3	Domænemodel . . . . .	8
<b>4</b>	<b>Analyse</b>	<b>9</b>
4.1	Aktører . . . . .	9
4.2	Use case . . . . .	9
4.3	Traceability matrix: Use cases og krav . . . . .	14
4.4	System sekvensdiagram . . . . .	16
<b>5</b>	<b>Design</b>	<b>17</b>
5.1	Klassediagram . . . . .	17
5.2	Sekvensdiagram . . . . .	19
<b>6</b>	<b>Implementering</b>	<b>20</b>
6.1	Grasp . . . . .	20
6.2	View, model, controller . . . . .	21
6.3	Fraskåret funktioner . . . . .	21
6.4	Bugs . . . . .	22
6.5	Hashmaps . . . . .	22
<b>7</b>	<b>Test</b>	<b>23</b>
7.1	Positiv test . . . . .	23
7.2	Negative test . . . . .	24
7.3	Unit-test . . . . .	25
7.4	Coverage test . . . . .	25
<b>8</b>	<b>Projektplanlægning</b>	<b>26</b>
8.1	Planlagt forløb . . . . .	26
8.2	Reelt forløb . . . . .	26
<b>9</b>	<b>Konklusion</b>	<b>28</b>
<b>10</b>	<b>Bilag</b>	<b>29</b>

## 1 Indledning

I dette projekt har vi lavet et matador spil, som skulle være så tæt på det virkelige brætspil som muligt, vi har dog måtte udlade nogle forskellige features for at kunne nå tidsfristen. Vores matador spil er sat op med 40 felter, der bliver vist på et bræt ved hjælp af en GUI. Vores spil har de mest essentielle dele af matador, heriblandt er f.eks. chancefelter, gader, færger, fængsel felt og mange andre. En normal tur fungerer mere eller mindre som en tur i det rigtige spil, først slår man med terningerne, derefter kan spillere lave handelsaftaler mellem hinanden, eller købe huse og hoteller på deres ejede gader. Selve programmet er lavet ud fra krav, som er vurderet med forskellig prioritet, derudover har programmet også use cases og sub use cases. Vi illustrerer også programmets struktur med diagrammer, såsom designklasse diagram og sekvens diagram. Projektet er designet med fokus på at være objekt orienteret, og på at følge GRASP principperne. Programmet er skrevet med Java som programmeringsprog og IntelliJ som udviklingsprogram.

## 2 Timeregnskab

Opgave	Josephine	Aleksander	Søren	Jonatan	Andreas	Theodor	Sum
Generelt LaTeX	6	0	0	0	0	0.75	<b>6.75</b>
Timeregnskab	1.5	0	0	0	0	0	<b>1.5</b>
Abstract og indledning	0.5	0	0	2	0.35	0	<b>2.85</b>
Vision	0.5	0	0	0	0.5	0	<b>1</b>
Krav	3.25	1.75	3.5	1	2.25	1.75	<b>13.5</b>
Sekvensdiagram	0.5	0	0	0	0	2.25	<b>2.75</b>
Design klassesdiagram	0	6	0	0	2.75	3.5	<b>12.25</b>
Domænemodel	1.5	0	1.5	0	0	0	<b>3</b>
System Sekvensdiagram	0	0	2	3	0	0	<b>5</b>
Aktører	0	0	0	1	0	0	<b>1</b>
Use case	5.5	0	0	4	0	4.5	<b>14</b>
Projektplanlægning	1	0	0	2	0	0	<b>3</b>
Konklusion	0	0	0	3	0	0	<b>3</b>
Implementering	0	0	0	2	6	0	<b>8</b>
Test	7.5	0	4	6	0	5.25	<b>22.75</b>
Kodning	23	44	41.25	30	40	28.5	<b>206.75</b>
Gennemgang og rettelser	2	0	0	4	1.75	0	<b>7.75</b>
<b>Sum</b>	<b>52.75</b>	<b>51.75</b>	<b>52.25</b>	<b>58</b>	<b>53.6</b>	<b>46.5</b>	<b>314.85</b>

## 3 Krav

### 3.1 Vision

Kunden vil gerne have et Matador spil, hvor det prioriteres at have et fejlfrit produkt, så kunden vil hellere have et lille produkt, der kører fejlfrit end et produkt, hvor mange features er halvt lavet færdig. Spillet skal kunne spilles mellem tre til seks spillere, som kan bevæge sig rundt på en spilleplade bestående af 40 felter, ved at slå med to terninger. Når en spiller når det sidste felt på brættet, forsætter spillerne med at bevæge sig rundt på pladen, indtil at en af spillernes pengebeholdning bliver negativ.

#### Felter

Der er overordnet 6 forskellige typer felter:

1. **Start:** Spillere modtager 4.000 kr. hver gang de passere startfeltet.
2. **Chance:** Når en spiller lander på en chance felt, trækker de et chance kort, hvis effekt derefter bliver udført.
3. **Gå i fængsel:** En spiller kan ryge i fængsel og ende på "I fængsel" feltet, hvis de lander på "Gå i fængsel" feltet, slår tre parslag i træk eller trækker et gå i fængsel chancekort. Man kan komme ud af fængslet, hvis man har et kom ud af fængsel chancekort, slår et parslag eller betaler 1.000 kr., hvorefter turen fortsættes som normalt.
4. **På besøg/Gratis parkering:** Der sker ikke noget, når en spiller lander her.
5. **Grunde:** Der er 3 typer af grunde, som spillerne kan købe, hvis de ikke allerede er ejet disse er, gader, færger og bryggerier, der hver har sin måde at udregne husleje på. Det er kun gader, hvor man kan bygge huse og hoteller på, husene og hotellerne får husleje til at stige.
6. **Skattefelter:** Der er to skattefelter, indkomstskat hvor man betaler 4000 eller 10% af sine samlede værdier. Ekstraordinærstatsskat, hvor der betales 2000.

### 3.2 Kravliste

ID	Funktionelle krav
<b>Must have</b>	
M01	Spillet skal spilles mellem 3-6 personer
M02	En spiller skal starte med 30.000 kr.
M03	En spiller slår med to terninger og bevæger sig det antal øjne der står på terningerne
M04	En spiller skal modtage 4.000 kr. når spilleren passerer start
M05	En spiller der standser på en ikke-ejet grund eller virksomhed, kan vælge at købe den af banken.
M06	En spiller der standser på et "prøv lykken"-felt skal trække et lykkekort
M07	En spiller skal i fængsel, hvis spilleren standser på feltet "De sættes i fængsel"
M08	En spiller der standser på feltet "fængsel", er blot på besøg
M09	En spiller kan komme ud af fængslet, ved at kaste 2 af samme slags, spilleren rykker det øjnene viser og har stadig ekstrakast.
M10	En spiller, som lander på et ejet felt skal betale husleje til spilleren, der ejer feltet.

M11	En spiller, som lander i fængsel får IKKE udbetalt 4.000 kr for at passere start.
M12	Spillet stopper, hvis en spiller løber tør for penge.
M13	Ejer man alle grunde med samme farve, får man dobbelt leje på ubebyggede grunde.
M14	Bygges der hus på en grund, øges lejen af denne grund
M15	Ejer man alle grunde med samme farve, har man ret til at bygge huse, til den pris der står på skøderne
<b>Should have</b>	
S01	En spiller der standser på feltet indkomstskat, har lov at betale 4.000kr, men kan vælge at betale 10% af sine værdier(kontanter, bygninger og den trykte pris for grunde og virksomheder også pantsatte), men spilleren skal vælge betalingsmåden, inden spilleren tæller sine værdier sammen.
S02	En spiller kan komme ud af fængslet ved at betale 1.000 kr. inden spilleren kaster.
S03	Hvis en spiller kaster 2 af samme slags, skal spilleren standse på et felt og derefter kaste igen.
S04	En spiller kan komme ud af fængsel, ved at benytte et løsladelseskort.
S05	En spiller kan ikke blive i fængslet mere end 3 omgange, får man ikke to af samme slags, når man kaster tredje gang, skal man betale 1.000 kr. og flytte det som øjnene viser.
S06	Indbyrdes handel med ubebyggede grunde er tilladt, til den pris spillerne kan blive enige om.
S07	Man må bygge hoteller, når man har bygget 4 huse på en grund.
S08	Prisen for et hotel skal være fem gange det af et hus.
S09	Spillet stopper når alle på nær en spiller er løbet tør for penge.
S10	Spillere skal kunne handle mellem hinanden med ejendomme og penge.
<b>Could have</b>	
C01	Man skal bygge jævnt; altså må man bygge de første huse, på de grunde man ønsker. Man må først bygge hus nr.2, når man har bygget et hus på hver af de andre grunde i gruppen og så videre.
C02	Hvis en spiller kaster 2 af samme slags 3 gange i træk, skal spilleren fængsles
C03	Spilleren kan sælge grunde tilbage til banken
C04	Har man bygget skal man sælge bygningerne tilbage til banken, inden grunden sælges.
C05	Har banken ingen bygninger, må man vente til der kommer nogle tilbage.
C06	Man kan pantsætte grunde og få den halve værdi udbetalt.
C07	Lander en spiller på en pantsat grund, betales der IKKE leje.
C08	Når en pantsætning ophæves betales den originale pantsætningsværdi plus 10%
C09	Alle huse på en grund skal sælges før den kan pantsættes
C10	Skylder en spiller flere penge end han ejer til banken, sælges alle ejede grunde på auktion, og spilleren går ud af spillet.
C11	Lykkekorts bunken skal være i række følge; altså en spiller der standser på et "prøv lykken"-felt skal tage det øverste lykkekort, når det er benyttet skal det ligges nederst i bunken

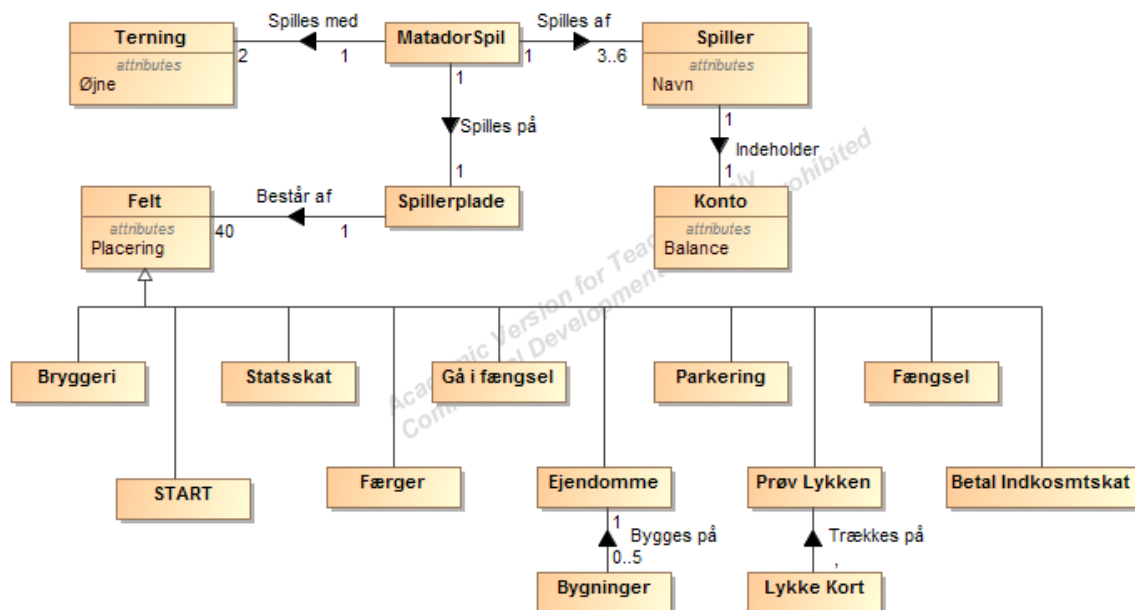
C12	Hvis en spiller ikke køber grunden spilleren lander på, skal grunden på auktion (Auktion fungerer således, at en ejendom sættes til auktion ved halv pris. Derefter byder spillere på tur indtil ingen vil byde mere.)
C13	Byder ingen på en auktion forbliver ejendommen uejet.
C14	En bygning kan til hver en tid sælges tilbage til banken ved halv pris.
<b>Won't have</b>	
W01	Skylder en spiller flere penge end han ejer til en spiller, skal han overdrage alt han ejer til sin kreditor efter at have solgt eventuelle bygninger til banken, og han går ud af spillet.

<b>ID</b>	<b>Ikke-funktionelle krav</b>
<b>Must have</b>	
IF01	Systemet skal have en GUI
IF02	Systemet skal kunne virke på DTUs databaser



### 3.3 Domænemodel

Denne domænemodel viser sammenhængen mellem de vigtigste elementer i spillet Matador. Denne bruges til inspiration, når klasserne skal laves i vores program og for at se hvordan klasserne skal arbejde sammen. Spillet Matador spilles af 3-6 spillere, som hver har en pengebeholdning. Spillet spilles med 2 terninger, som bestemmer hvor langt en spiller rykker på en spilleplade. Spillepladen består af 40 forskellige felter, hvor nogle af felterne kan have bygninger på sig og ved andre trækker man et Lykkekort. Alt dette kan ses på figur 1



Figur 1: Domænemodel over Matador

## 4 Analyse

### 4.1 Aktører

I systemet har vi de følgende aktører:

1. Bruger
2. Kunde

I dette program har vi kun 2 aktører. Bruger aktøren referere til de spillere, som fysisk spiller spillet, de har naturligvis en interesse i programmet, da spilleren aktivt bruger spillet. Da spillet bliver brugt aktivt af brugerne, så er brugerne primære aktører, som direkte bruger de forskellige use cases i systemet. Programmets anden aktør er kunden, som har bestilt spillet. Kunden er en offstage aktør, grunden er at kunden ikke har direkte forbindelse til projektet, men har nogle interesser i det som f.eks. reklame.

Aktørtype	Aktører
Primære	Bruger
Supporterende	-
Offstage	Kunde

### 4.2 Use case

I programmet er der en hoved use case "Spil Matador". Under denne er der til gengæld en del sub-use cases, som alle beskriver handlinger en spiller kan foretage sig under spillet.

#### Sub-use cases

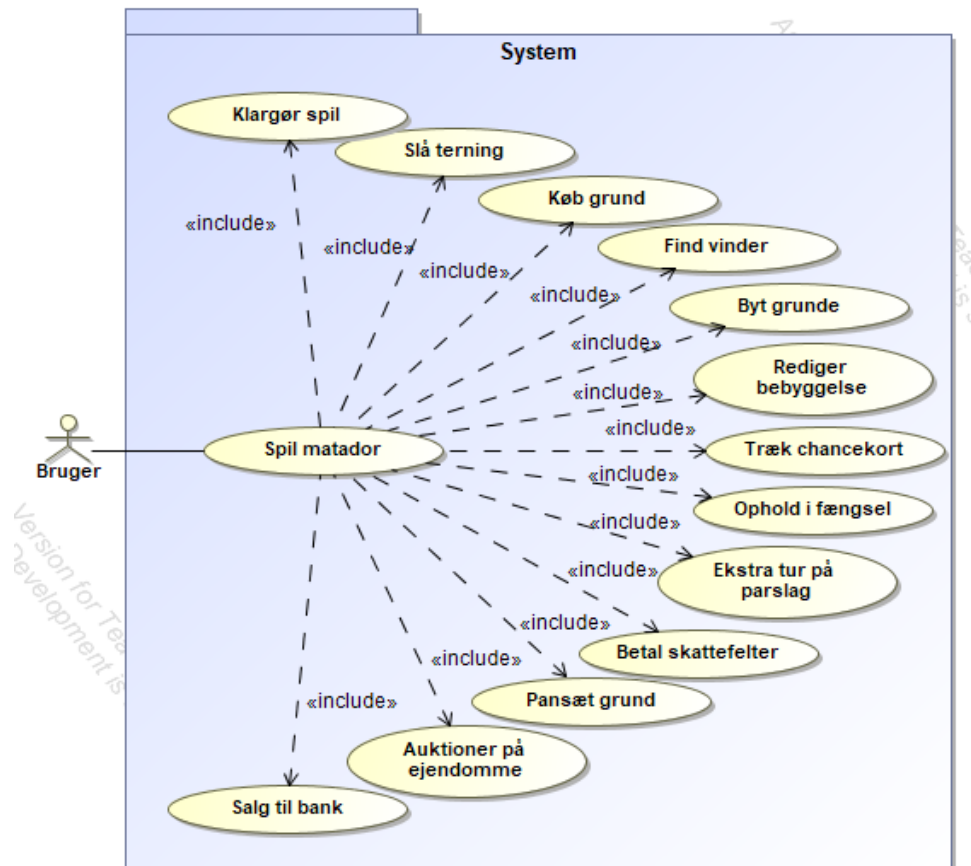
For at danne overblik over de forskellige krav, er der blevet lavet en sub-use case liste, som samler kravene i grupper. På denne måde kan vi sætte disse sub-use cases i rækkefølge efter hvornår de skal laves med de vigtigste sub-use cases øverst.

1. Klargør spil
2. Slå terninger
3. Køb grund
4. Find vinder
5. Byt grunde
6. Rediger bebyggelse
7. Træk chancekort
8. Ophold i fængsel
9. Ekstratur på parslag
10. Betal skattefelter
11. Pantsæt grund
12. Auktioner på ejendomme
13. Salg til bank

## Use case diagram

I det følgende use case diagram illustreres der, hvordan brugeren interagerer med use casen "Spil Matador" og hvordan denne inkluderer alle sub-use case'ne.

Aktøren, kunden, er ikke med i use case diagrammet, da kunden er en offstage aktør, som i dette tilfælde ikke bruger nogle af de use cases, som systemet indeholder. Altså er vores use case diagram blevet meget simpelt, da der kun er en aktør, som benytter sig af alle programmets use cases.



Figur 2: Use case diagram

## Fully dressed

Ved arbejde med use cases kan der benyttes tre forskellige grader af beskrivelse. Her er der tale om "brief", "casual" og "fully-dressed". Brief er den mindst omfattende og fully-dressed er den mest omfattende.

For use casen "Spil matador" samt de to sub-use cases "klargør spil" og "Find vinder" er der blevet lavet en fully dressed.

"Spil matador" er den bærende use case, hvor alle sub-use cases inkluderes.

Vi har derfor valgt ikke at indskrive de to sup-use case's mainflow i den fully dressed beskrivelse af "Spil matador". Derfor er de to sup-use case's mainflow, ikke indskrevet i mainflowet af "Spil matador", da det er underforstået.

<b>Use case:</b> Klargør spil
<b>ID:</b> SUC01
<b>Kort beskrivelse:</b> Spillerne forbereder spillet
<b>Primære aktører:</b> Spiller
<b>Preconditions:</b> 3-6 personer vil deltage i spillet.
<b>Main flow:</b> 1. Reglerne vises. 2. Antal spillere indtastes. 3. Spillernes navne indtastes. 4. Spillernes biler vælges. Trin 3 og 4 gentages for alle spillere.
<b>Postconditions:</b> Den første spiller er klar til sin tur

<b>Use case:</b> Find vinder
<b>ID:</b> SUC04
<b>Kort beskrivelse:</b> Der findes en vinder og spillet afsluttes.
<b>Primære aktører:</b> Spiller
<b>Sekundære aktører:</b> Ingen
<b>Preconditions:</b> Spillet er startet
<b>Main flow:</b> 1. Alle spillere på nær en er gået fallit. 2. Spillet udnævner vinderen.
<b>Postconditions:</b> Spillet er afsluttet
<b>Alternative flows:</b>

<b>Use case:</b> Spil matador
<b>ID:</b> UC01
<b>Kort beskrivelse:</b> Spillerne kaster med terningerne og rykker deres bil til nye felter. Alt efter feltet påvirkes spilleren.
<b>Primære aktører:</b> Spillere
<b>Sskundære aktører:</b> Ingen
<b>Preconditions:</b> Se UC01
<b>Main flow:</b> 1. Spilleren kaster med terningerne. 2. Ud fra terningernes øjne og spillerens position udregnes den nye position. 3. Spillerens brik flyttes til det nye felt. 4. Alt efter feltet bestemmer spillet hvad der sker. 5. Spillet beregner spillerens pengebeholdning og viser den. 6. Spillerens tur slutter og den næste spiller får sin tur. Trin 1-6 gentages indtil der er fundet en vinder.
<b>Postconditions:</b> Spillet ender når alle undtagen en er gået fallit.
<b>Alternative flows:</b> *a Spilleren kan ikke betale med sin pengebeholdning og må pantsætte ejendomme og sælge huse og hoteller. Hvis spilleren stadig ikke kan betale, taber spilleren. 1.1 Spilleren er i fængsel og har tre slag til at få to ens og blive løsladt. 1.1.1 Spilleren bruger sit løsladelseskort og løslades. 1.1.2 Spilleren betaler 1000kr og løslades. 1.2 Spilleren køber huse og eller hoteller til sine ejendomme. 4.1 Spilleren lander på en ikke-ejet ejendom. Spilleren køber ejendommen 4.1.1 Spilleren køber ikke ejendommen. De andre spillere holder en auktion om ejendommen 4.2 Spilleren lander på en ejet ejendom. Spilleren betaler husleje til ejeren. 4.2.1 Spilleren ejer selv ejendommen og skal derfor ikke betale noget. 4.2.2 Ejeren af ejendommen ejer alle felter med samme farve. Huslejen fordobles. 4.2.2.1 Der er huse eller et hotel på ejendommen. Spilleren betaler husleje i overensstemmelse med den specificerede leje. 4.2.3 Spilleren lander på en pantsat ejendom og skal derfor ikke betale husleje. 4.3 Spilleren lander på et chance-felt og trækker et chancekort. 4.4 Spilleren lander på indkomsskat-feltet og betaler 4000kr. 4.4.1 Spilleren vælger i stedet at betale 10% og udregner sin formue og betaler. 4.5 Spilleren lander på statsskat-feltet og betaler 2000kr.

- 4.6 Spilleren passerer start og spilleren får 4000kr.
- 4.7 Spilleren lander på gratis parkering og der sker ikke noget.
- 4.8 Spilleren lander på "På besøg i fængsel" og der sker ikke noget.
- 4.9 Spilleren lander på "De fængsles" og ryger i fængsel.
- 4.10 Spilleren lander på et ikke-ejet bryggeri og kan vælge at købe det.
- 4.10.1 Spilleren lander på sit eget bryggeri og skal ikke betale.
- 4.10.2 Spilleren lander på et ejet bryggeri og skal betale det han slog med terningen gange 100kr.
- 4.10.3 Ejeren af bryggeriet ejer også det andet bryggeri. Spilleren skal betale det dobbelte.
- 4.11 Spilleren lander på en ikke-ejet færge og kan vælge at købe den.
- 4.11.1 Spilleren lander på sin egen færge og skal ikke betale.
- 4.11.2 Spilleren lander på en ejet færge og skal betale 500kr.
- 4.11.3 Ejeren af færgen ejer også 1-3 andre færger. Spilleren skal betale 1000kr, 2000kr eller 4000kr.
- 6.1 Spilleren havde slået to ens og må rykke til et nyt felt
- 6.1.1 Spilleren har slået to ens tre gange i træk og ryger i fængsel

### 4.3 Traceability matrix: Use cases og krav

I følgende tabel ses sammenhængen mellem use cases og krav, de grønne er dem der er implementeret, de røde er endnu ikke implementeret.

Krav	Use Case	UC01	SUC01	SUC02	SUC03	SUC04	SUC05	SUC06	SUC07	SUC08	SUC09	SUC10	SUC11	SUC12	SUC13
M01			x												
M02		x													
M03					x										
M04		x													
M05						x									
M06									x						
M07										x					
M08										x					
M09										x	x				
M10					x										
M11										x					
M12						x									
M13					x										
M14								x							
M13								x							
S01												x			
S02										x					
S03											x				
S04										x					
S05										x					
S06							x								
S07								x							
S08								x							
S09						x									
C01								x							
C02										x					
C03															x
C04								x							x
C05								x							
C06													x		
C07													x		

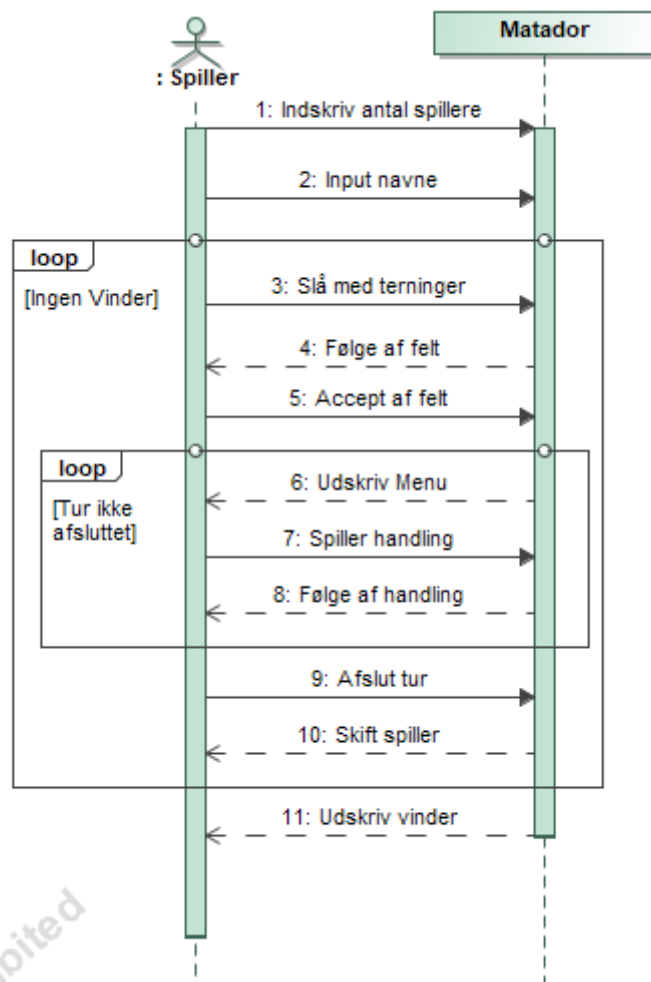
C08														x		
C09														x		x
C10															x	
C11								x								
C12			x												x	
C13															x	
C14																x
W01																
W00																



#### 4.4 System sekvensdiagram

System sekvensdiagrammet beskriver kommunikationen mellem spillere og programmet. I starten af spillet indskrives spilleren antallet af spillere og derefter gives navnene. Herefter kommer der et loop, hvori en spillers tur bliver udført. I slutningen af loopet sker der en ændring af spillerens information. Dette loop indeholder et andet loop, som illustrer de valgmuligheder en spiller har efter hver runde, og at spilleren kan vælge at gøre flere af de ting per runde. Grundlæggende er dette hvordan programmet forløber igennem hele spillet med mindre variationer.

I selve spillet er der mange alternative routes spillet kan køre imod. Spillere kan lande på felter, som har forskellige konsekvenser for spillet. Som følge af dette har vi lavet system sekvensdiagrammet meget overordnet med linjer, såsom *Følge af felt*, som ikke specificere, hvordan feltet faktisk reagere. Da feltets outcome kan have en meget stor ændring i programmets besvarelse, så har vi skrevet det således, at diagrammet illustrere det mest generale forløb af programmet for alle scenarier.



Figur 3: System sekvensdiagram

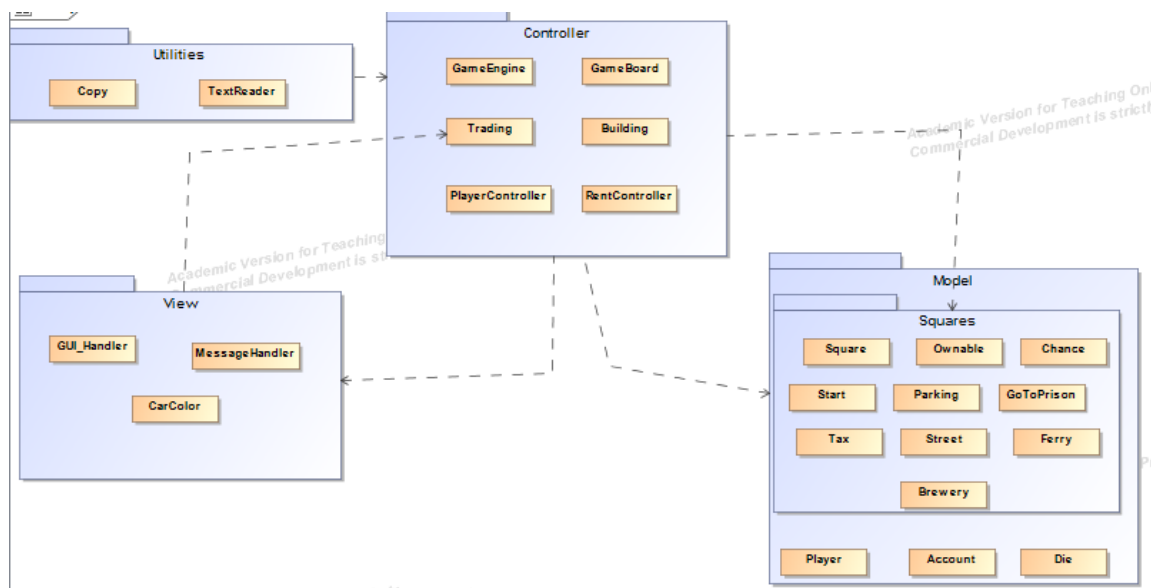
## 5 Design

### 5.1 Klassediagram

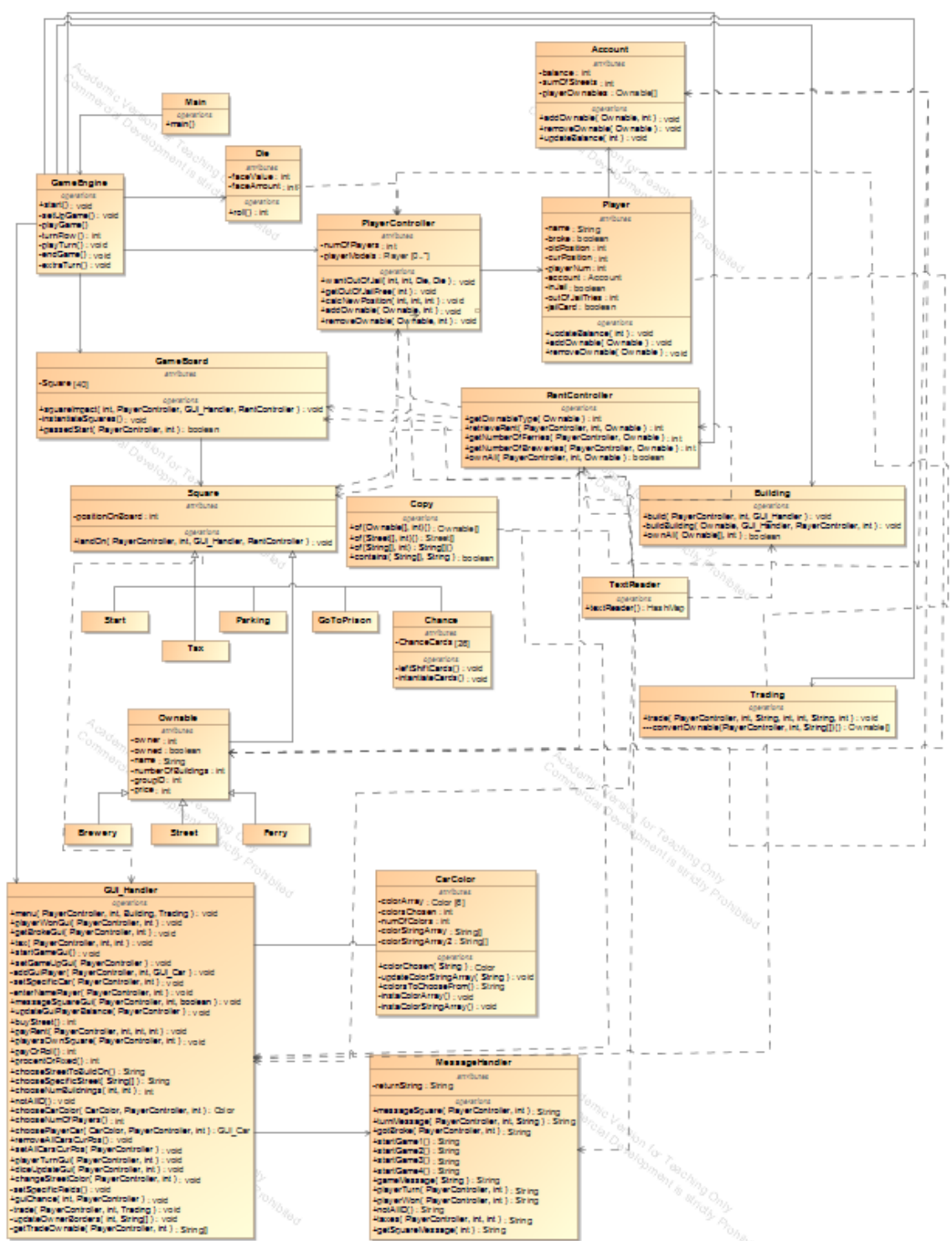
I designklassediagrammet, får man det store overblik over programmet. Designklassediagrammet er inspireret af domænemodellen. Et designklassediagram er et statisk UML-diagram, der omfatter klassernes navne og deres metoder og attributter. Desuden vises klassernes relationer også. Dette design klassediagram er lavet efter implementeringen, så det viser de reelle klasser, metoder og sammenhænge mellem klasser.

#### Pakkediagram

Her er pakkediagrammet, som giver overblik over hvilke pakker indenfor MVC klasserne tilhører.



Figur 4: Pakkediagram



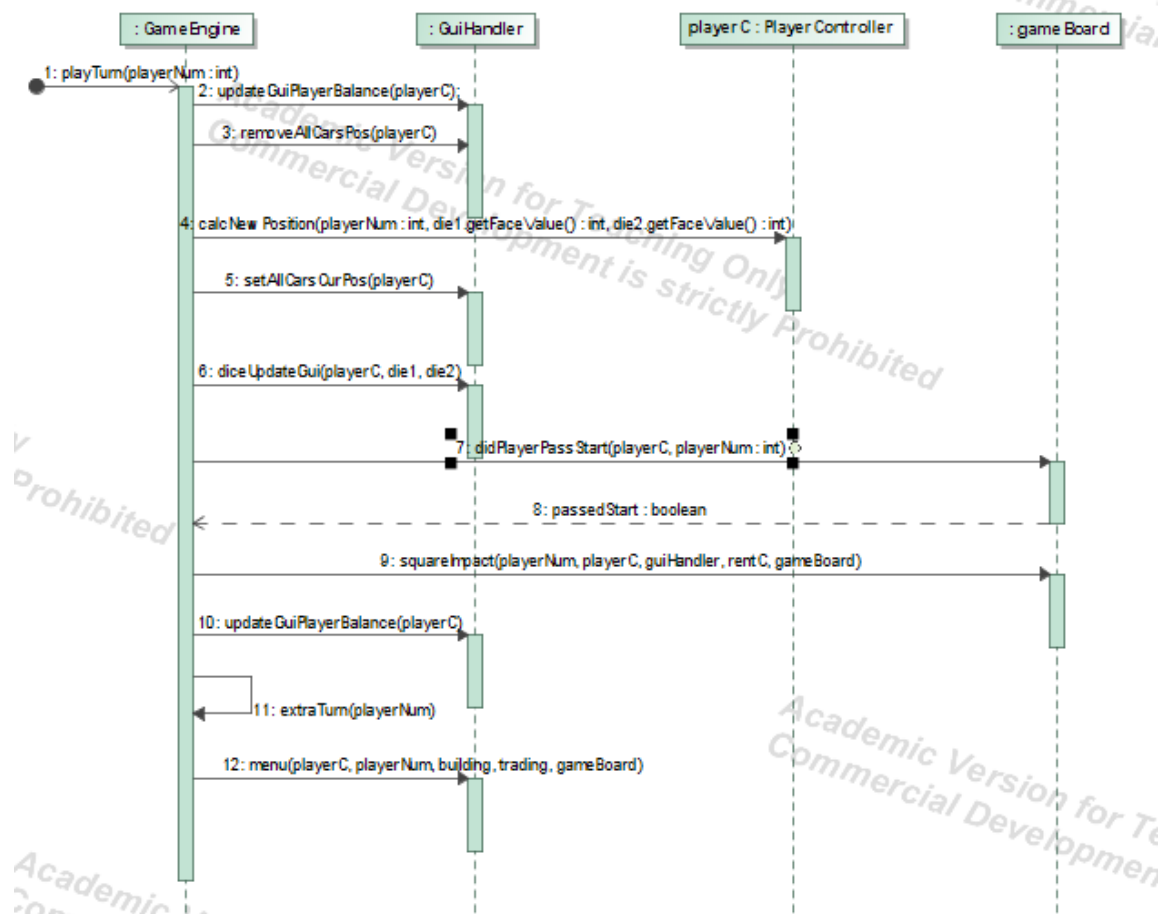
Figur 5: Design klassediagram

For at se større billeder se under bilag figur 24-26

## 5.2 Sekvensdiagram

Sekvensdiagrammet beskriver en central metode i vores spil nemlig `playTurn()`, som ligger i klassen `GameEngine`. `PlayTurn()` bestemmer forløbet i en typisk tur, derfor eksekvere der en række metoder, som påvirker spilleren.

Først opdateres spillerens balance, i forhold til GUI'en. Så fjernes spillerens bil, fra hvor den stod før i GUI'en. Så udregnes den nye position og bilen sættes på den nye position i GUI'en. Så opdateres terningerne i forhold til GUI'en. Og så videre.



Figur 6: Sekvensdiagram

## 6 Implementering

Udviklingsplatformen vi har benyttet os af, er Java, hvor koden er skrevet i IntelliJ, derudover har vi også benyttet os af den udleverede GUI, som vi har hentet fra Maven. Under udviklingsforløbet har vi mødt nogle udfordringer, hvilket har resulteret i, at ikke alle krav er blevet implementeret. Til nogle af kravene har vi lavet en alternativ løsning, mens andre slet ikke er blevet implementeret.

### 6.1 Grasp

Vores program er opbygget rundt om GRASP principperne, og vi har fulgt disse principper så godt som muligt. Dette har gjort, at programmet er blevet mere overskueligt, og at fejl vi har fundet i koden, har været isoleret til mindre dele af programmet.

#### High cohesion

Vi har sørget for, at vores klasser har et veldefineret ansvar og dermed fået high cohesion. Dette har gjort, at vi har lavet flere klasser, end hvad vi først regnede med, hver af disse klasser står for et specifikt område. Hvis man kigger på systemets controllere, så ses det, at hvert af disse klasser har et ansvarsområde. PlayerController metoden er et godt eksempel på dette, klassen sørger for at sende alt information rundt om spillerne. De "information expert"klasser som holder på informationen i programmet, har hver især kun viden om en mindre del af programmet, dette kan være information om f.eks. et slags felt eller spillernes kontoer. Dog har vi også GUIHandler klassen, som er klassen der arbejder med GUI'en, den har et meget stort ansvarsområde og er som konsekvens af dette over 600 linjer lang. Klassen kan godt blive delt ud på mindre klasser, som alle har deres eget mindre ansvarsområde, disse ansvarsområder kunne være input fra bruger eller opdatering af spillebrættet.

#### Low coupling

Low coupling lagde vi meget fokus på i udviklingen af vores program. Vi har prøvet at samle information fra forskellige klasser sammen, og få controllerne til at sende den information rundt. Under udviklingen af programmet havde vi et problem med landOn metoderne. Da landOn metoderne er nedarvet, og nogle af felterne skulle bruge forskellig information fra forskellige klasser, så blev sammenhængen mellem de klasser og controllerne meget stor. Derfor lavede vi om på koden, så vi tilsidst kunne fjerne nogle referencer, og dermed fjerne noget coupling fra metoderne og dermed også klasserne.

#### Creator

Vi har fordelt ansvaret for at lave objekter ud på flere klasser, og vi har typisk tildelt ansvaret for at lave instanser til controllerne. PlayerController klassen står for at lave alle Player objekter, klassen har dette ansvar, da PlayerControlleren står for at sende informationen rundt om de spillere. Det samme sker med GameBoard controlleren som laver instanser af alle felterne, og sørger for at lave det rigtige slags objekt til hver plads på boardet.

#### Information expert

En information expert klasse er mere eller mindre det samme som model fra MVC, vi har de klasser som har informationen i programmet i model pakken. De holder på den information som de forskellige objekter har. Her er f.eks. klassen Account som holder styr på et Player objekts pengebalance.

#### Controller

Controllerne i vores programs største fokus er at sende information rundt imellem klasser, de har også et ansvar for at lave beregninger. Som forklaret nedenfor har vi nogle forskellige controllere med forskellig

ansvarsområde. Til at håndtere input fra brugeren bruger vi GUI.Handler klassen. Den tager inputtet fra spilleren, og sender den information rundt til controllerne eller bruger selv inputet.

## 6.2 View, model, controller

Vi har inddelt vores program i view, controller og model og opdelt vores klasser i tilsvarende pakker. I view pakken har vi vores GUI.handler, der får information fra diverse controllere, som den bruger til at opdatere GUI'en. Det er GUI.handler, som sætter bilikonerne på brættet, der hvor spillerne rent faktisk er, sætter huse og hoteller og viser information og alle felterne på brættet. I virkeligheden burde den GUI som importeres fra Maven være det eneste i view, da det er den, som direkte spørger brugeren om information. På den måde er GUI.Handler mere en controller, men da vi ikke har direkte adgang til at ændre i Maven-GUI'ens source code, lader vi GUI.Handler være view, da vi ellers ikke ville have nogen view. MessageHandler kunne også være i controller, da den henter data fra et tekstdokument og sender det tilbage som String. Til sidst kunne CarColor være i model, da den indeholder farverne til bilerne.

I controller pakken har for eksempel PlayerController, GameBoard og RentController. Vores PlayerController holder styr på spillerne og deres konti, for eksempel ved at sætte spillerne position på brættet og opdatere deres balance på deres konti. Derudover sørger PlayerControlleren også for at få spillerne ud af fængslet. GameBoard klassen står for at instantierer alle vores Square objekter, metodekald for de specifikke squares, når en spiller lander på dem, og se om en spiller har passeret start. Klassen RentController udregner huslejen for de forskellige ejendomme, ud fra vores hashmaps, og tjekker om ejendommen er en gade, færge eller bryggeri.

I vores model pakke har vi de klasser, som indeholder diverse informationer, for eksempel vores Player og Account klasser. Vores Player klasse indeholder en masse variabler, såsom currentPosition, spillerens aktuelle position på brættet, og inJail, en boolean, som PlayerControlleren bruger til at se om en spiller sidder i fængsel. Account klassen indeholder for eksempel variabelen balance, som er en spillers pengemængde, og et array af alle en spillers ejede ejendomme. Resources er også en del af model, men ligger ikke i Model pakken, da vi gerne ville separere klasser og andre dokumenter.

## 6.3 Fraskåret funktioner

Grundet tidspress valgte vi ikke at implementere nogle sub-use cases, for at bruge tiden på at få de andre til at fungere. Ret tidligt i projektforsløbet besluttede vi at udelade auktioner på ejendomme, da vi kunne se, at den ville tage en del tid at implementere og ikke var absolut nødvendig for at kunne spille spillet. Men, hvis vi havde haft tid, kunne det fungere ved, at efter en ejendom var blev sat på auktion, ville alle spillere, på skift, forhøje budet på ejendommen, med et minimum på 50 kr. eksempel. Hvis en spiller synes budet er for højt, ville de kunne vælge at forlade auktionen. En spiller ville ikke kunne byde flere penge, end de har på deres konto. Hvis budet bliver højere end, hvad en spiller har på deres konto, vil denne spiller automatisk forlade auktionen. Når der tilsidst kun er en spiller tilbage, vil den spiller købe ejendommen for, hvad end det sidste bud er.

Mod slutningen af projektforsløbet besluttede vi udelade pantsætning af ejendomme. Pantsætning kunne fungere ved, at efter en ubebygget ejendom var blevet pantsat, ville ejeren modtage penge svarende til halvdelen af købsværdien for ejendommen. Hvis en anden spiller derefter langer på den pantsatte ejendom, ville de ikke skulle betale lejen. Ejeren af den pantsatte ejendom ville kunne hæve pantsætningen ved at betale det beløb, de fik udbetalt ved pantsætningen med en rente på 10%. Hvis en pantsat ejendom bliver solgt til en anden spiller, ville de skulle hæve pantsætningen med det samme eller betale 10% i rente oven i at skulle betale hele hævnningen senere.

## 6.4 Bugs

I vores færdige program er alle de fejl, som er fundet blevet rettet. Derfor skulle programmet være fejlfrit i de funktioner programmet har. Dog har der under implementeringen været nogle udfordringer, som senere er blevet løst. Disse kan ses nedenfor.

### Indkomst skat

Feltet "Betal indkomstskat eller 4.000 kr." fungerede ikke helt som det burde, ifølge Matador reglerne, i vores program. Dette var fordi, vi ikke havde lavet en metode til at udregne den samlede værdi af alle en spillers ejendomme, bygninger og pengebeholdning. Dette er blevet løst i en senere version, hvor getFortune metoden er blevet implementeret. Denne metode udregner den samlede værdi af en spiller.

### Salg af ejendom med huse

I en tidligere version af vores program var det muligt at bytte en ejendom, selvom der stadig er bygninger på den eller en anden ejendom af samme farve, hvilket Matador reglerne siger, at man ikke må. Dette har vi valgt ikke at fixe, da spillet stadig fungerer som normalt. Alle bygninger på den nyligt byttet ejendom bliver stadig ejendommen, både i koden, men også på GUI. Efter byttehandlen er det den nye ejer, der får huslejen, også den forhøjede husleje tilsvarende til alle de bygninger, der kom med i byttehandlen.

## 6.5 Hashmaps

I vores program har vi benyttet os af hashmaps, som vi bruger til at opbevare priser og tekst beskeder. Hashmaps fungerer ved man laver et register i et tekstdokument, hvorefter man kan kalde de informationer, der står på en bestemt linje i tekstdokumentet. For eksempel, hvis man vil have prisen for Rådhuspladsen, kan man kalde hashmapet squarePrice index 39 samt angive, at det er en integer man vil have fat på. Hvis man derimod vil have navnet for Rådhuspladsen, kan man kalde hashmapet StreetName index 39 og angive, at det nu er en string man vil have fat i. Hashmaps er også brugbare, hvis ens program skal have flere sprog, idet man kunne lave en metode, der bestemmer hvilket set af hashmaps programmet skal bruge, uden at skulle gå ind og ændre i sourcecoden. Dette har vi dog ikke benyttet os af i vores program, da det ikke var en del af kravene.

## 7 Test

### 7.1 Positiv test

Funktionelle krav	Positiv test	Bestået
M01	På bilag 8 kan det ses, at spillet skal have 3 til 6 spillere.	Bestået
M02	Hver spiller starter med 30.000 kr. som det ses i bilag 9.	Bestået
M03	Som vist på bilag 10, Spiller slår med terningerne og spilleren flytter sig den distance.	Bestået
M04	En spiller lander på eller passere start, modtager spilleren 4.000 kr. se bilag	Bestået
M05	Som det ses på bilag 11 ses det at når en spiller landet på en uejet grund, kan man vælge imellem at købe den eller ikke at gøre noget.	Bestået
M06	Som det ses på bilag 12 har spiller Bob landet på et chancefelt, og derfor trukket et chancekort.	Bestået
M07	Når en spiller lander på feltet "De fængslet" så kommer spilleren i fængslet, der er også et chancekort som har samme effekt. se bilag 13 og 14	Bestået
M08	Når en spiller lander på fængselsfeltet, så er den spiller bare på besøg.	Bestået
M09	Hvis en spiller er i fængsel, kan man vælge imellem at rulle med terningerne, for at få to ens, betale, eller bruge et jail card. Hvis man ruller med terningerne, skal man have to af samme slags for at komme ud.	Bestået
M10	Hvis en spiller lander på et felt ejet af en anden spiller, betaler den første spiller leje til ejeren.	Bestået
M11	Hvis man lander på gå i fængsel, eller man trækker chancekortet "Gå i fængsel", så indtjener man ikke 4.000 kr., hvis man passere start.	Bestået
M12	Som ses på bilag 15 stopper spillet med at kører, fordi spiller 1 ikke har flere penge. Spillet udskriver nogle Strings, som siger at spillet er slut og spillets vinder. Vinderen bliver basseret på hvilken spiller, der har den højeste pengemængde, når den første spiller går fallit.	Bestået
M13	Ejer man alle grunde med samme farve, får man dobbelt leje på ubebyggede grunde. Se bilag 16	Bestået
M14	Som det ses på bilag 17, bygges der hus på en grund, øges lejen af denne grund	Bestået
M15	Ejer man alle grunde med samme farve, har man ret til at bygge huse, til den pris der står på skøderne, ses på bilag 17	Bestået
S01	En spiller der lander på feltet indkomstskat, har lov at betale 4.000kr, eller 10% af sine værdier, se bilag 18	Bestået
S02	En spiller kan komme ud af fængslet ved at betale 1.000 kr. inden spilleren kaster med terningerne, se bilag 19	Bestået
S03	Hvis en spiller kaster 2 af samme slags, skal spilleren standse på et felt og derefter kaste igen. Se bilag 20	Bestået
S04	En spiller kan komme ud af fængsel, ved at benytte et løsladelseskort. Der vises ikke nogen besked om at kortet er brugt, man løslades på næste tur.	Bestået



S05	En spiller kan ikke blive i fængslet mere end 3 omgange, for man ikke to af samme slags, når man kaster tredje gang, skal man betale 1.000 kr. og flytte som øjnene viser.	Bestået
S06	Indbyrdes handel med ubebyggede grunde er tilladt, til den pris spillerne kan blive enige om, se bilag 21	Bestået
S07	Det er muligt at bygge et hotel uden at have 4 huse på en grund, hvilket er en fejl. Se bilag 22	Ej Bestået
S08	Prisen for et hotel skal være fem gange det af et hus, det ses at spiller 1 betaler 10000 for et hotel. Se bilag 23	Bestået
S10	Spillere skal kunne handle mellem hinanden med ejendomme og penge. Se bilag 21	Bestået
C01	Man skal bygge jævnt; altså må man bygge de første huse, på de grunde man ønsker. Man må først bygge hus nr.2, når man har bygget et hus på hver af de andre grunde i gruppen og så videre. Dette stemmer ikke, se bilag 22	Ej Bestået
C02	Hvis en spiller kaster 2 af samme slags 3 gange i træk, skal spilleren fængsles	Bestået
C05	Har banken ingen bygninger, må man vente til der kommer nogle tilbage.	Bestået
C11	Ved at printe det array som indeholder rækkefølgen af kortene ud i konsollen, kan det ses om den tal række, passer til de kort spillerne trækker. Dette er muligt da hvert chancekort også har sit eget tal.	Bestået

## 7.2 Negative test

I vores matador program er alt interaktion mellem bruger og source code låst bag en GUI. Derfor er det forholdsvis begrænset at kunne give forkerte inputs, fordi mange af steder hvor input er nødvendigt, laver GUI'en drop down menuer hvor man kun kan vælge forskellige givende input. Disse negative test fokuserer på det input, som bliver givet til programmet gennem GUI, og hvor programmet gør noget uønsket. Der er nogle forskellige steder hvor man giver input til programmet, og disse steder kan man inddele i nogle klasser. De forskellige slags inputs er: tryk på OK knap (med og uden drop down menu), indskrivning af Strings og de forskellige valg i menuen.

### Ok knapper

Ingen steder hvor der er ok knapper kommer der nogle fejl, lige meget om de har drop down menuer eller ej. programmet forsætter altid videre som det skal.

### Indskrivning af Strings

Den første af disse slags inputs er når spillerne skal vælge antallet af spillere til programmet. Her kan man skrive alt uden fejl, og spillet vender tilbage til det sammen spørgsmål indtil et får et lovligt input er givet. Her prøvede vi store tal, negative tal, kommatal, tegn og bogstaver men ingen havde nogen effekt.

### Valg i Menu

Det første valg i menuen er handel. Denne feature indeholder en del input fra spillerne, fordi en handel mellem spillere er kompleks, og indeholder mange forskellige inputs. Disse forskellige slags inputs fungerer typisk med drop down menu'er, som ikke giver nogle fejl, lige meget hvad man prøver at gøre ved den.

Dog er der en anden form for input i handlen mellem spillere. For at kunne handle med penge, skal en spiller skrive den mængde penge han vil give. Her kan en spiller kun indskrive ints, og disse kan kun være positive, derfor er det umuligt at skrive forkerte typer af værdier. Den eneste måde man kan prøve at få programmet til at gå ned, er ved at skrive en pengemængde, som er større en spillerens egen pengebeholdning, men dette er heller ikke muligt.

Det andet valg er valget om at købe bygninger, her er den eneste måde at interagere med programmet dropdown menuer, og ingen af de inputs man kan indskrive, giver nogen form for fejl. Hvis man prøver at købe bygninger på veje man ikke ejer, eller i farver hvor man ikke ejer dem alle sammen, så sender programmet en tilbage til den øverste menu, hvor man vælger imellem afslut spil, handel og bygninger. Det er umuligt at få programmet til at crashe eller bugge i de forskellige menu valg.

### 7.3 Unit-test

Unit-tests er automatiske tests. Vi har brugt JUnit4 i IntelliJ, til at lave vores unit-tests. En typisk test bruger en assertEquals metode, for at se om metodens output er det samme som forventet output. Hvis det virkelige resultat og det forventede stemmer overens er testen bestået. Unit-tests kan findes i mappen Test under src i programmet.

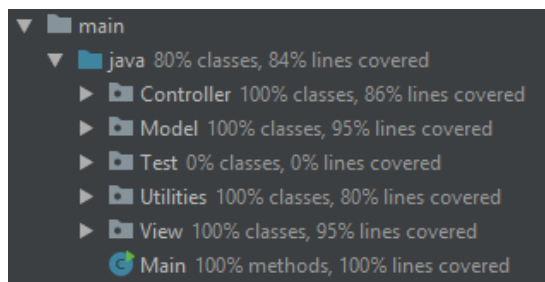
Test Case ID	Beskrivelse	Status
TC01	Test af startAccount(), for at sikre at en spiller for 30000	Bestået
TC02	Test af updateBalance(), for at sikre at penge bliver trukket eller indsat efter forventningen.	Bestået
TC03	Test af roll(), for at sikre at der ikke er tal under 1 eller over 6 i terningslagene.	Bestået
TC04	Test af instantiateSquares(), for at sikre at der bliver oprettet det rette antal af hver type felt.	Bestået
TC05	Test af passedStart, for at sikre at spilleren for penge over start.	Bestået
TC06	Test af calcNewPosition(), for at sikre spillerne for den rette position, når metoden benyttes.	Bestået
TC07	Test af wantOutOfJail(), for at sikre at spillerne kan betale 1000kr. og komme ud af fængslet.	Bestået
TC08	Test af playerWithHighestBalance(), for at sikre at spilleren med størst pengebeholdning er spilleren metoden vælger.	Bestået
TC09	Test af broke(), for at sikre at spilleren går fallit, hvis pengebeholdningen bliver negativ.	Bestået

### 7.4 Coverage test

Der er blevet lavet en coverage test, som undersøger hvor mange af metoderne og linerne i koden, som faktisk bliver brugt, når programmet har kørt. Denne test kom tilbage med høje procenter. Altså bliver de fleste metoder brugt. (Se tabel)

De metoder, som ikke bliver brugt, er kun "set" og "get" metoder, som kunne bruges, hvis programmet skulle udvides.

De liner kode, som ikke blev brugt er ofte særtilfælde, som tilfældigvis ikke er blevet ramt i denne gennemgang af programmet. Det er fx "Matadorlegatet"; et chancekort, som giver spilleren 40.000 kr, hvis spilleren ejer ganske lidt. Da spilleren, som trak dette chancekort ikke havde under 15.000 kr i samlet værdi, har programmet aldrig kørt de linjer koder, som giver spilleren de 40.000 kr.



Figur 7: Coverage Test

Element	Class	Method	Line
Controller	100%	98%	86%
Model	100%	95%	95%
Utilities	100%	80%	80%
View	100%	100 %	95%
Main	100%	100%	100%

Tabel 5: Coverage Test

## 8 Projektplanlægning

### 8.1 Planlagt forløb

Vores projekt er planlagt hovedsageligt efter de to milesten, som er lagt ind i projektets forløb. Derfor vil vi starte med at finde og give prioritet til alle de funktionelle krav i Matador. Herefter kigges der på use cases, og vi lægger planer for, hvilken rækkefølge de forskellige funktioner skal udvikles, og hvornår vi regner med at have disse funktioner færdige. Vi vil også lave forskellige diagrammer f.eks. design klasse diagram, til at hjælpe os med at skulle programmere programmet.

Efter det første milestensmøde begynder vi for alvor at programmere. Her udvikler vi de mest essentielle krav først, for at få lavet et program som kører, og kan blive spillet. I løbet af denne periode genevalueres kravene, for at se om vi kan få færre eller flere funktioner med, i den tid vi har tilbage til at kode i. Testene vil løbende blive lavet sammen med koden, og vi vil skrive videre på vores rapport, for at holde hele projektet opdateret.

Vi mødes hver dag i gruppen og arbejder på projektet sammen, vi starter hver dag med at snakke om hvad vi fik lavet dagen før, hvad vi mangler og hvem laver hvad. På den måde undgår vi, at der bliver lavet dobbeltarbejde og det er nemmere at holde styr på tidsplanen. Før milesten 2, skal vi have implementeret alle vores must have krav, og desuden have et program der kører fejlfrit.

I tiden efter det sidste milestensmøde, skal hele projektet blive færdigt, det vil sige, at der skal arbejdes mere på selve rapporten. Der skal desuden køres flere effektive test, der skal rettes i koden og der skal evt. tilføjes flere funktioner afhængig af, hvor tidspresset vi bliver. Desuden skal koden gennemgås og ryddes op i, der skal skrives kommentarer og variable navne skal være konsistente.

### 8.2 Reelt forløb

Projektet startede med at følge planen, der blev opstillet krav, use cases, diagrammer og modeller desuden blev alle mål til det første milestensmøde. Vi fik prioriteret kravene i fire kategorier, så vi vidste, hvilke krav vi skulle arbejde på først.

Efterfølgende var der dog en del udfordringer i forbindelse med fremgangsmåden. Dette skyldtes, at fremfor at arbejde ud fra koden fra Monopoly Junior direkte, valgte vi at "starte fra bunden" i den forstand, at vi tog et element af gangen, og gennemgik dets relevans for at skabe bedre forståelse af koden for gruppens medlemmer, og et bedre program uden irrelevant kode. Dette betød, at de første dage med arbejde med programmeringen gik fremskridtene langsommere end forventet. På trods af, at alle arbejdede på forskellige branches, opstod der alligevel nogle store merge konflikter, som skabte nogle problemer for gruppen. Hvilket betød, at få af vores must have krav fik vi ikke klar til det andet milestensmøde, her i

blandt "Bygninger/hoteller" og "Byt". Desuden har vi måtte erkende, at vi ikke ville nå sub-use casesne "Aktioner", "Pantsætning" og "Salg til bank".

Hen mod enden af processen forløb projektet meget tæt på den planlagte måde, vi fik lavet koden færdig og fik testet om de forskellige krav var opnået. Vi havde lavet positive test under forløbet af kodningen, men vi tjekkede alle til sidst for at finde eventuelle nye bugs, som kunne være opstået af omstrukturering af noget kode. Unit-test blev også opsat mod enden af processen. Vi løb ikke ind i noget større tidspres, da alle delen af programmet, hver især, blev afsluttet med tid tilbage. Dette gjorde vi f.eks. ved at sørge for ikke at starte nogle nye store kodningsprojekter, som nye funktioner de sidste par dage, vi afsluttede kun det vi var i gang med. Til sidst stod vi med et produkt vi var tilfredse med, både i forhold til rapport og source kode.

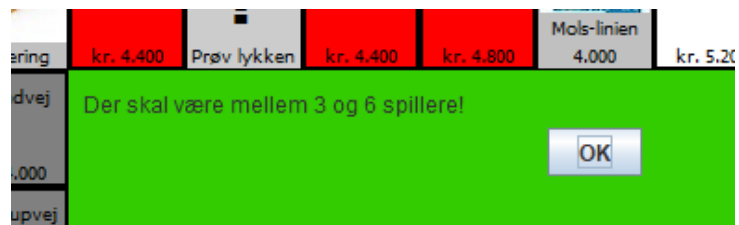
## 9 Konklusion

Selve processen i projektet fulgte den planlagte plan bedre end de forgående projekter. Vi havde det, vi ville have klar og lidt ekstra til det første milestemøde, til det andet var der nogle funktioner, vi gerne ville have haft lavet klar til der, men som blev færdig dagene efter mødet. Starten af kodning processen var langsommere end først forventet, da vi gerne ville forstå koden for det forgående Monopoly Junior, og kun implementere det, der var brugbart. Denne langsommere start har resulteret i, at koden er blevet markant bedre, men dog også givet os den langsommere start. Som nævnt i det planlagte forløb så startede dagene med, at vi fandt ud af, hvor vi stod henne i processen, hvilket resulterede i, at overblikket over hele projektet var ret godt. Tidligt inde i processen så vi, at der var nogle krav, der ikke ville kunne blive til noget. Forholdsvis tidligt i processen fik vi sat programmet op, så det kunne køre, og vi fik samlet alle de funktioner, vi havde klar. Denne bedre måde gjorde, at vi altid havde et produkt som virkede, og dermed gav en kæmpe hjælp til at finde ud af, hvor langt vores program var kommet.

I sidste ende kom vores produkt mere eller mindre ud, som vi havde forventet fra starten af. Alle vores must have krav er blevet implementeret, plus nogle should have f.eks. handel mellem spillere. GUI'en viser forløbet af spillet, den giver også mere eller mindre brugervenlighed til at kunne lave valg inde i spillet f.eks. handel mellem spillere. Nogle af de store funktioner som vores program ikke kom til at indeholde, men som normalt er en stor del af Matador, var funktioner, såsom pantsætning og auktioner. Disse ideer kunne man eventuelt arbejde videre med, men tidspresset er for stort, til, at vi kan implementere disse krav i tide. Selvom vi ikke har fået lavet alle vores should have krav, så har vi lavet to could have krav. Et af disse krav er et dæk med chancekort, dette implementerede vi fordi næsten hele koden til denne funktion allerede stod i Monopoly Junior, så derfor var det en lav arbejdsmængde for en bedre spilleroplevelse.

Processen i projektet har været den største forbedring over andre projekter, heriblandt CDIO-projekterne. Forløbet og arbejdsprocessen har været mere struktureret og realistisk end tidligere. I modsætning til de forgående CDIO opgaver, så var dette projekt blevet startet i god nok tid, til at få lavet et godt produkt i tide. Vi har lagt mere vægt på GRASP metoderne, og også blevet bedre til at overholde dem og bruge dem bedre f.eks. med low coupling og high cohesion. I modsætning til andre projekter har vi brugt en større mængde controllers, og disse controllers er meget mere specificeret til præcis deres ansvarsområde. Koden er pænere sat op og er mere effektiv og direkte, metoderne samt klasserne har et defineret ansvarsområde og holder sig mere eller mindre kun til det. Hvis der perspektiveres til vores ældre CDIO projekter, så er store dele af analysen også lavet bedre. Vi har fået bedre forståelse for use cases, sup use cases og krav. Ved kravene har vi brugt en anden og mere overskuelig måde at inddele de forskellige krav, hvor man meget nemmere kan se hvad der er vigtigst, hvad der skal være med og de krav som er mindre vigtige for spillet.

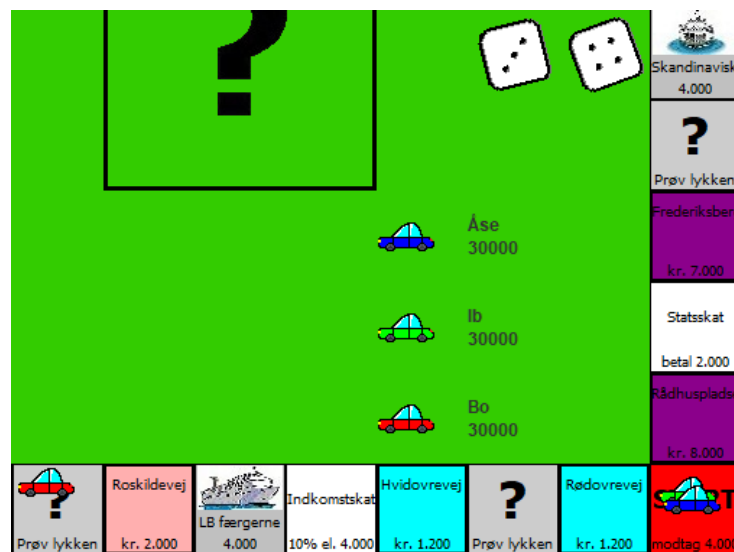
## 10 Bilag



Figur 8: Antal spillere



Figur 9: Start balance



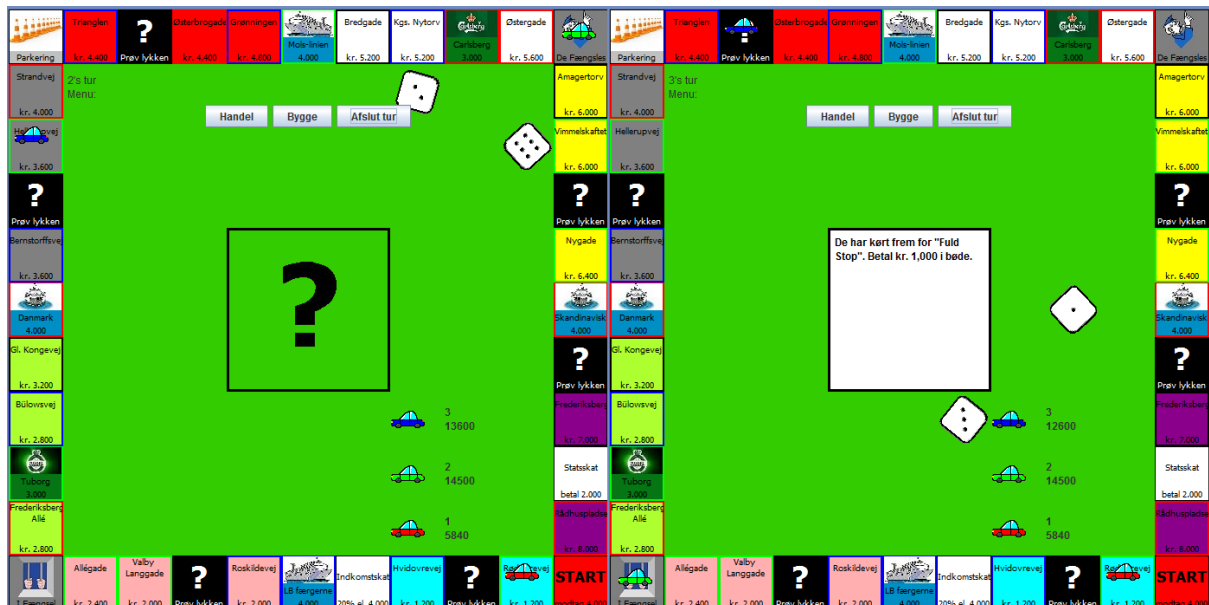
Figur 10: Bevægelse af en spiller, svarer til terning slaget



Figur 11: Køb af grunde



Figur 12: Chancekort

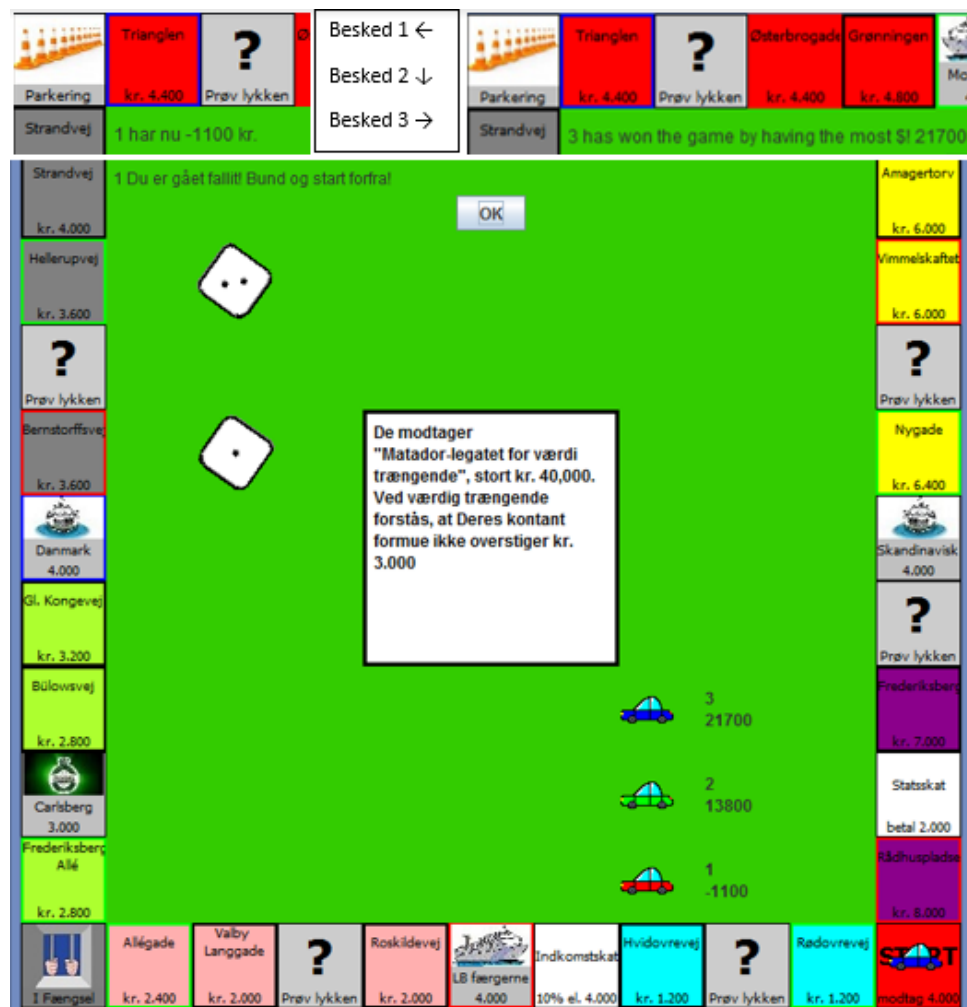


Figur 13: De fængsles - felt

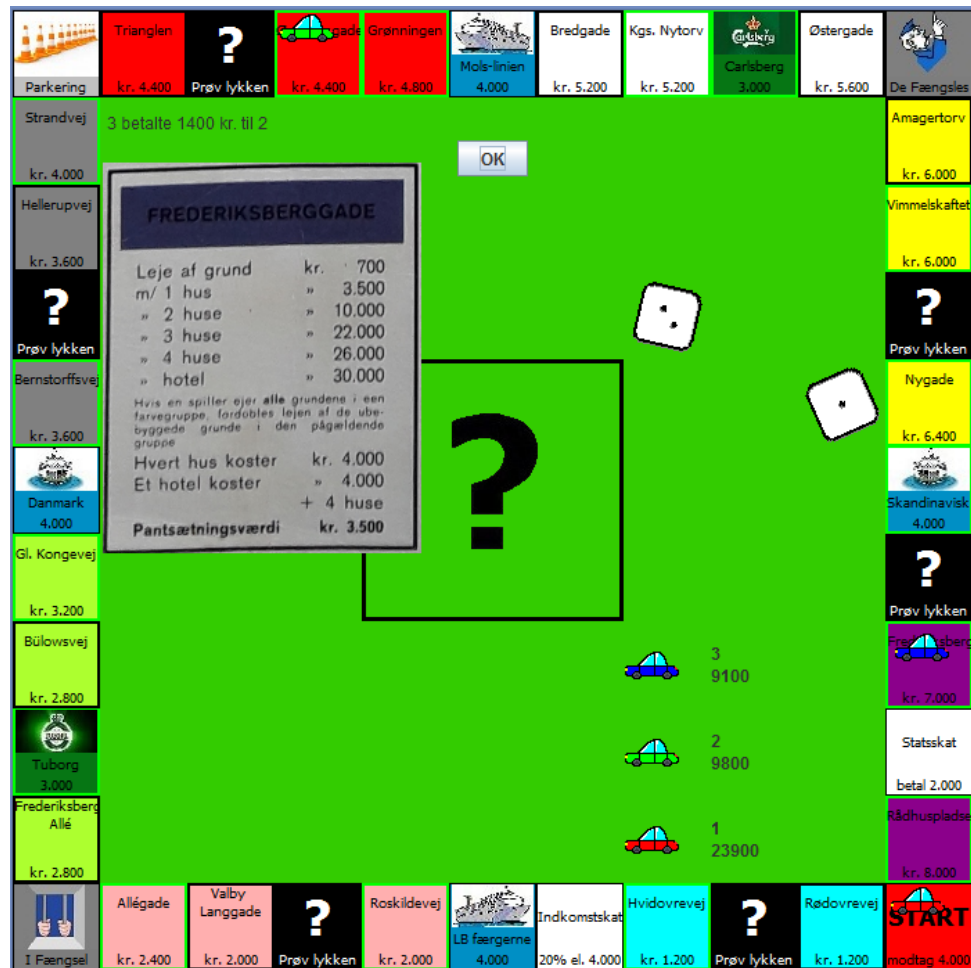


Figur 14: Gå i fængsel - Chancekort





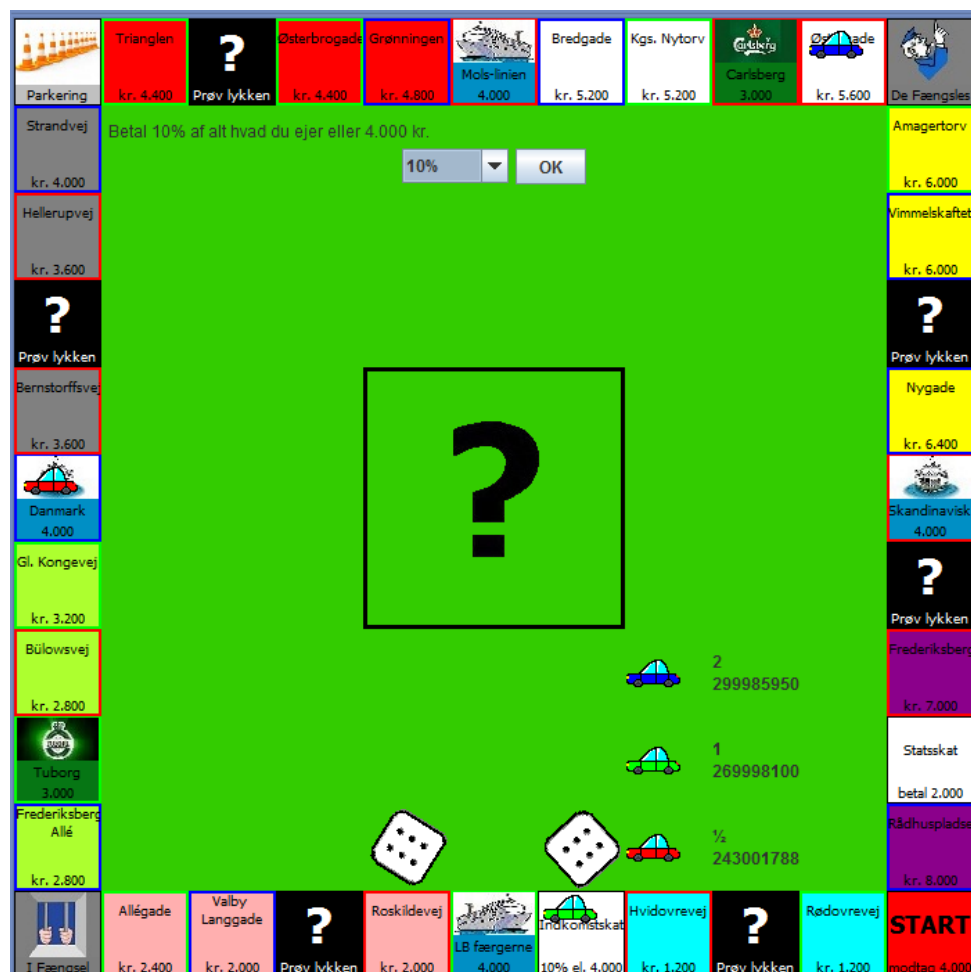
Figur 15: Spillet slutter når en spillers balance går i minus



Figur 16: Dobbelt leje når alle i en farve er ejet



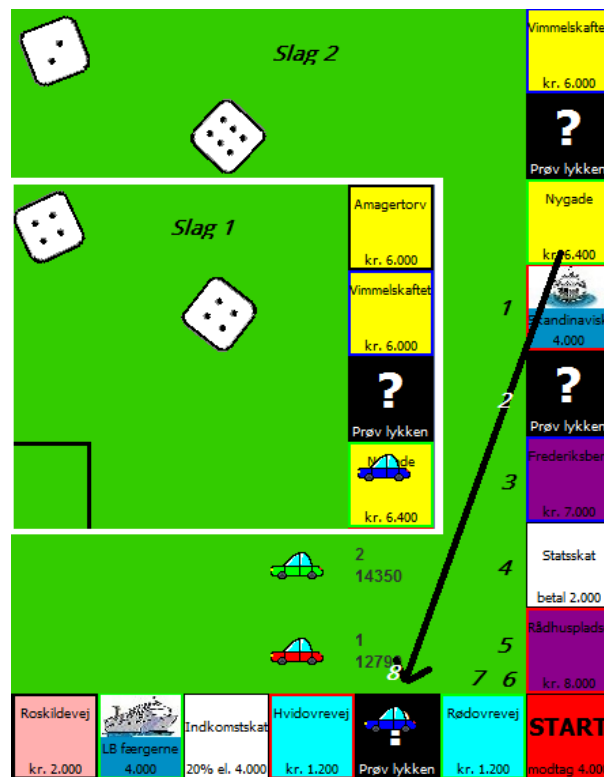
Figur 17: Køb af hus, øget leje



Figur 18: Indkomstskat



Figur 19: Køb sig ud af fængsel



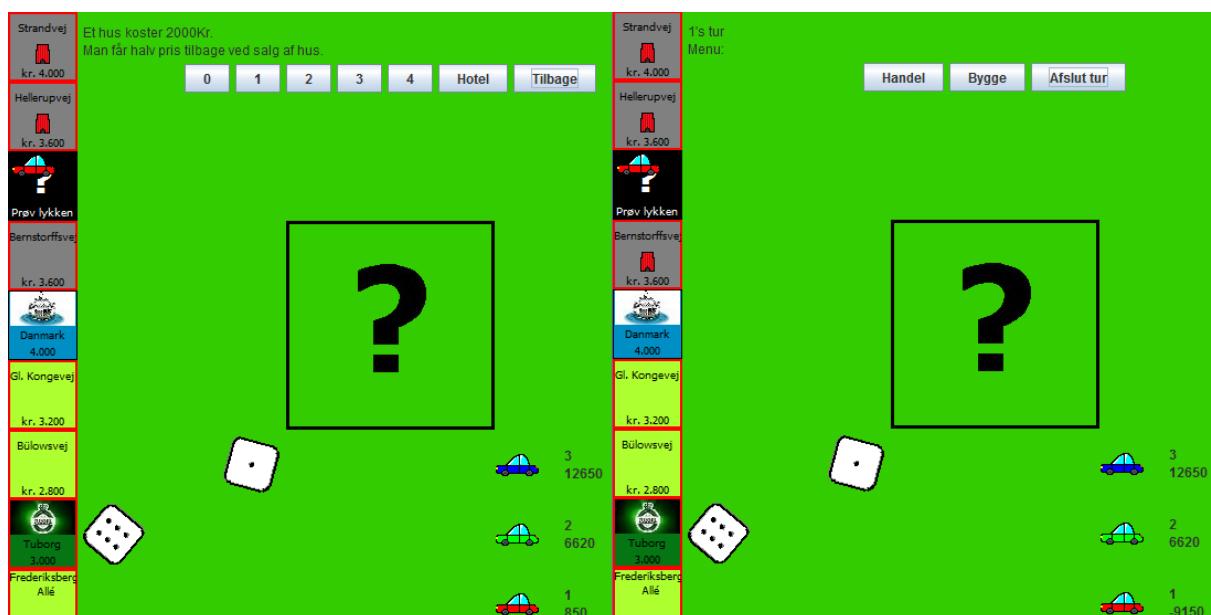
Figur 20: Ekstratur ved par slag



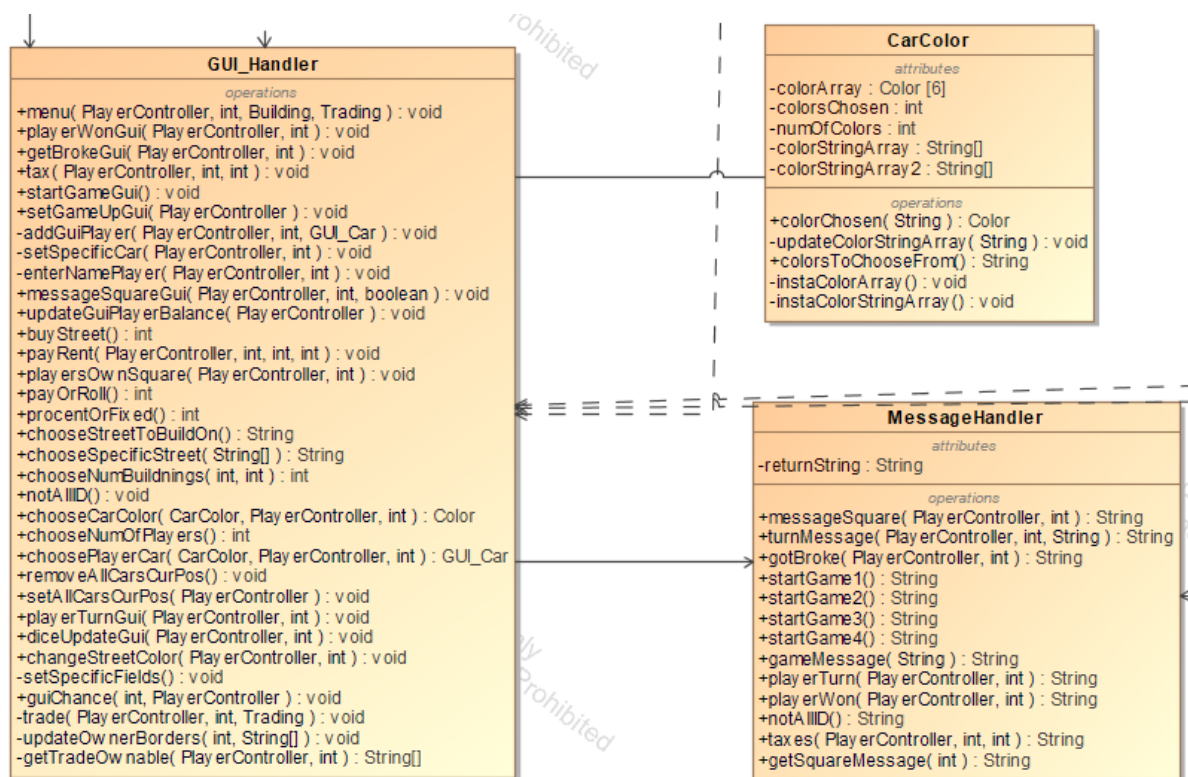
Figur 21: Handel



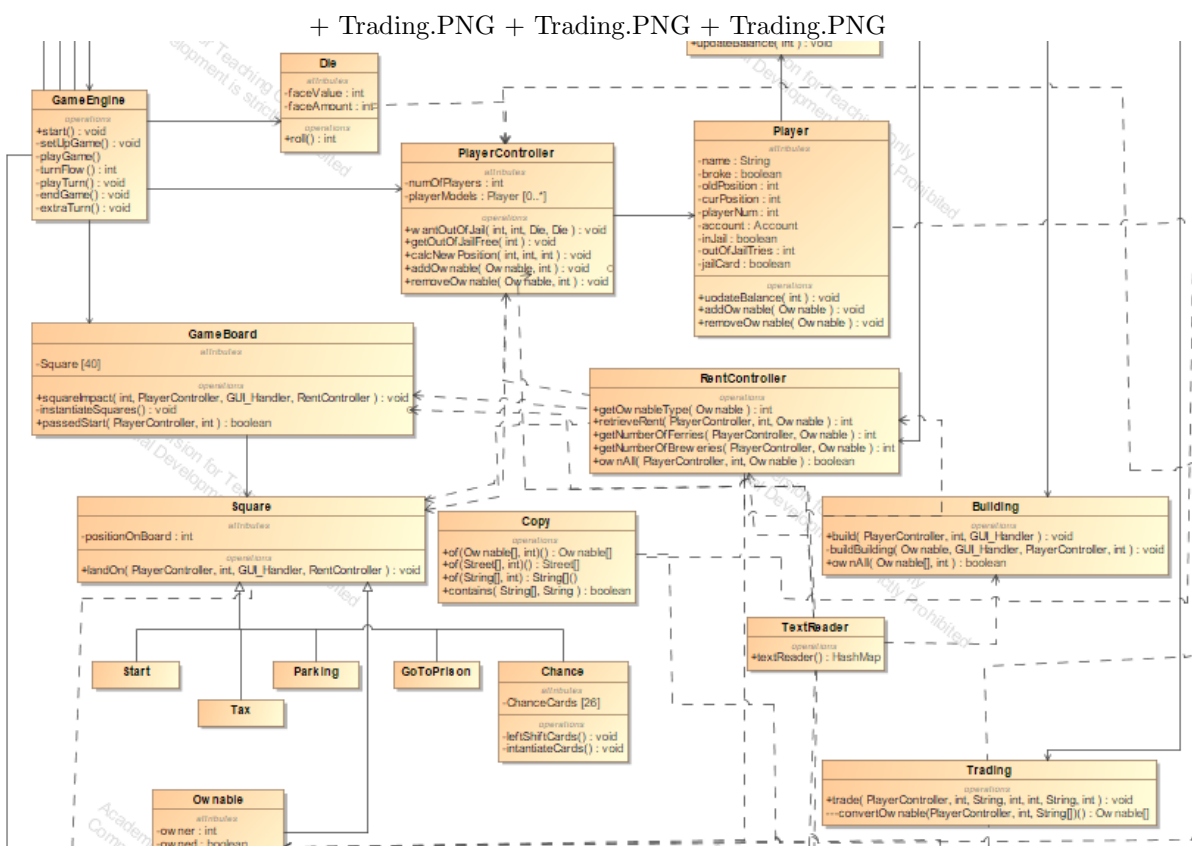
Figur 22: Hotel



Figur 23: Pris ved hotel

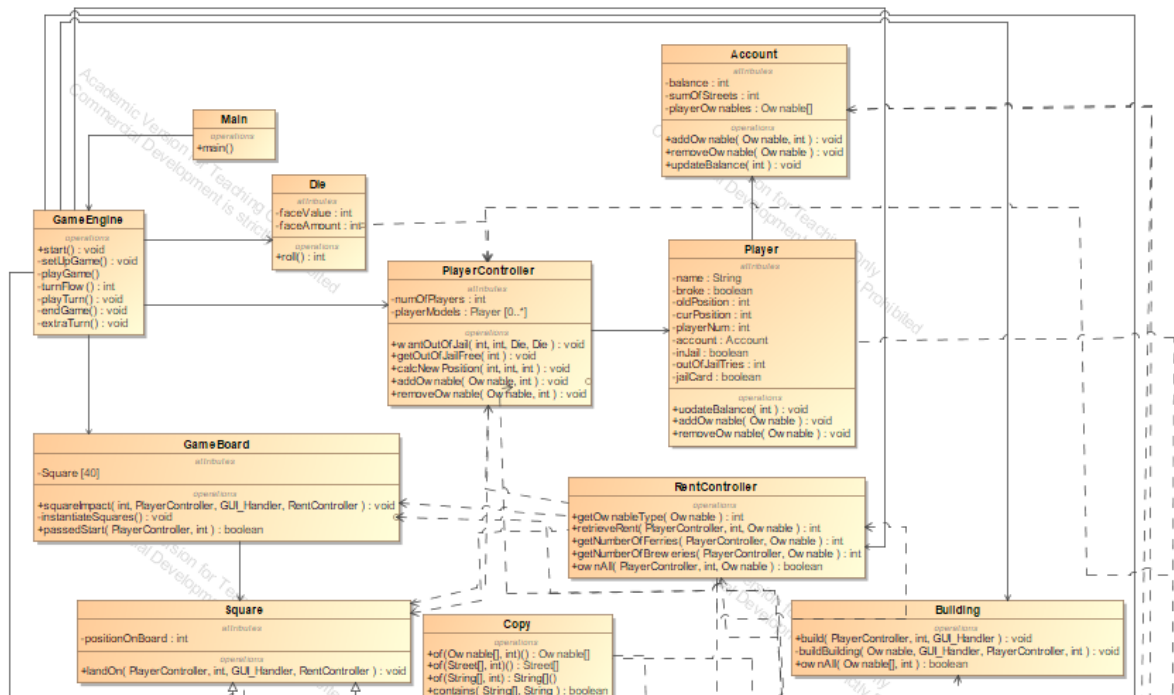


Figur 24: View



Figur 25: Trading





Figur 26: Main

100% classes, 86% lines covered in package 'Controller'			
Element ▲	Class, %	Method, %	Line, %
Building	100% (1/1)	100% (4/4)	68% (90/132)
GameBoard	100% (1/1)	83% (5/6)	96% (27/28)
GameEngine	100% (1/1)	100% (8/8)	100% (65/65)
PlayerController	100% (1/1)	100% (25/25)	94% (80/85)
RentController	100% (1/1)	100% (5/5)	94% (52/55)
Trading	100% (1/1)	100% (2/2)	100% (26/26)

Figur 27: Coverage test Controller

100% classes, 95% lines covered in package 'Model'			
Element ▲	Class, %	Method, %	Line, %
Squares	100% (10/10)	100% (28/28)	96% (172/179)
Account	100% (1/1)	75% (6/8)	89% (25/28)
Die	100% (1/1)	100% (3/3)	100% (7/7)
Player	100% (1/1)	95% (20/21)	97% (38/39)

Figur 28: Coverage test Model

100% classes, 96% lines covered in package 'Model.Squares'

Element ▲	Class, %	Method, %	Line, %
Brewery	100% (1/1)	100% (1/1)	100% (2/2)
Chance	100% (1/1)	100% (5/5)	93% (95/102)
Ferry	100% (1/1)	100% (1/1)	100% (2/2)
GoToPrison	100% (1/1)	100% (2/2)	100% (5/5)
Ownable	100% (1/1)	100% (10/10)	100% (38/38)
Parking	100% (1/1)	100% (1/1)	100% (2/2)
Square	100% (1/1)	100% (3/3)	100% (5/5)
Start	100% (1/1)	100% (2/2)	100% (4/4)
Street	100% (1/1)	100% (1/1)	100% (2/2)
Tax	100% (1/1)	100% (2/2)	100% (17/17)

Figur 29: Coverage test Square

100% classes, 80% lines covered in package 'Utilities'

Element ▲	Class, %	Method, %	Line, %
Copy	100% (1/1)	75% (3/4)	70% (19/27)
TextReader	100% (1/1)	100% (1/1)	100% (14/14)

Figur 30: Coverage test Utilities

100% classes, 95% lines covered in package 'View'

Element ▲	Class, %	Method, %	Line, %
CarColor	100% (1/1)	100% (6/6)	88% (46/52)
GUI_Handler	100% (1/1)	100% (34/34)	96% (403/417)
MessageHandler	100% (1/1)	100% (14/14)	100% (22/22)

Figur 31: Coverage test View