

CDIO - 3 ugers projekt



Lavet af:



Jonas Praëm (s144883)



Mads Justesen (s134834)



Alexander V. Pedersen (s145099)



Andreas Strange (s082853)

<u>Indledning</u>	<u>side 5</u>
<u>Kravspecifikation</u>	<u>side 5</u>
Generelle krav:	<u>side 5</u>
Krav til fængsel:	<u>side 5</u>
Krav til huse og grunden:	<u>side 6</u>
Krav til feltyper:	<u>side 6</u>
Krav til pantsætning:	<u>side 6</u>
Krav til chancekort:	<u>side 6</u>
<u>Use-Case Beskrivelser</u>	<u>side 7</u>
<u>Use-Case Diagram Kast Terninger:</u>	<u>side 7</u>
<u>Use-Case Diagram Køb Felt:</u>	<u>side 8</u>
<u>Use-Case Diagram Fængsel:</u>	<u>side 9</u>
<u>Use-Case Diagram Chancekort:</u>	<u>side 11</u>
Køb hus/hotel:	<u>side 12</u>
<u>Use-Case Diagram Penge mangel:</u>	<u>side 13</u>
<u>Domænemodel:</u>	<u>side 14</u>
Field hierarki:	<u>side 15</u>
Card hierarki:	<u>side 16</u>
<u>System sekvens diagram</u>	<u>side 17</u>
<u>Beskrivelse</u>	<u>side 17</u>
<u>BCE model</u>	<u>side 19</u>
<u>Kode</u>	<u>side 21</u>
<u>Controllere</u>	<u>side 21</u>
<u>Entiteter</u>	<u>side 29</u>
<u>Test</u>	<u>side 32</u>
<u>Deck test</u>	<u>side 32</u>
<u>Test af kort1</u>	<u>side 32</u>
<u>Test af kort 10</u>	<u>side 34</u>
<u>Test af Jail</u>	<u>side 36</u>
<u>Showcase</u>	<u>side 37</u>
<u>Code Coverage</u>	<u>side 38</u>
<u>Designklassediagram</u>	<u>side 40</u>
<u>Del 1 - Associationer mellem Controllers.</u>	<u>side 40</u>
<u>Del 2 - Controller til Entitet</u>	<u>side 41</u>
<u>Del 3 - Entiteter til nedarvinger.</u>	<u>side 42</u>
<u>Design Sekvensdiagram</u>	<u>side 43</u>
<u>Del 1</u>	<u>side 43</u>
<u>Del 2</u>	<u>side 44</u>
<u>Del 3</u>	<u>side 46</u>
<u>Diskussion</u>	<u>side 47</u>
<u>Konklusion</u>	<u>side 48</u>
<u>Bilag</u>	<u>side 49</u>

Feltliste	side 49
Chancekort liste	side 54
Kildekode	side 56
Launch	side 56
Controllers	side 56
Entities	side 56
entity package	
fields package	
deck package	
Test	side 57

Indledning

Denne rapport er dokumentation over det afsluttende projekt i kursus 02312 *Indledende programmering*, ved DTU, Januar 2015.

Projektet er at programmere et fuldt funktionelt matador spil i Eclipse. Spillet skal for så vidt muligt indeholde alle funktioner af det originale matadorspil.

I projektet er det prioriteret højere at lave et program der er spilbart, end at implementere alle funktioner.

Kravspecifikation

Generelle krav:

Funktionelle krav	Ikke funktionelle krav
<ul style="list-style-type: none">Der skal være 3-6 spillere	<ul style="list-style-type: none">Spillere slår terninger på skift
<ul style="list-style-type: none">Brikkerne går i ring på brættet	<ul style="list-style-type: none">Slår med raflebærge med 2 terninger
<ul style="list-style-type: none">Der er 40 felter på brættet	<ul style="list-style-type: none">Spillet skal kunne køres på alle DTU's windows computere
<ul style="list-style-type: none">Starter med 30.000 penge	
<ul style="list-style-type: none">Hvis du passerer start får du 4.000	
<ul style="list-style-type: none">Vinder fundet når alle undtagen 1 er gået bankerot.	
<ul style="list-style-type: none">Ekstra slag hvis man slår 2 terninger af samme slags.	

Krav til fængsel:

- Der skal være et fængsel, hvis man passerer start får du ikke penge. Hvis du ikke når at komme ud inden 3 ture, betaler du bøden.
- Man kan komme ud af fængsel ved: 2 ens - på 3 forsøg, chancekort eller ved at betale 1000 inden man kaster.
- Ved exit fra fængslet rykker man, som øjnene viser. (ingen slag efter exit)
- Slår man et par 3 gange i træk, ryger man i fængsel.

Krav til huse og grunden:

- Hvis du ejer alle grunde i samme farve, og en person lander på den, skal man betale dobbelt leje og ejeren har ret at bygge huse.

- Alle grunde skal have et hus før du må bygge et ekstra på.
- Et hotel er det samme som at have 5 huse.
- Går du bankarot bliver dine huse og grunde solgt til banken.

Krav til feltyper:

- Territory felter er til salg eller til leje.
- Refuge felter udbetaler en bonus.
- Tax felter giver muligheden mellem 10% afgift eller et fast beløb.
- Labor camp felter giver afgift på s. $100 * \text{terning slag} * \text{antal ejet labor camp}$.
- Fleet felter har en afgift på $500 * \text{antal ejet fleet felter}$.
- Territory, labor camp og fleet felter kan købes når de landes på medmindre de allerede er ejet.

Krav til pantsætning:

- Hvis din grund er pantsat kan du ikke modtage penge fra den.
- Du skal betale pantsætningens værdi tilbage + yderligere 10% for at upantsætte.
- Du får en pantsætning værdi ved pantsætning.

Krav til chancekort:

- Kortet bliver trukket øverst fra bunken og lagt nederst i bunken.
- Kortene skal være funktionelle.
- Hvis det er et fængselskort skal det kunne gemmes.

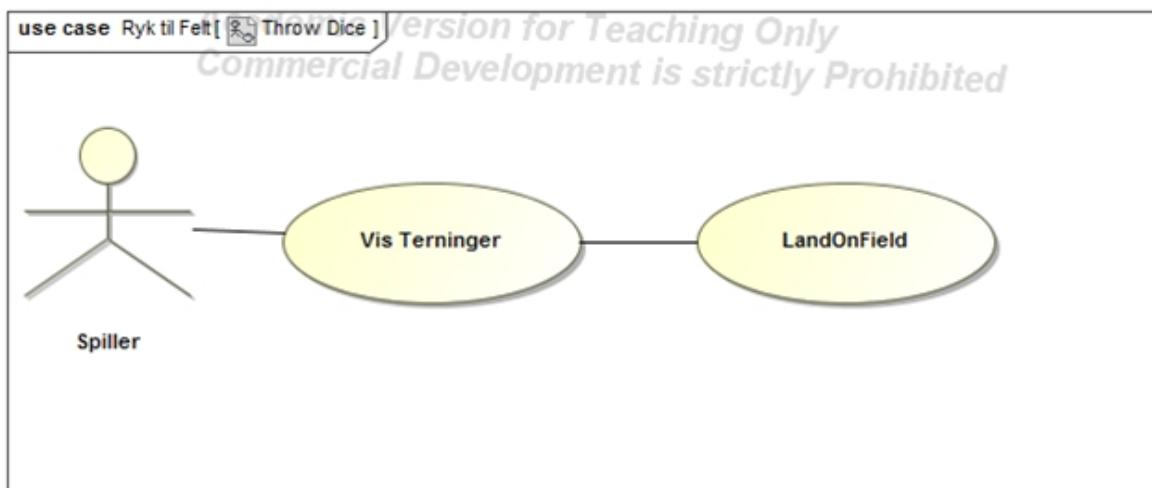
Use-Case Beskrivelser

Use Case:	Spiller kaster terninger
------------------	--------------------------

Use Case ID:	UC01
Primær aktør:	Spiller
Sekundær aktør:	Ingen
Preconditions:	Spillet er igangsat
Main flow:	<ol style="list-style-type: none"> 1. Spilleren vælger at kaste terningerne 2. Systemet kaster terningerne 3. Systemet viser antal øjne på terningerne i GUI 4. Systemet flytter spillerens bil til feltet 5. Systemet vælger spillerens risiko på feltet.
Postconditions:	Ingen
Alternative flows:	Ingen

Use-Case Diagram Kast Terninger:

(figur 1)

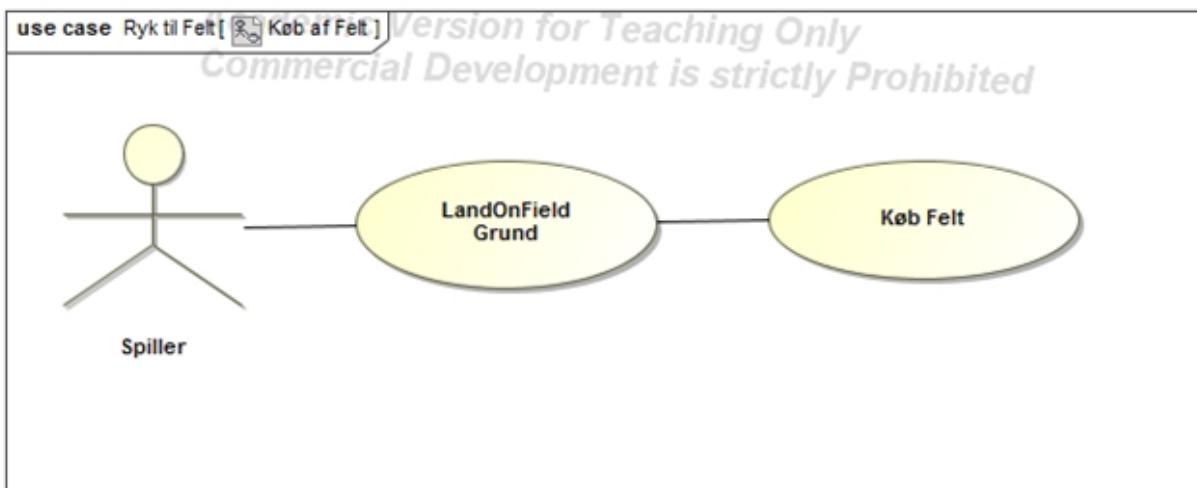


Use Case:	Spiller køber felt
Use Case ID:	UC02

Primær aktør:	Spiller
Sekundær aktør:	Ingen
Preconditions:	Spilleren har landet på et felt af typen Territory, som kan købes. Feltet er ikke allerede købt.
Main flow:	Spilleren har nu landet på et felt som kan købes. Systemet spørger via GUI om spilleren vil købe feltet. Spilleren trykker "Ja" Systemet tilføjer at feltet er ejet til felt klassen. Systemet trækker feltets værdi fra spillerens pengebeholdning.
Postconditions:	Feltet er nu ejet
Alternative flows:	Spilleren køber ikke feltet.

Use-Case Diagram Køb Felt:

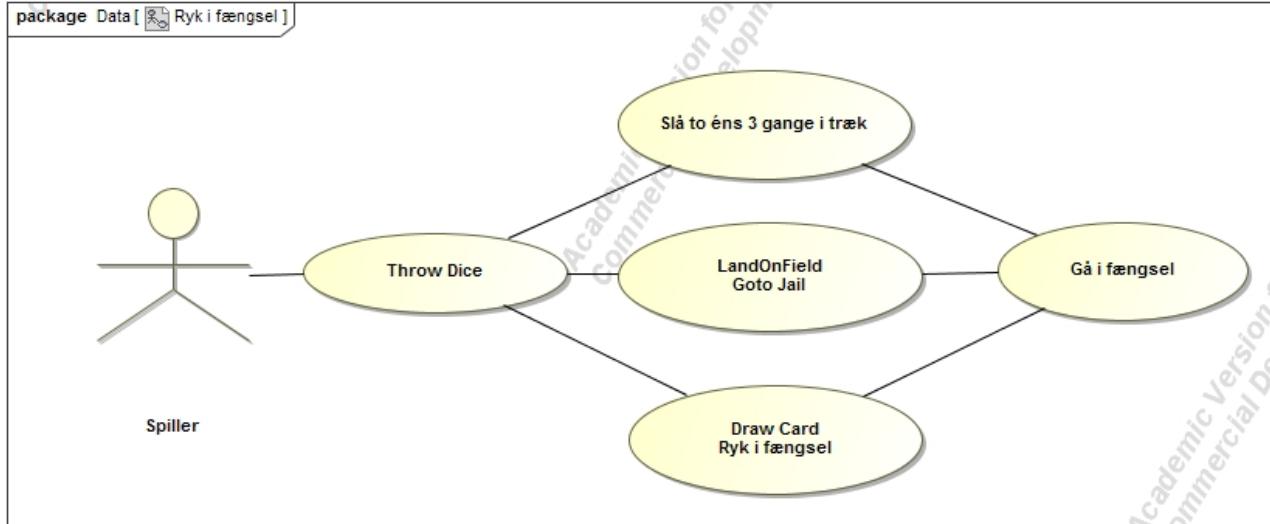
(figur 2)



Use Case:	Ryk i fængsel
------------------	---------------

Use case ID:	UC03
Primær aktør:	Spiller
Sekundær aktør:	Ingen
Preconditions:	Spilleren har kastet terningerne
Beskrivelse:	Spilleren kan gå i fængsel ved 3 scenarier. Det ene scenarie er ved at slå to éns 3 gange i træk. Det næste scenarie er ved at lande på feltet "Ryk i Fængsel". Det sidste scenarie er ved at trække chancekortet "Ryk direkte i fængsel".
Main flow:	<ol style="list-style-type: none"> 1. Spilleren har kastet terningerne 2. Systemet kaster terningerne 3. Systemet viser antal øjne på terningerne i GUI 4. Systemet flytter spillerens bil til feltet "Go to Jail". 5. Systemet flytter spilleren til feltet "Jail".
Postconditions:	Spilleren er sat i fængsel
Alternative flows:	<ol style="list-style-type: none"> 1. Spilleren slår to éns 3 gange i træk og ryger derfor i fængslet. 2. Spilleren trækker et chancekort som siger at spilleren skal rykkes i fængslet.

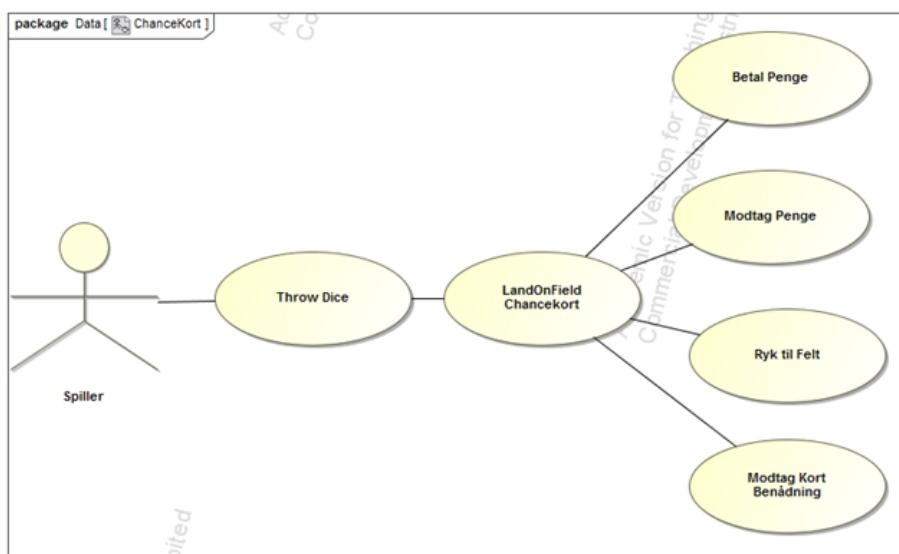
Use-Case Diagram Fængsel:
 (figur 3)



Use Case:	Chancekort
Use case ID:	UC04
Primær aktør:	Spiller
Sekundær aktør:	Ingen
Preconditions:	Spilleren lander på "Prøv Lykken"
Beskrivelse:	Spilleren trækker et kort og ét af fire overordnede scenarier udspiller sig. Spilleren skal enten modtage penge, betale penge, eller rykke i fængsel. Den sidste mulighed er at modtage et frikort, der kan bruges til at komme ud af fængsel.
Main flow:	<ol style="list-style-type: none"> 1. Spilleren har kastet terningerne 2. Systemet kaster terningerne 3. Systemet viser antal øjne på terningerne i GUI 4. Systemet flytter spillerens bil til feltet "Prøv Lykken". 5. Systemet viser et chancenkort. 6. Systemet udfører chancenkortets funktion.
Postconditions:	Scenarie 1-4
Alternative flows:	Ingen

Use-Case Diagram Chancenkort:

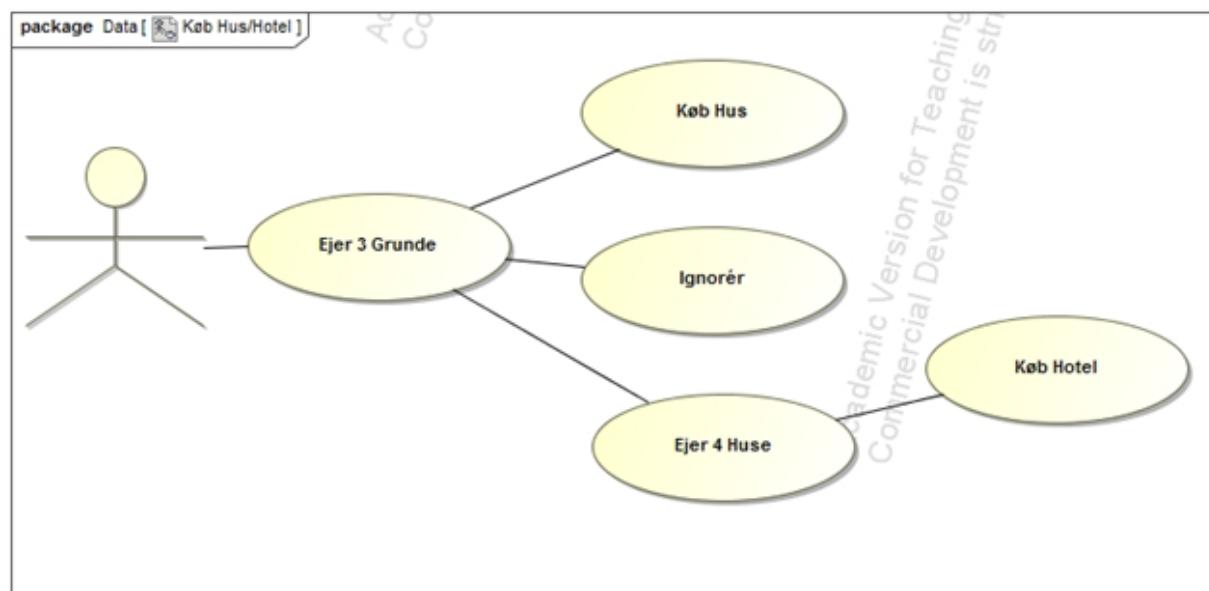
(figur 4)



Use Case:	Køb af hus/hotel
Use Case ID:	UC05
Primær aktør:	Spiller
Sekundær aktør:	Ingen
Preconditions:	Spilleren ejer 3 grunde af samme farve.
Beskrivelse:	Før spilleren starter sin tur kan denne købe op til 4 huse på hver grund. Først når spilleren har 4 huse på hver grund kan spilleren købe et hotel. Der kan være et hotel på hver grund.
Main flow:	<ol style="list-style-type: none"> Systemet spørger via GUI om spilleren vil købe huse. Spilleren trykker "ja" Systemet tilføjer antallet af købte huse på grunden.
Postconditions:	Der står antal købte huse på de ejede grunde.
Alternative flows:	<ol style="list-style-type: none"> Systemet spørger via GUI om spilleren vil købe huse. Spilleren trykker "nej" Spilleren fortsætter sin tur.

Køb hus/hotel:

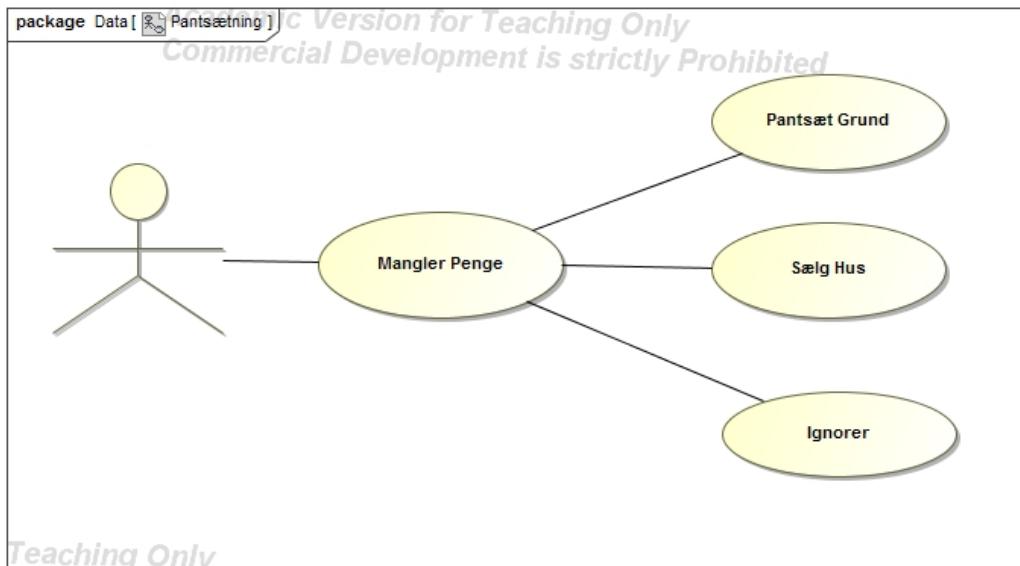
(figur 5)



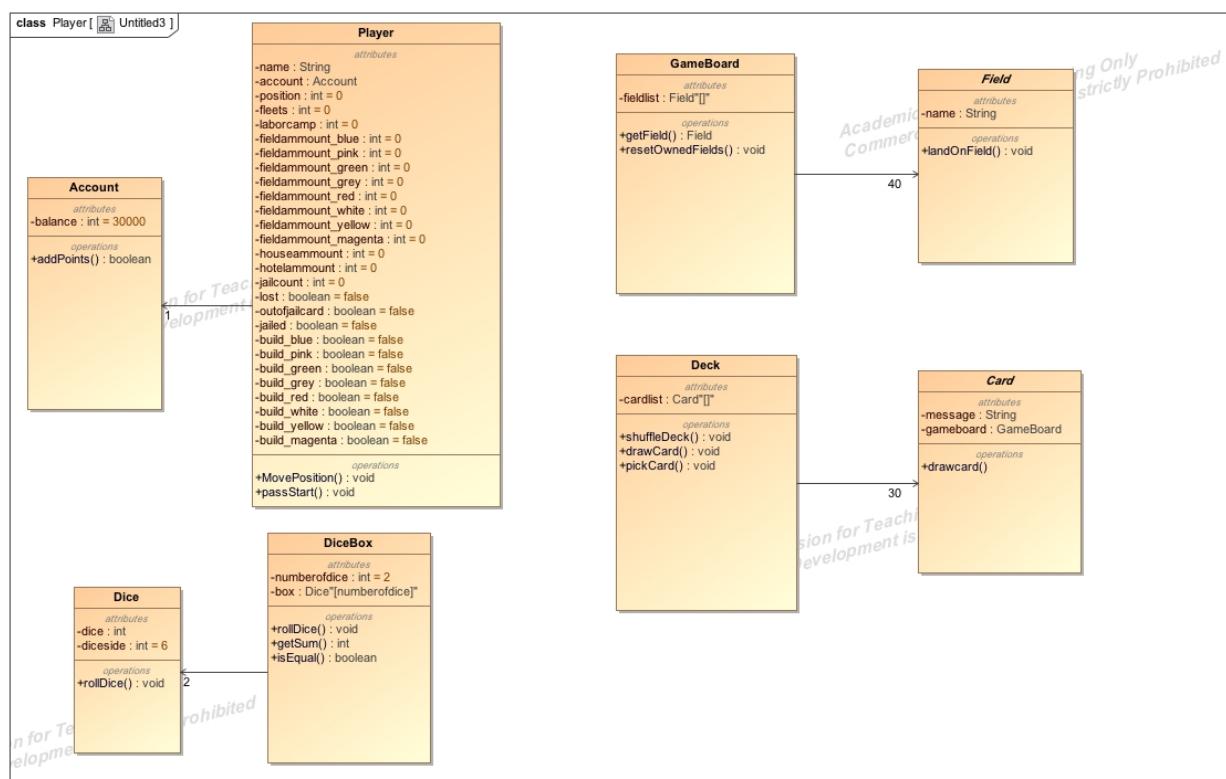
Use Case:	Pantsætning
Use Case ID:	UC06
Primær aktør:	Spiller
Sekundær aktør:	Ingen
Preconditions:	Spilleren mangler penge.
Beskrivelse:	Hvis en spiller mangler penge kan denne vælge at pantsætte sine ejede grunde. Hvis der er huse eller hoteller på de ejede grunde skal disse sælges først. En pantsat grund har den halve værdi af grunden.
Main flow:	<ol style="list-style-type: none"> 1. Systemet spørger via GUI om spilleren vil pantsætte grunde. 2. Spilleren trykker "ja" 3. Systemet tilføjer antallet af købte huse på grunden.
Postconditions:	Der står antal købte huse/hoteller på de ejede grunde.
Alternative flows:	<ol style="list-style-type: none"> 1. Systemet spørger via GUI om spilleren vil pantsætte grunde. 2. Spilleren trykker "nej" 3. Spilleren fortsætter sin tur.

Use-Case Diagram Penge mangel:

(figur 6)



Domænemodel:
(figur 7)



Ovenfor ses de overordnede klasser i vores program. Player klassen med en Account. DiceBox klassen med 2 (default) terninger. En GameBoard klasse med et array af Fields, og en Deck klasse med et array af Cards.

Både Card og Field er abstract klasser, der begge har et nedarvnings hierarki (se figur 8 og 9). Player klassen holder styr på alle variabler, der er unikke for spiller til spiller. Account klassen indeholder kun en balance, men er også den klasse sikrer at spilleren er i live. Altså den har kontrolmetoder (addPoints()), som kan påvirke lost boolean variablen i Player klassen.

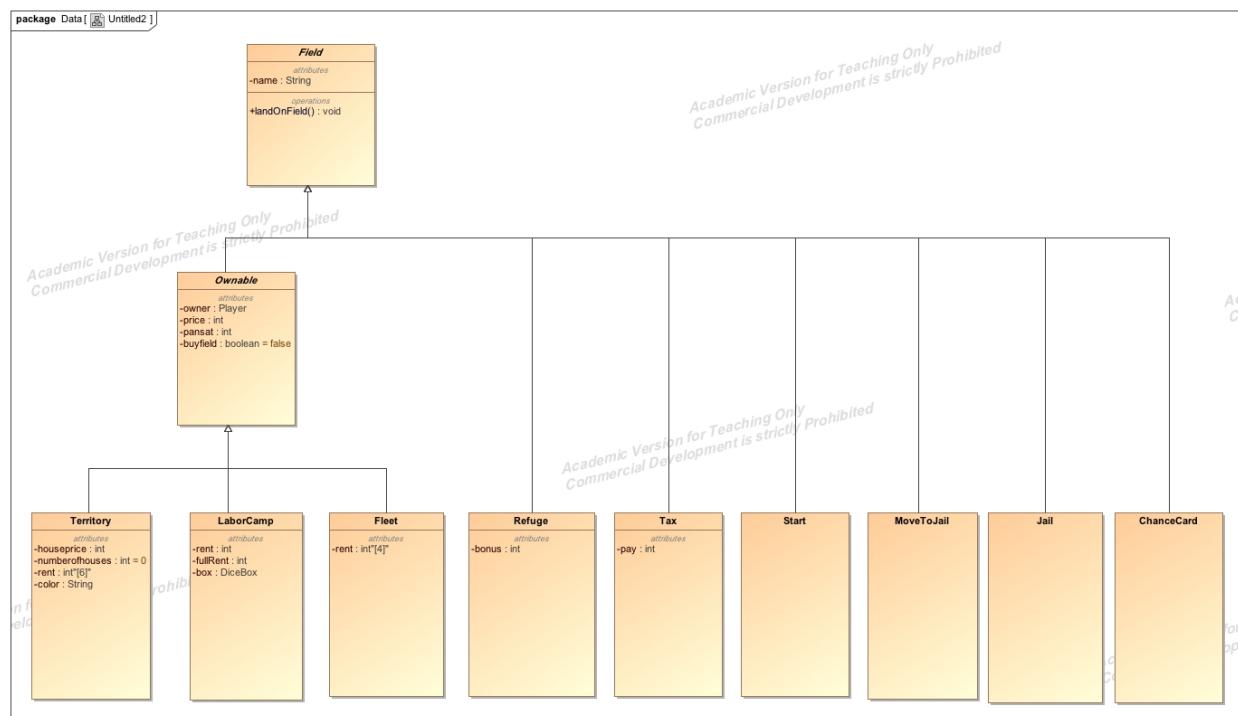
DiceBox er en klasse der indeholder et array af terninger, som kan ændres i længde. Dice har 2 variabler, som styre hvad sidehoved der vender opad på terningen - som er static, indtil terningen bliver kastet igen (rollDice()).

Deck klassen har servicemetoder, der kan blande kortbunken (Fisher-Yates algoritme brugt), samt en metode der trækker det øverste kort, og lægger det nederst i bunken (drawCard()). Derudover er der en pickCard() metode, der ikke ændre rækkefølgen på kortene, hvor det er kontrolerbart hvilke kort i bunken der skal trækkes. Denne metode er primært tænkt til testning af kortene.

GameBoard har kun en metode til at få fat i et bestemt Field, samt en servicemetode der fjerner owner fra alle Fields ejet af en specifik spiller.

Field hierarki:

(figur 8)



Field nedarer 2 grundtyper af Fields; Ownable Fields, og normale Fields. Ownable klassen er abstract, da felterne altid er enten Territory, LaborCamp eller Fleet. Alle Ownable klasser har attributerne til fælles: owner, price, pansat og en boolean variable (buyfield), der bruges til et brugervalg til at købe feltet.

Alle 3 Ownable klasser, holder styr på prisen for at købe feltet, huse og hvor dyrt det er at lande der, for en anden spiller end ejeren.

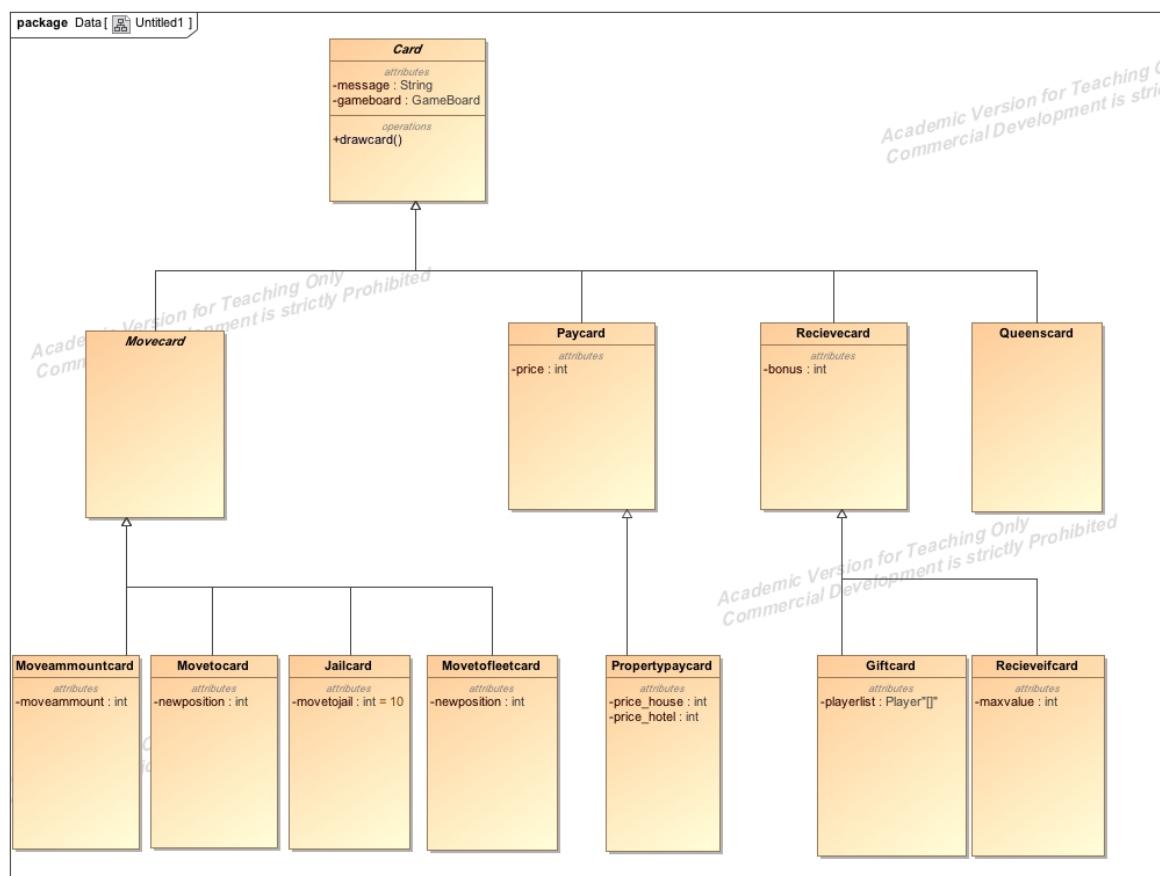
Der er i principippet 2 typer Tax felter, som er defineret i samme klasse (Tax). En der bare trækker penge fra spilleren, og en hvor spilleren kan vælge imellem en simpel pris eller 10% af totale kontante værdier.

Refuge giver en simpel pengeværdi til spilleren.

Resten af felterne har i principippet ingen funktion i klassen selv. I Stedet bliver deres funktion styret af FieldControlleren (se figur 10), da der her kommer kontakt til nogle klasser felterne ikke skal kende noget til.

Card hierarki:

(figur 9)



Et chancekort kan være én af de 10 klassetyper, dog har alle chancekort unikke beskeder/'message' (der er få chancekort, der er flere af i Deck klassen):

Moveammountcard: flytter spilleren med hensyn til spillerens nuværende pladsering. penge over start, hvis start passerer.

Movetocard: flytter spilleren til en bestemt position. penge over start, hvis start passerer.

Jailcard: flytter spilleren til fængsel, samt sætter Player variablen 'jailed' til 'true'.

Movetofleetcard: rykker spilleren til nærmeste rederi. penge over start, hvis start passerer.

Paycard: betal et beløb.

Propertypaycard: betal ud fra antal af bygninger.

Recievecard: få et beløb udbetalt

Giftcard: få penge fra alle andre spillere.

Recieveifcard: matador legatet.

Queenscard: frikort til fængslet. beholdes (altså sætter Player variablen 'outofjailcard' til true).

Kortet bliver sat nederst i bunken af kort med det samme, altså uafhængigt om spilleren har benyttet sig af kortet).

Ingen af kortene i bunken er af instancen Movecard eller Card. Derfor er klasserne lavet abstract.

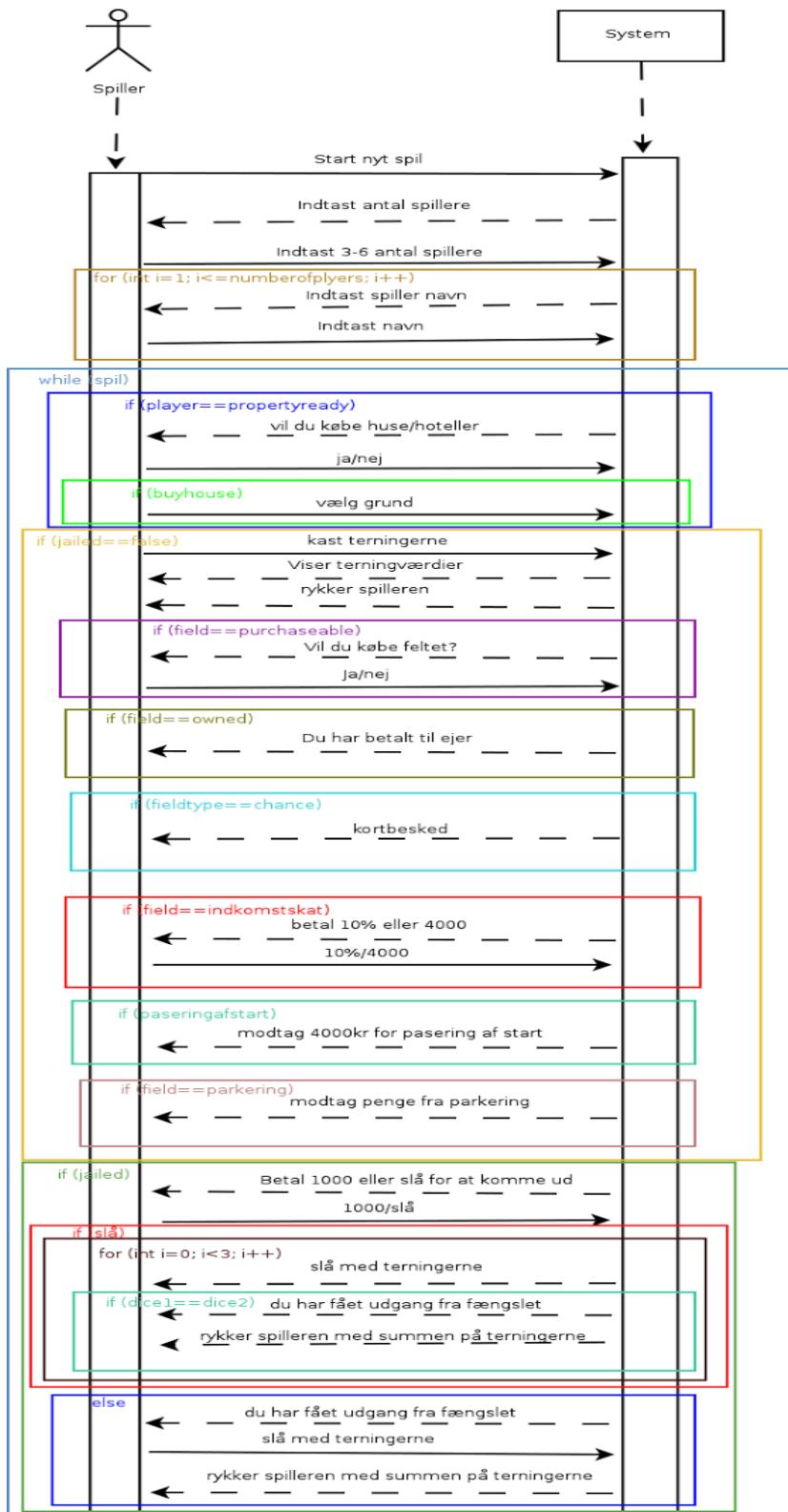
System sekvens diagram

Beskrivelse

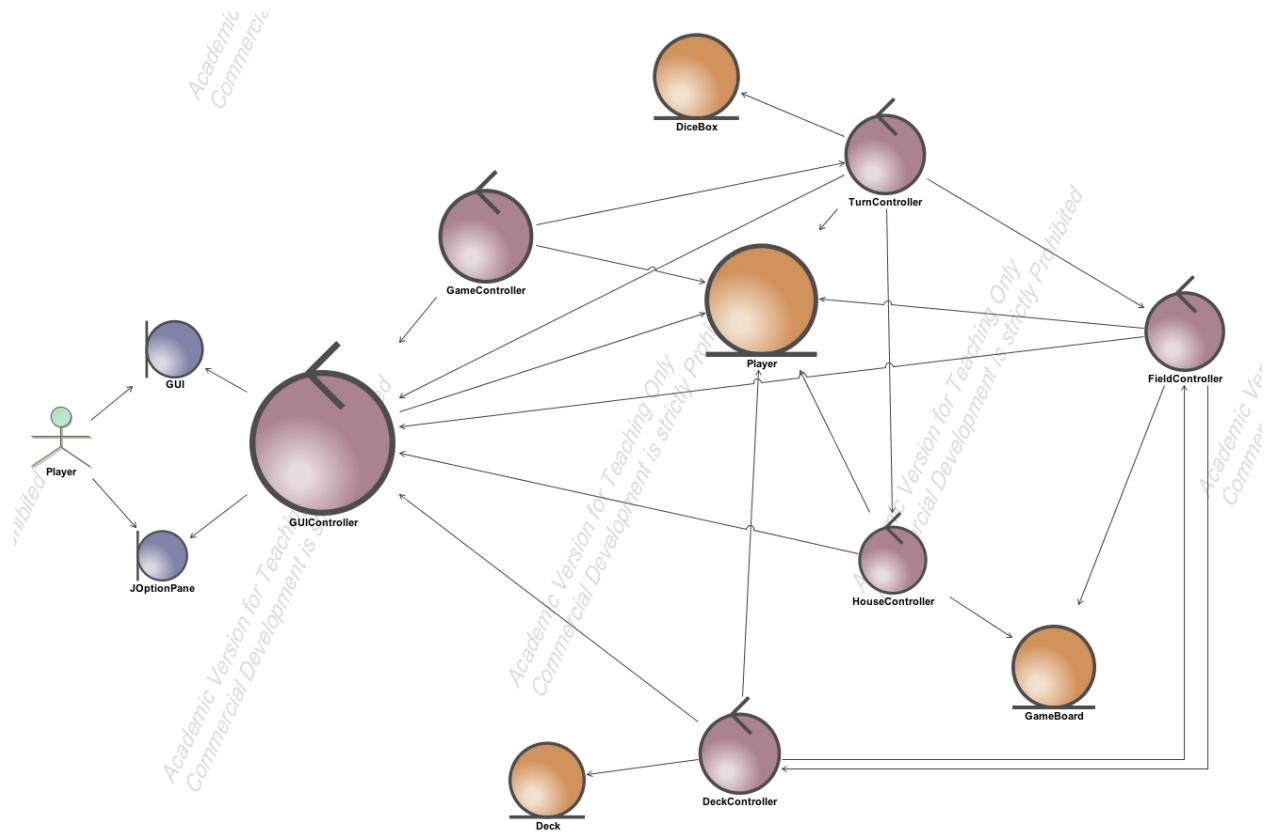
På figuren nedenfor kan man se vores System sekvens diagram, her kan man få et indblik i systemet og de interaktioner som foregår mellem systemet og aktøren spiller. Man kan altså på diagrammet se, at spilleren starter med at starte spillet. Spilleren får herefter besked på at indtaste et antal spillere på dem som skal spille. Herefter kommer vi ind i en for løkke, som får spilleren til at indtaste et navn indtil dette antal af spillere er opfyldt. Nu kommer vi så ind i en while løkke ser på om spillet er i gang og at der ikke allerede er fundet en vinder. Så bliver spilleren nu spurgt om personen har lyst til at købe nogle huse.

Såfremt spilleren ikke er fængslet, kaster spilleren fortsat med terningerne, og brugeren bliver herefter flyttet. Hvis det er tilfældet at en grund kan købes, bliver brugeren spurgt om denne vil købe. Hvis det nu er tilfældet at man lander på et felt som allerede er ejet, bliver ens penge betalt til denne spiller. Herefter er der chancekort som man også kan lande på, hvis man lander på sådan et felt, får man en besked om kortet man har trukket, efterfulgt af en handling som systemet sørger for. Nu er der så en indkomstskat hvor man enten kan vælge at betale 4000 point eller betale 10% af sin point beholdning. Nu er der så 2 måder at modtage penge på, dette er ved enten at lande på parkering eller at modtage penge efter at have passeret start. Dette er to måder hvorpå du kan modtage penge, ud over at modtage penge fra felter man selv ejer.

I dette spil kan du dog også blive fængslet, her kan du komme ud såfremt du betaler 1000 point, slår par eller efter 3 ture. Til sidst har vi altså hvis man blot lander på fængslet, hvor der ikke sker noget. Mange af felternes handlinger bliver primært udført af systemet til spilleren frem for at spilleren skal lave nogle interaktioner.



BCE model



Vi har oprettet 6 controllere: **GUIController**, **GameController**, **TurnController**, **HouseController**, **FieldController** og **DeckController**.

De har alle relation til **Player** entity. På den måde kan alle controllere altid refererer til hinanden ved brug af en specifik spiller, eller med et spiller array. Alle controllere har samtidig funktioner, der kan ændre værdier i spilleren.

GUIController skal hjælpe controllerne med at simplificerer de GUI processer der kommer i spillet. De 2 benyttede interfaces er den udleverede GUI samt `javax.swing.JOptionPane`. **JOptionPane** bliver brugt under oprettelsen af spillet, dette vil opdele de 2 processer, bruger skal igennem for henholdsvis at starte spillet og spille spillet.

JOptionPane og den udleverede GUI har ikke kontakt med aktøren samtidig.

GUIController tager også fat i player data, da controllerne kun skal refererer til en specifik player, hvor **GUIController**en vil tage sig af resten.

FieldController holder styr på de feltyper der bliver landet på, og med dens `landOnField()` metode, kører controlleren også igennem de GUI processer, der kræves på den specifikke felstype.

Fieldcontrolleren tager altså fat i DeckControlleren, hver gang der bliver landet på et 'prøv lykken' felt.

DeckControlleren vender tilbage til FieldControlleren, hvis korttypen der bliver landet på, ændre spillerens position. Så kan FieldControlleren udfører en ny landOnField() for den nye position.

DeckControlleren har samtidig en indbygget funktion, så når alle 30 kort i kortbunken er trukket, bliver kortbunken blandet igen. Dette sikre at spillerne ikke kommer til at kende kortbunkens rækkefølge.

TurnControlleren indeholder 2 turtyper der bliver kørt hendholdsvis hvis spilleren er i fængsel, eller spilleren skal have en normaltur. Den normale tur indeholder kun et slag fra terningen, og så en flytning af spilleren, hvorefter den executer landOnField() igennem Fieldcontrolleren. Fængselsturen afhænger af om spilleren har et frikort fra fængslet, og hvis ikke, afhænger det af spillerens valg om at slå for at komme ud, eller betale sig fra det.

Den holder også styr på turen der måske kommer, hvis spilleren slår sig ud af fængslet (intet slag, kun et ryk).

HouseControlleren holder styr på hvilke felter spilleren kan købe huse på. Den tilbyder så at bygge huse på disse grunde gennem GUIControlleren. HouseControlleren holder også styr på, om der bliver bygget jævnt på grundene. Dette bliver ordnet igennem nogle husfarve værdier i Player klassen.

GameControlleren styre rækkefølgen spillet spilles, samt at skifte spiller for hver tur, inden den beder TurnControlleren om at køre næste tur.

GameControlleren holder også styr på om spillet er vundet, og afslutter spillet, hvis dette er tilfældet, samt at initierer spillet.

Kode

Controllere

I vores GameController definerede vi 3 hovedprocesser, som et matadorspil består af: en setup fase, en spilfase og en fase, hvor vinderen af spillet vises.

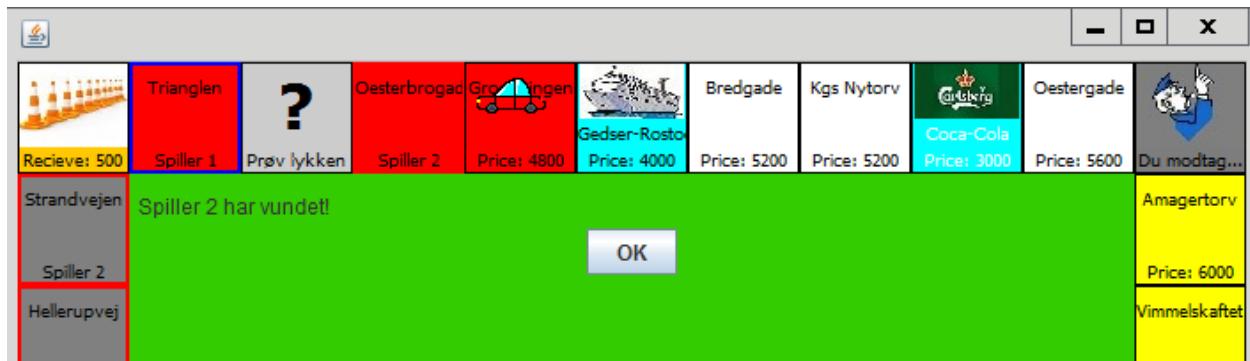
```
26     public void init() {  
27         setupGame();  
28         runGame();  
29         showWinner();  
30     }  
31 }
```

setupfasen er den fase hvor antal af spillere, samt navne på spillere, indtastes. Denne fase, benytter en anden GUI end resten af matadorspillet, da vi ville holde spil fasen og setup fasen adskilt for brugeren. setupfasen bruger java GUI'en 'JOptionPane', som vist på billede nedenfor.



spilfasen indeholder selve spillet, som er kørende, indtil 1 spiller er tilbage i spillet, og de andre er bankerot.

Den sidste fase der viser vinderen af spillet i matador GUI'en, er ikke andet end en simpel besked til spilleren der vandt, som vist i billedet nedenfor, hvorefter matador GUI'en lukker ned.



Når GameControlleren kører runGame() metoden, går spillet ind i en while løkke, der først stopper, når spillet er vundet. GameControlleren holder styr hvor mange spillere der stadig er med i spillet, og hvornår spillet skal afbrydes. Derudover holder den styr på hvilken spiller

tur, der skal køres. Dette fortæller den til TurnControlleren, der så kører den enkelte spillerens tur.

```
58    public void runGame() {
59        // The game continues as long as won equals false
60        while (!won) {
61            if (!playerlist[currentPlayer].getStatus()) {
62                // Runs turn for current Player
63                TurnC.runTurn(currentPlayer);
64                GUIC.newPosition(playerlist[currentPlayer]);
65                // If a player has lost, adds one to lostCount and reset the players owned fields
66                if (playerlist[currentPlayer].getStatus()) {
67                    GUIC.removePlayer(playerlist, currentPlayer);
68
69                gameboard.resetOwnedFields(playerlist[currentPlayer]);
70                lostCount++;
71
72                // If only one player is left, won is set to true
73                if (lostCount == playerAmmount - 1) {
74                    won = true;
75                    GUIC.showWin(playerlist, playerAmmount);
76                }
77            }
78        } // Changes player
79        changePlayer();
80    }
81
82    . . . . .
```

Når turen for spilleren skal køres, tjekker TurnControllern først, hvilken tur type spilleren skal have. Der er 2 tur typer; en fængsels tur og en normal tur. TurnControlleren tjekker selv, hvilken tur type spilleren skal have, ved at tjekke på spillerens fængsel status.

```
30    public void runTurn(int currentPlayer) {
31        if (playerlist[currentPlayer].isJailed())
32            runJailTurn(currentPlayer);
33        else
34            runNormalTurn(currentPlayer);
35    }
36
```

Fængselsturen har 3 former for ture. En hvis man har trukket et chancekort, med en fængsels benådelse fra dronningen. En hvis man vil betale for at komme ud, og en sidste, hvis man vil prøve at slå, for at komme ud - kræver 3 ens, hvor man har 3 forsøg pr. tur.

```

36 public void runJailTurn(Player[] playerlist, int currentPlayer){
37     // If player has a getoutofjailcard
38     if (playerlist[currentPlayer].hasOutofjailcard()) {
39         exitCard(playerlist[currentPlayer], currentPlayer);
40         runNormalTurn(playerlist, currentPlayer);
41     }
42     // If player pays for exit
43     else if (GUIC.jailOptions(playerlist[currentPlayer])) {
44         exitPay(playerlist[currentPlayer], currentPlayer);
45         runNormalTurn(playerlist, currentPlayer);
46     }
47     // If player wants to throw the dice for exit
48     else
49         exitThrow(playerlist, currentPlayer);
50 }

```

Spilleren har valget om at betale for at komme ud, hver tur, inden han vælger at slå for at komme ud. Hvis man betaler for at komme ud, eller har et frikort, får man bagefter en normal tur (se linje 40 og 45 på kildekoden ovenover). Kommer man derimod ud med et slag, får man en 'afterJailTurn', som kun rykker dig summen af terningerne - altså ikke noget ekstra slag, efter man er kommet ud. Prøver man at slå sig ud 3 gange uden det lykkes, bliver man tvunget til at betale 1000 kr. Derudover, får man ikke lov til at slå eller rykke. Player klassen har en lokal variabel, der holder styr på hvor mange ture, spilleren har siddet i fængsel: 'jailcount'.

```

106 public void exitCard(int currentPlayer) {
107     playerlist[currentPlayer].setJailed(false);
108     playerlist[currentPlayer].setJailcount(0);
109 }
110
111 public void exitPay(int currentPlayer) {
112     playerlist[currentPlayer].account.addPoints(-1000);
113     playerlist[currentPlayer].setJailed(false);
114     playerlist[currentPlayer].setJailcount(0);
115 }
116 public void exitThrow(int currentPlayer) {
117     if (playerlist[currentPlayer].getJailcount() < 3) {
118         for (int i=1; i<=3; i++) {
119             GUIC.showJailTurn();
120             box.rollDice();
121             GUIC.showDice(box.getDice1(), box.getDice2());
122             if (box.isEqual()) {
123                 playerlist[currentPlayer].movePosition(box.getSum());
124                 board.getField(playerlist[currentPlayer].getPosition()).landOnField(playerlist[currentPlayer]);
125                 playerlist[currentPlayer].setJailed(false);
126                 playerlist[currentPlayer].setJailcount(0);
127                 afterJailTurn(currentPlayer);
128                 break;
129             }
130         }
131         // If the player didn't get out
132         if (playerlist[currentPlayer].isJailed())
133             playerlist[currentPlayer].addJailcount();
134     }
135     // If the playerlist[currentPlayer] tried exiting 3 times (forced pay)
136     if (playerlist[currentPlayer].getJailcount() == 3) {
137         GUIC.showJailForcedPay();
138         playerlist[currentPlayer].account.addPoints(-1000);
139         playerlist[currentPlayer].setJailed(false);
140         playerlist[currentPlayer].setJailcount(0);
141     }
142 }

```

Den normale tur består af 2 faser. Først skal man vælge om man vil sælge eller købe huse, og når man selv beslutter at man er færdigt med køb og salg af huse, vælger spilleren at rulle med terningerne, som går over i turens næste fase. Når man vælger at købe eller sælge huse, tager TurnControlleren fat i HouseControlleren, som holder styr på hvilken grunde der kan købes og sælges huse på. Ved køb, sørger HouseControlleren for, at der bliver bygget jævnt. Ved salg, er dette ikke tilfældet. Så man kan altså sælge ujævnt, men kun købe jævnt. 2. fase, hvor man slår med terningerne, holder TurnControlleren styr på om der bliver slået par, hvis der gør, får spilleren en ekstra tur. Samtidig holder den styr på, at hvis der bliver slået par 3 gange i træk, rykker spilleren i fængsel.

Fase 1

```
208     public void optionsStartOfTurn(int currentPlayer){  
209         boolean keepBuyingSelling = true;  
210         while(keepBuyingSelling){  
211             choiceofTurn = GUIC.startOfTurn(playerlist, currentPlayer);  
212             if(choiceofTurn.equals("Koeb hus")){  
213                 buyHouse(currentPlayer);  
214             }  
215             else if(choiceofTurn.equals("Saelg hus")){  
216                 sellHouse(currentPlayer);  
217             }  
218             else if(choiceofTurn.equals("rul terning")){  
219                 keepBuyingSelling = false;  
220             }  
221         }  
222     }
```

Fase 2 (her er vist hele kørslen af en tur, fase 1 bliver kørt i linje 57, fase 1 bliver kørt i denne metode)

```

54    public void runNormalTurn(int currentPlayer) {
55        int count = 0;
56        boolean run = true;
57        optionsStart0fTurn(currentPlayer);
58        while(run){
59            if (count==2)
60                GUIC.twoPair();
61            else if (count==1)
62                GUIC.onePair();
63            // For demo or normal game run
64            if (mode==1){
65                box.rollDice();
66                GUIC.showDice(box.getDice1(), box.getDice2());
67            } else if (mode==2) {
68                box.setDice(0, 1);
69                box.setDice(1, 0);
70                GUIC.showDice(box.getDice1(), 1);
71            }
72
73            if(box.getSum() + playerlist[currentPlayer].getPosition() >= 40) {
74                GUIC.pastStart();
75                GUIC.updateBalance(playerlist[currentPlayer].getName(), playerlist[currentPlayer].account.getScore());
76            }
77            if (count !=3) {
78                GUIC.updatePosition(playerlist, currentPlayer, box.getSum());
79                FC.landOnField(currentPlayer);
80            }
81            if (playerlist[currentPlayer].isJailed())
82                run = false;
83            else if (box.isEqual()){
84                count++;
85            } else
86                run = false;
87            if (count>=3){
88                run = false;
89                playerlist[currentPlayer].setJailed(true);
90                playerlist[currentPlayer].setPosition(10);
91                GUIC.threePair();
92                GUIC newPosition(playerlist[currentPlayer]);
93            }
94        }
95    }

```

For hver tur, kontrollerer FieldControlleren hvor spilleren har landet, (se linje 79, på kildekoden ovenover). Når der landes på et felt, holder FieldControlleren styr på hvilken type felt der landes på. Udfra dette ved FieldControlleren, hvilken input der evt. skal komme fra spilleren, hvilken GUI beskeder spilleren skal have, og ellers, hvilken ekstra funktioner feltet har - som felt klassen, ikke ved noget om. Eksempler på dette er feltypen ChanceCard, hvor FieldControlleren får DeckControlleren til at trække et kort. Til sidst kører FieldControlleren en ekstra update på GUI.

```

21@ public void landOnField(int currentPlayer) {
22    // For Territories
23    if (gameboard.getField(playerlist[currentPlayer].getPosition()) instanceof fields.Territory)
24        landOnTerritory(currentPlayer);
25    // For Fleets
26    else if (gameboard.getField(playerlist[currentPlayer].getPosition()) instanceof fields.Fleet)
27        landOnFleet(currentPlayer);
28    // For LaborCamps
29    else if (gameboard.getField(playerlist[currentPlayer].getPosition()) instanceof fields.LaborCamp)
30        landOnLaborCamp(currentPlayer);
31    // For ChanceCard
32    else if (gameboard.getField(playerlist[currentPlayer].getPosition()) instanceof fields.ChanceCard)
33        DC.drawCard(currentPlayer);
34    // For MoveToJail
35    else if (gameboard.getField(playerlist[currentPlayer].getPosition()) instanceof fields.MoveToJail) {
36        GUIC.moveToJail();
37        gameboard.getField(playerlist[currentPlayer].getPosition()).landOnField(playerlist[currentPlayer]);
38        GUIC newPosition(playerlist[currentPlayer]);
39    }
40    // For Tax
41    else if (gameboard.getField(playerlist[currentPlayer].getPosition()) instanceof fields.Tax)
42        landOnTax(currentPlayer);
43    // For Refuge
44    else if (gameboard.getField(playerlist[currentPlayer].getPosition()) instanceof fields.Refuge)
45        landOnRefuge(currentPlayer);
46    // For Start
47    else if (gameboard.getField(playerlist[currentPlayer].getPosition()) instanceof fields.Start)
48        GUIC.startMessage(gameboard.getField(playerlist[currentPlayer].getPosition()).getName());
49    // For every other fields
50    else
51        gameboard.getField(playerlist[currentPlayer].getPosition()).landOnField(playerlist[currentPlayer]);
52
53    // Update on GUI
54    for (int i = 0; i<playerlist.length; i++) {
55        GUIC.newPosition(playerlist[i]);
56        GUIC.updateBalance(playerlist[i].getName(), playerlist[i].account.getScore());
57    }
58}

```

Når der landes på et ‘prøv lykken’ felt, fortæller FieldControlleren at der skal trækkes et kort til DeckControlleren. DeckControlleren trækker det øverste kort i Deck klassen, som så selv lægger kortet bag i bunken, samt flytter alle kortene en plads op (op visualiseret i kortbunken, men ned i array pladserne), som set på kildekoden nedenunder:

```

56@ public Card drawCard(Player player) {
57    cardlist[0].drawCard(player);
58    // puts the card in the bottom of the list/array
59    Card temp = cardlist[0];
60    for (int k = 1; k<cardlist.length; k++){
61        cardlist[k-1] = cardlist[k];
62    }
63    cardlist[cardlist.length-1]=temp;
64    return temp;
65}

```

Samtidig returnere denne metode et kort. Dette kort bliver brugt af DeckControlleren til at kontrollere hvilken korttype der blev trykket. For uover at udføre drawCard() funktionen i det specifikke kort (ligger i de forskellige kort klasser), kontrollerer DeckControlleren om kortet der er trukket, er et MoveCard. Hvis dette er tilfældet vender DeckControlleren tilbage til FieldControlleren, og udfører en ny landOnField() på spillerens nye position - se linje 36 og 37, på kildekoden nedenunder:

```

27     public void drawCard(int currentPlayer) {
28         GUIC.showMessage(deck.getMessage(0));
29         Card temp = deck.drawCard(playerlist[currentPlayer]);
30         cardsdrawn++;
31         GUIC newPosition(playerlist[currentPlayer]);
32         if (cardsdrawn >= decklength) {
33             deck.shuffleDeck();
34             cardsdrawn = 0;
35         }
36         if (temp instanceof MoveCard)
37             FC.landOnField(currentPlayer);
38         // ekstra update on GUI
39         GUIC.updateBalance(playerlist[currentPlayer].getName(), playerlist[currentPlayer].account.getScore());
40     }

```

Derudover holder DeckControlleren øje med hvor mange kort der er trukket fra kortbunken. Når alle 30 kort er trukket blander DeckControlleren kortlisten igen. Dette er for at sikre, at spillerne ikke kan gennemske kortbunken.

Sammenspillet imellem DeckControlleren og FieldControlleren er lavet, så controllerne har kendskab til hinanden, se linje 18 i kildekoden nedenunder.

```

14     public FieldController(GUIController GUIC, GameBoard gameboard, Player[] playerlist) {
15         this.GUIC = GUIC;
16         this.gameboard = gameboard;
17         this.playerlist = playerlist;
18         DC = new DeckController(GUIC, playerlist, gameboard, this);
19     }

```

Dette er gjort for at sikre, at der kun bliver oprettet én af hver controller, når spillet startes.

HouseControlleren holder styr på hvilken grunde der kan købes huse på, samt hvilken grunde der kan sælges huse på.

Metoden buyHouse() tilbyder spilleren at købe huse, indtil han ikke vil købe flere, eller han ikke har mulighed for at købe flere huse:

```
23 public void buyHouse(int currentPlayer){  
24     //check if you can buy.  
25     boolean moreHouses = true;  
26     while(moreHouses== true){  
27         for(int i=1; i<=8; i++){  
28             if(getPriceAndValue(currentPlayer, i, board) == true){  
29                 for(int q=1; q<=8; q++){  
30                     if(getBuild(q, currentPlayer) == true){  
31                         buildPlots(currentPlayer ,q);  
32                     }  
33                 }  
34             }  
35         }  
36         if(GUIC.offerMoreHouses()==false){  
37             moreHouses=false;  
38         }  
39     }  
40 }  
41 }
```

Metoden checkPossibleSell() tjekker hvilken grunde der kan sælges huse på, hvorefter den returnerer et String array med mulige valg af hussalg:

```
60 public String[] checkIfPossibleSell(int currentPlayer, GameBoard board){  
61     int arrayIndex = 0;  
62     int arraylength = 0;  
63     for(int i=1; i<=39; i++){  
64         if(board.getField(i).getNumberofhouses() > 0 && playerlist[currentPlayer].equals(board.getField(i).getOwner())){  
65             arraylength++;  
66         }  
67     }  
68     sellOptions = new String[arraylength];  
69     for(int i=1; i<=39; i++){  
70         if(board.getField(i).getNumberofhouses() > 0 && playerlist[currentPlayer].equals(board.getField(i).getOwner())){  
71             sellOptions[arrayIndex] = board.getField(i).getName();  
72             arrayIndex++;  
73         }  
74     }  
75     return sellOptions;  
76 }  
77 }
```

metoden sellHouse() sælger et bestemt hus, ved at sammenligne en String for grundens navn, med felterne på brættet. Når den har fundet feltet opdatere metoden spillerens balance, samt opdatere antal huse vist på GUI'en, samt køre en opdatering på spillerens balance på GUI'en:

```

44 //sell house
45 @
46 public void sellHouse(int currentPlayer, GameBoard board, String plot){
47     for(int i=1; i<=39; i++){
48         if(board.getField(i).getName().equals(plot)){
49             playerlist[currentPlayer].account.addPoints(getHousePrice(1)/2);
50             board.getField(i).addNumberofHouses(-1);
51             if(board.getField(i).getNumberofhouses()==5){
52                 GUIC.setHotel(i+1, false);
53                 removeHotel(currentPlayer);
54             }else{
55                 playerlist[currentPlayer].addHouseammount(-1);
56                 GUIC.setHouse(i+1, board.getField(i).getNumberofhouses());
57             }
58         }
59     }
60     GUIC.updateBalance(playerlist[currentPlayer].getName(), playerlist[currentPlayer].account.getScore());
61 }
```

Entiteter

Vores Deck klasse indeholder et array over alle chancekortene:

```

9@ public Deck(Player[] playerlist, GameBoard board){
10    cardlist[0] = new MoveToCard("Ryk frem til Grønningen. Hvis de passerer \"Start\", indkasser da 4000kr ", 24, board);
11    cardlist[1] = new MoveToFleetCard("Ryk brikkens frem til det nærmeste rederi og betal ejeren to gange den leje, "
12        + "han ellers er berettiget til, hvis selskabet ikke ejes af nogen kan De købe det af banken.", board);
13    cardlist[2] = new MoveToCard("Ryk frem til \"Start\"", 0, board);
14    cardlist[3] = new MoveAmmountCard("Ryk tre felter tilbage", -3, board);
15    cardlist[4] = new MoveToCard("Ryk frem til Frederiksberg Alle. Hvis de passerer \"Start\", indkasser da 4000kr", 11, board);
16    cardlist[5] = new MoveToCard("Tag med Mols-Linjen -- Flyt brikkens frem og hvis de passerer \"Start\", indkasser da 4000kr", 15, board);
17    cardlist[6] = new MoveToCard("Tag ind på Rådhuspladsen", 39, board);
18    cardlist[7] = new JailCard("Gå i Fengsel. Ryk direkte til fængsel. Selv om De passerer \"Start\", indkassere de ikke 4000kr", board);
19    cardlist[8] = new JailCard("Gå i Fengsel. Ryk direkte til fængsel. Selv om De passerer \"Start\", indkassere de ikke 4000kr", board);
20    cardlist[9] = new PayCard("Betal 3000kr for reparation af Deres vogn", 3000);
21    cardlist[10] = new PropertyPayCard("Oliepriserne er steget, og de skal betale: \n 500kr pr. hus \n 2000kr pr. hotel", 500, 2000);
22    cardlist[11] = new PayCard("Har måtte vedtage en parkeringsbøde. Betal 200kr i både", 200);
23    cardlist[12] = new PayCard("Betal deres bilforsikring på 1000kr", 1000);
24    cardlist[13] = new PayCard("Betal 3000kr for reparation af deres vogn", 3000);
25    cardlist[14] = new PayCard("De har været en tur i udlandet og hafft for mange cigaretter med hjem. Betal told 200kr", 200);
26    cardlist[15] = new PayCard("De har kørt frem for \"Full Stop\". Betal 1000kr i både", 1000);
27    cardlist[16] = new PayCard("De har modtaget Deres tænklægeregning. Betal 2000kr", 2000);
28    cardlist[17] = new PayCard("Ejendomsskatteerne er steget, Ekstra udgifter er: \n 800kr pr. hus \n 2300kr pr. hotel", 800, 2300);
29    cardlist[18] = new ReceiveCard("Nydelige vigtighederne er steget, Ekstra udgifter er: \n 800kr pr. hus \n 2300kr pr. hotel", 800, 2300);
30    cardlist[19] = new GiftCard("Det er Deres fødselsdag. Modtag af hver medspiller 200kr", 200, playerlist);
31    cardlist[20] = new ReceiveCard("De har vundet i Kasselotteriet. Modtag 500kr", 500);
32    cardlist[21] = new ReceiveCard("De havde en rekke med ellevte rigtige i tipning. Modtag 1000kr", 1000);
33    cardlist[22] = new ReceiveCard("Modtag udbytte af Deres aktier 1000kr", 1000);
34    cardlist[23] = new ReceiveCard("Grundet dyrtiden har De fået gageforhøjelse. Modtag 1000kr", 1000);
35    cardlist[24] = new ReceiveCard("Kommunen har eftergivet et kvartals skat. Henv i banken 3000kr", 3000);
36    cardlist[25] = new ReceiveCard("Modtag udbytte af Deres aktier 1000kr", 1000);
37    cardlist[26] = new ReceiveCard("Deres præmieobligation er kommet ud. De modtagter 1000kr af banken", 1000);
38    cardlist[27] = new ReceiveCard("De modtagter Deres aktieudbytte. Modtag 1000kr af banken", 1000);
39    cardlist[28] = new ReceiveCard("De modtagter \"Nataador-legatet for de værdige trængende\" stort 4000kr. "
40        + "Ved værdige trængende forstås, af Deres formue, "
41        + "d.v.s. Deres kontante penge + skærer + bygninger ikke overskrider 15000kr", 40000, 15000, board);
42    cardlist[29] = new QueensCard("I anledning af deres majestats fødselsdag benådes de herved for fængsel. "
43        + "Dette kort kan opbevares, indtil De får brug for det");
44}

```

Alle kortene er taget fra vores klasse hierarki over chancekortene. (se figur 9).

Når DeckControlleren oprettes, opretter controlleren et deck, med denne konstruktør.

Hvorefter DeckControlleren blander kortbunken.

Vores shuffleDeck() metoder benytter Fisher-Yates blandings algoritme. Denne algoritme tager et tilfældig tal imellem 0 og i - kaldt 'index'. Kortet på plads i, bliver midlertidig lagret i variablen a.

Tager så kortet på plads index, og sætter det til at være kortet på plads i. Kortet på plads i bliver så sat til at være vores kort 'a'.

Sådan kører algoritmen igennem hele bunken:

```

46@ public void shuffleDeck() {
47    // Fisher-Yates shuffle algorithm
48    Random r = new Random();
49    for (int i = cardlist.length-1; i>0; i--) {
50        int index = r.nextInt(i+1);
51        Card a = cardlist[index];
52        cardlist[index] = cardlist[i];
53        cardlist[i] = a;
54    }
55}

```

Samtidig har Deck klassen en metode der trækker et kort, og bagefter lægger kortet bag bunken.

Dette gøres ved at rykke alle kort en position op i arrayet (kort 0 ligger øverst, i vores optik), hvorefter det nederste kort bliver 'temp' kortet, altså det kort der blev trukket:

```
56    public Card drawCard(Player player) {  
57        cardlist[0].drawCard(player);  
58        // puts the card in the bottom of the list/array  
59        Card temp = cardlist[0];  
60        for (int k = 1; k<cardlist.length; k++){  
61            cardlist[k-1] = cardlist[k];  
62        }  
63        cardlist[cardlist.length-1]=temp;  
64        return temp;  
65    }  
66 }
```

I vores korttype RecievelfCard, som er matador legatet, har vi en algoritme der tjekker spillerens egenkapital. Denne algoritme kører også når der landes på feltet 'Statsskat', og spilleren vælger at betale 10% af deres egenkapital.

Algoritmen kører alle felter igennem, og tjekker om den nuværende spiller er ejer af felterne. Hvis det er tilfældet, lægges grundens værdi samt antal af huse gange huspriser, på den nuværende grund.

Egenkapitalen findes så ved at lægge kontanter, husværdier og grundværdier sammen.

I matador legatet, sammenlignes denne egenkapital med den tilladte egenkapital, for at se om spilleren er kvalificeret til at få bonusen udbetalt:

```
16    public void drawCard(Player player) {  
17        int networth;  
18        int ownableworth = 0;  
19        int buildingworth = 0;  
20        for (int i=0; i<40;i++) {  
21            if (player.equals(board.getField(i).getOwner())) {  
22                ownableworth += board.getField(i).getPrice();  
23                buildingworth += (board.getField(i).getNumberofhouses() * board.getField(i).getHouseprice());  
24            }  
25        }  
26        networth = player.account.getScore() + ownableworth + buildingworth;  
27        if (networth <= maxvalue)  
28            player.account.addPoints(bonus);  
29    }  
30 }
```

Vores Player klasse indeholder alle værdier der er unikke for spiller til spiller. Player klassen holder styr på spillerens position, spillerens tilstande, som fængsling, om de er med i spillet

eller er bankerot, om de har et frikort fra fængslet, om de skal betale dobbelt på et rederi (chancekort, der flytter dig til det nærmeste rederi (MoveToFleetCard)), og om de har lov til at bygge på de enkelte farver. Derudover holder Playerklassen styr på antal af huse og hoteller (bruges til chancekortet; PropertyPayCard, antal af grunde på diverse farvetyper, antal af rederier, antal af bryggerier, antal ture spilleren har siddet i fængsel og spillerens navn. Derudover indeholder Player klassen en Account, der holder styr på deres balance.

```
4 public class Player {
5     private String name;
6     private int position, fleets, laborCamp,
7         fieldammount_blue,fieldammount_pink,fieldammount_green,fieldammount_grey,
8         fieldammount_red,fieldammount_white,fieldammount_yellow,fieldammount_magenta,
9         houseammount,hotelammount, jailcount;
10    private boolean lost, outofjailcard,jailed,build_blue, build_pink, build_green,
11        build_grey, build_red, build_white, build_yellow, build_magenta, paydoublefleet;
12    public Account account = new Account();
13
14    // Object that stores the name and position of a player
15    public Player(String name) {
16        position = 0;
17        fleets = 0;
18        laborCamp = 0;
19        lost = false;
20        outofjailcard = false;
21        jailed = false;
22        jailcount = 0;
23        fieldammount_blue = 0;
24        fieldammount_pink = 0;
25        fieldammount_grey = 0;
26        fieldammount_green = 0;
27        fieldammount_red = 0;
28        fieldammount_white = 0;
29        fieldammount_yellow = 0;
30        fieldammount_magenta = 0;
31        houseammount = 0;
32        hotelammount = 0;
33        this.name = name;
34        build_blue = false;
35        build_pink = false;
36        build_green = false;
37        build_grey = false;
38        build_red = false;
39        build_white = false;
40        build_yellow = false;
41        build_magenta = false;
42        paydoublefleet = false;
43    }
```

Test

I forløbet med at lave programmet skulle vi også teste. Vi fulgte derfor en test-drevet udviklingsmetode. Altså testede vi efterhånden som vi lavede programmet, dette gjorde vi for at bekræfte at henholdsvis de metoder vi implementerede, efterhånden virkede. I dette afsnit tager vi udgangspunkt i nogle metoder som vi benytter os af, og tests spillets overordnede logik. Vi tager fat i henholdsvis deck klassen som sørger for vores chancekort og udfaldet af dem, vi tester også på at husene visuelt bliver fremvist på den udleverede GUI, samt på fængslet. Nedenfor tager vi udgangspunkt i nogle af disse scenarier.

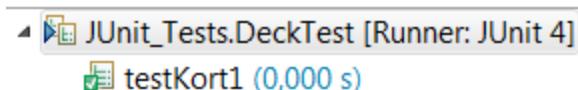
Deck test

Test af kort1

Da vi nu skulle have chancekort, blev vi nødt til at teste på kortene for at se om de gjorde det rigtige. Derfor lavede vi en test på hvert af korttyperne man kunne modtage, som set på domænemodellen for "card hierarki". For at komme lidt ind på nogle af disse, har vi først testet på et kort af typen "Moveto", her skulle en spiller blot blive flyttet over til et felt. Her kan vi se koden nedenfor:

```
18 @Test
19 public void testKort1() {
20     //Preconditions
21     DiceBox box = new DiceBox();
22     GameBoard board = new GameBoard(box);
23     Player[] players = new Player[3];
24     Deck deck = new Deck(players, board);
25     GUIcontroller GUIC = new GUIcontroller();
26     FieldController FC = new FieldController(GUIC, board, players);
27
28     players[0] = new Player("Spiller1");
29     players[1] = new Player("Spiller2");
30     players[2] = new Player("Spiller3");
31     players[0].setPosition(0);
32     //Test
33     //Move to "Groenningen"
34     deck.pickCard(players[0], 0);
35
36     //Postconditions
37     //Tjek om spilleren er på% feltet
38     assertEquals(24, players[0].getPosition());
39
40 }
```

På testen ovenfor kan man se at vi har vores test, som indeholder nogle objekter, disse er henholdsvis DiceBox, gameboardet, spillerne, decket, guicontrolleren og feltcontrolleren. Herefter har vi 3 spillere som bliver oprettet, disse skal være vores test aktører i kørslen. Nu går vi videre til testen i sig selv, denne består blot af at vi tager et specifikt kort ved brug af pickCard metoden. Så kommer postconditions hvor vi kigger på om spillerens position nu er 24, altså Grønningen. Ud fra denne JUnit test fik vi at testen bestod, at spilleren altså blev flyttet til felt 24 på gameboardet, som set på figuren nedenfor:



Nedenfor er vist en kort testrapport i JUnit over testens kørsel.

Testrapport af JUnit test for spiller trækker første chancekort.	Dato: 11/01/2015 Commit: bee177e58652050c9b193df6b096bfcb3f55fa28
Pre 1: Opret spiller (TestPerson)	Ok
Pre 2: Opret spiller (TestPerson2)	Ok
Pre 3: Opret spiller (TestPerson3)	Ok
Test 1: Træk første kort.	Ok
Post 1: Er CurrentField = 24?	Ok

På baggrund af dette kunne vi konkludere at testkort 1 virkede.

Test af kort 10

Ved dette kort trækkes der penge for reparation af bilen. Her skal vi altså tjekke på om spilleren har mistet penge. Vi har altså testet dette ved brug af JUnit så vi kan køre en serie af tests og se om der er nogle som giver os fejl. Nedenfor har vi opstillet koden til at udføre denne JUnit test på:

```
217 @Test
218 public void testKort10(){
219     //Preconditions
220     DiceBox box = new DiceBox();
221     GameBoard board = new GameBoard(box);
222     Player[] players = new Player[3];
223     Deck deck = new Deck(players, board);
224
225     players[0] = new Player("Spiller1");
226     players[1] = new Player("Spiller2");
227     players[2] = new Player("Spiller3");
228
229
230     //We land on chance card at field 7.
231     players[0].setPosition(7);
232
233
234
235     //Test
236     //Repair on car, 3000 bill.
237     deck.pickCard(players[0],9);
238
239
240
241     //Postconditions
242     //Check if the player lost 3000.
243     assertEquals(27000,players[0].account.getScore());
244
245 }
```

Man kan på billedet ovenfor se koden for testen, vi starter her med at have opsat vores preconditions som består af DiceBox, gameboard, et spiller array og deck. Herefter bliver spillerne initialiseret idet vi definerer pladserne i arrayet, herefter lander vi blot på chancekort feltet. Nu kommer vi så til test delen, her trækker vi så kort 9, som lyder på at der er kommet en reparation på bilen, og man derfor skal betale 3000.

Nu ser vi så i postconditions om spilleren rent faktisk blev trukket for penge i sin pengebeholdning. Vi benytter altså igen assertEquals til at se på om pengene er blevet trukket. Herefter fik vi fremvist resultatet som kan ses på billedet nedenfor:

 testKort10 (0,000 s)

Denne test bestod altså også, hvilket ville sige at testen gik igennem. Nu opsætter vi en testrapport nedenfor af programmets kørsel:

Testrapport af JUnit test for spiller trækker tiende chancekort.	Dato: 11/01/2015 Commit: bee177e58652050c9b193df6b096bfcb3f55fa28
Pre 1: Opret spiller (TestPerson)	Ok
Pre 2: Opret spiller (TestPerson2)	Ok
Pre 3: Opret spiller (TestPerson3)	Ok
Test 1: Træk tiende kort.	Ok
Post 1: Er TestPerson getscore == 27000?	Ok

Altså måtte det betyde at testen gik fejlfrit igennem.

Test af Jail

Når man lander på feltet Jail, bliver man rykket til jail og der er derfor nogle kriterier for om man må komme ud igen. Det kigger vi på i koden nedenfor:

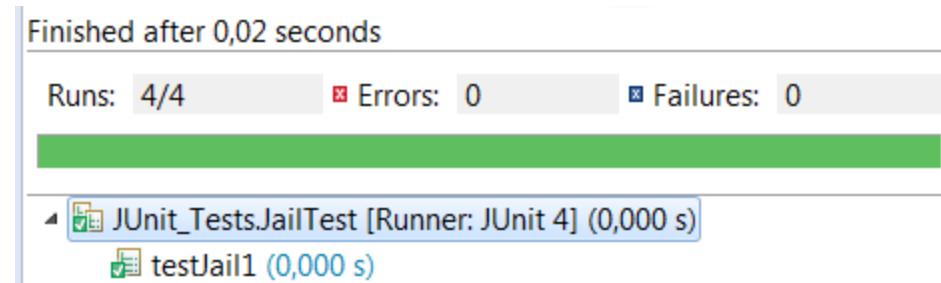
```
@Test
public void testJail1() {
    // Tests the getoutofjailcard
    DiceBox box = new DiceBox();
    GameBoard board = new GameBoard(box);
    Player[] playerlist = new Player[3];
    playerlist[0] = new Player("Spiller1");
    playerlist[1] = new Player("Spiller2");
    playerlist[2] = new Player("Spiller3");
    GUIcontroller GC = new GUIcontroller();
    TurnController TC = new TurnController(GC, board, playerlist);

    // Preconditions
    playerlist[0].setOutofjailcard(true);
    playerlist[0].setJailed(true);
    playerlist[0].setPosition(10);

    // Test
    TC.exitCard(playerlist[0], 0);

    // Postconditions
    assertEquals(false, playerlist[0].isJailed());
}
```

I koden starter vi med at sætte spilleren til at have et outofjailcard som man får fra chancekortene. Herefter sætter vi spilleren til at være jailed ved at sætte den true, nu flytter vi så spilleren til position 10 som altså er jail. Da dette nu er gjort bruger vi vores kort til at komme ud af fængslet igen. Vi tjekker nu efter med assertEquals for at se om spilleren stadigvæk er fængslet, og får altså resultatet som set på nedenstående billede:



Altså må testen have kommet igennem uden problemer.

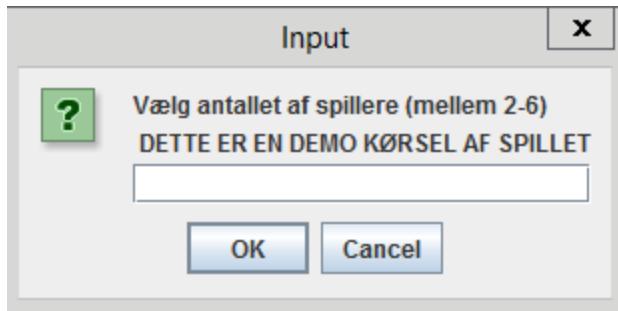
Showcase

I denne testcase udførte vi en test på alle felter som man kan lande på. Dette gør vi ved at lave et såkaldt autoraflebærger, som slår 1 med en terning hver gang vi kaster med terningerne. Dette gør altså at vi kan teste for konsekvenserne ved at lande på alle felterne på brættet. Denne demo er smart at udføre da vi kan komme til at lande på alle felterne, altså har vi nemt mulighed for at købe forskellige felter til at teste konsekvensen på dem, samt muligheden for at købe henholdsvis huse og hoteller på dem.

Ved at bruge denne showcase kan vi også se på om der bliver trukket de rigtige beløb og at alle felterne samt om man bliver rykket de rigtige steder hen. For eksempel benyttede vi denne for at opklare at man ikke rykkede i jail efter at have landet på movetojail fletet, vi får altså afsløret nogle meget basale bugs idet vi kører denne.

Denne showcase er også smart hvis man fik til opgave at vise produktet samt dens funktioner frem. Ved denne showcase, kan man få et godt indblik i hvordan systemet fungerer og hvad der sker hvis man lander på de forskellige felter. idet man lander på dem. Set fra kodens perspektiv har vi at vi kan kører 2 forskellige versioner af spillet, hvis vi indsætter at vi vil spille version 1 af spillet, spiller man blot det normale spil, men kører man showcasesen, spiller vi den anden version som gør at man kun slår 1 med en terning. Showcasen kan findes sammen med den oprindelige run.

Imodsætning til den normale spilkørsel, kan en showcase gennemkørsel af spillet, modtage 2-6 spillere, frem for 3-6.



Code Coverage

Vi har benyttet code coverage for at se på hvor meget af koden vi får testet i vores testSuite (alle vores JUnit-tests samlet).

Resultater af code coverage:

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
53_Final	38.7 %	4,452	7,059	11,511
src	38.7 %	4,452	7,059	11,511
control	15.2 %	769	4,298	5,067
HouseController.java	23.5 %	557	1,815	2,372
FieldController.java	1.4 %	21	1,525	1,546
TurnController.java	19.3 %	155	649	804
GameController.java	0.0 %	0	231	231
DeckController.java	31.6 %	36	78	114
boundary	3.8 %	55	1,405	1,460
GUIController.java	3.8 %	55	1,405	1,460
fields	51.1 %	906	868	1,774
Territory.java	41.0 %	191	275	466
Fleet.java	12.6 %	30	209	239
LaborCamp.java	8.3 %	12	132	144
Tax.java	12.9 %	16	108	124
GameBoard.java	90.1 %	589	65	654
Field.java	21.1 %	8	30	38
Refuge.java	27.8 %	10	26	36
Start.java	25.0 %	4	12	16
ChanceCard.java	57.1 %	4	3	7
Jail.java	57.1 %	4	3	7
Ownable.java	90.0 %	27	3	30
MoveToJail.java	84.6 %	11	2	13
entity	54.7 %	329	273	602
Player.java	57.6 %	231	170	401
DiceBox.java	48.0 %	59	64	123
Dice.java	27.7 %	13	34	47
Account.java	83.9 %	26	5	31

↳	deck	76.7 %	626	190	816
↳	Deck.java	80.2 %	333	82	415
↳	PropertyPayCard.java	22.0 %	11	39	50
↳	MoveToFleetCard.java	60.6 %	40	26	66
↳	GiftCard.java	80.6 %	58	14	72
↳	PayCard.java	72.4 %	21	8	29
↳	RecieveCard.java	50.0 %	7	7	14
↳	Card.java	50.0 %	6	6	12
↳	MoveAmmountCard.java	87.9 %	29	4	33
↳	QueensCard.java	50.0 %	4	4	8
↳	JailCard.java	100.0 %	16	0	16
↳	MoveCard.java	100.0 %	7	0	7
↳	MoveToCard.java	100.0 %	20	0	20
↳	RecievelfCard.java	100.0 %	74	0	74
↳	run	0.0 %	0	22	22
↳	Launch_Showcase.java	0.0 %	0	11	11
↳	Launch.java	0.0 %	0	11	11
↳	test_JUnit	99.8 %	1,767	3	1,770
↳	AllTests.java	0.0 %	0	3	3
↳	DeckTest.java	100.0 %	853	0	853
↳	HouseTest.java	100.0 %	213	0	213
↳	JailTest.java	100.0 %	530	0	530
↳	MoveToJailTest.java	100.0 %	29	0	29
↳	PastStartTest.java	100.0 %	47	0	47
↳	TestPurchase.java	100.0 %	95	0	95

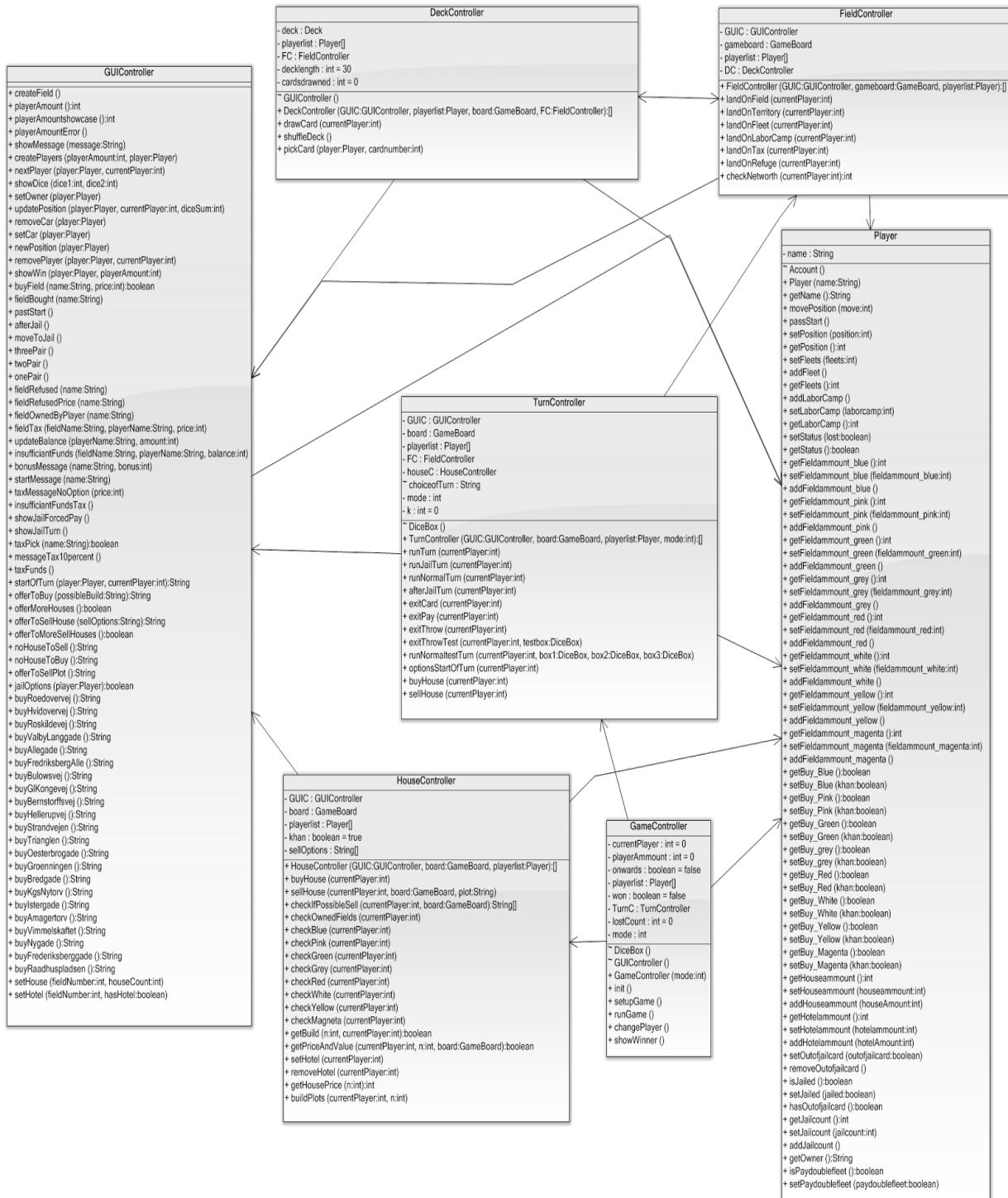
Vi har altså en samlet codecoverage på 38,7%. Vi kan se at der er testet mindst i vores controllere (control package, samt boundary package). Dette skyldes, at vi primært har benyttet blackbox testing for at teste spillogikken i controllerne.

Vores whitebox testing er altså primært testing på vores entiteter.

Designklassediagram

Del 1 - Associationer mellem Controllers.

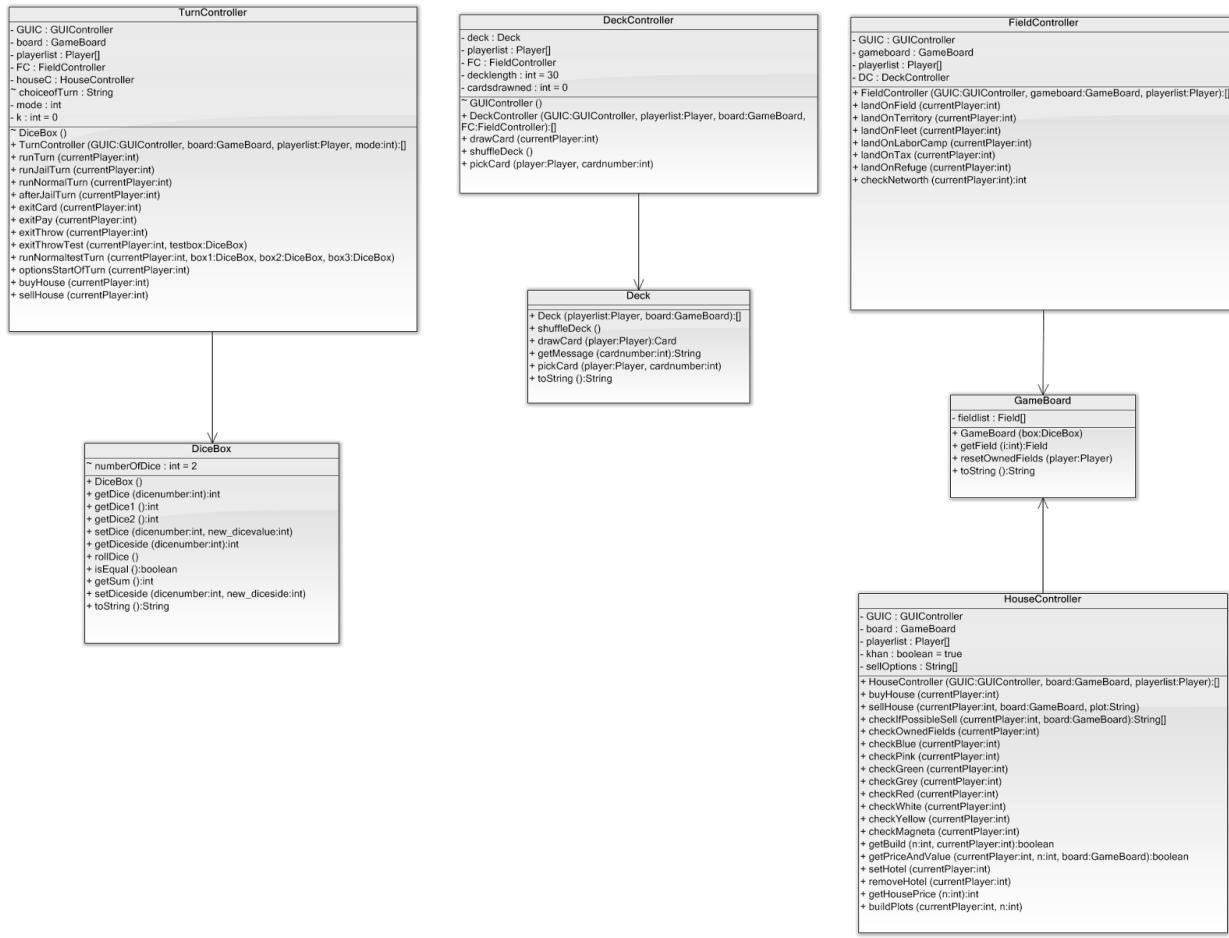
På billedet nedenfor er vist en del af Designklassediagrammet, dog er dette kun for vores Controllers, samt spiller entiteten. På billedet kan relationerne ses.



På billedet ovenfor kan vi se hvordan Controllernes relationer er imellem hinanden, vi ser altså at alle controllerne bortset fra GameController benytter sig af GUIControlleren, nemlig idet at vi igennem GUIControlleren får vist beskeder på GUI'en. Og grunden til at vi nemlig ikke har en association fra GameControlleren er idet at beskederen går igennem HouseControlleren og TurnControlleren. Ellers er stortset alle de andre klasser også forbundet til Player klassen.

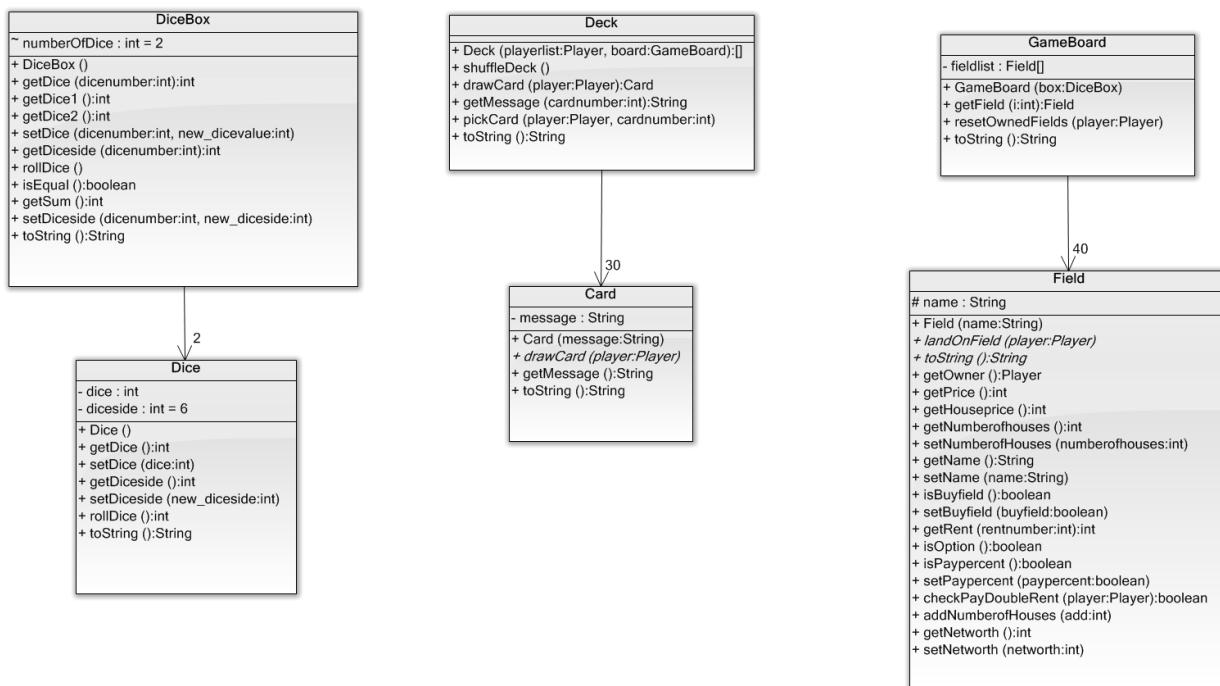
Del 2 - Controller til Entitet

Nedenfor er vist Controllernes relationer til vores entiteter som DiceBox, Deck og Gameboard. Altså kan man her få en fornemmelse af hvornår det rent faktisk er vi bruger controllerne i sammenhæng med de entiteter vi har, og hvordan de altså hænger sammen. På billedet nedenfor kan man se at TurnController bruger DiceBox idet at TurnController står for bevægelse på GUI'en samt generelt for hvordan spilleren bevæger sig og hvad der derefter skal ske. DeckControlleren står for at have styr på vores Deck, nemlig at have styr på hvornår et chancekort trækkes og hvad der derefter skal ske. FieldController bruger GameBoard, men det gør HouseController også. FieldControllerne står for risikoen for det givende felt hvorimod HouseControlleren står for placering af huse på felterne, HouseControlleren var også forbundet til GUI'en som set på forrige billede, altså har vi her at gøre med hvornår der skal placeres et hus på GUI'en, men der bliver her også sørget for at spilleren får den tekniske bonus idet at en modstander lander på feltet med det givende hus på.



Del 3 - Entiteter til nedarvinger.

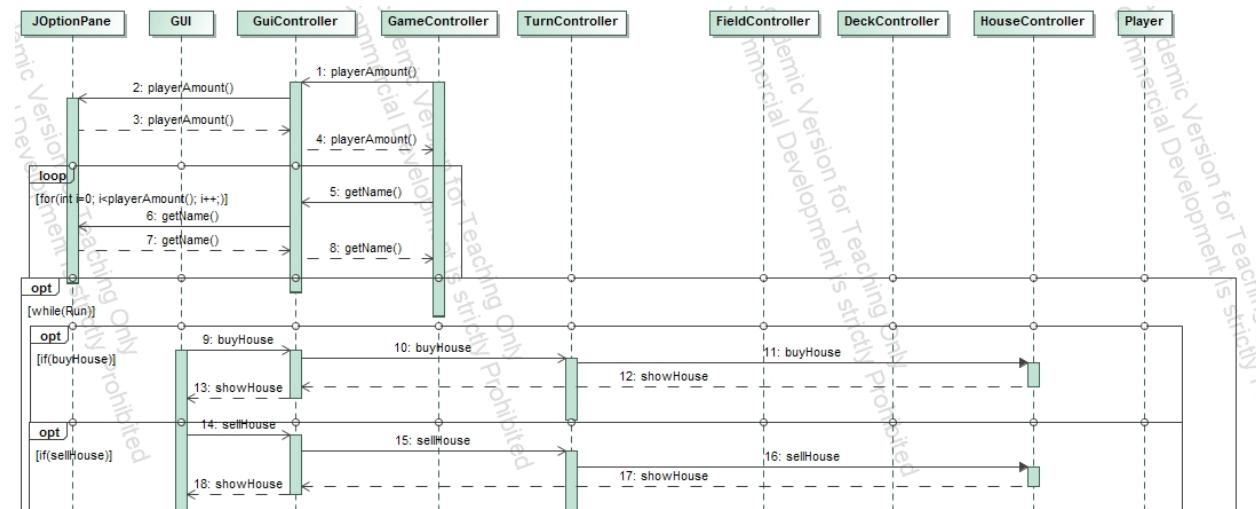
På billedet nedenfor, kan man se hvordan vi til sidst har nedarvet DiceBox, Deck og GameBoard, som Domænemodellen, har vi også her at DiceBox består af 2 Dice, altså må Dice være nedarvet fra DiceBox, herefter har vi Deck som består af 30 kort, altså må Card således også være nedarvet fra Deck og til sidst har vi så GameBoard som består af 40 felter. Yderligere består Card så af alle de kort typer som man kan trække, disse bliver derfor nedarvet i forhold til Card klassen, hvor Field er lidt i samme dur, nemlig at vi har forskellige felt typer som alle også bliver nedarvet til Field, hvor nogle dog først går igennem klassen Ownable, altså om felterne kan ejes.



Design Sekvensdiagram

Design Sekvensdiagrammet er delt op i 3 dele for overskuelighedens skyld, men er sammenhængende, så del 2 er fortsættelsen af del 1 osv.

Del 1



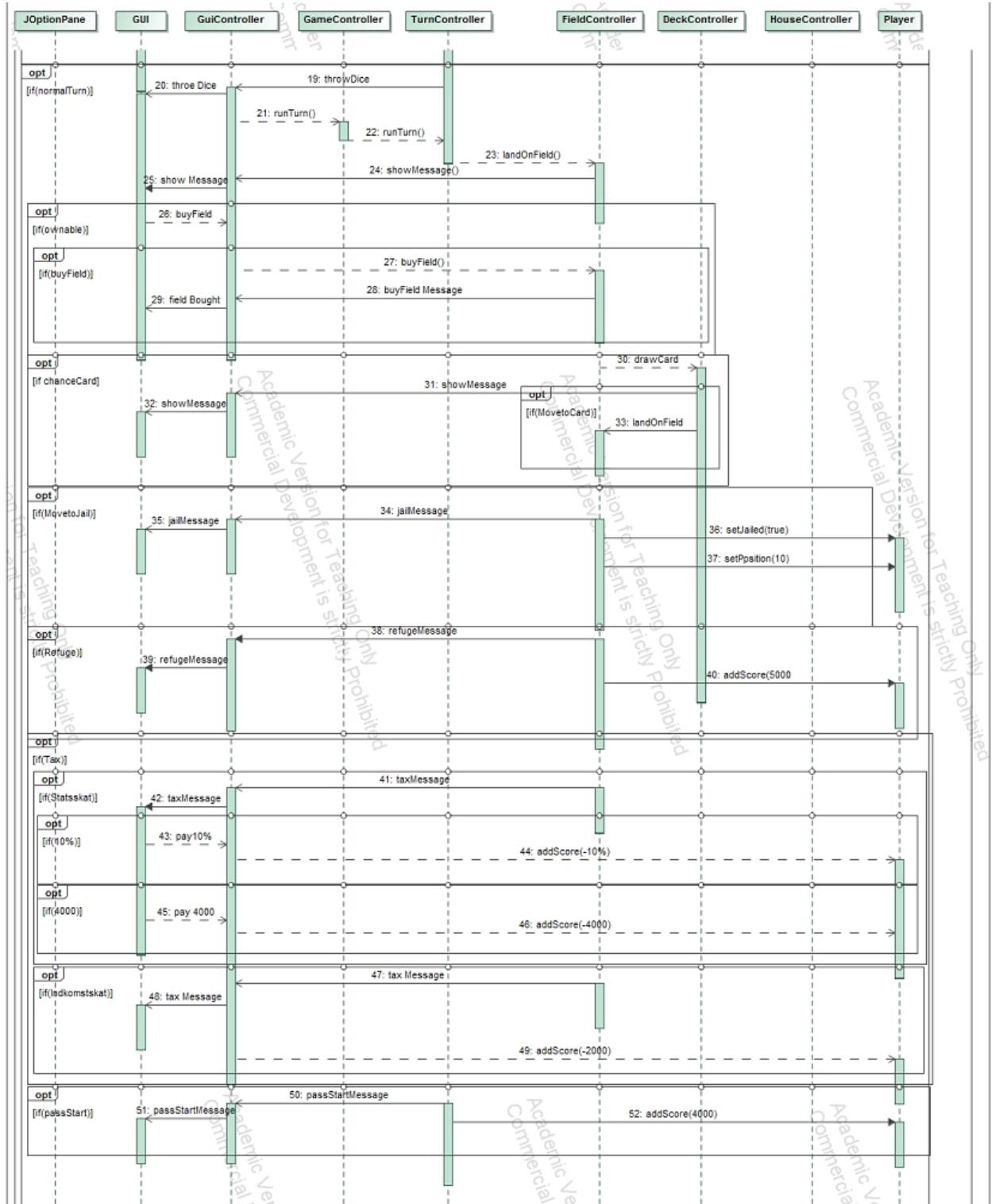
Diagrammet indeholder user interfaces, alle controllere, samt klassen player. De resterende klasser er udeladt.

Beskrivelse:

Ved opstart bedes brugerne at indtaste antal spillere i JOptionPane. En for - løkke startes hvori alle navne indtastes. Herefter starter en while-løkke som først afsluttes når en vinder er fundet. Fra GUI'en starter 3 if-sætninger; køb hus , sælg hus, eller rul terninger. Vælger spilleren "køb hus", sendes valget fra GUI'en til GuiControlleren videre til TurnControlleren og igen videre til HouseControlleren. Fra HouseControlleren afgøres det om hvilke huse, hvis nogen overhovedet, spilleren kan købe. Svaret sendes via GUIControlleren til GUI'en. Hvis spilleren vælger "sælg hus", starter den helt samme process som beskrevet i "køb hus".

Vælger spiller den tredje mulighed "rul terninger" starter en normal tur. Processerne i NormalTurn er vist i del 2.

Del 2

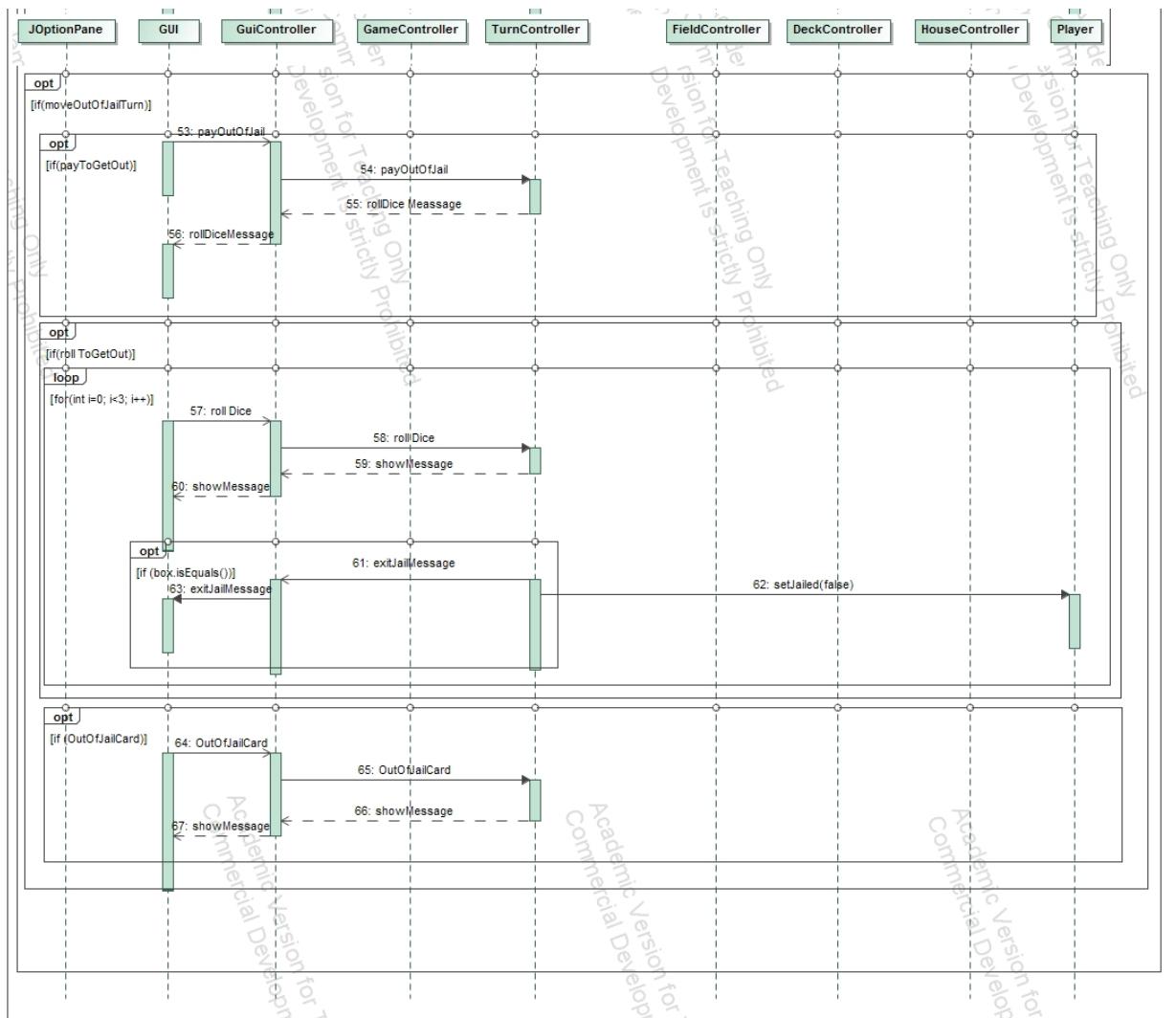


Først kaldes throwDice i TurnControlleren, værdien vises i GUI'en via GuiControlleren. Fra GuiControlleren sendes runTurn() der sendes til GameControlleren og videre til

TurnControlleren, herfra kaldes landOnField() metoden der sendes til FieldControlleren. I FieldControlleren afgøres det hvilket type af felt spilleren er landet på, og svaret sendes til GUI'en via GUIControlleren. I de resterende if-sætninger vises processerne for de forskellige typer af felter spilleren kan lande på. Der er grunde (Ownables), chancekort (ChanceCard), ryk i fængsel (MoveToJail), parkering (Refuge), og to typer af skat, hhv. statsskat og indkomstskat, og til sidst startfeltet.

I del 3 vises den særlige tur MoveOutOfJail, der kun forekommer når en spiller er i fængsel. Spilleren spørges fra GUI'en om enten at betale sig ud (payToGetOut), eller at slå for at komme ud (rollToGetOut). Den sidste mulighed er at spilleren har et benådningskort (MoveOutOfJailCard).

Del 3



Diskussion

Vores vigtigste udgangspunkt for programmet var at få implementationerne til at fungere ordentligt. Altså få udviklet et program uden nogle fatale bugs eller fatale crashes ved enkelte funktioner - et såkaldt 'stable build'. For at holde deadline og samtidig få opfyldt dette mål, har vi undladt nogle implementationer:

- Pantsætning af grunde
- Salg af grunde
- Internt handel mellem spillere

Derudover er der nogle features vi også kunne have tænkt os at implementere i de spilfunktioner, der er med i programmet:

En dropdown menu over de grunde der kan købes huse på. - (I vores release build, har vi en ja/nej user selection for hver grund der kan købes huse på. Dette er dog implementeret for salg af huse)

Vores fokus på at få lavet et stable build, har gjort, at vi har haft en relativt tidlig code freeze, hvorefter vi lagde fokus på bug fixing. Dette har været med til at sikre at vores udleverede program er næsten bug fri. (ingen rapporterede bugs indtil videre). Vi har gennemtestet vores build, og kan umiddelbart sige at spillet er bug frit, selvom man ikke kan være helt sikker på dette.

Programmet er dog designet, så implementationerne af de undladte spilfunktioner, ikke ville være besværlige at implementere. Men de er undladt, for at undgå at introducere nye bugs.

Forskellige deadlines:

Code freeze blev afholdt Onsdag den 14/01-2015 ved midnat.

Bug freeze blev afholdt Lørdag den 17/01-2015 ved midnat.

Dokumentations freeze blev afholdt Søndag den 18/01-2015 ved midnat.

Dokumentation foregik løbende, siden vi startede med designfasen.

I vores program har vi overholdt GRASP, ved ikke at lade vores entiteter kende til controllere eller boundaries. Vi har overholdt lav coupling ved at lade klasser som Deck og GameBoard være creators for de felter der skal bruges til spillet. Vi har overholdt high cohesion ved at sørge for at vores klasser kun kender til information, der er relevant for klassen selv.

Konklusion

Vi har lavet et matadorspil, der kan spilles af 3-6 spillere. Vi har implementeret køb af grunde, samt huse og hoteller. Felterne "prøv lykken", "fængsel", "gå i fængsel", "statsskat", "indkomstskat", og "start" virker efter hensigten. Vi har prioriteret at de implementerede funktioner virker stabilt, frem for at implementere alle funktioner. Derfor har vi ikke implementeret funktionerne pantsætning af grunde, salg af grunde, eller intern handel mellem spillerne.

Vi har stræbt efter at bygge koden op så den overholder GRASP. Vi har derfor lavet 6 controllere der styrer hver deres område, hhv.: GUIController, GameController, FieldController, DeckController, HouseController, TurnController.

Bilag

Feltliste

NR.	NAVN	Pris: (Grund, hus, pantsæt)	leje	1 HUS	2 HUSE	3 HUSE	4 HUS E	HOTEL
1	Start	indkassere 4000 når du passerer.						
2	Rødovrevej	1200, 1000, 600	40	200	600	1800	3200	5000
3	Prøv Lykken							
4	Hvidovrevej	1200, 1000, 600	80	400	1200	3600	6400	9000
5	Indkomstskat	betal 10% eller 4000kr.						
6	Helsingør-He lsingborg	4000, - , 2000	500, 1000, 2000, 4000					
7	Roskildevej	2000, 1000, 1000	120	600	1800	5400	8000	11000

8	Prøv Lykken							
9	Valby Langgade	2000, 1000, 1000	120	600	1800	5400	8000	11000
10	Allégade	2400, 1000, 1200	160	800	2000	6000	9000	12000
11	Fængsel							
12	Frederiksber g Allé	2800, 2000, 1400	200	1000	3000	9000	12500	15000
13	Tuborg	3000, - , 1500	80xsum, 200xsum					
14	Bülowsvej	2800, 2000, 1400	200	1000	3000	9000	12500	15000
15	Gl Kongevej	3200, 2000, 1600	240	1200	3600	10000	14000	18000
16	Mols-Linien	4000, - , 2000	500, 1000, 2000, 4000					

17	Bernstorffsvej	3600, 2000, 1800	280	1400	4000	11000	15000	19000
18	Prøv Lykken							
19	Hellerupvej	3600, 2000, 1800	280	1400	4000	11000	15000	19000
20	Strandvejen	4000, 2000, 2000	320	1600	4400	12000	16000	20000
21	Parkering	Modtag de penge der er blevet betalt i skat						
22	Trianglen	4400, 3000, 2200	360	1800	5000	14000	17500	21000
23	Prøv Lykken							
24	Østerbrogade	4400, 3000, 2200	360	1800	5000	14000	17500	21000
25	Grønningen	4800, 3000, 2400	400	2000	6000	15000	18500	22000

26	Gedser-Rostock	4000, - , 2000	500, 1000, 2000, 4000					
27	Bredgade	5200,3000,26 00	440	2200	6600	16000	19500	23000
28	Kgs Nytorv	5200,3000,26 00	440	2200	6600	16000	19500	23000
29	Coca-Cola	3000, - , 1500	80xsum, 200xsum					
30	Østergade	5600,3000,28 00	480	2400	7200	17000	20500	24000
31	De Fængsles	Ryk til fængsels felt.						
32	Amagertorv	6000,4000,30 00	520	2600	7800	18000	22000	25500
33	Vimmelskaft et	6000,4000,30 00	520	2600	7800	18000	22000	25500
34	Prøv Lykken							

35	Nygade	6400,4000,32 00	560	3000	9000	20000	24000	28000
36	Rødby-Puttg arden	4000, - , 2000	500, 1000, 2000, 4000					
37	Prøv Lykken							
38	Frederiksber ggade	7000,4000,35 00	700	3500	10000	22000	26000	30000
39	Statsskat	betal 2000 i skat.						
40	Rådhusplads en	8000,4000,40 00	1000	4000	12000	28000	34000	40000

Chancekort liste

nr.	Type	besked
1	MoveToCard	Ryk frem til Grønningen. Hvis de passerer "Start", indkassér da kr. 4000.
2	MoveToFleet Card	Ryk brikken frem til det nærmeste rederi og betal ejeren to gange den leje, han ellers er berettiget til. Hvis selskabet ikke ejes af nogen kan De købe det af banken.
3	MoveToCard	Ryk frem til "Start".
4	MoveAmmoun tCard	Ryk tre felter tilbage.
5	MoveToCard	Ryk frem til Frederiksberg Allé. Hvis de passerer "Start", indkassér da kr. 4000.
6	MoveToCard	Teg med Mols-Linien -- Flyt brikken frem og hvis de passerer "Start", indkassér da kr. 4000.
7	MoveToCard	Tag ind på Rådhuspladsen.
8	JailCard	Gå i fængsel. Ryk direkte til fængsel. Selv om De passerer "Start", indkasserer de ikke kr. 4000.
9	JailCard	Gå i fængsel. Ryk direkte til fængsel. Selv om De passerer "Start", indkasserer de ikke kr. 4000.
10	PayCard	Betal kr. 3000 for reparation af Deres vogn.
11	PropertyPay Card	Oliepriserne er steget, og de skal betale: kr.500 pr. hus, kr.2000 pr. hotel
12	PayCard	De har måtte vedtage en parkeringsbøde. Betal kr. 200 i bøde.
13	PayCard	Betal Deres bilforsikring kr. 1000.
14	PayCard	Betal kr. 3000 for reparation af deres vogn.
15	PayCard	De har været en tur i udlandet og haft for mange cigaretter med hjem. Betal told kr. 200.
16	PayCard	De har kørt frem for "Fuld Stop". Betal kr. 1000 i bøde.

17	PayCard	De har modtaget Deres tandlægeregning. Betal kr. 2000.
18	PropertyPay Card	Ejendomsskatterne er steget, ekstra udgifterne er: kr. 800 pr. hus, kr. 2300 pr. hotel
19	RecieveCard	Værdien af egenavl fra nyttehaven udgør kr. 200, som de modtager af banken.
20	GiftCard	Det er Deres fødselsdag. Modtag af hver medspiller kr. 200.
21	RecieveCard	De har vundet i Kasselotteriet. Modtag kr. 500.
22	RecieveCard	De havde en række med elleve rigtige i tipning. Modtag kr. 1000.
23	RecieveCard	Modtag udbytte af Deres aktier kr. 1000.
24	RecieveCard	Grundet dyrtiden har De fået gageforhøjelse. Modtag kr. 1000.
25	RecieveCard	Kommunen har eftergivet et kvartals skat. hæv i banken kr. 3000.
26	RecieveCard	Modtag udbytte af Deres aktier kr. 1000.
27	RecieveCard	Deres præmieobligation er kommet ud. de modtager kr. 1000 af banken.
28	RecieveCard	De modtager Deres aktieudbytte. Modtag kr. 1000 af banken.
29	RecievelfCard	De modtager "Matador-legatet for de værdige trængende", stort kr. 40000. Ved værdig trængende forstås, af Deres formue, d.v.s. Deres kontante penge + skøder + bygninger ikke overskridet kr. 15000.
30	QueensCard	I anledning af deres majestæts fødselsdag benådes de herved for fængsel. Dette kort kan opbevares, indtil de får brug for det, eller de kan sælge det.

Kildekode

*Kildekoden ligger i en separat fil kaldet 53_Final_Sourcecode.pdf (se eclipse projektet)

Launch

Launch	-- side 1
Launch_Showcase	-- side 2

Controllers

GUI Controller	-- side 3
Game Controller	-- side 11
Turn Controller	-- side 13
House Controller	-- side 18
Field Controller	-- side 29
Deck Controller	-- side 35

Entities

entity package

Account	-- side 36
Player	-- side 37
Dice	-- side 43
DiceBox	-- side 44

fields package

Field	-- side 46
GameBoard	-- side 48
Ownable	-- side 50
Fleet	-- side 51
LaborCamp	-- side 53
Territory	-- side 55
Jail	-- side 58
Refuge	-- side 59
Start	-- side 60
Tax	-- side 61
ChanceCard	-- side 66
MoveToJail	-- side 73

deck package

Card	-- side 63
Deck	-- side 64
GiftCard	-- side 67
JailCard	-- side 68
MoveAmmountCard	-- side 69
MoveCard	-- side 70
MoveToCard	-- side 71
MoveToFleetCard	-- side 72
PayCard	-- side 74
PropertyPayCard	-- side 75
QueensCard	-- side 76
RecieveCard	-- side 77
RecievelfCard	-- side 78

Test

AllTests	-- side 79
Deck Test	-- side 80
House Test	-- side 86
Jail Test	-- side 88
MoveToJailTest	-- side 92
Pass start Test	-- side 93
Test Purchase	-- side 94