

FINAL - Matador

Group 15 + 18

Deadline: 21th of January 2019

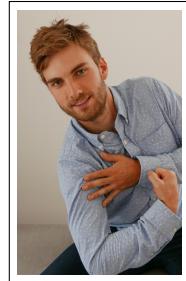
This report contains 45 pages excluding front page and appendix



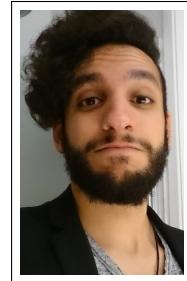
Rasmus Sander Larsen
s185097



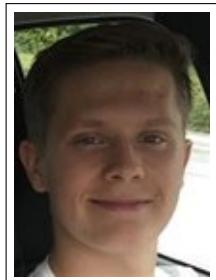
Søren Poulsen
s180905



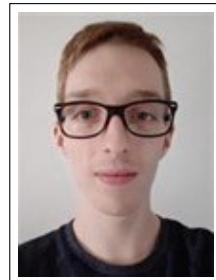
Alfred Röttger Rydahl
s160107



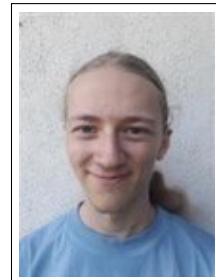
Noah F.M. Hamza
s185084



Nikolaj Tscharn Wassmann
s185082



Anders la Cour Lønborg
s185100



Johannes Verner Kornø Piil
s185114

1 Timeregnskab

| Opgave/Navn | Alfred | Søren | Rasmus | Noah | Johannes | Anders | Nikolaj |
|----------------------------|--------|-------|--------|------|----------|--------|---------|
| Design | 10 | 7 | 2 | 5 | 6 | 10 | 5 |
| - Navneord | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| - Designklasse | 0 | 0 | 0 | 0 | 1 | 12 | 0 |
| - System sekvens | 0 | 0 | 0 | 4 | 0 | 2 | 1 |
| - Sekvens | 0 | 0 | 0 | 0 | 7 | 0 | 0 |
| Implementering | | | | | | | |
| General opsætning | 4,5 | 3 | 10 | 0 | 2 | 3 | 2 |
| Bugfixing | 18 | 0 | 14 | 0 | 0 | 7 | 0 |
| model | | | | | | | |
| - Reader Package | 3 | 0 | 3 | 0 | 0 | 0 | 1 |
| - Cup package | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| - Player package | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| - Board package | 1 | 0 | 7 | 0 | 0 | 0 | 4 |
| - Chancecard package | 0 | 0 | 9 | 0 | 0 | 0 | 0 |
| Controller | | | | | | | |
| - FieldAction Package | 7 | 0 | 5 | 0 | 0 | 4 | 6 |
| - ChanceCardAction Package | 1 | 0 | 7 | 0 | 0 | 0 | 0 |
| - ExtraAction Package | 0 | 0 | 6 | 0 | 0 | 0 | 0 |
| - SetupController | 8 | 0 | 3 | 0 | 0 | 6 | 1 |
| - MainController | 2 | 0 | 8 | 0 | 0 | 0 | 0 |
| - GuiController | 2 | 0 | 2 | 0 | 0 | 0 | 2 |
| - TurnController | 6 | 0 | 7 | 0 | 0 | 0 | 5 |
| - FieldController | 4 | 0 | 7 | 0 | 0 | 0 | 3 |
| View | 1 | 0 | 3 | 0 | 0 | 0 | 1 |
| Test | 2 | 6 | 3 | 0 | 6 | 0 | 2 |
| - Die | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| - Cup | 0 | 0,5 | 0 | 0 | 0 | 0 | 0 |
| - Player | 0 | 2 | 0 | 0 | 0 | 0 | 0 |
| - Account | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| - Chancekort | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| - Game | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| - Board | 0 | 2 | 0 | 0 | 0 | 0 | 0 |
| Rapport | 0 | 17 | 4 | 35 | 20 | 17 | 16 |
| - Kravspecifikation | 0 | 3 | 0 | 0 | 0 | 0 | 0 |
| - Analyse | 0 | 3 | 0 | 5 | 3 | 4 | 2 |
| - Verifikation | 0 | 1 | 0 | 0 | 0 | 0 | 3 |
| - Implementering | 0 | 0 | 0 | 5 | 7 | 0 | 4 |
| - Testcase | 0 | 4 | 0 | 5 | 4 | 0 | 1 |
| Total | 71,5 | 51,5 | 102 | 59 | 56 | 65 | 60 |
| Nettoforbrug | 465 | timer | | | | | |

Indhold

| | |
|---|-----------|
| 1 Timeregnskab | |
| 2 Indledning | 2 |
| 3 Kravspecifikation | 2 |
| 3.1 "Must have"krav | 3 |
| 3.2 "Should have"krav | 3 |
| 3.3 "Could have"krav | 4 |
| 3.4 "Wont have"krav | 4 |
| 4 Analyse | 5 |
| 4.1 Use case | 5 |
| 4.2 Use case beskrivelser og Sub Use case | 6 |
| 4.3 Fully dressed | 7 |
| 4.4 Domænemodel | 7 |
| 4.5 Systemsekvensdiagram | 9 |
| 5 Design | 10 |
| 5.1 Anvendte GRASP mønstre | 10 |
| 5.2 Design klassediagram | 10 |
| 5.3 Sekvensdiagram | 15 |
| 6 Implementering | 17 |
| 6.1 Model package | 17 |
| 6.1.1 Board package | 17 |
| 6.1.2 ChanceCard package | 19 |
| 6.1.3 Cup package | 19 |
| 6.1.4 Reader package | 20 |
| 6.1.5 Player package | 20 |
| 6.2 Controller package | 20 |
| 6.2.1 ChanceCardManagement | 20 |
| 6.2.2 ExtraActionManagement | 21 |
| 6.2.3 FieldManagement | 22 |
| 6.3 View package | 24 |
| 7 Test | 25 |
| 7.1 JUnit test | 26 |
| 7.1.1 Terningen | 26 |
| 7.2 Testcase | 28 |
| 7.3 Bruger test | 30 |
| 7.3.1 TC01 - Instruktion | 30 |
| 7.3.2 TC01 - Testforløb | 30 |
| 7.3.3 TC03 - Instruktion | 30 |
| 7.3.4 TC03 - Testforløb | 30 |

| | |
|---------------------------------------|-----------|
| 8 Dokumentation | 31 |
| 8.1 Arv | 31 |
| 8.2 Abstract | 32 |
| 8.3 ArrayList | 32 |
| 8.4 Enum | 33 |
| 8.5 HashMap | 33 |
| 8.6 Collections | 34 |
| 8.7 Import af projekt - GIT | 35 |
| 8.8 Tidsplan | 35 |
| 8.9 Krav gennemgang | 37 |
| 8.9.1 Must have | 37 |
| 8.9.2 Should have | 37 |
| 8.9.3 Could have | 37 |
| 9 Projektforløb | 37 |
| 10 Konklusion | 38 |
| 10.1 Perspektivering | 38 |
| 11 Bilag | 39 |
| 11.1 Github repo | 39 |
| 11.2 Kildekode oversigt | 39 |
| 11.3 Chancekort | 41 |
| 11.4 Designklassediagram | 43 |
| 11.5 Use Case diagram | 44 |
| 11.6 SystemSekvensDiagramm | 45 |

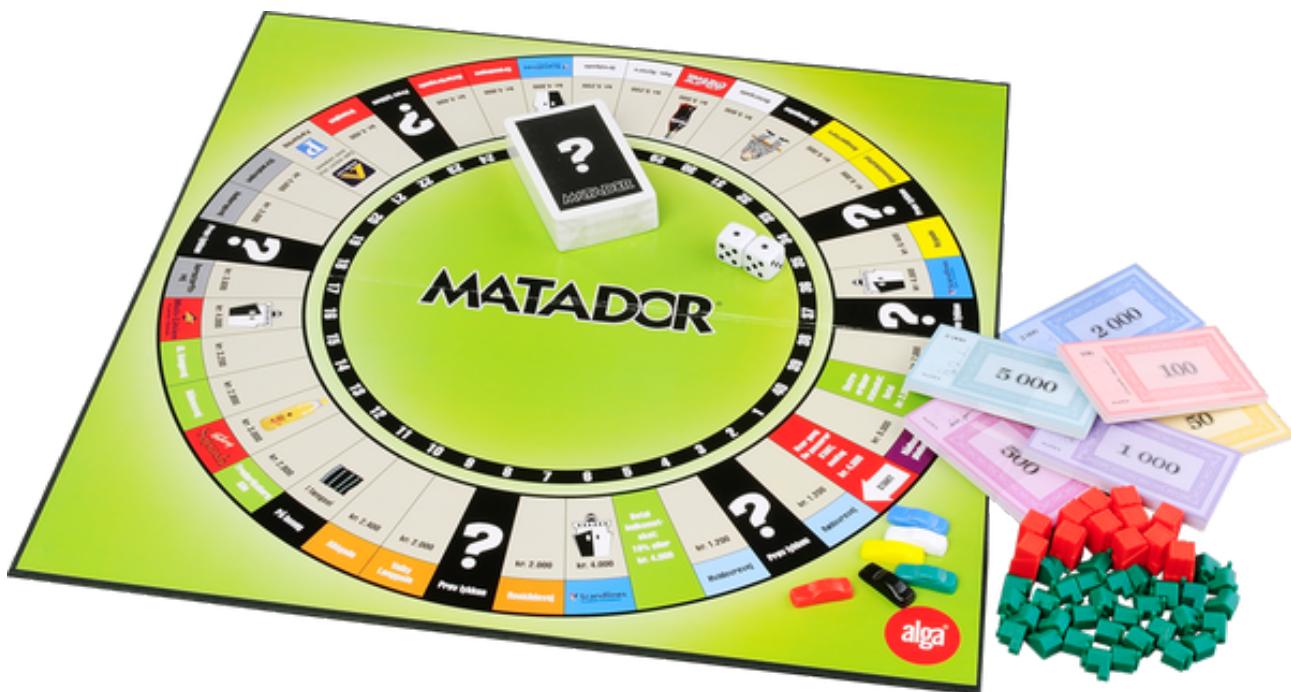
2 Indledning

Rapporten er udarbejdet i et samarbejde mellem gruppe 15 + 18 i 3-ugers forløbet til kurset "Indledende programmering - 02312".

Spillet tager udgangspunkt i det fysiske Matador spil. I Matador er der mellem 3-6 spillere, hvor de skiftevis slår to terninger og rykker rundt på en spilleplade. Spillepladen indeholder 40 felter - med 8 forskellige typer, der hver har eget navn og effekt.

Spillerne starter på 'Start fletet' og rykker deres brikker i urets retning. Hvis man lander på et ejendomsfelt, som ikke i forvejen er ejet, så har man mulighed for at købe det. Hvis man fravælger at købe grunden, så kommer den på auktion mellem de andre spillere. Lander man på et felt, som er ejet af en anden spiller, skal man betale dets husleje. Dette kan variere alt efter om en person ejer alle grunde af samme type, eller om man har bygget huse eller et hotel på grunden.

Hvis en spiller ikke har råd til at betale huslejen eller en anden udgift, kan man vælge at pantsætte en af sine grunde for den halve købssum. Har man ingen grunde og derfor ikke nok penge til at betale en udgift, så er man ude af spillet. Resten af ens penge bliver givet til den person, som man skulle betale huslejen hos. Spillet fortsætter således indtil alle er gået fallit uddover en spiller - og denne spiller er så vinderen af Matador.



Figur 1: Matador Spil

3 Kravspecifikation

Kravsspecifikationen fremgår i dette afsnit. De er blevet opdelt i grupper baseret på hvor vigtige kravene er for at have et fungerende Matador spil.

3.1 "Must have"krav

- (M₁) Spil mellem 3-6.
- (M₂) Spillerne skiftes til at slå med 2 terninger (6 sider).
- (M₃) Der skal være 40 felter.
- (M₄) Spillerne starter fra felt 1, "Start feltet".
- (M₅) Spillerne skal kunne lande på et felt og flytte videre der fra.
- (M₆) Spillerne bevæger sig altid til venstre (frem ad).
- (M₇) Alle spillere starter på 30.000 kr.
- (M₈) En spiller går fallit når vedkommende ikke kan betale en udgift.
 - Når en spiller går fallit, er spilleren ude af spillet.
- (M₉) Man spiller indtil der er kun én spiller, som ikke er gået fallit.

3.2 "Should have"krav

- (S₁) Når der slås 2 ens, får man en ekstra tur.
- (S₂) Hvis man slår to ens, tre gange i træk sendes man i fængsel og modtager ikke 4.000 kr. ved passering af start.
- (S₃) Der skal være 28 felter med ejendomme.
 - 22 felter er tildelt en farve gruppe, 2 af grupperne er på 2 felter, og 6 af grupperne er 3 felter.
 - 4 felter er færger.
 - 2 felter er bryggerierne
- (S₄) 12 special felter.
 - Der skal være 6 chance felter.
 - Der skal være 2 bøde felter
 - Der skal være 1 "Start"felt.
 - Der skal være 1 "På besøg i fængsel"felt.
 - Der skal være 1 "Gratis Parkering"felt.
 - Der skal være 1 "Gå i fængsel"felt.
- (S₅) Spillerne får 4.000 kr. hver gang de krydser start.
- (S₆) Spillerne har mulighed for at handle med grunde mellem hinanden.
- (S₇) Chance kort skal blandes mellem hvert spil.
 - Chancekortene skal ligge i en bunke og komme i den rækkefølge de ligger i, efter de er bandet ved spillets start.

- Kortet der rækkes skal efterfølgende lægges nederst i bunken med chancekort.

(S₈) Når man ejer alle grunde af en farve, får man dobbelt i standard huslejen.

- Det er muligt at købe huse og hoteller til grunden hvis man ejer alle i en farve.

3.3 "Could have"krav

(C₁) Spillerne kan komme ud af fængslet ved at betale 1.000 kr. eller slå 2 ens.

- En spiller i fængslet har 3 forsøg til at slå 2 ens.
- Evt. slag af 2 ens, gælder som spillerens ryk, startende fra felt 11 "I fængsel".

(C₂) Spillerne skal kunne pantsætte sine grunde.

(C₃) Spillerne skal kunne sælge deres huse og hoteller.

(C₄) Hvis en spiller ikke køber en grund, kommer den til auktion mellem de andre spillere.

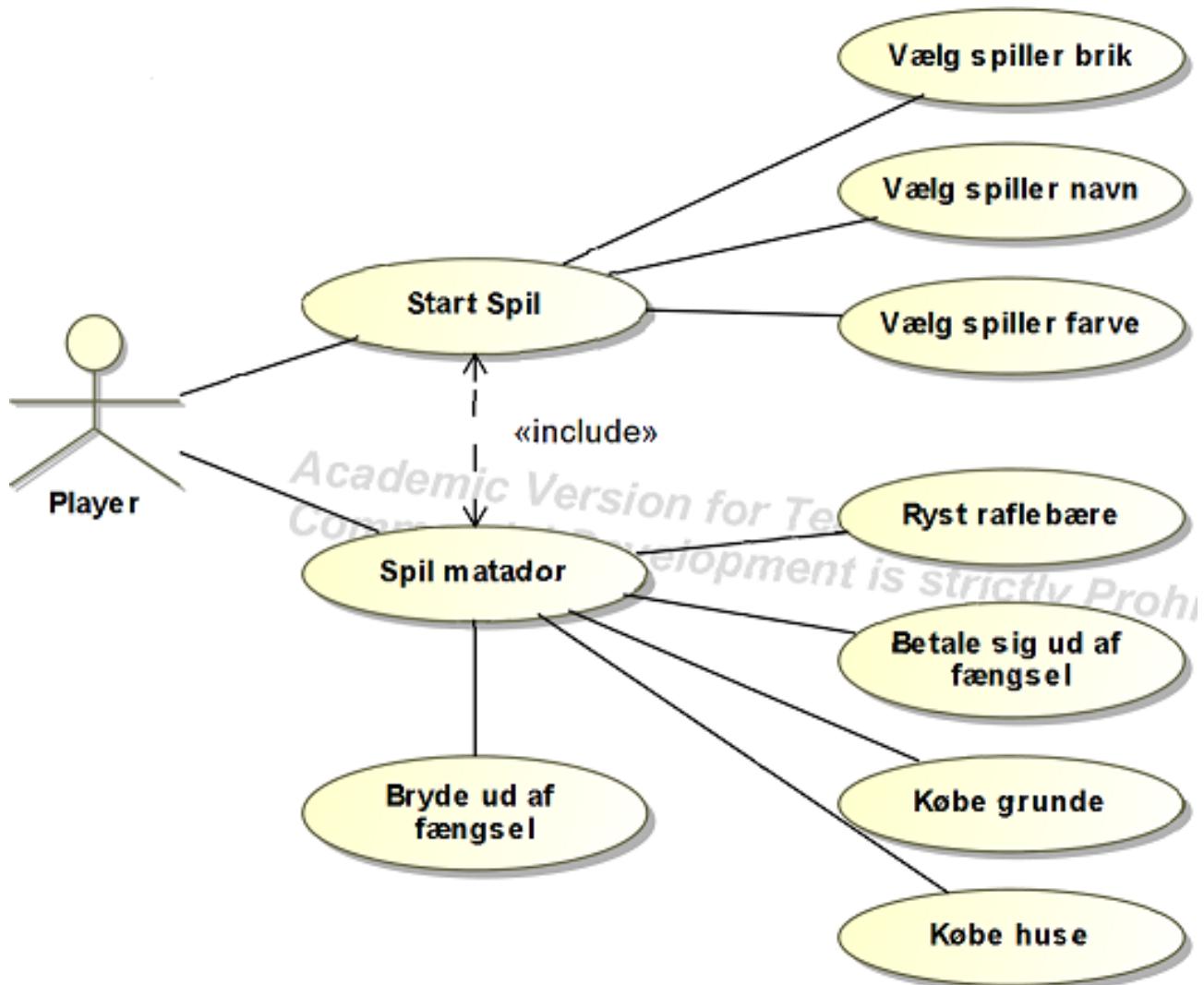
3.4 "Wont have"krav

(W₁) Maks antal huse og hoteller i spillet.

(W₂) Maks antal penge der er i spillet.

4 Analyse

4.1 Use case



Figur 2: Use Case Diagram med SubUseCases

Ovenfor ses et use case diagram, hvor den primære aktør er en Spiller. Dette diagram viser de vigtige elementer i systemet, dvs. "Start Spil" og "Spil Matador". Det ses også, at der er en flere sub-use cases tilhørende de to store use cases. Alle disse use cases bliver beskrevet nærmere nedenfor.

4.2 Use case beskrivelser og Sub Use case

| Use case | Beskrivelse |
|---|--|
| <u>1: Start Spil</u> - 1.1: Vælg spiller navn - 1.2: Vælg spiller farve | Spilleren skal kunne starte spillet Spilleren indtaster et navn Spilleren vælger farven på deres spillerbrik |
| <u>2: Spil Matador</u> - 2.1: Ryst raflebægeret - 2.2: Købe grunde - 2.3: Købe huse - 2.4: Betale sig ud af fængslet - 2.5: Bryde ud af fængslet | Spilleren skal kunne spille spillet Spilleren bruger raflebægeret, der indeholder to terninger, og får en talværdi Spilleren skal kunne købe en grund Spilleren skal kunne placere huse på sine grunde Spilleren skal kunne betale sig ud af fængslet Spilleren skal kunne slå sig ud af fængslet |

4.3 Fully dressed

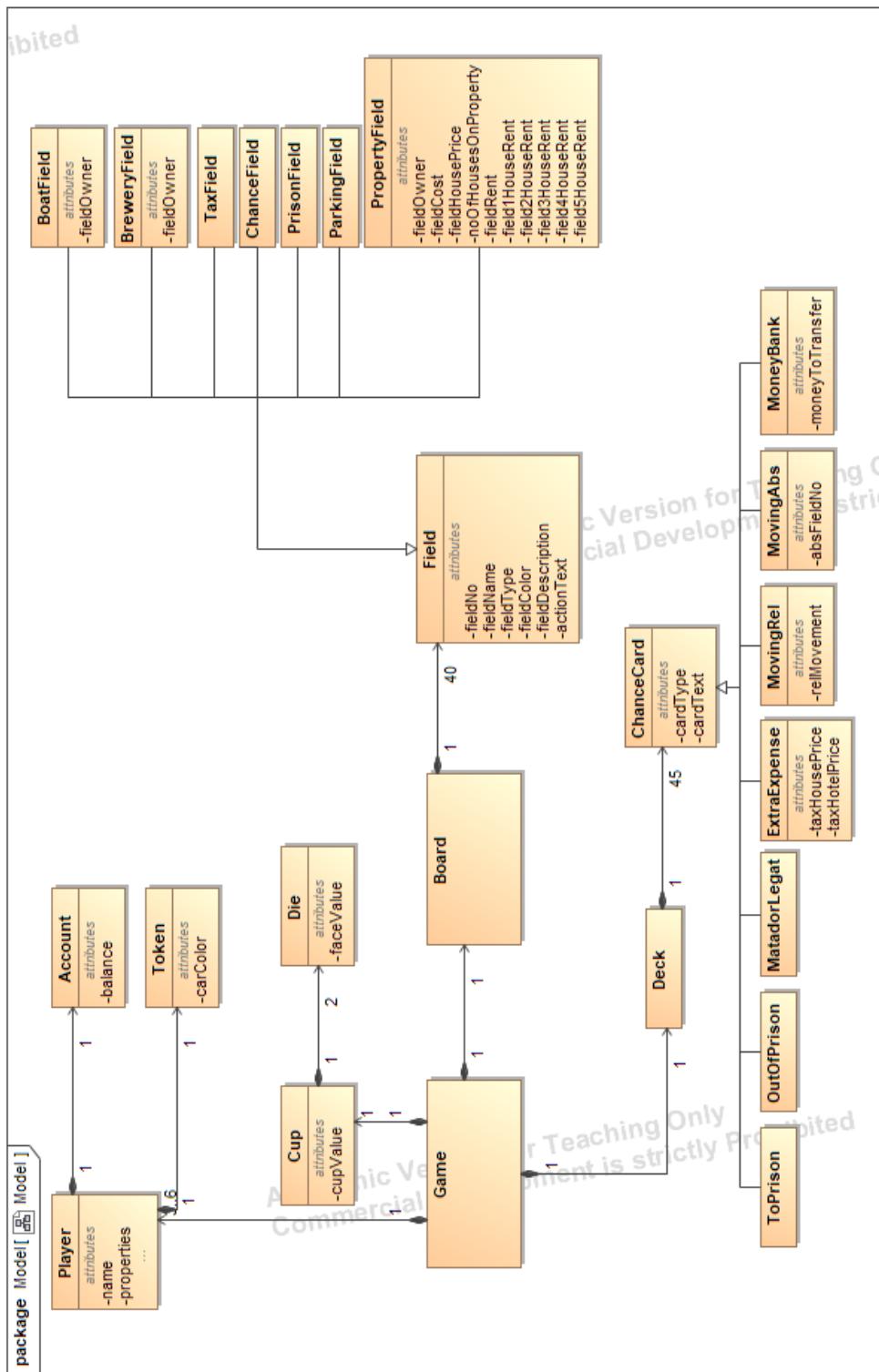
| Use case sektion | Beskrivelse |
|----------------------------|---|
| Navn | "Spil Matador" |
| Scope | Matador spil |
| Level | User goal |
| Primær aktør | Spilleren |
| Andre interesserter | IOOuterActive, Kunden |
| Preconditions | Spillet er startet |
| Postconditions | En vinder udnævnes |
| Vigtigste success-scenarie | <ol style="list-style-type: none"> 1. Spilleren slår med terningerne 2. Spillerens brik bevæges 3. Spilleren lander på et felt 4. Feltets effekt udføres |
| Alternative scenarier | <ol style="list-style-type: none"> 1. Spillere kan købe/sælge ejendomme 2. Spillere kan leje/udleje ejendomme 3. En spiller kan ryge i fængsel 4. En spiller kan "Gå på besøg" og "Gratis parkere" 5. En spiller kan trække og bruge et chancekort |
| Specielle krav | Systemet skal kunne køre på DTU's computere |
| Teknologi og dataliste | Graphical User Interface (GUI) åbner et separat vindue, som viser spillet. |
| Frekvens | Hver gang spillet startes |

4.4 Domænemodel

Der udarbejdes en domænemodel udfra kundens vision til projektet. Dette gøres ved at finde de relevante objekter fra den virkelige verden, som dette produkt ville bestå af, hvis det blev lavet fysisk. Denne domænemodel vises nedenfor.

I domænemodellen ses at alle klasserne starter ud fra 'Game'. 'Game' Klassen benytter 3-6 'Player', hvor hver af playerne har både en 'Account' og en 'Token' med en farve tilknyttet. 'Game' anvender en 'Cup' som bruger 2 'Die' for at finde en cupValue som bruge til at flytte rundt på 'Board'. 'Board' består selv af 40 'Field' som kan være en af de 7 typer af 'Field' der findes, alle 7 har super klassen 'Field' og arver egenskaberne 'Field' har. Til Game er der et 'Deck' som indholder 45 'ChanceCard', Chancekortene bliver mere specifikke i de nedarvede klasser, f.eks. 'MovingAbs' som har en 'absFieldNo' som bruger til at flytte spillerne til det specifikke felt.

'Game' kan ses om hele vores Controller Package.

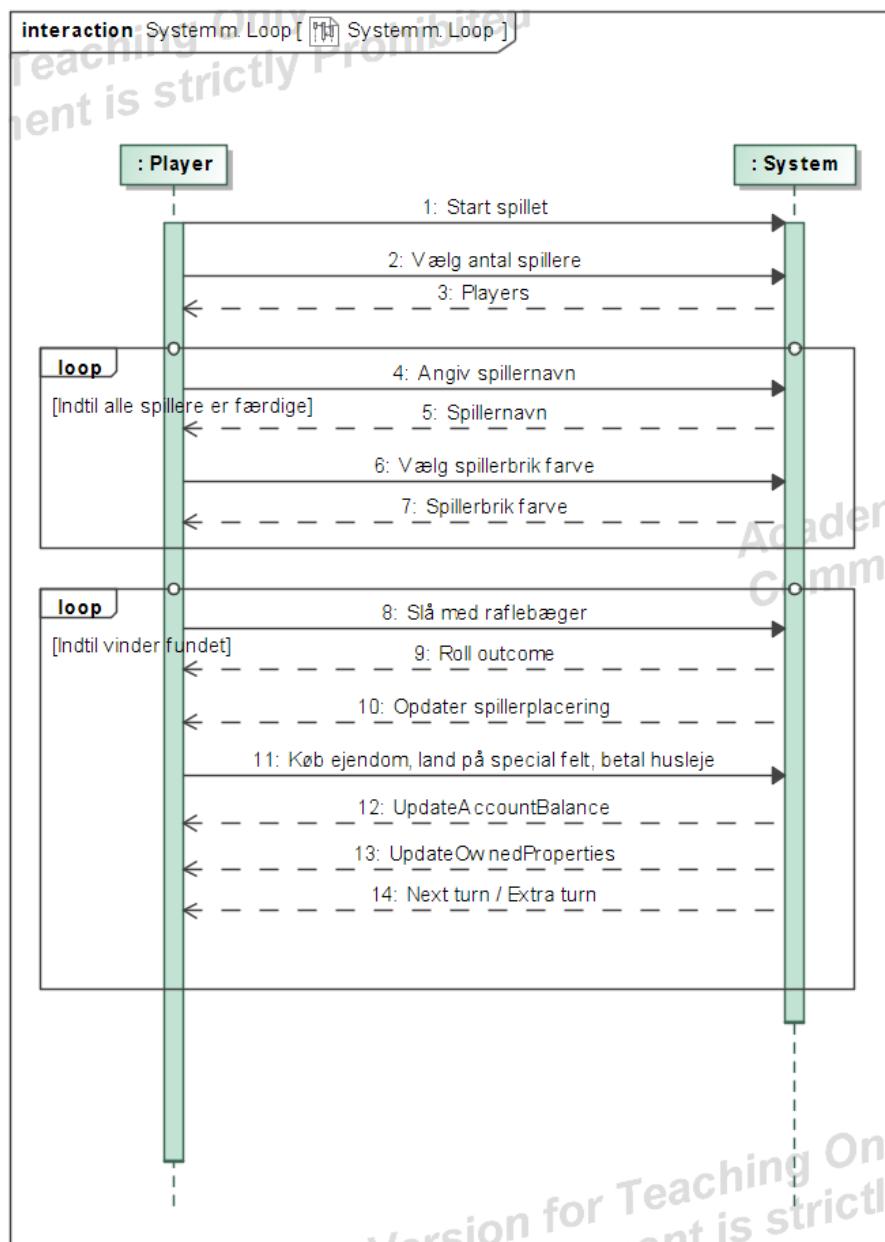


Figur 3: Domæne Model

4.5 Systemsekvensdiagram

Der er opstillet et systemsekvensdiagram, der viser hvordan spillere starter og spiller spillet. Til og med første loop-box beskriver i detalje, de trin der tages, når et nyt spil startes, mens den andet loop-box beskriver det mest grundlæggende, som en tur indeholder.

En bruger vælger, hvor mange som skal spille, hvorefter hver spiller får mulighed for at indtaste spillernavn, samt token-farve. Derefter skiftes spillerne til at slå med terningerne og lande på et felt, hvilket har en effekt på spilleren. Dette fortsætter indtil en vinder er fundet.



Figur 4: Systemsekvensdiagram

5 Design

For at opnå et arbejdsflow, der faciliterer en løsning af opgaven på en tidseffektiv og simpel vis, udarbejdes et udførligt design. Dette gøres vha. GRASP-mønstre, et designklassediagram samt et design sekvensdiagram. Disse beskrives nedenfor.

5.1 Andvendte GRASP mønstre

GRASP beskriver nogle retningslinjer til, hvordan man kan fordele ansvar til klasser og objekter, for at man kan opnå en logisk og let vedligeholdbar objektorienteret kode. GRASP udgør forskellige mønstre og principper, herunder har vi lagt særligt fokus på Creator, Information Expert, Controller, High cohesion og Low coupling.

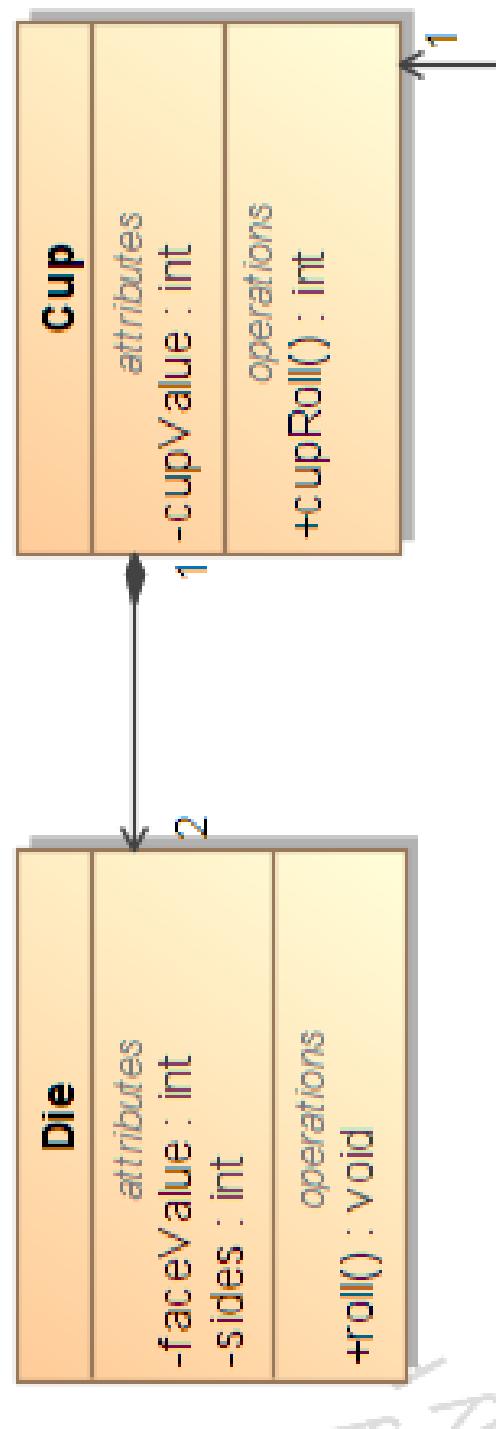
1. **Creator:** En klasse A skal være ansvarlig for at oprette nye instanser af en anden klasse B i visse situationer. F.eks. i klassen Cup bliver der instanceret 2 terninger, derfor er Cup en creator.
2. **Information Expert:** Der er tildelt de forskellige klasser, der har den nødvendige information til at udføre en handling, ansvaret for denne handling. F. eks har "PropertyField" kun ansvaret for, og information vedrørende, ejendomme.
3. **Controller:** En controller er et non-user interface objekt, som modtager og reagerer på en handling/event. Controlleren er ikke UI, men behandler brugerens input på en måde, som modellen kan bruge. F.eks. er der flere controllere hvor den ene "TurnController" styrer alt der fungere på en tur, også hvis scenariet er brugeren er i fængsel.
4. **Low Coupling:** Koblingen mellem klasser er et mål for, hvor afhængige klasserne er af hinanden. På samme måde er matador opbygget så f.eks. kun cup har kendskab til terningerne.
5. **High Cohesion:** Den enkelte klasse tildeles et let, forståeligt og ensartet ansvarsområde, eller eventuelt flere ansvarsområder, der er tæt relaterede til hinanden. Dette er der et godt eksempel på i forhold til Fields eller ChanceCard.
6. **Polymorfi:** Nedarvning (polymorfi) bruges f.eks. ved field klassen, der indeholder information som er gældene for alle fieldtyper, denne kaldes en abstrakt klasse, hvor de forskellige fieldtyper er subklasser som nedarver egenskaberne.

5.2 Design klassediagram

Der er blevet opstillet et design klassediagram på baggrund af domænemodellen, som viser attributter og operationer for de forskellige klasser samt deres indbyrdes forhold.

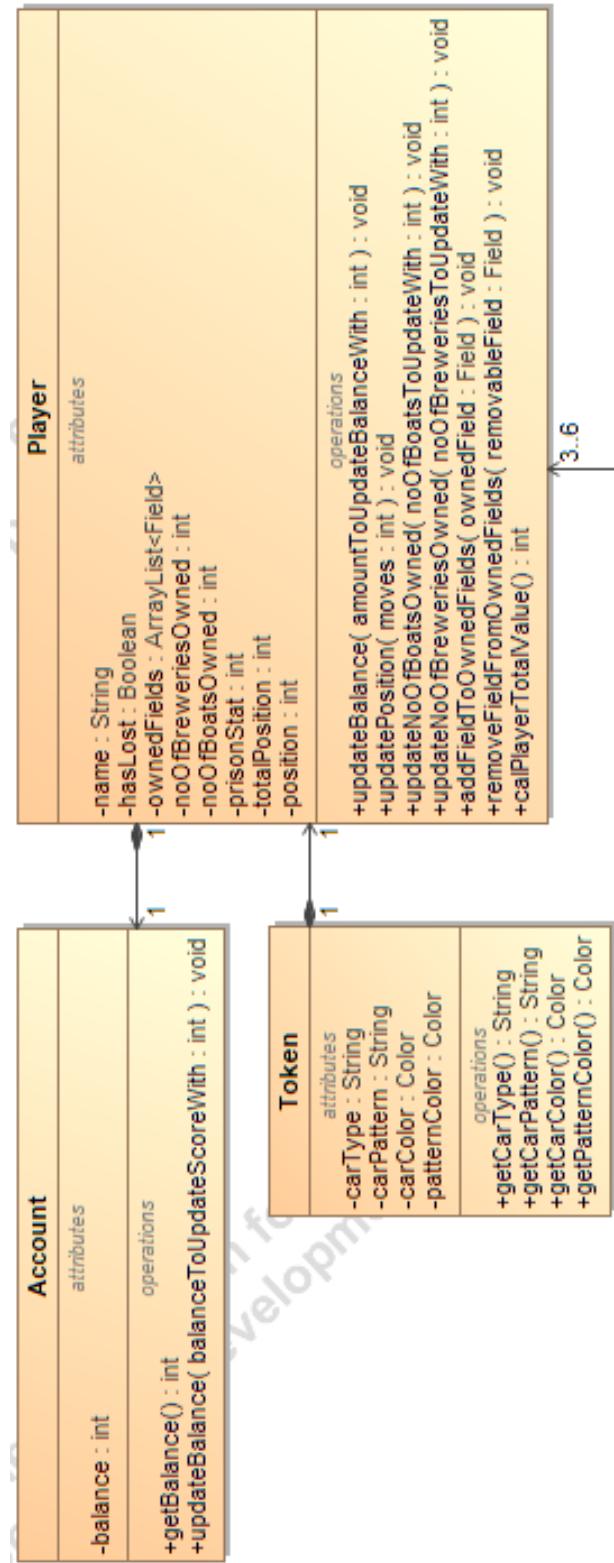
Diagrammet er blevet delt op i fire dele for at muliggøre et overblik over det, som bliver gennemgået i nedenstående sektion. De fire dele er 'Player', 'Cup', 'Deck' og 'Board'. Det fulde diagram kan findes i billaget.

'Cup' indeholder to terninger, som er nødvendige for at 'Cup' har nogen funktionalitet. "Die"klassen har information om terningernes værdi efter et slag, og indeholder en metode for at kunne kaste med terningenerne. 'Cup' klassen indeholder metoden for at kunne slå med to terningobjekter og komme frem til værdien af slaget.



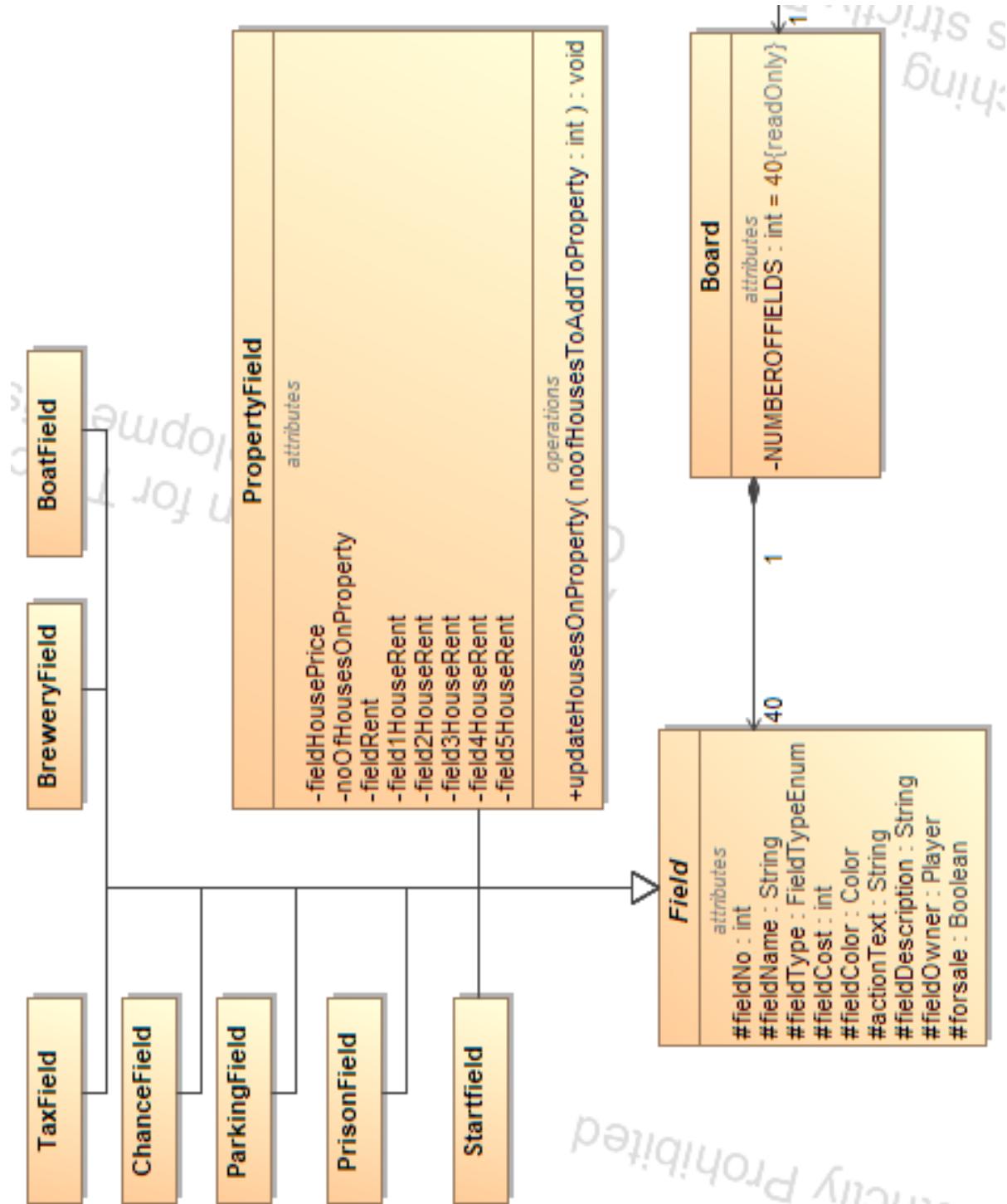
Figur 5: Raflebæger

Spillerne i Matador har hver deres spillebrik og konto. "Account"klassen indeholder spillerens pengebeholdning og metoder til at opdatere denne. "Token"klassen indeholder de forskellige informationer og metoder, der skal til for at lave spillebrikken og gøre den funktionel for spillerene. "Player"klassen indeholder information om spilleren, såsom navn, felter ejet, position på spillebrættet, værdi og status.



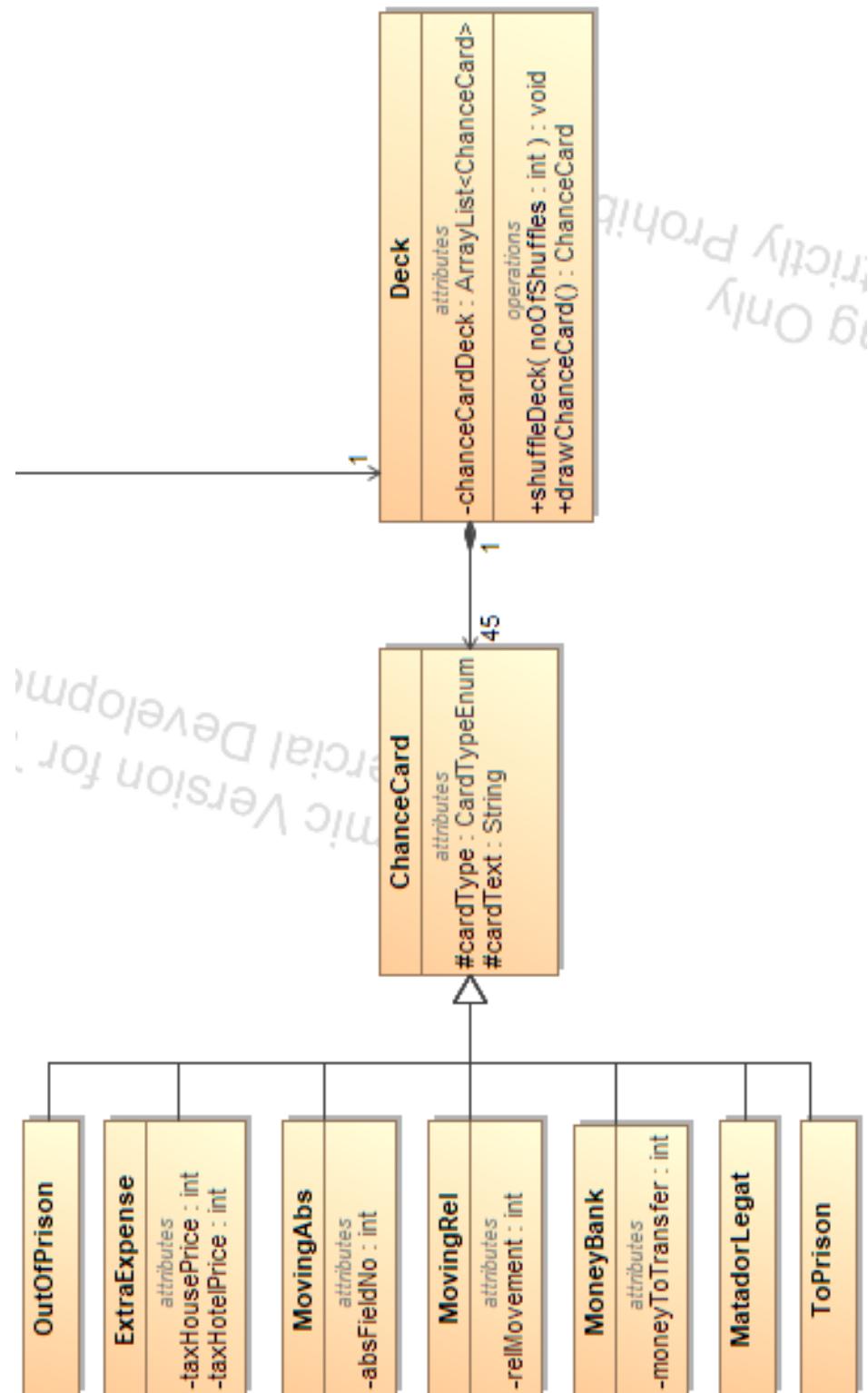
Figur 6: Spiller

Spillebrættet i Matador består af 40 felter, som hver har en effekt og tekst, som påvirker spillet gang når der landes på dem. De 40 felter er alle opdelt imellem 8 specifikke klasser, hvor felternes karakteristiske egenskaber kommer til syne. Der er en abstrakt 'Field' klasse, som alle de forskellige felt typer nedarver fælles egenskaber fra. "PropertyField"klassen indeholder også den information, det gør det muligt at sætte huse og hoteller på ejendomsfelterne.



Figur 7: Spillebræt

Chancekort bliver trukket hver gang man lander på et chancefelt. "Deck" klassen indeholder alle "ChanceCard" og har dem liggende, som i en "bunke". Denne "bunke" kan blandes og der kan trækkes det øverste kort, (lægges i bunden efterfølgende). Den abstrakte "ChanceCard" klasse indeholder information såsom korttype og korttekst, der nedarver til seks forskellige typer af chancekort. De seks forskellige chancekort klasser indeholder information vedrørende deres unikke egenskaber.



Figur 8: Chancekort

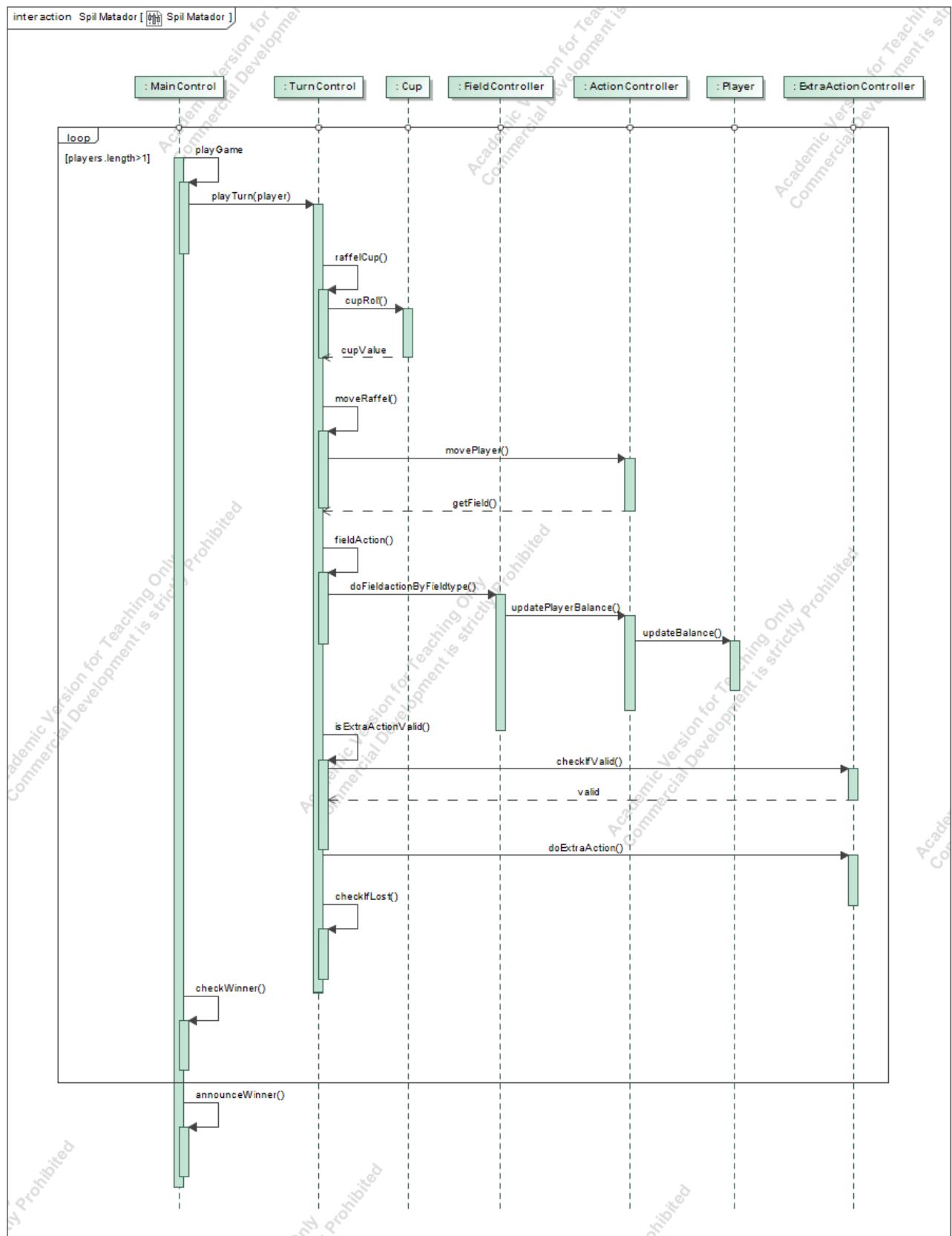
5.3 Sekvensdiagram

Sekvensdiagrammer viser en udveksling af meddelelser, altså metodekald mellem flere objekter, i en specifik, tidsbegrænset situation. Der lægges vægt på rækkefølgen og tiden når meddelelserne til objekter sendes.

Der er udarbejdet et design sekvensdiagram over use case 2 "Spil Matador". Der fremhæver de væsentlige aspekter af en tur. For at gøre diagrammet overskueligt, så er visse elementer af programmet og spillets gang ikke vist.

På diagrammet nedenunder kan man se, at så længe der er mere end én spiller tilbage, så fortsætter MainControlleren med at køre playTurn() i TurnControlleren for de forskellige spillere. Det første en spiller gør på sin tur er at slå med terningerne, fra objektet Cup, hvorefter deres spilbrik bliver rykket af ActionControlleren hen på et felt, i overensstemmelse med ternigsslaget. Dette felts action bliver så udført af FieldControlleren, hvilket som regel resulterer i, at ActionControlleren beder Spiller objektet om at opdatere sin balance.

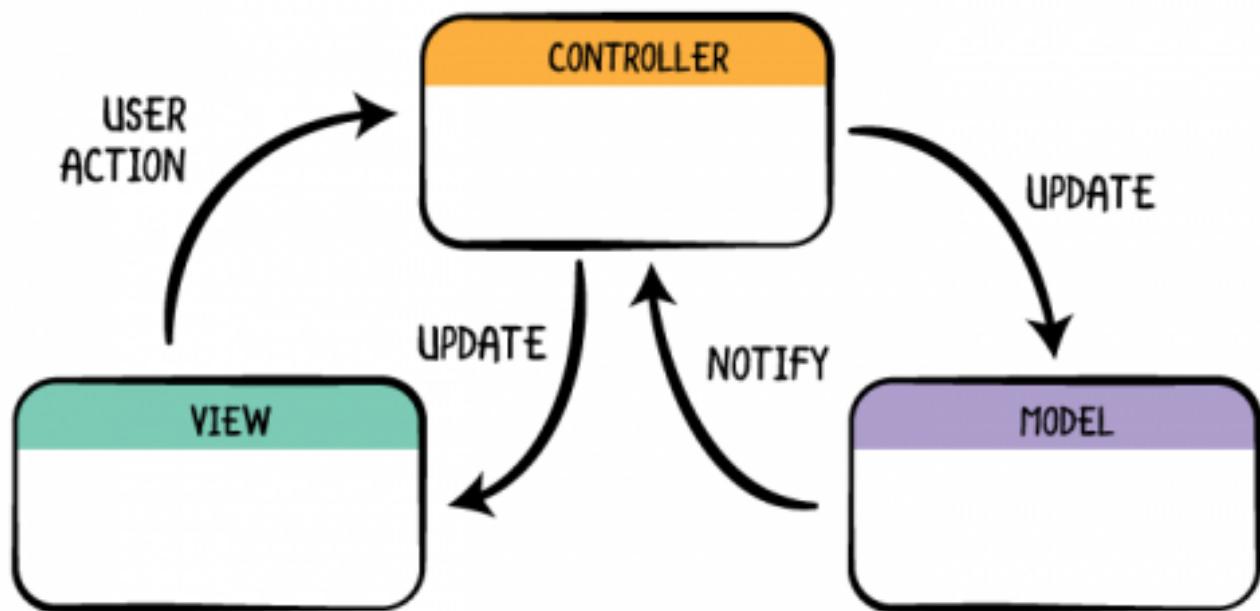
Efterfølgende tjekker TurnControlleren om spilleren har mulighed for at gøre mere, hvor den spørger ExtraActionControlleren om spilleren kan få lov til at udføre en ekstra handling. Hvis denne forespørgsel godkendes, aktiveres doExtraAction(), og spilleren har lov til at udføre sin ekstra handling. Til sidst i turen tjekkes der for, om spilleren er gået fallit eller ej, hvorefter turen slutter, og den næste spillers tur kan gå i gang.



Figur 9: Design sekvensdiagram

6 Implementering

Koden er inddelt i 3 hoved packages - model, view og controller. Dette opfylder MVC (Model View Controller) modellen. 'Model' er de centrale komponenter der er i systemet og fungere som lager af data. 'View' omhandler primært hvordan man ser data og logikken på en skærm, i dette tilfælde arbejdes der med en GUI (Graphic User Interface - Grafisk brugerflade). 'Controller' styrer eller kontrollerer systemet således, at de inputs man får konverteres så de andre dele forstår hvad der skal ske. Denne håndtering af funktioner og handlinger på baggrund af inputs, kan ses om logikken i programmet.



Figur 10: Model View Controller model

6.1 Model package

Model package er en samling af de centrale komponenter i systemet. Det bliver brugt til at lagre data og alt hvad der har med systemet at gøre. Det er altså softwarens dynamiske struktur, som brugeren benytter. Dette bliver viderefivet til Controlleres, som kontrollerer de forskellige inputs, der fås fra brugeren. De mange forskellige "cases" der findes i systemet ligger derfor her i Model package.

6.1.1 Board package

Board indeholder fire klasser: Board, BoardManager, Field, FieldTypeEnum. Den indeholder også en package (Fields), som indeholder ni klasser: BoatField, BreweryField, ChanceField, ParkingField, PrisonField, PropertyField, Startfield, Taxfield og FieldAdder. De første otte klasser i fields-pakken repræsenterer de typer felter, som brættet indeholder.

FieldAdder står for at tilføje felterne dvs. at den tilføjer de forskellige typer af felter. FieldAdder er en arraylist af forskellige felttyper.

Board package står for konstruktion af selve brættet, som spillet foregår på, dets felter og handlinger derpå.

Board klassen fastlægger brættets størrelse ved at have en "FINAL/konstant"enhed. Board klassen står altså for at lave et array med 40 fields.

BoardManager klassen har adgang til en oversigt over felterne og opretter brættet.

Field klassen er abstrakt og danner grundlag for de otte typer af felter, som brættet indeholder. Et felt består af et feltnummer (*int*), navn (*String*), type (*Enum*), pris (*int*), farvekode (*Color*), tekst (*String*), beskrivelse (*String*), ejer (*Player*), antal huse på feltet (*int*) og en indikator om feltet er til salg eller ej (*boolean*).

```
1 public abstract class Field {  
2  
3     /*  
4      ----- Fields -----  
5      */  
6     protected int fieldNo;  
7     protected String fieldName;  
8     protected FieldTypeEnum fieldType;  
9     protected int fieldCost;  
10    protected Color fieldColor;  
11    protected String actionText;  
12    protected String fieldDescription;  
13    protected Player fieldOwner;  
14    protected boolean forSale;  
15  
16    /*  
17     ----- Constructor -----  
18     */  
19  
20    protected Field (int fieldNo, FieldTypeEnum fieldType, String  
21                  fieldName,  
22                  String fieldDescription, int fieldCost, Color  
23                  fieldColor) {  
24        this.fieldNo = fieldNo;  
25        this.fieldType = fieldType;  
26        this.fieldName = fieldName;  
27        this.fieldDescription = fieldDescription;  
28        this.fieldCost = fieldCost;  
29        this.fieldColor = fieldColor;  
30        fieldOwner = null;  
31    }  
31 }
```

Figur 11: Udsnit af den abstrakte Field klasse

PropertyField holder styr på et felts pris og dets husleje, som ændrer sig baseret på hvor mange huse/hoteller der er på feltet.

6.1.2 ChanceCard package

ChanceCard indeholder fire klasser: Card, Deck, DeckManager og CardTypeEnum. Den indeholder også en package (cards), som indeholder 8 klasser; ExtraExpense, MatadorLegat, ToPrison MovingRel, MovingAbs, MoneyBank, OutOfPrison og ChanceCardAdder. De første seks klasser i card-pakken repræsenterer de forskellige typer af chancekort i programmet.

ChanceCardAdder står for at tilføje chancekort dvs. at den tilføjer de forskellige typer af chancekort. ChanceCardAdder er en arraylist af forskellige Chancekort, der hver bliver udtrukket ved drawChanceCard fra Deck Klassen.

ChanceCard package står for alt funktionalitet i forbindelse med chance kortene og stakken af chancekort.

ChanceCard klassen

Deck klassen blander decket med kort og sørger for at hver gang et kort bliver trukket, så ryger det ned i bunden af decket bagefter. Derudover

```
1 public class Deck {  
2  
3     /*  
4      ----- Fields -----  
5     */  
6  
7     private ArrayList<ChanceCard> chanceCardDeck;  
8  
9     /*  
10    ----- Constructor -----  
11     */  
12  
13    public Deck () {  
14        chanceCardDeck = new ArrayList<>();  
15    }  
16 }
```

Figur 12: Udsnit af Deck klassen

6.1.3 Cup package

Cup indeholder to klasser, Die og Cup. Denne package står for oprettelse og håndtering af terningerne og rafflebægret, der bruges i spillet. Den er designet til at man kan variere antallet af terninger og terningernes side.

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

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

returnerer et tilfældigt tal korresponderende til en af terningens sider.

6.1.4 Reader package

Reader klassen står for at indlæse informationen fra csv filerne, vi har, ind i Hashmaps. Dette lader os bruge de forskellige keys i HashMap, og på den måde sikre at vi let kan fikse fejl. Readeren står for at læse informationen der skal bruges til at chancekort, board og beskeder til brugeren.

6.1.5 Player package

Player indeholder tre klasser: Player, Account og Token. Denne package har ansvaret for spilleren og indeholder vedkommendes pengebeholdning, spillebrik, ejede felter og deres position på brættet.

6.2 Controller package

Controller package er en samling af alle controllere og forskellige Managements såsom chanceCardManagement. Denne package indeholder 5 hoved controllere og 3 Management packages.

Hvor modelpakken blev brugt til at beskrive de forskellige objekter, samt deres attributter. Controllerpakken bruges til at styre logikken bag de beskrevne objekters initiation. Her defineres de forskellige objekters metoder vedrørende selve spillet, og deres eksekvering bliver udført.

Controller pakken består af de fem controllere: MainController, SetupController, GuiController, TurnController samt GeneralActionController.

MainControlleren indeholder logikken bag selve spillet som helhed, og det er derfra Setup- og TurnControllerne bliver initieret. SetupControlleren har logikken bag opsætningen af spillepladen, chancekortene samt spillerne selv. GuiControlleren indeholder logikken bag hvordan Gui'en håndteres, mens TurnControlleren indeholder logikken for en spillers tur. GeneralActionControlleren indeholder generelle handlinger, som bruges på tværs af flere logikker, men bruges ofte som bindeled mellem at spilleren kan interagere med gui'en, f.eks. at spillerbrikken rykker sig, eller at man kan købe felter.

Derudover indeholder Controller pakken også de 3 managementpakker: chanceCardManagement, extraActionManagement og fieldManagement.

6.2.1 ChanceCardManagement

ChanceCardAction klassen er en abstrakt klasse og danner grundlaget for alle chanceCardActions. Der findes syv typer af chanceCardActions der nedarver egenskaberne: ExtraExpenseAction, MatadorLegatAction, MoneyBankAction, MovingAbsAction, MovingRelAction, OutOfPrisonArction og ToPrisonAction. Her indeholder f.eks. de to "Prison"relaterede klasser logikken bag chancekort der smider spilleren i fængsel, eller chancekort der giver fribilletter ud af fængslet.

ChanceCardController klassen sørger for at lagre logikken for hvilken handling skal foretages alt efter hvilket chancekort der trækkes. Dette bliver gjort med en simpel switch-statement, hvor fieldtypen bestemmer hvilket udfald der forløber. Nedenstående billede viser et udsnit af koden bag ChanceCardController.

```

1 public class ChanceCardController {
2
3
4     /**
5      * Runs the chanceCardAction matching the cardType of inputted
6      * ChanceCard.
7      * @param player This is the player that does the CardAction.
8      * @param currentChanceCard This is the ChanceCard the Action is
9      * done for.
10     */
11    public void doChanceCardActionFromCardType (Player player,
12        ChanceCard currentChanceCard) {
13        switch (currentChanceCard.getCardType()) {
14            case moneyBank:
15                MoneyBankAction moneyBankAction = new
16                    MoneyBankAction(guiController,messageMap,
17                        generalActionController);
18                moneyBankAction.chanceCardAction(player,currentChanceCard);
19                break;
20            case movingAbs:
21                MovingAbsAction movingAbsAction = new
22                    MovingAbsAction(guiController,messageMap,
23                        generalActionController,board);
24                movingAbsAction.chanceCardAction(player,currentChanceCard);
25                break;
26            case movingRel:
27                MovingRelAction movingRelAction = new
28                    MovingRelAction(guiController, messageMap,
29                        generalActionController,board);
30                movingRelAction.chanceCardAction(player,
31                    currentChanceCard);
32                break;

```

Figur 13: Udsnit af ChanceCardController

6.2.2 ExtraActionManagement

SellFieldAction er en ExtraAction der nedarver egenskaber. Klassen her sikrer at man kan sælge en grund, hvis der f.eks. er en anden spiller som godt kunne tænke sig en af dine grunde. Man får selv lov til at vælge hvor meget man vil sælge grunden for og til hvilken spiller.

BuyHousesAction er en ExtraAction der nedarver egenskaber. Denne klasse står for at man kan købe huse på en grund, hvis man ejer alle felter af samme farve. Det, at man maksimalt kan købe fem huse, og at huslejen forøges efter hvor mange huse man ejer, defineres også her.

ExtraAction er den abstrakte klasse der nedarves fra af Buy- og SellFieldAction.

```

1  public abstract class ExtraAction {
2
3      /*
4      ----- Fields -----
5      */
6
7      protected ExtraActionType_Enum extraActionType;
8      protected GuiController guiController;
9      protected HashMap<String, String> messageMap;
10     protected GeneralActionController generalActionController;
11
12     /*
13     ----- Constructor -----
14     */
15
16     public ExtraAction ( GuiController guiController,
17                         HashMap<String, String> messageMap,
18                         GeneralActionController generalActionController)
19     {
20         this.guiController = guiController;
21         this.messageMap = messageMap;
22         this.generalActionController = generalActionController;
23     }
24 }
```

Figur 14: Udsnit af den abstrakte klasse ExtraAction

ExtraActionController er en controller der styrer ekstra handlingerne. Denne controller modtager input og videre stiller den opgave der skal udføres. Dette gøres sammen med **ExtraAction**, **BuyHousesAction** og **SellFieldAction**, så controlleren ved præcist hvad den eksekverer.

6.2.3 FieldManagement

Dette er en package der indeholder de forskellige typer "Actions" der sker når man lander på et felt. fieldManagement package indeholder en abstrakt klasse **FieldAction**, **FieldController** og en package med otte forskellige typer felter, der findes når en spiller bevæger sig rundt på brættet i spillet.

FieldAction er en abstrakt og danner grundlaget for alle FieldActions. Der findes otte typer af FieldActions der nedarver egenskaberne.

FieldController står for at kontrollere hvilket type felt spillerne lander på. Ud fra det specifikke felt udføres der en Action, som FieldController initialisere. Alt dette styres af en switch-statement der aflæser i Hashmappet typen af feltet. Så når spillerne lander på f.eks. et Propertyfelt, så vil den action af property blive kaldt.

GuiController klassen er en Controller der styre Gui klassen, som ligger i view package. Vi benytter derfor til at transmittere info fra View til Model og tilbage igen, ligesom MVC modellen fremviser.

MainControl klassen står for at starte spillet og er med til at initialisere spillebrættet og Turn logic.

```
1  /**
2   * This method starts the game.
3   * @return Returns non-zero if an unexpected shutdown happens.
4   */
5  public int letsGo()
6  {
7
8      try
9      {
10          // Set the game up, and display
11          setup();
12
13          //region GameLoop
14          do {
15              for (Player currentPlayer : playerArrayList) {
16
17                  turn(currentPlayer);
18
19                  if (extraActionChecker(currentPlayer)) {
20                      extraActions(currentPlayer);
21                  }
22              }
23
24              // Clean and removes a player if hasLost=True.
25              findingHasLostPlayersAndRemovesThem();
26
27          }
28          // Continue while there's more than 1 player left
29          while (playerArrayList.size()>1);
30
31          // ANNOUNCES THE WINNER.
32          announceWinner();
33
34          //endregion
35
36          return 0;
37      }
38      catch (Exception e)
39      {
40          e.printStackTrace();
41          return -1;
42      }
43 }
```

Figur 15: Den primære sekvens ved opstart af spillet

Her ses den primære mekanisme der starter og køre spillet. Spillet kører så længe der er mere end en

spiller tilbage.

SetupControl klassen står får at sætte alt op heriblandt spillepladen, spillerene, Hashmap< String, String > messageMap, Chancekord deck, og farvene som spillerne kan vælge i starten af spillet. Med andre ord så står denne klasse for alt opstætning før spillets start, og den med til at udføre startmekanismen sammen med **MainControl klassen** og **GuiController klassen**

6.3 View package

View package indeholder **GUI klassen** der har ansvaret for den grafiske brugerflade der benyttes. GUI klassen bliver her benyttet til at opsætte det virtuelle spillebræt. GUI klassen har altså derfor kun noget at gøre med den grafiske brugerflade. Den kan vise en besked på skærmen (gui.showmessage("String")), vise et chancekort (gui.displayChanceCard()), tage imod inputs fra brugerne og meget mere. Spillepladen bliver opsat som følgende:

```

1 public class Gui {
2
3     /*
4      ----- Fields -----
5     */
6
7     //<editor-fold desc="Fields">
8     private GUI gui;
9     private GUI_Field[] fields;
10    private ArrayList<GUI_Player> players;
11    private Color backgroundColor = Color.GRAY;
12    private Color textColor = Color.BLACK;
13    //</editor-fold>
14
15    /*
16     ----- Constructors -----
17     */
18
19    //<editor-fold desc="Constructors">
20    /**
21     * This Constructor creates a visual board, with the given fields.
22     * @param fields The list of Field's as Field[]
23     */
24    public Gui ( Field[] fields ) {
25
26        // Initialize Player ArrayList
27        this.players = new ArrayList<>();
28
29        // Create the GUI_Field array
30        this.fields = createFields(fields);
31
32        // Start GUI
33        gui = new GUI(this.fields, backgroundColor);
34
35    }
36 }

```

Figur 16: Udsnit af Gui i View package

Funktionen her Gui (Field[] fields) laver en array af fields (felter), som dermed kan bruges i spillet.

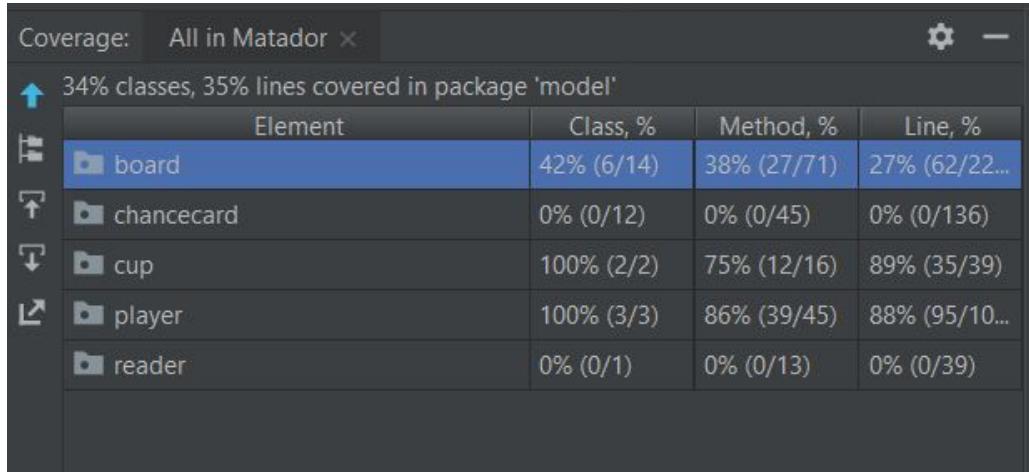
7 Test

Testene er blevet delt op henholdsvis i automatiske tests og bruger tests. Dertil er der 3 situationer, som er blevet testet igennem for at se de fungerer som tiltænkt.

- Start af spil (Bruger test)
- Vind spil (Bruger test)
- Terningen (Junit test)

7.1 JUnit test

Der er flere automatiske test af pakken "Model". Disse tests checker om de metoder, som de forskellige klasser har, fungerer som tiltænkt. Der er en code coverage på 35 procent af alle linjer kode i model.



Figur 17: Code coverage

7.1.1 Terningen

Terningen er blevet testet ved at simulere 600.000 slag, hvorefter antallet af slar der blev 1 udprintes for at se om den er inden for en normal fordeling med maks fire procents udsving (96.000 - 104.000). Fejl kan forkomme på grund af der er en lille mulighed for at terningen slår uden for normal fordelingen.

```

1 // Max deviation limits for the conventional dice
2     int upperBoundCon = (int) ( (amountOfRolls/die.getSides() ) *
3         1.04 );
4     int lowerBoundCon = (int) ( (amountOfRolls/die.getSides() ) *
5         0.96 );
6     // Max deviation limits for the 6 sided dice
7     int upperBound6 = (int) ( (amountOfRolls/die6.getSides() ) *
8         1.04 );    //
9     int lowerBound6 = (int) ( (amountOfRolls/die6.getSides() ) *
10        0.96 );   //
11    /*
12     * Test conventional dice
13     */
14    for ( int i=0 ; i < amountOfRolls ; i++ ) {
15
16        // Roll the dice and put it into conValues

```

```
13     die.roll();
14     conValues[die.getFaceValue()-1] += 1;           // Minus one
15     because of index
16 }
17 for ( int value : conValues ) {
18     System.out.println(value);
19     assertTrue(value > lowerBoundCon && value < upperBoundCon);
20 }
```

7.2 Testcase

Nogle af de mest centrale funktionelle krav er blevet testet vha. testcases. Dette sikrer at de forskellige krav bliver overholdt, og at kunden får det ønskede system.

| Test Case ID | TC01 |
|-----------------|--|
| Summary | "Start spil" |
| Requirements | M1, M3, M4 og M7. Spillet skal initiere så 3-6 spillere kan spille på en plade med 40 felter, hvor de hver især starter med 30.000 kr. |
| Preconditions | Spillet er endnu ikke startet. |
| Postconditions | Spillet er nu klart til at spille, og man kan se forskel på spillerne. |
| Test procedure | Der tændes for spillet og antal spillere mellem 3-6 vælges. Herefter skifter spillerne til at vælge et bruger navn og spillebrik farve. Spillerne med lavere numre har første prioritet, og det er umulig at foretage samme valg som den foregående spiller. |
| Test data | Indsat data: 3 (antal spillere), 1 (spiller 1's navn), sort (1's farve), 2 (spiller 2's navn), hvid (2's farve), 3 (spiller 3's navn) og rød (3's farve). |
| Expected result | Spillet er nu klart til at køre med det valgte antal spillere, og det er spiller 1's tur. |
| Actual result | Spillet startede korrekt op |
| Status | Bestået |
| Tested by | Søren Poulsen |
| Date | 18/01/2019 |

| Test Case ID | TC02 |
|---------------------|---|
| Summary | "Bevægelse på brættet" |
| Requirements | M2, M5 og M6. Spilleren skal slå med to terninger, hvorefter spillerbrikken rykkes ifølge værdien af terningerne |
| Preconditions | Spillerens tur er lige begyndt |
| Postconditions | Spilleren har slået med terningerne og rykket hen til et nyt felt |
| Test procedure | Spillerens tur begynder, der bliver slået med terningerne, og spillerens brik flyttes en antal felter, der svarer til terningesummen. Dette foregår i omløbsretningen med uret. |
| Test data | Der trykkes på ok, hvorefter terningerne bliver kastet og spilleren rykkes til et nyt felt |
| Expected result | Spilleren rykkes alt efter hvad der bliver slået |
| Actual result | Spilleren slog, og brikken blev flyttet samme antal felter, som summen af terningernes øjne. |
| Status | Bestået |
| Tested by | Johannes Piil |
| Date | 18/01/2019 |

| Test Case ID | TC03 |
|---------------------|--|
| Summary | "Vind spillet, ved at de andre går falit" |
| Requirements | M8. Når alle spillere, undtagen én, er gået falit, så vil den sidste spiller tilbage blive udkåret som vinder |
| Preconditions | Spillet er lige startet (ligesom fremgangsmåden i TC01) |
| Postconditions | Én spiller står tilbage som vinderen |
| Test procedure | 3-6 spillere er i gang med spillet, køber grunde, betaler husleje og skat, hvorefter alle på nær én går fallit. Den ene tilbageblivende spiller udnævnes til vinderen. |
| Test data | De 3 spillere skal spille indtil en af dem har vundet |
| Expected result | Vinderen udnævnes, når alle andre spillere er gået fallit |
| Acutal result | Da den sidste spiller gik fallit, så fik denne mulighed for at sælge sine grunde, hvorefter de blev kåret som vinder. |
| Status | Bestået |
| Tested by | Søren Poulsen |
| Date | 20/01/2019 |

7.3 Bruger test

Der er udarbejdet tre testcases hvor TC01 og TC03 begge er lavet som brugere test.

7.3.1 TC01 - Instruktion

Instruktion i hvordan testen skal forløbe, Brugerne skal ikke have forklaret hvordan de gør de forskellige ting, da det skal være intuitivt. Hvis brugerne spørger om hjælp, så skal der svares på deres spørgsmål og det skal noteres.

- Instruktøren sætter programmet igang.
- Brugere bliver bedt om at følgende:
 - Sæt spillet op til 3 spillere.
 - Player 1 skal hedde "Jens" og have farven rød.
 - Player 2 skal hedde "Ida" og have farven blå.
 - Player 3 skal hedde "Bob" og have farven sort.
- Når spillet printer teksten "så er alle vist på plads og spillet kan endelig begynde !" er test TC01 over

7.3.2 TC01 - Testforløb

Spillet blev sat op af 3 forskellige test personer uden nogen spørgsmål. Alle anvendte musen i stedet for at trykke enter.

7.3.3 TC03 - Instruktion

I denne test bliver spillet gjort klar til brugerene, hvorefter de bliver bedt om at spille spillet indtil der er en vinder. Brugerne skal ikke guides igennem hvordan spillet fungerer. Hvis de har spørgsmål, så skal de besvares og noteres.

- Instruktøren sætter spillet op sådan at brugerne starter med 10.000 penge, og initialiserer spillerne ligesom i TC01 (Dog med valg frie navne).
- Brugerne bliver nu bedt om at spille indtil der kun er en tilbage i spillet med penge.

7.3.4 TC03 - Testforløb

Efter 15 minuter var testen færdig og spiller 2 vandt. Der var ikke nogen problemer i selve spillet, men spillerne blev dog skuffede over, at det ikke var muligt at købe huse og hoteller.

8 Dokumentation

8.1 Arv

Arv benyttes til at simplificere et design, når flere klasser har en større del til fælles. Objekter kan arve data og funktioner fra et andet objekt og på den måde udvide dem, altså en eller flere klasser kan nedarve egenskaber, metoder og attributter fra en anden klasse.

Dette inddes strukturalt med en "Superklasse"(hovedklasse) samt en eller flere "Subklasser"(underklasser). En eller flere af disse subklasser nedarver fælles egenskaber fra superklassen, men de specialiserer sig i noget forskelligt.

```
1 public class BoatAction extends FieldAction {  
2  
3     /*  
4      ----- Fields -----  
5     */  
6  
7     private Field currentField;  
8     private int rentFromNoOfBoats = 0;  
9     private String keyForBoatsOwned = null;  
10    private GeneralActionController generalActionController;  
11  
12    /*  
13     ----- Constructors -----  
14    */  
15  
16    public BoatAction(Player player, HashMap<String, String> messageMap,  
17                      GuiController guiController,  
18                      GeneralActionController  
19                      generalActionController, Field  
20                      currentField) {  
21        super(player, messageMap, guiController);  
22        this.currentField = currentField;  
23        this.generalActionController = generalActionController;  
24    }  
25}
```

Figur 18: En klasse der nedarver egenskaber

Her ses det at BoatAction arver egenskaberne fra FieldAction. Det beskrives øverst ved "BoatAction extends FieldAction". Det bliver beskrevet i constructoren hvad der præcist bliver arvet. Det beskrives ved "super(player,messageMap,guiController);"

8.2 Abstract

Formålet ved abstracte klasser er at håndtere kompleksiteten af et program ved at gemme gentagende og ofte ufærdig information, som man så arbejder videre på i andre klasser. Der skal defineres subklasser af abstrakte klasser for at nyttiggøre disse. Det giver ikke mening at instancere abstrakte klasser, da disse indeholder tomme, eller halvskrevne, metoder, som derfor ikke ville kunne køres. Når man vil benytte sig af en abstract klasse, så benytter man sig i stedet for af en instance af en subklasse til den abstracte klasse.

```
1 public abstract class FieldAction {  
2  
3     /*  
4     ----- Fields -----  
5     */  
6  
7     protected Field currentField;  
8     protected GuiController guiController;  
9     protected Player player;  
10    protected HashMap<String, String> messageMap;  
11  
12    /*  
13    ----- Constructors -----  
14    */  
15  
16    public FieldAction(Player player, HashMap<String, String>  
17        messageMap, GuiController guiController) {  
18        this.player = player;  
19        this.messageMap = messageMap;  
20        this.guiController = guiController;  
21    }  
22  
23    /*  
24    ----- Public Methods -----  
25    */  
26  
27    public abstract void action();  
28  
29 }
```

Figur 19: En abstrakt klasse

8.3 ArrayList

Som et alternativ til en traditionel java array kan man benytte sig af en ArrayList. Forskellen mellem en normal java array og en ArrayList er at en ArrayList er mere fleksibel, da man kan ændre størrelsen på den. Derimod vil man ved en normal array ikke frit kunne ændre dens størrelse, men man bliver i stedet nød til at oprette en ny array for at få en anden størrelse.

En ArrayList kan være meget nyttig i programmer, hvor man gerne vil kunne gemme/hente forskellige typer information af en valgfri datatype, i en valgfri mængde af elementer.

8.4 Enum

En enum er en speciel 'klasse', der repræsenterer en gruppe af værdier, som er public, static og final. En enum kan indeholde konstruktører, attributter, metoder osv., men kan ikke skabe objekter eller extend til andre klasser. Det bruges når man har værdier, som ikke kommer til at ændre sig, f. eks dage i en uge eller felterne på et matador bræt. Brug af enum til at definere konstanter gør koden mere læsbar, dokumenterer ligeud listen af accepterede værdier, forhindrer invalide værdier i at skabe fejl i programmet.

```
1 public enum FieldTypeEnum {  
2     Prison, Start, Boat, ChanceCard, Brewery, Property, Tax, Parking  
4 }  
5 }
```

Figur 20: Enum klasse

8.5 HashMap

En HashMap fungerer på den måde, at man kan finde gemt data ved at kigge på dataens tildelte nøgler. Nøgler og deres tilhørende værdier bliver ikke gemt i en rækkefølge når de bliver indsat i en HashMap. Siden at nøgler og værdier i en HashMap hænger sammen, så vil hukommelsesforbrug naturligvis være højere end hos en ArrayList der gemmer sit elements værdi alene og tildeler elementet et indeks internt. ArrayList indeholder elementet der er sorteret efter deres index nummer, og da en HashMap ikke sortere de forskellige nøgler og værdier, vil en HashMap være mindre optimal at bruge, hvis man ønsker at gå frem og tilbage i sin gemte data. Hvis man derimod skal bruge bestemte stykker data, hvor rækkefølgen dataen blev tilføjet er ligegyldigt, kan det være nemmere at bruge en HashMap, hvor man her kan bruge bestemte nøgler til at finde dataen.

```
1 Key;FieldNumber;FieldType;FieldName;FieldDescription;FieldCost;FieldColorRGB;  
    FieldHousePrice;FieldRent;FieldRent1House;FieldRent2House;  
    FieldRent3House;FieldRent4House;FieldRent5House, \\  
2  
3 field1;1;Start;Start;Her starter spillet;0;76:187:23;0;0;0;0;0;0;0;0;0 \\  
4  
5 field2;2;Property;Rodovrevej;Rodovrevej går gennem både Islev og Rodovre  
    forbi Rodovre centrum hvor den til sidst ender ved  
    Roskildevæj.;1200;115:194:251;1000;50;250;750;2250;4000;6000
```

Figur 21: Udsnit af .csv fil

Her ses det hvordan et HashMap kan se ud i en .csv fil.

```

1  public void readFileIntoHashMap(HashMap<String ,String> hashMap) {
2
3      try {
4
5          filePath =
6              getClass().getClassLoader().getResource(fileName).getPath()
7                  .replace("%20", " ");
8
9          bufferedReader = new BufferedReader(new
10             FileReader(filePath));
11
12         while ((line = bufferedReader.readLine()) != null) {
13
14             String[] tempKeyAndValue = line.split(splitter);
15
16             infoArrayIntoHashMap(hashMap ,tempKeyAndValue);
17
18         }
19     } catch(FileNotFoundException e){
20         e.printStackTrace();
21     } catch(IOException e){
22         e.printStackTrace();
23     }
24 }
```

Figur 22: Læsning af .csv fil ind til et HashMap

Her ser man hvordan de forskellige keys til et HashMap bliver læst og lagret. Det bliver bl.a. brugt til at opsætte de vilkårlige felter i spillet.

8.6 Collections

Collections er et framework som giver mulighed for at lave operationer på arrays. Dette program bruger shuffle. Uden at instantiere et objekt kan man benytte dens algoritme. Grunden til at man bruger denne API frem for selv at lave en, er at det selv ville tage længere tid at udføre en sådan funktion. Det gør det muligt at bruge operationer på arrays.

```

1  public Deck () {
2      chanceCardDeck = new ArrayList<>();
3  }
4
5  public void shuffleDeck (int noOfShuffles) {
6      for (int i = 1; i <= noOfShuffles; i++) {
7          Collections.shuffle(chanceCardDeck);
8      }
9  }

```

Figur 23: Udsnit af Deck, ved brug af Collections API

Her benyttes "Collections" til at blande en stak chancekort.

8.7 Import af projekt - GIT

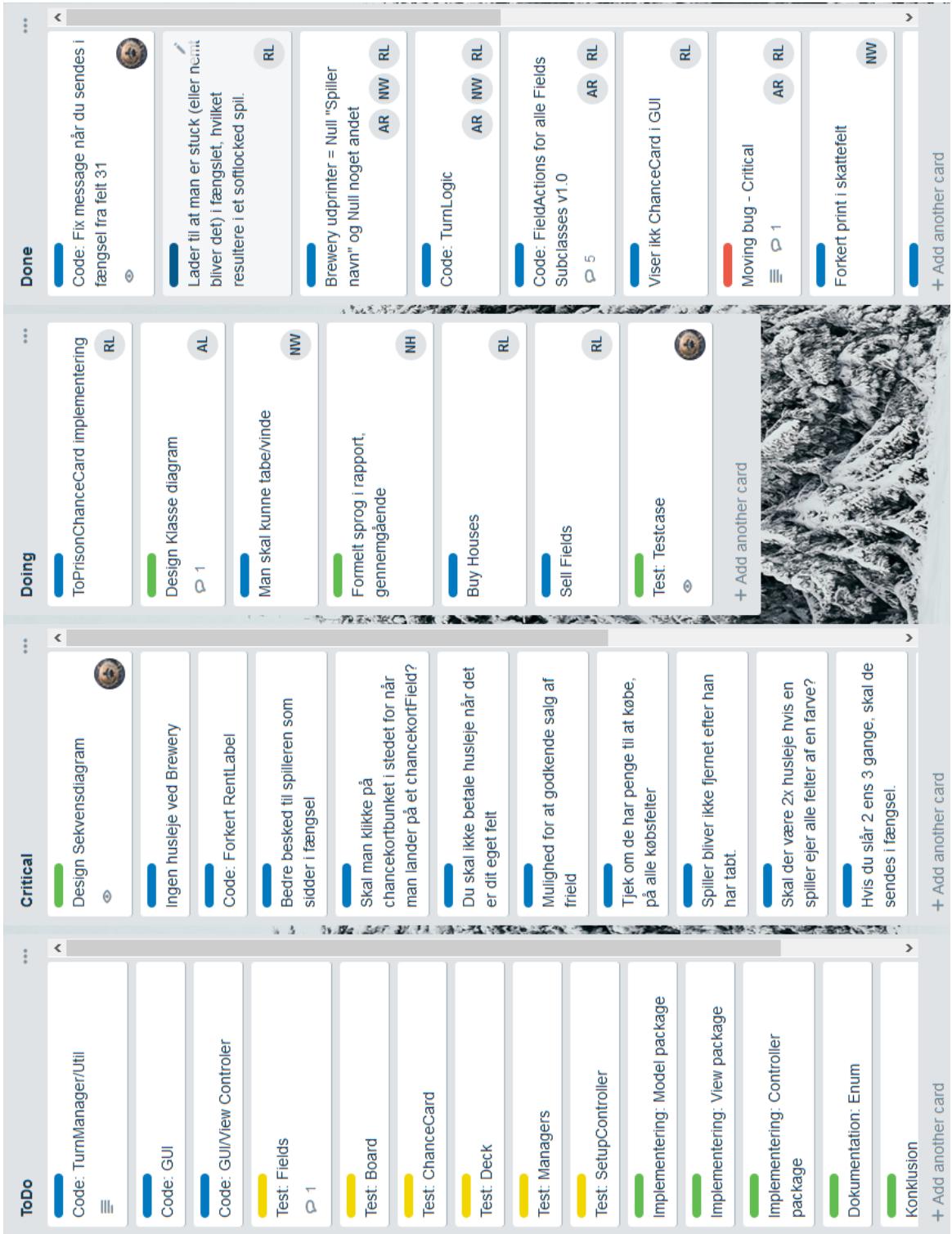
Spillet er udviklet med IntelliJ IDEA 2018.2.7 (Build 182.5107.41) og Windows 10 version 1803. Til at lagre projektet er der benyttet Git/GitHub, og for at skabe den grafiske brugerflade er der brugt Maven til at importere det brugte GUI bibliotek. For at importere projektet så man selv kan redigere og køre programmet, så skal man have Git installeret. Dernæst følger man disse trin:

1. Åben IntelliJ IDEA
2. Vælg "Checkout from Version Control" i VCS menu
3. Indsæt linket fra GitHub (findes på forsiden af denne opgave)
4. Så følger man vejledningen, der kommer på skærmen, og så har man kildekoden til projektet

Hvis du ønsker at køre projektet skal du bare højre klikke på Main i mappen "SRC -> main -> Java -> Main" og klikke "Run". Så skulle programmet køre uden problemer.

8.8 Tidsplan

Der er inkluderet et billede af en trello folder, som blev brugt løbende under projektforløbet til at holde styr på de forskellige arbejdsopgaver, deres deadlines, og hvilket medlem af gruppen som skulle arbejde på det. Billedet er fra d. 18-01-2019, dvs. fire dage inden projektets endelige deadline.



Figur 24: Trello Tidsplan

8.9 Krav gennemgang

8.9.1 Must have

Alle krav under "Must have" er blevet implementeret og er funktionelle.

8.9.2 Should have

De fleste af "should have" kravene er blevet implementeret. Kravet om at en spiller får dobbelt husleje, hvis de ejer alle grunde af en farve, blev ikke implementeret.

8.9.3 Could have

Alle krav mht. fængsel er blevet implementeret. Grunde kommer til auktion, hvis den som lander på feltet ikke ønsker at købe den, og spillere kan sælge ejendomme. Det er ikke muligt at pantsætte en ejendom.

| Krav | Opfyldt | Ikke opfyldt | Krav | Opfyldt | Ikke opfyldt |
|------|---------|--------------|------|---------|--------------|
| M1 | X | | S1 | X | |
| M2 | X | | S2 | X | |
| M3 | X | | S3 | X | |
| M4 | X | | S3.1 | X | |
| M5 | X | | S3.2 | X | |
| M6 | X | | S3.3 | X | |
| M7 | X | | S4 | X | |
| M8 | X | | S4.1 | X | |
| M8.1 | X | | S4.2 | X | |
| | | | S4.3 | X | |
| C1 | X | | S4.4 | X | |
| C1.1 | X | | S4.5 | X | |
| C1.2 | X | | S4.6 | X | |
| C2 | | X | S5 | X | |
| C3 | | X | S6 | X | |
| C4 | X | | S7 | X | |
| | | | S7.1 | X | |
| W1 | | X | S7.2 | X | |
| W2 | | X | S8 | | X |
| | | | S8.1 | X | |

9 Projektforløb

Strukturen for opgaven følger United Process. Det er blevet benyttet GRASP-mønstre gennem opgaven og i særdeleshed også i programmeringsfasen. Normalt bliver UP inddelt i iterationer - typisk over nogle uger, hvor man står med et færdigt fungerende software ved afslutning af hver iteration, dvs. at processen er inkrementel. Dette CDIO4 projekt udfolder sig kun over 3 uger, så man kan kalde det

en enkelt iteration af et stort projekt, hvor de tre forrige CDIO opgaver også indgår (som tidligere iterationer).

Inception fasen (startfasen) er udført ved starten af projektet, da opgaven og dele af dens business case er defineret på forhånd. Produktet er udarbejdet på baggrund af det originale Matador spil. Elaboration fasen indeholder analyse af den stillede opgave ved at oprette en kravsspecifikation og gennem use-case beskrivelser og diagrammer beskrive kravene til systemet. Der bliver opstillet statiske og dynamiske modeller til at understøtte softwarens opbygning. GRASP-mønstrene har været en fast en del af designer under hele forløbt for at sikre et godt opbygget program.

Implementering starter tidligt i arbejdsprocessen, da man ved iterativ udvikling skaber softwaren ved at bygge use-casene. Unified Process er use-case drevet i det man ønsker at realisere use-casene. Alt dette bliver opnået ved brug af versionstyring gennem Github. Dette sikrer at alle gruppens medlemmer kan arbejde på softwaren og følge op på hinandens arbejde. Ved at bruge "Branches", så kan et gruppemedlem f.eks. arbejde med "Turn"mekanismen, mens en anden kan arbejde med "Fields". Den sidste fase i Unified Process er Transition. Dette betyder i vores tilfælde at opgaven skal afleveres.

10 Konklusion

Opgaven lød som følgende: "Udvikl en version af det fulde Matador Spil". Der er på baggrund af denne problemstilling udviklet et Matador Spil, hvor størstedelen af spillet er ligesom det fysiske brætspil. Til det er der også udført en fyldestgørende rapport over hvordan programmet er blevet udviklet. Her er der blevet vist modeller over selve spillet, beskrevet hvordan det hænger sammen og diverse testrapporter over spillet. På den måde er opgaven blevet løst så de vigtigste elementer fra spillet fungerer.

Nogle af delene fra et fysisk Matador spil blev ikke implementeret, eftersom nogle af dem ikke gav mening, f.eks. at have et bestemt antal penge til rådighed at deles om, men der blev foretaget et valg tidligt i forløbet over hvilke krav, som gruppen prioriterede højest for at få skabt et fungerende Matador spil. Alle disse krav og opfyldt, og det færdige produkt lever op til vores krav.

10.1 Perspektivering

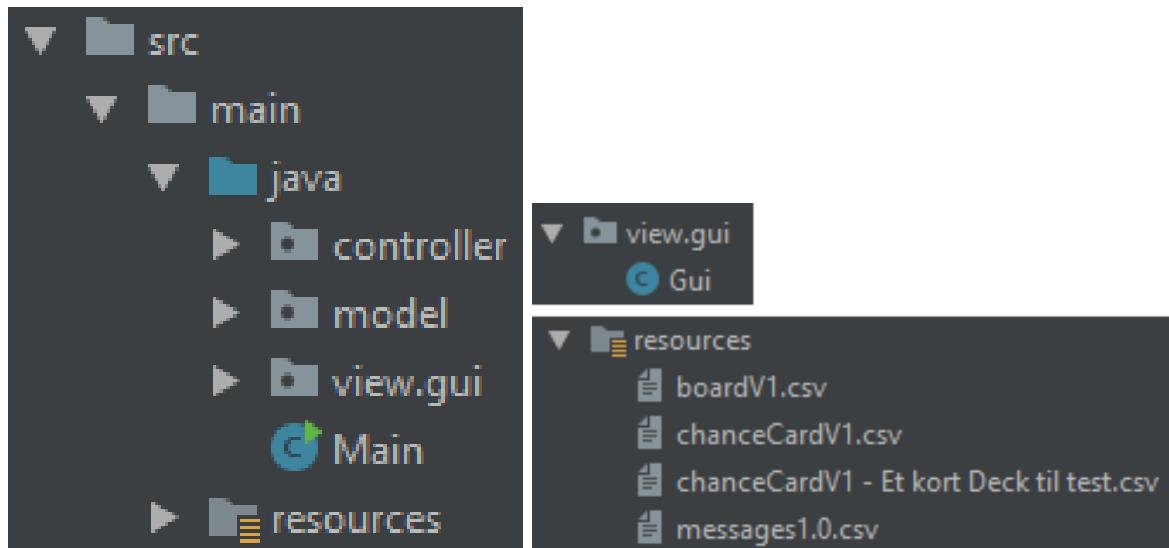
For at skabe et fuldendt Matador spil, så skulle man som spiller f.eks. kunne pantsætte sine grunde, bytte dem med andre spillere, sælge sine huse/hoteller og mere til. Disse krav var der ikke tid til at implementere, og derfor blev de nedprioriteret.

11 Bilag

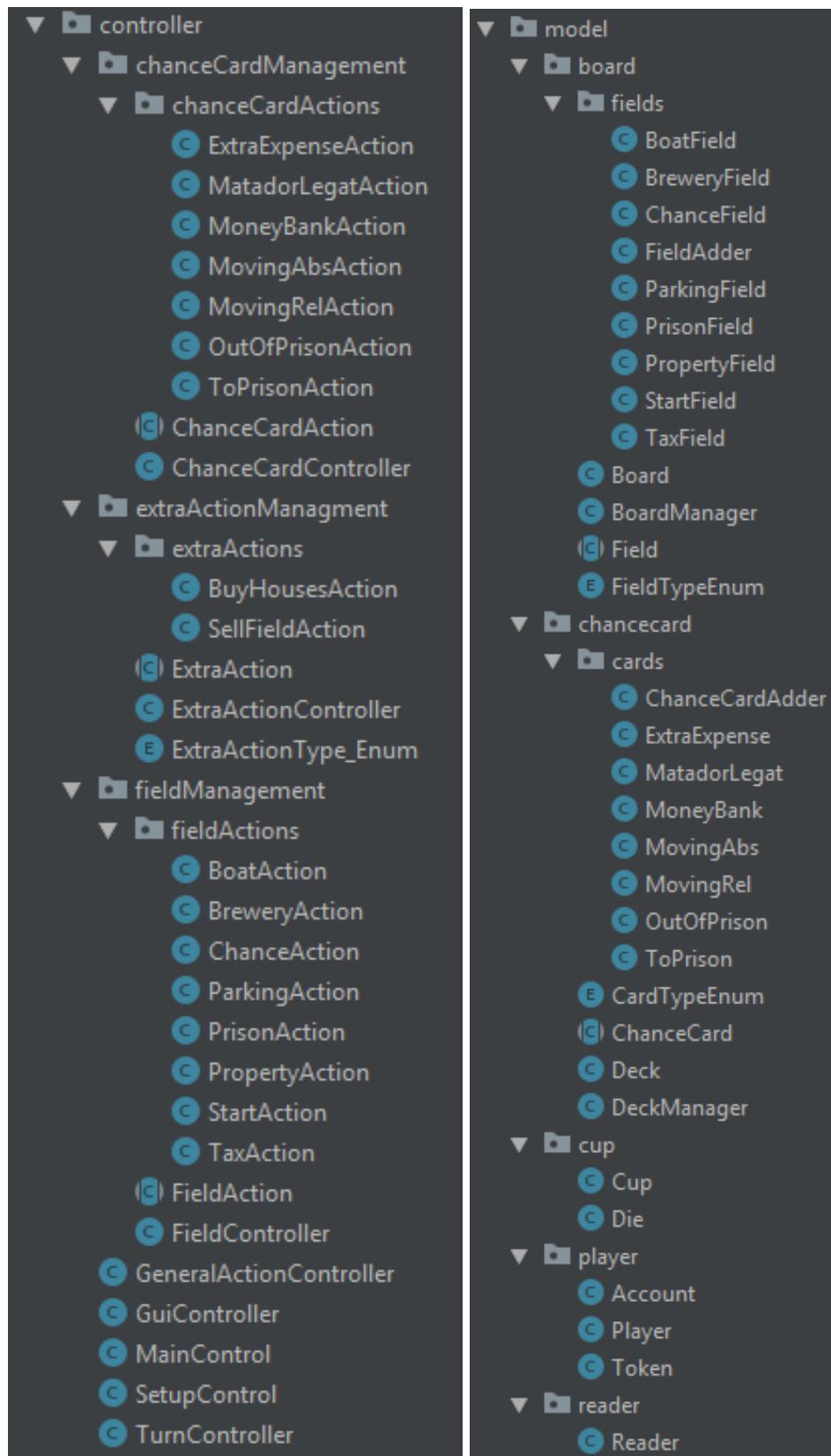
11.1 Github repo

<https://github.com/Logstor/CDIO4>

11.2 Kildekode oversigt



Figur 25: Oversigt Samlet, Resources + view



Figur 26: Oversigt controller + model

11.3 Chancekort

Parameter;cardTypes;cardText;moneyToTransfer;housePrice;hotelPrice;absFieldNo;relMoving
card1;extraExpense;Oliepriserne er steget og De skal betale: kr 500 pr. hus og kr. 2000 pr. hotel;0;500;2000;0;0
card2;extraExpense;Ejendomsskatterne er steget. Ekstraudgifterne er: kr 800 pr. hus og kr. 2300 pr. hotel;0;800;2300;0;0
card3;moneyAll;Det er deres fødselsdag. Modtag af hver medspiller kr 200.;200;0;0;0;0
card4;moneyAll;De skal holde familiefest og får et tilskud af hver medspiller på kr 500.;500;0;0;0;0
card5;moneyAll;De har lagt penge ud for et sammenskudsgilde. Mærkværdigvis betaler alle straks. Modtag fra hver medspiller kr. 500.;500;0;0;0;0
card6;matadorLegat;De modtager 'Matador-Legatet for værdigt trængende' på kr 40.000. Ved værdig trængende forstås, at Deres formue, dvs. Deres kontante penge + skøder + bygninger, ikke overstiger kr 15.000.;40000;0;0;0;0
card7;moneyBank;De har modtaget Deres tandlægeregning. Betal kr. 2.000.:-2000;0;0;0;0
card8;moneyBank;De har købt 4 nye dæk til Deres vogn. Betal kr. 1.000.:-1000;0;0;0;0
card9;moneyBank;Værdien af egen avl fra nyttehaven udgør kr 200, som De modtager af banken.;200;0;0;0;0
card10;moneyBank;Modtag udbytte af Deres Aktier - kr. 1.000;1000;0;0;0;0
card11;moneyBank;Grundet dyrtiden har De fået gageforhøjelse. Modtag kr 1.000.;1000;0;0;0;0
card12;moneyBank;De har solgte nogle gamle møbler på auktion. Modtag kr 1.000 af banken.;1000;0;0;0;0
card13;moneyBank;Betal kr 3.000 for reparation af Deres vogn.:-3000;0;0;0;0
card14;moneyBank;Kommunen har eftergivet et kvartals skat. Hæv i banken kr. 3.000;3000;0;0;0;0
card15;moneyBank;Deres præmieobligation er udtrukket. De modtager kr. 1.000 af banken.;1000;0;0;0;0
card16;moneyBank;Betal kr. 200 for levering af 2 kasser øl.:-200;0;0;0;0
card17;moneyBank;De har fået en parkeringsbøde. Betal kr. 200 i bøde.:-200;0;0;0;0
card18;moneyBank;Betal kr. 3.000 for reparation af deres vogn.:-3000;0;0;0;0
card19;moneyBank;Modtag udbytte af Deres Aktier - kr. 1.000;1000;0;0;0;0
card20;moneyBank;De har vundet i Klasselotteriet. Modtag kr. 500.;500;0;0;0;0
card21;moneyBank;De har været en tur i udlandet og haft for mange cigaretter med hjem. Betal told kr. 200;200;0;0;0;0
card22;moneyBank;De har vundet i Klasselotteriet. Modtag kr. 500.;500;0;0;0;0
card23;moneyBank;Deres præmieobligation er udtrukket. De modtager kr. 1.000 af banken.;1000;0;0;0;0
card24;moneyBank;Betal Deres bilforsikring - kr. 1.000;-1000;0;0;0;0
card25;moneyBank;Betal for vognvask og smøring kr. 300;-300;0;0;0;0
card26;moneyBank;De modtager Deres aktieudbytte. Modtage kr. 1.000 af banken.;1000;0;0;0;0
card27;moneyBank;De havde en række med elleve rigtige i tipning. Modtag kr. 1.000.;1000;0;0;0;0
card28;moneyBank;De har kørt frem for 'Fuldt stop'. Betal kr. 1.000 i bøde.:-1000;0;0;0;0
card29;movingAbs;Tag ind på Rådhuspladsen.;0;0;0;39;0
card30;movingAbs;Ryk frem til 'START'.:0;0;0;0;0
card31;movingAbs;Ryk frem til Vimmelskaftet. Hvis De passerer 'START', indkassér da kr. 4.000;0;0;0;32;0
card32;movingAbs;Ryk frem til Frederiksberg Allé. Hvis De passerer 'START', indkassér da kr. 4.000;0;0;0;11;0
card33;movingAbs;Tag med Mols-Linien. Flyt brikkens frem, og hvis De passerer 'START', indkassér da kr. 4.000;0;0;0;15;0
card34;movingAbs;Ryk frem til Grønningen. Hvis De passerer 'START', indkassér da kr. 4.000;0;0;0;24;0
card35;movingAbs;Ryk frem til 'START'.:0;0;0;0;0
card36;movingAbs;Ryk frem til Strandvejen. Hvis De passerer 'START', indkassér da kr. 4.000;0;0;0;19;0
card37;movingRel;Ryk tre felter tilbage.:0;0;0;0;-3
card38;movingRel;Ryk tre felter tilbage.:0;0;0;0;-3
card39;movingRel;Ryk tre felter frem.:0;0;0;0;3
card40;toPrison;Gå i fængsel. Ryk direkte til fængslet. Selv om De passerer 'START', indkasserer De

ikke kr. 4.000;0;0;0;11;0

card41;toPrison;Gå i fængsel. Ryk direkte til fængslet. Selv om De passerer 'START', indkasserer De ikke kr. 4.000;0;0;0;11;0

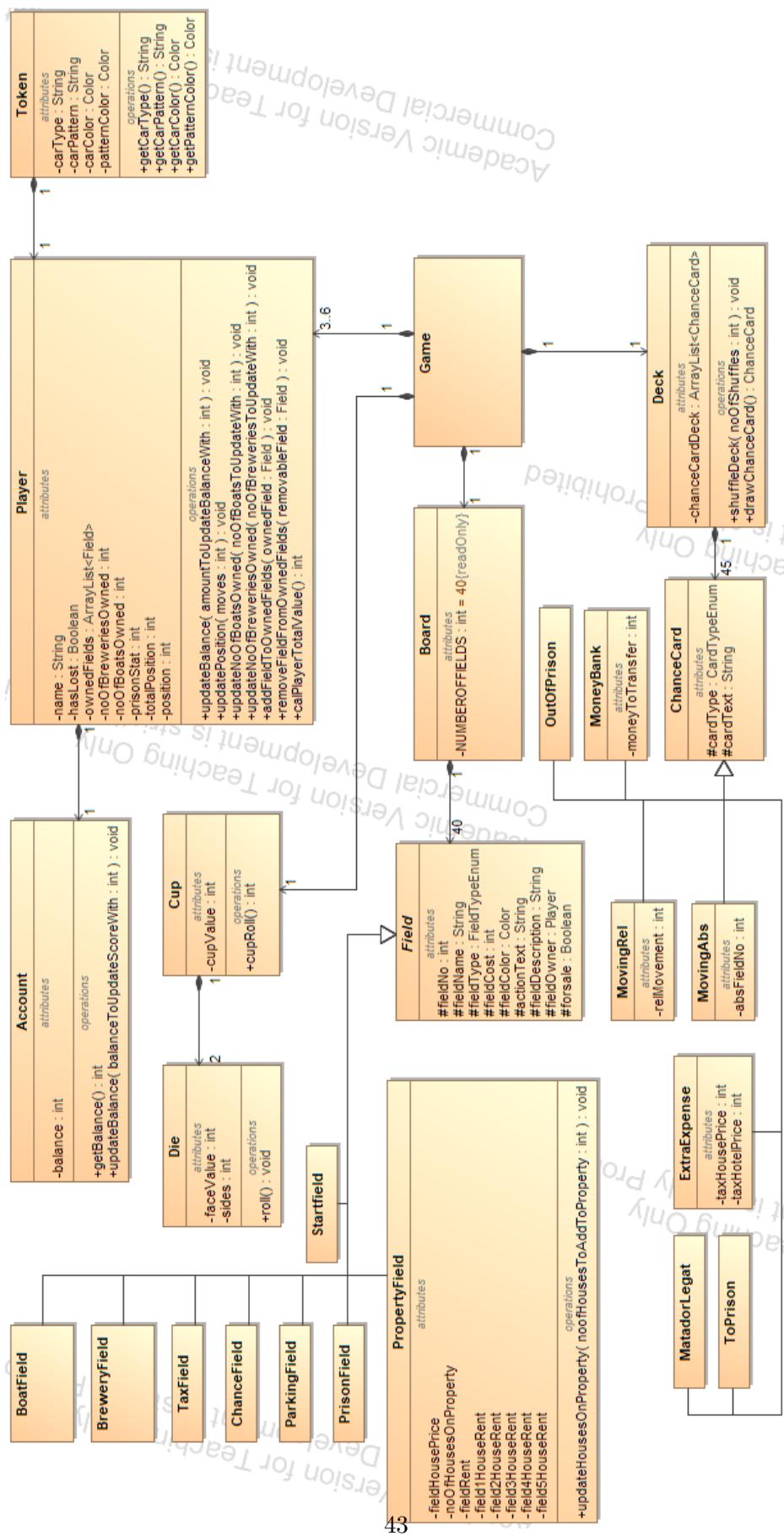
card42;nearestBoat;Ryk brikken frem til det nærmeste rederi og betal ejeren to gange den leje, han ellers er berettiget til. Hvis selskabet ikke ejes af nogen, kan De købe det er banken.;0;0;0;0;0

card43;nearestBoat;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 er banken.;0;0;0;0;0

card44;outOfPrison;I anledning af kongens fødselsdag benådes De herved for fængsel. Dette kort kan opbevares, indtil De får brug for det, eller De kan sælge det.;0;0;0;0;0

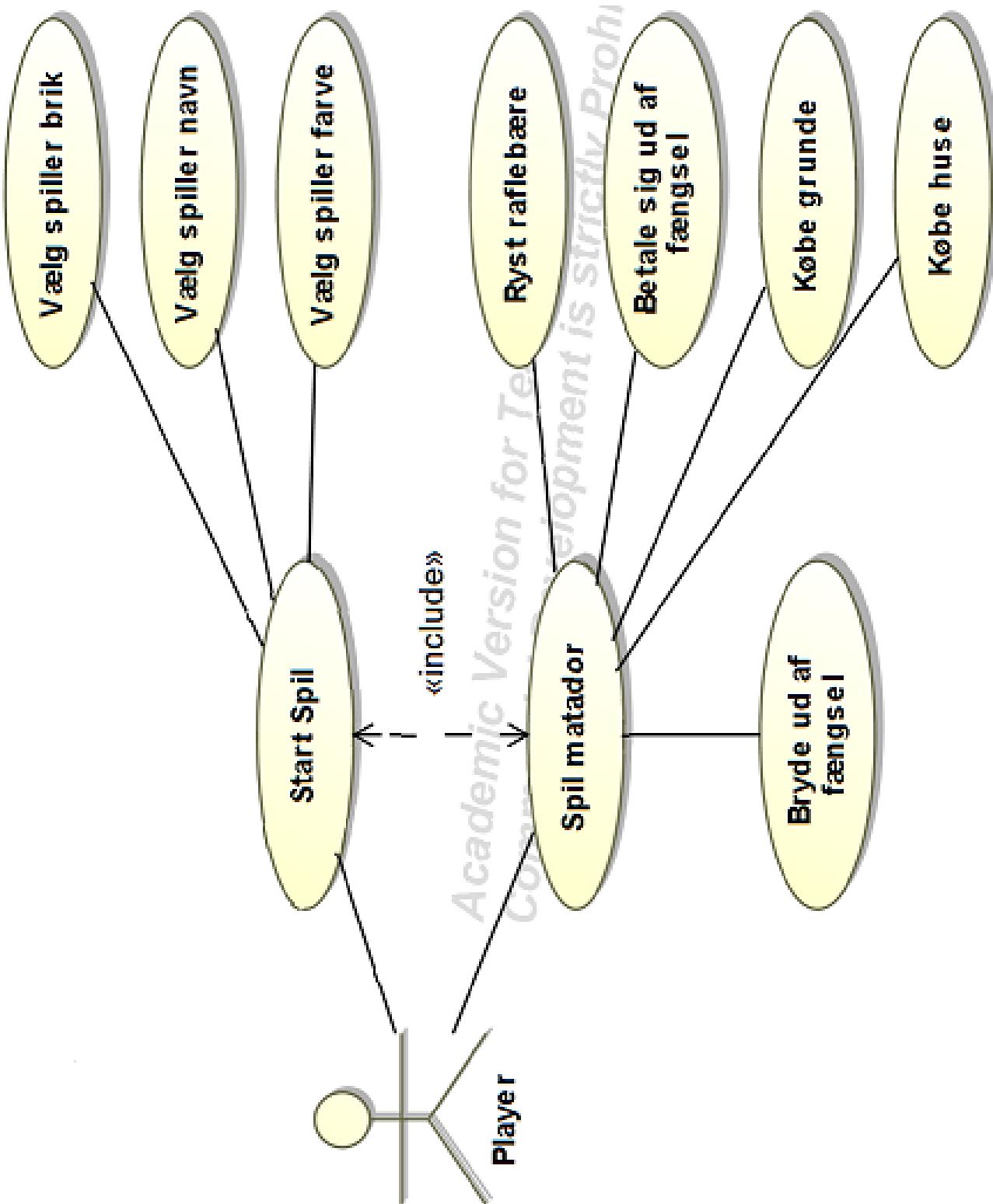
card45;outOfPrison;I anledning af kongens fødselsdag benådes De herved for fængsel. Dette kort kan opbevares, indtil De får brug for det, eller De kan sælge det.;0;0;0;0;0

11.4 Designklassediagram



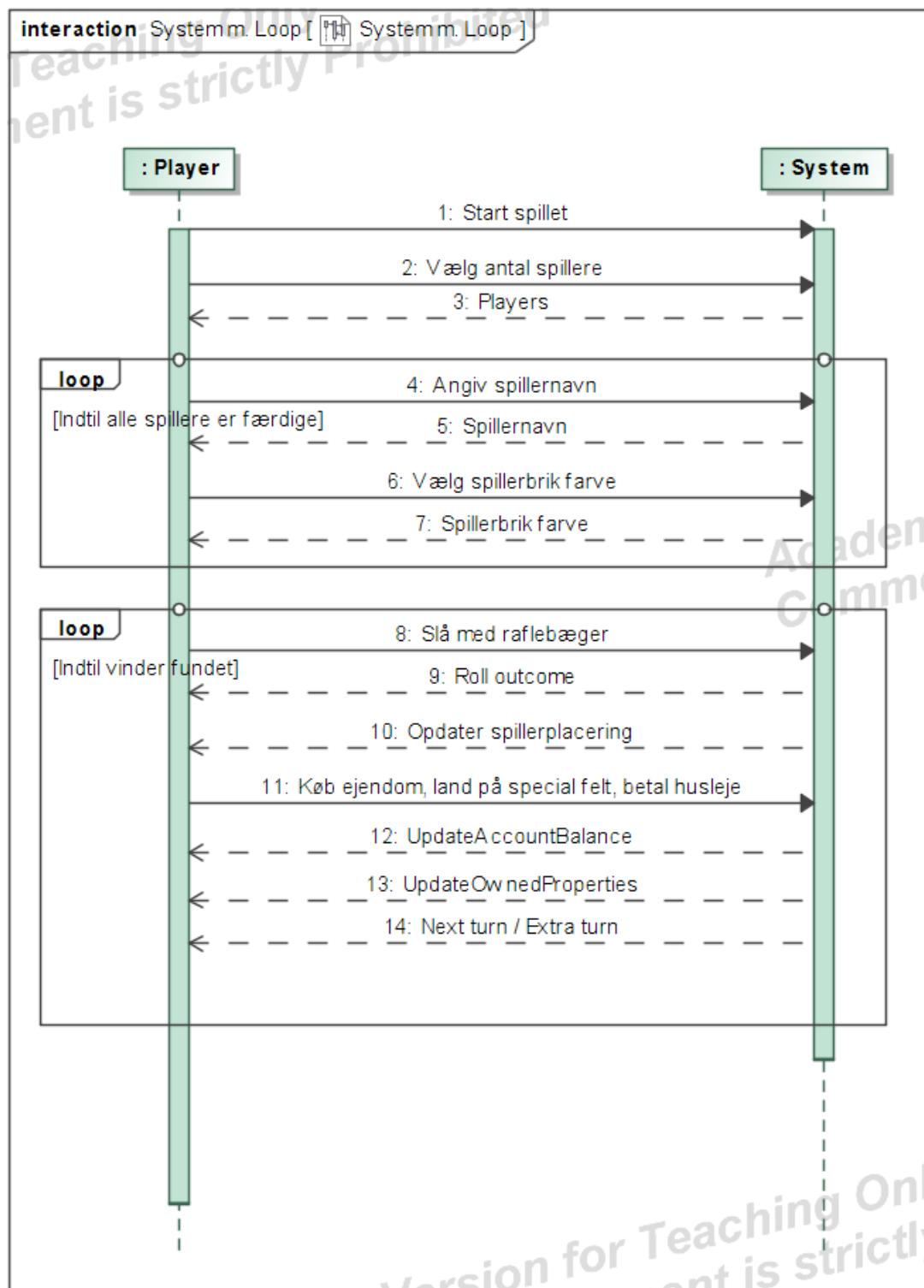
Figur 27: Design klassediagram

11.5 Use Case diagram



Figur 28: Use Case Diagram med SubUseCases

11.6 SystemSekvensDiagramm



Figur 29: Systemsekvensdiagram