G15 CDIO final

Indledende Programmering

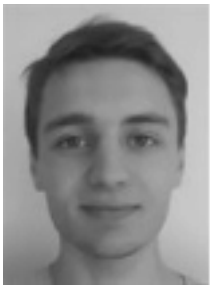
02312-14

Afleveringsfrist: 20. Januar 2020

Github, Group 15 Repository

Gruppe 15



Christian	*Kamilia*	*Oliver*	*Tobias*	*Søren*	*Alpha*
s184140	*s195466*	*s176352*	*s195458*	*s182881*	*s190614*

Denne rapport indeholder 42 sider inkl. bilag og denne side.

Abstract

This project is the conclusion of the course “Indledende programmering”. The project builds around the previous CDIO projects from the first semester. This final assignment is to make the fully functional version of the game “Matador” using the previous projects as help, in terms of the programs architecture and specific functionalities, as well as a complete report describing the process and final product, but with the biggest goal being to add all features you can find in the original game. The development process has been to work on the report and product in parallel while using the techniques and guidelines taught in the three CDIO Courses from first semester. That means creating a general idea of the architecture and requirements for the program before beginning the programming, and then move on to create the product using Object-oriented programming and MVC to have a clean and simple program. Lastly test and finish everything to see if all requirements has been accounted for. These entities have been ensured by the three deadlines for the project to make sure it was a smooth experience and everything was as it should be throughout the project and this has concluded into a fully functional program as well as a comprehensive report describing the product.

Timeregnskab

Dato	Krav og Analyse	Design	Implementering	Test
6/1/2020	12	5	15	0
7/1/2020	10	2	20	8
8/1/2020	10	5	20	2
9/1/2020	10	1	25	0
10/1/2020	12	6	25	1
11/1/2020	0	0	0	0
12/1/2020	0	0	0	0
13/1/2020	6	9	10	0
14/1/2020	3	8	6	4
15/1/2020	3	4	2	9
16/1/2020	3	5	2	7
17/1/2020	1	2	0	0
18/1/2020	0	0	0	0
19/1/2020	0	0	0	0
20/1/2020	0	0	0	0
Timer i alt	**70**	**47**	**125**	**31**

Navn	Krav og Analyse	Design	Implementering	Test	I alt
Christian	14	10	20	6	**50**
Kamilia	5	4	10	2	**21**
Alpha	10	8	25	9	**52**
Søren	13	8	20	9	**50**
Oliver	10	5	30	2	**47**
Tobias	18	12	20	3	**53**
Timer i alt	**70**	**47**	**125**	**31**	**273**

Tabel: 0.1 Timeregnskab

Indholdsfortegnelse

1. Indledning

Som videre arbejde af de tidligere CDIO opgaver og som afslutning af faget “Indledende programmering”, er vi blevet bedt om at udarbejde et færdigt program til simulering af et af Matador spil i en dansk udgave. Programmet skal være mellem 3-6 spillere, hvor man blandt andet kan lande på forskellige felter, trække chancekort, pantsætte sin ubebyggede grunde og hvoraf balancen ændrer sig herefter. Reglerne er taget med udgangspunkt i en nyere version af matador hvoraf grundene og chance kortene er fra en ældre version. Dette har gjort at vi har justeret nogle af spillets værdier for at give en god spiloplevelse.

Til at udarbejde produktet, har vi gjort brug for IntelliJ, Github og Magicdraw til hhv, at programmere, versionere og lave diagrammer og modeller. Valget af programmer gør at vi alle kan arbejde selvstændigt samtidig med at man deler det med hinanden. Selve Matador spillet er taget som udgangspunkt i de forskellige krav og designet ud fra vores domæne diagram. Derudover har vi videreudviklet programmet ud fra vores use case og feature liste og de dele af programmet som vi i forvejen havde arbejdet med i de foregående CDIO opgaver er blevet tilføjet som set passende. Alle de centrale funktioner for spillet er blevet tilføjet til programmet og kun meget få regler vi har set som værende enten ubetydelige eller mindre vigtige er ikke med i det endelige program.

2. Analyse

2.1 Krav

Kundens vision:

Kunden ønsker sig den fuldstændige udgave af spillet "Matador". Den endelige version skal kunne køre på Windows maskinerne, i de forskellige databarer rundt på DTU. Spillet skal foregå mellem 3-6 spillere på en spilleplade med 40 felter. Alle de forskellige felter fra det originale Matador skal være repræsenteret på spillepladen herunder: start, grunde, chancekort, på besøg i fængsel, fængsel, helle, færgekort og ølkort. Hertil skal der også være de 32 chancekort med deres individuelle konsekvenser og effekter. Alt dette kan findes under "Bilag 1" og "Bilag 2".

Reglerne følger en nyere version af spillet men alt andet - herunder felter, chancekort og spilleplade - er fra en ældre version. Dette har dog ikke nogen betydning bortset fra at vi selv har justeret nogle af værdierne for at spillet kan fungere optimalt og er retfærdigt. Man skal kunne købe eller sælge grunde og grunde skal sættes på auktion hvis man lander på et ledigt felt man ikke vil købe. Der skal være leje, man skal kunne købe huse og alle de resterende regler. Vinderen findes når kun en spiller er tilbage, som hermed er blevet Matador.

Kravspecifikation:

Kravene bliver prioriteret ud fra MoSCoW-princippet:

M - Must have **S** - Should have **C** - Could have **W** - Won't have

Nedenfor ses de "Must have" krav vi har vurderet til projektet, og den generelle spiloplevelse. Den fulde kravliste kan findes i "*Bilag 3"*.

Funktionelle krav:

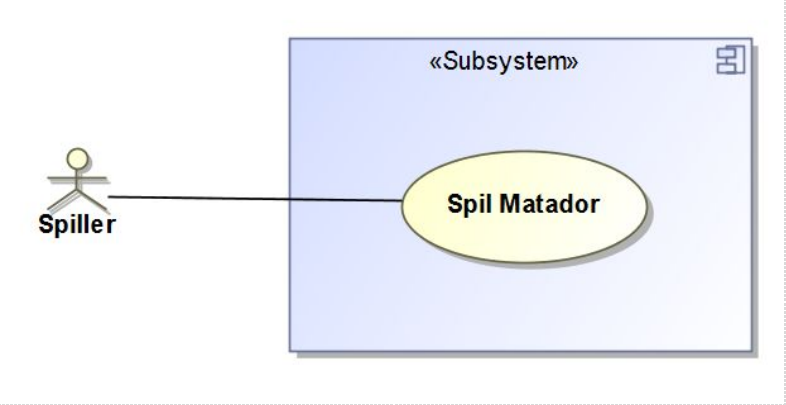
Id.	Beskrivelse	Prioritet
RQF01	Spillet skal være mellem tre til seks personer	M
RQF03	Spillerne skiftes til at kaste to terninger	M
RQF04	Spillerne rykker skiftevis rundt, i en længde bestemt af summen af de to terninger	M
RQF05	Spillebrættet består af felter	M
RQF06	Spillebrættet har felt kategorierne; start, ejendomme, skat, chancekort, på besøg i fængsel, fængsel, helle, færgekort og ølkort.	M
RQF37	Hvis en spiller vælger "Afslut tur" går turen videre, og næste spiller kaster med terningerne.	M
RQF38	En spillere modtager en ekstra tur hvis de slår to ens	M
RQF48	En spillere udgår hvis de løber tør penge ressource, til at opholde en pengebeholdning over 0 kr	M

RQF49	En spillere begynder med 3000 kr	M
RQF50	Spillet stopper når kun en enkle spillere står tilbage	M

Ikke funktionelle krav:

Id.	Beskrivelse	Prioritet
RQIF01	Spillet skal kunne tilgås via maskinerne i DTU's databarer, uden bemærkelsesværdige forsinkelser.	M
RQIF02	Der skal udarbejdes en beskrivelse af minimumskrav samt vejledning i hvordan kildekoden compiles, installeres og afvikles. Dette inkluderer en beskrivelse af hvordan koden importeres fra et git repository.	M

2.2 Use-Case



Figur: 2.2.1 Use-Case Diagram

Da vi ligesom de tidligere CDIO opgaver har at gøre med et spil er vores use-case rimelig simpel, da alt aktøren kan gøre er at spille spillet. Dermed har vi taget mere fokus i vores Brief, Casual og Fully dressed, som gennemgår hele spillets gang og de forskellige muligheder spilleren har inde i spillet.

Brief Use Case
UC1: Spil Matador Man spiller tre til seks spillere på en plade med 40 felter. I spillet rykker hver spiller rundt. Spillet er slut når alle spillere er gået bankerot bortset fra en og den sidste spiller tilbage vinder.

Casual Use Case - UC1: Spil Matador
Spillet starter med aktørerne initialiserer spillerne. Mængden af spillere går fra tre til seks, med den bestemte startkapital. Hertil slår systemet så med terningerne på hver spillers tur og flytter spilleren til det rigtige felt. På hvert felt skal spilleren udføre feltets beskrivelse. Chancekort har mulighed for at give eller tage penge samt uddele frikort til at undgå fængsel. Spillet er slut ved at alle spillere er gået bankerot bortset fra en og dermed er vinderen den sidste spiller tilbage som bliver Matador

Fully dressed	
Use Case Navn	UC1: Spil Matador
Scope	Matador spil
Level	User goal - Spille Matador
Primær aktør	Spiller
Interessenter	Ingen
Preconditions:	Spillerne initialiseres med deres individuelle navne og med den bestemte startkapital
Postconditions/ Success Guarantee	Når en spiller står tilbage, stopper spillet.
Hoved Succes scenarie	1. Programmet tjekker om spilleren er i fængsel 2. Spilleren kaster med terningerne 3. Spilleren lander på et felt 4. Kan forhandle med andre spillere 5. Kan pantsætte grund 6. Kan købe huse på eget fuldendte sæt af grunde 7. Tjekker om spilleren er gået fallit 8. Tjekker om Spillet er færdig
Frequency of Occurrence:	3-6 gange per runde afhængig af antal spillere.

Alternative-flows
1. Fængsel tjekker-flow
1.1 Spilleren er ikke i fængsel 1.a Spilleren går til scenarie 2 1.2 Hvis spilleren er i fængsel 2.a Spilleren kan betaler sig ud, går til scenarie 2 2.b Spilleren benytter “Get out of jail”-kort, går til scenarie 2 2.c Spilleren slår to ens, hvorefter spilleren rykkes antal felter
3. Lande på et felt-flow
3.1 Lander på en grund 1.a Hvis grunden er ledig 1.a Kan grunden købes 1.a Spilleren betaler banken, og ejer nu grunden 1.b Kan sættes auktion 1.a Spillerne kan byde på grunden, og vinderen køber grunden til det vindende bud 3.2 Grunden ejes af en anden 2.b Spilleren betaler angivet leje til ejeren 3.3 Lander på “Chancekort”

3.a Spillet trækker et chancekort 3.b Udfør chance kortet beskrivelse 3.4 Lander på “gå i fængsel” 4.a Spillerens brik rykkes og ryger i fængsel 3.5 Lander på “Fængsel” 5.a Spillerens brik er på besøg i fængsel, og kan fortsætte uden straf, ved næste kast 3.6 Lander på “Helle” eller “Start” 6.a Spilleren står på et fristed, indtil næste kast
4. Forhandle-flow
4.1 Hvis spiller ønsker at forhandle 1.a Spilleren vælger en modspillers grund, og præsentere købsprisen. Herefter går handlen igennem 4.2 Hvis spilleren ikke ønsket at forhandler 2.a Spillet fortsætter uden forhandling
5. Pantsætte grund-flow
5.1 Hvis spiller ønsker at pantsætte grund 1.a Spilleren pantsætte grund, der bliver inaktiv, og modtager halvdelen af den oprindelige købspris 5.2 Hvis spilleren ikke ønsker at pantsætte grund 2.a Spillet fortsætter uden pantsætning
6. Køb af huse-flow
6.1 Hvis spilleren har fuldendt sæt af grunde, og tilhørende nødvendige ressourcer 1.a Kan spiller købe hus på en 6.2 Hvis spilleren ikke besidder fuldendte sæt, penge eller lyst til at købe hus 2.a Spillet fortsætter
7. Tjek fallit-flow
7.1 Hvis spilleren ikke har flere ressourcer til betaling af nuværende udgifter 1.a Spilleren udgår af spillet 7.2 Hvis spilleren har aktiver for at betale sine nuværende udgifter 2.a Spilleren betaler, og spillet fortsætter
8. Tjek om spil er færdigt-flow
8.1 Hvis kun en spille er tilbage 1.a Spillet stopper, og matadoren/vinderen er vundet 8.2 Hvis mere end 1 spiller er tilbage 2.b Spillet fortsætter

2.3 Features

2.3.1 Feature liste

Nedenfor er de overordnede features af programmet kort beskrevet. Vi vil senere benytte disse features til planlægning. Alle features er blevet udarbejdet ved at gennemgå alle reglerne for spillet og gennemgå vores use-case der beskriver, hvordan et helt spil ville udfolde sig. Dermed får vi de forskellige scenarier for spillet

Features id.	Titel	Beskrivelse	Krav dækning
F01	Kast terning	Spiller benytter sig af en GUI knap, hvorefter tilfældige terningekast præsenteres i GUI'et. Spilleren rykkes summen af slaget på spillepladen.	RQF03, 'RQF04, RQF38,RQF39, RQF42
F02	Køb grund	Spilleren har muligheden for at købe grunden de lander på	RQF08, RQF09
F03	Fængsel mekanik	I tilfælde af fængsling har spilleren tre muligheder for at komme ud. spilleren kan forsøge at slå dobbelt med terningerne. Betale kaution eller bruge et benådnings chancekort og komme ud gratis.	RQF39, RQF40, RQF41, RQF46
F04	Pansæt grund	Den nuværende spiller kan pantsætte en grund for at modtage penge tilsvarende til halvdelen af grundens pris. Man kan ikke modtage leje fra en grund der er pantsat	RQF32, RQF33, RQF34, RQF35, RQF36,
F05	Sælge grund/hus	Spilleren kan sælge en grund til en anden spiller	RQF18, RQF19, RQF29,
F06	Chancekort	Spilleren kan trække et chancekort med forskellige effekter som påvirker spilleren enten negativt eller positivt.	RQF47,
F07	Passer start	Spilleren modtager 200,- når passerer start	RQF07
F08	Gå fallit	Spilleren udgår af spillet	RQF48
F09	Spillet slut	Den sidste spiller på brættet har vundet	RQF50
F10	Helle	Det er et fristed indtil næste tur	RQF45
F11	Køb hus/hotel	Spilleren har mulighed for at købe et hus, i slutningen af deres tur hvis de ejer alle grundene af den bestemte farve	RQF10, RQF11, RQF12, RQF13, RQF14, RQF15, RQF16, RQF17, RQF28

F12	Auktion	Hvis en spiller ikke vil købe grunden de er landet på, går grunden på auktion, hvor spillerne byder på grunden, hvor det højeste bud vinder grunden for den givne pris.	RQF20, RQF21, RQF22, RQF23, RQF24, RQF25, RQF26,
F13	Initialiser spil	Opstart af spillets helt generelle rammer	RQF01, RQF02, RQF05, RQF06, RQF27, RQF37, RQF43, RQF44, RQF49
F14	Forhandle	Mulighed for at spiller kan internt forhandle grunde mellem hinanden	RQF30, RQF31

2.3.2 Prioritetsliste

Nedenfor ses en oversigt over features indsat i en prioriteringsmatrix, hvor deres placering afgøres af sværhedsgraden af opgaven ved implementering og betydningen for det færdige spil. Vurderingen er lavet med udgangspunkt i tidligere erfaringer og i forhold til hvor essentielle de er for at spillet kan fungere.

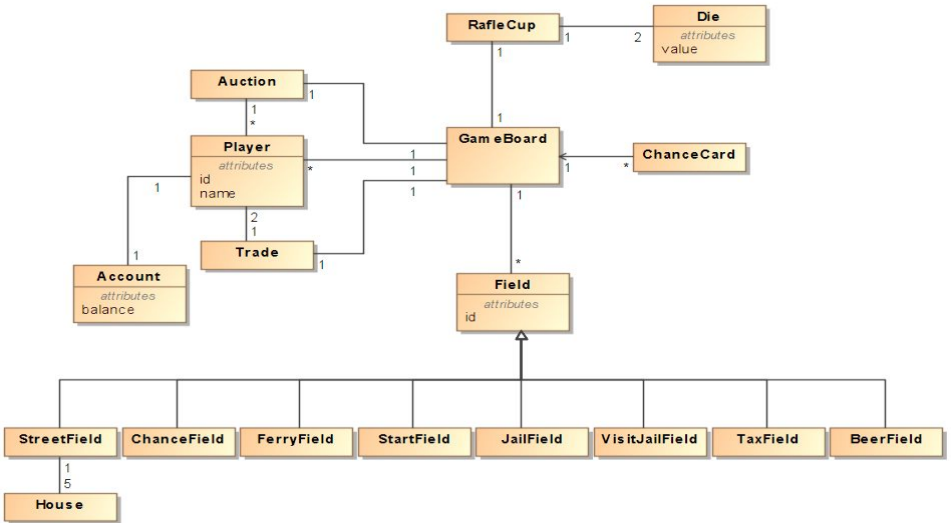
	Overflødig	Brugbar	Vigtig	Meget vigtig	Essentiel
Let	F10			F07	F01
Okay			F03	F09	F02, F13
Normal		F04		F08	
Udfordrende		F12	F11, F06,F14		
Svær		F05			

Tabel: 2.3.2.1 Prioritetsmatrix

Ud fra tabellen laver vi følgende prioriteringsliste af vores arbejde:

1. F02
2. F13
3. F01
4. F08
5. F06
6. F011
7. F14
8. F09
9. F07
10. F03
11. F05
12. F12
13. F04
14. F10

2.4 Domæne diagram

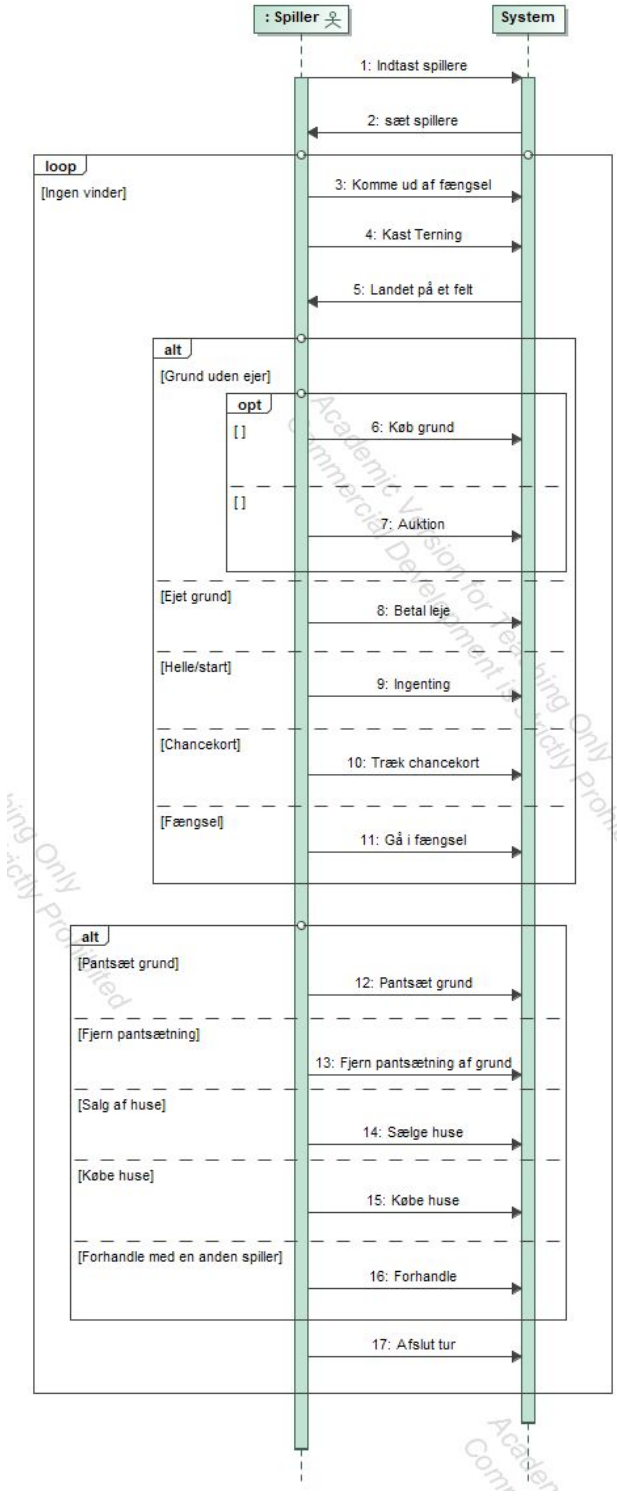


Figur: 2.4.1 Domæne Diagram

Det allerførste vi lavede efter vi at have dannet gruppe var at få et overblik over hvordan koden skulle være og de forskellige klasser ved at lave en skitse af vores domæne diagram. Dog gik vi meget hurtigt videre til klassediagrammet da det var en af kravene for den første deadline 3 dage inde i projektet.

Vores domæne diagram minder meget om det tidligere projekt men vores endelige klassediagram og struktur for programmet er en del anderledes end domæne diagrammet. Igennem projektet fandt vi mange ting vi gerne ville have anderledes end vores oprindelige design, enten fordi vi fandt det mere intuitivt eller fordi det var nemmere at arbejde med. Dermed har vi lagt det meste af vores fokus i klassediagrammet.

2.5 System sekvensdiagram



Figur: 2.5.1 Systemsekvensdiagram

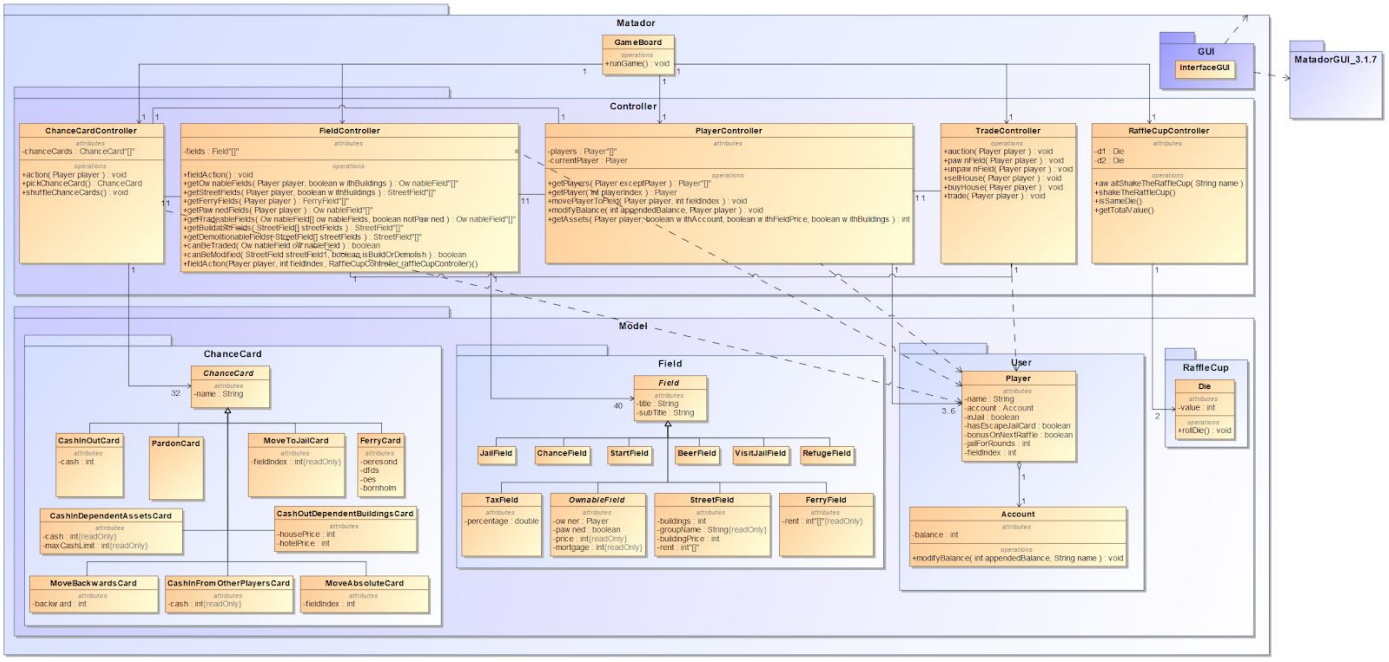
Projektet besidder kun en use case “Spil Matador”, herved dækker system sekvensdiagrammet hele spillet og dets funktioner overordnet. Her ses spil oplevelsen, som en black-box. Spillet igangsættes ved at en spilleren indtaster antal spiller og tilhørende navne. Hvorefter systemet sætter spillere på spillepladen.

Spilleren begynder med at kaste terningerne, hvor den samlede værdi af de to terninger bestemmer, hvor langt spilleren rykkes på pladen. Alt efter hvilket slags felt spilleren lander på, fremtræder forskellige valgmuligheder. Herunder er grunde som enten er ejede eller ikke. Helle og start som ikke gør noget. Chancekort som trækker et tilfældigt kort. Og fængsel som sender spilleren direkte i fængsel.

Efter en spiller har slået med terningerne har han også en række muligheder i forhold til de grunde han ejer. Han kan pantsætte grunde eller fjerne pantsætning på grunde. han har også mulighed for at købe huse eller fjerne dem på grunde som tillader at man kan bygge huse på dem. Og han kan også forhandle med andre spillere om de grunde han ejer. Til sidst skal spilleren altid slutte sin tur og loopet kører indtil kun en spiller er tilbage.

3. Design

3.1 Klassediagram



Figur: 3.1.1 Klassediagram

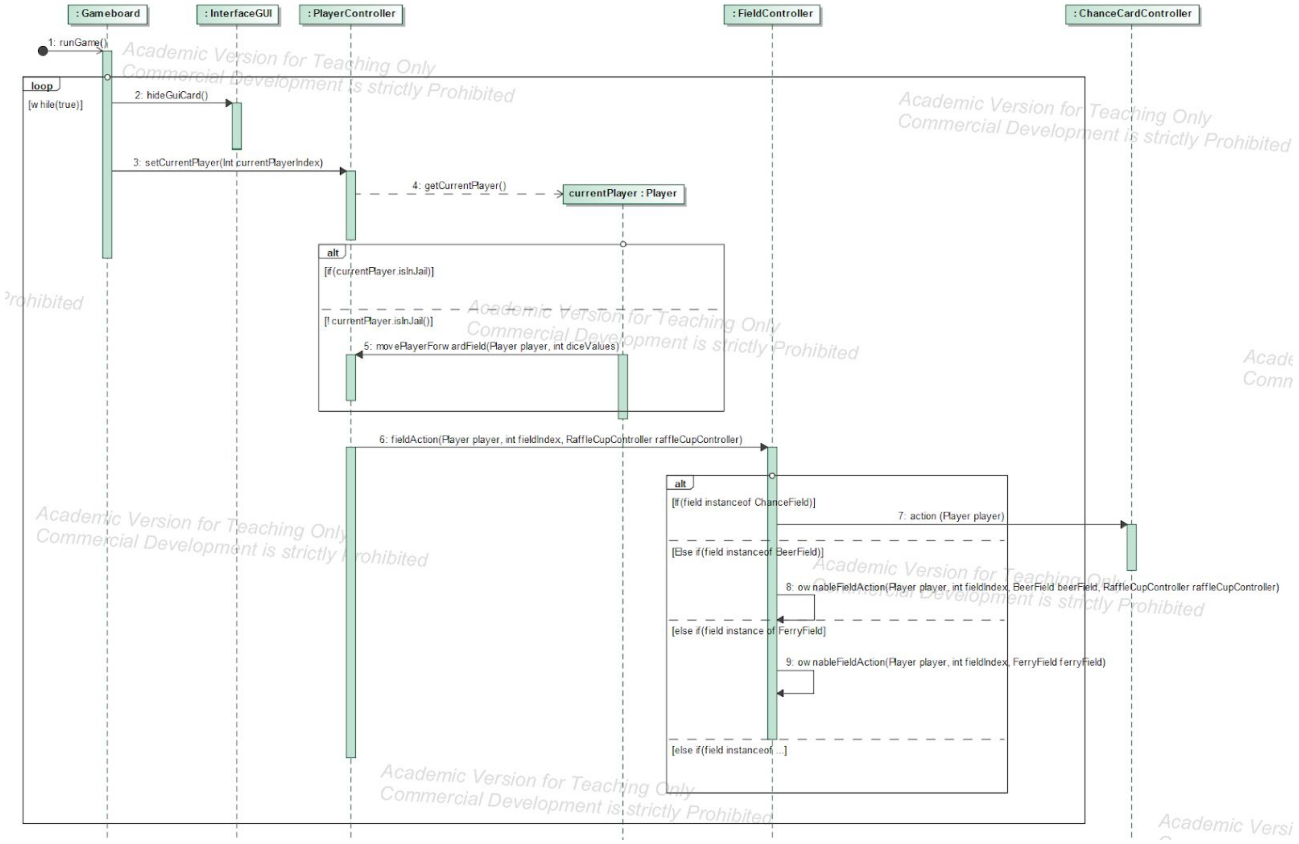
Vores klassediagram er en repræsentation af vores endelige produkts struktur og de forskellige attributter og funktioner der hertil indgår under hver klasse. Vi har ændret diagrammet meget i forhold til den første deadline *(M1)* for projektet. Dette er grundet vi ikke vidste præcist hvordan projektet skulle se ud samt, hvordan programmet skulle fungere og fandt nogle forskellige steder, hvor vi ville ændre på måden projektet var sat op. Hovedfokuset har været at følge MVC og inkorporere og følge så mange af GRASP mønstrene som muligt. Hertil har vi selvfølgelig også sat meget fokus på at selve programmet skulle virke hvilket også har været med til hvorfor der er sket ændringer i løbet af projektet. En større udgave kan ses under "*Bilag 4"*

Vi har udeladt mange af de forskellige get/set metoder og mindre interessante attributter fra klassediagrammet for at det skulle være mere læseligt og nemmere at forstå. Det er fordi at mange af de metoder og attributter er forventede og ikke tilføjer noget til programmet som ikke allerede kan ses på de relationer klasserne har. Controllerne har relationer til hver deres relevante modeller samt til de andre controllere for at kunne kommunikere med hinanden og udføre de ønskede metoder.

Vi kan se under ChanceCard- og Field-packagen at de begge har en abstrakt klasse som de andre klasser så kan arve fra. Hertil har field en ekstra abstrakt klasse "*Ownablefield*" som de tre typer af felter der kan være ejet arver fra for at gøre det nemmere at sætte ejere på de forskellige grunde og tildele grunde til de forskellige spillere. User-packagen indeholder player og account og disse indeholder spillerens balance, fieldIndex samt om spilleren er i fængsel. Packagen raffle-cup står for vores terninger.

Gameboard er der hvor selve spillet kører og løkken for om spillet stadig er i gang kan findes. Gameboard har hertil kun relationer til de forskellige controllere som står for logikken og for at interagere med de forskellige modeller. Det er opsat arkitektonisk så alle controllere kun har relationer med præcist de modeller og controllere som er nødvendigt. Og dette er gjort for at gøre programmet og arkitekturen så god som muligt.

3.2 Sekvensdiagram

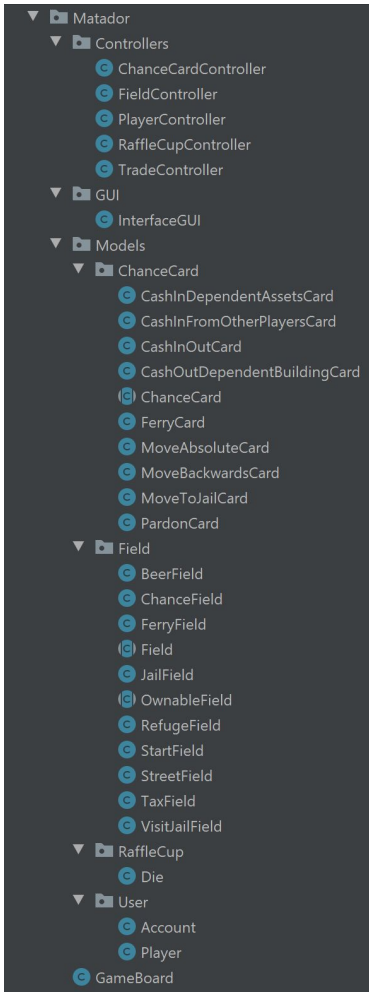


Figur: 3.2.1 Sekvensdiagram

Sekvensdiagrammet viser systemets virkemåde på et toplevel. Hele processen påbegyndes af funktionen runGame(), der kaldes på et gameboard objekt. GameBoardets konstruktør kaldes ved objekt dannelsen i en Main klasse. Gameboard konstruktøren står for opsætte GUI-boardet, derudover kalder den alle controllernes konstruktører, som står for at oprette felter, spiller, trade og chancekort. Herefter påbegyndes et while(true) loop, som kører igennem en spillers tur. Loopet begynder med at tjekke om spilleren befinder sig i fængsel. Hvis spilleren ikke befinder sig i fængsel flyttes spilleren ved movePlayerForwardField, hvorefter fieldAction, tjekker hvilket felt spilleren er landet på. Alt efter hvilket slags felt, kaldes forskellige metoder. Større visning af sekvensdiagrammet ses i *"Bilag 5"*

4. Implementering

4.1 Koden



Projektets struktur
Vi har fulgt princippet omkring MVC, Model View Controller, hvor vores modeller befinder sig i packagen Models, Controllere i mappen Controllers og View ligger som en del af MatadorGUI:3.1.7 der bliver tilgået fra den statiske klasse InterfaceGUI. Man kan betragte InterfaceGUI som en snitflade, bindeleddet, mellem logik og brugerflade. Det smarte ved InterfaceGUI klassen er sågar at vi altid vil kunne bruge GUI'en fra vores kode. Så fx. når en model bliver opdateret kan vi "notify" GUI'en om at en model er blevet opdateret. Eftersom der også ofte bliver skrevet en besked til brugeren er det nemt at bruge "InterfaceGUI.showMessage()" for at vise en besked og vi slipper altså derfor for at sende et GUI objekt med rundt til alle steder hvor GUI'en skal opdateres.

Abstrakte klasser
Før vi begyndte implementeringen havde vi analyseret nogle klasser vi mente skulle være abstrakte. Dette gælder Field, OwnableField og ChanceCard klasserne. Disse abstrakte klasser repræsenterer blot hvad deres arvefølge har til fælles. Fields vil altid have et navn, OwnnableFields vil altid have en lejepris og ChanceCards vil også altid have et navn. Disse klasser kan heller ikke instantieres alene men skal instantieres via de klasser som arver derfra. På grund af arv er vi dog nødsaget til at lave nogle tjek via instanceof. For eksempel hvis man lander på et felt så ved programmet kun at du er landet på et Field og vi er derfor nødsaget til at tjekke hvilken konkret felt det er.

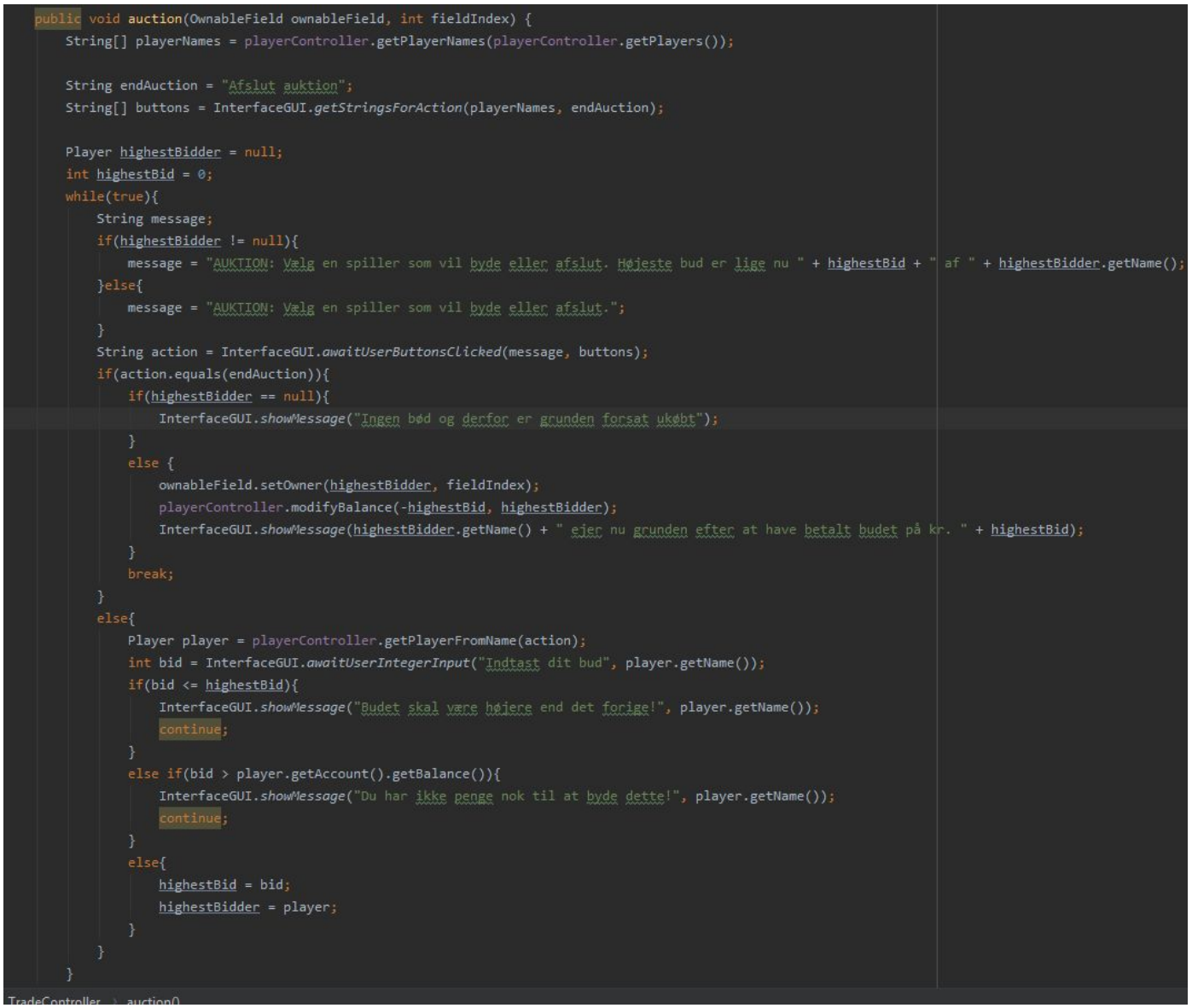
Når vi ved hvilken konkret felt spilleren er landet på kan vi foretage en aktion på det pågældende felt.

```
public void fieldAction(Player player, int fieldIndex, RaffleCupController raffleCupController){
    Field field = this.getFields()[fieldIndex];

    if(field instanceof BeerField){...}
    else if(field instanceof ChanceField){...}
    else if(field instanceof FerryField){...}
    else if(field instanceof JailField){...}
    else if(field instanceof RefugeField){
        //RefugeField refugeField = (RefugeField) field; //Der sker ikke en dyt her "Pause felt"
    }
    else if(field instanceof StartField){
        //StartField startField = (StartField) field; //Der sker heller ikke en dyt her
    }
    else if(field instanceof StreetField){...}
    else if(field instanceof TaxField){...}
    else if(field instanceof VisitJailField){
        //VisitJailField visitJailField = (VisitJailField) field; //Sker heller ikke en dyt her
    }
}
```

4.2.1 Interessant kode eksempel

Auction metoden:
Hvis en spiller lander på en grund, der ikke er ejet af en anden spiller, og vælger ikke at købe den pågældende grund så hedder reglen at grunden skal gå på auktion. Når en grund ryger på auktion er det den følgende metode i TradeControlleren der bliver kaldt.



```
public void auction(OwnableField ownableField, int fieldIndex) {
    String[] playerNames = playerController.getPlayerNames(playerController.getPlayers());

    String endAuction = "Afslut auktion";
    String[] buttons = InterfaceGUI.getStringsForAction(playerNames, endAuction);

    Player highestBidder = null;
    int highestBid = 0;
    while(true){
        String message;
        if(highestBidder != null){
            message = "AUKTION: Vælg en spiller som vil byde eller afslut. Højeste bud er lige nu " + highestBid + " af " + highestBidder.getName();
        }else{
            message = "AUKTION: Vælg en spiller som vil byde eller afslut.";
        }
        String action = InterfaceGUI.awaitUserButtonsClicked(message, buttons);
        if(action.equals(endAuction)){
            if(highestBidder == null){
                InterfaceGUI.showMessage("Ingen bød og derfor er grunden forsat ukøbt");
            }
            else {
                ownableField.setOwner(highestBidder, fieldIndex);
                playerController.modifyBalance(-highestBid, highestBidder);
                InterfaceGUI.showMessage(highestBidder.getName() + " ejer nu grunden efter at have betalt budet på kr. " + highestBid);
            }
            break;
        }
        else{
            Player player = playerController.getPlayerFromName(action);
            int bid = InterfaceGUI.awaitUserIntegerInput("Indtast dit bud", player.getName());
            if(bid <= highestBid){
                InterfaceGUI.showMessage("Budet skal være højere end det forige!", player.getName());
                continue;
            }
            else if(bid > player.getAccount().getBalance()){
                InterfaceGUI.showMessage("Du har ikke penge nok til at byde dette!", player.getName());
                continue;
            }
            else{
                highestBid = bid;
                highestBidder = player;
            }
        }
    }
```

TradeController › auction()

Det første der sker er, at vi henter alle spiller objekter for at hente alle spillernavne i et String array. Når spiller navnene er hentet forberedes de valgmuligheder man skal have i forbindelse med auktionen. Her skal man bare kunne vælge en spiller der vil byde eller afslutte auktionen.

Hvis man vælger "Afslut auktionen" så vil 1) hvis nogen har budt så vil den højeste byder betale og få ejerskab på feltet, 2) hvis ingen byder så vil intet opdateres og turen går videre. Bemærk at "break;" ligger her for at afbryde while løkken.

Hvis man klikker på en af spillerne så vil spiller objektet blive hentet ud fra det konkrete navn, navnet skal være unikt og derfor kan man hente spillerens unikke objekt. Den pågældende spiller vil få besked om at indtaste deres bud og det bliver så kun godkendt, hvis det er højere end forrige bud og så længe spilleren selv har penge til sit eget bud.

getStreetFields

Et andet interessant kode eksempel er metoden getStreetFields i fieldControlleren.

```
public StreetField[] getStreetFields(){...}
public StreetField[] getStreetFields(Player player){...}
public StreetField[] getStreetFields(Player player, boolean withBuildings){...}
```

Denne metode har to overloads som gør det muligt at få diverse StreetFields ud fra nogle forskellige præmisser.

1) getStreetFields() returnerer alle StreetFields der eksisterer i fields listen.
2) getStreetFields(Player player) returnerer alle StreetFields som er ejet af en konkret spiller
3) getStreetFields(Player player, boolean withBuildings) returnerer alle StreetField som er ejet af en konkret spiller og withBuilding afgør om det kun skal være StreetFields med bygninger eller om det kun skal være StreetFields uden bygninger.

1) getStreetFields()

```
public StreetField[] getStreetFields(){
    int newSize = 0;
    for(Field field : fields){
        if(field instanceof StreetField){
            newSize++;
        }
    }
    StreetField[] result = new StreetField[newSize];
    int i = 0;
    for(Field field : fields){
        if(field instanceof StreetField){
            result[i] = (StreetField) field;
            i++;
        }
    }
    return result;
}
```

Da vi benytter os af Arrays skal vi først finde og sætte størrelsen på arrayet for bagefter at pege på objektet på det pågældende felt. Det vigtige tjek her er "if(field instanceof StreetField)"

2) getStreetFields(Player player)

```
public StreetField[] getStreetFields(Player player){
    int newSize = 0;
    StreetField[] streetFields = getStreetFields();
    for(StreetField streetField : streetFields){
        if(streetField.getOwner() == player){
            newSize++;
        }
    }
    StreetField[] result = new StreetField[newSize];
    int i = 0;
    for(StreetField streetField : streetFields){
        if(streetField.getOwner() == player){
            result[i] = streetField;
            i++;
        }
    }
    return result;
}
```

Der gælder det samme her som i 1) at arrayets størrelse afgøres først for efterfølgende at blive peget på en værdi. Bemærk også her at den benytter sig af getStreetFields(), altså den uden parametre, for at finde udelukkende StreetFields. Det vigtige tjek her er "if(streetField.getOwner() == player)".

3) getStreetFields(Player player, boolean withBuildings)

```
public StreetField[] getStreetFields(Player player, boolean withBuildings){
    int newSize = 0;
    StreetField[] streetFields = getStreetFields(player);
    for(StreetField streetField : streetFields){
        if(withBuildings && streetField.getBuildings() > 0){
            newSize++;
        }
        else if(!withBuildings && streetField.getBuildings() == 0){
            newSize++;
        }
    }
    StreetField[] result = new StreetField[newSize];
    int i = 0;
    for(StreetField streetField : streetFields){
        if(withBuildings && streetField.getBuildings() > 0){
            result[i] = streetField;
            i++;
        }
        else if(!withBuildings && streetField.getBuildings() == 0){
            result[i] = streetField;
            i++;
        }
    }
    return result;
}
```

Igen gælder det samme med at arrayet størrelse skal identificeres og sættes først for efterfølgende at få udfyldt værdier på de pågældende indexes.
Denne metode benytter sig af altså først af getStreetFields(Player player) for at hente de felter som spilleren ejer. Når spilleren har de felter han ved han ejer så skal der tjekkes om hvorvidt der er bygninger på felterne eller ej. Hvis withBuildings er *true* er det altså kun de grunde der har over 0 bygninger der bliver returneret og omvendt hvis withBuildings er *false*, er det kun de grunde der har 0 bygninger der bliver returneret.

4.2 Grasp

GRASP som står for General Responsibility Assignment Software Patterns bruges til objekt orienteret design til at allokere ansvar til klasser.

Creator
En Creator angiver, hvem der skal være ansvarlig for skabelsen af en ny instans af en klasse. I vores tilfælde er det GameBoardet der er ansvarlig for at oprette de forskellige controllers. Disse controllers bliver så sendt videre så de kan bruges af de forskellige controllers, men ansvaret for at oprette disse controllers ligger altså hos GameBoard.java.

Information Expert
Information Expert er et princip der fortæller noget om en klasse har præcis det ansvar den skal have. I vores eksempel kan vi forklare det ud fra TradeController. Denne controller er expert i alt hvad der drejer sig om handel mellem bank og spiller og mellem spiller til spiller. TradeControlleren har altså kun de metoder som hører inden for det ovenstående.

Controller
Controlleren tildeler ansvar til håndtering af inputs fra en brugergrænseflade. Da UI ligger separat og da UI indeholder en model for sig, har vi ikke kunne knytte vores controllers direkte til UI ændringer. I stedet bliver de navigeret videre fra, for eksempel, GameBoard til TradeControlleren, hvor TradeControlleren så har ansvaret for at hente og bruge de oplysninger den har brug for.

Polymorphism
Polymorfi omhandler arv og forskellige implementationer af en metode. Vi har tre abstrakte klasser som kan arves og benyttes til polymorfi. Disse klasser er ChanceCard, Field og OwnableField. StreetField og FerryField arver fra OwnableField. StreetField sætter sin constructor både i buildingPrice, rent og groupName, hvor FerryField ikke sætter noget men blot kalder super() som StreetField også gør. De arver altså begge OwnableField og kalder constructoren i OwnableField via super(), men de udfører stadig noget kode i deres egen constructor for sig.

Low Coupling
Her er der fokus på lav afhængighed mellem klasser. Der er en klar lav kobling til de modeller som er brugt i programmet. PlayerController har kun kobling til andre controllers og til Player modellen. FieldController ligeledes bare til Field modellen i stedet og ChanceCardControlleren har også kun kendskab til de andre controllers og til ChanceCard modellen.

High Cohesion
High Cohesion omhandler tydelig ansvarsfordeling. Igen viser controllerne dette. PlayerController omhandler spilleren, FieldController omhandler felterne og ChanceCardControlleren omhandler chancekort. Dette er selvfølgelig også gældende for vores modeller.

5. Test

5.1 JUnit tests

5.1.1 ChanceCardControllerTest

Den første unit test blev skrevet tidligt i forløbet. Vi har skrevet test af metoderne shuffleDeck() og pickCard() i ChanceCardController-klassen. Testene afgør for det første, om alle kort fra det ublandede deck er repræsenteret i det nu blandede deck. Derefter testes om et gennemløb af hele det blandede dæk, giver et udfald af hvert af alle de mulige kort.

Testmetoden i de to tilfælde ligner hinanden. I shuffleCards()-testen sammenlignes dækket før og efter blanding. Vi anvender et assertTrue()-statement, som over et itereret forEach-loop over elementer i det oprindelige array afgør, om hvert element er repræsenteret i det blandede array. Testen går igennem.

```
37      @org.junit.Test
38      public void shuffleCards() {
39          ChanceCardController chanceCards = new ChanceCardController();
40          ChanceCard[] copiedDeck = Arrays.copyOf(chanceCards.getDeck(), chanceCards.getDeck().length);
41          chanceCards.shuffleCards();
42          ChanceCard[] shuffledDeck = chanceCards.getDeck();
43          assertEquals(copiedDeck.length, shuffledDeck.length);
44          for (ChanceCard card : copiedDeck) {
45              assertTrue( condition: indexOfChanceCardArray(shuffledDeck, card) != -1);
46          }
47      }
```

Testen til pickCard-metoden er af lignende opbygning. Her tester vi ved hjælp af et array af booleans om alle kort er repræsenteret i et gennemløb af hele dækket.

```
49      @org.junit.Test
50      public void pickCard() {
51          ChanceCardController chanceCards = new ChanceCardController();
52          ChanceCard[] copiedDeck = Arrays.copyOf(chanceCards.getDeck(), chanceCards.getDeck().length);
53          Boolean[] isTaken = new Boolean[copiedDeck.length];
54          initBooleanArray(isTaken, trueFalse: false);
55          for (int index = 0; index < copiedDeck.length; index++) {
56              ChanceCard card = chanceCards.pickCard();
57              isTaken[indexOfChanceCardArray(copiedDeck, card)] = true;
58          }
59          assertTrue(allTrueFalseArray(isTaken, trueFalse: true));
60      }
```

Arrays.copyOf kopierer et array og ændrer størrelsen. Vi kan altså derfor oprette et kopi af et dæk med en ny størrelsen uden at referencerne er med.

5.2 Brugertest

I brugertesten testes programmet ud fra en bruger der ikke har nogen kendskab til koden. Her er formålet, at få en fundamental viden om hvad brugerens tanker om programmet er, hvad der virker og ikke mindst, hvor der i fremtiden skal optimeres så programmet fungerer bedst muligt. Vi har derudover også haft fokus på at teste brugervenligheden af produktet, for at sørge for at programmet er så intuitiv som muligt. Der har dog været nogle udfordringer med brugertest, da vi ikke har haft en platform til at uddelegere vores test version. Vi har derfor ikke været i stand til at lave en større åben beta og har derfor lavet en mindre mere kvalitativ brugertest. Brugertesten fungerer mere som en form for interview, hvor testeren først kører programmet igennem uden hjælp, hvorefter der besvares nogle spørgsmål om, hvad de oplevede om programmet.

Spørgsmål

Spørgsmålene er lavet sådan, at vi får en bred ide om, hvad brugeren synes om programmet, samt at brugeren kan komme med forslag til hvad der kunne gøres bedre. Dette kunne være ting udviklerne måske ikke har tænkt på, eller ikke troede var vigtigt før brugeren kommenterede det.

- Hvis du oplevede tvivl på et tidspunkt i gennemgangen af spillet, hvad var årsagen?
- Er opsætningen til at forstå?
- Hvad er det sværeste ved brugen af programmet?
- Var der noget der overraskede?
- Hvad kunne gøres bedre/hvad manglede der?
- Ville du bruge dette program?

Opsætning

Vi har udviklet programmet og dets brugergrænseflade, så det er så intuitivt som muligt. Dette er blevet gjort ved, at når systemet udfører noget, vil der komme en beskrivelse på brættet. Chancekort vil blive vist i midten, og informationer om hvad der bliver udført ovenover. Der er også sat knapper op der viser, hvilke muligheder man har på et givent tidspunkt, f.eks. vil der kun være en knap, hvis det eneste man kan er at slå terningerne. Efter man har slået, vil der så komme flere knapper, der viser de nye muligheder, hvor hver knap har hver deres funktion f.eks. som afslut tur, køb af huse, pantsætning osv.
Der er også blevet taget højde for grafisk information om penge og ejerskab af grundene. Da de fleste godt kan lide grafisk tilbagemelding, er der blevet gjort, at man altid kan se alle spillernes penge nede i højre hjørne, samt hver grund der ejes af en spiller, vil have ejerens farve som ramme.

Kommentarer fra tester

Spørgsmål:	***Svar:***
Hvis du oplevede tvivl på et tidspunkt i gennemgangen af spillet, hvad var årsagen?	Der var som sådan ikke et tidspunkt, hvor jeg var i tvivl om noget. Jeg kunne finde ud af opsætningen og bruge programmet relativt hurtigt og uden problemer.
Er opsætningen til at forstå?	Opsætningen minder meget om det traditionelle Matador brætspil, så det er nemt at sætte sig ind i, hvad man skal. Derudover var alle mulighederne og knapperne sat overskueligt op såsom når man skal vælge en grund, var der en nem overskuelig dropdown liste af de ejendomme man ejer.
Hvad er det sværeste ved brugen af programmet?	Der var ikke rigtig noget, der var svært, men hvis jeg skulle sige noget, ville det være at se, hvilke af de andres spilleres grunde der var pantsat. I det nuværende program er man nødt til at klikke på en grund for at se om den er pantsat når det ikke er ens tur.

Var der noget der overraskede?	Der var sådan set, alt det man ville forvente i et Matador spil, så der var ikke noget der overraskede.
Hvad kunne gøres bedre/hvad manglede der?	Det med at programmet kun kan køres på en computer, og at man hele tiden skulle bytte plads kunne være bedre. Det ville være bedre hvis man også havde muligheden for at spille på to forskellige computere. For at gøre spillet mere interessant kunne man tilføje nogle flere grafiske animationer.
Ville du bruge dette program?	Som sagt i spørgsmålet ovenover, kunne nogle flere grafiske animationer gøre det sjovere at spille spillet, hvis det var kunne jeg godt forestille mig selv at spille spillet på et andet tidspunkt.
Ekstra kommentar	Overordnet synes jeg at programmet minder om Matador, og jeg stødte ikke på nogen fejl.

Vores kommentar på brugertest
Ud fra kommentarerne fra vores testbruger, kan vi se at vores program er intuitivt og har et simpelt UI, som brugeren nemt kan finde rundt i. Udover det har vi også fået kendskab til områder der kan gøres bedre i fremtiden, vel at mærke at de påpegede ting ikke har været vores fokusområde i dette forløb. Så vi har ud fra brugertesten fået bekræftet at vores program er intuitivt og nemt at håndtere og fundet steder der kan forbedres i fremtiden.

5.3 Databar test

Eftersom en af kravspecifikationerne er, at programmet skal kunne køre på en computerne på DTU's databarer, er det testet hvorvidt dette er opfyldt. For at tjekke for det, er følgende steps udført:

1. Installer programmet gennem Github eller ved brug af USB stik.
2. Eksekver programmet
3. Spillet startes op
4. Programmet køres igennem, uden at programmet får fejl eller går ned

Når alle disse test er gennemført på en af DTU's databar maskiner, kan der konkluderes at programmet udfører kravet om, at programmet skal kunne køres på DTU's databarer. Da vi har udført alle test, kan vi konkludere at databar testen er gennemført og bestået.

6. Projektplanlægning

6.1 Tidsplan

Tidsplanen for vores produkt og features er baseret på vores prioriteringsliste. Hvor de mest essentielle features for Matador-oplevelsen implementeres først. Derudover er vores rapport elementer er baseret på de mellemliggende deadlines M1, M2 og final.

Aktivitet	Underaktivitet	6/1	7/1	8/1	9/1	10/1	11/1	12/1	13/1	14/1	15/1	16/1	17/1	18/1	19/1	20/1
Rapport																Aflevering
Indledning	Abstract															
Krav og analyse	Kravliste															
	Use Case diagram															
	Domæne diagram															
	Systemsekvensdiagram															
Design	Designklassediagram															
	Sekvensdiagram															
Implementering																
Test	Brugertest															
	Junit test															
Konklusion																
Produkt og Features																
	Konfiguration															
Features	Kast Terning															
	Køb grund															
	Køb hus/hotel															
	Fængsel mekanik															
	Pansæt Grund															
	Forhandle indbyrdes															
	Chancekort															
	Passer start															
	Gå fallit															
	Spillet slut															
	Helle															
	Auktion															
	Sælg grund/chancekort															

6.2 Projektforløb

Projektet begyndte med sammenkoblingen af to grupper. Derfor begyndte vi med gennemgang af de to gruppers tidligere opgaver, med fokus på vellykkede elementer. Derudover begyndte vi på produktet fra nyt. Det sikrede begge grupper var godt indforstået med koden og dens indhold. Tidligt i projektet skulle vi møde en deadline M1, hvilket sikrede overvejelse af projektets omfang og dets elementer. Til kodningen havde den ene gruppe erfaret succes, ved at arbejde to og to. Denne sparrings mulighed, viste sig også succesfuldt i dette projekt. Vi opdelte features iblandt os, hvorefter de blev udviklet. Feature opdeling, var en effektiv måde at uddele opgaver, og gjorde det nemt at følge fremgangen i projektet.

Selve projektplanlægningen kontra de dage vi arbejdede stemte ikke helt overens, da vi var meget effektive den første uge og blev hurtigt færdige med projektet. Det vil sige at vores tidsplan ikke blev brugt så meget til hvad der skulle laves de individuelle dage men blev brugt mere til hvad der var vigtigst og hvilken rækkefølge tingene skulle laves. Kort beskrevet, de vigtigste elementer til projektet blev lavet først ud fra tidsplanen, som gav et overblik over de features som var højest prioriteret.

Rapportskrivningen benyttede vi samme uddeling af elementer, hvorefter vi gennemgik alle elementer sammen. Sammenkoblingen af de to grupper viste sig være fordelagtig, grundet nye ideer, evner og perspektiv.

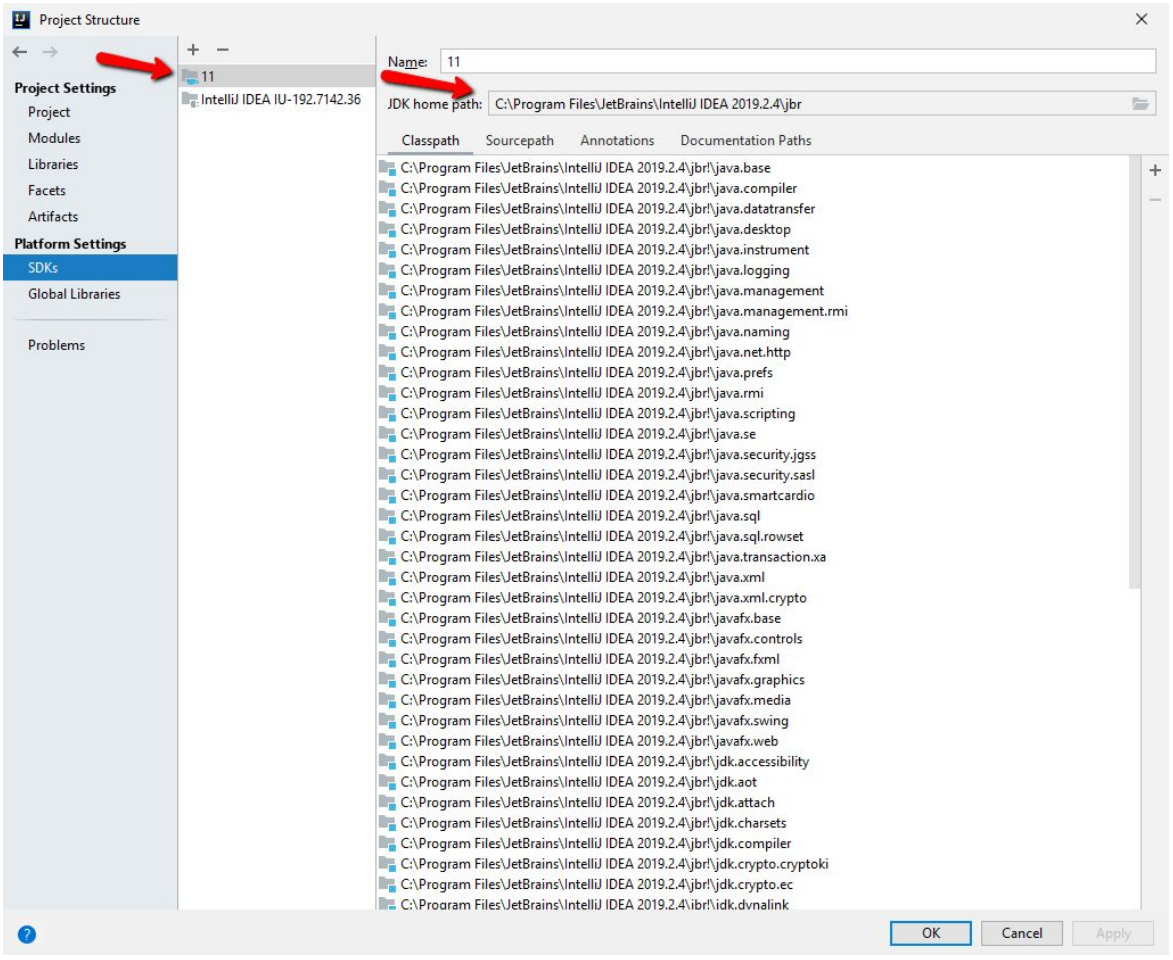
7. Konfigurationsstyring

7.1 Udviklingsmiljø

Før at hente og køre projektet skal du bruge.

- Intellij IDEA 2019.2.4 (Ultimate edition)
- Java SDK 11.0.0
- Git

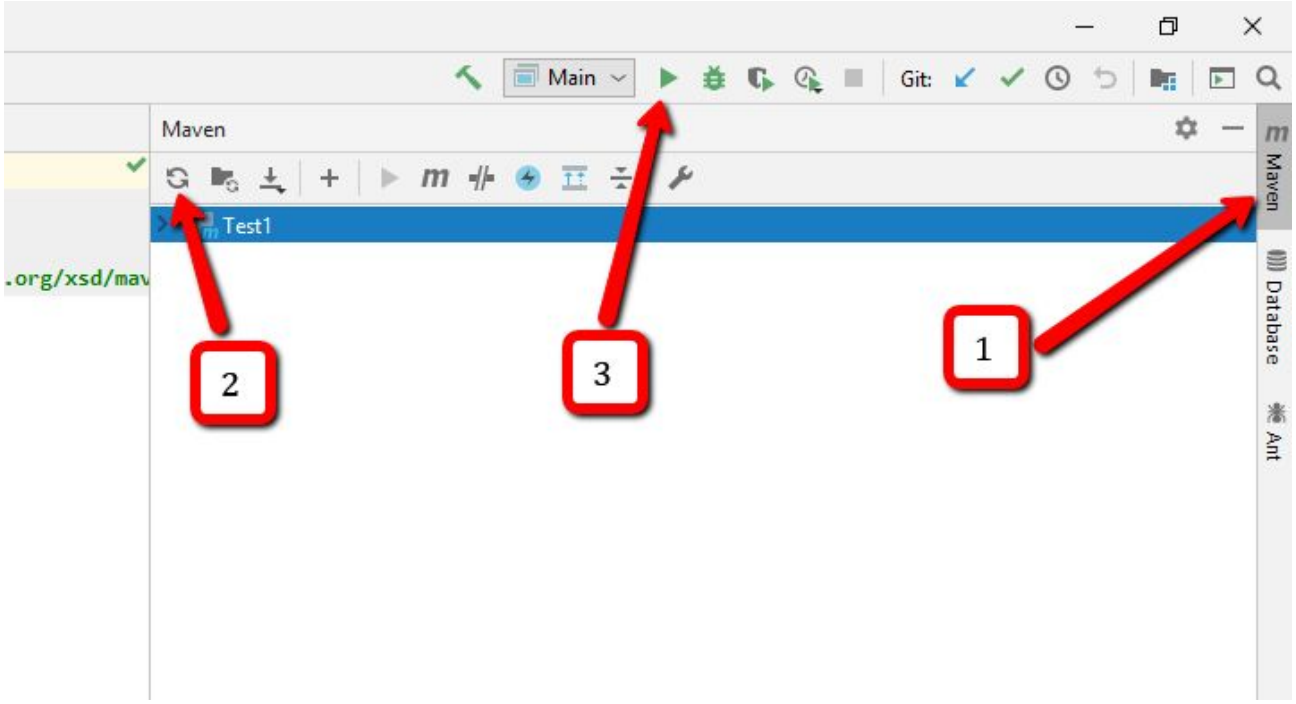
På Intellijs startside vælges “Check out from Version Control” -> “Git” og skriv https://github.com/creoludifico/15_final i URL og hent. Når programmet er åben så vælges File -> Project Structure. I menuen vælges SDKs.



Det er vigtigt at Name er præcist 11 og det peger på en SDK der er version 11.

Klik Ok.

Før du kører programmet skal pom.xml hente det eksterne bibliotek matadorgui:3.1.7. Dette hentes ved hjælp af Maven. Maven er et framework til at hente eksterne biblioteker og det kan også bruges til at udgive egne biblioteker.



Klik på Maven oppe i højre side af Intellij (1) og vælg derefter "Sync" (2) vist på billedet. Nu burde matadorgui:3.1.7 hentes til projektet og efterfølgende kan det køres ved at klikke på den grønne pil (3).

7.2 Versionering

Til versionsstyring har vi gjort brug af Git, så det har været muligt at kunne sidde flere personer at arbejde på koden, på samme tid. Vi har i dette projekt forsøgt at holde udvikling i en udviklings-branch, så der ikke blev kodet direkte i master-branchen, ved at have en dev branch (development). Ved commit-meddelelser har vi prøvet at skrive en sigende beskrivelse af den specifikke handling, for det pågældende commit, så man nemmere kan få et overblik over hvilken feature/handling der er tale om.

Man kan hente programmet via intellij på følgende måder
Github link: https://github.com/creoludifico/15_final

1) Det kan hentes på Intellij startside ved at klikke på Check out from Version Control -> Git, hvorved git repository link indsættes
2) Hvis man har et andet projekt kørende kan man klikke oppe i top menuen VCS -> Check out from Version Control -> Git, hvorved git repository link indsættes
3) Det kan hentes ved at åbne en command prompt, cmd, og skrive git clone https://USERNAME:PASSWORD@github.com/creoludifico/15_final

Erstat USERNAME med brugernavn og PASSWORD med kodeord.

4) Det kan også hentes som zip fil, ved at gå ind på Github repository og klikke på download -> Download ZIP

1,2,3 kræver at Git er installeret.

Under udviklingen af programmet har vi benyttet version styring, herunder Git, til at udvikle produktet. Vores repository ligger som nævnt tidligere her https://github.com/creoludifico/15_final.

8. Konklusion

Under arbejdet af denne rapport har vi gjort brug af metoder vi har lært i løbet af 1. semester. Dette inkluderer UML, Versionsstyring & testmetoder og indledende programmering. Vi har haft et godt samarbejde hvor vi hver dag har diskuteret og fundet frem til gode løsninger både ved hjælp af kommunikation og brug af tavle til at tegne de ideer man har for hinanden.

I løbet af projektet har der også været en del udfordringer med MVC da vi har skulle benytte os af et UI som allerede havde en model tilknyttet. Dette har forårsaget at vi i praksis har skulle opdatere to modeller når noget data havde ændret sig. Unit testen har derfor også været problematisk da UI'et har påvirket vores automatiserede test, fordi når vi har opdateret vores modeller har vi opdateret UI'et og mange unit tests har derfor ikke været mulige fordi UI'et ikke kørte. Vi konkluderet derfor at vi kun benyttede os af unit test der hvor det gav mening og vi har så fokuseret på andre tests såsom brugertest.

Selve produktet er færdiggjort og på grund af det gode samarbejde og det fokus der er blevet lagt i produktet opfylder vi de indledende krav.

9 Bilag

Bilag 1: Chancekort

1. Modtag udbytte af deres aktier: kr. 50,00.
2. De modtager »Matador-legatet« for værdig trængende, stort kr. 2000.00. Ved værdig trængende forstås, at deres formue, d.v.s. deres kontante penge + skøder + bygninger, ikke overstiger kr. 750.00
3. Tag ind på Rådhuspladsen
4. Ryk frem til Grønningen. Hvis De passerer »Start«, indkasser da kr. 200,00.
5. Tag med Øresundsbåden - Flyt brikken frem, og hvis De passerer »Start«, indkassér kr. 200,00.
6. 3x. Gå i fængsel. Ryk direkte til fængslet. Selv om De passerer "Start", indkasserer De ikke kr. 200,00.
7. De har måttet vedtage en parkeringsbøde Betal kr. 20,00 til banken
8. 2x. Ryk tre felter tilbage.
9. Ryk frem til »Start«
10. Kul- og kokspriserne er steget, og De skal betale: kr. 25,00 pr. hus og kr. 125,00 pr. hotel.
11. Ejendomsskatterne er steget, ekstraudgifterne er: kr. 50,00 pr. hus, kr. 125,00 pr. hotel.
12. De har lagt penge ud til sammenskudsgilde. Mærkværdigvis betaler alle straks. Modtag fra hver medspiller kr. 25,00.
13. Værdien af egen avl fra nyttehaven udgør kr. 200,00 som De modtager af banken.
14. 2x. Ryk brikken frem til det nærmeste dampskibsselskab og betal ejeren to gange den leje, han ellers er berettiget til. Hvis selskabet ikke ejes af nogen kan De købe det af banken.
15. De har anskaffet et nyt dæk til Deres vogn Indbetal kr. 100,00
16. De har kørt frem for »Fuld Stop«. Betal kr. 100,00 i bøde.
17. Betal for vognvask og smøring kr. 10,00
18. De har været en tur i udlandet og haft for mange cigarer med hjem. betal told kr. 20,00.
19. De har måttet vedtage en parkeringsbøde. Betal kr. 20,00 til banken
20. Grundet på dyrtiden har de fået gageforhøjelse Modtag kr. 25,00
21. 2x 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.
22. Betal for vognvask og smøring kr. 10,00.
23. De har solgt Deres gamle klude. Modtag kr. 20,00
24. De har rettidigt afleveret deres abonnementskort. Depositum kr. 1,00 udbetales Dem af banken.
25. Manufakturvarerne er blevet billigere og bedre, herved sparet De kr. 50,00. som De modtager af banken
26. Efter auktionen på Assistenshuset, hvor de havde pantsat Deres tøj, modtager De ekstra kr. 108,00.
27. Deres præmieobligation er kommet ud. De modtager kr. 100,00 af banken.

Modtag/afgiv simpelt beløb: 1, 7, 13, 15, 16, 17, 18, 19, 20, 22, 23, 24, 25, 26, 27
Modtag afhængigt af samlet formue: 2
Modtag/afgiv afhængig af ejendom: 10, 11
Modtag/afgiv fra andre spillere: 12
Flyt absolut (inkl passér start): 3, 4, 5, 9
Flyt absolut UDEN startbonus: 6
Flyt antal felter frem/tilbage: 8

Benådning: 21

Meget specifik: 14

Bilag 2: Grund-kort

Farve	navn	Leje	Leje m/x hus	Hus, pris	Hotel, Pris	Pantsætnings værdi
	RAADHUS- PLADSEN	50.kr	1 hus. 200.kr 2 huse. 600.kr 3 huse. 1400.kr 4 huse. 1700.kr hotel. 2000.kr	200.kr	200.kr + 4 huse	200.kr
	FREDERIKSBERG- GADE	35.kr	1 hus. 175.kr 2 huse. 500.kr 3 huse. 1100.kr 4 huse. 1300.kr hotel. 1500.kr	200.kr	200.kr + 4 huse	175.kr
	AMAGERTORV	26.kr	1 hus. 130.kr 2 huse. 390.kr 3 huse. 900.kr 4 huse. 1100.kr hotel. 1275.kr	200.kr	200.kr + 4 huse	150.kr
	NYGADE	28.kr	1 hus. 150.kr 2 huse. 450.kr 3 huse. 1000.kr 4 huse. 1200.kr hotel. 1400.kr	200.kr	200.kr + 4 huse	160.kr
	VIMMELSKAFTET	26.kr	1 hus. 130.kr 2 huse. 390.kr 3 huse. 900.kr 4 huse. 1100.kr hotel. 1275.kr	200.kr	200.kr + 4 huse	150.kr
	FREDERIKSBERG ALLE	10.kr	1 hus. 50.kr 2 huse. 150.kr 3 huse. 450.kr 4 huse. 625.kr hotel. 750.kr	100.kr	100.kr + 4 huse	70.kr
	BÜLOWSVEJ	10.kr	1 hus. 50.kr 2 huse. 150.kr 3 huse. 450.kr 4 huse. 625.kr hotel. 750.kr	100.kr	100.kr + 4 huse	70.kr
	GL. KONGEVEJ	12.kr	1 hus. 60.kr 2 huse. 180.kr 3 huse. 500.kr 4 huse. 700.kr hotel. 900.kr	100.kr	100.kr + 4 huse	80.kr

	VALBY LANGGADE	6.kr	1 hus. 30.kr 2 huse. 90.kr 3 huse. 270.kr 4 huse. 400.kr hotel. 550.kr	50.kr	50.kr + 4 huse	50.kr
	ROSKILDEVEJ	6.kr	1 hus. 30.kr 2 huse. 90.kr 3 huse. 270.kr 4 huse. 400.kr hotel. 550.kr	50.kr	50.kr + 4 huse	50.kr
	ALLÉGADE	8.kr	1 hus. 40.kr 2 huse. 100.kr 3 huse. 300.kr 4 huse. 450.kr hotel. 600.kr	50.kr	50.kr + 4 huse	60.kr
	HVIDOVRE	4.kr	1 hus. 20.kr 2 huse. 60.kr 3 huse. 180.kr 4 huse. 320.kr hotel. 540.kr	50.kr	50.kr + 4 huse	30.kr
	RØDOVREVEJ	2.kr	1 hus. 10.kr 2 huse. 30.kr 3 huse. 90.kr 4 huse. 160.kr hotel. 250.kr	50.kr	50.kr + 4 huse	30.kr
	ØSTERBROGADE	18.kr	1 hus. 90.kr 2 huse. 250.kr 3 huse. 700.kr 4 huse. 875.kr hotel. 1050.kr	150.kr	150.kr + 4 huse	110.kr
	GRØNNINGEN	20.kr	1 hus. 100.kr 2 huse. 300.kr 3 huse. 750.kr 4 huse. 925.kr hotel. 1100.kr	150.kr	150.kr + 4 huse	120.kr
	TRIANGLEN	18.kr	1 hus. 90.kr 2 huse. 250.kr 3 huse. 700.kr 4 huse. 875.kr hotel. 1050.kr	150.kr	150.kr + 4 huse	110.kr
	HELLERUPVEJ	14.kr	1 hus. 70.kr 2 huse. 200.kr 3 huse. 550.kr 4 huse. 750.kr hotel. 950.kr	100.kr	100.kr + 4 huse	90.kr

	STRANDVEJ	16.kr	1 hus. 80.kr 2 huse. 220.kr 3 huse. 600.kr 4 huse. 800.kr hotel. 1000.kr	100.kr	100.kr + 4 huse	100.kr
	BERNSTORFFSVEJ	14.kr	1 hus. 70.kr 2 huse. 200.kr 3 huse. 550.kr 4 huse. 750.kr hotel. 950.kr	100.kr	100.kr + 4 huse	90.kr
	ØSTERGADE	22.kr	1 hus. 120.kr 2 huse. 360.kr 3 huse. 850.kr 4 huse. 1025.kr hotel. 1200.kr	150.kr	150.kr + 4 huse	140.kr
	KGS. NYTORV	22.kr	1 hus. 110.kr 2 huse. 330.kr 3 huse. 800.kr 4 huse. 975.kr hotel. 1150.kr	150.kr	150.kr + 4 huse	130.kr
	BREDGADE	22.kr	1 hus. 110.kr 2 huse. 330.kr 3 huse. 800.kr 4 huse. 975.kr hotel. 1150.kr	150.kr	150.kr + 4 huse	130.kr

navn	Leje	Leje m/x ejendomme	Pantsætnings værdi
D/S BORNHOLM 1866	25.kr	2 Dampskibsselskaber ejes 50.kr 3 Dampskibsselskaber ejes 100.kr 4 Dampskibsselskaber ejes 200.kr	100.kr
DAMPSKIBSSELSKABET A/S ØRESUND	25.kr	2 Dampskibsselskaber ejes 50.kr 3 Dampskibsselskaber ejes 100.kr 4 Dampskibsselskaber ejes 200.kr	100.kr
DAMPSKIBSSELSKABET Ø. K	25.kr	2 Dampskibsselskaber ejes 50.kr 3 Dampskibsselskaber ejes 100.kr 4 Dampskibsselskaber ejes 200.kr	100.kr
DAMPSKIBSSELSKABET . D. F. D. S.	25.kr	2 Dampskibsselskaber ejes 50.kr 3 Dampskibsselskaber ejes 100.kr 4 Dampskibsselskaber ejes 200.kr	100.kr
CARLSBERG BRYGGERIET	4 gange så meget som slaget brugt til at lande på feltet	begge ejendomme: 10 gange så meget som slaget brugt til at lande på feltet	75.kr
BRYGGERIET TUBORG	4 gange så meget som slaget brugt til at lande på feltet	begge ejendomme: 10 gange så meget som slaget brugt til at lande på feltet	75.kr

Bilag 3: Kravspecifikation

Funktionelle krav:

Id.	Beskrivelse	Prioritet
RQF01	Spillet skal være mellem tre til seks personer	M
RQF02	Spillerne skal kunne indtaste deres ønskede navne, i en længde mellem 3-15 karakter.	S
RQF03	Spillerne skiftes til at kaste to terninger	M
RQF04	Spillerne rykker skiftevis rundt, i en længde bestemt af summen af de to terninger	M
RQF05	Spillebrættet består af felter	M
RQF06	Spillebrættet har felt kategorierne; start, ejendomme, skat, chancekort, på besøg i fængsel, fængsel, helle, færgekort og ølkort.	M
RQF07	Hver gang spilleren passerer ‘start’ modtager man 200 kr.	S
RQF08	Hvis en spiller lander på en ledig ejendom, færgekort eller ølkort har spilleren muligheden for enten at købe ejendommen	S
RQF09	Hvis en spiller lander på en ikke ledig grund, skal spilleren betale leje til ejeren	S
RQF10	Ejendomme kommer i sæt angivet med ens farver	S
RQF11	Hvis en spiller ejer alle ejendomme af samme farve - så kan de indkræve dobbelt leje, fra andre spillere	S
RQF12	Hvis man ejer alle ejendomme af samme farve så har spilleren mulighed for at bygge et hus, hvilket koster en vis mængde penge afhængig af ejendommen	S
RQF13	Hus på en ejendom forøger leje andre spillere skal betale	S
RQF14	Det er kun muligt at bygge på ejendomme	S
RQF15	Hvis man ønsker at bygge huse, skal man bygge jævnt dvs. hvis man ønsker at have to huse på en grund, skal der først være et på alle grunde.	S
RQF16	En spiller kan maks have fire huse på en grund, herefter kan spilleren tilkøbe et hotel.	S
RQF17	Det er ikke muligt at bygge mere end et hotel	S

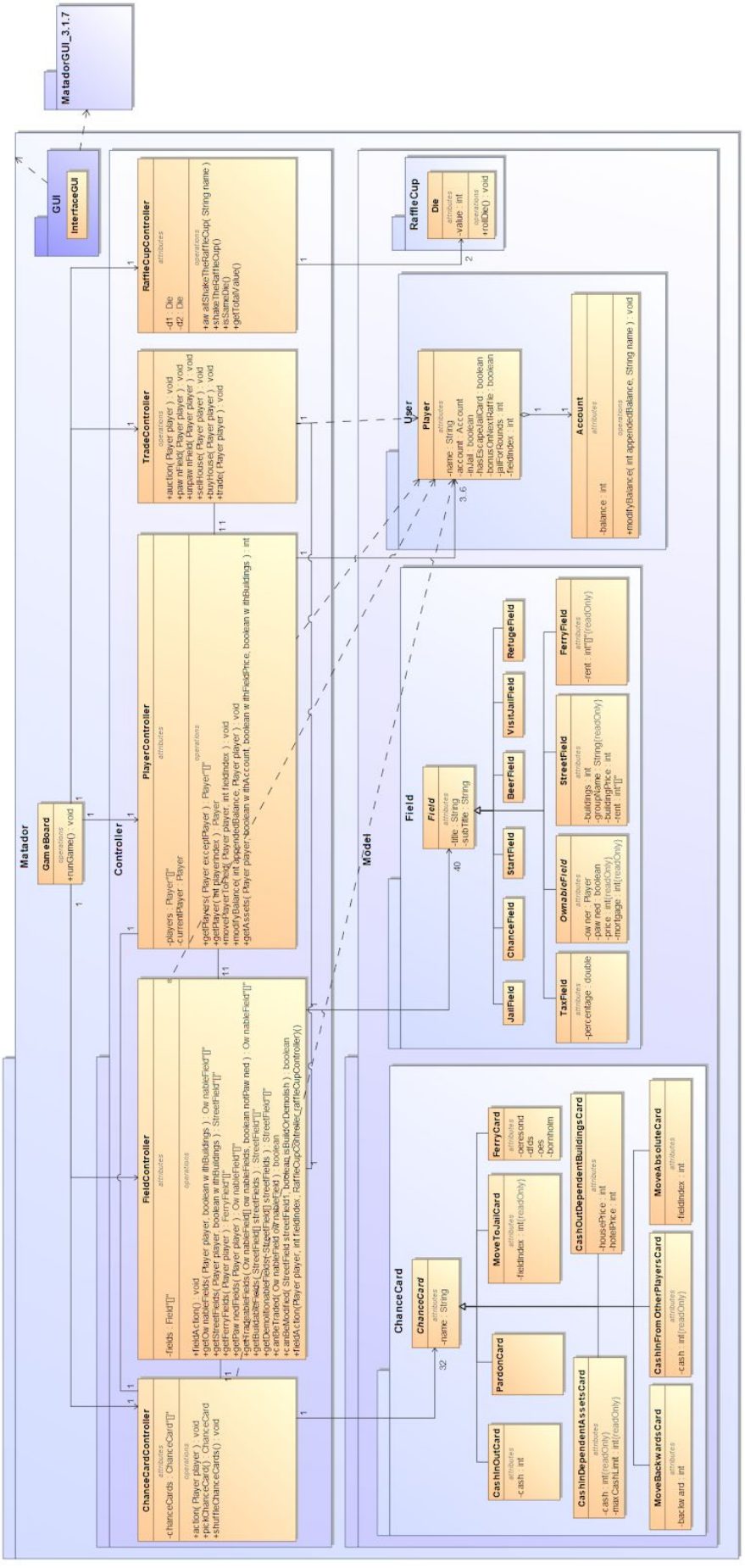
RQF18	Spilleren kan sælge huse, hvilket også skal gøres jævnt	S
RQF19	Hvis en spiller sælger et hus falder lejen på grunden	S
RQF20	Hvis en spiller spiller ikke ønsker at købe den ledige ejendom, færgekort eller ølkort, sættes den på auktion	S
RQF21	Auktionsprisen sættes af spilleren der takkede nej, og skal minimum være 1 kr	S
RQF22	Auktionen foregår mellem de resterende spiller	S
RQF23	Personer byder ved at vælge deres navn i GUI, hvorefter de kan indtaste en pris.	S
RQF24	Auktionen stopper når der klikkes "Afslut auktion"	S
RQF25	Personen med det højst bud for grunden, når der klikkes "Afslut aktion"	S
RQF26	Hvis der klikkes "Afslut auktion", uden nogle bud, forbliver grunden ledig.	S
RQF27	Efter køb grund/betal leje segmentet præsenteres muligheder, "Køb hus", "Sælg hus", "Forhandle med en anden spiller", "Pansæt grund", "Fjern pantsætning af grund" og "Afslut tur"	S
RQF28	Vælges "Køb hus", præsenteres spilleren enten for "Ingen mulige bygge grunde", eller præsenteres for mulige byggegrunde. Spilleren vælger den ønskede grund, hvorefter der bekræftes ved "Ok"	S
RQF29	Vælges "Sælg hus" præsenteres spilleren enten for "Ingen hus at sælge", eller præsenteres for mulige huse at sælge. Spilleren vælger den ønskede grund, hvorefter der bekræftes ved "Ok"	S
RQF30	Vælges "Forhandle med en anden spiller" kan spillere vælger en af sin ejet grund, hvorefter du vælger anden spiller der ønske købe den grund. Den modsatte spiller kan indtaste ønsket købspris, og bekræfte. Herved overgå grunden til den anden spiller	C
RQF31	Hvis man ønsker at forhandle med en grund, skal grunden være ubebygget	C
RQF32	Vælges "Pansæt grund" kan spillere vælge ejet ubyggede grunde, hvorefter spilleren modtager halvdelen af købsprisen ved pantsætningen	C
RQF33	På en pantsat grund kan der ikke opkræves leje	C

RQF34	Pantsatte grunde kan købes af andre spillere	C
RQF35	Hvis en anden spiller købe en pantsat grund, skal spilleren betale for at ophæve pantsætning, udover forhandlingsprisen	C
RQF36	Vælges “Fjern pantsætning af grund”, kan spilleren vælge i mellem pantsætte grund, hvorved bekræftelsen, betales 110% af pantsætnings prisen	C
RQF37	Hvis en spiller vælger “Afslut tur” går turen videre, og næste spiller kaster med terningerne.	M
RQF38	En spillere modtager en ekstra tur hvis de slår to ens	M
RQF39	Hvis en spillere tre gange slår to ens ryger de direkte i fængsel	S
RQF40	Hvis en spillere ryger i fængsel kan de komme ud ved betale 100kr i begyndelsen af deres tur, slå to ens, hvorefter de rykke antal flette terningerne viser eller benytte “Get out of jail”-card, hvorefter spiller kan kaste terningerne.	S
RQF41	Hvis en spillere ejer to færge felter, bliver lejen fordobblet på begge grunde	S
RQF42	Hvis en spiller lander på et ejet øl-felt, er lejen den samlede terningsværdi gange lejen på feltet.	C
RAF43	Hvis spilleren lander på et indkomstskat felt skal de enten betale 200kr eller 10% af deres penge beholdning	S
RQF44	Hvis spilleren lander på et ekstraordinær statsskat felt skal de betale 100kr	S
RQF45	‘Helle’ feltet er et fristed, indtil man skal kaste igen.	S
RQF46	Hvis en spiller lander på ‘Gå i fængsel’, skal man direkte i fængsel og modtager ikke penge hvis man passerer start.	S
RQF47	Hvis en spiller lander på et chancekort, skal de trække en af de 32 chancekort. Aktionen præsenteres ved en tekst, hvorefter aktionen påføres spilleren.	S
RQF48	En spillere udgår hvis de løber tør penge ressource, til at opholde en pengebeholdning over 0 kr	M
RQF49	En spillere begynder med 3000 kr	M
RQF50	Spillet stopper når kun en enkle spillere står tilbage	M

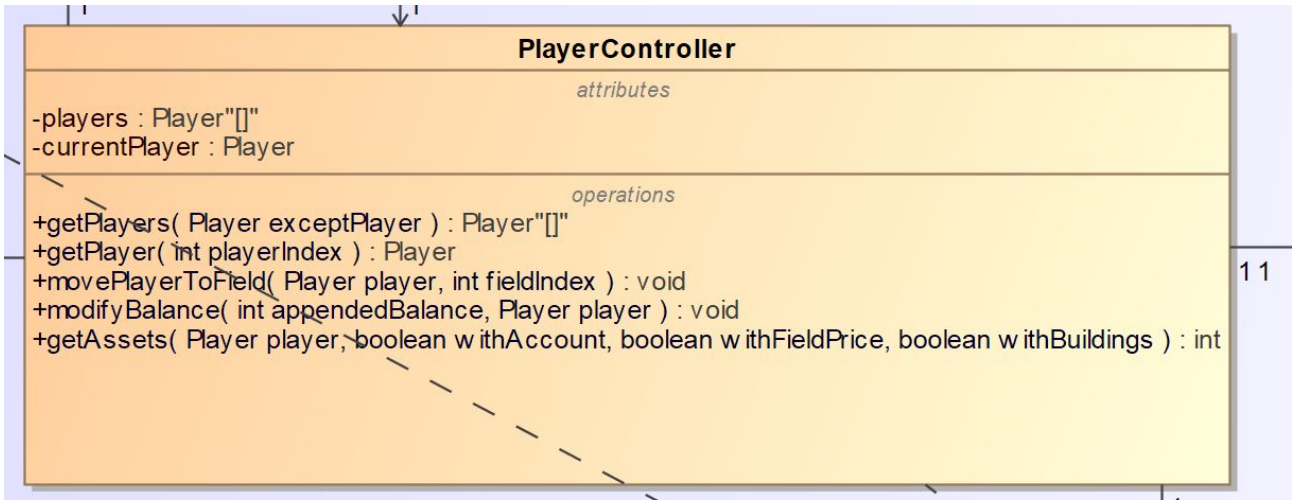
Ikke funktionelle krav:

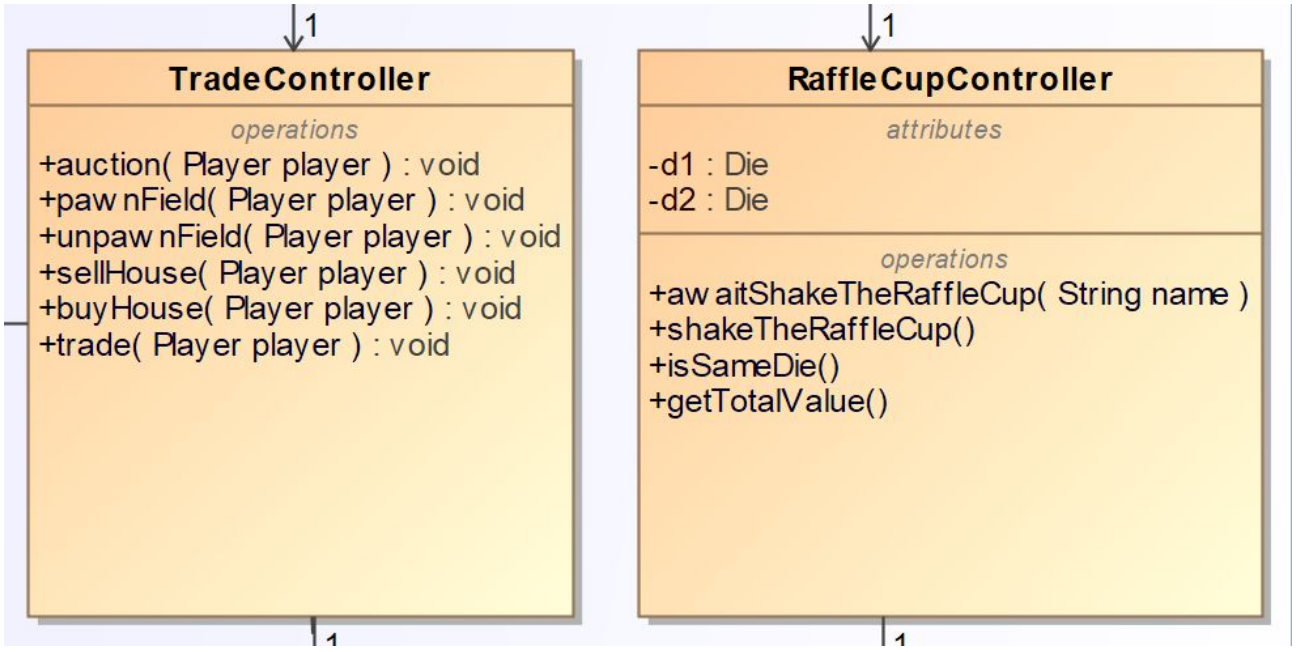
Id.	Beskrivelse	Prioritet
RQIF01	Spillet skal kunne tilgås via maskinerne i DTU's databarer, uden bemærkelsesværdige forsinkelser.	M
RQIF02	Der skal udarbejdes en beskrivelse af minimumskrav samt vejledning i hvordan kildekoden compiles, installeres og afvikles. Dette inkluderer en beskrivelse af hvordan koden importeres fra et git repository.	M

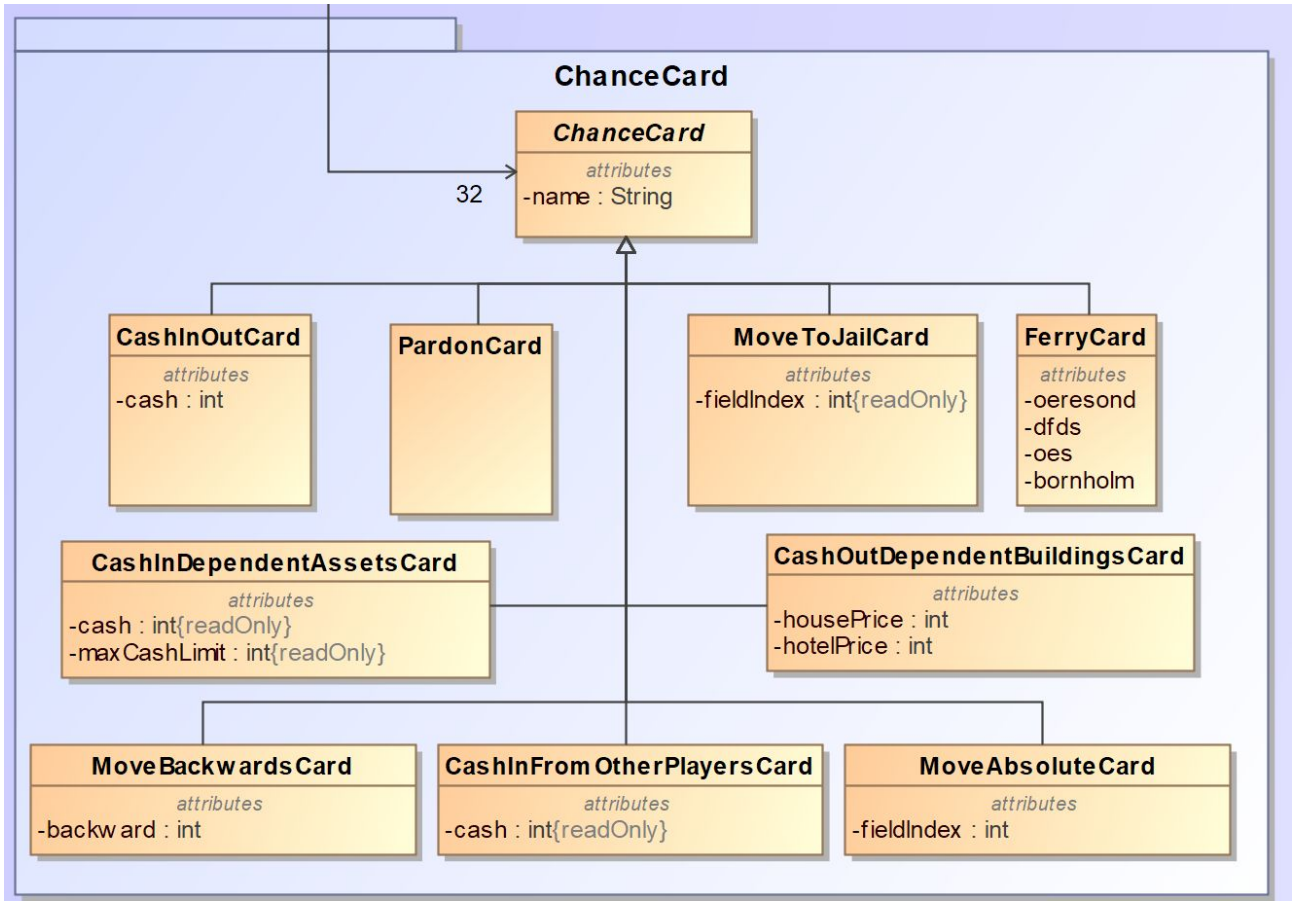
Bilag 4: Klassediagram

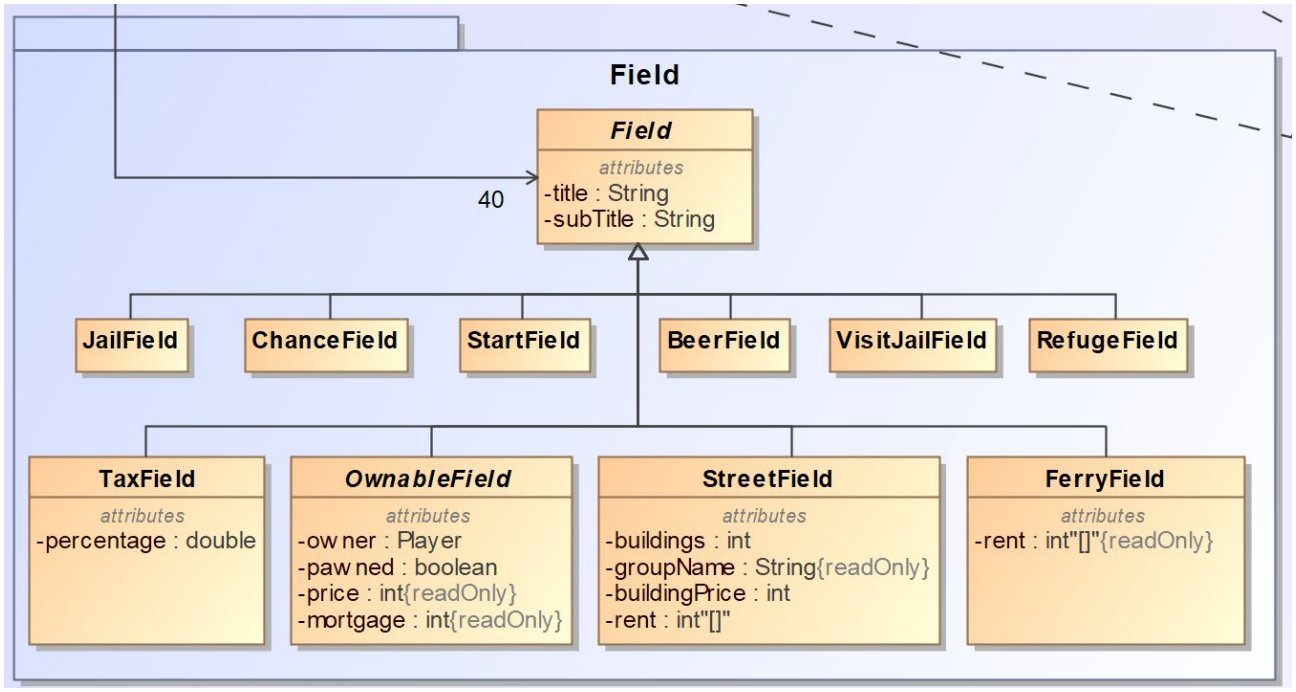


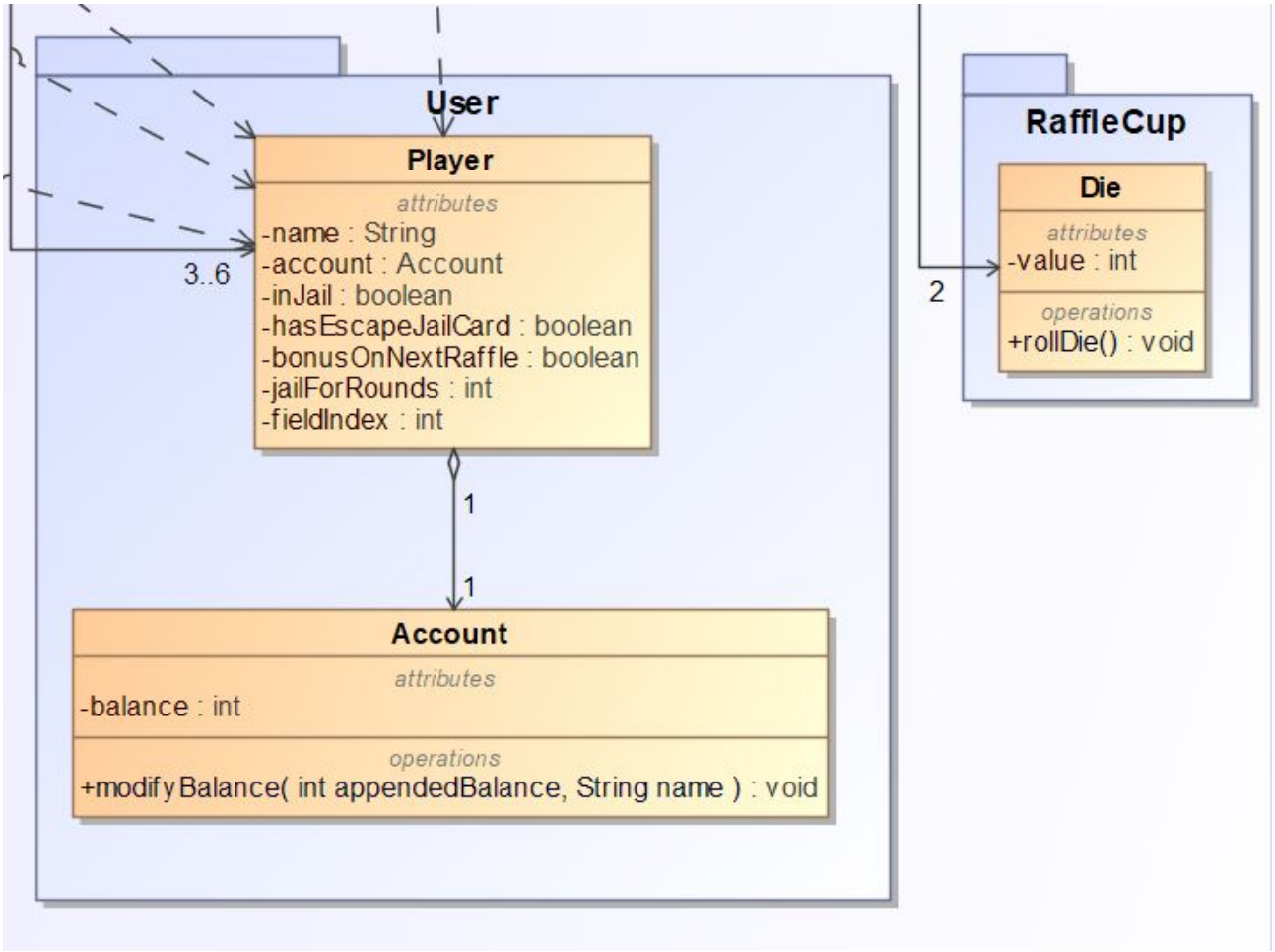

ChanceCardController
attributes
-chanceCards : ChanceCard"[]"
operations
+action(Player player) : void
+pickChanceCard() : ChanceCard
+shuffleChanceCards() : void
FieldController
attributes
-fields : Field"[]"
operations
+fieldAction() : void
+getOwnableFields(Player player, boolean withBuildings) : OwnableField"[]"
+getStreetFields(Player player, boolean withBuildings) : StreetField"[]"
+getFerryFields(Player player) : FerryField"[]"
+getPawnedFields(Player player) : OwnableField"[]"
+getTradeableFields(OwnableField[] ownableFields, boolean notPawned) : OwnableField"[]"
+getBuildableFields(StreetField[] streetFields) : StreetField"[]"
+getDemolitionableFields(StreetField[] streetFields) : StreetField"[]"
+canBeTraded(OwnableField ownableField) : boolean
+canBeModified(StreetField streetField1, boolean isBuildOrDemolish) : boolean
+fieldAction(Player player, int fieldIndex, RaffleCupController raffleCupController)()
1
11


PlayerController
attributes
-players : Player"[]"
-currentPlayer : Player
operations
+getPlayers(Player exceptPlayer) : Player"[]"
+getPlayer(int playerIndex) : Player
+movePlayerToField(Player player, int fieldIndex) : void
+modifyBalance(int appendedBalance, Player player) : void
+getAssets(Player player, boolean withAccount, boolean withFieldPrice, boolean withBuildings) : int
1 1

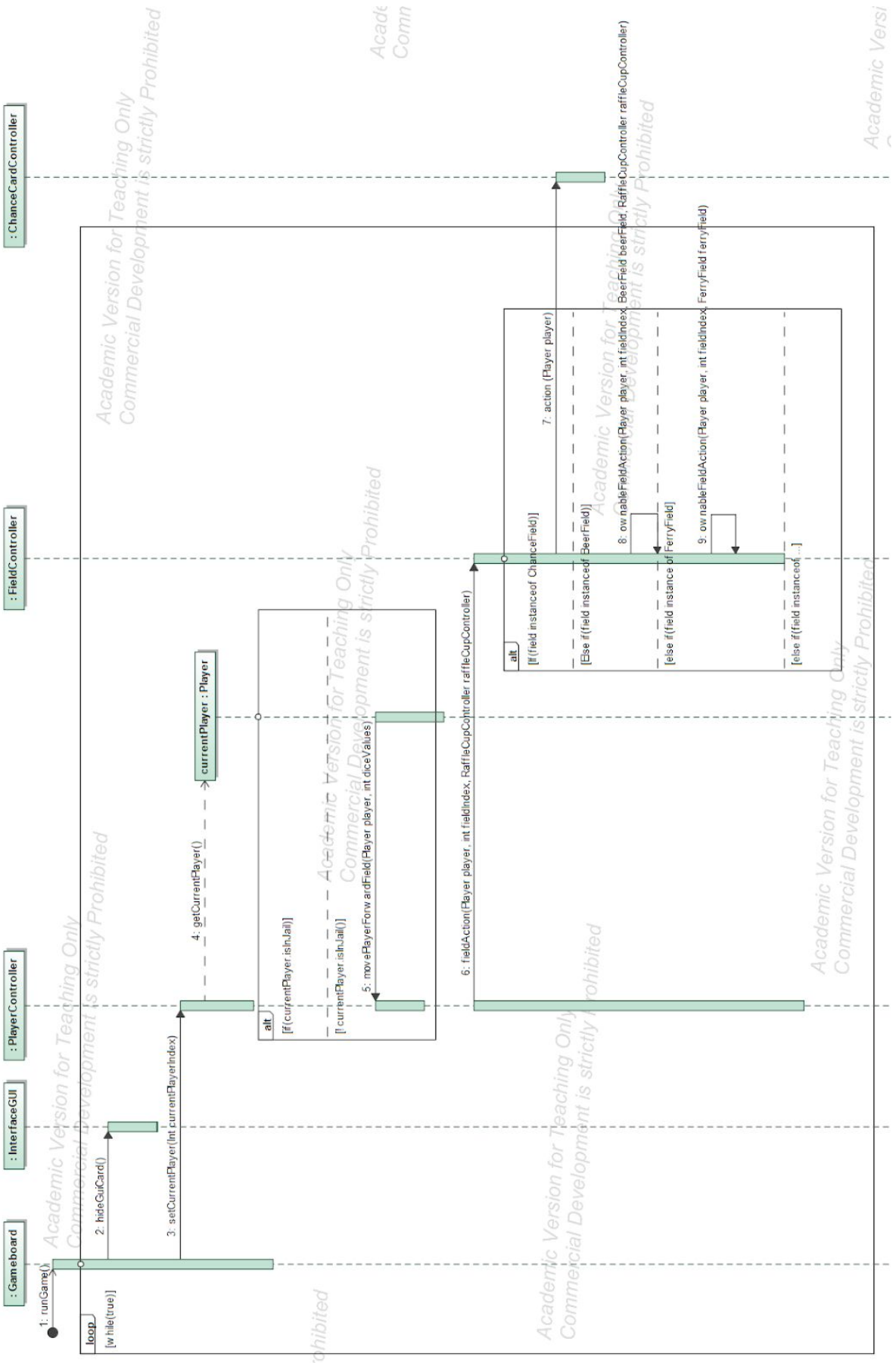

1
TradeController
operations
+auction(Player player) : void
+pawnField(Player player) : void
+unpawnField(Player player) : void
+sellHouse(Player player) : void
+buyHouse(Player player) : void
+trade(Player player) : void
1
RaffleCupController
attributes
-d1 : Die
-d2 : Die
operations
+awaitShakeTheRaffleCup(String name)
+shakeTheRaffleCup()
+isSameDie()
+getTotalValue()

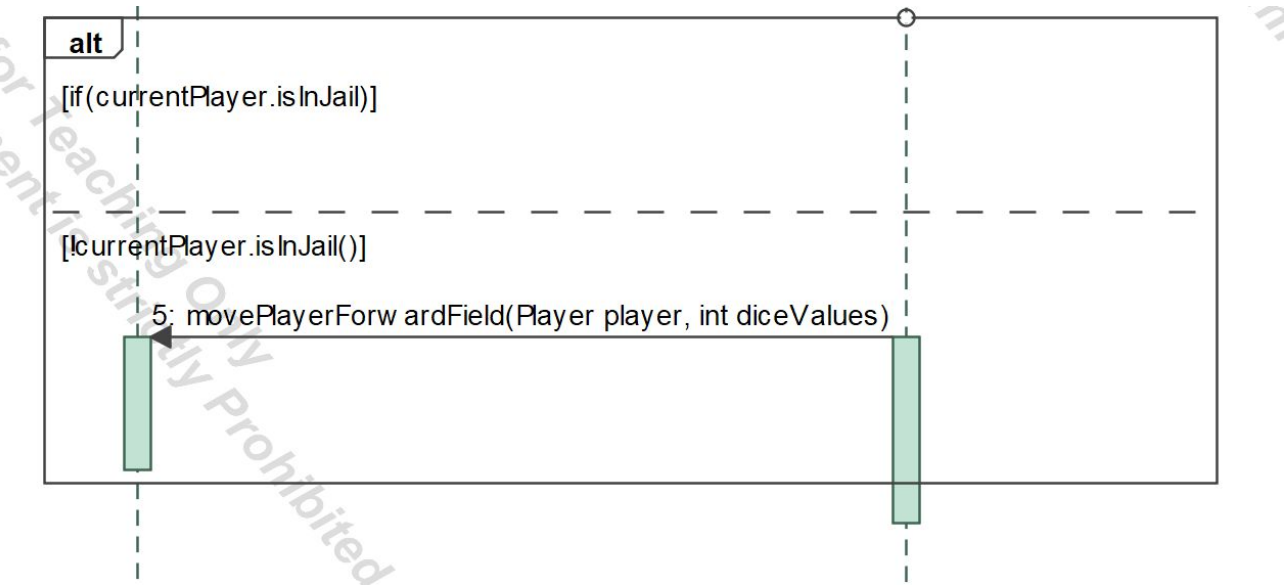

ChanceCard
ChanceCard
attributes
-name : String
32
CashInOutCard
attributes
-cash : int
PardonCard
MoveToJailCard
attributes
-fieldIndex : int{readOnly}
FerryCard
attributes
-oeresond
-dfds
-oes
-bornholm
CashInDependentAssetsCard
attributes
-cash : int{readOnly}
-maxCashLimit : int{readOnly}
CashOutDependentBuildingsCard
attributes
-housePrice : int
-hotelPrice : int
MoveBackwardsCard
attributes
-backward : int
CashInFromOtherPlayersCard
attributes
-cash : int{readOnly}
MoveAbsoluteCard
attributes
-fieldIndex : int

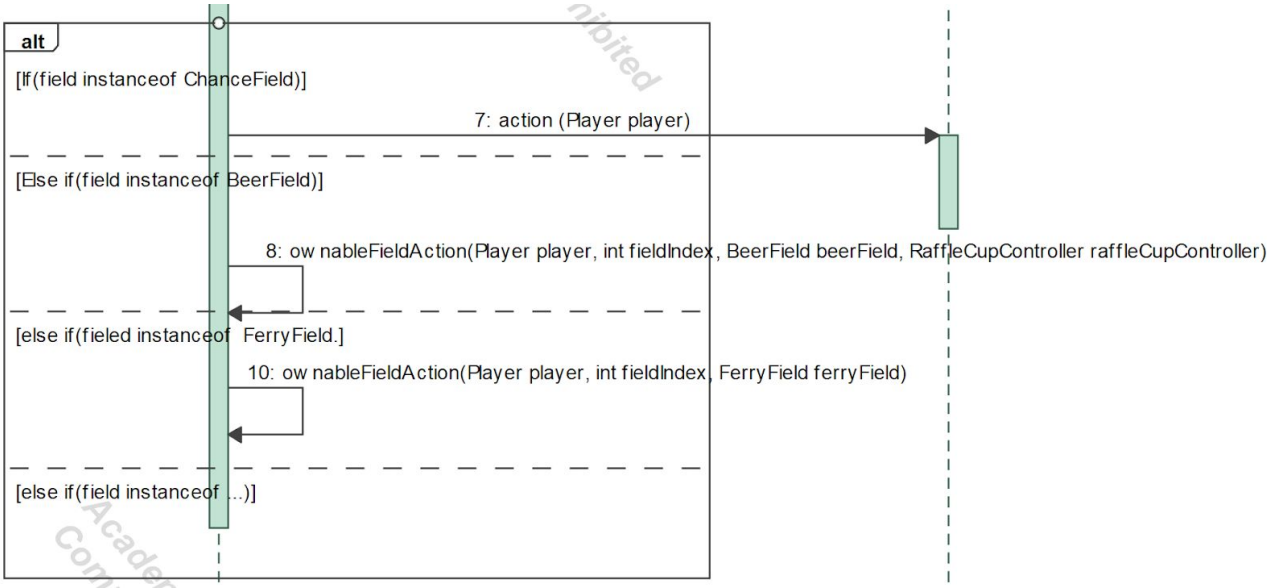

Field
Field
attributes
-title : String
-subTitle : String
40
JailField
ChanceField
StartField
BeerField
VisitJailField
RefugeField
TaxField
attributes
-percentage : double
OwnableField
attributes
-owner : Player
-pawned : boolean
-price : int{readOnly}
-mortgage : int{readOnly}
StreetField
attributes
-buildings : int
-groupName : String{readOnly}
-buildingPrice : int
-rent : int"[]"
FerryField
attributes
-rent : int"[]"{readOnly}


User
Player
attributes
-name : String
-account : Account
-inJail : boolean
-hasEscapeJailCard : boolean
-bonusOnNextRaffle : boolean
-jailForRounds : int
-fieldIndex : int
3..6
1
1
Account
attributes
-balance : int
operations
+modifyBalance(int appendedBalance, String name) : void
RaffleCup
Die
attributes
-value : int
operations
+rollDie() : void
2

Bilag 5: Sekvensdiagram




alt
[if(currentPlayer.isInJail)]
[!currentPlayer.isInJail()]
5: movePlayerForw ardField(Player player, int diceValues)


alt
[If(field instanceof ChanceField)]
7: action (Player player)
[Else if(field instanceof BeerField)]
8: ow nableFieldAction(Player player, int fieldIndex, BeerField beerField, RaffleCupController raffleCupController)
[else if(fieled instanceof FerryField.]
10: ow nableFieldAction(Player player, int fieldIndex, FerryField ferryField)
[else if(field instanceof ...)]