Projektopgave jan 2014

02312 Indledende programmering

Projekt navn: *CDIO – M1*

Gruppe nr: *53.*

Afleveringsfrist: *Mandag den 20/01 2014 Kl. 12:00*

Denne rapport er afleveret via Campusnet (der skrives ikke under)

Denne rapport indeholder *24* sider incl. denne side.

Studie nr, Efternavn, Fornavne Underskrift

S133991, Larsen, Anders _____

S133970, Hansen, Kristin _____

S134010, Jensen, Lars Peter _____

Indhold

Final - Gruppe 53

Time-regnskab Version 07-01-2014

Dato	Deltager	Design	Implementering	Test	Dokumentering	Andet	I alt
06-01-2014	Anders	1	0	0	1,5	0	2,5
	Kristin	1	0	0	1,5	0	2,5
	Lars Peter	1	0	0	1,5	0	2,5
07-01-2014	Anders	1	0	0	2	0,5	3,5
	Kristin	0,5	0	0	0,5	0	1
	Lars Peter	1	0	0	2,3	0,5	3,8
08-01-2014	Anders	1	0,5	0	1	1,5	4
	Kristin	1,5	0,75	0	1	0,75	4
	Lars Peter	0,5	0,5	0	1	1,7	3,7
09-01-2014	Anders	0	4	0	0	0	4
	Kristin	0	4	0	0	0	4
	Lars Peter	0	4	0	0	0	4
10-01-2014	Anders	0	1,5	0	1,5	0	3
	Kristin	0	0,5	0	0	3	3,5
	Lars Peter	1	1	0	1,5	0	3,5
11-01-2014	Anders	0	2	1	0	0,5	3,5
	Kristin	1	1	1	0	0,5	3,5
	Lars Peter	0	2	0,5	0,5	0,5	3,5
12-01-2014	Anders	0	0	0	0	0	0
	Kristin	0	3	1	0	0	4
	Lars Peter	0	1	0,5	0	0	1,5
13-01-2014	Anders	1	3	2	0	0	6
	Kristin	1	3	2	0	0	6
	Lars Peter	0	3	1	0	1	5
14-01-2014	Anders	0,5	0,5	0,5	0	0	1,5
	Kristin	0	0	0	0	0	0
	Lars Peter	0,5	0,5	0,5	0	0	1,5
15-01-2014	Anders	0	1	1	0	1	3
	Kristin	0	1	1	0	1	3
	Lars Peter	0	1	1	0	1	3
16-01-2014	Anders	0	2	0	0	0	2
	Kristin	0	6	0	0	0	6
	Lars Peter	0	2	0	0	0,5	2,5
17-01-2014	Anders	0	5	0	0	1	6
	Kristin	0	8	0	0	0	8
	Lars Peter	0	4	0	0	2	6
18-01-2014	Anders	0	6	0	0	0,5	6,5
	Kristin	0	6	2	0	0	8
	Lars Peter	0	5	1	0	0	6
19-01-2014	Anders	0	1,5	3	8	0,5	13
	Kristin	0	3	2	6	2	13
	Lars Peter	0	1	4	8	0	13
Sum		13,5	88,25	25	37,8	19,95	184,5

I alt pr. person		
	Anders	58,5
	Kristin	66,5
	Lars Peter	59,5

Indledning

Afleveringen er sidste led i et længere projektforløb, hvor denne sidste rapport, gerne skulle afspejle gruppens evner indenfor særligt programmering. Herunder er der blevet lagt fokus på designprincipper som f.eks. GRASP.

Udover rapporten, er der blevet udarbejdet et matadorspil. Spillet skal afspejle i en så høj grad som muligt det oprindelige brætspil, mens det selvfølgelig skal kunne afvikles på DTU's computere.

Programmeringens dokumentation sker særligt gennem de udviklet diagrammer for systemet, herunder domænemodel, interaktionsdiagrammer (BCE og sekvensdiagram), samt designklassediagrammer.

For designdiagrammerne, er der kun blevet udviklet designsekvensdiagrammer for de nødvendige og kritiske klasser, som er blevet implementeret siden seneste CDIO aflevering. Herunder er det blevet vurderet, at sekvensdiagrammer for køb og salg af huse/hoteller, auktion og pantsætning, har været vigtige at få med.

Vedr. programmeringen, er der blevet sat fokus på, at vi kan eftervise, at vi kan arbejde med arrays, hvorfor nogle af løsningerne for metoder og des lige, sagtens kunne være lavet anderledes (f.eks. med arraylist). Polymorfi har også været et fokusområde, som dog allerede var implementeret gennem den seneste aflevering.

Analyse

Afgrænsnings plan (MoSCoW)

Must:

- Dokumentation
    - Javadocs til dokumentation af kode
    - Detaljerede beskrivelser af klasserne:
        - Fields (mere bestemt Streets)
        - Forskellige controllere
        - Chance (Prøv lykken)
        - Jail (fængsel)
- Features
    - Alle Klasser
    - Alt brug skal forgå i GUI
    - Ownable funktionerne (eje, leje samt købe)
    - Land on field funktionen
    - Chance, Jail og buyout funktioner
    - Vinder og taber(fallit) funktioner

Should:

- Dokumentation
    - Domæne model
    - BCE model
    - Use case model (Diagram samt beskrivelse)

    - Test samt Beskrivelse
    - Design klasse diagram
    - Design sekvens diagram
    - Grasp beskrivelse
- Features
    - Huse og hotel funktionerne
    - Ekstra slag ved et ens slag
    - Flere ejerskabs bonus ved Brewery(bryggeri) og Fleet(rederi)
    - Tax felt funktion
    - Ingen køb i først runde
    - Slå ens for at komme ud af fængsel

Could:

- Dokumentation
    - Furps beskrivelse
- Features
    - 3 x ens slag i træk smider spiller i fængsel
    - Pantsætning af Streets funktion
    - Jævnfordeling af huse på et gade område

Want:

- Dokumentation
    - Undersøgelser
        - Kvantitativt
        - Kvalitativt
- Features
    - Grafisk førelse af bilernes bevægelse
    - Auktioner funktion for ikke købt grund
    - Bytte funktion blandt spillerne

Sorted out:

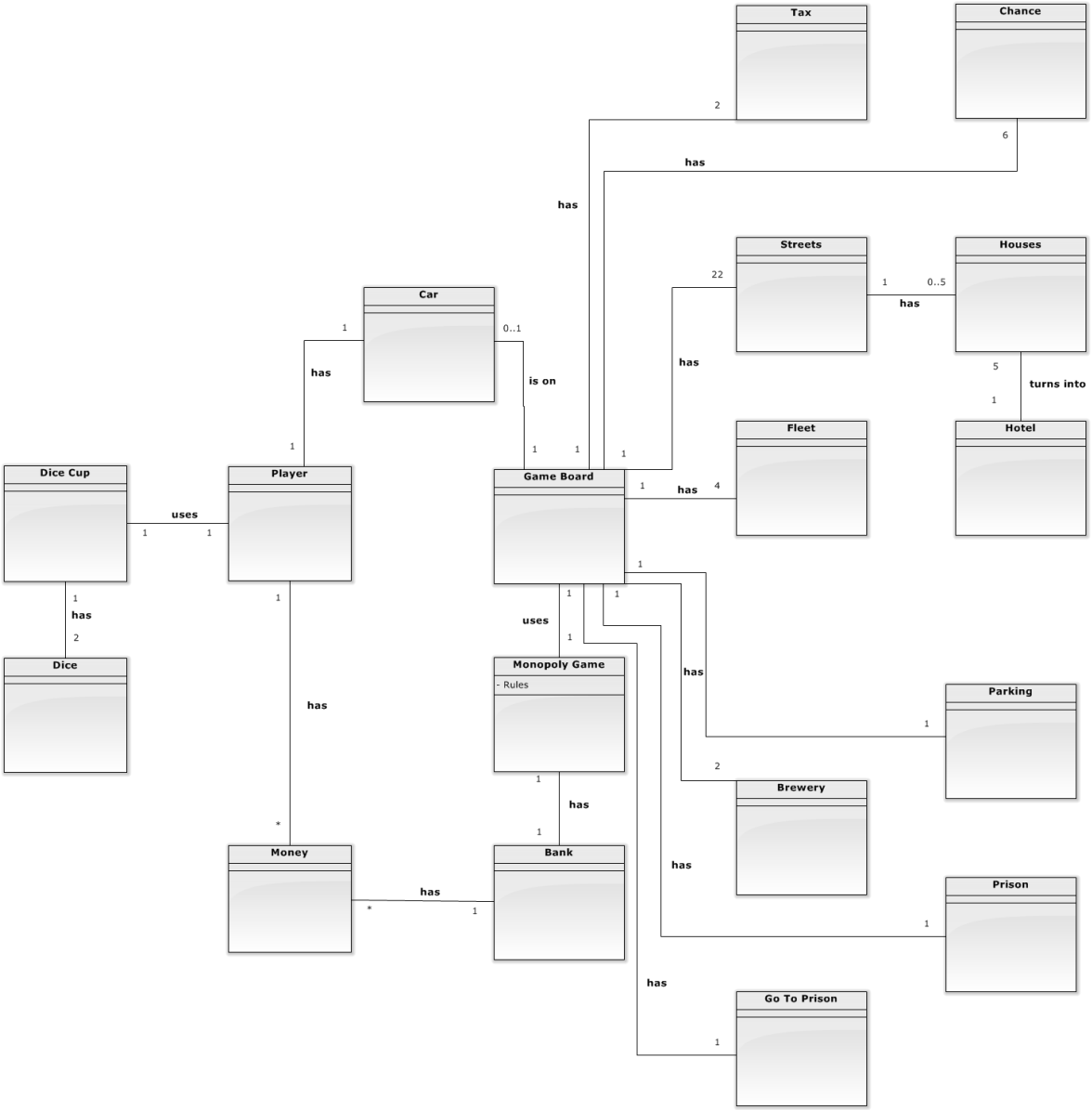
- Dokumentation
    - Genbrugte klasser
- Features
    - Glemme at opkræve leje fra en anden spiller
    - ”hurtigt spil”
    - Begrænsning af totalt antal huse og hoteller i spillet

Kravspecificering

- 2 - 6 spillere
- Hver spiller har en pengebeholdning
- Spilleren har en pengebeholdning med en startsum på 30.000 kroner
- 2 terninger
- Hver spiller slår med to terninger
- Slås to ens opnås ekstra slag
- Slås to ens, tre gange i træk, ryger man i fængsel
- 40 felter, med værdierne 1 – 40, felt 1 er startfelt
- Felterne er opdelt i:
    - Start
    - Gader
    - Rederier
    - Bryggerier
    - Prøv lykken
    - Skat
    - Fængsel
    - Parkering
- Felterne har events og dertilhørende valg/informationer. Disse beskrivelser er anskaffet fra Matador® fra BRIO® A/S
- Når en spiller slår med terningerne, rykker personen antal givne felter frem og får opstillet de tilhørende valg til feltet.
- Vinderen er den spiller, der har en formue på >0, mens alle andre har en formue på <0
- Opnås en formue <0 udgår, og taber, den pågældende spiller, spillet fortsættes dog
- Hvorvidt der er opnået en vinder angives efter at alle spillere har haft lige mange ture (ekstra-slag tæller ikke som ture)
- Hvis de to sidste spillere taber i samme tur, er spillet uafgjort
- Lejen på et felt kan forhøjes ved at købe huse på ens grunde
- Husene skal bygges ligeligt
- Husene kan sælges
- Grundene kan pantsættes
- Skal kunne spilles på DTU's databarer (Windows 7, service-pack 1, 64-bit, java 1.6)

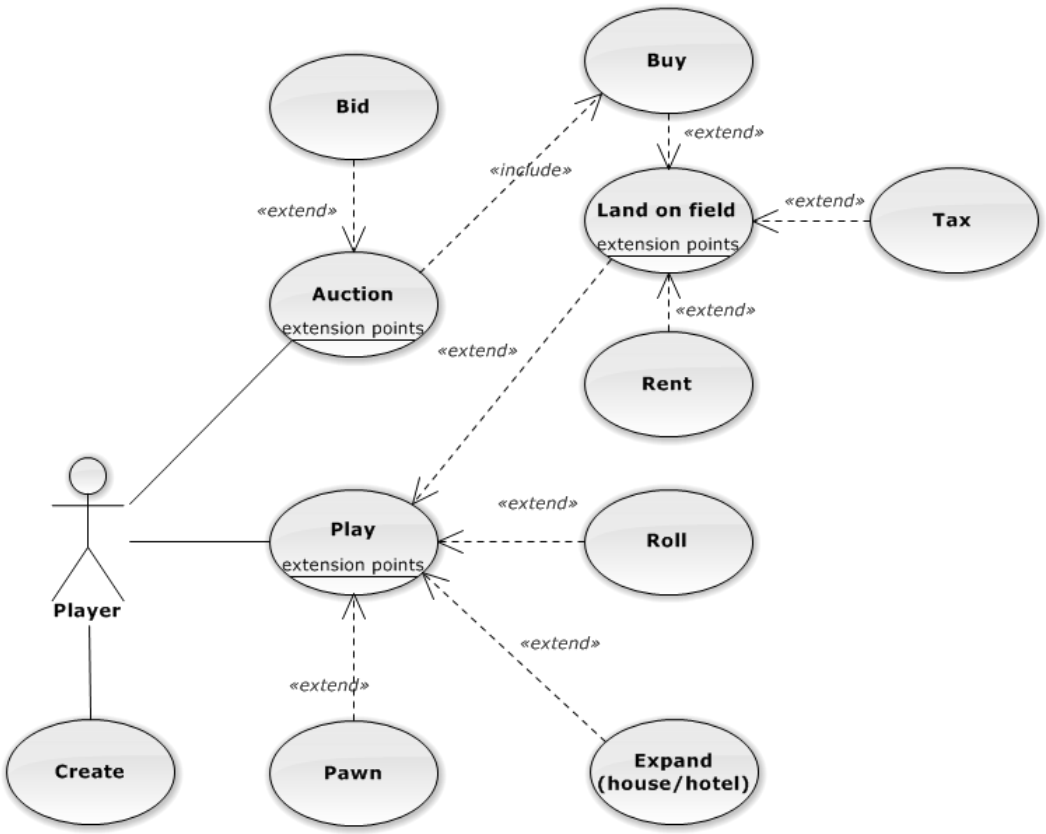
Domæne model

Domæne modellen er den model der beskriver system så tæt på virkeligheden som muligt. Det anvendes ikke til at kode efter, men derimod som en sikkerhed, så programmet kan planlægges.
Ud fra domænemodellen laves Use Case diagrammerne også med det samme, det kan lade sig gøre, da man ved hjælp af domæne modellen kan kortlægge alle de features der bør være med i spillet. Domæne modellen er derfor mest hjælp til ham der opretter spillet, frem for at være til hjælp for andre programmører der skal overtage vedligeholdelsen af spillet.



Figur 1

Use Case Diagram



Figur 2

Use Case Beskrivelse

Der er jf. Use Case diagrammet en enkelt use case:

- Play game.

Use Case "Play game"

Scope: Brætspil (spil).
Level: Bruger mål.
Primary actor: 2-6 spillere.
Stakeholders and Interest:
- Spiller: Ønsker at spille spillet.
- Firma: Ønsker et matador spil, der kan afvikles på Windows-platformene i DTUs databarer.
Preconditions: minimum to spillere som ønsker at spille spillet.
Success guarantee: Der skal findes en vinder eller et remis.
Main success scenario:
1. Der vælges antal af spillere fra 2 til 6
2. Spillerne vælger et navn
3. Spilleren flytter til et felt, der bestemmes ud fra det antal øjne, han slår med terningerne
4. Udfører handling alt efter feltets type
5. Turen går videre til næste spiller i rækken
6. Spillet fortsætter til alle spillere på nær en er bankerot, eller remis

Extensions:
1. Når en spiller lander på et "Fleet"-felt, kontrolleres om feltet er ejet
a. Hvis feltet er ejet kontrollere systemet hvor mange "Fleet"-felter ejeren ejer.
b. Ud fra hvor mange felter ejeren ejer betales en afgift til ejeren på
1. "Fleet"-felts ejerskab = 500
2. "Fleet"-felters ejerskab = 1000
3. "Fleet"-felters ejerskab = 2000
4. "Fleet"-felters ejerskab = 4000
c. Hvis det pågældende felt som spilleren lander på ikke er ejet, bliver spilleren tilbudt at købe feltet for 4000
2. Når en spiller lander på et "Brewery"-felt, kontrolleres om feltet er ejet
a. Hvis feltet er ejet kontrollere systemet om hvor mange "Breweries"-felter ejeren ejer.
b. Ud fra hvor mange felter ejeren ejer betales en afgift til ejeren på
1. "Breweries"-felts ejerskab = 100 x terningernes sum
2. "Breweries"-felters ejerskab = 200 x terningernes sum
c. Hvis det pågældende felt som spilleren lander på ikke er ejet, bliver spilleren tilbudt at købe feltet for 3000
3. Når en spiller lander på "Street", kontrolleres om feltet er ejet
a. Hvis feltet er ejet skal spilleren betale en leje på mellem 100-4000
b. Hvis feltet ikke er ejet kan det købes for mellem 1000-8000
4. Når man lander på "Tax" – feltet ekstra skat, skal der betales 2000
5. Når man lander på "Tax" – feltet indkomstskat, har spilleren muligheden enten at betale 4000 eller 10 % af spillerens formue
6. Når man lander på et "Chance"-felt får man et kort med et event, der kan have positiv indflydelse eller negativ indflydelse på din position i spillet.

Special requirements:
- Spillet skal kunne afvikles på Windows-platformene i databarerne på DTU.
Frequency of occurence: En gang pr. spil.

Use Case "Auction"

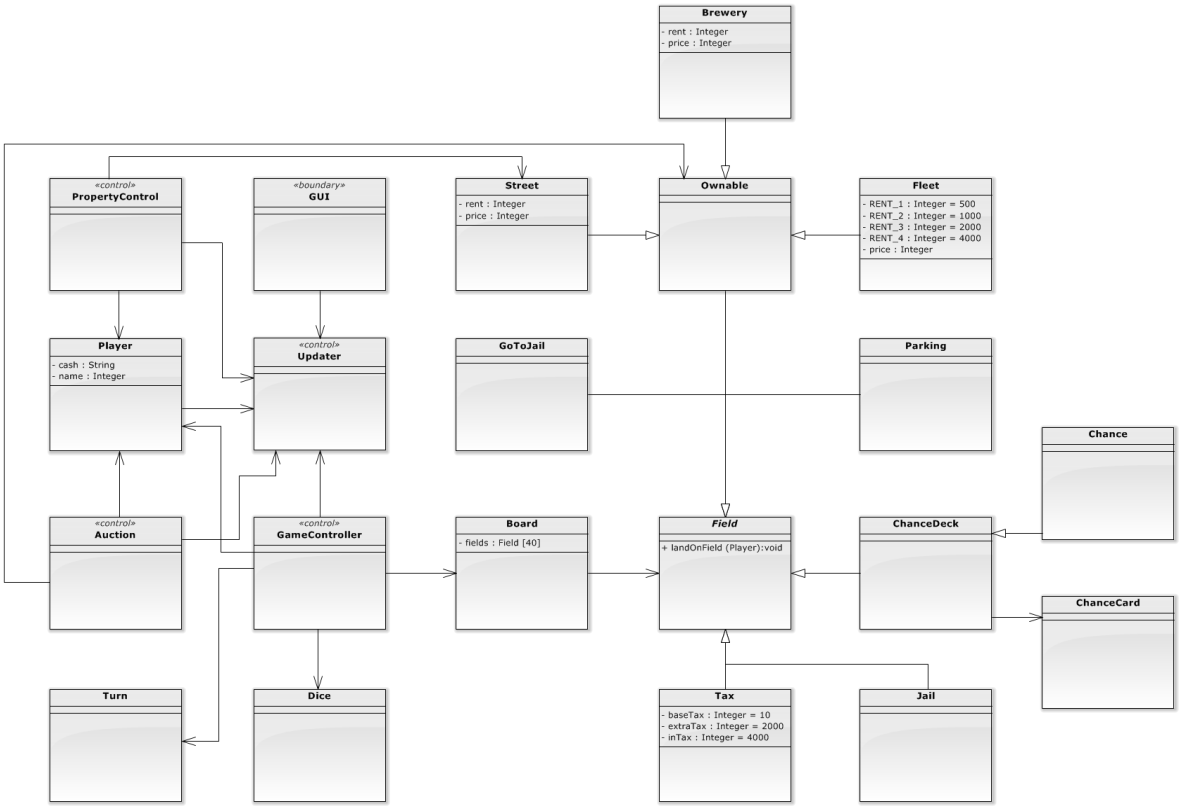
Success guarantee: Der skal enten findes en køber, eller grunden skal gøres mulig igen.
Main success scenario:
1. Spilleren der lander på et felt vælger ikke at købe det og starter nu en auktion
2. De andre spillere vælger hvor vidt om de vil være med til auktionen
3. Spillerne der er med i auktionen får mulighed for at byde på grunden
4. vinderen af auktionen får grunden

Extension

1. Ingen vil være med i auktionen
a. Grunden bliver ikke købt og den bliver sat til salg som før igen.
2. Der er kun 2 spillere
a. den anden spiller får mulighed for at købe grunden til salgspris.

BCE

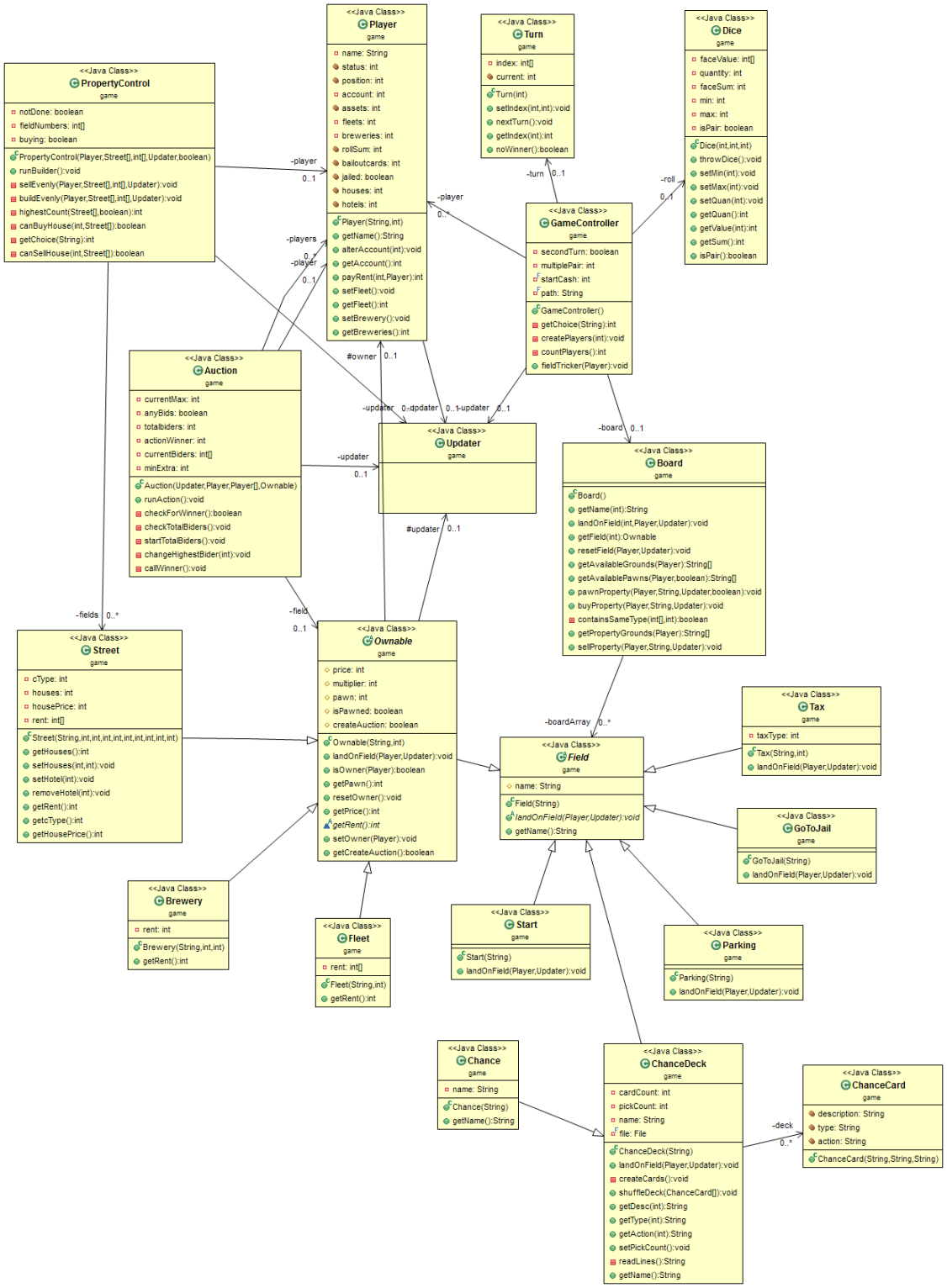
BCE-modellen er delt op i Controllere, Entities og Boundaries. Den eneste Boundary er GUI'en da det er det eneste sted der er noget der visuelt sker. Alle Controllere har <<Control>> skrevet øverst. De fleste af Controllerne er ikke udelukkende Controllere, men da det er deres hovedfunktion er de blevet tildelt titlen. Ud fra BCE modellen kan det ses at GUI'en har en pil i modsat retning af en normal boundary, dette er netop fordi den er grafisk og kræver noget svar på den info den selv sender.



Figur 2.5

Design Klasse Diagram

Design klasse diagrammet viser alle relationer mellem klasserne. Dette diagram er lavet ud fra selve programmet og er derefter ryddet op i. Design klasse diagrammet er et diagram der nemt kan læses af andre programmører, hvilket gør det nemt at give en opgave videre.



Figur 3

GRASP

GRASP står for "General Responsibility Assignment Software Patterns" og er en måde at arbejde med objekt orienteret design, når der skal kodes større systemer. Anvendes GRASP korrekt skabes der et overskueligt og let læseligt program.

GRASP arbejder med ni forskellige inddelinger af klasserne i et program. Klasserne i programmet bliver inddelt efter, hvordan de arbejder med objekterne de råder over.
Inddelingerne vi har arbejdet med i dette kursus lyder således:

- Controller
- Creator
- Information Expert
- High Cohesion
- Low Coupling
- Polymorphism

Controller

Controllerne er de klasser der skal styre forløbet. En controller klasse vil oftest være den klasse, der samler det hele, inden det bliver videregivet til den User Interface brugeren kommer i kontakt med. Man kan beskrive det som laget lige bag brugerens User interface.

GRASP's controller skal hovedsageligt stå for at uddelegere og holde styr på arbejdet og ikke gøre noget arbejde selv.

Gruppens controllere holder sig så tæt på GRASP's principper som muligt, Game Controlleren kan dog forbedres, da den selv står for nogle metoder, som at blive fængslet. Dette blev en nødvendighed grundet tidspres.

Creator

Creator klasserne står for at oprette objekterne der arbejdes med. Målet er at de ikke gør andet end at oprette objekter, som controlleren kan flytte rundt på.

I vores tilfælde har vi en Board klasse der både fungerer som Creator og som Controller. Dette kunne være delt yderligere, hvilket kunne have givet overflødige klasser og høj kobling. En mellem vej kunne dog have været fundet med mere tid.

Information Expert

Informations-eksperterne er de klasser der indeholder information, og som andre klasser henter informationen, de skal anvende, fra.

Som tidligere principper inden for GRASP er det vigtigt at en informations-ekspert holder sig til næsten kun at være det.

Et eksempel fra koden er klassen Turn, der udelukkende indeholder information andre anvender.

High Cohesion
High cohesion, eller høj sammenhørighed, er noget man stiler efter inden for GRASP. Høj sammenhørighed hører oftest sammen med lav binding, hvilket jeg vender tilbage til. Høj sammenhørighed opstår når man sørger for at have fokus på det man skal anvende og undgår for meget irrelevant information.

Dette er noget gennemgående, der sørges for i hele systemet.

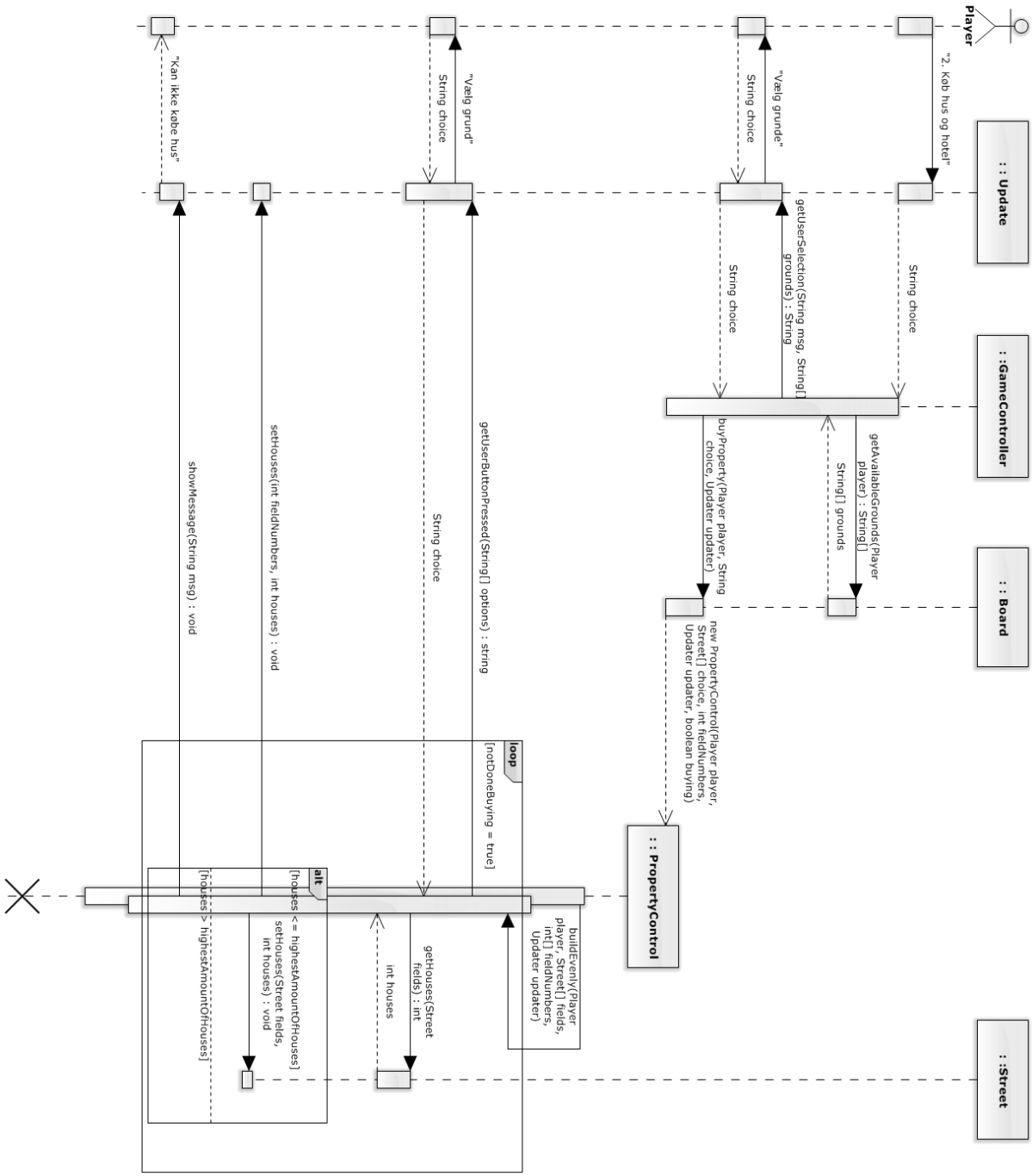
Low Coupling
Low coupling, lav binding på dansk, sørges for ved at have så få associationer klasserne imellem. Man forsøger at opnå den mængde bindinger, der er nødvendige for at klasserne kan fungere med hinanden. Dette skaber, som tidligere nævnt, også høj sammenhørighed klasserne imellem.

Ligesom den høje sammenhørighed, er lav binding også noget, der sørges for, gennem hele opbygningen af systemet.

Polymorphism
Polymorfi har forskellige betydninger når det kommer til kodning, men i forhold til GRASP er det relationen mellem klasser. Det ses ofte ved nedarving og kan tydeligt følges i den LandOnField metode der anvendes.

Implementering
I implementerings-afsnittet tager vi fat i tre af de dele der var nye for os i projektet, Køb og salg af huse, Auktion og Pantsætning.

Køb og salg af huse, Sekvensdiagram
Den store udfordring ved køb og salg af huse, var at få fat i grundene, der var fra samme gruppe, og som var ejet af brugeren. Særligt når ”updater”-objektet jævnligt skal fødes med strenge, og samtidigt returnerer strenge. Der blev forsøgt at aflaste både Board og GameController så vidt som muligt, ved at lave en seperat ”controller”, som tager sig af selve opdateringen af Street-felternes antal huse, i forhold til, hvad der blev valgt fra brugerens side af. Værd at notere sig er, at PropertyControl kun eksisterer, så længe brugeren har køb-/salgsprocessen kørende.


Player
: : Update
: :GameController
: : Board
: : PropertyControl
: :Street
"2. Køb hus og hotel"
"Vælg grunde"
String choice
getUserSelection(String msg, String[] grounds) : String
String choice
getAvailableGrounds(Player player) : String[]
String[] grounds
buyProperty(Player player, String choice, Updater updater)
new PropertyControl(Player player, Street[] choice, int fieldNumbers, Updater updater, boolean buying)
"Vælg grund"
String choice
getUserButtonPressed(String[] options) : string
String choice
"Kan ikke købe hus"
showMessage(String msg) : void
setHouses(int fieldNumbers, int houses) : void
loop
[notDoneBuying = true]
buildEvenly(Player player, Street[] fields, int[] fieldNumbers, Updater updater)
alt
[houses <= highestAmountOfHouses]
setHouses(Street fields, int houses) : void
[houses > highestAmountOfHouses]
getHouses(Street fields) : int
int houses

Figur 4

Auktion, Sekvensdiagram

I auktion har det været udfordrende at finde på en måde at afgøre at der skal afholdes en auktion, uden at bryde vores GRASP.

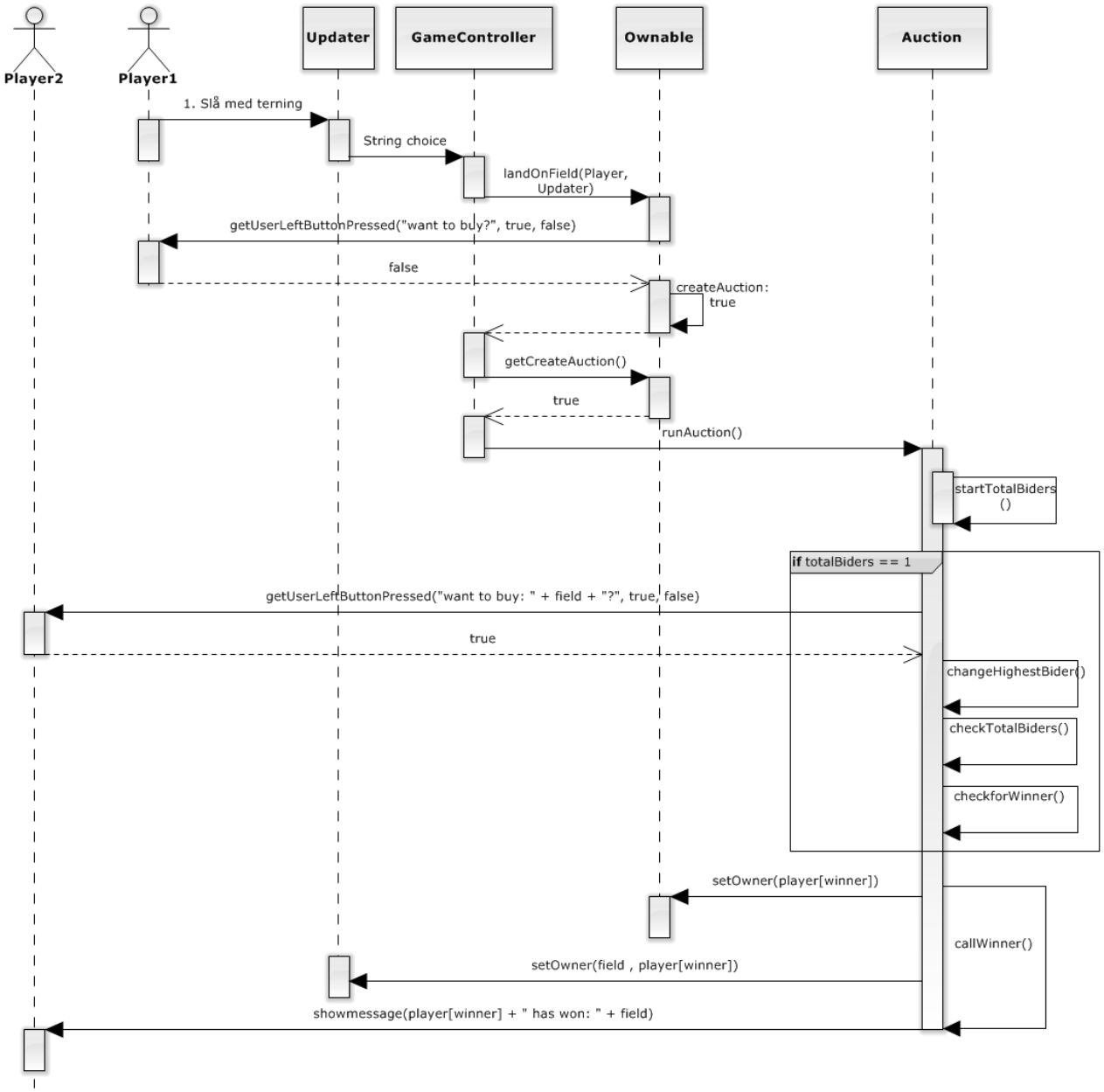
Vi ville gerne holde vores landOnField metoder i fields som void, så vi måtte tilføje en get-metode i ownables for at kunne skaffe information i vores GameController fra Ownables.

Men det for at undgå at få for mange unødvendige metoder i de andre klasser har vi kun tilføjede getCreateAuction i ownables, hvilket så giver problemer efter som vi, hverken har været land på et felt

bliver spurgt om der skal laves en auktion, hvilket fører til en ClassCastException.

Dette har vi så drejet uden om ved at benytte os af try og catch exception, hvilket har resulteret i vores auktions metode.

Figur 4.5

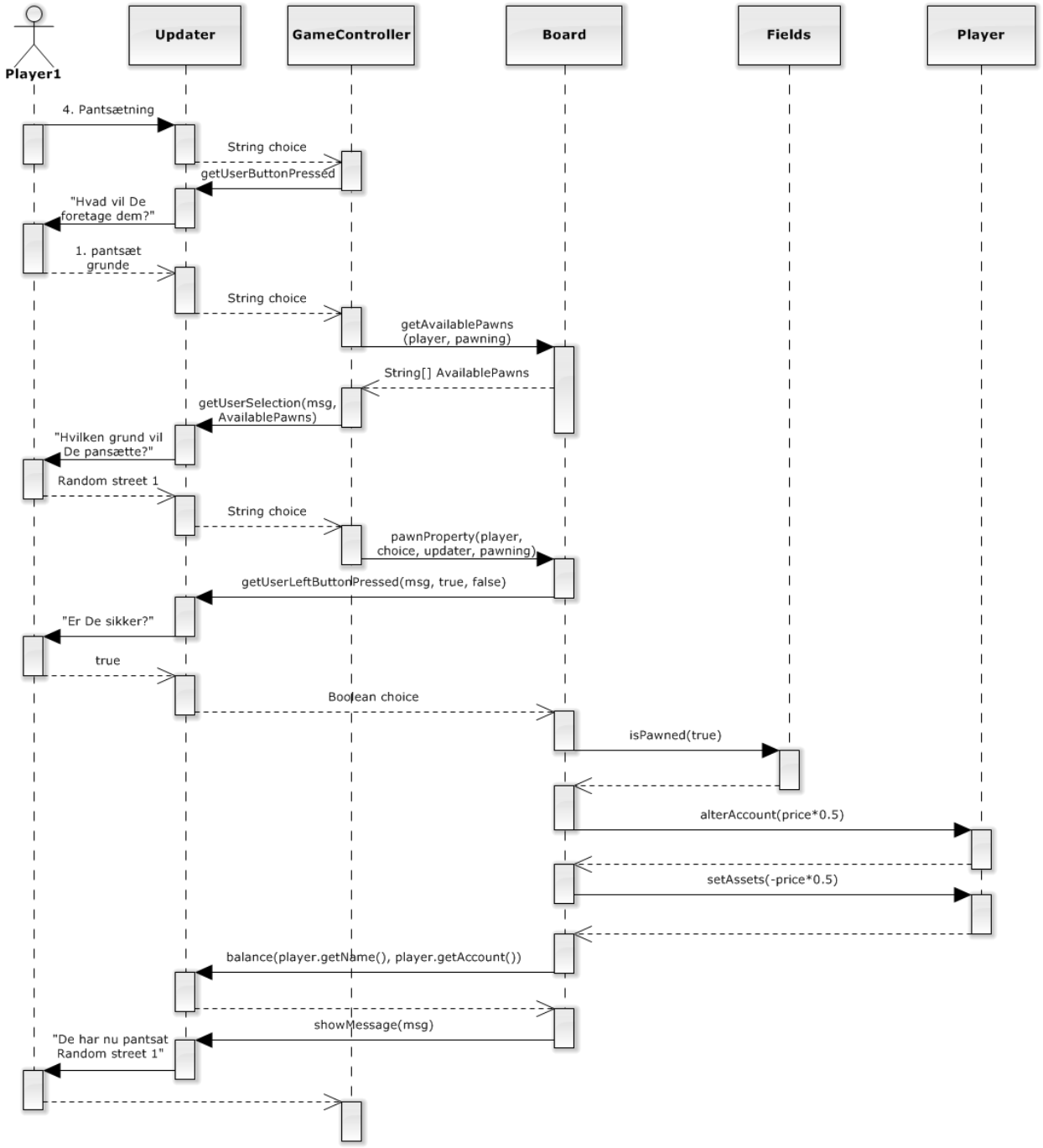


Pantsætning af grunde, Sekvensdiagram

Pantsætning har vi kunne implementere i vores program ved simpelt at indsætte nogle metoder i nogle af de eksisterende klasser vi havde.

Hvilket har gjort det nemt at implementere sent i vores programmerings forløb, samt lit ingen yderligere lastning på forholende mellem klasserne.

Det har været godt at få indført, da det giver en meget bedre fleksibilitet hos spilleren, og han kan have en større følelse af at han har et indspil på spillet udfald i sidste ende.



Figur 5

Test

Testen blev besværliggjort, da der ikke har været den fornødne indsigt i JUnit til at kunne udføre de korrekte tests. I stedet er der blevet søgt efter fejl manuelt, på de klasser der siden seneste aflevering er blevet implementeret – herunder køb/salg af huse/hoteller, aukton og pantsætning.

Manuel test

Måden der blev testet på, var, at ændre antallet af terninger til en enkelt, samt at der kun kunne kastes værdien 1, så der blev rykket gradvist fremad på brættet, felt for felt. Der blev tilføjet to spillere til spillet, hvor spiller 1, som får lov til at kaste først, ville få lov til at kaste så længe han ikke rykkede i fængsel. Der blev derfor købt flg. grunde:

- Hvidovrevej
- Rødovrevej
- Roskildevej

- Valby Langgade
- Allégade
- Frederiksberg Allé

```
private Board board = new Board();
private Dice roll = new Dice(1, 1, 1);
private boolean secondTurn = false;
```

Figur 6 – Parametrene for dice bliver ændret til 1, 1, 1 – dvs. der kan slås minimum 1, maksimalt 1 og der slås med 1 terning

På denne måde kan der testes for, om det er muligt at købe huse til grupper af grunde af både 2 og 3 (Hvidovrevej og Rødovrevej er en gruppe for sig, mens Roskildevej, Valby Langgade og Allégade er en gruppe for sig.) Til sidst skulle den liste af mulige grunde ikke returnere Frederiksberg Allé, da spiller 1 ikke ejer alle grunde af samme type.

Samtidigt bliver auktion, skattefeltet 10% / 4.000 og besøg af fængsel også testet (prøv lykken-felterne er diskuteret længere nede.)

- Auktion på Scandlines (Helsingør-Helsingborg), hvor spiller 2 siger ja til at købe.
- For skattefelt bliver der valgt 10%.

Spiller 1 bruger altså 11.600 på køb af grunde, og 10 % af sin beholdning på skat, og skulle gerne slutte med 15.640, som der kan købes huse for.

Når alle grundene er købt, kan der nu testes for, om der kan købes huse.

Der vil blive købt huse til de blå felter, samt til de tre orange felter.

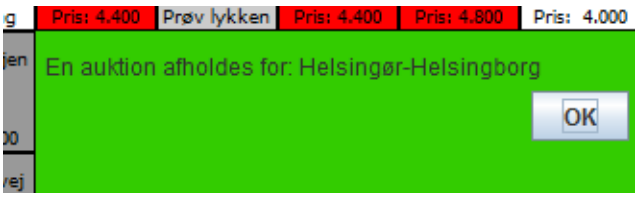
Der skal testes for, om der bliver købt jævnt på felterne, så der ikke kan købes f.eks. 2 huse på et felt, mens de andre ikke har minimum 1 hus hver.

Figur 6 – Spiller 1 køber løs

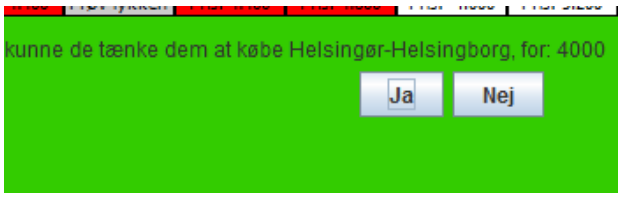
Der skal yderligere testes for, om der kan købes hoteller.

Hvis der bliver købt to huse til de blå felter, samt et enkelt til hver af de orange felter, bliver der brugt

Når husene er købt, skal de sælges igen. Her skal der igen testes for, at der bliver solgt huse på en sådan måde, at der stadig er en jævn bebyggelse af huse på grundene.



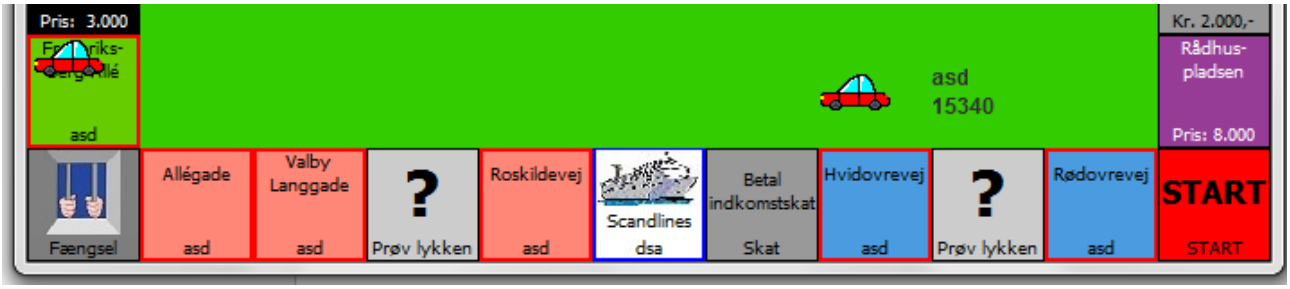
Figur 8



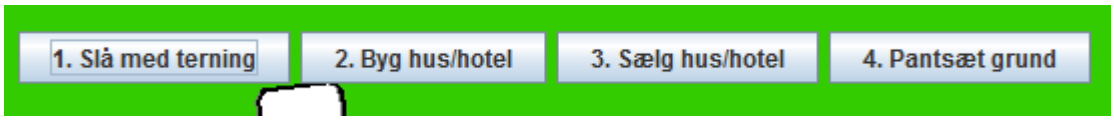
Figur 7

Husene sælges selvfølgelig til den halve pris, som de bliver købt til.

Figur 9 – det endelige resultat for auktionen. Da der kun er en enkelt spiller, bliver spilleren spurgt, om han/hun vil købe grunden til udbudspris.



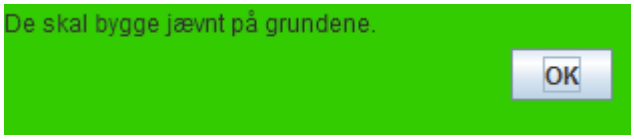
Figur 10 – endeligt kan der ses, at rød bil (spiller 1) ejer de korrekte grunde, mens blå bil (spiller 2) ejer Scandlines, som han/hun vandt på auktion.



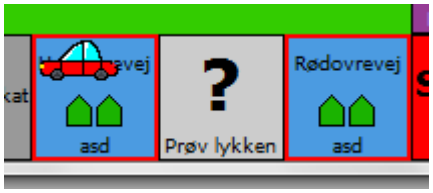
Figur 11 – Spiller 1 vælger selvfølgelig "2. Byg hus/hotel"

Figur 12 – vi får en liste med grunde som ejes, og som der købes grunde til

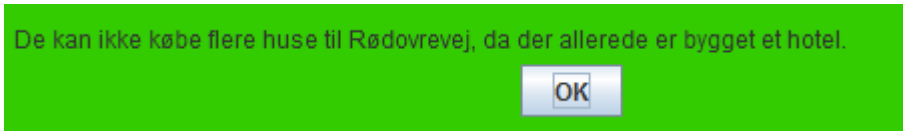
Figur 13 – Der købes grund til Rødovrevej og der noteres, at 1.000 bliver trukket fra konto



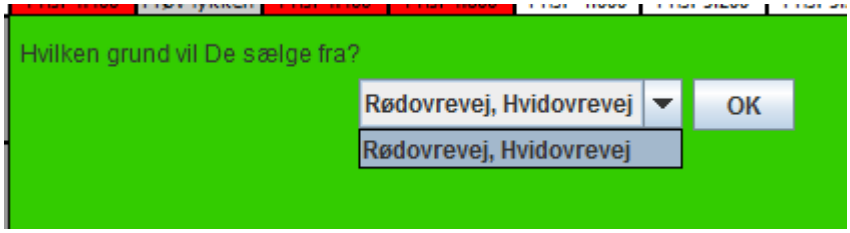
Figur 14 – og der returneres en fejlmeddelelse, at der ikke kan købes et hus, da reglen om jævn bebyggelse ellers ikke bliver overholdt.



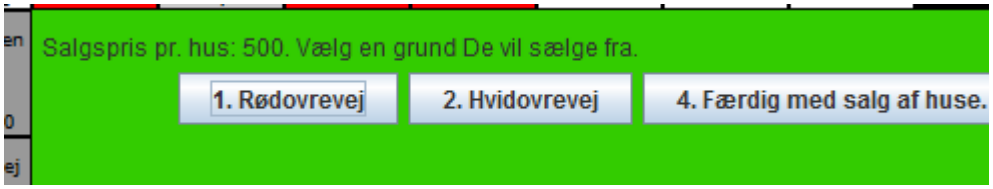
Figur 15 – GUI er korrekt opdateret med det samme antal huse, som der er blevet købt for.



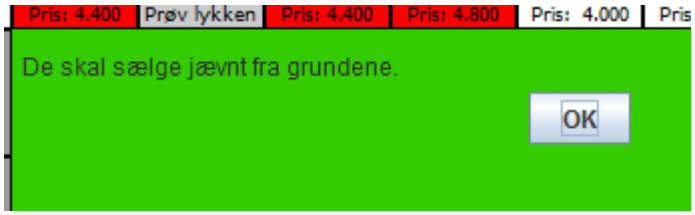
Figur 16 – der bliver kontrolleret for, om der er hoteller (og derfor er det ikke muligt at købe flere huse), og ser også at GUI bliver opdateret korrekt.



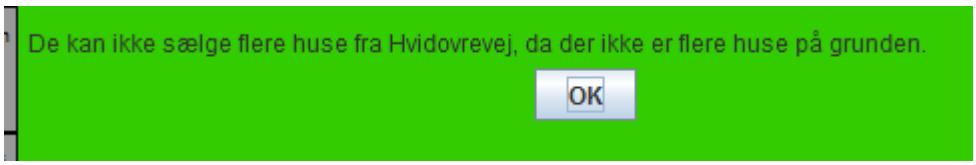
Figur 17 – der bliver korrekt listet Rødovrevej og Hvidovrevej, som mulige grunde at sælge ud fra, da de indeholder huse.

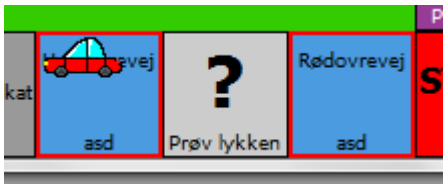


Figur 18 – der sælges ud af Rødovrevej, og igen forsøges der at sælge ud, så der ikke er jævn bebyggelse.



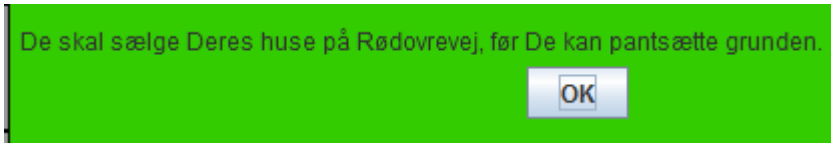
Figur 19 – korrekt fejl ved forsøg på at sælge ud, så der ikke er jævn bebyggelse.

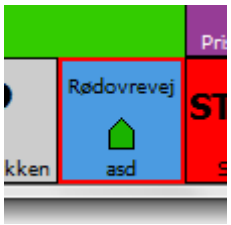




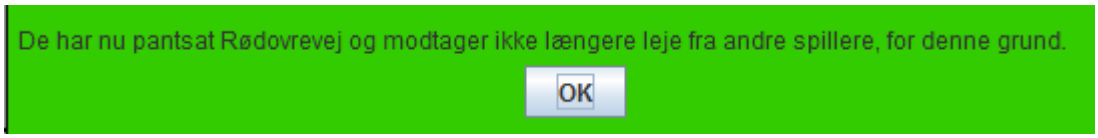
Figur 20 – og der kan selvfølgelig ikke sælges flere huse, hvis der ingen huse er. Samtidigt bliver GUI korrekt opdateret, og fjerner grundene.

Fra spillemenuen vælger vi nu at pantsætte. Inden vi gør sådan, køber vi huse til Rødovrevej og Hvidovrevej. Det skulle ikke være muligt at pantsætte grunde, der indeholder huse.





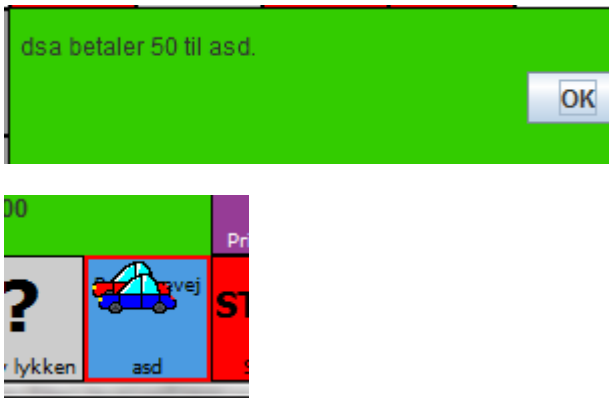
Figur 21 – og der opstår selvfølgelig korrekt fejl. Man bliver bedt om at sælge sine huse, før man kan pantsætte grunden. Husene bliver solgt manuelt fra spillemenuen.



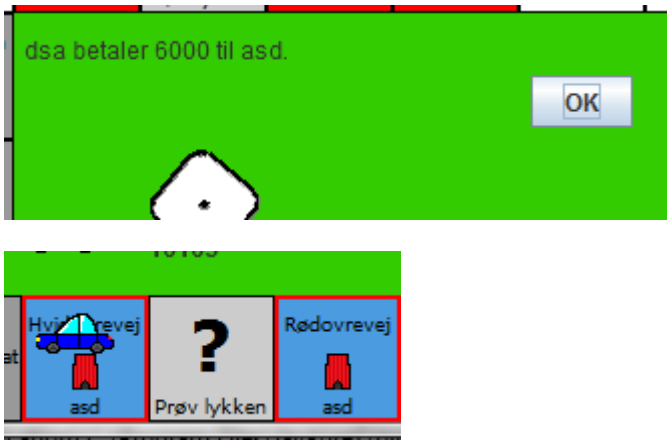
Figur 22 – efter at have solgt grundene, er det nu muligt at pantsætte grunden.

Test af leje med huse på grund

Der skal nu testes for at der bliver betalt korrekt leje. Muligheden for at slå en ekstra gang bliver fjernet, mens spiller 1 køber de blå felter. Spiller 2 vil naturligvis lande på Rødovrevej og betale almindelig leje, før spiller 1 når at købe Hvidovrevej, og derfor har mulighed for at købe huse til grundene. Spiller 1 køber efter at have landt på



Figur 23 – spiller 2 (blå) betaler korrekt 50 kr. til spiller 1 (rød), som ejer feltet..



Figur 24 – spiller 2 betaler korrekt 6.000 for at lande på Hvidovrevej, som har et hote bygget.

Kendte fejl

Af bekendte fejl vedr. køb og salg af huse, samt pantsætning af disse grunde, er der flg. fejl:

- Hvis man forsøger at købe huse, uden at eje nogen grunde (ingen grunde overhovedet), bliver der lavet en kritisk fejl, hvor spillet stopper med at fungere (nullPointerException, da der bliver sendt et tomt array.)
- Hvis man forsøger at sælge huse, uden at eje nogen grunde, sker der samme nullPointerException (af samme grund.)
- Samme fejl sker der, hvis man forsøger at pantsætte, uden at eje nogen grunde.
- Hvis man har en gruppe af grunde af samme type, hvor der er bygget huse på et eller to af grundene, kan den sidste grund, som ikke har noget hus på sig, blive pantsat (hvilket er mod reglerne.)

Konklusion

Gruppen har gennem de sidste to uger formået at lave et spilbart matadorspil, med meget få manglende features. Der er blevet anvendt det gruppen har lært i kurset gennem det sidste 13-ugers semester. Konklusionen er som følger, det er lykkedes at kode et matador spil der virker. Det kunne optimeres og andre features kunne tillægges, men grundet tidspres og muligheder er visse features fravalgt i det endelige produkt.

Mangler

Da det var opgaven at lave et så virkeligt matadorspil som muligt, blev der selvfølgelig taget udgangspunkt I, hvordan man spiller matador og dets spilleregler. Mens det selvfølgelig havde været ønsket, var der på forhånd en opfattelse af, at der ikke ville kunne blive udviklet samtlige features, som der også er at finde I matador.

I forhold til MoSCoW, blev de fleste features dog alligevel udviklet – selvom helt ned til "Could"-listen.

Der er dog også mangler mellem de udviklet features, som ikke helt er kommet I land. Bl.a. blev der som "Must" sat op, at "Prøv lykken"-felterne skulle virke. De blev aldrig helt færdige (nogle af "Prøv lykken"-kortene fra det oprindelige spil blev sløjfet.)

En komplet liste over manglende funktioner I forhold til MoSCoW:

Must:

- Dokumentation
    - ~~Javadocs til dokumentation af kode~~
    - ~~Detaljerede beskrivelser af klasserne:~~
        - ~~Fields (mere bestemt Streets)~~
        - ~~Forskellige controllere~~
        - ~~Chance (Prøv lykken)~~
        - ~~Jail (fængsel)~~
- Features
    - ~~Alle Klasser~~
    - ~~Alt brug skal forgå i GUI~~
    - ~~Ownable funktionerne (eje, leje samt købe)~~
    - ~~Land on field funktionen~~
    - Chance, Jail og buyout funktioner (Chance/Prøv lykken kun delvist færdiggjort)
    - Vinder og taber(fallit) funktioner (mens man kan gå fallit, har man ikke mulighed for at sælge huse/hoteller og pantsætte, skulle man have haft mulighed for at betale den opkrævet leje herefter.)

Should:

- Dokumentation
    - ~~Domæne model~~
    - ~~BCE model~~

    - ~~Use case model (Diagram samt beskrivelse~~)
    - Test samt Beskrivelse (test blev ikke tilfredsstillende, grundet manglende indsigt i JUnit.)
    - ~~Design klasse diagram~~
    - ~~Design sekvens diagram~~
    - ~~Grasp beskrivelse~~
- Features
    - ~~Huse og hotel funktionerne~~
    - ~~Ekstra slag ved et ens slag~~
    - ~~Flere ejerskabs bonus ved Brewery(bryggeri) og Fleet(rederi)~~
    - ~~Tax felt funktion~~
    - Ingen køb i først runde
    - ~~Slå ens for at komme ud af fængsel~~

Could:

- Dokumentation
    - Fuld FURPS+ beskrivelse
- Features
    - ~~3 x ens slag i træk smider spiller i fængsel~~
    - ~~Pantsætning af Streets funktion~~
    - ~~Jævn fordeling af huse på et gade område~~

Want:

- Dokumentation
    - Undersøgelser
        - Kvantitativt
        - Kvalitativt
- Features
    - ~~Grafisk førelse af bilernes bevægelse~~
    - ~~Auktioner funktion for ikke købt grund~~
    - Bytte funktion blandt spillerne

Sorted out:

- Dokumentation
    - Genbrugte klasser
- Features
    - Glemme at opkræve leje fra en anden spiller
    - ”hurtigt spil”
    - Begrænsning af totalt antal huse og hoteller i spillet

Ser man på listen som helhed, og hvad der er blevet nået i forhold til det, der ikke blev nået, er der umiddelbart tilfredshed at kunne spore. Det var måske lidt naivt at tro, at vi ”kun” med tre gruppemedlemmer, kunne nå at få lavet det hele.

Udover manglerne, viste der sig også nogle småfejl hist og her, som der ikke har været tid til at rette. Om end de fejl der er blevet fundet ikke er ønskelige, er det fejl, som hvis man er bekendt med dem, aktivt kan undgås og derved ikke have indflydelse på spillets gang.

Værd at notere sig er dog, at én af de features, som der er aktuelt i det oprindelige matadorspil, nemlig det at kunne bytte grunde, viser sig at gøre spillet meget langtrukken, og chancen for at man nogensinde får mulighed for at købe grunde, stort set ikke er mulig, med mindre man spiller i meget lang tid, og kun er to mand – dette skulle være blevet forudsat tidligere i analysen, og prioteret højere.

Under alle omstændigheder, er der blevet lavet et matadorspil, som i en umiddelbar tilfredsstillende grad kan spilles, selv med de få fejl der har vist sig. Og det ville, hvis de få fejl var blevet rettet, blevet til et tæt på fuldt tilfredsstillende produkt.

Kode

Auction.java

```
1 package game;
2
3 public class Auction {
4 private int currentMax;
5 private boolean anyBids = false;
6 private int totalbiders = 0;
7 private int actionWinner; // For array list
8 private int[] currentBiders;
9 private Player player;
10 private Player[] players;
11 private Updater updater;
12 private Ownable field;
13 private int minExtra = 100; //Minimum for the difference in the next bid
14
15 /**
16 * Auction Constructor </p>
17 * Created every time a Auction is announced
18 * @param player - The player which denied the purchase, and isn't allowed to make a
bid
19 * @param players - The whole player Array
20 * @param field - The field the auction runs over
21 */
22 public Auction(Updater updater, Player player, Player[] players, Ownable field) {
23
24 this.field = field;
25 this.currentMax = field.getPrice();
26 this.currentBiders = new int[players.length];
27 this.player = player;
28 this.players = players;
29 this.updater = updater;
30
31 }
32 /**
33 * run Action
34 * Runs the Action with the infomation given from the Auction Constructor
35 */
36 public void runAction(){
37
38 updater.showMessage("En auktion afholdes for: " + field.getName());
39
40 int i = 0;
41
42 startTotalBiders();
43 //If there is only one possible buyer from the start (normally 2 player game)
44 if (totalbiders == 1){
45 while(i < players.length){
46 if(currentBiders[i] == 1 && players[i].getAccount() >= currentMax){
47 if(updater.getUserLeftButtonPressed(players[i].getName() + " kunne de
tænke dem at købe " + field.getName() + ", for: " + field.getPrice(), "Ja" , "Nej")){
48 anyBids = true;
49 changeHighestBider(i);
50 checkTotalBiders();
51 checkForWinner();
52 }
```

```
53 else {
54 currentBiders[i] = 0;
55 checkTotalBiders();
56 checkForWinner();
57 }
58 }
59 else {
60 currentBiders[i] = 0;
```

```
61 checkTotalBiders();
62 checkForWinner();
63 }
64 i++;
65 }
66 }
67 //If there is more then one possible buyter from the start (3 or more active
players left in game)
68 if(totalbiders > 1){
69 while( i < players.length){
70
71 // Situation with more than 2 players - ask to bid on the already bidded
field
72 if(currentBiders[i] == 1 && anyBids && totalbiders >= 1 &&
players[i].getAccount() >= (currentMax + minExtra)) {
73 if(updater.getUserLeftButtonPressed(players[i].getName() + " kunne de
tænke dem at byde på " + field.getName() + ". Minimum bud: " + (currentMax + minExtra),
"Ja" , "Nej")){
74 currentMax = updater.getUserInteger("Hvor meget kunne de tænke dem
at byde? (minimum: " + (currentMax + minExtra) + "):", (currentMax + minExtra),
players[i].getAccount());
75 changeHighestBider(i);
76 //TODO set new currentMax and Find out how to identify person which
has highest current bid ( evt. currentBiders[i] = 2)
77 }
78 else currentBiders[i] = 0;
79 }
80
81 //Situation with more than 2 players - ask to bid on the not already bidded
field
82 else if(currentBiders[i] == 1 && !anyBids && totalbiders >= 1 &&
players[i].getAccount() >= (currentMax)) {
83 if(updater.getUserLeftButtonPressed(players[i].getName() + " kunne de
tænke dem at byde på " + field.getName() + ", for: " + field.getPrice(), "Ja" ,
"Nej")){
84 anyBids = true;
85 changeHighestBider(i);
86 }
87 else {
88 currentBiders[i] = 0;
89 }
90
91 }
92 else if(currentBiders[i] == 1 && players[i].getAccount() < (currentMax +
minExtra)){
93 currentBiders[i] = 0;
94 updater.showMessage(players[i].getName() + ", de har desværre ikke nok
penge i deres pengebeholdning til at kunne byde");
95 }
```

```
96 checkTotalBiders();
97 i++;
98 if (i >= players.length) i = 0;
99 if(checkForWinner() == true) i = 7;
100 else if (totalbiders == 0) i = 7;
101 }
102 }
103 callWinner();
104 }
105
106 /**
107 * Check for Winner
108 * @return True if there is a winner, otherwise false
109 * </P>
```

```
110 * Check if there is only 1 totalbiders left and sets actionwinner to the number of
the
winner on array
111 */
112 private boolean checkForWinner(){
113 if (totalbiders == 1) {
114 for(int i = 0 ; i < players.length ; i++){
115 if(currentBiders[i] == 2){
116 actionWinner = i;
117 return true;
118 }
119 }
120 return false;
121
122 }
123 // else if (totalbiders == 0){
124 // for(int i = 0 ; i < players.length ; i++){
125 // currentBiders[i] = 0;
126 // }
127 // return true;
128 // }
129
130 else return false;
131 }
132
133 /**
134 * check TotalBiders
135 * </p>
136 * Set totalbiders to the amount of people bidding, used while the auction is in
progress
137 */
138 private void checkTotalBiders(){
139 // Check how many players are allowed to bid (active players, and not the player
that could've bought the place)
140 this.totalbiders = 0;
141 for(int i = 0; i<players.length;i++){
142 if(!players[i].equals(player)){
143 if(players[i].getStatus() >= 0 && currentBiders[i] == 1){
144 this.currentBiders[i] = 1;
145 this.totalbiders++;
146 }
147 if(players[i].equals(player)){
148 this.currentBiders[i] = 0;
```

```
149 }
150 if(players[i].getStatus() >= 0 && currentBiders[i] == 2){
151 this.totalbiders++;
152 }
153 }
154 }
155 }
156
157 /**
158 * start TotalBiders
159 * </p>
160 * Set totalbiders to the amount of people bidding, used before the auction starts
161 */
162 private void startTotalBiders(){
163 this.totalbiders = 0;
164 for(int i = 0; i<players.length;i++){
165 if(!players[i].equals(player)){
166 if(players[i].getStatus() >= 0 && currentBiders[i]!=2){
167 this.currentBiders[i] = 1;
168 this.totalbiders++;
```

```
169 }
170 if(players[i].equals(player)){
171 this.currentBiders[i] = 0;
172 }
173 if(players[i].getStatus() >= 0 && currentBiders[i] == 2){
174 this.totalbiders++;
175 }
176 }
177 }
178 }
179
180 /**
181 * change HigestBider
182 * @param j - the player (int from array) which is now the highestbider
(currentBiders[player] = 2)
183 */
184 private void changeHighestBider(int j) {
185 for(int i = 0 ; i < players.length ; i++){
186 if(currentBiders[i] == 2) currentBiders[i] = 1;
187 }
188 currentBiders[j] = 2;
189 }
190
191 /**
192 * call Winner
193 * announce who's the winner, set owner and removes money
194 * </p>
195 * If there is no Winner it will end the Auction
196 */
197 private void callWinner(){
198 if(totalbiders == 1){
199 updater.showMessage(players[actionWinner].getName() + " har vundet auktionen på
" + field.getName() + " med sit bud på: " + currentMax);
200 field.setOwner(players[actionWinner]);
201 updater.setOwner(player.getPosition(), players[actionWinner].getName());
202 players[actionWinner].alterAccount(-currentMax);
203 players[actionWinner].setAssets((int)(field.getPrice() * 0.5));
```

```
204 updater.balance(players[actionWinner].getName(),
players[actionWinner].getAccount());
205 if (field instanceof Brewery){
206 players[actionWinner].setBrewery();
207 }
208 if (field instanceof Fleet){
209 players[actionWinner].setFleet();
210 }
211
212
213 }
214 else if(totalbiders <= 0){
215 updater.showMessage("Der var ingen bud på " + field.getName());
216 }
217 }
218 }
219
```


Board.java

```
1 package game;
2
3 public class Board {
4 private Field[] boardArray = new Field[40];
5
6 public Board() {
7 boardArray[0] = new Start("Start");
8 boardArray[1] = new Street("Rødovrevej", 1200, 1, 50, 250, 750, 2250, 4000, 6000,
1000);
9 boardArray[2] = new Chance("Create");
10 boardArray[3] = new Street("Hvidovrevej", 1200, 1, 50, 250, 750, 2250, 4000, 6000,
1000);
11 boardArray[4] = new Tax("Indkomstskat", 0);
12 boardArray[5] = new Fleet("Helsingør-Helsingborg", 4000);
13 boardArray[6] = new Street("Roskildevej", 2000, 2, 100, 600, 1800, 5400, 8000,
11000, 1000);
14 boardArray[7] = boardArray[2];
15 boardArray[8] = new Street("Valby Langgade", 2000, 2, 100, 600, 1800, 5400, 8000,
11000, 1000);
16 boardArray[9] = new Street("Allégade", 2400, 2, 150, 800, 2000, 6000, 9000, 12000,
1000);
17 boardArray[10] = new VisitJail("Fængsel");
18 boardArray[11] = new Street("Frederiksberg Allé", 2800, 3, 200, 1000, 3000, 9000,
12500, 15000, 2000);
19 boardArray[12] = new Brewery("Tuborg", 3000, 100);
20 boardArray[13] = new Street("Bülowsvej", 2800, 3, 200, 1000, 3000, 9000, 12500,
15000, 2000);
21 boardArray[14] = new Street("Gl. Kongevej", 3200, 3, 250, 1250, 3750, 10000, 14000,
18000, 2000);
22 boardArray[15] = new Fleet("Mols-Linien", 4000);
23 boardArray[16] = new Street("Bernstorffsvej", 3600, 4, 300, 1400, 4000, 11000,
15000, 19000, 2000);
24 boardArray[17] = boardArray[2];
25 boardArray[18] = new Street("Hellerupvej", 3600, 4, 300, 1400, 4000, 11000, 15000,
19000, 2000);
26 boardArray[19] = new Street("Strandvejen", 4000, 4, 350, 1600, 4400, 12000, 16000,
20000, 2000);
27 boardArray[20] = new Parking("Parkering");
28 boardArray[21] = new Street("Trianglen", 4400, 5, 350, 1800, 5000, 14000, 17500,
```

```
21000, 3000);
29 boardArray[22] = boardArray[2];
30 boardArray[23] = new Street("Østerbrogade", 4400, 5, 350, 1800, 5000, 14000, 17500,
21000, 3000);
31 boardArray[24] = new Street("Grønningen", 4800, 5, 400, 2000, 6000, 15000, 18500,
22000, 3000);
32 boardArray[25] = new Fleet("Gedser-Rostock", 4000);
33 boardArray[26] = new Street("Bredgade", 5200, 6, 450, 2200, 6600, 16000, 19500,
23000, 3000);
34 boardArray[27] = new Street("Kgs. Nytorv", 5200, 6, 450, 2200, 6600, 16000, 19500,
23000, 3000);
35 boardArray[28] = new Brewery("Carlsberg", 3000, 100);
36 boardArray[29] = new Street("Østergade", 5600, 6, 500, 2400, 7200, 17000, 20500,
24000, 3000);
37 boardArray[30] = new GoToJail("Ryk i fængsel");
38 boardArray[31] = new Street("Amagertorv", 6800, 7, 550, 2600, 7800, 18000, 22000,
25000, 4000);
39 boardArray[32] = new Street("Vimmelskaftet", 6000, 7, 550, 2600, 7800, 18000,
22000, 25000,4000);
40 boardArray[33] = boardArray[2];
41 boardArray[34] = new Street("Nygade", 6400, 7, 600, 3000, 9000, 20000, 24000,
28000, 4000);
42 boardArray[35] = new Fleet("Rødby-Puttgarden", 4000);
Page 1
Board.java
43 boardArray[36] = boardArray[2];
44 boardArray[37] = new Street("Frederiksberg", 7000, 8, 700, 3500, 10000, 22000,
26000, 30000, 4000);
45 boardArray[38] = new Tax("Statsskat",1);
46 boardArray[39] = new Street("Rådhuspladsen", 8000, 8, 1000, 40000, 12000, 28000,
34000, 40000, 4000);
47 }
48
49 /**
50 * get Name
51 * @param i - The field number (from array)
52 * @return The name of the chosen field
53 */
54 public String getName(int i) {
55 return boardArray[i].getName();
56 }
57
58 /**
59 * landOnField
60 * @param i - The field number (from array)
61 * @param player - The player which has landed on the chosen field
62 * @param updater - The Updater
63 */
64 public void landOnField(int i, Player player, Updater updater) {
65 boardArray[i].landOnField(player, updater);
66 }
67
68 /**
69 * get Field
70 * @param i - The field number (from array)
71 * @return Ownable field which was chosen
72 */
73 public Ownable getField(int i) {
74 return (Ownable) boardArray[i];
```

```
75 }
76
77 /**
78 * reset Field
79 * @param player - The player you would like to reset
80 * @param updater - The Updater
81 * </p>
82 * resets all ownership for the given player.
83 */
84 public void resetField(Player player, Updater updater) {
85 for(int i = 0; i < boardArray.length; i++) {
86 if(boardArray[i] instanceof Ownable) {
87 if(((Ownable) boardArray[i]).isOwner(player)) {
88 ((Ownable) boardArray[i]).resetOwner();
89
90 updater.removeOwner((i + 1));
91 updater.setHotel((i + 1), false);
92 updater.setHouses((i + 1), 0);
93 }
94 }
95 }
96 }
97 /**
98 * get Available Grounds
99 * @param player - The choice of player
100 * @return - The fields the chosen player owns
101 */
102 public String[] getAvailableGrounds(Player player) {
```

```
103 int count = 0;
104
105 for(int i = 0; i < boardArray.length; i++) {
106 if(boardArray[i] instanceof Street) {
107 if(((Street) boardArray[i]).isOwner(player)) {
108 count++;
109 }
110 }
111 }
112
113 Street[] available = new Street[count];
114 int[] availableTypes = new int[9];
115
116 count = 0;
117
118 for(int i = 0; i < boardArray.length; i++) {
119 if(boardArray[i] instanceof Street) {
120 if(((Street) boardArray[i]).isOwner(player)) {
121 available[count] = (Street) boardArray[i];
122 count++;
123 }
124 }
125 }
126
127 count = 0;
128
129 for(int i = 0; i < available.length; i++) {
130 for(int j = 0; j < available.length; j++) {
131 if(i != j) {
```

```
132 if(available[i].getName() == "Rødovrevej" || available[i].getName() ==
"Rådhuspladsen") {
133 if(available[i].getcType() == available[j].getcType()) {
134 availableTypes[count] = available[i].getcType();
135 count++;
136 }
137 }
138 else {
139 for(int k = 0; k < available.length; k++) {
140 if(i != k && j != k) {
141 if((available[i].getcType() == available[j].getcType()) &&
(available[i].getcType() == available[k].getcType())) {
142 if(!(containsSameType(availableTypes,
available[i].getcType()))) {
143 availableTypes[count] = available[i].getcType();
144 count++;
145 }
146 }
147 }
148 }
149 }
150 }
151 }
152 }
153
154 String[] availableGrounds = new String[count];
155
156 for(int i = 0; i < availableGrounds.length; i++)
157 availableGrounds[i] = "";
158
159 count = 0;
160 int counter = 0;
161
```

```
162 for(int a : availableTypes) {
163 for(Field b : boardArray) {
164 if(b instanceof Street) {
165 if(a == ((Street) b).getcType()) {
166 if(((Street) b).getName() == "Rødovrevej" || ((Street) b).getName()
== "Hvidovrevej" || ((Street) b).getName() == "Rådhuspladsen" || ((Street) b).getName()
==
"Frederiksberggade") {
167 if(counter == 1) {
168 availableGrounds[count] += ((Street) b).getName();
169 }
170 else {
171 availableGrounds[count] += ((Street) b).getName() + ", ";
172 counter++;
173 }
174 }
175 else {
176 if(counter == 2) {
177 availableGrounds[count] += ((Street) b).getName();
178 }
179 else {
180 availableGrounds[count] += ((Street) b).getName() + ", ";
181 counter++;
182 }
```

```
183 }
184 }
185 }
186 }
187
188 counter = 0;
189 count++;
190 }
191
192 System.out.println(availableGrounds[0]);
193
194 return availableGrounds;
195 }
196 /**
197 * get Available Pawns
198 * @param player - The choice of player
199 * @param pawning - choice between pawn or unpawn (true = pawn , false = unpawn)
200 * @return list of available pawns (or unpawns depens on choice of @param pawning)
201 */
202 public String[] getAvailablePawns(Player player, boolean pawning) {
203 int count = 0;
204
205 for(int i = 0; i < boardArray.length; i++) {
206 if(boardArray[i] instanceof Ownable) {
207 if(((Ownable) boardArray[i]).isOwner(player)) {
208 count++;
209 }
210 }
211 }
212
213 String[] available = new String[count];
214
215 count = 0;
216
217 for(int i = 0; i < boardArray.length; i++) {
218 if(boardArray[i] instanceof Ownable) {
219 if(((Ownable) boardArray[i]).isOwner(player)) {
220 if(pawning && !((Ownable) boardArray[i]).isPawned)) {
221 available[count] = ((Ownable) boardArray[i]).getName();
```

```
222 count++;
223 }
224 else if(!pawning && ((Ownable) boardArray[i]).isPawned) {
225 available[count] = ((Ownable) boardArray[i]).getName();
226 count++;
227 }
228 }
229 }
230 }
231
232 return available;
233 }
234
235 /**
236 * pawn Property
237 * @param player - The choice of player
238 * @param choice - The name of the choice of field
239 * @param updater - The Updater
```

```
240 * @param pawning - choice between pawn or unpawn (true = pawn , false = unpawn)
241 */
242 public void pawnProperty(Player player, String choice, Updater updater, boolean
pawning) {
243 Ownable field = null;
244
245 for(int i = 0; i < boardArray.length; i++) {
246 if(boardArray[i] instanceof Ownable) {
247 if(((Ownable) boardArray[i]).getName().hashCode() == choice.hashCode()) {
248 field = (Ownable) boardArray[i];
249 }
250 }
251 }
252
253 int a = (int) Math.round((field.price * 0.5));
254
255 if(pawning) {
256 if(updater.getUserLeftButtonPressed("Er De sikker på at De vil pantsætte " +
field.getName() + ", og modtage " + a + "?", "Ja", "Nej")) {
257 if((field instanceof Street) && (((Street) field).getHouses() > 0)) {
258 updater.showMessage("De skal sælge Deres huse på " + field.getName() +
", før De kan pantsætte grunden.");
259 }
260 else {
261 field.isPawned = true;
262
263 player.alterAccount(a);
264 player.setAssets(-a);
265
266 updater.balance(player.getName(), player.getAccount());
267 updater.showMessage("De har nu pantsat " + field.getName() + " og
modtager ikke længere leje fra andre spillere, for denne grund.");
268 }
269 }
270 }
271 else {
272 if(updater.getUserLeftButtonPressed("Vil De gerne tilbagekøbe " +
field.getName() + " for " + (int) (Math.round(a * 1.1)) + "?", "Ja", "Nej")) {
273 field.isPawned = false;
274
275 player.alterAccount((int) -(Math.round(a * 1.1)));
276 player.setAssets(a);
277
278 updater.balance(player.getName(), player.getAccount());
Page 5
Board.java
279 updater.showMessage("De tilbagekøbt " + field.getName() + " for " + (int)
(Math.round(a * 1.1)) + ", og modtager nu igen leje for grunden fra andre spillere.");
280 }
281 }
282 }
283
284 /**
285 * buy Property
286 * @param player - The choice of player
287 * @param choice - The names of the choice of group of streets
288 * @param updater - The Updater
289 */
290 public void buyProperty(Player player, String choice, Updater updater) {
```

```
291 Street[] fields = null;
292 fields = new Street[choice.split(", ").length];
293
294 int count = 0;
295 int[] fieldNumbers = new int[3];
296
297 for(String a : choice.split(", ")) {
298 for(int i = 0; i < boardArray.length; i++) {
299 if(boardArray[i] instanceof Street) {
300 if(((Street) boardArray[i]).getName().hashCode() == a.hashCode()) {
301 fields[count] = (Street) boardArray[i];
302 fieldNumbers[count] = i + 1;
303
304 count++;
305 }
306 }
307 }
308 }
309
310 @SuppressWarnings("unused")
311 PropertyControl pControl = new PropertyControl(player, fields, fieldNumbers,
updater, true);
312 pControl = null;
313 }
314
315 /**
316 * contains Same Type
317 * @param array - The array of the field types
318 * @param v - the field type you would like to test
319 * @return true if param:v exist in param:array, else false
320 * </p>
321 * tests if the chosen field type is in the chosen array.
322 */
323 private boolean containsSameType(int[] array, int v ) {
324 for (int e : array)
325 if(e == v)
326 return true;
327
328 return false;
329 }
330
331 /**
332 * get PropertyGrounds
333 * @param player - The choice of player
334 * @return The Streets which HAS houses on them for the chosen player
335 */
336 public String[] getPropertyGrounds(Player player) {
337 int count = 0;
338
```

```
339 for(int i = 0; i < boardArray.length; i++) {
340 if(boardArray[i] instanceof Street) {
341 if(((Street) boardArray[i]).isOwner(player) && ((Street)
boardArray[i]).getHouses() > 0) {
342 count++;
343 }
344 }
345 }
```

```
346
347 Street[] available = new Street[count];
348 int[] availableTypes = new int[9];
349
350 count = 0;
351
352 for(int i = 0; i < boardArray.length; i++) {
353 if(boardArray[i] instanceof Street) {
354 if(((Street) boardArray[i]).isOwner(player) && ((Street)
boardArray[i]).getHouses() > 0) {
355 available[count] = (Street) boardArray[i];
356 count++;
357 }
358 }
359 }
360
361 count = 0;
362
363 for(int i = 0; i < available.length; i++) {
364 for(int j = 0; j < available.length; j++) {
365 if(i != j) {
366 if(available[i].getName() == "Rødovrevej" || available[i].getName() ==
"Rådhuspladsen") {
367 if(available[i].getcType() == available[j].getcType()) {
368 availableTypes[count] = available[i].getcType();
369 count++;
370 }
371 }
372 else {
373 for(int k = 0; k < available.length; k++) {
374 if(i != k && j != k) {
375 if((available[i].getcType() == available[j].getcType()) &&
(available[i].getcType() == available[k].getcType())) {
376 if(!(containsSameType(availableTypes,
available[i].getcType()))) {
377 availableTypes[count] = available[i].getcType();
378 count++;
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387
388 String[] availableGrounds = new String[count];
389
390 for(int i = 0; i < availableGrounds.length; i++)
391 availableGrounds[i] = "";
392
393 count = 0;
394 int counter = 0;
395
```

```
396 for(int a : availableTypes) {
397 for(Field b : boardArray) {
398 if(b instanceof Street) {
```

```
399 if(a == ((Street) b).getcType()) {
400 if(((Street) b).getName() == "Rødovrevej" || ((Street) b).getName()
== "Hvidovrevej" || ((Street) b).getName() == "Rådhuspladsen" || ((Street) b).getName()
==
"Frederiksberggade") {
401 if(counter == 1) {
402 availableGrounds[count] += ((Street) b).getName();
403 }
404 else {
405 availableGrounds[count] += ((Street) b).getName() + ", ";
406 counter++;
407 }
408 }
409 else {
410 if(counter == 2) {
411 availableGrounds[count] += ((Street) b).getName();
412 }
413 else {
414 availableGrounds[count] += ((Street) b).getName() + ", ";
415 counter++;
416 }
417 }
418 }
419 }
420 }
421
422 counter = 0;
423 count++;
424 }
425
426 return availableGrounds;
427 }
428 /**
429 * sell Property
430 * @param player - The choice of player
431 * @param choice - The coice of Ownables which the chosen player would like to sell
432 * @param updater - The Updater
433 */
434 public void sellProperty(Player player, String choice, Updater updater) {
435 Street[] fields = null;
436 fields = new Street[choice.split(", ").length];
437
438 int count = 0;
439 int[] fieldNumbers = new int[3];
440
441 for(String a : choice.split(", ")) {
442 for(int i = 0; i < boardArray.length; i++) {
443 if(boardArray[i] instanceof Street) {
444 if(((Street) boardArray[i]).getName().hashCode() == a.hashCode()) {
445 fields[count] = (Street) boardArray[i];
446 fieldNumbers[count] = i + 1;
447
448 count++;
449 }
450 }
451 }
452 }
453
454 // Stik mod alle designprincipper - viderudvikling nødvendig.
```

```
455 PropertyControl pControl = new PropertyControl(player, fields, fieldNumbers,
```

Board.java
```
updater, false);
456 pControl = null;
457 }
458 }
459
```


Brewery.java
```
1 package game;
2
3 public class Brewery  extends Ownable {
4 private int rent;
5
6 /**
7 * Brewery Constructor
8 * @param Name - The name of the Brewery
9 * @param Price - The Price of the Brewery
10 */
11 public Brewery(String name, int price, int rent){
12 super(name, price);
13 this.rent = rent;
14 }
15
16 /**
17 * get Rent
18 * @return returns rent based on the amount of owners breweries and players roll
19 * </p>
20 * except if brewery is pawned, than returns -1
21 */
22 public int getRent() {
23 if(this.isPawned)
24 return -1;
25 else
26 return (this.rent * owner.getBreweries() * multiplier);
27 }
28 }
29
```


Chance.java
```
1 package game;
2
3 public class Chance  extends ChanceDeck {
4 private String name;
5
6 /**
7 * Chance Constructor
8 * @param Name - The name of the Field
9 */
10 public Chance(String name){
11 super(name);
12 }
13
14 /**
15 * get Name
16 * @return The name of chance field
```

```
17 */
18 public String getName() {
19 return this.name;
20 }
21 }
```


ChanceCard.java

```
1 package game;
2
3 public class ChanceCard {
4 private String description;
5 private String type;
6 private String action;
7
8 /**
9 * ChanceCard Constructor
10 * @param description - description string on the constructed ChanceCard
11 * @param type - type of event (loose money, gain money or move player)
12 * @param action - amount of money to loose/gain or moves to make
13 */
14 public ChanceCard(String description, String type, String action) {
15 this.description = description;
16 this.type = type;
17 this.action = action;
18 }
19
20 /**
21 * get Description
22 * @return Description for ChanceCard
23 */
24 public String getDescription() {
25 return description;
26 }
27
28 /**
29 * set Description
30 * @param description - Change description of ChanceCard to the chosen description
31 */
32 public void setDescription(String description) {
33 this.description = description;
34 }
35
36 /**
37 * get Type
38 * @return Type of the ChanceCard
39 */
40 public String getType() {
41 return type;
42 }
43
44 /**
45 * set Type
46 * @param type - set type of the ChanceCard
47 */
48 public void setType(String type) {
49 this.type = type;
50 }
51
```

```
52 /**
53 * get Action
54 * @return Action of the ChanceCard
55 */
56 public String getAction() {
57 return action;
58 }
59
60 /**
61 * set Action
62 * @param action - set action of the ChanceCard
Page 1
ChanceCard.java
63 */
64 public void setAction(String action) {
65 this.action = action;
66 }
67 }
68
Page 2

ChanceDeck.java
1 package game;
2
3 import java.io.BufferedReader;
8
9 public class ChanceDeck extends Field {
10 private ChanceCard[] deck = new ChanceCard[34];
11 private int cardCount = 0;
12 private int pickCount = 0;
13 private String name;
14
15 // FINALS
16 private final File file = new File("materials/chance.txt");
17
18 /**
19 * Chance Deck Constructor
20 * @param name - The name of The ChanceDeck (unused)
21 */
22 public ChanceDeck(String name) {
23 super(name);
24
25 createCards();
26 shuffleDeck(deck);
27 }
28
29 /**
30 * land On Field
31 * @param player - The choice of player
32 * @param updater - The Updater
33 */
34 public void landOnField(Player player, Updater updater) {
35 updater.showMessage("Træk et \"prøv lykken\"-kort");
36 updater.showMessage(deck[pickCount].getDescription());
37
38 switch(Integer.parseInt(deck[pickCount].getType())) {
39 case 0:
40 // Matadorlegat
41 break;
```

```
42 case 1:
43 // Modtag penge
44 player.alterAccount(Integer.parseInt(deck[pickCount].getAction()));
45 break;
46 case 2:
47 // Modtag penge, af hver spiller
48 // Skal lige..... umiddelbart ret svaer at lave, uden at bryde alle
designprincipper
49 break;
50 case 3:
51 // Betal penge
52 player.alterAccount(Integer.parseInt(deck[pickCount].getAction()));
53 break;
54 case 4:
55 // Ryk brik
56 int newPosition = player.getPosition() +
Integer.parseInt(deck[pickCount].getAction());
57 player.setPosition(newPosition);
58
59 // TODO - Virker IKKE!
60 // Hvis "ryk tre felter tilbage" traekkes, og man staar paa felt 2, fikser vi
lige manuelt position
61 if(newPosition == -1) {
62 newPosition = 40;
63 player.setPosition(newPosition - 1);
Page 1
ChanceDeck.java
64 }
65
66 updater.position(newPosition, player.getName());
67
68 break;
69 case 5:
70 // Ryk direkte
71 // Naar man traekker disse kort, og man skal rykke til et felt, og det felt er
ejet, har vi et problem
72 // - selv hvis det IKKE er ejet, kan det ikke tilbydes til spilleren...
73 // Vi har nemlig fra ChanceDeck ingen kendskab til hverken de andre fields
eller til andre spillere,
74 // og vi kan ikke kalde landOnField her fra
75 // - samme problem som med "Modtag penge, af hver spiller", hvor de andre
spillere ikke er bekendte
76
77
78 if(Integer.parseInt(deck[pickCount].getAction()) == 11) {
79 // Ryk i faengsel - kan i det mindste laves... ^_^
80 player.setJailed(true);
81 player.setPosition(11);
82 }
83 updater.position(Integer.parseInt(deck[pickCount].getAction()),
player.getName());
84
85 break;
86 case 6:
87 // Ryk direkte og betal
88 // Igen... ingen kendskab...
89 break;
90 case 7:
91 // Ryk til rederi
```

```
92 // Ingen kendskab.
93 break;
94 case 8:
95 // Betal pr. hus/hotel
96 String[] str = deck[pickCount].getAction().split("xx");
97 int houses = Integer.parseInt(str[0]);
98 int hotels = Integer.parseInt(str[1]);
99
100 player.alterAccount(-(player.getHouses() * houses));
101 player.alterAccount(-(player.getHotels() * hotels));
102 break;
103 case 9:
104 // Bail
105 player.setBailoutcards(1);
106 break;
107 default:
108 break;
109 }
110
111 setPickCount();
112 }
113
114 /**
115 * create Cards
116 * </p>
117 * creates every single card (ChanceCard)
118 */
119 private void createCards() {
120 String str = readLines();
121
```



```
122 for(String field : str.split("\\|\\|")) {
123 String[] attributes = field.split(";;");
124
125 String a = attributes[0].split("::")[1];
126 String b = attributes[1].split("::")[1];
127 String c = attributes[2].split("::")[1];
128 String d = attributes[3].split("::")[1];
129
130 if(Integer.parseInt(d) > 1) {
131 for(int i = 0; i < 2; i++) {
132 deck[cardCount] = new ChanceCard(a, b, c);
133
134 cardCount++;
135 }
136 }
137 else {
138 deck[cardCount] = new ChanceCard(a, b, c);
139
140 cardCount++;
141 }
142 }
143 }
144
145 /**
146 * shuffle Deck
147 * @param deck - The coice of ChanceCard array
148 * </p>
```

```
149 * suffle the ChanceCard array
150 */
151 public void shuffleDeck(ChanceCard[] deck) {
152 Random rnd = new Random();
153
154 for (int i = deck.length - 1; i > 0; i--) {
155 int index = rnd.nextInt(i + 1);
156
157 // Simple swap
158 ChanceCard a = deck[index];
159 deck[index] = deck[i];
160 deck[i] = a;
161 }
162 }
163
164 /**
165 * get Desc
166 * @param i - The choice of ChanceCard (from array)
167 * @return Description of the chosen ChanceCard
168 */
169 public String getDesc(int i) {
170 return deck[i].getDescription();
171 }
172
173 /**
174 * get Type
175 * @param i - The choice of ChanceCard (from array)
176 * @return Type of the chosen ChanceCard
177 */
178 public String getType(int i) {
179 return deck[i].getType();
180 }
181
182 /**
183 * get Action
```

```
184 * @param i - The choice of ChanceCard (from array)
185 * @return Action of the chosen ChanceCard
186 */
187 public String getAction(int i) {
188 return deck[i].getAction();
189 }
190
191 /**
192 * set Pick Count
193 * adds 1 to pickCount.
194 * </p>
195 * Shuffles deck and reset pickCount if there is no more cards left
196 */
197 public void setPickCount() {
198 this.pickCount++;
199
200 if(this.cardCount <= this.pickCount) {
201 shuffleDeck(this.deck);
202 this.pickCount = 0;
203 }
204 }
205
```

```
206 /**
207 * read Lines
208 * @return String with content of all ChanceCards from .txt file
209 * </p>
210 * Used to create cards
211 */
212 private String readLines() {
213 String content = "";
214
215 try {
216 BufferedReader in = new BufferedReader(new FileReader(file));
217
218 String line = null;
219 while ((line = in.readLine()) != null) {
220 if(line.trim().indexOf("#") == 0)
221 continue;
222 content += line.trim();
223 }
224
225 in.close();
226 }
227 catch (IOException e) {
228 e.printStackTrace();
229 }
230
231 return content;
232 }
233
234 /**
235 * get Name
236 * @return Name of ChanceDeck (unused)
237 */
238 public String getName() {
239 return this.name;
240 }
241 }
242
```


Dice.java

```
1 package game;
2
3 public class Dice {
4 private int[] faceValue;
5 private int quantity, faceSum, min, max;
6 private boolean isPair;
7
8 /**
9 * Dice Constructor
10 * @param min - The minimum facevalue of dice
11 * @param max - The maximum facevalue of dice
12 * @param quantity - The amount of dices
13 */
14 public Dice(int min, int max, int quantity) {
15 this.quantity = quantity;
16 this.min = min;
17 this.max = max;
18 }
19 /**
```

```
20 * "Roll" the dices. <p>
21 * Giving each dice a randomized facevalue
22 */
23 public void throwDice() {
24 faceValue = new int[quantity];
25
26 for(int i = 0; i < quantity; i++)
27 faceValue[i] = (int) (Math.random() * max + min);
28 }
29 /**
30 * setMin
31 * @param min - minimum facevalue for dices
32 */
33 public void setMin(int min) {
34 this.min = min;
35 }
36 /**
37 * setMax
38 * @param max - maximum facevalue for dices
39 */
40 public void setMax(int max) {
41 this.max = max;
42 }
43 /**
44 * setQuan
45 * @param quantity - the amount of dices
46 */
47 public void setQuan(int quantity) {
48 this.quantity = quantity;
49 }
50 /**
51 * getQuan
52 * @return the amount of dices
53 */
54 public int getQuan() {
55 return quantity;
56 }
57 /**
58 * getValue
59 * @param i - The number of a dice (Dice (i+1))
60 * @return The facevalue of the choosen dice, if dice doesn't exist returns 0.
61 */
62 public int getValue(int i) {
Page 1
Dice.java
63 if(i > quantity || i < 0)
64 return 0;
65 else
66 return faceValue[i];
67 }
68 /**
69 * getSum
70 * @return The sum of the facevalue on all dices
71 */
72 public int getSum() {
73 faceSum = 0;
74
75 for(int i = 0; i < quantity; i++)
76 faceSum += faceValue[i];
```

```
77
78 return faceSum;
79 }
80 /**
81 * isPair <p>
82 * Checks for equality of all the dices
83 * @return true if dices are equal, else false
84 */
85 public boolean isPair() {
86 isPair = true;
87
88 for(int i = 0; i < (quantity - 1) && isPair; i++)
89 if(faceValue[i] != faceValue[i + 1])
90 isPair = false;
91
92 return isPair;
93 }
94 }
95
```


Field.java

```
1 package game;
2
3 abstract public class Field {
4 protected String name;
5
6 /**
7 * Field Constructor
8 * @param name - The name of the field
9 */
10 public Field (String name) {
11 super();
12 this.name = name;
13 }
14
15 /**
16 * Land On Field
17 * @param Player - The player who lands on the field
18 */
19 public abstract void landOnField(Player player, Updater updater);
20
21 /**
22 * getName
23 * @param Returns the name of the field
24 */
25 public String getName() {
26 return name;
27 }
28 }
29
```


Fleet.java

```
1 package game;
2
3 public class Fleet extends Ownable {
4 private int[] rent = {500, 1000, 2000, 4000};
5
```

```java
/**
 * Fleet Constructor
 * @param name - The name of the Fleet
 * @param price - The price of the Fleet
 */
public Fleet(String name, int price){
super(name, price);
}

/**
 * get Rent
 * @return returns rent based on the amount of owners fleets
 * </p>
 * except if fleets is pawned, than returns -1
 */
public int getRent() {
if(this.isPawned)
return -1;
else
return rent[owner.getFleet() - 1];
}
}

```


GameController.java

```java
package game;

import java.awt.Color;

public class GameController {
private Player[] player;
private Updater updater = new Updater();
private Turn turn;
private Board board = new Board();
private Dice roll = new Dice(1, 6, 2);
private boolean secondTurn = false;
private int multiplePair = 0;

// FINALS
private final int startCash = 30000;
private final String path = "materials/fields.txt";

/**
 * GameController Constructor
 * </p>
 * When created runs the game process
 */
public GameController() {
// Create board with proper names and descriptions
updater.create(path);

// First receive amount of players by input, then create an array with the amount
of players and their names
createPlayers(countPlayers());

// Keep playing the game 'till someone is victorious
do {
// Check if current player hasn't already lost
```

```
34 if((turn.getIndex(turn.getCurrent()) == 0) &&
(player[turn.getCurrent()].getJailed() == false)) {
35 String str =
updater.getUserButtonPressed(player[turn.getCurrent()].getName() + ", det er Deres
tur.\n",
"1. Slå med terning", "2. Byg hus/hotel", "3. Sælg hus/hotel", "4. Pantsæt grund");
36
37 switch(getChoice(str)) {
38 case 1:
39 // Do a new roll with dice
40 roll.throwDice();
41
42 // Assign position-values
43 int currentPosition = player[turn.getCurrent()].getPosition();
44 int newPosition = currentPosition + roll.getSum();
45
46 // Draw the roll
47 updater.setDice(roll.getValue(0), roll.getValue(1));
48
49 // Move the piece smoothly
50 updater.movePiece(player[turn.getCurrent()], newPosition,
currentPosition);
51
52 player[turn.getCurrent()].setRollSum(roll.getSum());
53
54 // Get an extra turn if dices are pair
55 if(roll.isPair()){
56 secondTurn = true;
57 multiplePair++;
58 if(multiplePair >= 3){
Page 1
GameController.java
59 player[turn.getCurrent()].setJailed(true);
60 player[turn.getCurrent()].setPosition(11);
61 updater.position(player[turn.getCurrent()].getPosition(),
player[turn.getCurrent()].getName());
62 updater.showMessage("De slog to ens tre gange i streg, og ryger
direkte i fængsel.");
63 break;
64 }
65 }
66
67 // Make the mechanics of the field start
68 if(multiplePair != 3)
69 fieldTricker(player[turn.getCurrent()]);
70 // If Ownable and didn't buy it, run an Auction
71 try{
72 if(board.getField(player[turn.getCurrent()].getPosition()-1).getCre
ateAuction()){
73 int auctionField = (player[turn.getCurrent()].getPosition()-1);
74 Ownable f = board.getField(auctionField);
75 Auction auction = new Auction(updater,
player[turn.getCurrent()], player, f);
76 auction.runAction();
77 auction = null;
78 }
79 }
80 catch(Exception VisitJail) {
81 // TODO System.err.println(VisitJail);
```

```
82 }
83 break;
84 case 2:
85 str = updater.getUserButtonPressed("Hvad vil De foretage dem?", "1. Byg
huse/hoteller", "2. Vend tilbage til spilmenu");
86
87 switch(getChoice(str)) {
88 case 1:
89 board.buyProperty(player[turn.getCurrent()],
updater.getUserSelection("Hvilken grund vil De købe hus/hotel til?",
board.getAvailableGrounds(player[turn.getCurrent()])), updater);
90 break;
91 default:
92 break;
93 }
94
95 secondTurn = true;
96 break;
97 case 3:
98 str = updater.getUserButtonPressed("Hvad vil De foretage dem?", "1.
Sælg huse/hoteller", "2. Vend tilbage til spilmenu");
99
100 switch(getChoice(str)) {
101 case 1:
102 board.sellProperty(player[turn.getCurrent()],
updater.getUserSelection("Hvilken grund vil De sælge fra?",
board.getPropertyGrounds(player[turn.getCurrent()])), updater);
103 break;
104 default:
105 break;
106 }
107
108 secondTurn = true;
109 break;
110 case 4:
```

```
111 str = updater.getUserButtonPressed("Hvad vil De foretage dem?", "1.
Pantsæt grunde", "2. Tilbagekøb pantsat grund", "3. Vend tilbage til spilmenu");
112
113 switch(getChoice(str)) {
114 case 1:
115 board.pawnProperty(player[turn.getCurrent()],
updater.getUserSelection("Hvilken grund vil De pantsætte?",
board.getAvailablePawns(player[turn.getCurrent()], true)), updater, true);
116 break;
117 case 2:
118 board.pawnProperty(player[turn.getCurrent()],
updater.getUserSelection("Hvilken grund vil De gerne tilbagekøbe?",
board.getAvailablePawns(player[turn.getCurrent()], false)), updater, false);
119 break;
120 default:
121 break;
122 }
123
124 secondTurn = true;
125 break;
126 default:
127 // Do a new roll with dice
```

```
128 roll.throwDice();
129
130 // Assign position-values
131 int currentPositionDefault = player[turn.getCurrent()].getPosition();
132 int newPositionDefault = currentPositionDefault + roll.getSum();
133
134 // Draw the roll
135 updater.setDice(roll.getValue(0), roll.getValue(1));
136
137 // Move the piece smoothly
138 updater.movePiece(player[turn.getCurrent()], newPositionDefault,
currentPositionDefault);
139
140 // Make the mechanics of the field start
141 fieldTricker(player[turn.getCurrent()]);
142 break;
143
144 }
145 }
146
147 else if(player[turn.getCurrent()].getJailed() == true) {
148 if(player[turn.getCurrent()].getBailoutcards() > 0) {
149 if(updater.getUserLeftButtonPressed("Hvordan vil De komme ud af
fængslet", "Slå med terningen", "Bruge et løsladelseskort")) {
150 //Roll dice to bail out
151 roll.throwDice();
152
153 updater.setDice(roll.getValue(0), roll.getValue(1));
154
155 for(int i = 0; i < 2 ; i++) {
156 roll.throwDice();
157 updater.setDice(roll.getValue(0), roll.getValue(1));
158
159 if(roll.isPair()) {
160 updater.showMessage("De har slået et par! \nDe kommer nu ud
af fængslet");
161 player[turn.getCurrent()].setJailed(false);
162 updater.movePiece(player[turn.getCurrent()],
(player[turn.getCurrent()].getPosition() + roll.getSum()),
player[turn.getCurrent()].getPosition());
Page 3
GameController.java
163 secondTurn = true;
164 i = 3;
165 }
166 else {
167 if (i < 2)
168 updater.showMessage("De slog ikke et par, men har " +
(2-i) + "forsøg tilbage");
169 else
170 updater.showMessage("De slog stadig ingen par og skal
forsat sidde i fængsel til det er Deres tur igen");
171 }
172 }
173 }
174 else {
175 //Use bail out card
176 player[turn.getCurrent()].setBailoutcards(-1);
177 player[turn.getCurrent()].setJailed(false);
```

```
178 secondTurn = true;
179 updater.showMessage("De har benyttet dem af deres benådnings kort
og kan nu trave frit rundt igen");
180 }
181 }
182 else {
183 if(updater.getUserLeftButtonPressed("Hvordan vil De komme ud af
fængslet", "Slå med terningen", "Betal 1000,-")) {
184 //Roll dice to bail out
185 roll.throwDice();
186 updater.setDice(roll.getValue(0), roll.getValue(1));
187
188 for(int i = 0; i<=2 ; i++) {
189 roll.throwDice();
190 updater.setDice(roll.getValue(0), roll.getValue(1));
191
192 if(roll.isPair()) {
193 updater.showMessage("De har slået et par! \nDe kommer nu ud
af fængslet");
194 player[turn.getCurrent()].setJailed(false);
195 updater.movePiece(player[turn.getCurrent()],
(player[turn.getCurrent()].getPosition() + roll.getSum()),
player[turn.getCurrent()].getPosition());
196 secondTurn = true;
197 i = 3;
198 }
199 else {
200 if(i < 2)
201 updater.showMessage("De slog ikke et par, men har " +
(2-i) + "forsøg tilbage");
202 else
203 updater.showMessage("De slog stadig ingen par og skal
forsat sidde i fængsel til det er Deres tur igen");
204 }
205 }
206 }
207 else{
208 //Pay 1000 to bail out
209 updater.showMessage("De har betalt dem ud af fængslet");
210 player[turn.getCurrent()].setJailed(false);
211 player[turn.getCurrent()].alterAccount(-1000);
212 secondTurn = true;
213 }
214 }
215 } // else if line 102
```

```
216
217 // Next player's turn
218 if(!secondTurn){
219 turn.nextTurn();
220 multiplePair = 0;
221 }
222 else
223 secondTurn = false;
224 } while(turn.noWinner());
225
226 // End GUI-session, when game is done.
227 updater.close();
```

```
228 }
229
230 /**
231 * get Choice
232 * @param str - String with "x. msg" x being a number for choice
233 * @return int value for choice ( x )
234 */
235 private int getChoice(String str) {
236 return Integer.parseInt(str.split("\\. ")[0]);
237 }
238
239 /**
240 * create Players
241 * @param n - The number of players chosen for the game
242 * Create an array with n length, and add players to GUI with the names provided via
user input
243 */
244 private void createPlayers(int n) {
245 String name;
246 Color[] colorSet = { Color.RED, Color.BLUE, Color.BLACK, Color.GREEN,
Color.YELLOW,
Color.CYAN };
247
248 player = new Player[n];
249
250 for (int i = 0; i < n; i++) {
251 name = updater.getUserString("Indtast spiller " + (i + 1) + "'s navn");
252
253 updater.addPlayer(name, startCash, colorSet[i]);
254 updater.showMessage(name + " spiller nu med.");
255
256 player[i] = new Player(name, startCash);
257 turn.setIndex(i, 0);
258 }
259 }
260
261 /**
262 * count Players
263 * @return amount of players chosen to play the game
264 * </p>
265 * Count amount of players by user input
266 */
267 private int countPlayers() {
268 int i = updater.getUserInteger("Vælg antal spillere (mindst 2, maks 6).", 2, 6);
269
270 this.turn = new Turn(i);
271
272 return i;
273 }
274
275 /**
```



```
276 * field Tricker
277 * @param player - The choice of player
278 * </p>
279 * For trickering the field mechanics for a specific field for the chosen player
280 */
281 public void fieldTricker(Player player) {
```

```
282 // Which field has the player landed on (minus 1, since we're dealing with an array
from 0-39)
283 board.landOnField((player.getPosition() - 1), player, updater);
284
285 // If the player landed on a field, which he couldn't afford landing on
286 // then reset his owned fields
287 // TODO - correctly losing
288 // need to make sure that the player's assets is accounted for and sold (ownable
fields are pawned before losing)
289 // if these make it possible for him to stay in the game
290 if (player.getStatus() < 0) {
291 board.resetField(player, updater);
292 turn.setIndex(-1, turn.getCurrent());
293 }
294 }
295 }
296
```


GoToJail.java

```
1 package game;
2
3 public class GoToJail extends Field {
4
5 /**
6 * GoToJail Constructor
7 * @param Name - The name of the Field
8 */
9 public GoToJail(String name){
10 super(name);
11 }
12
13 /**
14 * land On Field
15 * @param player - The choice of player
16 * @param updayer - The Updater
17 * </p>
18 * Jails the player and sets player position to 11
19 */
20 public void landOnField(Player player, Updater updater){
21 player.setJailed(true);
22 player.setPosition(11);
23 updater.position(player.getPosition() ,player.getName());
24 }
25 }
26
```


Main.java

```
1 package game;
2
3 public class Main {
4
5 @SuppressWarnings("unused")
6 public static void main(String[] args) {
7 GameController g = new GameController();
8 }
9 }
```

Ownable.java

```java
package game;

abstract public class Ownable extends Field {
protected Player owner;
protected int price;
protected int multiplier;
protected int pawn;
protected Updater updater;
protected boolean isPawned = false;
protected boolean createAuction;

/**
* Ownable Constructor
* @param Name - The name of the Field
* @param Price - The price of the field
*/
public Ownable(String name, int price){
super(name);
this.price = price;
}

/**
* land On Field
* @param player - The choice of player
* @param updater - The Updater
* </p>
* Runs functions for buying if no owners, else makes transfer to owner from player
*/
public void landOnField(Player player, Updater updater) {
this.updater = updater;

if (owner == null) {
if (updater.getUserLeftButtonPressed("Vil De købe " + name + " for " + price +
"?", "Ja", "Nej")) {
if (player.getAccount() >= price) {
owner = player;
owner.alterAccount(-price);
owner.setAssets((int)(price * 0.5));

if(this instanceof Brewery) {
owner.setBrewery();
}
if(this instanceof Fleet) {
owner.setFleet();
}

updater.showMessage(player.getName() + " har købt og ejer nu " + name);
updater.setOwner(player.getPosition(), player.getName());
} else {
updater.showMessage("De har ikke nok penge til at købe denne grund.");
}
}
else {
this.createAuction = true;
}
} else if (!isOwner(player)) {
this.multiplier = player.getRollSum();

```

```
58 if(this.getRent() == -1) {
59 updater.showMessage("Grunden er pantsat, og De betaler derfor ikke leje til
" + owner.getName() + ".");
60 }
Page 1
Ownable.java
61 else {
62 updater.showMessage(player.getName() + " betaler " + this.getRent() + " til
" + owner.getName() + ".");
63 player.payRent(getRent(), owner);
64 }
65 }
66 }
67
68 /**
69 * is Owner?
70 * @param player - The choice of player
71 * @return true if the chosen player is owner, else false.
72 */
73 public boolean isOwner(Player player) {
74 return player == owner;
75 }
76
77 /**
78 * get Pawn
79 * @return int value for pawned or not
80 */
81 public int getPawn() {
82 return this.pawn;
83 }
84
85 /**
86 * reset Owner
87 * </p>
88 * sets owner of Ownable to null
89 */
90 public void resetOwner() {
91 owner = null;
92 }
93
94 /**
95 * get Price
96 * @return buy price on Ownable
97 */
98 public int getPrice(){
99 return this.price;
100 }
101
102 /**
103 * get Rent
104 * @return rent on Ownable
105 */
106 abstract int getRent();
107
108 /**
109 * set Owner
110 * @param player - The choice of player
111 * </p>
112 * sets owner to the chosen player
```

```
113 */
114 public void setOwner(Player player){
115 owner = player;
116 }
117
118 /**
119 * get CreateAuction
120 * @return true if a Auction needs to be made, else false
121 */
```
Page 2

Ownable.java
```
122 public boolean getCreateAuction(){
123 if(createAuction){
124 createAuction = false;
125 return !createAuction;
126 }
127 else return false;
128 }
129 }
130
```
Page 3

Parking.java
```
1 package game;
2
3 public class Parking extends Field {
4
5 /**
6 * Parking Constructor
7 * @param Name - The name of the Field
8 */
9 public Parking(String name){
10 super(name);
11 }
12
13 /**
14 * land On Field
15 * @param player - The choice of player
16 * @param updater - The Updater
17 * </p>
18 * Shows message on GUI that tells you have landed on a parking spot
19 */
20 public void landOnField(Player player, Updater updater){
21 updater.showMessage("De parkerer bilen, og spiser en hjemmepakket rugbrødsmad.");
22 }
23 }
24
```
Page 1

Player.java
```
1 package game;
2
3 public class Player {
4 private String name;
5 private int status;
6 private int position;
7 private int account;
8 private int assets;
9 private int fleets;
```

```
10 private int breweries;
11 private int rollSum;
12 private int bailoutcards;
13 private boolean jailed;
14 private int houses;
15 private int hotels;
16 private Updater updater = new Updater();
17
18 /**
19 * Player Constructor
20 * @param name - The name you would like to give the player
21 * @param account - The amount you would like to set the players account to
22 */
23 public Player(String name, int account) {
24 this.name = name;
25 this.account = account;
26 this.assets = 0;
27 this.position = 1;
28 this.breweries = 0;
29 this.fleets = 0;
30 this.bailoutcards = 0;
31 this.status = 0;
32
33 }
34 /**
35 * get Name
36 * @return The name of the player
37 */
38 public String getName() {
39 return this.name;
40 }
41 /**
42 * get Status
43 * @return The activity status of the player (0 = active , -1 = inactive)
44 */
45 public int getStatus() {
46 return this.status;
47 }
48 /**
49 * get Position
50 * @return The position on the board of the player
51 */
52 public int getPosition() {
53 return position;
54 }
55 /**
56 * set Position
57 * @param position - Set the position on the board of the player
58 */
59 public void setPosition(int position) {
60 this.position = position;
61 }
62 /**
```

```
63 * set Status
64 * @param status - Set the activity status of the player (0 = active , -1 =
inactive)
65 */
```

```
66 public void setStatus(int status) {
67 this.status = status;
68 }
69 /**
70 * alter Account
71 * @param amount - The amount you would like to add to the players account, if
possible.
72 * </p>
73 * If impossible account set to 0, and player activity status is set to inactive.
74 */
75 public void alterAccount(int amount) {
76 if ((this.account + amount) >= 0) {
77
78 this.account += amount;
79 }
80
81 else {
82 this.account = 0;
83 setStatus(-1);
84
85 updater.removeCar(this.name);
86 updater.showMessage(this.name + ", De er gået fallit - alle Deres grunde er
solgt til banken.");
87 }
88
89 updater.balance(this.name, this.account);
90 }
91 /**
92 * get Account
93 * @return The amount on the players account
94 */
95 public int getAccount() {
96 return this.account;
97 }
98
99 /**
100 * set Assets
101 * @param amount - Adds the amount to the players current assets amount
102 */
103 public void setAssets(int amount) {
104 assets += amount;
105 }
106
107 /**
108 * get Assets
109 * @return The players assets + his account
110 */
111 public int getAssets() {
112 return (assets + account);
113 }
114
115 /**
116 * pay Rent
117 * @param rent - The amount that is being transfered
118 * @param leaser - The player which the given player has to pay
119 * @return The activity status of the given player
120 */
121 public int payRent(int rent, Player leaser) {
122 int amount;
```

```
123
124 if (rent > this.account) {
125 amount = this.account;
126 setStatus(-1);
127 }
128 else {
129 amount = rent;
130 alterAccount(-amount);
131 }
132
133 leaser.alterAccount(amount);
134
135 return this.status;
136 }
137
138 /**
139 * set Fleet
140 * </p>
141 * add 1 to the given players fleet amount
142 */
143 public void setFleet() {
144 this.fleets += 1;
145 }
146
147 /**
148 * get Fleet
149 * @return The amount of fleets the given player has
150 */
151 public int getFleet() {
152 return this.fleets;
153 }
154
155 /**
156 * set Brewery
157 * </p>
158 * add 1 to the given players brewery amount
159 */
160 public void setBrewery() {
161 this.breweries += 1;
162 }
163
164 /**
165 * get Breweries
166 * @return The amount of breweries the given player has
167 */
168 public int getBreweries() {
169 return this.breweries;
170 }
171
172 /**
173 * get Rollsum
174 * @return The sum of the latest roll the given player has made
175 */
176 public int getRollSum() {
177 return this.rollSum;
178 }
179
```

```java
/**
* set Rollsum
* @param rollSum - The given players latest roll
*/
public void setRollSum(int rollSum) {
```

```java
this.rollSum = rollSum;
}
/**
* set Jailed
* @param jailed - is the player jailed? True = yes , false = no
*/
public void setJailed(boolean jailed){
this.jailed = jailed;
}

/**
* get Jailed
* @return The statement of if player is in jail. True = yes, false = no
*/
public boolean getJailed(){
return jailed;
}

/**
* set Bail get out cards
* @param i - the increase of bailoutcards.
*/
public void setBailoutcards(int i){
this.bailoutcards += i;
}

/**
* get Bail get out cards
* @return the amount of bailoutcards the given player has
*/
public int getBailoutcards(){
return this.bailoutcards;
}

/**
* get Houses
* @return The amount of houses the player own across the board
*/
public int getHouses() {
return houses;
}

/**
* set Houses
* @param houses Set the total amount of houses the player owns
*/
public void setHouses(int houses) {
this.houses = houses;
}

/**
* get Hotels
```

```
237 * @return The amount of hotels the player own across the board
238 */
239 public int getHotels() {
240 return hotels;
241 }
242
243 /**
244 * set Hotels
245 * @param hotels Set the total amount of hotels the player owns
246 */
```

Player.java
```
247 public void setHotels(int hotels) {
248 this.hotels = hotels;
249 }
250 }
251
```


PropertyControl.java
```
1 package game;
2
3 public class PropertyControl {
4 private boolean notDone = true;
5 private Player player;
6 private Street[] fields;
7 private int[] fieldNumbers;
8 private Updater updater;
9 private boolean buying;
10
11 /**
12 * Property Control
13 * @param player - The choice of player
14 * @param fields - Street array of the group of streets
15 * @param fieldNumbers - int array for Streets array (from original array)
16 * @param updater - The Updater
17 * @param buying - The choice of buying (true) or selling (false)
18 */
19 public PropertyControl(Player player, Street[] fields, int[] fieldNumbers, Updater
updater, boolean buying) {
20 do {
21 if(buying)
22 this.buildEvenly(player, fields, fieldNumbers, updater);
23 else
24 this.sellEvenly(player, fields, fieldNumbers, updater);
25 } while(notDone);
26 }
27
28 /**
29 * run Builder
30 * </p>
31 * runs sellEvenly or buildEvenly until done
32 */
33 public void runBuilder() {
34 do {
35 if(buying)
36 this.buildEvenly(player, fields, fieldNumbers, updater);
37 else
38 this.sellEvenly(player, fields, fieldNumbers, updater);
```

```
39 } while(notDone);
40 }
41
42 /**
43 * sell Evenly
44 * @param player - The choice of player
45 * @param fields - Street array of the group of streets
46 * @param fieldNumbers - int array for Streets array (from original array)
47 * @param updater - The Updater
48 * </p>
49 * runs sell progress for houses and hotels and does it evenly
50 */
51 private void sellEvenly(Player player, Street[] fields, int[] fieldNumbers, Updater
updater) {
52 String[] options = null;
53
54 if(fields.length == 3) {
55 options = new String[4];
56
57 for(int i = 0; i < fields.length; i++)
58 options[i] = (i + 1) + ". " + fields[i].getName();
59
60 options[3] = "4. Færdig med salg af huse.";
Page 1
PropertyControl.java
61 }
62
63 else if(fields.length == 2) {
64 options = new String[3];
65
66 for(int i = 0; i < fields.length; i++)
67 options[i] = (i + 1) + ". " + fields[i].getName();
68
69 options[2] = "4. Færdig med salg af huse.";
70 }
71
72 String choice = updater.getUserButtonPressed("Salgspris pr. hus: " +
(fields[0].getHousePrice() / 2) + ". Vælg en grund De vil sælge fra.", options);
73
74 switch(getChoice(choice)) {
75 case 1:
76 if(fields[0].getHouses() > 0) {
77 if(canSellHouse(0, fields)) {
78 if(fields[0].getHouses() == 5) {
79 fields[0].removeHotel(fieldNumbers[0]);
80 }
81 else {
82 fields[0].setHouses(fieldNumbers[0], (fields[0].getHouses() - 1));
83 }
84
85 player.alterAccount((fields[0].getHousePrice() / 2));
86 player.setAssets(-((int) Math.floor((0.5 *
fields[0].getHousePrice()))));
87 }
88 else
89 updater.showMessage("De skal sælge jævnt fra grundene.");
90 }
91 else
92 updater.showMessage("De kan ikke sælge flere huse fra "+
```

```
fields[0].getName() + ", da der ikke er flere huse på grunden.");
93 break;
94 case 2:
95 if(fields[1].getHouses() > 0) {
96 if(canSellHouse(1, fields)) {
97 if(fields[1].getHouses() == 5) {
98 fields[1].removeHotel(fieldNumbers[1]);
99 }
100 else {
101 fields[1].setHouses(fieldNumbers[1], (fields[1].getHouses() - 1));
102 }
103
104 player.alterAccount((fields[1].getHousePrice() / 2));
105 player.setAssets(-((int) Math.floor((0.5 *
fields[1].getHousePrice()))));
106 }
107 else
108 updater.showMessage("De skal sælge jævnt fra grundene.");
109 }
110 else
111 updater.showMessage("De kan ikke sælge flere huse fra "+
fields[1].getName() + ", da der ikke er flere huse på grunden.");
112 break;
113 case 3:
114 if(fields[2].getHouses() > 0) {
115 if(canSellHouse(2, fields)) {
116 if(fields[2].getHouses() == 5) {
117 fields[2].removeHotel(fieldNumbers[2]);
Page 2
PropertyControl.java
118 }
119 else {
120 fields[2].setHouses(fieldNumbers[2], (fields[2].getHouses() - 1));
121 }
122
123 player.alterAccount((fields[2].getHousePrice() / 2));
124 player.setAssets(-((int) Math.floor((0.5 *
fields[2].getHousePrice()))));
125 }
126 else
127 updater.showMessage("De skal sælge jævnt fra grundene.");
128 }
129 else
130 updater.showMessage("De kan ikke sælge flere huse fra "+
fields[2].getName() + ", da der ikke er flere huse på grunden.");
131 break;
132 case 4:
133 notDone = false;
134 break;
135 }
136 }
137
138 /**
139 * build Evenly
140 * @param player - The choice of player
141 * @param fields - Street array of the group of streets
142 * @param fieldNumbers - int array for Streets array (from original array)
143 * @param updater - The Updater
144 * </p>
```

```
145 * runs buy progress for houses and hotels and does it evenly
146 */
147 private void buildEvenly(Player player, Street[] fields, int[] fieldNumbers, Updater
updater) {
148 String[] options = null;
149
150 if(fields.length == 3) {
151 options = new String[4];
152
153 for(int i = 0; i < fields.length; i++)
154 options[i] = (i + 1) + ". " + fields[i].getName();
155
156 options[3] = "4. Færdig med køb af huse.";
157 }
158
159 else if(fields.length == 2) {
160 options = new String[3];
161
162 for(int i = 0; i < fields.length; i++)
163 options[i] = (i + 1) + ". " + fields[i].getName();
164
165 options[2] = "4. Færdig med køb af huse.";
166 }
167
168 String choice = updater.getUserButtonPressed("Pris pr. hus: " +
fields[0].getHousePrice() + ". Vælg en grund De vil købe hus på.", options);
169
170 switch(getChoice(choice)) {
171 case 1:
172 if(player.getAccount() - fields[0].getHousePrice() >= 0)
173 if(fields[0].getHouses() <= 4) {
174 if(canBuyHouse(0, fields)) {
175 if(fields[0].getHouses() == 4) {
Page 3
PropertyControl.java
176 fields[0].setHotel(fieldNumbers[0]);
177 }
178 else {
179 fields[0].setHouses(fieldNumbers[0], (fields[0].getHouses() +
1));
180 }
181
182 player.alterAccount(-fields[0].getHousePrice());
183 player.setAssets((int) Math.floor((0.5 *
fields[0].getHousePrice())));
184 }
185 else
186 updater.showMessage("De skal bygge jævnt på grundene.");
187 }
188 else
189 updater.showMessage("De kan ikke købe flere huse til "+
fields[0].getName() + ", da der allerede er bygget et hotel.");
190 else
191 updater.showMessage("De har ikke råd til at købe huset.");
192 break;
193 case 2:
194 if(player.getAccount() - fields[1].getHousePrice() >= 0)
195 if(fields[1].getHouses() <= 4) {
196 if(canBuyHouse(1, fields)) {
```

```
197 if(fields[1].getHouses() == 4) {
198 fields[1].setHotel(fieldNumbers[1]);
199 }
200 else {
201 fields[1].setHouses(fieldNumbers[1], (fields[1].getHouses() +
1));
202 }
203
204 player.alterAccount(-fields[1].getHousePrice());
205 player.setAssets((int) Math.floor((0.5 *
fields[1].getHousePrice())));
206 }
207 else
208 updater.showMessage("De skal bygge jævnt på grundene.");
209 }
210 else
211 updater.showMessage("De kan ikke købe flere huse til "+
fields[1].getName() + ", da der allerede er bygget et hotel.");
212 else
213 updater.showMessage("De har ikke råd til at købe huset.");
214 break;
215 case 3:
216 if(player.getAccount() - fields[2].getHousePrice() >= 0)
217 if(fields[2].getHouses() <= 4) {
218 if(canBuyHouse(2, fields)) {
219 if(fields[2].getHouses() == 4) {
220 fields[2].setHotel(fieldNumbers[2]);
221 }
222 else {
223 fields[2].setHouses(fieldNumbers[2], (fields[2].getHouses() +
1));
224 }
225
226 player.alterAccount(-fields[2].getHousePrice());
227 player.setAssets((int) Math.round((0.5 *
fields[2].getHousePrice())));
228 }
229 else
Page 4
PropertyControl.java
230 updater.showMessage("De skal bygge jævnt på grundene.");
231 }
232 else
233 updater.showMessage("De kan ikke købe flere huse til "+
fields[2].getName() + ", da der allerede er bygget et hotel.");
234 else
235 updater.showMessage("De har ikke råd til at købe huset.");
236 break;
237 case 4:
238 notDone = false;
239 break;
240 }
241 }
242
243 /**
244 * highest Count
245 * @param fields - Street array of the group of streets
246 * @param buying - The choice of buying (true) or selling (false)
247 * @return the higest amount of houses on one street from street array
```

```
248 */
249 private int highestCount(Street[] fields, boolean buying) {
250 int highestCount = 0;
251 boolean same = true;
252
253 for(Street a : fields)
254 if(a.getHouses() > highestCount)
255 highestCount = a.getHouses();
256
257 for(Street a : fields)
258 if(a.getHouses() != highestCount)
259 same = false;
260
261 if(same && buying)
262 highestCount = 0;
263
264 return highestCount;
265 }
266
267 /**
268 * can Buy House
269 * @param i - the field number for the choice of street
270 * @param fields - Street array of the group of streets
271 * @return true if able to buy house on chosen street, else false
272 */
273 private boolean canBuyHouse(int i, Street[] fields) {
274 if(highestCount(fields, true) == 0)
275 return true;
276 else if(fields[i].getHouses() < highestCount(fields, true))
277 return true;
278
279 return false;
280 }
281
282 /**
283 * get Choice
284 * @param str - the choice string
285 * @return the number indicator for the string
286 */
287 private int getChoice(String str) {
288 return Integer.parseInt(str.split("\\.")[0]);
289 }
290
```

```
291 /**
292 * can Sell House
293 * @param i - the field number for the choice of street
294 * @param fields - Street array of the group of streets
295 * @return true if able to sell house on chosen street, else false
296 */
297 private boolean canSellHouse(int i, Street[] fields) {
298 if(fields[i].getHouses() == highestCount(fields, false))
299 return true;
300
301 return false;
302 }
303 }
304
```

Start.java

```java
package game;

public class Start extends Field {

/**
* Start Constructor
* @param Name - The name of the Field
*/
public Start(String name){
super(name);
}

/**
* land On Field
* @param player - The choice of player
* @param updater - The Updater
* </p>
* show message on GUI that you have landed on start and will recieve 4000. transfer 4000 to the chosen players account
*/
public void landOnField(Player player, Updater updater){
updater.showMessage("De lander på START, og modtager 4.000.");
player.alterAccount(4000);
updater.balance(player.getName(), player.getAccount());
}
}

```


Street.java

```java
package game;

public class Street extends Ownable {
private int cType;
private int houses;
private int housePrice;
private int[] rent = new int[6];

/**
* Street Constructor
* @param Name - The name of the street
* @param Price - The price for buying the street
* @param CType - The group of streets
* @param Rent0 - The rent with 0 houses on the street
* @param Rent1 - The rent with 1 house on the street
* @param Rent2 - The rent with 2 houses on the street
* @param Rent3 - The rent with 3 houses on the street
* @param Rent4 - The rent with 4 houses on the street
* @param Rent5 - The rent with 1 hotel on the street
*/

public Street (String name, int price, int cType, int rent0, int rent1, int rent2, int rent3, int rent4, int rent5, int housePrice) {
super(name, price);
this.housePrice = housePrice;
this.cType = cType;
```

```
26 this.houses = 0;
27 this.rent[0] = rent0;
28 this.rent[1] = rent1;
29 this.rent[2] = rent2;
30 this.rent[3] = rent3;
31 this.rent[4] = rent4;
32 this.rent[5] = rent5;
33 }
34
35 /**
36 *
37 * @return The amount of houses.
38 */
39 public int getHouses() {
40 return houses;
41 }
42
43 /**
44 * set Houses
45 * @param field - The field number for which we want to buy houses for.
46 * @param houses - Sets the amount of houses for the field.
47 */
48 public void setHouses(int field, int houses) {
49 this.houses = houses;
50
51 updater.setHouses(field, houses);
52 }
53
54 /**
55 * set Hotel
56 * @param field - The field number for which we want to buy hotel for
57 */
58 public void setHotel(int field) {
59 this.houses = 5;
60
61 updater.setHotel(field, true);
Page 1
Street.java
62 }
63
64 /**
65 * remove Hotel
66 * @param field - The field number for which we want to remove hotel from
67 * @param houses - The new amount of houses for the field
68 */
69 public void removeHotel(int field) {
70 this.houses = 4;
71
72 updater.setHotel(field, false);
73 updater.setHouses(field, houses);
74 }
75
76 /**
77 *
78 * @return The amount of rent for the field, based on how many houses the field has.
79 */
80 public int getRent() {
81 if(this.isPawned)
82 return -1;
```

```
83 else
84 return rent[this.getHouses()];
85 }
86
87 /**
88 *
89 * @return The cluster for which the field is a part of
90 */
91 public int getcType() {
92 return this.cType;
93 }
94
95 /**
96 * get House Price
97 * @return the price per house on Street
98 */
99 public int getHousePrice() {
100 return this.housePrice;
101 }
102 }
103
```


Tax.java

```
1 package game;
2
3 import boundaryToMatador.GUI;
4
5 public class Tax  extends Field {
6 private int taxType;
7
8 /**
9 * Tax Constructor
10 * @param Name - The name of the Field
11 * @param taxType - taxType = 0 : 10% or 4.000kr else taxType != 0 : 2.000kr
12 */
13 public Tax(String Name,int taxType){
14 super(Name);
15 this.taxType = taxType;
16
17 }
18 /**
19 * LandOnField
20 * <p>
21 * @param player - The player who lands on the field
22 */
23 public void landOnField(Player player, Updater updater){
24 if (taxType == 0){
25 if(updater.getUserLeftButtonPressed(player.getName() + ", de skal betale
indkomstskat: betal 10% af deres aktiver eller 4000kr,-", "10%", "4000kr,-")){
26 //10%
27 updater.showMessage("De betaler " + (int)(player.getAssets() * 0.1) + "kr,-
i indkomstskat");
28 player.alterAccount(-((int)(player.getAssets() * 0.1)));
29 updater.balance(player.getName(), player.getAccount());
30 }
31 else{
32 //4k
33 GUI.showMessage("De betaler 4000kr,- i indkomstskat");
```

```
34 player.alterAccount(-4000);
35 updater.balance(player.getName(), player.getAccount());
36 }
37
38 }
39 else {
40 //2k
41 GUI.showMessage("De skal betale Ekstra ordinær statsskat: Betal 2000kr,-");
42 player.alterAccount(-2000);
43 updater.balance(player.getName(),player.getAccount());
44 }
45 }
46 }
47
```


Turn.java

```
1 package game;
2
3 public class Turn {
4 private int[] index;
5 private int current;
6
7 /**
8 * Turn Constructor
9 * @param index - the amount of players
10 */
11 public Turn(int index) {
12 this.index = new int[index];
13 this.current = 0;
14
15 for(int i = 0; i < this.index.length; i++)
16 this.index[i] = 1;
17 }
18
19 /**
20 * set Index
21 * @param i - the choice of player (from array)
22 * @param index - set activity
23 */
24 public void setIndex(int i, int index) {
25 this.index[i] = index;
26 }
27
28 /**
29 * set Current
30 * @param current - current turn
31 */
32 public void setCurrent(int current) {
33 this.current = current;
34 }
35
36 /**
37 * next Turn
38 * </p>
39 * Makes the current turn to the next active player
40 */
41 public void nextTurn() {
42 current++;
```

```
43
44 if (current >= index.length)
45 current = 0;
46
47 if(index[current] == -1) {
48 boolean notActive = true;
49
50 do {
51 if(this.current == -1)
52 current++;
53 else
54 notActive = false;
55
56 if (current >= index.length)
57 current = 0;
58
59 } while(notActive);
60 }
61 }
62
```


Turn.java

```
63 /**
64 * get Index
65 * @param i - the choice of player (from array)
66 * @return the activity of the chosen player
67 */
68 public int getIndex(int i) {
69 return this.index[i];
70 }
71
72 /**
73 * get Current
74 * @return which players turn it is (for array)
75 */
76 public int getCurrent() {
77 return this.current;
78 }
79
80 /**
81 * no Winner
82 * @return false if there is a winner, else true
83 */
84 public boolean noWinner() {
85 int status = index.length;
86
87 for (int i = 0; i < index.length; i++)
88 status += index[i];
89
90 if (status == 1)
91 return false;
92 else
93 return true;
94 }
95 }
96
```


Updater.java

```java
package game;

import java.awt.Color;

public class Updater {

/**
 * Updater Constructor
 * </p>
 * Constructs updater
 */
public Updater(){

}

/**
 * balance All
 * @param players - The player array
 * </p>
 * Updates the balance on the GUI for all players
 */
public void balanceAll(Player[] players){

for(int i = 0 ; i < players.length ; i++ ){
GUI.setBalance(players[i].getName(), players[i].getAccount());
}
}

/**
 * position All
 * @param players - The player array
 * </p>
 * Updates the position on the GUI for all players
 */
public void positionAll(Player[] players){
for(int i = 0 ; i < players.length ; i++ ) {
GUI.removeAllCars(players[i].getName());
GUI.setCar(players[i].getPosition(), players[i].getName());
}
}

/**
 * balance
 * @param player - The player which you would like to update
 * </p>
 * Updates the balance on the GUI for the chosen player
 */
public void balance(String name, int account){
GUI.setBalance(name, account);
}

/**
 * position
 * @param player - The player which you would like to update
 * Updates the position on the GUI for the chosen player
 */
public void position(int fieldNumber, String name){
GUI.removeAllCars(name);
GUI.setCar(fieldNumber, name);
```

```
62 }
63
64 /**
Page 1
Updater.java
65 * move Piece Tester
66 * @param player - The player which you would like to move
67 * @param newPosition - The position you wish the player to move to
68 * @param currentPosition - The position the player is already on
69 */
70 public void movePiece(Player player, int newPosition, int currentPosition) {
71 if(newPosition > 40) {
72 newPosition -= 40;
73
74 // First move the piece the last steps before hitting START
75 for (int f = 1; f <= (40 - currentPosition); f++) {
76 GUI.removeAllCars(player.getName());
77 GUI.setCar((currentPosition + f), player.getName());
78 sleep(100); // When testing, set to 1, or get bored
79 }
80
81 // Now move the piece the fields after START
82 for (int f = 1; f <= newPosition; f++) {
83 if(f == 2){
84 player.alterAccount(4000);
85 GUI.setBalance(player.getName(), player.getAccount());
86 }
87 GUI.removeAllCars(player.getName());
88 GUI.setCar(f, player.getName());
89 sleep(100); // When testing, set to 1, or get bored
90 }
91 } else {
92 // Move the piece the require fields
93 for (int f = (currentPosition + 1); f <= newPosition; f++) {
94 GUI.removeAllCars(player.getName());
95 GUI.setCar(f, player.getName());
96 sleep(100); // When testing, set to 1, or get bored
97 }
98 }
99
100 player.setPosition(newPosition);
101 }
102
103 /**
104 * sleep
105 * @param n - The amount of time to wait in milliseconds
106 */
107 private void sleep(int n) {
108 long start, end;
109
110 start = System.currentTimeMillis();
111
112 do {
113 end = System.currentTimeMillis();
114 } while ((end - start) < (n));
115 }
116
117
//********************************************************************************
```

```
**********************************
// Everything after this point is direct GUI commands
//*************************************************************************************
**********************************

/**
* create
* @param path - the choice of string
* </p>
```

```
* calls GUI.create(path)
*/
public void create(String path){
GUI.create(path);
}

/**
* add Player
* @param name - Name of player
* @param balance - account of player
* @param color - color on the players car
* </p>
* adds player to GUI
*/
public void addPlayer(String name, int balance, Color color){
GUI.addPlayer(name, balance, color);
}

/**
* set Owner
* @param fieldNumber - The choice of field (number on board)
* @param name - The choice of player (string name)
* </p>
* sets Owner on GUI for chosen field to chosen player
*/
public void setOwner(int fieldNumber, String name){
GUI.setOwner(fieldNumber, name);
}

/**
* remove Owner
* @param fieldNumber - The choice of field (number on board)
* </p>
* removes owner on GUI for chosen field
*/
public void removeOwner(int fieldNumber){
GUI.removeOwner(fieldNumber);
}

/**
* remove Car
* @param name - The choice of player (string name)
*</p>
*removes all cars from GUI for the chosen player
*/
public void removeCar(String name){
GUI.removeAllCars(name);
```

```
172 }
173
174 /**
175 * set Houses
176 * @param fieldNumber - The choice of field (number on board)
177 * @param houseCount - amount of houses
178 * </p>
179 * sets the amount of houses on the chosen field
180 */
181 public void setHouses(int fieldNumber, int houseCount){
182 GUI.setHouses(fieldNumber, houseCount);
183 }
184
185 /**
186 * set Hotel
```

```
187 * @param fieldNumber - The choice of field (number on board)
188 * @param hasHotel - has hotel (true) or not (false)
189 * </p>
190 * sets or remove hotel on the chosen field
191 */
192 public void setHotel(int fieldNumber, Boolean hasHotel){
193 GUI.setHotel(fieldNumber, hasHotel);
194 }
195
196 /**
197 * Displays a message to the user and awaits the button pressed response
198 * @param msg - The message that promts the user
199 * @param buttons - A number of strings that should be printed on the buttons the
user
can press
200 * @return The string from the button that the user pressed
201 */
202 public String getUserButtonPressed(String msg, String... buttons){
203 return GUI.getUserButtonPressed(msg, buttons);
204 }
205
206 /**
207 * Shows two dice on the board. The dice will have the specified values, but the
placement is random.
208 * @param faceValue1 - int [1:6]
209 * @param faceValue2 - int [1:6]
210 * (If a parameter is out of bounds nothing will happen!) Uses random rotation.
211 */
212 public void setDice(int faceValue1, int faceValue2){
213 GUI.setDice(faceValue1, faceValue2);
214 }
215
216 /**
217 * Displays a message to the user and awaits the integer response. Only values
between
min and max are allowed.
218 * @param msg - The message that promts the user
219 * @param min - The minimum value the user is allowed to enter
220 * @param max - The maximum value the user is allowed to enter
221 * @return The integer that the user selected.
222 */
223 public int getUserInteger(String msg, int min, int max){
```

```
224 return GUI.getUserInteger(msg, min, max);
225 }
226
227 /**
228 * Displays a message to the user and awaits the boolean response.
229 * @param msg - The message that promts the user
230 * @param trueButton - The text that should appear on the left button
231 * @param falseButton - The text that should appear on the right button
232 * @return True if the left button is pressed by the user. False otherwise.
233 */
234 public boolean getUserLeftButtonPressed(String msg, String trueButton, String
falseButton){
235 return GUI.getUserLeftButtonPressed(msg, trueButton, falseButton);
236 }
237
238 /**
239 * Displays a message to the user.
240 * @param msg - The message to print
241 */
242 public void showMessage(String msg){
243 GUI.showMessage(msg);
244 }
```

Updater.java
```
245
246 /**
247 * Displays a message to the user and awaits the response.
248 * @param msg - The message that promts the user
249 * @return The string that the user has entered
250 */
251 public String getUserString(String msg){
252 return GUI.getUserString(msg);
253 }
254
255 /**
256 * close
257 * </p>
258 * closes GUI
259 */
260 public void close(){
261 GUI.close();
262 }
263
264 /**
265 * Displays a dropdown to the user and awaits the response.
266 * @param msg - The message which prompts the user
267 * @param options - An array with possible options to choose between from the
dropdown.
268 * @return
269 */
270 public String getUserSelection(String msg, String[] options) {
271 return GUI.getUserSelection(msg, options);
272 }
273 }
274
```


VisitJail.java
```
1 package game;
```

```
2
3 public class VisitJail  extends Field {
4
5 /**
6 * VisitJail Constructor
7 * @param Name - The name of the Field
8 */
9 public VisitJail(String name){
10 super(name);
11 }
12
13 /**
14 * land On Field
15 * @param Player - The choice of player
16 * @param updater - The Updater
17 * </p>
18 * show message in GUI that you have visited the jail
19 */
20 public void landOnField(Player player, Updater updater){
21 updater.showMessage("De er på besøg i fængslet,\n og vinker til de indsatte, mens De
kører forbi.");
22 }
23 }
24
```


Bilag

Chance Kort

Matador chancekort

Modtag penge

Kort:
(De modtager ”Matador-legatet for værdigt trængende” på kr. 40.000. Ved værdigt trængende forstås, at deres formue, dvs. Deres kontante penge + skøder+ bygninger, ikke overstiger 15.000.)

Kort:
(Det er Deres fødselsdag Modtag af hver medspiller kr.200)

Kort:
(Modtag udbytte af deres aktier – 1.000. kr.)

Kort:
(De har vundet i klasselotteriet. Modtag kr. 500.)

Kort:
(Kommunen har eftergivet et kvartals skat. Hæv i banken kr. 3.000.)

Kort:
(De har lagt penge ud til et sammenskudsgilde. Mærkværdigvis betaler alle straks. Modtag fra hver medspiller kr. 500.)

Kort:x2
(De modtager deres aktieudbytte. Modtag kr. 1.000 af banken.)

Kort:
(Grundet dyrtiden har De fået gageforhøjelse Modtag kr. 1.000.)

Kort:
(De havde en række med elleve rigtige i tipning. Modtag kr. 1.000)

Kort:
(De har solgt nogle gamle møbler på auktion modtag kr. 1.000 af banken.)

Kort:
(Deres præmieobligation er udtrukket. De modtager kr. 1.000 af banken.)

Kort:
(De har vundet i Klasselotteriet. Modtag kr. 500.)

Kort:
(De skal holde familiefest og får et tilskud fra hver medspiller på kr. 500)

Kort:
(Værdien af egen avl fra nyttehaven udgør kr. 200, som De modtager af banken.)

Kort:
(Deres præmieobligation er udtrukket de Modtager kr. 1.000 af banken.)

Mist Penge
Kort:
(Betal kr. 3.000 for reparation af Deres vogn.)

Kort:
(Betal kr. 3.000 for reparation af Deres vogn.)

Kort:
(De har modtaget Deres tandlægeregning. Betal 2.000.)

Kort:
(De har fået en parkeringsbøde betal kr. 200 i bøde.)

Kort:
(Betal kr. 200 for levering af 2 kasser øl.)

Kort:
(Betal deres bilforsikring – kr. 1.000)

Kort:
(De har været en tur i udlandet og haft for mange cigaretter med hjem. Betal told kr. 200)

Kort:
(Betal for vognvask og smøring kr.300.)

Kort:
(De har købt 4 nye dæk til Deres vogn. Betal kr. 1.000.)

Kort:
(De har kørt frem for ”Fuldt stop”. Betal kr. 1.000 i bøde.)

Bevæge sig

Kort:x2
(Ryk brikken frem til det nærmeste rederi og betal ejeren to gange den leje, hvis han ellers er berettiget til. Hvis selskabet ikke ejes af nogen, kan De købe det af banken.)

Kort:
(Tag med Mols-linien. Flyt brikken frem, og hvis De passerer ”START” indkassér da kr. 4.000.)

Kort: x2
(Gå i fængsel. Ryk direkte til fængslet. Selv om de passerer ”START”, indkasserer de ikke kr. 4.000.)

Kort:
(Tag med den nærmeste færge. Flyt brikken frem, og hvis de passerer ”START” indkassér da kr. 4.000)

Kort:
(Tag ind på Rådhuspladsen)

Kort:
(Ryk frem til Strandvejen. Hvis de passerer ”START” indkassér da kr. 4.000.)

Kort:
(Ryk frem til Grønningen. Hvis de passerer ”START” indkassér da kr. 4.000.)

Kort:
(Ryk frem til Vimmelskaftet Hvis de passerer ”START” indkassér da kr.4.000.)

Kort:
(Ryk tre felter frem)

Kort:x2
(Ryk frem til ”START”.)

Kort:x2
(Ryk tre felter tilbage)

Kort:
(Ryk frem til Frederiksberg Allé. Hvis de passerer ”START”, indkassér da kr. 4.000)

Betal(Huse)

Kort:
(Ejendomsskatterne er steget. Ekstraudgifterne er: kr. 800 pr. hus, kr. 2.300 pr. hotel)

Kort:
(Oliepriserne er steget, og De skal betale: kr. 500 pr. hus kr 2.000 pr. hotel.)

Speciale
Kort: x2
(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.)