# Matador

## Projekt og gruppe info:

**Danmarks Tekniske Universitet**
**Underviser:** Henrik Hauge
**Projektnavn:** Project in Software Development
**Gruppe nr.:** 4
**Arbejdsperiode:** 07-02-2014 - 28-04-2014
**Afleveringsfrist:** 29-04-2014
Rapporten indeholder følgende antal sider alt inklusiv: 116 hvor af siderne 27 til 116 er bilag.
Studie numre og navne på projektdeltagere:
Mads Viborg Jørgensen - s134221@student.dtu.dk

Jakob Sabinsky - s101480@student.dtu.dk

Magnus Brandt Sløgedal - s103185@student.dtu.dk

# Indholdsfortegnelse

# Forord og indledning

Projektet er det primære produkt af kurset 02802 Project in software development.

I kurset har der været fokus på Objektorienteret programmering i java og samarbejdet mellem dette og databaser.
Af kurset blev det forventet af kunne bruge ArrayLists, SQL, lagdeling til program designet og grundlæggende terminologi fra den relationelle model.
Det forventes endvidere at kunne designe et program og en database ud fra en givet problemstilling via en trin-baseret strategi og kunne bruge versionskontrol.

Vi har i dette projekt brugt Eclipse som vores primære JAVA-udviklingsmiljø, ud over de normale java biblioteker i JAVA version 7, har vi brugt “mysql-connector-java-5.1.28-bin.jar” til at oprette forbindelse til vores SQL database. Til at bygge databasen har vi brugt Mysql-workbench og Heidisql, og til at modellere vores program og vores database har vi brugt Microsoft Visio, og Software Ideas Modeller.

# Problemstilling og baggrund

## Projektbeskrivelse

Formålet med projektet er at vi skal lave et matadorspil. Spillet skal kunne spilles af 2-6 spillere, hvor reglerne er ligesom i et klassisk matadorspil. Spillet slutter først når alle på nær en er bankerot. Spillet skal endvidere kunne gemmes via en MySQL database, og det gemte spil loades.
For at få vores system til at opfylde kundens krav bedst muligt, og få svar på de spørgsmål der måske ikke har været beskrevet i det skriftlige dokument vi har fået udleveret, har vi under forløbet haft mulighed for at spørge vores hjælpelærer til råds, der har ageret projektleder fra virksomheden spillet skal laves til.

Følgende afsnit vil opsummere de vigtigste detaljer af kravene fra kunden til produktet.

## Kravspecifikationer

Formålet med systemet er at lave et funktionelt matadorspil, der lever op til kundens krav.

### Funktionelle krav

For at gøre de funktionelle krav overskuelige er de delt op i 5 kategorier:

1) Setup
2) Spillet
3) Huse
4) Pantsætning
5) Fallit

**Setup:**

- Spillere - 2 - 6
- Startkapital - kr. 30.000
- 32 grønne huse - 12 hoteller
- Spillet skal kunne gemmes og det gemte spil skal kunne loades.

**Spillet:**

- En spiller ad gangen slår med to D6 terninger, og rykker sin brik det viste antal felter
- Passere en spiller start Indkasserer spiller kr. 4.000
- Standser spiller på et "Prøv Lykken" felt, trækkes et chancekort. Disse placeres, efter de er brugt, nederst i bunken.
- Standser spiller på et felt der ikke ejes af nogen, har spiller mulighed for at købe det
    - Vælger spiller ikke at købe det, sætter banken det på auktion, som alle, inc.spilleren der landede der, kan deltage I.
- Standser spiller på "de sættes i fængsel" går spiller direkte i fængsel, uden at inkasere start-penge"
- Slår spiller to ens, får spiller et ekstra kast.
- ○ Slår spiller to ens, 3 gange i træk, får spilleren **IKKE** det tredje ekstra slag, men skal i stedet gå i fængsel, man indkassere ikke penge ved start når spiller går i fængsel.
- Spiller kan komme ud af fængsel ved at:
    - betale en bøde på kr. 1.000, **Før** man kaster med terningerne
    - Ved at benytte et af benådnings-kortene
    - ved at slå to ens
    - Man flytter da det antal øjnene viser, og har et extra kast
- Man kan ikke være I fængsel mere end tre omgange, får man ikke to ens tredje omgang må man betale bøden på kr. 1.000 og flytte det antal øjne man slog.
- Mens man er fængslet kan man stadig købe og sælge på lige fod med andre
- Standser man på "Fængsel" er man blot på besøg
- Indkomstskat: betales enten kr. 4.000 eller 10% af ens samlede aktiver (Kontanter, bygninger og den trykte pris for grunde og virksomheder (også pantsatte).
    - Betalingsmåden skal vælges før man tæller aktiver sammen
- Spillerne må ikke låne indbyrdes.

- Glemmer man at kræve leje af en medspiller, har man tabt sin ret, når nr. 2 efter ham har kastet.

**Huse:**

- Ejer man alle grundene inden for en farve, får man dobbelt leje på disse.
    - Man har også ret til "når som helst" at bygge huse på disse grunde.
- Der skal bygges jævnt, dvs. hus 2 må først bygges når alle grunde i en farve har 1 hus.
- Et hotel må kun bygges når der er fire huse på alle grunde i en farve.
    - Når et hotel købes indleveres de 4 huse på grunden til banken
    - Prisen for et hotel er 5 gange prisen for et hus
- Banken køber husene tilbage til halv pris
- Indbyrdes handel med ubebyggede grunde etc. er spillerne tilladt til den pris, de kan blive enig om.
    - Har man bygget, skal man sælge bygningerne tilbage til banken, inden man kan afhænde nogen grund i den pågældende gruppe.
- Er der ikke nok bygninger, må man vente til der bliver nogen tilgængelige
    - Har banken ikke nok, udbydes bygningerne ved auktion.

**Pantsætning:**

- Man kan pantsætte **ubebyggede** grunde til det beløb der står på bagsiden af skødet
    - Renten på pantsatte grunde er 10% af lånet, der betales sammen med lånet når pantsætningen ophæves.
    - Man kan ikke kræve leje af pantsatte grunde.
- Banken giver kun Iøn mod pantsætnings sikkerhed.
- Hvis en pantsat ejendom sælges, og køberen ikke straks hæver pantsætningen, må han alligevel betale 10%, hvis han senere hæver pantsætningen, atter 10%.

**Fallit:**

- Skylder en spiller mere end han han har (efter salg af assets), skal han overgive alt til sin kreditor, og han udgår af spillet.

Dette var de funktionelle krav til systemet. I praksis, næsten kun matador reglerne. Når man designer sådan et system ligger der dog mere i systemet end at det blot virker og følger funktionalitets kravene.

### Non-funktionelle krav

De nonfunktionelle krav er de krav der sættes til systemet for at give brugeren en bedre oplevelse herunder at systemet ikke fejler og ikke er fx langsomt. Herudover er de non-funktionelle krav til systemet de vi sætter til systemet mulighed for fremtidig ændring, optimering og dermed generelt hvor pænt systemet er programmeret.

**Usability**
Spillet skal være brugervenligt, således at nye spillere kan sætte sig direkte ned med spillet og begynde uden en længere vejledning.

**Reliability**
Det kræves at spillet ikke har nogle fejl, som gør at spillet ikke kan fungere som planlagt uden at crashe spontant.

**Performance**
Systemet forventes ikke har have større forsinkelser, efter spillet er blevet startet op.

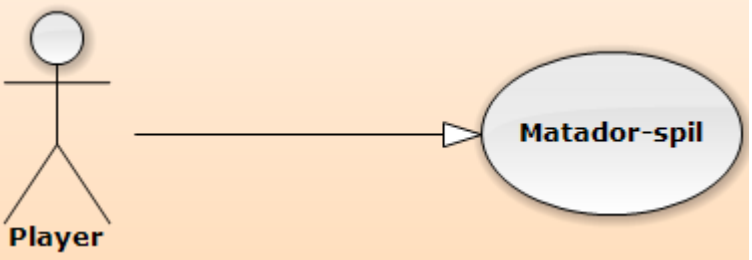
**Implementation**
Spillet skal være nemt at gå ind og ændre i. Dette er alt fra hvis man ønsker at ændre beskederne, der kommer ud fra felterne til hvis man ønsker at ændre i prislejet.

Dette var kravene til systemet, og ud fra disse kan vi nu begynde bevæge os videre til en analyse af de centrale problemstillinger.

# Analyse

## Use case diagram

Formålet med diagrammet er at vise kunden hvilke forskellige interaktioner aktører kan udføre, alt efter aktørens rolle.




## Use case beskrivelser

**Matadorspil use case.**

Use case:
Spil
Level:
User-goal
Primary Actor:
Spillerne

Preconditions:
Der skal være mellem 2 og 6 spillere til at spille spillet
Computeren skal have Java installeret

Succes Guarantee:
For at vores use case ”Matadorspil" skal være en succes kræver det at spillet kan spilles fra start til slut uden nogen form for fejl.
Endvidere skal det være muligt, at gemme spillet og lukke dette ned, for på senere tidspunkt at åbne spillet og fortsætte.

Main Success Scenario:

- Spillet bliver oprettet
- Spillerne bliver oprettet med navn og farve
- Der sendes en besked til bruger, om nu er spillet i gang.
- Navnet på den første spiller bliver hentet. Og denne bliver nu bedt om at slå med terning.
- Spiller går ind i et loop som ender når spillers tur er slut
- Spillerens bil bliver rykket i GUI det antal felter som terningerne viser
- Spilleren tager den aktion som feltet kræver
- Hvis spilleren slog to ens i første slag, gentages tur
- Slår spiller to ens tredje gang sættes spiller i fængsel
- Slår spiller ikke to ens, afsluttes dennes tur
- Er spillers konto på nul eller minus erklæres spiller for konkurs og er ude af spillet næste spiller slår med terningerne

Mulige udvidelser:

- Hvis en spiller lander på en grund, og denne ikke ejes af en anden spiller, kan grunden købes.
- Ejer en spiller grunden en anden spiller lander på, skal der betales leje til ejeren
- Ejer en spiller alle grundene i samme farve, f ar spilleren dobbelt husleje.
- Ejer spilleren alle grundene i samme farve, kan spilleren købe huse til sine grunde. Det er et krav at der bygges jævnt på grundene, dvs. et hus ad gangen på hver grund.
- Ejer spilleren 4 huse på alle sine grunde i samme farve, kan spilleren købe hoteller til sine grunde.
- Det er en forudsætning at spilleren har penge til at betale, før spilleren kan købe.
- Spiller skal sælge sine huse tilbage til banken, før spiller kan sælge grunden.
- Lander spilleren på ”Prøv lykken", trækkes der et prøv lykken kort, og følger ordren på lykke-kortet.
- Lander spilleren på fængsel, er spiller kun på besøg.
- Lander spilleren på feltet ”De sættes i fængsel" skal spilleren rykke i fængsel.
- Passere spillere start modtager spiller kr. 4000
- Spiller kan komme ud af fængsel ved at betale kr. 1000, slå 2 ens eller bruge et løsladelseskort fra ”prøv lykken".
- Lander en spiller på feltet ”indkomst skat" vælger spiller om spiller ønsker at betale enten 10% af samlede værdier eller kr. 4000
- Lander en spiller på ”Ekstraordinær statsskat" betales kr. 2000.

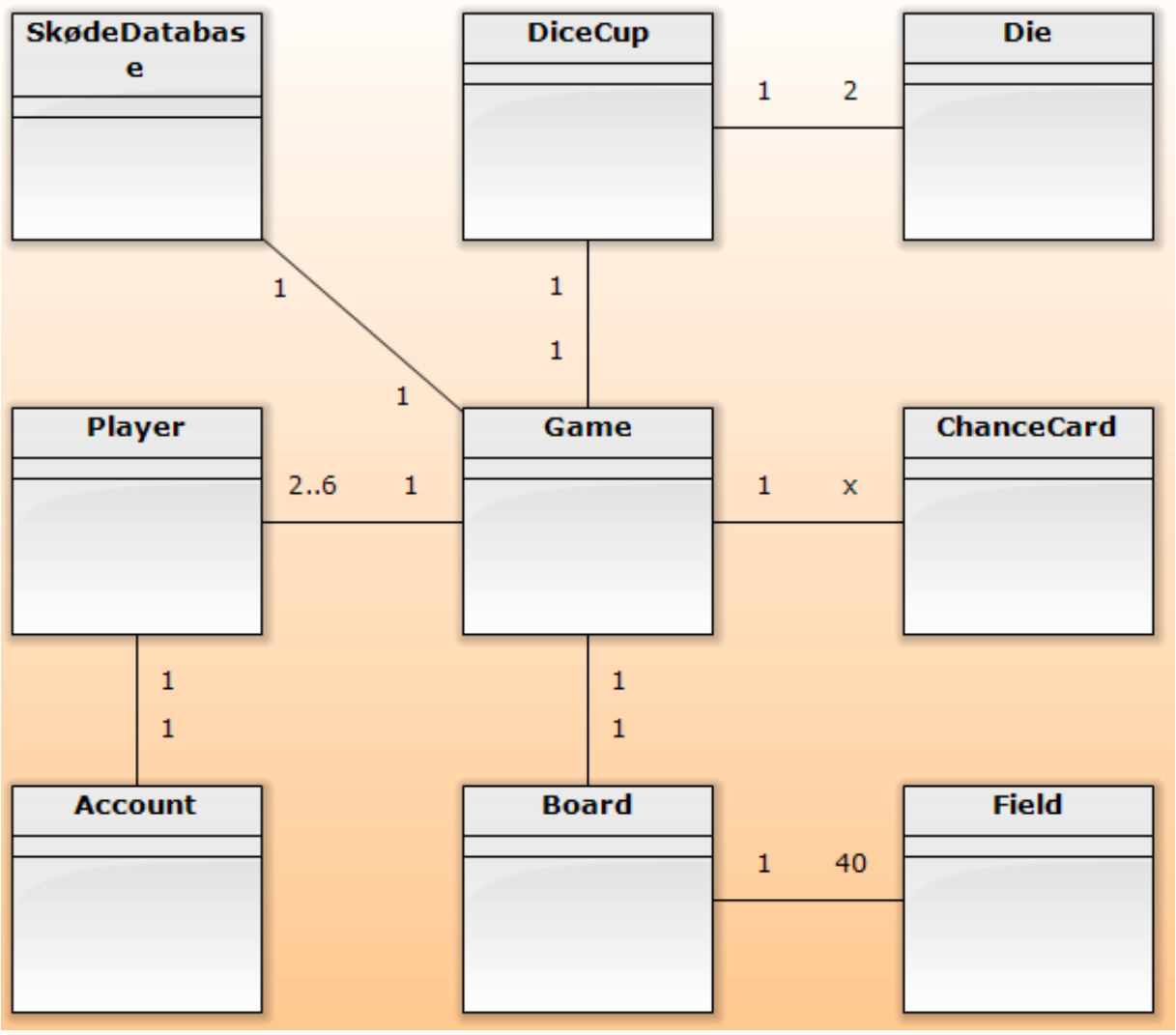
## System Sekvensdiagram

System sekvens diagrammet viser hvordan spillere og system integrere med hinanden. Her vises hvilke informationer spillerne giver til systemet, og hvad systemet returnerer med. Hermed får man nemt et godt overblik over programmets datastruktur.

Kan downloades i stor udgave her.
Ellers ses i Bilag B - SSD.

## Domænemodel

En domænemodel illustrerer sammenhænge mellem de forskellige klasser, ens kode er konstrueret med. Det er vigtigt at lave denne model inden man begynder at kode, da det særligt er her vi kan optimere i forholde til  GRASP.



## Den valgte løsning

Vi har valgt at se bort fra mulighederne for at sælge sine felter på auktion samt pantsætte, da vi så dette som den laveste prioritet ift. at spille et matadorspil.

Endvidere har vi valgt at fokuserer på brugerens oplevelse og lave en GUI, da dette også forsimpler brugerens forståelse af spillet og at brugeren kan overskue hvilket felter han ejer, hvor sin og sine medspilleres lokations er osv.

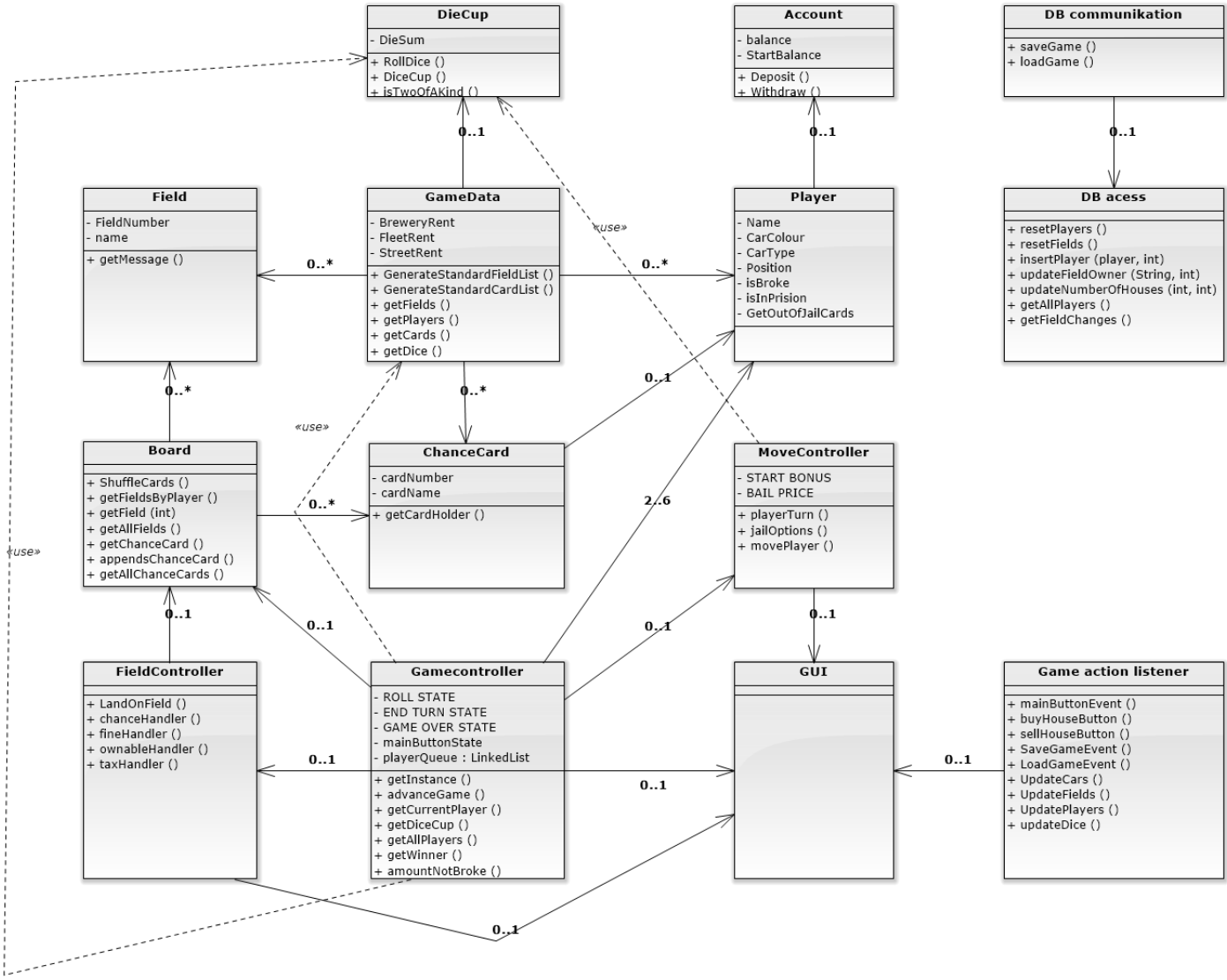
Valg af design og strukturering af kode vil kommende i det følgende afsnit.

# Implementering

## Struktur af projekt som helhed

Da vi har valgt at bruge  egen GUI, der i modsætning til den udleverede GUI, er event drevet, har vores projekt taget form derefter. Dette betyder at vores program har taget form af en statemachine, der venter på at der bliver trykket på en knap, og reagere alt efter hvordan de databærende klasser ser ud.
States styres i GameController klassen i metoden advanceGame(), der er tre mulige states: `ROLL_STATE`, `END_TURN_STATE` og `GAME_OVER_STATE`. Disse states er navngivet efter hvad der sker næste gang man trykker på "Hoved-knappen", og spillet vil hoppe mellem de første 2 indtil en vinder er fundet.
Hver gang `ROLL_STATE`, køres (ved at der trykkes på knappen), kalder spillet movementController() der har ansvaret for at flytte spilleren, og herefter kaldes fieldController() der har ansvaret for hvad der sker når en spiller lander på et felt.
når `END_TURN_STATE` kører, skifter GameController til næste spiller, og går tilbage til `ROLL_STATE`, forudsat at der stadig er mere end 1 spiller aktiv.

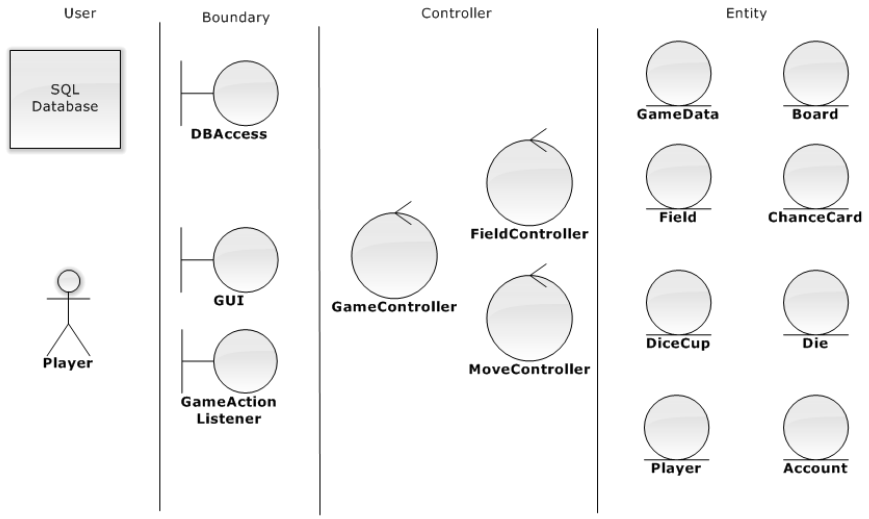
## Design klasse diagram




Ved et design-klassediagram forklarer vi sammenhængen mellem klasserne, samt hvilke attributter og metoder klasserne bruger.

## BCE modellen.

Vi har bygget dette projekt efter BCE - modellen. Vi har et front-end Boundary lag, bestående af klasserne GUI() startMenuDialog(), og GameActionListener(), et Control lag bestående af GameController(), MoveController() og FieldController() samt et entitets lag med en masse databærende klasser, så som Player() og Field(). Vi har også et back-end Boundary lag bestående af DBCommunication() og DBAccess() der står for at tale med vores SQL-database.




## GRASP - General Responsibilty Assignment Software Patterns

Vi har i dette projekt fulgt de 5 GRASP principper. Vi vil her komme med eksempler på, hvor vi i vores opgave har anvendt principperne.

1. Creator

I objekt orienteret software udvikling arbejder vi med objekter. Disse skal oprettes et sted, i vores spil er har vi bl.a. objekter i for af spillepladens felter. Disse opretter vi i klassen Board da det er her den bliver brugt.

2. Information Expert

Dette princip går ud på at bestemme, hvor et bestemt ansvar for funktionalitet skal ligge. Man kigger på hvilken slags data der kræves for at opgaven kan udføres. Det er er mest hensigtsmæssigt at lægge det i den klasse hvor det i forvejen bliver brugt mest.
Et eksempel på dette er vores “if” sætning der checker om en spiller har passeret start, denne bliver brugt i MoveControlleren, da det er her informationen, om en spillers rotation på spillepladen bliver bliver brugt mest.

3. Lav kobling

Vi har gået efter at lave en så lav kobling som muligt mellem vores klasser, Det har den betydning at klasserne bliver mindre afhængige af hinanden og er derfor også nemmere at ændre på uden at lave rav i de andre klasser. Og når klasserne er mindre afhængige af hinanden bliver de også nemmere at bruge i andre systemer.

4. Controller
Controllere er, hvor vi gemmer systemets logik, og controllerne er det første efter vores boundary. Vi har 3 controllere GameController, MoveController og FieldController. Det er dem der uddelegere og udfører diverse opgaver der kræves af systemet. Vores controllere laver ikke meget af arbejdet selv, i stedet uddeler den opgaverne til de andre klasser.

5. Høj binding
Når vi taler høj binding handler det om at sine klasser simple og relevante. Det er ikke meningen af en klasse skal være stor og udføre mange opgaver. Så ender det med at vi får en høj kobling, hvor en klasse er for kompleks og har kontakt til for mange klasser. Derfor har vi mange små klasser i vores kode. Tag for eksempel pakken med chancekortene, her har vi valgt at lave en klasse for, for de enkelte typer af chancekort. Man kunne godt have valgt at lave dem i en klasse der hed ChanceCard. Vi valgte at gøre som vi har gjort, for at få lav kobling og høj binding.

## Packages

### Game

I pakken Game har vi 9 klasser:
Account klassen er lavet fordi alle spillere skal have en konto. Når en spiller bliver oprettet får han en konto med et start kapital på kr. 30.000. Klassen har også metoder der gør at en konto f.eks ikke man ikke kan indsætte et negativt beløb eller at det ikke er muligt at hæve mere end der er tilrådighed.

Board klassen er her vi har vores array af felter “fields”, ligesom metoden der ser, hvilke felter der er ejet af en bestemt spiller og returnere et array med felterne i.
Nedenfor er den kode vi har lavet til at blande Chance kortene.

```
private ChanceCard[] ShuffleCards(ChanceCard[] cardArray)
{
    Random rand = new Random();
    for (int i = 0; i < cardArray.length; i++)
    {
        int randomPosition = rand.nextInt(cardArray.length);
        // swap objects at i and randomPosition
        ChanceCard temp = cardArray[i];
        cardArray[i] = cardArray[randomPosition];
        cardArray[randomPosition] = temp;
    }
    return cardArray;
}
```

Die klassen er, hvor vi har lavet funktionen på en seks sidet terning, der efter at være blevet "rolled" vil returnere en "face value" og DiceCup klassen er her vi sætter to terninger sammen og returnere et fælles sum. Her bliver også tjekket om der er blevet slået to ens flere gange i træk.

FieldController er klassen er, hvor alle funktionerne tilhørende felterne bliver klaret. Det er f.eks. her at en spiller bliver sendt i fængsel, hvis han lander på "Go to jail", her er der også metoder til at håndtere de felter der er ownable samt skat og chance kortene.

GameController klassen er hvor spillet styres fra, det er her spillet bliver oprettet, den holder styr på hvis tur det er, og den afslutter spillet når der er en vinder.

GameData klassen indeholder data omkring spillet, bl.a. et todimensionelt array med priser for leje af de forskellige grunde, både uden og med hhv. 1, 2, 3, 4 huse samt hotel.

```
/*to dimentionelt array af leje i orden af:
{grund, hus, 2 huse, 3 huse, 4 huse, hotel}*/
private int [][] streetRent = {
        {50, 250, 750, 2250, 4000, 6000},
        {50, 250, 750, 2250, 4000, 6000},
        {100, 600, 1800, 5400, 8000, 11000},
        {100, 600, 1800, 5400, 8000, 11000},
        {150, 800, 2000, 6000, 9000, 12000},
        {200, 1000, 3000, 9000, 12500, 15000},
        {200, 1000, 3000, 9000, 12500, 15000},
        {250, 1250, 3750, 10000, 14000, 18000},
        {300, 1400, 4000, 11000, 15000, 19000},
        {300, 1400, 4000, 11000, 15000, 19000},
        {350, 1600, 4400, 12000, 16000, 20000},
        {350, 1800, 5000, 14000, 17500, 21000},
        {350, 1800, 5000, 14000, 17500, 21000},
        {400, 2000, 6000, 15000, 18500, 22000},
        {450, 2200, 6600, 16000, 19500, 23000},
        {450, 2200, 6600, 16000, 19500, 23000},
        {500, 2400, 7200, 17000, 20500, 24000},
        {550, 2600, 7800, 18000, 22000, 25000},
        {550, 2600, 7800, 18000, 22000, 25000},
        {600, 3000, 9000, 20000, 24000, 28000},
        {700, 3500, 10000, 22000, 26000, 30000},
        {1000, 4000, 12000, 28000, 34000, 40000}};
```

I denne klasse har vi også et array med alle felterne i med navn, nummer, pris, farve og type (street, shipping, chance osv.).
Vi har også et array med alle chance kortene.

```
private ChanceCard[] generateStandartCardList()
{
    ChanceCard[] cards = new ChanceCard[33];

    cards[0] = new MoneyGift    (1,  "De modtager Deres aktieudbytte. Modtag kr. 1000 af banken. ", 1000);
    cards[1] = new MovedToField (2,  "De rykkes til start", 1);
    cards[2] = new GoToJailCard     (3,  "Gå i fængsel. Ryk direkte til fængslet. Selv om De passerer Start,"
          + " indkassere de ikke kr. 4000 ");
    cards[3] = new GoToJailCard     (4,  "Gå i fængsel. Ryk direkte til fængslet. Selv om De passerer Start,"
          + " indkassere de ikke kr. 4000 ");
    cards[4] = new Fine         (5,  "De har været en tur i udlandet og haft for mange cigaretter med "
          + "hjem. Betal told kr. 200", 200);
    cards[5] = new Fine         (6,  "De har modtaget Deres tandlægeregning. Betal kr. 2000", 2000);
    cards[6] = new MoneyGift    (7,  "De havde en række med elleve rigtige i tipning. Modtag kr. 1000", 1000);
    cards[7] = new MoneyGift    (8,  "Deres præmieobligation er kommet ud. De modtager kr. 1000 af banken.", 1000);
    cards[8] = new MoneyGift    (9,  "Deres præmieobligation er kommet ud. De modtager kr. 1000 af banken.", 1000);
    cards[9] = new MoneyGift    (10, "Det er Deres fødselsdag. Modtag af hver medspiller kr. 200", 0);
    cards[10] = new MoneyGift   (11, "Værdien af egen avl fra nyttehaven udgør kr. 200, som De modtager af banken. ", 200);
    cards[11] = new Fine        (12, "Betal Deres bilforsikring kr. 1000" , 1000);
    cards[12] = new Fine        (13, "Ejendomsskatterne er steget, ekstraudgifterne er: kr 800 pr. hus, kr 2300 pr. hotel.", 0);
    cards[13] = new MovedToField(14, "Ryk frem til Grønningen. Hvis De passerer Start inkassér da kr. 4000 ", 25 );
    cards[14] = new MovedToField(15, "Ryk brikken frem til det nærmeste rederi og betal ejeren to gange den "
          + "leje, han ellers er berettiget til. Hvis selskabet ikke ejes af nogen  kan De købe det af banken. ", 0);
    cards[15] = new MovedToField(16, "Ryk brikken frem til det nærmeste rederi og betal ejeren to gange den "
          + "leje, han ellers er berettiget til. Hvis selskabet ikke ejes af nogen  kan De købe det af banken. ", 0);
    cards[16] = new MovedToField    (17, "Tag med Mols-Linjen --- flyt brikken frem, og hvis De passerer Start"
          + " inkassér da kr. 4000 ", 16);
    cards[17] = new JailSafed   (18, "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.");
    cards[18] = new JailSafed   (19, "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");
    cards[19] = new MoneyGift   (20, "Grundet dyrtiden har De fået gageforhøjelse. Modtag kr. 1000 ", 1000);
    cards[20] = new MovedToField(21, "Ryk frem til Frederiksberg Allé. Hvis De passerer Start inkassér kr. 4000 ", 12);
    cards[21] = new MoneyGift   (22, "De har vundet i Klasselotteriet. Modtag kr. 500 ", 500);
    cards[22] = new MovedToField(23, "Tag ind på Rådhuspladsen.", 40 );
    cards[23] = new MovedToField(24, "Ryk tre felter tilbage.", -3);
    cards[24] = new Fine        (25, "Oliepriserne er steget, og De skal betale: kr. 500 pr. hus,  kr. 2000 pr. hotel.", 0);
```

MoveController klassen er, hvor spillerens tur bliver håndteret. Hvis en spiller sidder i fængsel når det bliver hans tur, vil en metode i denne klassen prøve at få spilleren ud, enten ved hjælp af et “Get Out Of Jail” kort eller ved betaling. Ellers kan spilleren komme ud ved at slå 2 ens med terningerne.
Det er også her at en spiller får sin “start bonus” når han passere start. Nedenfor kan man se, hvordan vi har besluttet at løse denne opgave.

```
private void movePlayer(Player currentPlayer)
{
    if (currentPlayer.getPosition() + diceCup.getSum() > 40)
    {
        currentPlayer.setPlayerPosition(currentPlayer.getPosition() + diceCup.getSum() - 40);
        try
        {
            currentPlayer.getAccount().deposit(START_BONUS);
        }
        catch (IllegalAmountException e)
        {
            System.err.println(e.getMessage());
            e.printStackTrace();
            System.exit(0);
        }
    }
    else
        currentPlayer.setPlayerPosition(currentPlayer.getPosition() + diceCup.getSum());
}
```

Player klassen er hvor en spiller bliver oprettet med alle de variabler der skal bruges.

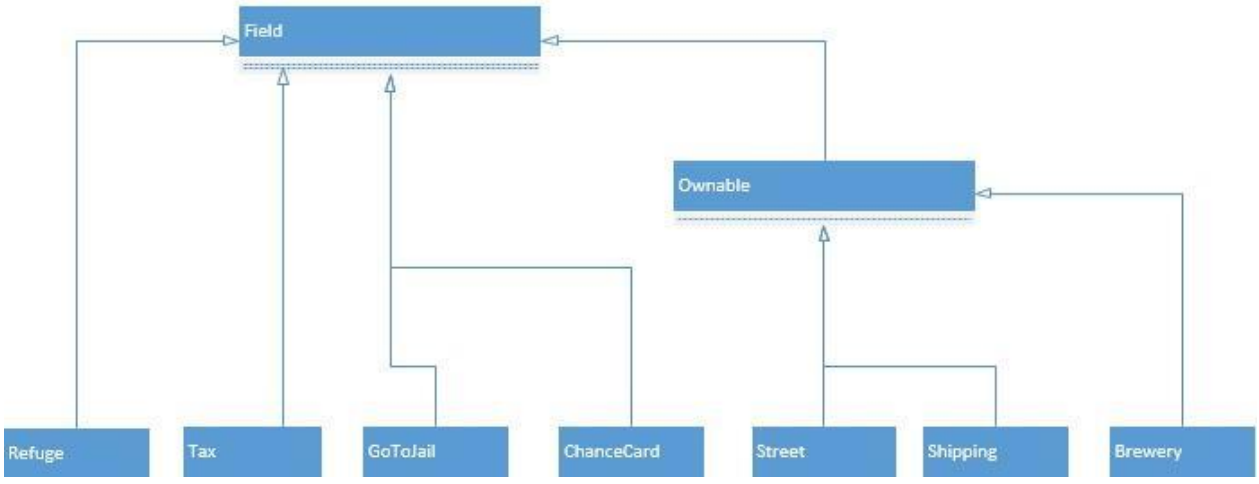
Der er også lavet en get og set metode til de enkelte variabler, disse bliver brugt af FieldController og MoveController.

### Game.chance_cards

I mappen Game.chance_cards har vi en ChanceCard klasse samt klasserne Fine, GoToJail, JailSaved, MoneyGift og MovedToField. Disse er alle nedarvet fra ChanceCard, da alle chancekortene har som udgangspunkt samme grunddata men med forskellige metoder.
Selve metoderne bliver håndteret i FieldController, og chance kortenes data er i GameData klassen.

### Game.fields

Vores pakke Game.fields rummer 9 klasser. I disse klasser har vi inddelt vores forskellige felter fra spillet. Felterne har forskellige funktioner, og derfor er der forskellige handlinger der skal tages hensyn til.
Vi har her for overskuelighedens skyld lavet et arvehieraki. Vi har i spillet 7 forskellige typer af felter, hvor nogle af felterne har samme funktioner.
Nedenunder er det illustreret, hvilke klasser der arver fra hvem.
Field er superklassen som alle andre arver fra, fælles for dem alle er at de har et navn og et nummer. Sub-klasserne til Field er: Refuge, Tax, GoToJail, ChanceCard og Ownable.
Ownable er også en super klasse, og fælles for de klasser/felter, der arver fra Ownable, er at disse kan alle købes, de har en pris og de kan lejes, hvis altså de er ejet. Der er tre sub-klasser til Ownable: Street, Shipping og Brewery. De er dog, hver især forskellige og har derfor separate klasser.




### Gui

Gui og den efterfølgende pakke startMenuGui er bygget af Magnus Brandt Sløgedal sideløbende med det foregående kursus (02313 Udviklingsmetoder til IT-Systemer), og vi betragter dem i dette projekt som ekstra.
Der er 2 klasser i gui der er relevante for projektet, den første af disse er GUI. GUI er indgangen til resten af gui-mappen, og den har metoder til at ændre hvad der vises på brugerfladen, den

reffere til GameGUI via en privat singleton refference, og sikrer således at der altid kun findes 1 GUI.
Den anden vigtige klasse er  GameActionListener, denne er samlepunkt for alle ActionEvents der sendes fra GUI'en, og den har metoder til at behandle disse, det er primært knapperne på GUI'en, samt et windowClosing event. Herudover kalder den GUI for at opdatere efter hvert event.

### StartMenuGui

StartMenuGui indeholder en custom dialog StartMenuDialog, der er bygget til at levere os et GameData() object med de nødvendige data for at starte spillet. Den er bygget ved at extende en JDialog, og udnytte dens modale egenskaber til at returnere en værdi når dialogen lukkes.

## MySQL database

I opgavebeskrivelsen er det et krav at Matadorspillet skal kunne gemmes. Det for at spillerne senere kan genoptage spillet. Vi gemmer spillet i en MySQL database.
Databasen har to tabeller "Felter" og "player".


I tabellen "Felter" har vi fire kolonner, hvor "felt_nummer" er primær nøgle. kolonne tre "felt_ejer" er fremmednøgle til "Player" tabellen, den viser hvem der er ejer af et givent felt. Har feltet værdien "null", har feltet ikke nogen ejer. Har feltet en ejer, vil det være spillerens id altså "id_player", "Player" tabellens primærnøgle der er værdien i "felt_ejer". Der ud over har tabellen kolonnerne "felt_navn" og "antal_bygninger", "felt_navn" har navnet på det givne felt f.eks. "Hvidovrevej" eller "Rådhuspladsen". Kolonnen "antal_bygninger" gemmer, hvor mange bygninger der er på det givne felt, fra null til 5 bygninger. Hvor 1, 2, 3, 4 er antallet af huse og 5 er et hotel.
Tabellen Player har "id_player" som primær nøgle, og "player_name" gemmer spillerens navn. Der bliver også gemt, hvilken position spilleren har på pladen, hvor stor spillerens pengebeholdning er. Hvis spilleren er i fængsel bliver der gemt, hvor mange omgange spilleren

har siddet der. Der bliver også gemt, hvor mange "get_out_of_jail_cards" en spiller har, ligesom bil farve, bil type og selvfølgelig også, hvis tur det er.

Til at tilgå databasen har vi udviklet klassen DBAccess, denne bruger preparedstatements til at hente og skrive data til en SLQ database vi har kørende på en gigahost-server. Denne server er i øjeblikket hardcoded i klassen. DBAccess har metoder til at resette databasen, gemme spiller- og felt-data, og loade spiller- og felt-data.
Vi har lavet en DBCommunication klasse til at håntere den DBAccess, og bruge den til at sende og modtage vores data, i to statiske metoder henholdsvis saveGame() og loadGame().

### Save Game

For at gemme et spil har vi behov for at resette databasen, dette gør vi for players ved at kalde

```
DELETE * FROM player;
ALTER TABLE player AUTO_INCREMENT = 1;
```

dette sletter alle data, og resetter primary key counteren til 1

for felterne kalder vi:

```
UPDATE fields SET field_owner = NULL , number_of_houses = NULL;
```

da felterne ikke ændres voldsomt har vi valgt blot at opdatere deres data i stedet for at slette dem helt.
Da vi nu har databasen på vores default form, kan vi indsætte data fra vores nuværende spil. Vi starter med at indsætte spillerne, da de er foreign-key i felternes field_owner row, Vi indsætter en ny spiller af gangen med:

```
INSERT INTO player (player_name, field_position, account_balance,
prison_turn, get_out_of_jail_cards, car_color, car_type,
player_turn) VALUES (?,?,?,?,?,?,?,?);
```

indtil vi har alle spillerne, "?" her er en del af prepared-statement syntaksen, og bliver erstatte i java med de korresponderende data.
Til sidst opdatere vi felterne med nye ejere og bygninger:

```
UPDATE fields SET field_owner = (SELECT id_player FROM player
where player_name = ?) WHERE field_number = ?;
```

her henter vi de ønskede player id, ud fra player_name for at opfylde vores foreign-key refference.

### Load Game

For at loade et spil henter vi alle data fra player-skemaet, og alle ændrede data fra felt-skemaet. dette gøres for player temmelig simplet med:

```
SELECT * FROM player;
```

Dette giver os alle data, og da vi skal bruge dem alle er der ingen grund til at vælge mere fra. Vores select fra fields bliver lidt mere kompliceret, da vi her ønsker at få den refferede owner med, vi bliver derfor nød til at joine de to databaser:

```
SELECT field_number, player_name, number_of_houses FROM fields
JOIN player ON fields.field_owner = player.id_player WHERE
field_owner IS NOT null;
```

dette giver os felt nummeret, som vi bruger som reference for at placere ting rigtigt på java-side, navnet på ejeren, samt antallet af huse.

# Test

Vi har hovedsageligt tested vores program via Brugertests, dog har vi også anvendt JUnit test klasser til specifikke ting, og til at nærstudere fejl. Vi har brugt eclipses indbyggede debugging tool, til at analysere fejl.

## Uddybning af black box testing via JUNIT

Der er foretaget 3 af denne type tests.

**1 ChanceCardTest**

En vigtig del af spillet er chance kortene. Der er utrolig mange af slagsen og de har næsten alle sammen forskellig funktionalitet. Med disse forudsætninger er der dømt til at være noget der går galt når der skal hard-codes så meget.

Derefter er der valgt at lave en test, der simulerer at man lander på et chancekortfelt, hvorefter man får at vide hvad kortet gør og derefter får udprintet alle potentielle ændringer.

Alle chancekort ændre på en eller flere af følgende parametre:

- Player position
- Player Balance
- Player get out of jail cards

Derfor vil der i testens udprint kunne ses at der er sket en forkert ændring eller ingen ændring ift. hvad der stod på kortet.

**2 DatabaseTest**

Databasen er en af de vigtigste elementer af spillet, da de lange matador spil skal kunne gemmes. Testen er simpel og tester hvorvidt det potentielt set gemte data er gemt og kan loades.

**3 testBrewery**

Under perioden at spillet blev programmeret opstod der flere NullPointerExeceptions, hvoraf dette var en af dem. Derfor blev det besluttet af lave en specifik test til at løse problemet da der kom en fejl i konsollen hver gang spillet blev kørt.

## Brugervejledning og eksempel

Programmet er bygget til og testet på en windows-maskine, hvorfor dette som udgangspunkt forudsættes.

Følgende er fremgangsmåden på en computer, der kører windows, der allerede har programmet Eclipse installeret.

*1)* ***Åben filhåndteringsprogrammet***
Start programmet Eclipse
Dette kan tage lidt tid.

*2)* **Importer filen**
I toolbaren i programmet vælger du følgende:
File > Import > Import Existing Projects Into Workspace > Select Archive
File > Browse > (vent et øjeblik) vælg Matador_gr04.zip > Finish

Spillet er importeret ind i din lokale version af eclipse på din computer, også kaldet “Workspace”. Koden bagom spillet vil nu kunne ses i venstre side af programmet under ”Package Explorer” som “Matador_2”.

*3)* **Kør spillet**
Tast enten ”Ctrl + F11”, eller åben mappe systemet matador_2-->src→(default package), hvorefter du vil se en fil ved navn: “Main.java”. Højreclick på filen vælg “Run as” og dernæst “Java Application”.

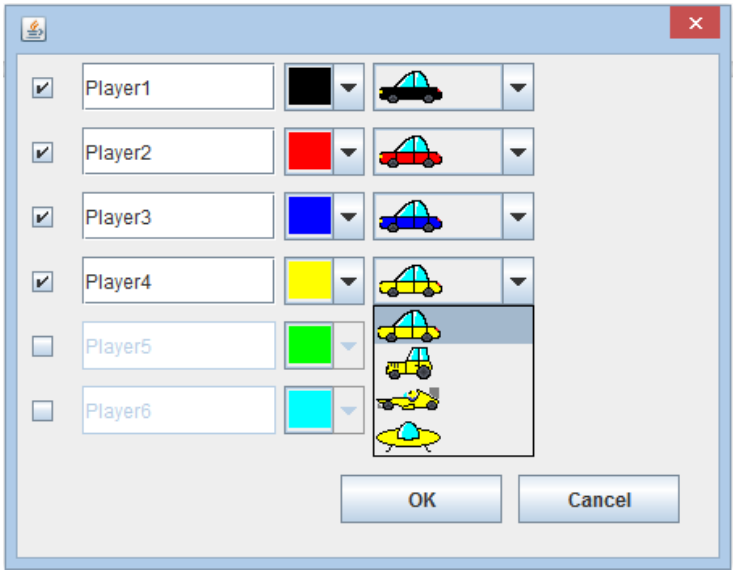
Spillet skullet nu meget gerne poppe up som en box externt fra eclipse.

Vi har valgt at benytte os af en online database, der er offentlig tilgængelig, hvorfor der ikke skal bekymres om data. Ønsker du at oprette tabellerne i din egen personlige mySQL kan du benytte SQL scriptet der er kan ses i Bilag a. Dette script er også vedlagt som Database.sql, i projectmappen.

## Brugervejledning

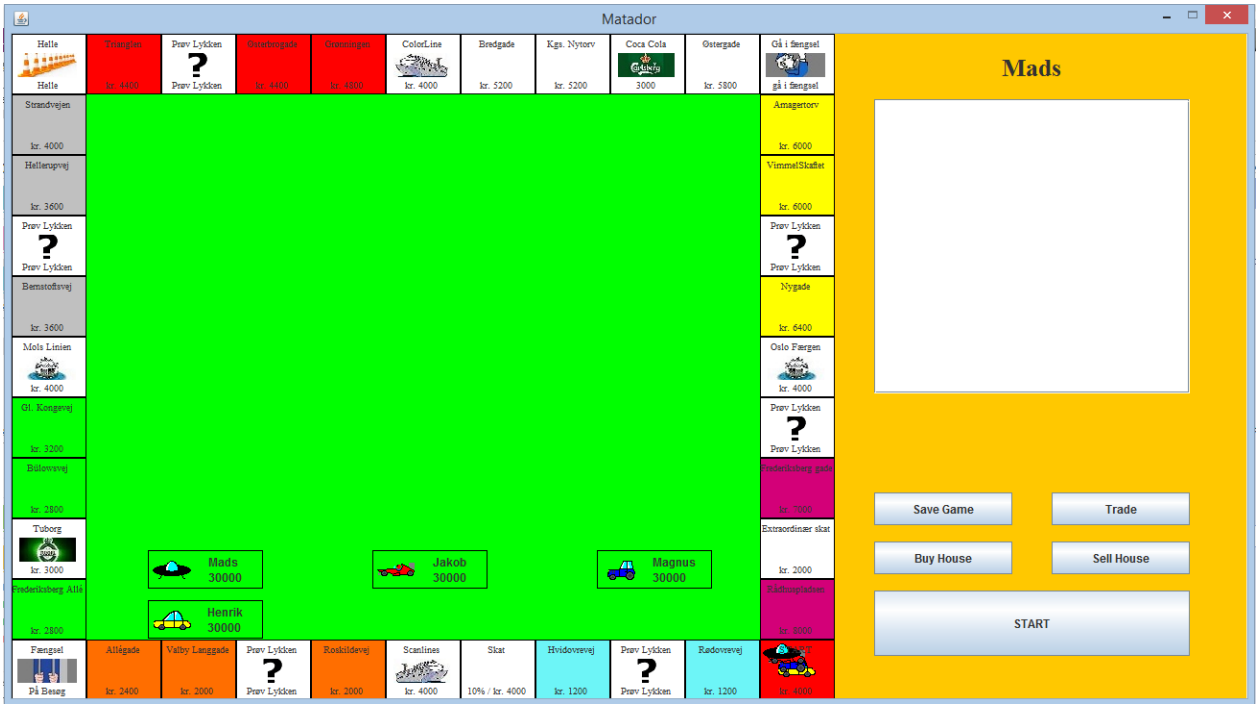
Følgende er en kort gennemgang af programmet.
Ved programmets start vil man se en menu.
Det åbenlyse valg er enten her start new game eller load game. Trykker man load game springer man næste trin over.


Trykker man start new game kommer man til initialiseringen af spillet, hvor man kan vælge type af bil,farve, navnet på sine spillet og antallet af spillere.

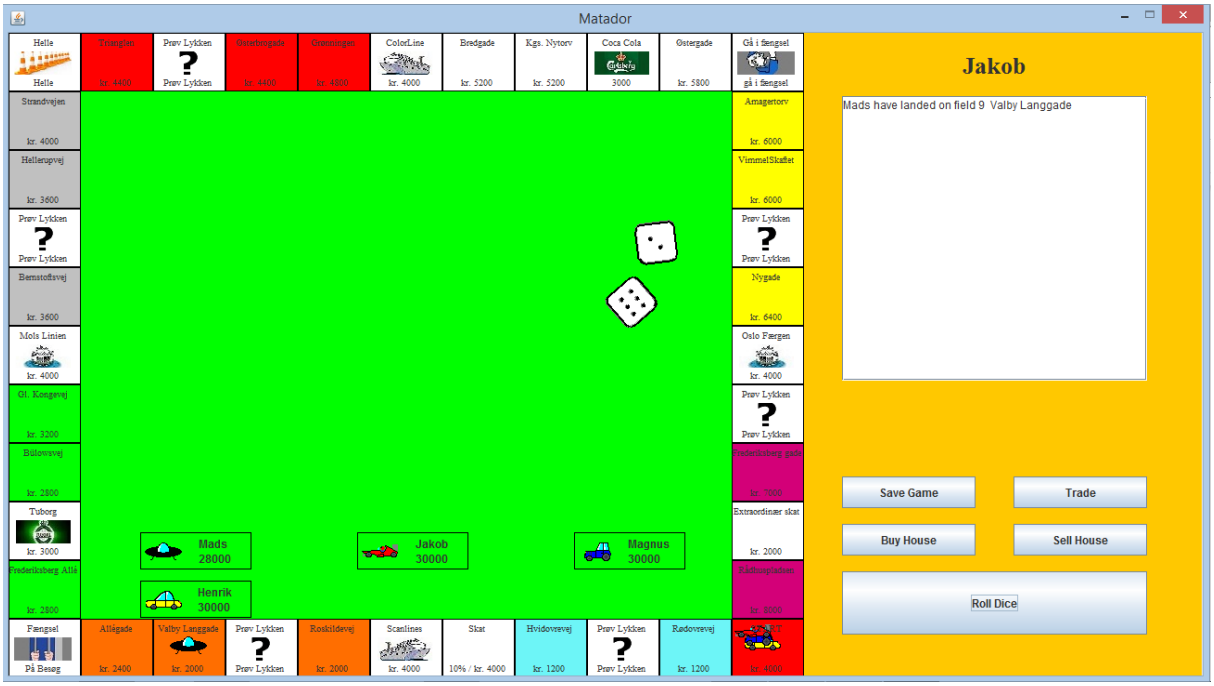



Når dette er gjort trykker man OK.

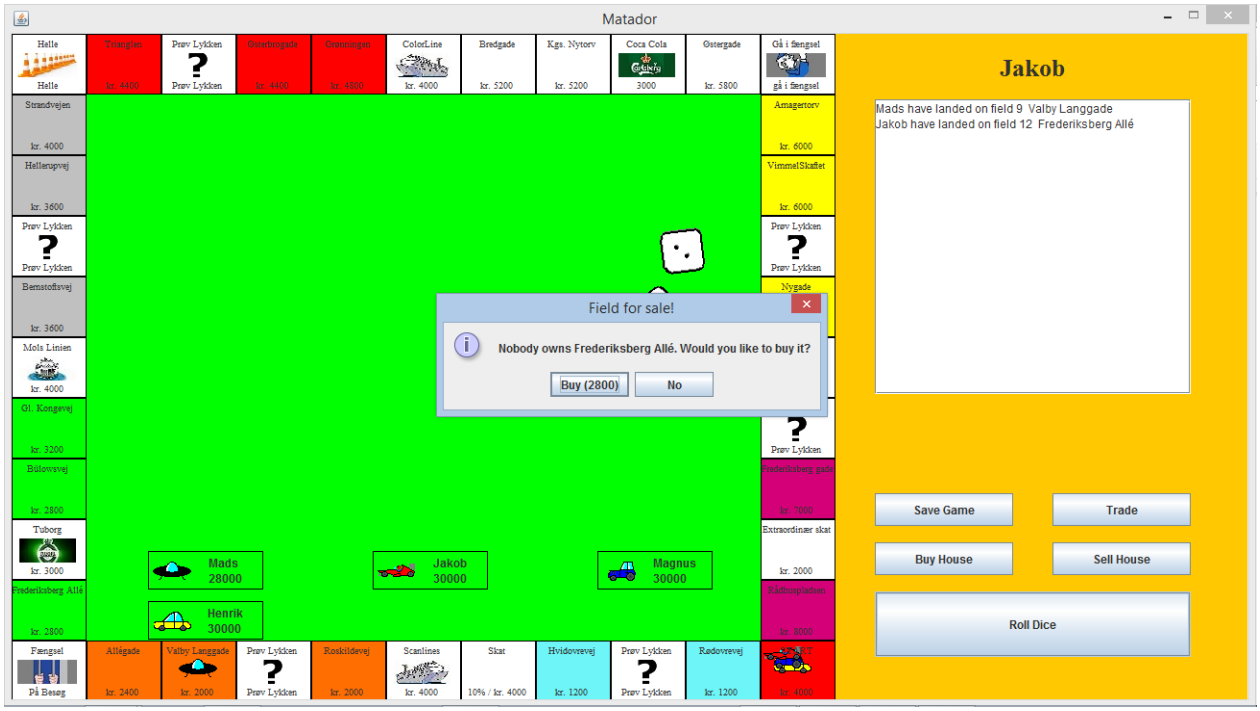
Når der er blevet trykket OK kommer man ind i spillet. Ligeså snart man trykker start vil første spillers tur starte.
Hvis man har loadet et spil vil man når man trykker start starte den spillers tur spillet blev lukket ved og brættet se identisk ud med da man gemte spillet.



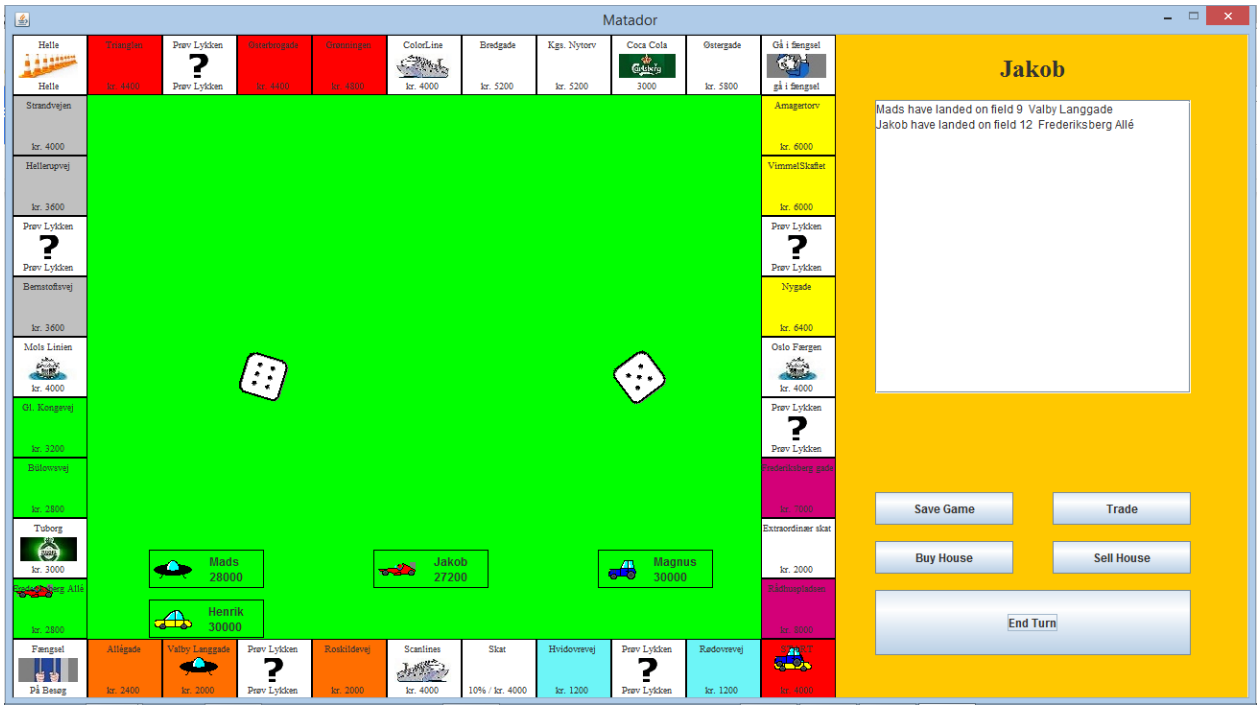

Som player kan man følge med i hvad der sker ved at kigge i chatdialogen i øverst højre side og benytte sig af funktionerne nederst i højre side.
Primær knappen skifter navn og er lige nu roll Dice, hvilket ved klik flytter spilleren, ændren placeringen og værdierne på terningerne (medmindre spilleren slår det samme).



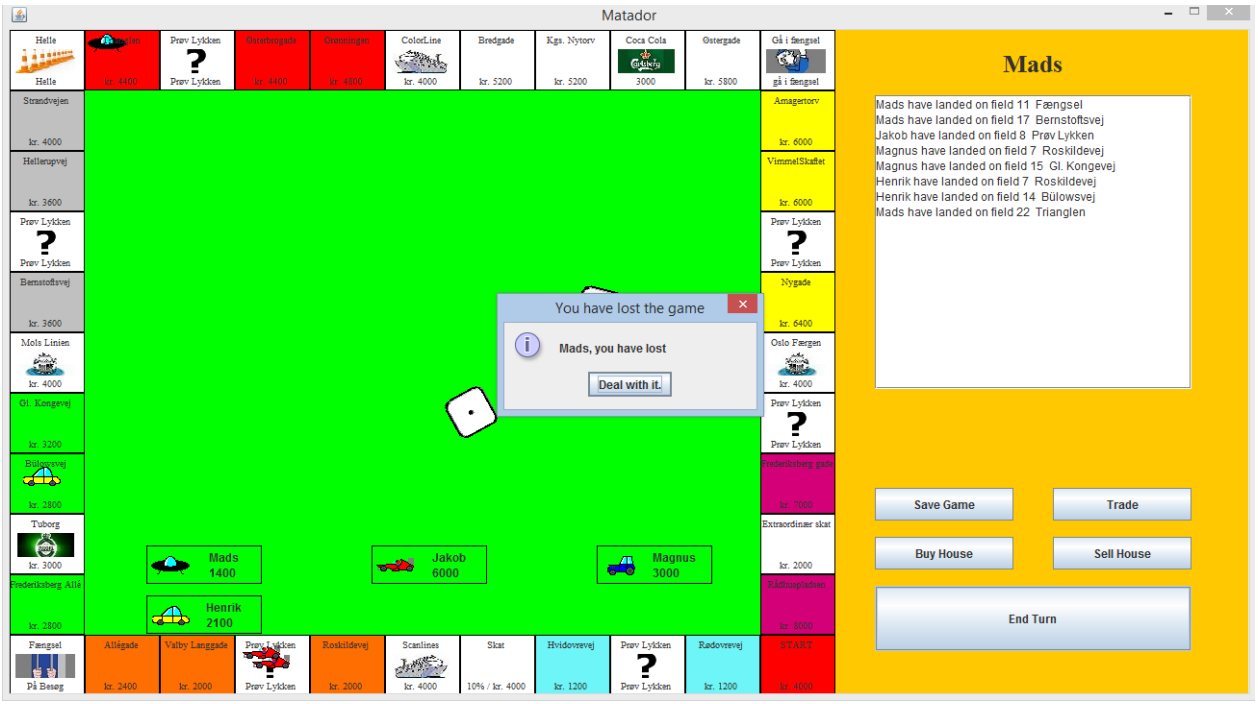

Når spilleren har rullet med terningerne vil der afhængigt af feltet komme en tilhørende dialogbox.




Når man ikke hverken vil købe huse, har rullet med terningen de gange man vil osv. trykker man End Turn.




Taber en spiller spillet vil der komme en dialogbox op fortæller spilleren dette.




Samtidig vil en spiller få en box hvis spilleren vinder og den primære knap vil skrive game over.

# Konklusion

Vi har I løbet af dette projekt bygget et matadorspil i fire løst definerede iterationer. Første iteration fokuserede på at få spillet op og stå, med helt basis funktionalitet, så som at flytte biler, skifte spiller, her foregik en stor del af vores designarbejde. Anden iteration fokuserede på at tilføje detaljer, så som køb af felter, fængsling, og falit. Tredje iteration fokuserede på at designe og tilføjede SQL funktionalitet og færdiggøre tidligere funktioner. Sidste iteration har fokuseret på at implementere huse, og rette fejl i koden.

Vi har fundet en del fejl i vores program, som vi endnu ikke har fået rettet, dette er primært på grund af tidsbegrænsninger:

- Fejl i forbindelse med huse:
    - Tegn hus på brættet
    - Slet hus-billede når nyt købes.
    - Sørg for at man kun kan bygge huse når man har alle felter i en farve.
    - Sørg for man kun kan købe 5 huse pr felt.
    - Sørg for man kun kan købe huse jævnt distribueret over en farve.
- Fallit spillere skal fjernes fra GUI.
- Bilerne "hopper" rundt på de individuelle felter

Til trods for fejlene er vi kommet ret tæt på en simulering af et matador spil. Der er dog stadig en del ting der kunne forbedres. Det første ville være at flytte visse hardkodede værdier ud i en `.properties` fil eller database, dette gælder ting som startkapitalen på et nyt spil eller url'en til SQL-databasen. Der er også enkelte designelementer der kunne ændres, der er som eksempel en del statiske elementer i designet der godt kunne trænge til at blive tænkt igennem en extra gang.

# Appendiks

## Bilag a - SQL Script

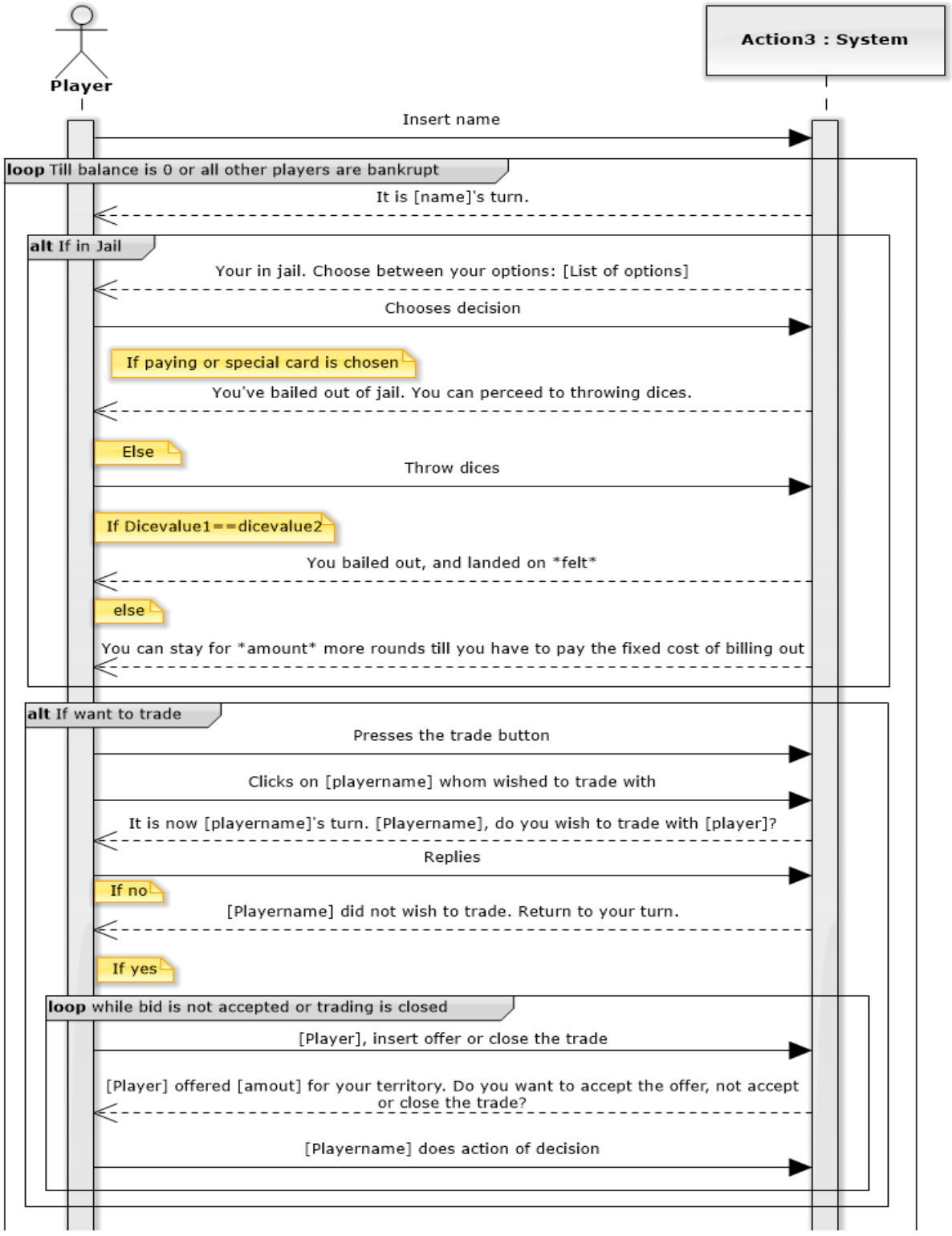
her er vedlagt et script der opretter et nyt schema kaldet jakobsabinsky_matador der har de nødvendige tabeller fields og player. Dette script er også tilgængelig som Database.sql i projectmappen.
Scriptet er lavet ved at dumpe databasen ved hjælp af værktøjet HeidiSQL

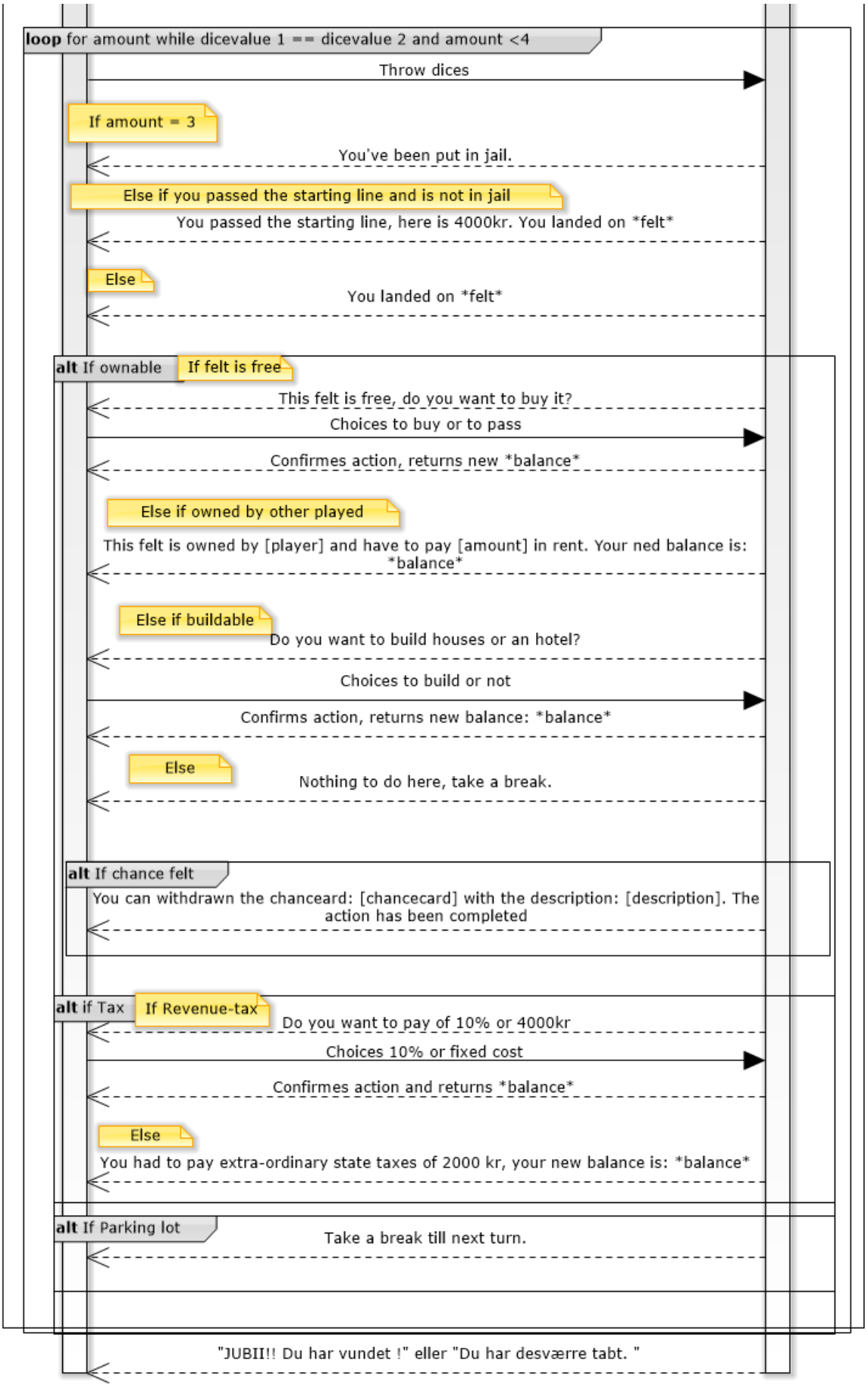
```
-- --------------------------------------------------------
-- Host:                         mysql7.gigahost.dk
-- Server version:               5.1.73-1-log - (Debian)
-- Server OS:                    debian-linux-gnu
-- HeidiSQL Version:             8.3.0.4694
-- --------------------------------------------------------

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET NAMES utf8 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO'
*/;

-- Dumping database structure for jakobsabinsky_matador
DROP DATABASE IF EXISTS `jakobsabinsky_matador`;
CREATE DATABASE IF NOT EXISTS `jakobsabinsky_matador` /*!40100 DEFAULT
CHARACTER SET latin1 */;
USE `jakobsabinsky_matador`;


-- Dumping structure for table jakobsabinsky_matador.fields
DROP TABLE IF EXISTS `fields`;
CREATE TABLE IF NOT EXISTS `fields` (
  `field_number` int(11) NOT NULL,
  `field_name` varchar(45) DEFAULT NULL,
  `field_owner` int(11) DEFAULT NULL,
  `number_of_houses` int(11) DEFAULT NULL,
  PRIMARY KEY (`field_number`),
  KEY `field_owner_idx` (`field_owner`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

-- Dumping data for table jakobsabinsky_matador.fields: 40 rows
DELETE FROM `fields`;
/*!40000 ALTER TABLE `fields` DISABLE KEYS */;
```

```
INSERT INTO `fields` (`field_number`, `field_name`, `field_owner`,
`number_of_houses`) VALUES
     (1, 'Start', NULL, NULL),
     (2, 'Rødovrevej', NULL, NULL),
     (3, 'Prøv Lykken', NULL, NULL),
     (4, 'Hvidovrevej', NULL, NULL),
     (5, 'SKAT', NULL, NULL),
     (6, 'Scanlines', 1, NULL),
     (7, 'Roskildevej', NULL, NULL),
     (8, 'Prøv Lykken', NULL, NULL),
     (9, 'Valby Langgade', NULL, NULL),
     (10, 'Allégade', 1, NULL),
     (11, 'Fængsel', NULL, NULL),
     (12, 'Frederiksberg Allé', 4, NULL),
     (13, 'Tuborg', NULL, NULL),
     (14, 'Bülowsvej', NULL, NULL),
     (15, 'Gl. Kongevej', NULL, NULL),
     (16, 'Mols Linien', NULL, NULL),
     (17, 'Bernstofsvej', 4, NULL),
     (18, 'Prøv Lykken', NULL, NULL),
     (19, 'Hellerupvej', NULL, NULL),
     (20, 'Strandvejen', NULL, NULL),
     (21, 'Helle', NULL, NULL),
     (22, 'Trianglen', NULL, NULL),
     (23, 'Prøv Lykken', NULL, NULL),
     (24, 'Østerbrogade', NULL, NULL),
     (25, 'Grønningen', NULL, NULL),
     (26, 'Color Line', NULL, NULL),
     (27, 'Bredgade', NULL, NULL),
     (28, 'Kgs. Nytorv', NULL, NULL),
     (29, 'Coca Cola', NULL, NULL),
     (30, 'Østergade', NULL, NULL),
     (31, 'Gå i fængsel', NULL, NULL),
     (32, 'Amagertorv', NULL, NULL),
     (33, 'Vimmelskaftet', NULL, NULL),
     (34, 'Prøv Lykken', NULL, NULL),
     (35, 'Nygade', NULL, NULL),
     (36, 'Oslo Færgen', NULL, NULL),
     (37, 'Prøv Lykken', NULL, NULL),
     (38, 'Frederiksberg gade', NULL, NULL),
     (39, 'Extraordinær skat', NULL, NULL),
     (40, 'Rådhuspladsen', NULL, NULL);
/*!40000 ALTER TABLE `fields` ENABLE KEYS */;
```

```
-- Dumping structure for table jakobsabinsky_matador.player
DROP TABLE IF EXISTS `player`;
CREATE TABLE IF NOT EXISTS `player` (
  `id_player` int(11) NOT NULL AUTO_INCREMENT,
  `player_name` varchar(45) DEFAULT NULL,
  `field_position` int(11) NOT NULL,
  `account_balance` int(11) DEFAULT NULL,
  `prison_turn` int(11) DEFAULT NULL,
  `get_out_of_jail_cards` int(11) DEFAULT NULL,
  `car_color` int(11) NOT NULL,
  `car_type` int(11) NOT NULL,
  `player_turn` int(11) DEFAULT NULL,
  PRIMARY KEY (`id_player`)
) ENGINE=MyISAM AUTO_INCREMENT=5 DEFAULT CHARSET=latin1;

-- Dumping data for table jakobsabinsky_matador.player: 4 rows
DELETE FROM `player`;
/*!40000 ALTER TABLE `player` DISABLE KEYS */;
INSERT INTO `player` (`id_player`, `player_name`, `field_position`,
`account_balance`, `prison_turn`, `get_out_of_jail_cards`, `car_color`,
`car_type`, `player_turn`) VALUES
     (1, 'Player3', 10, 23600, 0, 0, -16776961, 0, 0),
     (2, 'Player4', 8, 31000, 0, 0, -256, 0, 1),
     (3, 'Player1', 12, 25800, 0, 0, -16777216, 0, 2),
     (4, 'Player2', 17, 23800, 0, 0, -65536, 0, 3);
/*!40000 ALTER TABLE `player` ENABLE KEYS */;
/*!40101 SET SQL_MODE=IFNULL(@OLD_SQL_MODE, '') */;
/*!40014 SET FOREIGN_KEY_CHECKS=IF(@OLD_FOREIGN_KEY_CHECKS IS NULL, 1,
@OLD_FOREIGN_KEY_CHECKS) */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
```

**Bilag B - SSD**


Player
Action3 : System
Insert name
loop Till balance is 0 or all other players are bankrupt
It is [name]'s turn.
alt If in Jail
Your in jail. Choose between your options: [List of options]
Chooses decision
If paying or special card is chosen
You've bailed out of jail. You can perceed to throwing dices.
Else
Throw dices
If Dicevalue1==dicevalue2
You bailed out, and landed on *felt*
else
You can stay for *amount* more rounds till you have to pay the fixed cost of billing out
alt If want to trade
Presses the trade button
Clicks on [playername] whom wished to trade with
It is now [playername]'s turn. [Playername], do you wish to trade with [player]?
Replies
If no
[Playername] did not wish to trade. Return to your turn.
If yes
loop while bid is not accepted or trading is closed
[Player], insert offer or close the trade
[Player] offered [amout] for your territory. Do you want to accept the offer, not accept or close the trade?
[Playername] does action of decision


loop for amount while dicevalue 1 == dicevalue 2 and amount <4
Throw dices
If amount = 3
You've been put in jail.
Else if you passed the starting line and is not in jail
You passed the starting line, here is 4000kr. You landed on *felt*
Else
You landed on *felt*
alt If ownable
If felt is free
This felt is free, do you want to buy it?
Choices to buy or to pass
Confirmes action, returns new *balance*
Else if owned by other played
This felt is owned by [player] and have to pay [amount] in rent. Your ned balance is: *balance*
Else if buildable
Do you want to build houses or an hotel?
Choices to build or not
Confirms action, returns new balance: *balance*
Else
Nothing to do here, take a break.
alt If chance felt
You can withdrawn the chanceard: [chancecard] with the description: [description]. The action has been completed
alt if Tax
If Revenue-tax
Do you want to pay of 10% or 4000kr
Choices 10% or fixed cost
Confirmes action and returns *balance*
Else
You had to pay extra-ordinary state taxes of 2000 kr, your new balance is: *balance*
alt If Parking lot
Take a break till next turn.
"JUBII!! Du har vundet !" eller "Du har desværre tabt. "

## Bilag C - JAVA source code

Her er vedlagt Kildekoden for projectet, vi har dog udeladt en del af gui koden, da GUI'en som tidligere nævnt ikke anses som en reel del af projectet. Skulle man alligevel ønske at se Gui'en kildekode er den frit tilgængelig i den online aflevering.

---
Main.java
---

```
import game.GameController;
import game.GameData;
import gui.GUI;

import javax.swing.SwingUtilities;

import startmenugui.StartMenuDialog;

public class Main
{
	public static void main(String[] args)
	{
		SwingUtilities.invokeLater(new Runnable()
		{
			@Override
			public void run()
			{
				//Do everything in here, this means everything is on
the EDT-thread
				//if large tasks are required they should be
dispatched in another thread
				//to not block the EDT (Blocked EDT = frozen GUI).
				GameData options = new
StartMenuDialog().startDialog();
				GameController.createInstance(options);
				new GUI().create();
			}
		});
	}
}
```

---
GameController.java
---

```
package game;

import game.fields.Field;
import gui.GUI;
import java.util.LinkedList;

public class GameController
{
```

```
	//-------------------------------------------------------
	//Singleton design pattern
	//-------------------------------------------------------
	private static GameController instance = null;

	/**
	 * Used to return the GameController object.
	 * @return The Gamecontroller Instance.
	 */
	public static GameController getInstance()
	{
		if(instance == null)
			System.err.println("Error no instance exists");
		return instance;
	}
	/**
	 * Creates a Gamecontroller with a specific options object, this MUST be
called
	 * before any calls to getInstance to have any effect.
	 * @param options A GameOptions object containing startup data for the
game.
	 * @return The Gamecontroller Instance.
	 */
	public static GameController createInstance(GameData options)
	{
		if(instance == null)
			instance = new GameController(options);
		else
			System.err.println("GameController instance already
exists!!!");
		return instance;
	}
	//-------------------------------------------------------

	public static final int ROLL_STATE = 0;
	public static final int END_TURN_STATE = 1;
	public static final int GAME_OVER_STATE = 2;
	private int mainButtonState = 0;

	private DiceCup dice;
	private Board board;
	private Player currentPlayer;
	private MoveController moveController;
	private FieldController fieldController;
	private LinkedList<Player> playerQueue;
	private GUI gui = new GUI();

	private GameController(GameData data)
	{
```

```
            this.dice = data.getDice();
            this.board = new Board(data.getFields(), data.getCards());
            this.moveController = new MoveController(data.getDice());
            this.fieldController = new FieldController(board);

            this.playerQueue = new LinkedList<Player>();
            for (int i = 0; i < data.getPlayers().length; i++)
            {
                  playerQueue.add(data.getPlayers()[i]);
            }

            this.currentPlayer = playerQueue.remove();

      }
      public void advanceGame()
      {

            if(mainButtonState != GAME_OVER_STATE)
            {
                  if(amountNotBroke() == 1)
                  {
                        mainButtonState = GAME_OVER_STATE;
                  }

                  if(currentPlayer.isBroke())
                  {
                        // Prints when a player looses
                        String[] options1 = { "Deal with it."};

                        int nrOfOptions = 1;
                        String[] options = new String[nrOfOptions];
                        for (int i = 0; i < options.length; i++)
                        {
                              options[i] = options1[i];
                        }

                        String message = getCurentPlayer().getName() + ", you
have lost";
                        while((gui.getUserButtonPressed(options, message, "You
have lost the game")) == -1);

                  }

                  if (mainButtonState == ROLL_STATE)
                  {
                        boolean playerBroke =
moveController.playerTurn(currentPlayer);
```

```
                        if(!playerBroke)
                              playerBroke =
fieldController.LandOnField(currentPlayer, currentPlayer.getPosition());

                        currentPlayer.setBroke(playerBroke);

                        if(!dice.isTwoOfAKind() || currentPlayer.isBroke() ||
currentPlayer.isInPrisson())
                              mainButtonState = END_TURN_STATE;
                  }
                  //This runs when the player end his turn
                  else if(mainButtonState == END_TURN_STATE)
                  {
                        if (!currentPlayer.isBroke())
                        {
                              //add player back to the queue
                              playerQueue.add(currentPlayer);
                        }
                        //pick next player from the queue
                        currentPlayer = playerQueue.remove();
                        currentPlayer.setTwoOfAKindRollCount(0);
                        mainButtonState = ROLL_STATE;
                  }
            }
            if(mainButtonState == GAME_OVER_STATE){
            // You have won message.
            String[] options1 = { "Proceed"};

            int nrOfOptions = 1;
            String[] options = new String[nrOfOptions];
            for (int i = 0; i < options.length; i++)
            {
                  options[i] = options1[i];
            }

            String message = getWinner().getName() + ", you have won!
Contragratulations";
            while((gui.getUserButtonPressed(options, message, "You have won
the game")) == -1);
            }
      }

      private int amountNotBroke()
      {
            int sum = 0;
            Player[] players = getAllPlayers();
            for (int i = 0; i <players.length; i++)
            {
                  if(!players[i].isBroke())
```

```
                  sum++;
      }
      return sum;
}

public Player getCurentPlayer()
{
      return currentPlayer;
}

public Player getWinner()
{
      Player[] players = getAllPlayers();
      int Avalue = 0;
      for (int i = 0; i <players.length; i++)
      {
            if(!players[i].isBroke())
                  Avalue = i;
      }
      return players[Avalue];
}

public Player[] getAllPlayers()
{
      Player[] players = new Player[this.playerQueue.size() + 1];

      players[0] = currentPlayer;

      for (int i = 0; i < this.playerQueue.size(); i++)
      {
            players[i+1] = playerQueue.get(i);
      }


      return players;
}

public int getCurrentState()
{
      return mainButtonState;
}

public DiceCup getDiceCup()
{
      return this.dice;
}

public Field[] getFields()
```

```
	{
		return this.board.getAllFields();
	}
}

-----------------------------------------------------------------------
MoveController.java
-----------------------------------------------------------------------
package game;

import game.Account.IllegalAmountException;
import game.Account.InsufficientFundsException;
import gui.GUI;

public class MoveController
{
	private DiceCup diceCup;
	private GUI gui;
	private final int START_BONUS = 4000;
	private final int BAIL_PRICE = 1000;

	public MoveController(DiceCup dice)
	{
		diceCup = dice;
		gui = new GUI();
	}

	/**
	 * @param currentPlayer The active player.
	 * @return true if the player has gone broke, false otherwise.
	 */
	public boolean playerTurn(Player currentPlayer)
	{
		//Check if player is in prison
		if (currentPlayer.isInPrisson() == true)
		{
			//try to get him out (cards and cash)
			jailOptions(currentPlayer);
		}
		//then roll some dice
		diceCup.rollDice();

		//did he roll two of a kind?
		if (diceCup.isTwoOfAKind())
		{

	currentPlayer.setTwoOfAKindRollCount(currentPlayer.getTwoOfAKindRollCoun
t() + 1);
		}
```

```
            //how many times did he roll two of a kind?
            if(currentPlayer.getTwoOfAKindRollCount() == 3)
            {
                  currentPlayer.setPlayerPosition(11);
                  currentPlayer.setInPrisson(true);
                  currentPlayer.setTwoOfAKindRollCount(0);
                  return false;
            }

            //Check again, did he get out?
            if (currentPlayer.isInPrisson() == false)
            {
                  //If he did, move his ass.
                  movePlayer(currentPlayer);
                  //and then land him on the new field.
            }
            return false;
      }

      private void jailOptions(Player currentPlayer)
      {
            {

                  String[] options1 = { "Roll dices", "Pay 1000kr", "Use
Chance Card"};

                  int nrOfOptions = 1;
                  if (currentPlayer.getAccount().getBalance() > BAIL_PRICE) //
Changed from >= to >, shouldn't it be like that?
                        nrOfOptions++;
                  if(currentPlayer.getGetOutOfJailCards() > 0)
                        nrOfOptions++;

                  String[] options = new String[nrOfOptions];
                  for (int i = 0; i < options.length; i++)
                  {
                        options[i] = options1[i];
                  }

                  int choise = -1;
                  String message = "You're in jail. Choose between these " +
options.length + " options";
                  while((choise = gui.getUserButtonPressed(options, message,
"Jail Options")) == -1);

                  if (choise == 0 )
                  {
```

```
                        diceCup.rollDice();

      currentPlayer.setPrisonTurnCount(currentPlayer.getPrisonTurnCount() +
1);

                        if (diceCup.isTwoOfAKind())
                        {
                              currentPlayer.setInPrisson(false);
                        }

                        if (currentPlayer.getPrisonTurnCount() == 3) {
                              currentPlayer.setPrisonTurnCount(0);

                              try
                              {
                                    currentPlayer.getAccount().withdraw(1000);
                                    currentPlayer.setInPrisson(false);
                              }
                              catch (InsufficientFundsException e)
                              {
                                    //should not be possible
                              }
                              catch (IllegalAmountException e)
                              {
                                    System.err.println(e.getMessage());
                                    e.printStackTrace();
                                    System.exit(0);
                              }
                        }

                  }

                  if (choise == 1)
                  {
                        // Withdraws 1000 from currentPlayer
                        try
                        {
                              currentPlayer.getAccount().withdraw(1000);
                              currentPlayer.setInPrisson(false);
                        }
                        catch (InsufficientFundsException e)
                        {
                              //should not be possible
                        }
                        catch (IllegalAmountException e)
                        {
                              System.err.println(e.getMessage());
                              e.printStackTrace();
                              System.exit(0);
```

```
                        }
                  }

                  if (choise == 2)
                  {
                        currentPlayer.setInPrisson(false);

      currentPlayer.setGetOutOfJailCards(currentPlayer.getGetOutOfJailCards()
- 1);
                  }
            }
      }

      private void movePlayer(Player currentPlayer)
      {
            if (currentPlayer.getPosition() + diceCup.getSum() > 40)
            {
                  currentPlayer.setPlayerPosition(currentPlayer.getPosition()
+ diceCup.getSum() - 40);
                  try
                  {
                        currentPlayer.getAccount().deposit(START_BONUS);
                  }
                  catch (IllegalAmountException e)
                  {
                        System.err.println(e.getMessage());
                        e.printStackTrace();
                        System.exit(0);
                  }
            }
            else
                  currentPlayer.setPlayerPosition(currentPlayer.getPosition()
+ diceCup.getSum());
      }
}
```

---

FieldController.java

---

```
package game;

import game.Account.IllegalAmountException;
import game.Account.InsufficientFundsException;
import game.chance_cards.ChanceCard;
import game.chance_cards.Fine;
import game.chance_cards.GoToJailCard;
import game.chance_cards.JailSafed;
import game.chance_cards.MoneyGift;
```

```
import game.chance_cards.MovedToField;
import game.fields.Chance;
import game.fields.Ownable;
import game.fields.Street;
import game.fields.Tax;
import gui.GUI;

public class FieldController
{
	private GUI gui = new GUI();
	private Board board;

	//Normalt er man ikke glad for "løse tal" i koden, og laver variable som
vi har gjort i Movecontroller:
	// fx START_BONUS = 4000;. Er det noget vi burde gør her, eller i hvert
fald overveje noget smartere?
	// Lige nu er der i hvert fald utrolig mange.

	public FieldController(Board board)
	{
		this.board = board;
	}

	public boolean LandOnField(Player p, int fieldNr)
	{
		gui.appendTextToTextArea(p.getName() +  " " +
board.getField(fieldNr).getMessage());


		if(board.getField(fieldNr) instanceof game.fields.GoToJail)
		{
			p.setPlayerPosition(11);
			p.setInPrisson(true);
		}

		if (board.getField(fieldNr) instanceof Tax)
			return taxHandler(p, fieldNr);

		if (board.getField(fieldNr) instanceof Ownable)
			return ownableHandler(p, fieldNr);

		if (board.getField(fieldNr) instanceof Chance)
		{
			return chanceHandler(p, fieldNr);
		}

		return false;
	}
```

```
	private boolean chanceHandler(Player p, int nr)
	{
		ChanceCard chanceCard = board.getChanceCard();

		String[] options1 = { "Proceed game"};

		int nrOfOptions = 1;
		String[] options = new String[nrOfOptions];
		for (int i = 0; i < options.length; i++)
		{
			options[i] = options1[i];
		}

		String message = chanceCard.getCardName();
		while((gui.getUserButtonPressed(options, message, "You draw a
chance card")) == -1);

		// Controller for chance cards that sends player to jail.
		if(chanceCard instanceof GoToJailCard)
		{
			p.setPlayerPosition(11);
			p.setInPrisson(true);
			board.appendChanceCard(chanceCard);
		}

		// Controller for chance cards that saves player from jail.
		if(chanceCard instanceof JailSafed)
		{
			// Remember to not put card back into queue.
			p.setGetOutOfJailCards(p.getGetOutOfJailCards() + 1);
		}

		// Controller for chance cards that gives money to the player.
		if(chanceCard instanceof MoneyGift)
		{
			chanceMoneyGiftHandler(p, (MoneyGift) chanceCard);
			board.appendChanceCard(chanceCard);
		}

		// Controller for chance cards that moves player an fixed amount
or to a specific field.
		if(chanceCard instanceof MovedToField)
		{
			board.appendChanceCard(chanceCard);
			return chanceMoveToFieldHandler(p, (MovedToField)
chanceCard);
		}
```

```
            //Handler for chance cards that costs money.
            if(chanceCard instanceof Fine)
            {
                  board.appendChanceCard(chanceCard);
                  return fineHandler(p, ((Fine) chanceCard));
            }
            return false;
      }

      private boolean fineHandler(Player p, Fine chanceCard)
      {
            if(chanceCard.getCardNumber() == 13 || chanceCard.getCardNumber()
== 25)
            {
                  if(chanceCard.getCardNumber() == 13)
                  {
                        //Card number == 13

                        try {

      p.getAccount().withdraw(calculateTotalFine(p,800,2300));
                        } catch (InsufficientFundsException e) {
                              e.printStackTrace();
                              return true;
                        } catch (IllegalAmountException e) {
                              e.printStackTrace();
                        }
                  }
                  else
                  {
                        // Card number == 25
                        try {

      p.getAccount().withdraw(calculateTotalFine(p,500,2000));
                        } catch (InsufficientFundsException e) {
                              e.printStackTrace();
                              return true;
                        } catch (IllegalAmountException e) {
                              e.printStackTrace();
                        }
                  }
            }
            // All other Fine Chancecards
            else {
                  try {
                        p.getAccount().withdraw(chanceCard.getFine());
                  }
                  catch (InsufficientFundsException e)
```

```
                {
                        e.printStackTrace();
                        return true;
                }

                catch (IllegalAmountException e)
                {
                        e.printStackTrace();
                }

        }
        return false;
    }

    private int calculateTotalFine(Player p, int houseFine, int hotelFine) {
        Ownable[] fields = Board.getFieldsByPlayer(p);
        int totalFine = 0;
        for (int i = 0; i < fields.length; i++)
        {
                if(fields[i] instanceof Street){
                        if(((Street) fields[i]).getHouses() > 4)
                                totalFine = totalFine + hotelFine;
                        else
                        {
                                totalFine = totalFine + (((Street)
fields[i]).getHouses() * houseFine);
                        }
                }


        }
        return totalFine;
    }




    private boolean chanceMoveToFieldHandler(Player p, MovedToField
chanceCard)
    {
        if(chanceCard.getCardNumber() == 24)
        {
                p.setPlayerPosition(p.getPosition() - 3);
        }

        else
        {
```

```
                  if(chanceCard.getCardNumber() == 15 ||
chanceCard.getCardNumber() == 16)
                  {
                        // move player to nearest Shipping and withdraw double
rent if owned.¨
                        // Shipping numbers are: 5, 15, 25, 35

                        int n =-1;
                        if(p.getPosition() > 35 && p.getPosition() < 5)
      { n = 0;}
                        if(p.getPosition() > 5 && p.getPosition() < 15)
      { n = 1;}
                        if(p.getPosition() > 15 && p.getPosition() < 25)
      { n = 2;}
                        if(p.getPosition() > 25 && p.getPosition() < 35)
      { n = 3;}

                        p.setPlayerPosition(5+10*n);
                        Ownable o = (Ownable) board.getField(p.getPosition());

                        // withdraws 2 times rent if the field has an owner.
                        if (o.getOwner() != null)
                        {
                              try
                              {
                                    p.getAccount().withdraw(o.getRent() * 2);

      o.getOwner().getAccount().deposit(o.getRent() * 2);
                              }
                              catch (InsufficientFundsException e)
                              {
                                    // we end here if the player does'nt have
enough money to pay rent
                                    return true;
                              }
                              catch (IllegalAmountException e)
                              {
                                    // A real error happened, stop execution
                                    System.err.println(e.getMessage());
                                    e.printStackTrace();
                                    System.exit(0);
                              }
                        }
                  }
                  else
                  {
                        if(p.getPosition() > chanceCard.getMoveAmount() &&
chanceCard.getCardNumber() != 2 & chanceCard.getCardNumber() != 24)
                        {
```

```
                              try
                              {
                                    p.getAccount().deposit(4000);
                              }
                              catch (IllegalAmountException e)
                              {
                                    e.printStackTrace();
                              }
                        }
                        p.setPlayerPosition(chanceCard.getMoveAmount());
                  }

            }
            return false;
      }
      private void chanceMoneyGiftHandler(Player p, MoneyGift chanceCard)
      {
            if(chanceCard.getCardNumber() == 10 || chanceCard.getCardNumber()
== 27)
            {
                  if(chanceCard.getCardNumber() == 10)
                  {
                        //Card number == 10
                        try
                        {

      p.getAccount().deposit(200*GameController.getInstance().getAllPlayers().
length); // Depositing 200 times the amount of players in game
                        } catch (IllegalAmountException e)
                        {
                              e.printStackTrace();
                        }
                  }
                  else
                  {
                        // Card number == 27 (matador-legat)
                        if(p.getAccount().getBalance() > 15000)
                        {
                              // do nothing
                        }
                        else
                        {
                              try
                              {
                                    p.getAccount().deposit(40000);
                              }
                              catch (IllegalAmountException e)
                              {
                                    e.printStackTrace();
```

```
                        }
                    }
                }
            }
            else {
                // All other MoneyGift Chancecards
                try
                {
                    p.getAccount().deposit(chanceCard.getReward());
                }
                catch (IllegalAmountException e)
                {
                    e.printStackTrace();
                }
            }

        }

        private boolean ownableHandler(Player p, int nr)
        {
            Ownable o = (Ownable) board.getField(nr);

            // someone owns this field
            if (o.getOwner() != null)
            {
                try
                {
                    p.getAccount().withdraw(o.getRent());
                    o.getOwner().getAccount().deposit(o.getRent());
                }
                catch (InsufficientFundsException e)
                {
                    // we end here if the player does'nt have enough money
to pay rent
                    return true;
                }
                catch (IllegalAmountException e)
                {
                    // A real error happened, stop execution
                    System.err.println(e.getMessage());
                    e.printStackTrace();
                    System.exit(0);
                }

            }
            // nobody owns this field
            else
            {
                String[] options = { "Buy (" + o.getPrice() + ")", "No" };
```

```
                  int choise = -1;
                  //loops the choice if the "close window-X" is pressed
                  while((choise = gui.getUserButtonPressed(options, "Nobody
owns " + o.getName() + ". Would you like to buy it?", "Field for sale!")) == -
1);

                  if (choise == 0)
                  {
                        try
                        {
                              p.getAccount().withdraw(o.getPrice());
                              o.setOwner(p);
                        }
                        catch (InsufficientFundsException e)
                        {
                              // we end here if the player does'nt have enough
money to buy the field.
                              // maybe this should be checked before the
option is given.
                              return true;
                        }
                        catch (IllegalAmountException e)
                        {
                              // A real error happened, stop execution
                              System.err.println(e.getMessage());
                              e.printStackTrace();
                              System.exit(0);
                        }
                  }
            }
            return false;
      }

      private boolean taxHandler(Player p, int nr)
      {
            //Cast field to tax
            Tax t = (Tax) board.getField(nr);

            // if we're at field 5 we have a choice, else we just withdraw
cash.
            if (p.getPosition() == 5)
            {
                  // array with the options thats should be on the buttons.
                  String[] options = { "Pay 4000kr.", "Pay 10%" };
                  // show a choice menu, with the above options.
                  int i = -1;
                  while((i = gui.getUserButtonPressed(options, "Choose
Payment", "Tax")) == -1);
```

```
                  if (i == 0)
                  {
                        try
                        {
                              p.getAccount().withdraw(t.getRevenueRate());
                        }
                        catch (InsufficientFundsException e)
                        {
                              return true;
                        }
                        catch (IllegalAmountException e)
                        {
                              System.err.println(e.getMessage());
                              e.printStackTrace();
                              System.exit(0);
                        }
                  }
                  else
                  {
                        try
                        {
                              p.getAccount().withdraw((p.getTotaltAssets()) *
t.getTaxRate());
                        }
                        catch (InsufficientFundsException e)
                        {
                              return true;
                        }
                        catch (IllegalAmountException e)
                        {
                              System.err.println(e.getMessage());
                              e.printStackTrace();
                              System.exit(0);
                        }
                  }
            }
            else
            {
                  try
                  {
                        p.getAccount().withdraw(t.getExtraOrdinaryRate());
                  }
                  catch (InsufficientFundsException e)
                  {
                        return true;
                  }
                  catch (IllegalAmountException e)
                  {
                        System.err.println(e.getMessage());
```

```
                        e.printStackTrace();
                        System.exit(0);
                  }
            }
            return false;
      }
}
```

---

Player.java

---

```
package game;

import game.fields.Ownable;
import java.awt.Color;

public class Player
{
      private String name;
      private Color carColor;
      private int carType;
      private int position = 1;
      private Account account = new Account();
      private boolean isBroke = false;
      private boolean inPrisson = false;
      private int twoOfAKindRollCount = 0;
      private int prisonTurnCount = 0;
      private int getOutOfJailCards = 0;

      public Player(String name, Color carColor, int carType)
      {
            this.name = name;
            this.carColor = carColor;
            this.carType = carType;

      }

      public String getName()
      {
            return name;
      }

      public void setName(String name)
      {
            this.name = name;
      }

      public int getPosition()
      {
            return position;
```

```
}

public void setPlayerPosition(int position)
{
	this.position = position;
}

public Account getAccount()
{
	return account;
}

public void setAccount(Account account)
{
	this.account = account;
}

public Color getCarColour()
{
	return carColor;
}

public void setCarColour(Color carColor)
{
	this.carColor = carColor;
}

public int getCarType()
{
	return carType;
}

public void setCarType(int carType)
{
	this.carType = carType;
}

public void setBroke(boolean broke)
{
	this.isBroke = broke;
}

public boolean isBroke()
{
	return isBroke;
}

public boolean isInPrisson()
```

```
{
	return inPrisson;
}

public void setInPrisson(boolean inPrisson)
{
	this.inPrisson = inPrisson;
}

public int getTotaltAssets()
{
	Ownable[] o = Board.getFieldsByPlayer(this);
	int priceSum = 0;
	for (int i = 0; i < o.length; i++)
	{
		priceSum = priceSum + o[i].getPrice();
	}

	return getAccount().getBalance() + priceSum;
}

public int getTwoOfAKindRollCount()
{
	return twoOfAKindRollCount;
}

public void setTwoOfAKindRollCount(int twoOfAKindRollCount)
{
	this.twoOfAKindRollCount = twoOfAKindRollCount;
}

public int getPrisonTurnCount()
{
	return prisonTurnCount;
}

public void setPrisonTurnCount(int prisonTurnCount)
{
	this.prisonTurnCount = prisonTurnCount;
}

public int getGetOutOfJailCards()
{
	return getOutOfJailCards;
}

public void setGetOutOfJailCards(int getOutOfJailCards)
{
```

```
            this.getOutOfJailCards = getOutOfJailCards;
      }

}
```

---

GameData.java

---

```
package game;

import game.chance_cards.ChanceCard;
import game.chance_cards.Fine;
import game.chance_cards.GoToJailCard;
import game.chance_cards.JailSafed;
import game.chance_cards.MoneyGift;
import game.chance_cards.MovedToField;
import game.fields.Brewery;
import game.fields.Chance;
import game.fields.Field;
import game.fields.Refuge;
import game.fields.Shipping;
import game.fields.Street;
import game.fields.Street.Group;
import game.fields.Tax;

public class GameData
{
      private Player[] players;
      private Field[] fields;
      private ChanceCard[] cards;
      private DiceCup dice;
      private int[] breweryRent = {100};
      private int[] fleetRent = {250};

      /*to dimentionelt array af leje i orden af:
      {grund, hus, 2 huse, 3 huse, 4 huse, hotel}*/
      private int [][] streetRent = {
                  {50, 250, 750, 2250, 4000, 6000},
                  {50, 250, 750, 2250, 4000, 6000},
                  {100, 600, 1800, 5400, 8000, 11000},
                  {100, 600, 1800, 5400, 8000, 11000},
                  {150, 800, 2000, 6000, 9000, 12000},
                  {200, 1000, 3000, 9000, 12500, 15000},
                  {200, 1000, 3000, 9000, 12500, 15000},
                  {250, 1250, 3750, 10000, 14000, 18000},
                  {300, 1400, 4000, 11000, 15000, 19000},
                  {300, 1400, 4000, 11000, 15000, 19000},
                  {350, 1600, 4400, 12000, 16000, 20000},
                  {350, 1800, 5000, 14000, 17500, 21000},
                  {350, 1800, 5000, 14000, 17500, 21000},
```

```
            {400, 2000, 6000, 15000, 18500, 22000},
            {450, 2200, 6600, 16000, 19500, 23000},
            {450, 2200, 6600, 16000, 19500, 23000},
            {500, 2400, 7200, 17000, 20500, 24000},
            {550, 2600, 7800, 18000, 22000, 25000},
            {550, 2600, 7800, 18000, 22000, 25000},
            {600, 3000, 9000, 20000, 24000, 28000},
            {700, 3500, 10000, 22000, 26000, 30000},
            {1000, 4000, 12000, 28000, 34000, 40000}};


public GameData()
{
      this.players = null;
      this.dice = new DiceCup();
      this.fields = generateStandardFieldList();
      this.cards = generateStandartCardList();
}

private Field[] generateStandardFieldList()
{
      Field[] fields = new Field[40];

      //                              --TYPE--     --NR--    --NAME--
      --BASE RENT--     --PRICE--   --GROUP--
      fields[0]  = new Refuge       (1,   "Start");
      fields[1]  = new Street       (2,   "Rødovrevej",
streetRent[0],          1200,             Group.BLUE);
      fields[2]  = new Chance       (3,   "Prøv Lykken");
      fields[3]  = new Street       (4,   "Hvidovrevej",
streetRent[1],          1200,             Group.BLUE);
      fields[4]  = new Tax          (5,   "SKAT");
      fields[5]  = new Shipping     (6,   "Scanlines",
fleetRent,        4000);
      fields[6]  = new Street       (7,   "Roskildevej",
streetRent[2],          2000,             Group.PINK);
      fields[7]  = new Chance       (8,   "Prøv Lykken");
      fields[8]  = new Street       (9,   "Valby Langgade",
streetRent[3],          2000,             Group.PINK);
      fields[9]  = new Street       (10,  "Allégade",
streetRent[4],          2400,             Group.PINK);
      fields[10] = new Refuge       (11,  "Fængsel");
      fields[11] = new Street       (12,  "Frederiksberg Allé",
streetRent[5],          2800,             Group.GREEN);
      fields[12] = new Brewery      (13,  "Tuborg",
breweryRent,            3000,             dice);
      fields[13] = new Street       (14,  "Bülowsvej",
streetRent[6],          2800,             Group.GREEN);
```

```
      fields[14] = new Street       (15,  "Gl. Kongevej",
streetRent[7],          3200,             Group.GREEN);
      fields[15] = new Shipping     (16,  "Mols Linien",
fleetRent,        4000);
      fields[16] = new Street       (17,  "Bernstoftsvej",
streetRent[8],          3600,             Group.GREY);
      fields[17] = new Chance       (18,  "Prøv Lykken");
      fields[18] = new Street       (19,  "Hellerupvej",
streetRent[9],          3600,             Group.GREY);
      fields[19] = new Street       (20,  "Strandvejen",
streetRent[10],         4000,             Group.GREY);
      fields[20] = new Refuge       (21,  "Helle");
      fields[21] = new Street       (22,  "Trianglen",
streetRent[11],         4400,             Group.RED);
      fields[22] = new Chance       (23,  "Prøv Lykken");
      fields[23] = new Street       (24,  "Østerbrogade",
streetRent[12],         4400,             Group.RED);
      fields[24] = new Street       (25,  "Grønningen",
streetRent[13],         4800,             Group.RED);
      fields[25] = new Shipping     (26,  "ColorLine",
fleetRent,        4000);
      fields[26] = new Street       (27,  "Bredgade",
streetRent[14],         5200,             Group.WHITE);
      fields[27] = new Street       (28,  "Kgs. Nytorv",
streetRent[15],         5200,             Group.WHITE);
      fields[28] = new Brewery      (29,  "Coca Cola",
breweryRent,            3000,       dice);
      fields[29] = new Street       (30,  "Østergade",
streetRent[16],         5600,             Group.WHITE);
      fields[30] = new game.fields.GoToJail(31,       "Gå i fængsel");
      fields[31] = new Street       (32,  "Amagertorv",
streetRent[17],         6000,             Group.YELLOW);
      fields[32] = new Street       (33,  "VimmelSkaftet",
streetRent[18],         6000,             Group.YELLOW);
      fields[33] = new Chance       (34,  "Prøv Lykken");
      fields[34] = new Street       (35,  "Nygade",
streetRent[19],         6400,             Group.YELLOW);
      fields[35] = new Shipping     (36,  "Oslo Færgen",
fleetRent,        4000);
      fields[36] = new Chance       (37,  "Prøv Lykken");
      fields[37] = new Street       (38,  "Frederiksberg gade",
streetRent[20],         7000,             Group.PURPLE);
      fields[38] = new Tax          (39,  "Extraordinær skat");
      fields[39] = new Street       (40,  "Rådhuspladsen",
streetRent[21],         8000,             Group.PURPLE);

      return fields;
}
```

```
	private ChanceCard[] generateStandartCardList()
	{
		ChanceCard[] cards = new ChanceCard[33];

		cards[0] = new MoneyGift	(1,  "De modtager Deres
aktieudbytte. Modtag kr. 1000 af banken. ", 1000);
		cards[1] = new MovedToField (2,  "De rykkes til start", 1);
		cards[2] = new GoToJailCard		(3,  "Gå i fængsel. Ryk
direkte til fængslet. Selv om De passerer Start,"
				+ " indkassere de ikke kr. 4000 ");
		cards[3] = new GoToJailCard		(4,  "Gå i fængsel. Ryk
direkte til fængslet. Selv om De passerer Start,"
				+ " indkassere de ikke kr. 4000 ");
		cards[4] = new Fine		(5,  "De har været en tur i udlandet
og haft for mange cigaretter med "
				+ "hjem. Betal told kr. 200", 200);
		cards[5] = new Fine		(6,  "De har modtaget Deres
tandlægeregning. Betal kr. 2000", 2000);
		cards[6] = new MoneyGift	(7,  "De havde en række med elleve
rigtige i tipning. Modtag kr. 1000", 1000);
		cards[7] = new MoneyGift	(8,  "Deres præmieobligation er
kommet ud. De modtager kr. 1000 af banken.", 1000);
		cards[8] = new MoneyGift	(9,  "Deres præmieobligation er
kommet ud. De modtager kr. 1000 af banken.", 1000);
		cards[9] = new MoneyGift	(10, "Det er Deres fødselsdag.
Modtag af hver medspiller kr. 200", 0);
		cards[10] = new MoneyGift	(11, "Værdien af egen avl fra
nyttehaven udgør kr. 200, som De modtager af banken. ", 200);
		cards[11] = new Fine		(12, "Betal Deres bilforsikring kr.
1000" , 1000);
		cards[12] = new Fine		(13, "Ejendomsskatterne er steget,
ekstraudgifterne er: kr 800 pr. hus, kr 2300 pr. hotel.", 0);
		cards[13] = new MovedToField(14, "Ryk frem til Grønningen. Hvis De
passerer Start inkassér da kr. 4000 ", 25 );
		cards[14] = new MovedToField(15, "Ryk brikken frem til det
nærmeste rederi og betal ejeren to gange den "
				+ "leje, han ellers er berettiget til. Hvis selskabet
ikke ejes af nogen  kan De købe det af banken. ", 0);
		cards[15] = new MovedToField(16, "Ryk brikken frem til det
nærmeste rederi og betal ejeren to gange den "
				+ "leje, han ellers er berettiget til. Hvis selskabet
ikke ejes af nogen  kan De købe det af banken. ", 0);
		cards[16] = new MovedToField  (17, "Tag med Mols-Linjen --- flyt
brikken frem, og hvis De passerer Start"
				+ " inkassér da kr. 4000 ", 16);
		cards[17] = new JailSafed	(18, "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.");
```

```
            cards[18] = new JailSafed     (19, "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");
            cards[19] = new MoneyGift     (20, "Grundet dyrtiden har De fået
gageforhøjelse. Modtag kr. 1000 ", 1000);
            cards[20] = new MovedToField(21, "Ryk frem til Frederiksberg Allé.
Hvis De passerer Start inkassér kr. 4000 ", 12);
            cards[21] = new MoneyGift     (22, "De har vundet i
Klasselotteriet. Modtag kr. 500 ", 500);
            cards[22] = new MovedToField(23, "Tag ind på Rådhuspladsen.", 40
);
            cards[23] = new MovedToField(24, "Ryk tre felter tilbage.", -3);
            cards[24] = new Fine          (25, "Oliepriserne er steget, og De
skal betale: kr. 500 pr. hus,  kr. 2000 pr. hotel.", 0);
            cards[25] = new Fine          (26, "Betal kr. 3000 for reparation
af Deres vogn. ", 3000);
            cards[26] = new Fine          (27, "Betal kr. 3000 for reparation
af Deres vogn. ", 3000);
            cards[27] = new MoneyGift     (28, "De modtager Matador-legatet
for værdigt trængende, "
                        + "stort kr. 40000 Ved værdigt trængende forstås, at
Deres formue, "
                        + "d.v.s. Deres kontante penge + skøder + bygninger
ikke overstiger kr. 15000 ", 0);
            cards[28] = new Fine          (29, "Kommunen har eftergivet et
kvartals skat. Hæv i banken kr. 3000 ", 3000);
            cards[29] = new MoneyGift     (30, "Modtag udbytte af Deres aktier
kr. 1000", 1000);
            cards[30] = new MoneyGift     (31, "Modtag udbytte af Deres aktier
kr. 1000", 1000);
            cards[31] = new Fine          (32, "De har kørt frem for Fuld
Stop. Betal kr. 1000 i bøde. ", 1000);
            cards[32] = new Fine          (33, "De har måttet vedtage en
parkeringsbøde. Betal kr. 200 i bøde.", 200);

            return cards;
      }



      public ChanceCard[] getCards()
      {
            return cards;
      }

      public void setCards(ChanceCard[] cards)
      {
            this.cards = cards;
```

```
	}

	public Field[] getFields()
	{
		return fields;
	}

	public void setFields(Field[] fields)
	{
		this.fields = fields;
	}

	public Player[] getPlayers()
	{
		return players;
	}

	public void setPlayers(Player[] players)
	{
		this.players = players;
	}

	public DiceCup getDice()
	{
		return dice;
	}

	public void setDice(DiceCup dice)
	{
		this.dice = dice;
	}
}
```

---

Account.java

---

```
package game;

public class Account {
	private int balance;
	private final static int DEFAULT_START_BALANCE = 30000;

	public Account()
	{
		this(DEFAULT_START_BALANCE);
	}

	public Account(int startBalance)
	{
```

```
            this.balance = startBalance;
      }

      /**
       * Adds to the balance of this account
       * @param amount The amount to add, must be > 0
       * @throws IllegalAmountException If the amount was below than 0
       */
      public void deposit (int amount) throws IllegalAmountException
      {
            if (amount < 0)  // deposit value is negative
                  throw new IllegalAmountException("Cannot deposit negative
integers");
            else
                  balance += amount;
      }

      /**
       * Withdraws an amount from this accounts balance, must be smaller or
equal to the total balance of the account
       * @param amount The amount to withdraw
       * @throws InsufficientFundsException If there were to few funds to
withdraw amount
       * @throws IllegalAmountException If amount was a negative number.
       */
      public void withdraw (int amount) throws InsufficientFundsException,
IllegalAmountException
      {
            if (amount > balance)  //limits to not withdraw more than there is
                  throw new InsufficientFundsException("Inssuficient funds to
withdraw: " + amount);
            else if (amount < 0) //prevents calling incorrectly with negative
integers
                  throw new IllegalAmountException("Cannot withdraw a negative
amount");
            else //withdraw amount from the balance
                  balance -= amount;
      }

      public int getBalance ()
      {
            return balance;
      }

      /**
       * Changes the balance.
       * @param newBalance The new balance
       * @deprecated Don't use this, use {@link Account#withdraw(int)} and
{@link Account#deposit(int)} instead
```

```
	 */
	@Deprecated
	public void setBalance (int newBalance)
	{
		balance = newBalance;
	}

	//-----------------------------------------------------------------
	//custom exceptions for this account
	//-----------------------------------------------------------------

	@SuppressWarnings("serial")
	public class IllegalAmountException extends Exception
	{
		public IllegalAmountException(String string)
		{
			super(string);
		}
	}

	@SuppressWarnings("serial")
	public class InsufficientFundsException extends Exception
	{
		public InsufficientFundsException(String string)
		{
			super(string);
		}
	}
}
```

---

Board.java

---

```
package game;

import game.chance_cards.ChanceCard;
import game.fields.Field;
import game.fields.Ownable;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Random;

public class Board
{
	private static Field[] fields;
	private LinkedList<ChanceCard> chanceCards = new
LinkedList<ChanceCard>();
```

```
	public Board(Field[] fields, ChanceCard[] cards)
	{
		ChanceCard[] shuffledCards = ShuffleCards(cards);
		for (int i = 0; i < shuffledCards.length; i++)
		{
			chanceCards.add(shuffledCards[i]);
		}
		Board.fields = fields;
	}

	/**
	 * Private method to shuffle the array of cards using Durstenfeld's
algorithm
	 * @param cardArray the list of cards to shuffle
	 * @return the same list of cards, but now in random order.
	 */
	private ChanceCard[] ShuffleCards(ChanceCard[] cardArray)
	{
		Random rand = new Random();
		for (int i = 0; i < cardArray.length; i++)
		{
			int randomPosition = rand.nextInt(cardArray.length);
			// swap objects at i and randomPosition
			ChanceCard temp = cardArray[i];
			cardArray[i] = cardArray[randomPosition];
			cardArray[randomPosition] = temp;
		}
		return cardArray;
	}

	/**
	 * Gets all the fields owned by a given player.
	 * @param p The player.
	 * @return An array containing all fields that this player owns.
	 */
	public static Ownable[] getFieldsByPlayer(Player p)
	{
		ArrayList<Ownable> temp = new ArrayList<Ownable>();

		for (int i = 0; i < fields.length; i++)
		{
			if(fields[i] instanceof Ownable)
				if(((Ownable)fields[i]).getOwner() == p)
					temp.add((Ownable) fields[i]);
		}

		Ownable[] tempArr = new Ownable[temp.size()];
		for (int i = 0; i < temp.size(); i++)
```

```
			{
				tempArr[i] = temp.get(i);
			}
			return tempArr;
		}

		/**
		 * Return the field at position "nr"
		 * @param nr The number of the field [1-40], this method manages the
array offset.
		 * @return The field at position "nr" on the board.
		 */
		public Field getField(int nr)
		{
			return fields[nr - 1];
		}
		/**
		 * Returns all the fields.
		 * @return An array containing all fields on the board.
		 */
		public Field[] getAllFields()
		{
			return fields;
		}
		/**
		 * Removes the first element in the list and returns it.
		 * @return The first card in the list.
		 */
		public ChanceCard getChanceCard()
		{
			return chanceCards.remove();
		}

		/**
		 * Appends the given card to the end of the list
		 * @param card The card to append.
		 */
		public void appendChanceCard(ChanceCard card)
		{
			chanceCards.add(card);
		}

		public LinkedList<ChanceCard> getAllChanceCards()
		{
			return chanceCards;
		}
}
```

---

DiceCup.java

```
-----------------------------------------------------------------------
package game;

public class DiceCup
{
	private Die die1;
	private Die die2;
	private int dieSum;

	public DiceCup()
	{
		this.die1 = new Die();
		this.die2 = new Die();
	}

	/**
	 * Roll the dice and return the new die-sum.
	 * @return The sum of the new facevalues after the dice have been
rolled.
	 */
	public int rollDice()
	{
		dieSum = (die1.roll() + die2.roll());
		return dieSum;
	}

	/**
	 * Get the sum of the two dice
	 * @return an integer equal to the sum of the dice
	 */
	public int getSum()
	{
		return dieSum;
	}

	/**
	 * Get the current facevalues of the dice.
	 * @return An integer array with the two facevalues.
	 */
	public int[] getDiceFaceValues()
	{
		int[] DiceFaceValues = new int[2];
		DiceFaceValues[0] = die1.getFaceValue();
		DiceFaceValues[1] = die2.getFaceValue();
		return DiceFaceValues;
	}

	/**
	 * Checks if the current roll is a two of a kind roll.
```

```
       * @return An integer equal to the facevalue [1-6] of the dice IF they
are the same, or 0 if they are not the same.
       */
      public boolean isTwoOfAKind()
      {
            int twoOfaKind = 0;
            if (die1.getFaceValue() == die2.getFaceValue())
                  twoOfaKind = die1.getFaceValue();
            return (twoOfaKind > 0);
      } // Consider making boolean
}
```

---

Die.java

---

```
package game;

import java.util.Random;
/**
 * A class representing a single D6 die.
 * @author
 *
 */
public class Die
{
      private int faceValue;
      private Random rand;

      //constructor
      public Die()
      {
            //creates a random object with the seed 42.
            rand = new Random();
            this.roll();
      }

      // metode til at kaste terningen
      public int roll()
      {
            //gets a number between 0 and 5, then offsets it by 1 to match a
regular die.
            this.faceValue = rand.nextInt(6) + 1;
            return this.faceValue;
      }

      public int getFaceValue()
      {
            return faceValue;
      }
```

```
	public void setFaceValue(int faceValue)
	{
		this.faceValue = faceValue;
	}

	@Override
	public String toString()
	{
		return "facevalue: " + this.faceValue;
	}
}
```

---

Field.java

---

```
package game.fields;

public abstract class Field
{
	private int fieldNumber;
	private String name;

	public Field(int nr, String name)
	{
		this.fieldNumber = nr;
		this.name = name;
	}

	public abstract String getMessage();

	public int getFieldNumber()
	{
		return this.fieldNumber;
	}

	public void setFieldNumber(int fieldNumber)
	{
		this.fieldNumber = fieldNumber;
	}


	public String getName()
	{
		return this.name;
	}

	public void setName(String name)
	{
		this.name = name;
```

```
	}
}
```

---

Ownable.java

---

```
package game.fields;

import game.Player;

public abstract class Ownable extends Field
{
	private int price;
	private int rent[];
	private Player owner = null;

	protected Ownable(int nr, String name, int[] rent, int price)
	{
		super(nr, name);
		this.rent = rent;
		this.price = price;
	}

	public abstract int getRent();

	public Player getOwner()
	{
		return this.owner;
	}

	public int getPrice()
	{
		return this.price;
	}

	public void setOwner(Player owner)
	{
		this.owner = owner;
	}

	protected int[] getBaseRent()
	{
		return rent;
	}
}
```

---

Brewery.java

---

```
package game.fields;
```

```
import game.Board;
import game.DiceCup;

public class Brewery extends Ownable
{
	private DiceCup diceCup;

	public Brewery(int nr, String name, int rent[], int price, DiceCup dice)
	{
		super(nr, name, rent, price);
		this.diceCup = dice;
	}

	@Override
	public String getMessage()
	{
		return "have landed on field " + this.getFieldNumber() + "  " +
this.getName();
	}

	@Override
	public int getRent()
	{
		int dieSum = this.diceCup.getSum();

		return getBaseRent()[0] * getNumberOfBreweries() * dieSum;
	}

	public int getNumberOfBreweries()
	{
		Field[] fields = Board.getFieldsByPlayer(this.getOwner());
		int count = 0;
		for (int i = 0; i < fields.length; i++)
		{
			if(fields[i] instanceof Brewery)
				count++;
		}
		return count;
	}
}
-----------------------------------------------------------------------
Chance.java
-----------------------------------------------------------------------
package game.fields;

public class Chance extends Field
{
```

```
	public Chance(int nr, String name)
	{
		super(nr, name);
	}

	@Override
	public String getMessage()
	{
		return "have landed on field " + this.getFieldNumber() + "  " +
this.getName();
	}
}
```

GoToJail.java

```
package game.fields;

public class GoToJail extends Field
{
	public GoToJail(int nr, String name)
	{
		super(nr, name);
	}

	@Override
	public String getMessage()
	{
		return "have landed on field " + this.getFieldNumber() + "  " +
this.getName();
	}
}
```

Refuge.java

```
package game.fields;

public class Refuge extends Field
{
	public Refuge(int nr, String name)
	{
		super(nr, name);
	}

	@Override
	public String getMessage()
	{
		return "have landed on field " + this.getFieldNumber() + "  " +
this.getName();
	}
```

```
}


------------------------------------------------------------------------
Shipping.java
------------------------------------------------------------------------
package game.fields;

import game.Board;

public class Shipping extends Ownable
{
	public Shipping(int nr, String name, int rent[], int price)
	{
		super(nr, name, rent, price);
	}

	@Override
	public String getMessage()
	{
		return "have landed on field " + this.getFieldNumber() + "  " +
this.getName();
	}

	@Override
	public int getRent()
	{
		return (int)(Math.pow(2, getNumberOfShippings()) *
getBaseRent()[0]);
	}

	private int getNumberOfShippings()
	{
		Field[] fields = Board.getFieldsByPlayer(this.getOwner());
		int count = 0;
		for (int i = 0; i < fields.length; i++)
		{
			if(fields[i] instanceof Shipping)
				count++;
		}
		return count;
	}
}
------------------------------------------------------------------------
Street.java
------------------------------------------------------------------------
package game.fields;
```

```
import game.Board;

public class Street extends Ownable
{
	public enum Group
	{
		BLUE(2),
		PINK(3),
		GREEN(3),
		GREY(3),
		RED(3),
		WHITE(3),
		YELLOW(3),
		PURPLE(2);

		private int groupSize;

		Group(int size)
		{
			this.groupSize = size;
		}

		int getGroupSize()
		{
			return this.groupSize;
		}
	}

	private Group group;
	private int houses = 0;

	public Street(int nr, String name, int rent[], int price, Group group)
	{
		super(nr, name, rent, price);
		this.group = group;
	}

	@Override
	public String getMessage()
	{
		return "have landed on field " + this.getFieldNumber() + "  " +
this.getName();
	}

	@Override
	public int getRent()
	{
		if (houses == 0)
```

```
                  return getSeriesMultiplier() * getBaseRent()[0] ;

            else
                  return getBaseRent()[houses];

      }

      /**
       * Returns a multiplier 1 or 2 depending on how many fields of a color a
player owns.
       * @return 2 if the player owns all fields in this color, 1 otherwise.
       */
      public int getSeriesMultiplier()
      {
            int count = 0;
            Field[] ownersFields = Board.getFieldsByPlayer(this.getOwner());
            for (int i = 0; i < ownersFields.length; i++)
            {
                  if(ownersFields[i].getClass() == Street.class)
                        if(((Street) ownersFields[i]).getGroup() ==
this.group)
                              count++;
            }
            return count == this.group.getGroupSize() ? 2 : 1;
      }

      private Group getGroup()
      {
            return this.group;
      }

      public int getHouses()
      {
            return this.houses;
      }

      public void setHouses(int houses)
      {
            this.houses = houses;
      }
}
-----------------------------------------------------------------------
Tax.java
-----------------------------------------------------------------------
package game.fields;

public class Tax extends Field {
```

```
	private final int taxRate = 1/10;
	private final int revenueRate = 4000;
	private final int extraOrdinaryRate = 2000;

	public Tax(int nr, String name)
	{
		super(nr,name);
	}

	public int getTaxRate()
	{
		return this.taxRate;
	}

	public int getRevenueRate()
	{
		return this.revenueRate;
	}

	public int getExtraOrdinaryRate()
	{
		return this.extraOrdinaryRate;
	}

	@Override
	public String getMessage()
	{
		return "have landed on field " + this.getFieldNumber() + "  " +
this.getName();
	}

}
```

---

ChanceCard.java

---

```
package game.chance_cards;

import game.Player;

public class ChanceCard
{
	private int cardNumber;
	private String cardName;
	private Player holder = null;

	public ChanceCard(int nr, String name)
	{
		this.cardNumber = nr;
```

```
		this.setCardName(name);
	}

	public String getCardName() {
		return cardName;
	}

	public void setCardName(String cardName) {
		this.cardName = cardName;
	}

	public int getCardNumber() {
		return cardNumber;
	}

	public void setCardNumber(int cardNumber) {
		this.cardNumber = cardNumber;
	}

	public Player getHolder() {
		return holder;
	}

	public void setHolder(Player holder) {
		this.holder = holder;
	}

}
```

---

Fine.java

---

```
package game.chance_cards;

public class Fine extends ChanceCard {

	private int fine;

	public Fine(int nr, String name, int fine) {
		super(nr, name);
		this.setFine(fine);
		// TODO Auto-generated constructor stub
	}

	public int getFine() {
		return fine;
	}

	public void setFine(int fine) {
```

```
            this.fine = fine;
      }

}
```

---

GoToJailCard.java

---

```
package game.chance_cards;

public class GoToJailCard extends ChanceCard{

      public GoToJailCard(int nr, String name) {
            super(nr, name);
            // TODO Auto-generated constructor stub
      }
}
```

---

JailSafed.java

---

```
package game.chance_cards;

public class JailSafed extends ChanceCard{

      public JailSafed(int nr, String name) {
            super(nr, name);
            // TODO Auto-generated constructor stub
      }

}
```

---

MoneyGift.java

---

```
package game.chance_cards;

public class MoneyGift extends ChanceCard{

      private int reward;

      public MoneyGift(int nr, String name, int reward) {
            super(nr, name);
            this.setReward(reward);
            // TODO Auto-generated constructor stub
      }

      public int getReward() {
            return reward;
      }
```

```
	public void setReward(int reward) {
		this.reward = reward;
	}

}
```

---

MovedToField.java

---

```
package game.chance_cards;

public class MovedToField extends ChanceCard {

	private int moveAmount;

	public MovedToField(int nr, String name, int moveAmount) {
		super(nr, name);
		this.setMoveAmount(moveAmount);
		// TODO Auto-generated constructor stub
	}

	public int getMoveAmount() {
		return moveAmount;
	}

	public void setMoveAmount(int moveAmount) {
		this.moveAmount = moveAmount;
	}

}
```

---

ChanceCardTest.java

---

```
package test;

import game.Board;
import game.FieldController;
import game.GameData;
import game.Player;
import game.chance_cards.ChanceCard;
import game.fields.Chance;
import game.fields.Field;

import java.awt.Color;

import org.junit.Test;

public class ChanceCardTest {
```

```
Player player = new Player("TESTPLAYER", Color.blue, 2);
Field[] Testfield =
{
new Chance(1, "TestChancecard"),
new Chance(2, "TestChancecard"),
new Chance(3, "TestChancecard"),
new Chance(4, "TestChancecard"),
new Chance(5, "TestChancecard"),
new Chance(6, "TestChancecard"),
new Chance(7, "TestChancecard"),
new Chance(8, "TestChancecard"),
new Chance(9, "TestChancecard"),
new Chance(10, "TestChancecard"),
new Chance(11, "TestChancecard"),
new Chance(12, "TestChancecard"),
new Chance(13, "TestChancecard"),
new Chance(14, "TestChancecard"),
new Chance(15, "TestChancecard"),
new Chance(16, "TestChancecard"),
new Chance(17, "TestChancecard"),
new Chance(18, "TestChancecard"),
new Chance(19, "TestChancecard"),
new Chance(20, "TestChancecard"),
new Chance(21, "TestChancecard"),
new Chance(22, "TestChancecard"),
new Chance(23, "TestChancecard"),
new Chance(24, "TestChancecard"),
new Chance(25, "TestChancecard"),
new Chance(26, "TestChancecard"),
new Chance(27, "TestChancecard"),
new Chance(28, "TestChancecard"),
new Chance(29, "TestChancecard"),
new Chance(30, "TestChancecard"),
new Chance(31, "TestChancecard"),
new Chance(32, "TestChancecard"),
new Chance(33, "TestChancecard"),
new Chance(34, "TestChancecard"),
new Chance(35, "TestChancecard"),
new Chance(36, "TestChancecard"),
new Chance(37, "TestChancecard"),
new Chance(38, "TestChancecard"),
new Chance(39, "TestChancecard"),
new Chance(40, "TestChancecard"),
new Chance(41, "TestChancecard"),
new Chance(42, "TestChancecard"),
};
GameData gamedata = new GameData();

Board board = new Board(Testfield, gamedata.getCards());
```

```
	FieldController fieldcontroller = new FieldController(board);

	@Test
	public void chanceCardTest()
	{
	for (int i = 0; i < gamedata.getCards().length; i++) {
	ChanceCard chanceCard = board.getChanceCard();
	fieldcontroller.LandOnField(player, 1);

	System.out.println("Test number" + (i+1) + "/" +
gamedata.getCards().length);
	System.out.println("Player position:" + player.getPosition());
	System.out.println("Player Balance:" +
player.getAccount().getBalance());
	System.out.println("Player get out of jail cards:" +
player.getGetOutOfJailCards());
	System.out.println("Chance card number:" + chanceCard.getCardNumber());
	System.out.println("Chance card name:" + chanceCard.getCardName());
	System.out.println();

	}

	}


}
```

---

DatabaseTest.java

---

```
package test;

import game.GameData;
import game.fields.Ownable;
import game.fields.Street;
import dbacces.DBCommunication;

public class DatabaseTest
{
	@org.junit.Test
	public void loadGameTest()
	{
		GameData data = DBCommunication.loadGame();
		for (int i = 0; i < data.getPlayers().length; i++)
		{
			System.out.println("Player[" + i + "] = " +
data.getPlayers()[i].getName());
		}
		System.out.println();
```

```
            for (int i = 0; i < data.getFields().length; i++)
            {
                  if(data.getFields()[i] instanceof Ownable &&
((Ownable)data.getFields()[i]).getOwner() != null)
                        System.out.println(data.getFields()[i].getName() + " :
" + ((Ownable) data.getFields()[i]).getOwner().getName());
                  if(data.getFields()[i] instanceof Street &&
((Ownable)data.getFields()[i]).getOwner() != null)
                        System.out.println("houses: " +
((Street)data.getFields()[i]).getHouses());
            }

      }

//    @org.junit.Test
//    public void testFields()
//    {
//          DBAccess dba = new DBAccess();
//          try
//          {
//                Object[][] fieldData = dba.getFieldChanges();
//                for (int i = 0; i < fieldData.length; i++)
//                {
//                      System.out.println((int)fieldData[i][0]);
//                      System.out.println((String)fieldData[i][1]);
//                      System.out.println((int)fieldData[i][2]);
//                }
//          }
//          catch (SQLException e)
//          {
//                e.printStackTrace();
//          }
//    }
//
//    @org.junit.Test
//    public void test()
//    {
//          DBAccess dba = new DBAccess();
//          Player[] players = {new Player("magnus", Color.RED, 2)};
//          players[0].setPlayerPosition(1);
//          players[0].setGetOutOfJailCards(0);
//          players[0].setPrisonTurnCount(0);
//
//          try
//          {
//                dba.insertPlayer(players[0], 0);
//                Player[] playersFromDB = dba.getAllPlayers();
//
//                System.out.println(playersFromDB.length);
```

```
//
//                for (int i = 0; i < playersFromDB.length; i++)
//                {
//                      System.out.println(i);
//                      System.out.println(playersFromDB[i].getName());
//                }
//
//                dba.resetPlayers();
//          }
//          catch (SQLException e)
//          {
//                e.printStackTrace();
//          }
//    }

}
```

---

testBrewery.java

---

```
package test;

import game.Board;
import game.DiceCup;
import game.Player;
import game.fields.Brewery;
import game.fields.Field;

import java.awt.Color;

import org.junit.Test;

public class testBrewery
{
      @Test
      public void test()
      {
            int[] rent = {200};
            DiceCup dice = new DiceCup();
            dice.rollDice();
            Player p = new Player("hej", Color.RED, 2);
            Brewery b = new Brewery(0, "lala", rent, 200, dice);
            b.setOwner(p);
            Field[] fields = new Field[1];
            fields[0] = b;

            // Board board = new Board(fields, new GameData().getCards()); //
not used

            System.out.println(Board.getFieldsByPlayer(p)[0].getName());
```

```
            System.out.println(b.getNumberOfBreweries());
            System.out.println(dice.getSum());
            System.out.println(b.getRent());
      }

}
```

---

DBAccess.java

---

```
package dbacces;

import game.Player;

import java.awt.Color;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;

public class DBAccess
{
      private String driver = "com.mysql.jdbc.Driver";
      private String database_url =
"jdbc:mysql://mysql7.gigahost.dk:3306/jakobsabinsky_matador";
      private String username = "jakobsabinsky";
      private String password = "Helena11";

      /**
       * Deletes all players from the "player" table
       * @return the number of tuples changed in the DataBase
       * @throws SQLException
       */
      public void resetPlayers() throws SQLException
      {
            Connection connection = null;
            PreparedStatement statement = null;
            PreparedStatement statement2 = null;

            try
            {
                  Class.forName(driver);
                  connection = DriverManager.getConnection(database_url,
username, password);
                  String sql = "DELETE FROM player";

                  statement = connection.prepareStatement(sql);
                  statement.executeUpdate();
```

```
                sql = "ALTER TABLE player AUTO_INCREMENT = 1";

                statement2 = connection.prepareStatement(sql);
                statement2.executeUpdate();
            }
            catch (ClassNotFoundException e)
            {
                e.printStackTrace();
            }
            finally
            {
                statement.close();
                connection.close();
            }
        }

        /**
         * Resets the data in the "fields" table.
         * Set its back to: field_owner = NULL and number_of_houses = NULL
         * @return The number of tuples changed.
         * @throws SQLException
         */
        public int resetFields() throws SQLException
        {
            Connection connection = null;
            PreparedStatement statement = null;
            int rows = 0;

            try
            {
                Class.forName(driver);
                connection = DriverManager.getConnection(database_url,
username, password);
                String sql = "UPDATE fields SET field_owner = NULL ,
number_of_houses = NULL";
                statement = connection.prepareStatement(sql);

                rows = statement.executeUpdate();
            }
            catch (ClassNotFoundException e)
            {
                e.printStackTrace();
                return -1;
            }
            finally
            {
                statement.close();
                connection.close();
```

```
            }

            return rows;
      }

      /**
       * Inserts a new tuple in the player-table, representing the given
player.
       *
       * @param player the player to save in the database.
       * @param player_turn the players position in the turn-queue.
       * @return The number of tuples changed.
       * @throws SQLException
       */
      public int insertPlayer(Player player, int player_turn) throws
SQLException
      {
            Connection connection = null;
            PreparedStatement statement = null;
            int rows = 0;

            try
            {
                  Class.forName(driver);
                  connection = DriverManager.getConnection(database_url,
username, password);
                  String sql = "INSERT INTO player (player_name,
field_position, account_balance, prison_turn,"
                                                                  +
"get_out_of_jail_cards, car_color, car_type, player_turn)"
                                                                  + "VALUES
(?,?,?,?,?,?,?,?)";
                  statement = connection.prepareStatement(sql);

                  statement.setString(1, player.getName());
                  statement.setInt(2, player.getPosition());
                  statement.setInt(3, player.getAccount().getBalance());
                  statement.setInt(4, player.getPrisonTurnCount());
                  statement.setInt(5, player.getGetOutOfJailCards());
                  statement.setInt(6, player.getCarColour().getRGB());
                  statement.setInt(7, player.getCarType());
                  statement.setInt(8, player_turn);

                  rows = statement.executeUpdate();
            }
            catch (ClassNotFoundException e)
            {
                  e.printStackTrace();
                  return -1;
```

```
            }
            finally
            {
                  statement.close();
                  connection.close();
            }

            return rows;
      }
      /**
       * Changes the owner of a given field
       * @param owner The name of the owner, this player must exist in the
player-table
       * @param fieldNr The number of the field (1-40)
       * @return The number of tuples changed.
       * @throws SQLException
       */
      public int updateFieldOwner(String owner, int fieldNr) throws
SQLException
      {
            Connection connection = null;
            PreparedStatement statement = null;
            int rows = 0;

            try
            {
                  Class.forName(driver);
                  connection = DriverManager.getConnection(database_url,
username, password);
                  String sql = "UPDATE fields SET field_owner = (SELECT
id_player FROM player where player_name = ?) WHERE field_number = ?";
                  statement = connection.prepareStatement(sql);

                  statement.setString(1, owner);
                  statement.setInt(2, fieldNr);

                  rows = statement.executeUpdate();
            }
            catch (ClassNotFoundException e)
            {
                  e.printStackTrace();
                  return -1;
            }
            finally
            {
                  statement.close();
                  connection.close();
            }
```

```
            return rows;
      }

      /**
       * Changes the number of houses build on a given field (0-5).
       * @param houses the number of houses
       * @param fieldNr The number of the field (1-40)
       * @return The number of tuples changed.
       * @throws SQLException
       */
      public int updateNumberOfHouses(int houses, int fieldNr) throws
SQLException
      {
            Connection connection = null;
            PreparedStatement statement = null;
            int rows = 0;

            try
            {
                  Class.forName(driver);
                  connection = DriverManager.getConnection(database_url,
username, password);
                  String sql = "UPDATE fields SET number_of_houses = ? WHERE
field_number = ?";
                  statement = connection.prepareStatement(sql);

                  statement.setInt(1, houses);
                  statement.setInt(2, fieldNr);

                  rows = statement.executeUpdate();
            }
            catch (ClassNotFoundException e)
            {
                  e.printStackTrace();
                  return -1;
            }
            finally
            {
                  statement.close();
                  connection.close();
            }

            return rows;
      }

      /**
       * returns all the players currently in the database
       * @return An array of players
       * @throws SQLException
```

```
	 */
	@SuppressWarnings("deprecation")
	public Player[] getAllPlayers() throws SQLException
	{
		Connection connection = null;
		PreparedStatement statement = null;
		Player[] result = null;

		try
		{
			Class.forName(driver);
			connection = DriverManager.getConnection(database_url,
username, password);
			String sql = "SELECT * FROM player";
			statement = connection.prepareStatement(sql);


			ResultSet data = statement.executeQuery();

			//get the data out into a format we can work with.
			ArrayList<Player> players = new ArrayList<Player>();
			ArrayList<Integer> playerTurn = new ArrayList<Integer>();
			while(data.next())
			{
				Player temp = new Player(data.getString(2), new
Color(data.getInt(7)), data.getInt(8));
				temp.setPlayerPosition(data.getInt(3));
				temp.getAccount().setBalance(data.getInt(4));
				temp.setPrisonTurnCount(data.getInt(5));
				temp.setGetOutOfJailCards(data.getInt(6));
				players.add(temp);
				playerTurn.add(data.getInt(9));
			}

			//sort the players according to their turn
			result = new Player[players.size()];
			for (int i = 0; i < result.length; i++)
			{
				result[playerTurn.get(i)] = players.get(i);
			}
		}
		catch (ClassNotFoundException e)
		{
			e.printStackTrace();
			return null;
		}
		finally
		{
			statement.close();
```

```
                connection.close();
            }

            //return the sorted list of players
            return result;
      }

      /**
       * returns a double Object array, containing the field_number,
owner_name and number_of_houses in that order.
       * for all field where the owner is not null.
       * @return A double object array, with the field data
       * @throws SQLException
       */
      public Object[][] getFieldChanges() throws SQLException
      {
            Connection connection = null;
            PreparedStatement statement = null;
            Object[][] result = null;

            try
            {
                  Class.forName(driver);
                  connection = DriverManager.getConnection(database_url,
username, password);
                  String sql = "SELECT field_number, player_name,
number_of_houses FROM fields JOIN player ON fields.field_owner =
player.id_player WHERE field_owner IS NOT null";
                  statement = connection.prepareStatement(sql);

                  //format of resultset: field_number, player_name,
number_of_houses
                  ResultSet data = statement.executeQuery();

                  int columnCount = data.getMetaData().getColumnCount();
                  ArrayList<Object[]> fieldArrayList = new
ArrayList<Object[]>();
                  while(data.next())
                  {
                        Object[] row = new Object[columnCount];
                        row[0] = data.getInt(1);
                        row[1] = data.getString(2);
                        row[2] = data.getInt(3);

                        fieldArrayList.add(row);
                  }
                  result = fieldArrayList.toArray(new Object[0][0]);

            }
```

```
            catch (ClassNotFoundException e)
            {
                  e.printStackTrace();
                  return null;
            }
            finally
            {
                  statement.close();
                  connection.close();
            }
            //return a double array containing the data from the fields
            return result;
      }
}
```

---

DBCommunication.java

---

```
package dbacces;

import game.GameController;
import game.GameData;
import game.Player;
import game.fields.Field;
import game.fields.Ownable;
import game.fields.Street;

import java.sql.SQLException;

public class DBCommunication
{
      public static void saveGame()
      {
            DBAccess dba = new DBAccess();

            //Clear the database.
            try
            {
                  dba.resetPlayers();
                  dba.resetFields();
            }
            catch (SQLException e)
            {
                  e.printStackTrace();
            }
```

```
            //Save player data to the database
            Player[] players =
GameController.getInstance().getAllPlayers();
            for (int i = 0; i < players.length; i++)
            {
                  try
                  {
                        dba.insertPlayer(players[i],i);
                  }
                  catch (SQLException e)
                  {
                        e.printStackTrace();
                  }
            }

            //save field data to the database
            Field[] fields = GameController.getInstance().getFields();
            for (int j = 0; j < fields.length ; j++)
            {
                  if(fields[j] instanceof Ownable &&
((Ownable)fields[j]).getOwner() != null)
                  {
                        try
                        {
                              dba.updateFieldOwner(((Ownable)
fields[j]).getOwner().getName(), fields[j].getFieldNumber());
                        }
                        catch (SQLException e)
                        {
                              e.printStackTrace();
                        }
                  }
                  if(fields[j] instanceof Street &&
((Street)fields[j]).getHouses() != 0)
                  {
                        try
                        {
                              dba.updateNumberOfHouses(((Street)
fields[j]).getHouses(), fields[j].getFieldNumber());
                        }
                        catch (SQLException e)
                        {
                              e.printStackTrace();
                        }
```

```
                }
            }
        }

        public static GameData loadGame()
        {
            DBAccess dba = new DBAccess();

            //create new Gamedata, object.
            GameData data = new GameData();

            //get player data from DB and load them into Gamedata.
            try
            {
                Player[] players = dba.getAllPlayers();
                Object[][] fieldChanges = dba.getFieldChanges();
                Field[] fields = data.getFields();

                //for each of the changed fields
                for (int i = 0; i < fieldChanges.length; i++)
                {
                    //find the coresponding fieldnumber in the field
list
                    for (int j = 0; j < fields.length; j++)
                    {
                        if(fields[j].getFieldNumber() ==
(int)fieldChanges[i][0] && fields[j] instanceof Ownable)
                        {
                            //find the matching player, and set
him as the owner
                            for (int k = 0; k < players.length;
k++)
                            {

        if(players[k].getName().equals((String)fieldChanges[i][1]))

        ((Ownable)fields[j]).setOwner(players[k]);
                            }
                            //if in addition the field is a
street, update the number of houses.
                            if(fields[j] instanceof Street)

        ((Street)fields[j]).setHouses((int)fieldChanges[i][2]);
                        }
                    }
```

```
                }

                data.setPlayers(players);
                data.setFields(fields);
            }
            catch (SQLException e)
            {
                e.printStackTrace();
            }

            //return gamedata with the new playerlist + updated field
list
            return data;
      }



}
```

---

GameActionListener.java

---

```
package gui;

import game.Board;
import game.GameController;
import game.Player;
import game.fields.Field;
import game.fields.Ownable;
import game.fields.Street;

import java.awt.KeyEventDispatcher;
import java.awt.event.KeyEvent;
import java.util.LinkedList;

import dbacces.DBCommunication;

public class GameActionListener implements KeyEventDispatcher
{
      GUI gui = new GUI();
      public GameActionListener(){}

      //-----------------------------------------------------------------
      //Methods to handle Action events
      //-----------------------------------------------------------------
      public void buyHouseButtonEvent()
```

```
     {
          Ownable[] fields =
Board.getFieldsByPlayer(GameController.getInstance().getCurentPlayer())
;

          LinkedList<Street> buildableStreets = new
LinkedList<Street>();
          for (int i = 0; i < fields.length; i++)
          {
               if(fields[i] instanceof Street)
               {
                    if(((Street)fields[i]).getSeriesMultiplier() ==
2);
                         buildableStreets.add((Street)fields[i]);
               }
          }
          if(buildableStreets.size() > 0)
          {
               String[] options = new
String[buildableStreets.size()];
               for (int i = 0; i < buildableStreets.size(); i++)
               {
                    options[i] = buildableStreets.get(i).getName();
               }

               String choise = gui.getUserSelection(options, "What
field would you like to build a house on?", "Buy House");

               for (int i = 0; i < buildableStreets.size(); i++)
               {

     if(buildableStreets.get(i).getName().equals(choise))
                    {
                         try
                         {
                              //withdraw cash


     buildableStreets.get(i).setHouses(buildableStreets.get(i).getHous
es() + 1);
                         }
                         catch (Exception e)
                         {
                              //print not enough money
                         }
```

```
                              //update the GUI
                              switch(buildableStreets.get(i).getHouses())
                              {
                              case 0:
                                    break;
                              case 1:

      gui.setFieldPicture(buildableStreets.get(i).getFieldNumber(),
FieldPanel.Builder.HOUSE1);
                                    break;

                              case 2:

      gui.setFieldPicture(buildableStreets.get(i).getFieldNumber(),
FieldPanel.Builder.HOUSE2);
                                    break;

                              case 3:

      gui.setFieldPicture(buildableStreets.get(i).getFieldNumber(),
FieldPanel.Builder.HOUSE3);
                                    break;

                              case 4:

      gui.setFieldPicture(buildableStreets.get(i).getFieldNumber(),
FieldPanel.Builder.HOUSE4);
                                    break;

                              case 5:

      gui.setFieldPicture(buildableStreets.get(i).getFieldNumber(),
FieldPanel.Builder.HOTEL);
                                    break;
                              }
                        }
                  }
            }
            else
                  gui.appendTextToTextArea("You cant build any houses");
      }

      public void mainButtonEvent()
      {
```

```
            int oldState =
GameController.getInstance().getCurrentState();
            GameController.getInstance().advanceGame();
            this.updateCars();
            this.updateDice(oldState);
            this.updatePlayers();
            this.updateFields();
            if(GameController.getInstance().getCurrentState() ==
GameController.ROLL_STATE)
            {
                  gui.setMainButtonText("Roll Dice");
            }
            if(GameController.getInstance().getCurrentState() ==
GameController.END_TURN_STATE)
            {
                  gui.setMainButtonText("End Turn");
            }
            if(GameController.getInstance().getCurrentState() ==
GameController.GAME_OVER_STATE)
            {
                  gui.setMainButtonText("Game over " +
GameController.getInstance().getWinner().getName() + " has won");
            }
      }

      public void sellHouseButtonEvent()
      {
            System.out.println("Sell button was pressed");
      }

      public void saveGameEvent()
      {
            System.out.println("Save game event");
            DBCommunication.saveGame();
      }

      public void loadGameEvent()
      {
            System.out.println("Load game event");
      }

      public void tradeButtonEvent()
      {
            // TODO Auto-generated method stub
```

```
	}

	public void saveGameButtonEvent()
	{
		// TODO Auto-generated method stub

	}

	//-----------------------------------------------------------------
	//Method to handle key events
	//-----------------------------------------------------------------

	//called when a keyEvent is fired. This is called once for
KEY_PRESSED, and once for KEY_RELEASED
	//therefore they should be separated, example: if = KEY_RELEASED
will only catch one of the events
	@Override
	public boolean dispatchKeyEvent(KeyEvent e)
	{
		if (e.getID() == KeyEvent.KEY_RELEASED)
		{
			if (e.getKeyCode() == KeyEvent.VK_ENTER)
			{

			}
		}
		return false;
	}

	//-----------------------------------------------------------------
	//Method to handle window closing events
	//-----------------------------------------------------------------
	public void windowClosingEvent()
	{
		System.exit(0);
	}

	//-----------------------------------------------------------------
	//Methods to update the GUI after each "main button" event.
	//-----------------------------------------------------------------

	private void updateCars()
	{
		Player[] players =
GameController.getInstance().getAllPlayers();
```

```
            for (int i = 0; i < players.length; i++)
            {
                  gui.removeAllCars(players[i].getName());
                  gui.setCar(players[i].getPosition(),
players[i].getName());
            }


      gui.setCurrentPlayerName(GameController.getInstance().getCurentPl
ayer().getName());
      }

      private void updateFields()
      {
            Field[] fields = GameController.getInstance().getFields();
            for (int i = 0; i < fields.length; i++)
            {
                  if(fields[i] instanceof Ownable &&
((Ownable)fields[i]).getOwner() != null)
                        gui.setFieldOwner(i + 1, ((Ownable)
fields[i]).getOwner().getName());
                  if(fields[i] instanceof Ownable)
                        gui.setFieldPrice(i + 1, ((Ownable)
fields[i]).getPrice());
                  if(fields[i] instanceof Ownable /*&&
((Ownable)fields[i]).getOwner() != null*/ &&
((Ownable)fields[i]).getRent() != 0)
                        gui.setFieldRent(i + 1, ( (Ownable)
fields[i]).getRent());
            }
      }

      private void updatePlayers()
      {
            Player[] players =
GameController.getInstance().getAllPlayers();
            for (int i = 0; i < players.length; i++)
            {
                  gui.setPlayerMoney(players[i].getName(),
players[i].getAccount().getBalance());
            }
      }

      private void updateDice(int state)
      {
```

```
            if(state == GameController.ROLL_STATE)

      gui.setDice(GameController.getInstance().getDiceCup().getDiceFace
Values()[0],
GameController.getInstance().getDiceCup().getDiceFaceValues()[1]);
      }
}
```

---

GUI.java

---

```
package gui;

import game.GameController;
import game.Player;

import java.awt.Color;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

/**
 *
 * @author Magnus Brandt Sløgedal
 *
 */
public class GUI
{
      public GUI(){}

      /**
       * creates a GUI, with all the fields of a Standart Matador game.
       */
      public void create()
      {
            GameGUI.getInstance();
            initialSetup();
      }

      /**
       * Creates a GUI with the specified list of fields
       * @param fields The list of fields that the GUI should show
       */
      public void create(FieldPanel[] fields)
      {
```

```
            GameGUI.createCustomGUI(fields);
            initialSetup();
      }
      /**
       * Private method to do the initiation work on the GUI
       */
      private void initialSetup()
      {
            Player[] players =
GameController.getInstance().getAllPlayers();
            //loop all players
            for (int i = 0; i < players.length; i++)
            {
                  //add players
                  addPlayer(players[i].getName(),
                              players[i].getCarType(),
                              players[i].getCarColour());
                  //add cars
                  setCar(1, players[i].getName());

                  //add money
                  this.setPlayerMoney(players[i].getName(),
players[i].getAccount().getBalance());
            }
            setCurrentPlayerName(players[0].getName());
      }


      /**
       * sets the two dice of the game to the specified integers, then
moves them to a random
       * place on the board, and rotates them at random.
       * @param i Dice one face-value
       * @param j Dice two face-value
       */
      public void setDice(int i, int j)
      {
            GameGUI.getInstance().setDice(i, j);
      }

      public void setPlayerMoney(String playerName, int amount)
      {
            GameGUI.getInstance().setPlayerMoney(playerName, amount);

      }
```

```
	/**
	 * Displays a line of text at the top of the control area.
	 * @param name The name to display
	 */
	public void setCurrentPlayerName(String name)
	{
		GameGUI.getInstance().setActivePlayerName(name);
	}

	/**
	 * adds a player to the field
	 * @param playerName The Name of the player
	 * @param carType An integer 0-3 specifying the type of car to
show for this player
	 * @param carColor A Color object specifying the main color of
the car
	 * @return True if there was lass than 6 players displayed, and
the car was a integer between 0 and 3 inclusive, False otherwise.
	 */
	public boolean addPlayer(String playerName, int carType, Color
carColor)
	{
		if (carType > 3)
			return false;
		GameGUI.getInstance().addPlayer(playerName, carType,
carColor);
		return true;
	}

	/**
	 * Removes a given player from the board.
	 * @param playerName the name of the player to remove.
	 * @return true if the player was successfully removed, false
otherwise.
	 */
	public boolean removePlayer(String playerName)
	{
		return GameGUI.getInstance().removePlayer(playerName);
	}

	/**
	 * displays a text-string on the board
	 * @param text the String to display
	 */
```

```
	public void setDisplayedText(String text)
	{
		GameGUI.getInstance().setDisplayedText(text);
	}

	/**
	 * changes the text of the Buy button
	 * @param text The new text for the button
	 */
	public void setMainButtonText(String text)
	{
		GameGUI.getInstance().setButtonText(0, text);
	}

	/**
	 * changes the text of the Buy button
	 * @param text The new text for the button
	 */
	public void setSellButtonText(String text)
	{
		GameGUI.getInstance().setButtonText(2, text);
	}

	/**
	 * changes the text of the Buy button
	 * @param text The new text for the button
	 */
	public void setBuyButtonText(String text)
	{
		GameGUI.getInstance().setButtonText(1, text);
	}

	/**
	 * asks the user to select from list of options
	 * @param options An array of string to chose from, the string at
[0] will be used as the default choise
	 * @param message The message accompanying this choise
	 * @param title The title of the created Frame with the choise
	 * @return The string that the user chose
	 */
	public String getUserSelection(String[] options, String message,
String title)
	{
		return (String) JOptionPane.showInputDialog(new JFrame(),
				message, title, JOptionPane.PLAIN_MESSAGE, null,
```

```
                    options, options[0]);
      }

      /**
       * asks the user to select a button
       * @param options An array of string to chose from, the string at
[0] will be used as the default choise
       * @param message The message accompanying this choise
       * @param title The title of the created Frame with the choise
       * @return An integer indicating what the user chose,
corresponding to the position in the options array.
       */
      public int getUserButtonPressed(String[] options, String message,
String title)
      {
            return JOptionPane.showOptionDialog(null, message, title,
                    JOptionPane.YES_NO_OPTION,
JOptionPane.INFORMATION_MESSAGE,
                    null, options, options[0]);
      }

      /**
       * asks the user to select from list of options
       * @param options options An array of string to chose from, the
string at [0] will be used as the default choise
       * @param message message The message accompanying this choise
       * @param title The title of the created Frame with the choise
       * @param imagePath the path to the image accompanying this
choise
       * @return The string that the user chose
       */
      public String getUserSelection(String[] options, String message,
String title, String imagePath)
      {
            return (String) JOptionPane.showInputDialog(new JFrame(),
                    message, title, JOptionPane.QUESTION_MESSAGE, new
ImageIcon(ImageFactory.CreateImage(imagePath)),
                    options, options[0]);
      }

      /**
       * Asks the user to answer yes / no / cancel
       * @param message The message accompanying this choise
       * @return an integer, indicating the choise: 0 = Yes, 1 = No, 2
= Cancel
```

```
	 */
	public int getUserYesNoCancelChoise(String message)
	{
		return JOptionPane.showConfirmDialog(new JFrame(), message);
	}

	/**
	 * Asks the user to answer yes / no / cancel
	 * @param message The message accompanying this choise
	 * @param title The title of the created Frame with the choise
	 * @param imagePath the path to the image accompanying this
choise
	 * @return an integer, indicating the choise: 0 = Yes, 1 = No, 2
= Cancel
	 */
	public int getUserYesNoCancelChoise(String message, String title,
String imagePath)
	{
		return JOptionPane.showConfirmDialog(new JFrame(),
				message, title, JOptionPane.YES_NO_CANCEL_OPTION,
JOptionPane.YES_NO_CANCEL_OPTION,
				new
ImageIcon(ImageFactory.CreateImage(imagePath)));
	}

	/**
	 * Asks the user to input a textString
	 * @param message The message accompanying this choise
	 * @param initialValue The initial value shown in the textBox
	 * @return The string supplied by the user
	 */
	public String getUserTextString(String message, String
initialValue)
	{
		return JOptionPane.showInputDialog(message, initialValue);
	}

	/**
	 * Asks the user to input a textString
	 * @param message The message accompanying this choise
	 * @param title The title of the created Frame with the choise
	 * @param imagePath the path to the image accompanying this
choise
	 * @param initialValue The initial value shown in the textBox
	 * @return The string supplied by the user
```

```
	 */
	public String getUserTextString(String message, String
title,String imagePath, String initialValue)
	{
		return (String) JOptionPane.showInputDialog(new JFrame(),
				message, title, JOptionPane.QUESTION_MESSAGE, new
ImageIcon(ImageFactory.CreateImage(imagePath)),
				null, initialValue);
	}

	/**
	 * Disposes of the GUI, and sets the singleton instance to null,
enabling the creation of a new GUI
	 */
	public void destroyGUI()
	{
		GameGUI.getInstance().destroyGUI();
	}

	/**
	 * Sets the picture of a given field
	 * @param fieldNumber the integer specifying the field
	 * @param pictureName The string specifying the picture
(available statically via FieldPanel.Builder)
	 */
	public void setFieldPicture(int fieldNumber, String picturename)
	{
		GameGUI.getInstance().setFieldPicture(picturename,
fieldNumber);
	}

	/**
	 * Sets the owner of a given field
	 * @param fieldNumber the integer specifying the field
	 * @param owner The string specifying the new owners name
	 */
	public void setFieldOwner(int fieldNumber, String owner)
	{
		GameGUI.getInstance().setOwner(owner, fieldNumber);
	}

	/**
	 * Sets the title of a given field
	 * @param fieldNumber the integer specifying the field
	 * @param title The string specifying the new title of the field
```

```
	 */
	public void setFieldTitle(int fieldNumber, String title)
	{
		GameGUI.getInstance().setTitle(title, fieldNumber);
	}

	/**
	 * Sets the subtext of a given field
	 * @param fieldNumber the integer specifying the field
	 * @param subText The string specifying the new subtext of the
field
	 */
	public void setFieldSubText(int fieldNumber, String subText)
	{
		GameGUI.getInstance().setSubtext(subText, fieldNumber);
	}

	/**
	 * Sets the price of a given field
	 * @param fieldNumber the integer specifying the field
	 * @param price The string specifying the new price of the field
	 */
	protected void setFieldPrice(int fieldNumber, String price)
	{
		GameGUI.getInstance().setPrice(price, fieldNumber);
	}

	protected void setFieldRent(int fieldNumber, String rent)
	{
		GameGUI.getInstance().setRent(rent, fieldNumber);
	}

	/**
	 * Sets the price of a given field
	 * @param fieldNumber the integer specifying the field
	 * @param price The integer specifying the new price of the field
	 */
	protected void setFieldPrice(int fieldNumber, int price)
	{
		GameGUI.getInstance().setPrice("" + price, fieldNumber);
	}

	protected void setFieldRent(int fieldNumber, int rent)
	{
		GameGUI.getInstance().setRent("" + rent, fieldNumber);
```

```
	}

	/**
	 * Sets the priceText of a given field, example "price" when not
owned versus "rent" when someone owns the field
	 * @param fieldNumber the integer specifying the field
	 * @param priceText The string specifying the new priceText of
the field
	 */
	public void setFieldPriceText(int fieldNumber, String priceText)
	{
		GameGUI.getInstance().setPriceText(priceText, fieldNumber);
	}

	/**
	 * Appends a string to the games textArea
	 * @param text the string to append
	 */
	public void appendTextToTextArea(String text)
	{
		GameGUI.getInstance().appendText(text);
	}

	/**
	 * clears all text from the games textArea
	 */
	public void clearTextFromTextArea()
	{
		GameGUI.getInstance().clearTextField();
	}

	/**
	 * Adds a car to the given field
	 * @param fieldNumber the number of the field
	 * @param cartype An integer 0-3 (inclusive) specifying the type
of car to show
	 * @param carNr a number 1-6(inclusive) specifying the layer the
car should be at
	 * @param color The Color of the car
	 */
	public void setCar(int fieldNumber, String playerName)
	{
		Player[] players =
GameController.getInstance().getAllPlayers();
		for (int i = 0; i < players.length; i++)
```

```
            {
                  if(players[i].getName().equals(playerName))
                        GameGUI.getInstance().setCar(fieldNumber,
players[i].getCarType(), i + 1, players[i].getCarColour());
            }
      }

      /**
       * Removes a given car from the given field
       * @param fieldNumber  the number of the field
       * @param carNr the number 1-6(inclusive) specifying the layer
the car that should be removed is at
       */
      public void removeCar(int fieldNumber, int carNr)
      {
            GameGUI.getInstance().removeCar(fieldNumber, carNr);
      }

      public void removeAllCars(String name)
      {
            Player[] players =
GameController.getInstance().getAllPlayers();
            for (int i = 0; i < players.length; i++)
            {
                  if(players[i].getName().equals(name))
                  {
                        for (int j = 1; j <= 40; j++)
                        {
                              removeCar(j, i + 1);
                        }
                  }
            }
      }
}
```

---

NewGameDialog.java

---

```
package startmenugui;

import game.Player;
import gui.ImageFactory;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
```

```
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.image.BufferedImage;
import java.util.ArrayList;

import javax.swing.DefaultComboBoxModel;
import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.event.PopupMenuEvent;
import javax.swing.event.PopupMenuListener;

@SuppressWarnings("serial")
public class NewGameDialog extends JDialog implements ActionListener,
PopupMenuListener
{
	private JPanel panel;
	private JTextField[] playerNames;
	private BufferedImage[] carArray;
	private MyImageIcon[] colors;
	private JComboBox<Icon>[] carTypes;
	private JComboBox<MyImageIcon>[] colorComboBox;
	private JCheckBox[] checkBoxArray;
	private JButton okButton;
	private JButton cancelButton;

	private boolean okTrue = false;

	@SuppressWarnings("unchecked")
	public NewGameDialog()
	{
		panel = new JPanel();

		this.setModalityType(ModalityType.APPLICATION_MODAL);
		this.setPreferredSize(new Dimension(450,350));
		this.addWindowListener(new WindowAdapter() {
```

```
            @Override
            public void windowClosing(WindowEvent e)
            {
                  ((JDialog) e.getSource()).setVisible(false);
                  ((JDialog) e.getSource()).dispose();
            }
      });
      this.pack();
      this.getContentPane().add(panel);

      panel.setLayout(null);
      playerNames = new JTextField[6];

      colors = new MyImageIcon[12];
      colors[0] = createColorIcon(Color.BLACK);
      colors[1] = createColorIcon(Color.RED);
      colors[2] = createColorIcon(Color.BLUE);
      colors[3] = createColorIcon(Color.YELLOW);
      colors[4] = createColorIcon(Color.GREEN);
      colors[5] = createColorIcon(Color.CYAN);
      colors[6] = createColorIcon(Color.PINK);
      colors[7] = createColorIcon(new Color(211, 0, 120));
//Purple
      colors[8] = createColorIcon(new Color(255, 110, 0));
//Orange
      colors[9] = createColorIcon(Color.MAGENTA);
      colors[10] = createColorIcon(Color.WHITE);
      colors[11] = createColorIcon(new Color(0x009999));
//lyseblå?

      carArray = ImageFactory.setupCarIcons();
      carTypes = new JComboBox[6];
      checkBoxArray = new JCheckBox[6];
      colorComboBox = new JComboBox[6];

      //trash data for first setup
      JLabel[] arr = new JLabel[1];
      arr[0] = new JLabel();

      for (int i = 0; i < playerNames.length; i++)
      {
            playerNames[i] = new JTextField("Player" + (1 + i));
            playerNames[i].setBounds(40, 5 + i * 40, 120, 30);
            colorComboBox[i] = new JComboBox<MyImageIcon>(colors);
            colorComboBox[i].setSelectedIndex(i);
```

```
                colorComboBox[i].setBounds(165, 5 + i * 40, 50, 30);
                colorComboBox[i].addActionListener(this);
                colorComboBox[i].addPopupMenuListener(this);
                carTypes[i] = new JComboBox<Icon>();
                carTypes[i].setModel(updateCarTypesColor(i));
                carTypes[i].setSelectedIndex(0);
                carTypes[i].setBounds(220, 5 + i * 40, 100, 30);
                checkBoxArray[i] = new JCheckBox();
                checkBoxArray[i].addActionListener(this);
                checkBoxArray[i].setBounds(5, 5 + i * 40, 30, 30);
                if(i == 0 || i == 1)
                      checkBoxArray[i].setSelected(true);
                if(i!= 0 && i != 1)
                      setCheckBoxState(i, false);

                panel.add(playerNames[i]);
                panel.add(colorComboBox[i]);
                panel.add(carTypes[i]);
                panel.add(checkBoxArray[i]);
          }

          okButton = new JButton("OK");
          okButton.setBounds(200, 260, 100, 30);
          okButton.addActionListener(this);
          panel.add(okButton);
          cancelButton = new JButton("Cancel");
          cancelButton.setBounds(310, 260, 100, 30);
          cancelButton.addActionListener(this);

          panel.add(cancelButton);
     }

     public Player[] showNewGameMenu()
     {

     this.setLocation((Toolkit.getDefaultToolkit().getScreenSize().wid
th / 2) - (this.getWidth() / 2),
(Toolkit.getDefaultToolkit().getScreenSize().height / 2) -
(this.getHeight() / 2));
          this.setVisible(true);

          if(okTrue)
                return generatePlayerArray();
          return null;
     }
```

```
	@Override
	public void actionPerformed(ActionEvent e)
	{
		int playercount = 0;

		for (int i = 0; i < 6; i++)
		{
			if (checkBoxArray[i].isSelected())
				playercount++;

			if (e.getSource() == colorComboBox[i])
			{
				carTypes[i].setModel(updateCarTypesColor(i));
			}
			else if (e.getSource() == checkBoxArray[i])
			{
				setCheckBoxState(i,
checkBoxArray[i].isSelected());
			}
		}
		if (e.getSource() == okButton)
		{
			okTrue = true;
			this.setVisible(false);
			this.dispose();
		}
		else if (e.getSource() == cancelButton)
		{
			okTrue = false;
			this.setVisible(false);
			this.dispose();
		}

		if (playercount < 2)
			okButton.setEnabled(false);
		else
			okButton.setEnabled(true);

	}

	@SuppressWarnings("unchecked")
	@Override
	public void popupMenuWillBecomeVisible(PopupMenuEvent e)
	{
```

```
            ((JComboBox<MyImageIcon>)
e.getSource()).setModel(updateColorComboboxModel());
      }

      private Player[] generatePlayerArray()
      {
            //hvow many players are there?
            int numberOfPlayers = 0;
            for (int i = 0; i < checkBoxArray.length; i++)
            {
                  if (checkBoxArray[i].isSelected())
                  {
                        numberOfPlayers++;
                  }
            }
            Player[] players = new Player[numberOfPlayers];
            for (int i = 0; i < players.length ; i++)
            {
                  players[i] = new Player(playerNames[i].getText(),
((MyImageIcon)colorComboBox[i].getSelectedItem()).getColor(),
carTypes[i].getSelectedIndex());
            }
            return  players;
      }

      private void setCheckBoxState(int line, boolean state)
      {
            playerNames[line].setEnabled(state);
            colorComboBox[line].setEnabled(state);
            carTypes[line].setEnabled(state);
      }

      private DefaultComboBoxModel<MyImageIcon>
updateColorComboboxModel()
      {
            ArrayList<MyImageIcon> colorArr = new
ArrayList<MyImageIcon>();
            for (int i = 0; i < colors.length; i++)
            {
                  boolean isSelectedColor = false;
                  for (int j = 0; j < colorComboBox.length; j++)
                  {
                        MyImageIcon temp = (MyImageIcon)
colorComboBox[j].getSelectedItem();
```

```
                    if (temp != null &&
temp.getColor().equals(colors[i].getColor()))
                         isSelectedColor = true;
               }
               if (!isSelectedColor)
               {
                    colorArr.add(colors[i]);
               }
          }
          MyImageIcon[] tempArr = new MyImageIcon[colorArr.size()];
          for (int i = 0; i < colorArr.size(); i++)
               tempArr[i] = colorArr.get(i);

          return new DefaultComboBoxModel<MyImageIcon>(tempArr);
     }

     private MyImageIcon createColorIcon(Color c)
     {
          BufferedImage buffImg = new BufferedImage(30, 30,
BufferedImage.TYPE_INT_RGB);
          Graphics g = buffImg.getGraphics();
          g.setColor(c);
          g.fillRect(0, 0, buffImg.getWidth(), buffImg.getHeight());
          MyImageIcon icon = new MyImageIcon(buffImg);
          icon.setColor(c);
          return icon;
     }

     private DefaultComboBoxModel<Icon> updateCarTypesColor(int
playerRow)
     {
          //the default color of the images(red) used to search and
replace
          final int DEFAULTPRIMARYCOLOR = 0xffff0000;

          Icon[] iconArr = new Icon[carArray.length];

          for (int i = 0; i < carArray.length; i++)
               iconArr[i] = new
ImageIcon(ImageFactory.replaceColor(carArray[i],
((MyImageIcon)colorComboBox[playerRow].getSelectedItem()).getColor(),
DEFAULTPRIMARYCOLOR));

          return new DefaultComboBoxModel<Icon>(iconArr);
     }
```

```
	private class MyImageIcon extends ImageIcon
	{
		public MyImageIcon(BufferedImage img)
		{
			super(img);
		}
		private Color thisColor;

		private Color getColor()
		{
			return thisColor;
		}

		private void setColor(Color c)
		{
			this.thisColor = c;
		}
	}

	//------------------------------------------------------------
	//unused events
	//------------------------------------------------------------
	@Override
	public void popupMenuCanceled(PopupMenuEvent e)
	{
		// TODO Auto-generated method stub

	}

	@Override
	public void popupMenuWillBecomeInvisible(PopupMenuEvent e)
	{
		// TODO Auto-generated method stub

	}


}
```

---

StartMenuDialog.java

---

```
package startmenugui;

import game.GameData;
```

```
import game.Player;

import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JPanel;

import dbacces.DBCommunication;

@SuppressWarnings("serial")
public class StartMenuDialog extends JDialog implements ActionListener
{
	private game.GameData options = new GameData(); // Hvorfor bliver
denne oprettet her?
	private JButton[] buttons;
	private JPanel panel;

	public StartMenuDialog()
	{
		this.setModalityType(ModalityType.APPLICATION_MODAL);
		panel = new JPanel(new FlowLayout());
		buttons = new JButton[4];

		for (int i = 0; i < buttons.length; i++)
		{
			buttons[i] = new JButton();
			buttons[i].addActionListener(this);
			buttons[i].setPreferredSize(new Dimension(200,60));
			panel.add(buttons[i]);
		}

		buttons[0].setText("Start New Game");
		buttons[1].setText("Load Game");
		buttons[2].setText("Options");
		buttons[3].setText("Quit");

		this.setPreferredSize(new Dimension(250, 310));
		this.setTitle("Matador");
```

```
            this.add(panel);
            this.pack();

            this.addWindowListener(new WindowAdapter() {
                  @Override
                  public void windowClosing(WindowEvent e)
                  {
                        System.out.println("lalala");
                        ((JDialog) e.getSource()).setVisible(false);
                        ((JDialog) e.getSource()).dispose();
                  }
            });
      }

      public game.GameData startDialog()
      {

      this.setLocation((Toolkit.getDefaultToolkit().getScreenSize().wid
th / 2) - (this.getWidth() / 2),
(Toolkit.getDefaultToolkit().getScreenSize().height / 2) -
(this.getHeight() / 2));
            this.setVisible(true);

            return options;
      }

      @Override
      public void actionPerformed(ActionEvent e)
      {
            //new game button
            if(e.getSource() == buttons[0])
            {
                  //show new frame to set options
                  Player[] players = new
NewGameDialog().showNewGameMenu();
                  if(players != null)
                  {
                        options.setPlayers(players);
                        this.setVisible(false);
                        this.dispose();
                  }
            }
            //load game button
            else if(e.getSource() == buttons[1])
            {
```

```
                this.options = DBCommunication.loadGame();
                this.setVisible(false);
            }
            //options button
            else if(e.getSource() == buttons[2])
            {
                //may be unnecessary
                //              showOptionsMenu();
            }
            //quit button
            else if(e.getSource() == buttons[3])
            {
                System.out.println("Quit Game");
                System.exit(0);
            }
        }
}
```