

3-ugers Projektopgave jan 2014

02312-14 Indledende programmering

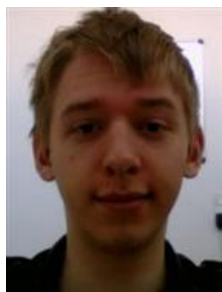
Projektnavn: 51_Final

Gruppe nr: 51.

Afleveringsfrist: mandag den 20/01 2014 Kl. 12:00

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

Denne rapport indeholder 49 sider inkl. denne side.



s123064, Nielsen, Martin _____



s134000, Budtz, Christian _____



s134004, Vørmadal, Rúni Egholm _____

Matador Final							
Time-regnskab	Ver. 2013-09-07						
Dato	Deltager	Design	Impl.	Test	Dok.	Andet	Ialt
6/1/2014	Christian	3	3				6
6/1/2014	Martin	2					2
6/1/2014	Jens-Ulrik	2					2
6/1/2014	Runi	4				2	6
6/1/2014	Mustafa	2					2
7/1/2014	Christian		6	1			7
7/1/2014	Martin	3	1				4
7/1/2014	Jens-Ulrik	2	4				6
7/1/2014	Runi	3				2	5
7/1/2014	Mustafa					1	1
8/1/2014	Christian	1	6	1			8
8/1/2014	Rúni		2		1	2	5
8/1/2014	Martin		6				6
9/1/2014	Martin		4				4
9/1/2014	Christian		4				4
9/1/2014	Rúni		3			1	4
10/1/2014	Rúni		1				1
10/1/2014	Christian		4	1			5
10/1/2014	Martin		5				5
11/1/2014	Rúni		6				6
11/1/2014	Christian		9	2			11
11/1/2014	Martin		7				7
12/1/2014	Christian		5	1			6
13/1/2014	Martin		7	1			8
13/1/2014	Christian		3	1			4
13/1/2014	Rúni		9	2			11
14/1/2014	Martin		8	1.5		1	10.5
14/1/2014	Christian		6	3	1	1	11
14/1/2014	Rúni		9	2			11
15/1/2014	Christian		4	2	1		7
15/1/2014	Rúni		10	2			12
15/1/2014	Martin		4	4			8
16/1/2014	Christian		1	0.5	2		3.5
16/1/2014	Rúni		7	3			10
17/1/2014	Christian		2	4	6		12
17/1/2014	Martin		1		5	2	8
17/1/2014	Rúni	2	7	3			12
18/1/2014	Christian		1	1	3		5
18/1/2014	Martin		1			6	7
18/1/2014	Rúni		3			4	7
19/1/2014	Christian		2	2	4		8
19/1/2014	Martin					7	7
19/1/2014	Rúni		4	1	2		7
20/1/2014	Martin					4	4
20/1/2014	Christian					4	4
20/1/2014	Rúni					4	4
	Sum	24	165	39	25	41	294
Efter person							0
	Christian	4	56	19.5	17	5	101.5
	Runi	9	61	13	3	15	101
	Martin	5	44	6.5	5	20	80.5
	Jens-Ulrik	4	4	0	0	0	8
	Mustafa	2	0	0	0	1	3
	Sum	24	165	39	25	41	294

Indholdsfortegnelse

[Indholdsfortegnelse](#)

[Indledning](#)

[Kravspecificering](#)

[Domænemodel](#)

[Use-cases](#)

[Afklaring og antagelser](#)

[Non funktionelle krav](#)

[Ikke honorerede krav](#)

[Designmodeller](#)

[Design sekvensdiagrammer](#)

[Design Patterns](#)

[BCE pattern](#)

[GRASP pattern](#)

[Singleton Pattern](#)

[Hjælper klasser](#)

[Implementering](#)

[GameController](#)

[PlayerController:](#)

[TradeController](#)

[BoardController](#)

[Board](#)

[Field klasser](#)

[CardController:](#)

[Kort Klasserne:](#)

[Test og kvalitetssikring](#)

[Kvalitet - 'FURPS+'](#)

[Konklusion](#)

[Bilag](#)

[Kildekode](#)

[Sub Use Case Diagrammer](#)

[Testdokumentation](#)

[Scenarie 'Gå fallit'](#)

[Scenarie 'Undgå fallit'](#)

[Scenarie 'Vind Spillet'](#)

[Scenarie 'Køb grund'](#)

[Scenarie køb og salg af huse](#)

[Litteraturliste](#)

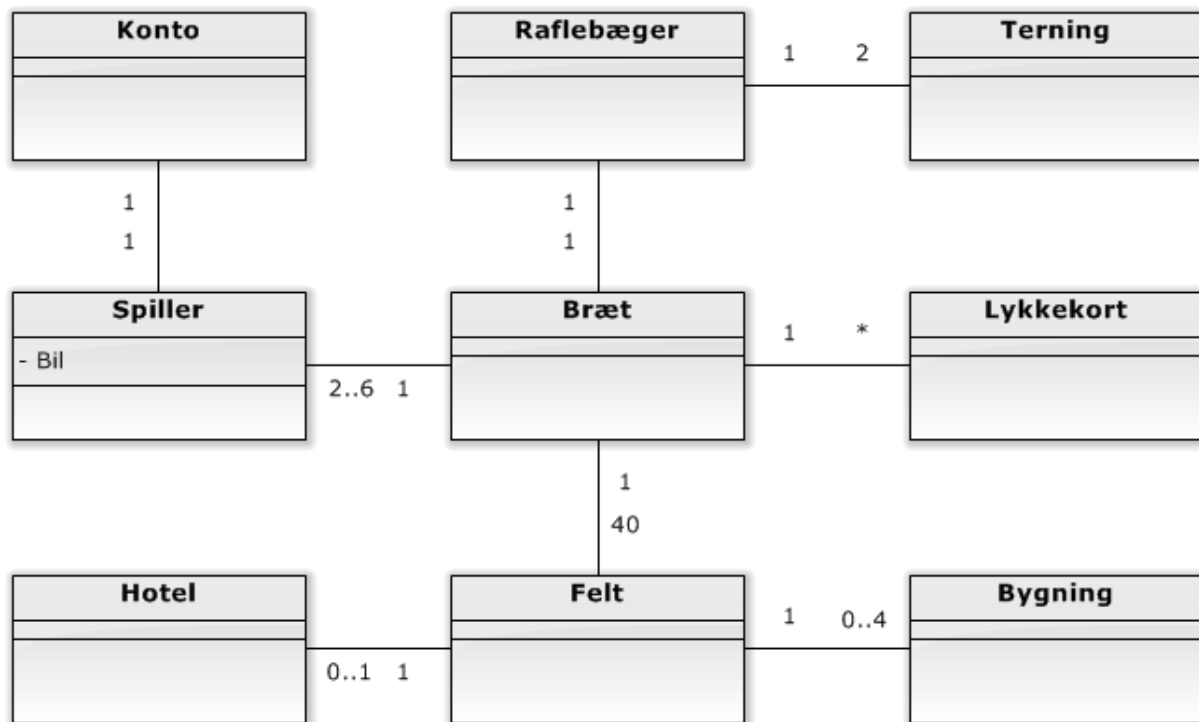
Indledning

Vi har fået til opgave at lave et Matadorspil. For at kunne løse dette problem er det vigtigt for os først at kunne forstå problemet inden vi begynder at arbejde på løsningen. Derfor har vi valgt at starte med at se på et klassisk Matadorspil.

Kravsificering

Domænemodel

For at kunne implementere Matadorspillet har vi analyseret hvad spillet består af. Til dette formål har vi startet med at illustrerer spillet i en domæne model. Derudover afhænger interaktionen mellem aktørerne og vores program af selve opsætningen for et Matadorspil, vil vi derfor kigge på hvordan spillet Matador helt konkret fungerer.



Domænemodel for Matador.

I vores Domænemodel har vi opnået overblik over de fysiske elementer, der udgør Matador. Matadorspillet består dog ikke kun af fysiske objekter, men også af regler. Derfor har vi prøvet at gengive de eksakte regler, som vi har fundet beskrevet i spillet. Selvom vores mål er at have færdigudviklet et komplet matadorspil, hvor spillerne har samme muligheder for at spille spillet ligesom de finder i virkeligheden, så har vi

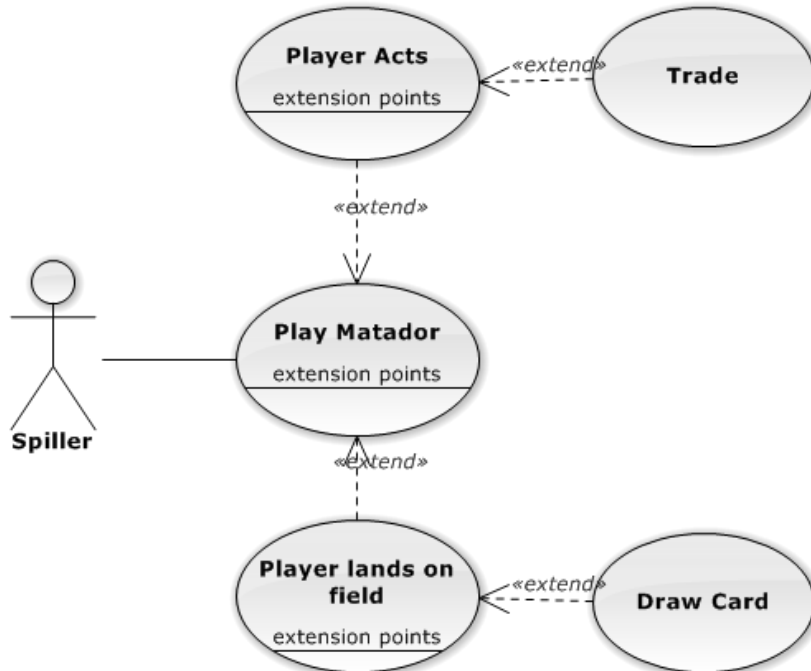
fokuseret på at implementere et matadorspil med vægt på at skabe de ting vi ser som mest nødvendige først. Vi har derfor delt spillet op i use cases, som vi har prioriteret efter hvilke vi synes er vigtige at have med, da vi må være forberedte på, at der kan være nogle, vi ikke når at få implementeret.

Use-cases

I vores Matador spil har vi en aktør - spilleren. Vi har brudt spillet ned i en main use-case "Play Matador", der består af at spillerne efter tur slår med to seks-sidede terninger og rykker deres bil rundt på pladen efter terningslaget. Som precondition er spillet startet, sprog og antal spillere valgt, og spillerne har indtastet deres brugernavne. Spillerne får tildelt 30.000 i startkapital. Spillebræt, kort og skøder er som i den moderne udgave af Matador. Som postcondition findes en vinder og spillet slutter.

Vi har brudt resten af spillet op i mindre use cases, som vi har behandlet separat. Disse use cases har vi behandlet som Extensions af vores main use case.

- 'Player acts' - Handlinger der initieres af spilleren.
- 'Trade' - Sub Use case af Player acts - indbyrdes handel.
- 'Draw Card' - Spilleren trækker prøv lykken kort.
- 'Player Lands on Field' - De handlinger, der sker som følge af at man lander på et felt.



Main Use Case 'Play Matador' Diagram.

Ovennævnte Sub use cases er stadig abstrakte og omhandler ikke specifikke use cases, men repræsenterer de mindste use cases, der giver en meningsfuld kompleksitet og

sammenhæng og dermed mappes til controllers.

Sub use cases er konkretiseret i Use Case diagrammer i bilagene. Kort ridset op kan de beskrives som følger.

‘Player acts’ kan deles og konkretiseres som at flytte bilen ‘Move Car’ og ‘Trade’, hhv. flytte bilens placering på pladen og handel, der igen kan konkretiseres som køb/salg og pantsætning/ophævelse af pantsætning af grunde, køb/salg af huse og indbyrdes handel med grunde (og evt. benådninger).

‘Player lands on field’ kan specificeres yderligere ud fra hvilket felt det drejer sig om - eks. er ‘Land on Street’ en Sub Use Case af Land on ownable. ‘Land on Street’ tillader spilleren at købe feltet (da det er Ownable), hvis det ikke er ejet og tvinger ham til at betale leje, hvis det er ejet af en anden spiller. Har spilleren alle grunde i en farve må han bygge huse på grundene (der skal dog bygges jævnt¹). Brewery og Shipping er ligeledes extensions af Ownable. ‘Land on Chance’ initerer ‘Draw card’ - der igen kan extendes af eks. ‘Go To Jail’. Go To jail er også included i ‘Land on Go to Jail’. Sub Use Casen ‘Auction’, der sker når en spiller vælger ikke at købe en Ownable er ikke anført, ligesom den ikke er implementeret.

Afklaring og antagelser

Det er specificeret i reglerne at bunken kun blandes en gang, hvorefter kortene lægges tilbage i bunden af bunken.

Vi har antaget, at når der skal betales 10% skat af ens formue, tæller skøder for fuld værdi, og ikke pantsætningsværdi. Det samme gælder huse. De tæller for købeværdi, og ikke salgsværdi.

Non funktionelle krav

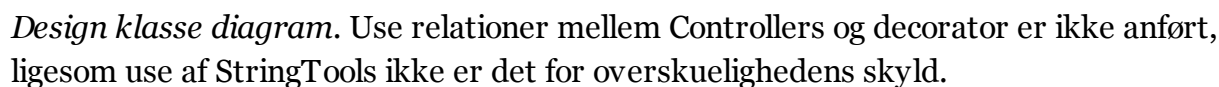
Spillet skal kunne afvikles på computere i dtu’s databar - dvs. på nuværende tidspunkt med java 1.6.

Ikke honorerede krav

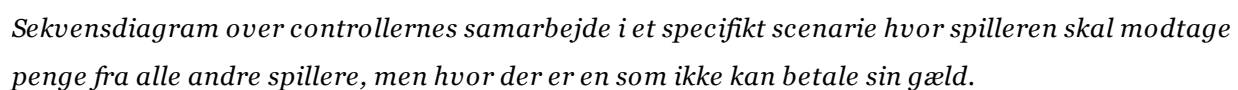
Vi har ikke implementeret matadorlegatet. Ej heller har vi muligheden for at holde auktion. Man kan ikke sælge benådningskortet. I stedet er tiden brugt på at forsøge at opnå reliability i de implementerede funktioner.

¹ Ved jævnt forstås at der ikke må bygges på en grund der allerede har flere huse end de andre i samme gruppe.

Ud fra domænemodel og use case diagrammer har vi modelleret nedenstående klassediagram. De 4 extensions i vores Main Use case er mappet til controllere, de øvrige komponenter i spillet er omsat til entities. De faktiske spillebrikker er dog omsat til en placering hos spillerne og sedlerne er reduceret til en balance på spillerens konto. Account har ikke et modstykke i Domænemodel heller og repræsenterer således 'pure fabrication'. To hjælperklasser - Shuffler og StringTools er også anført.



Nedenfor ses et sekvens diagram som viser kaldene som controllerne laver til hinanden i et specifikt scenarie. Terningerne er kastet og gameControlleren kalder move(..) i playerController som flytter bilen, derefter kalder den LandOnField(..) i BoardController, som overloader sin egen metode. Dette er på grund af scenariet hvor en spiller efter at have trukket et “prøv lykken” kort, skal betale dobbelt.



Fordi feltet er af typen “Prøv lykken” kalder BoardControlleren drawCard(..) i CardControlleren som trækker det første kort i den shuffede bunke. Fordi at kortet er en instans af receiveFromPlayersCard, som betyder at alle spillere skal betale til den aktive, bliver en for loop kørt over alle spillere, som sørger for at de betaler, og hvis en spiller ikke kan betale bliver han bedt om at rejse pengene, dette sker ved at CardControlleren kalder handleInsufficientFunds(..) i PlayerControlleren, som returnerer en true eller false alt efter om spilleren kan rejse pengene eller ej. Fordi spilleren skal betale, og ikke bare mangler penge til et hus eller lignende, bliver spilleren kørt fallit og alle hans grunde bliver overført til den spiller han gik fallit til.

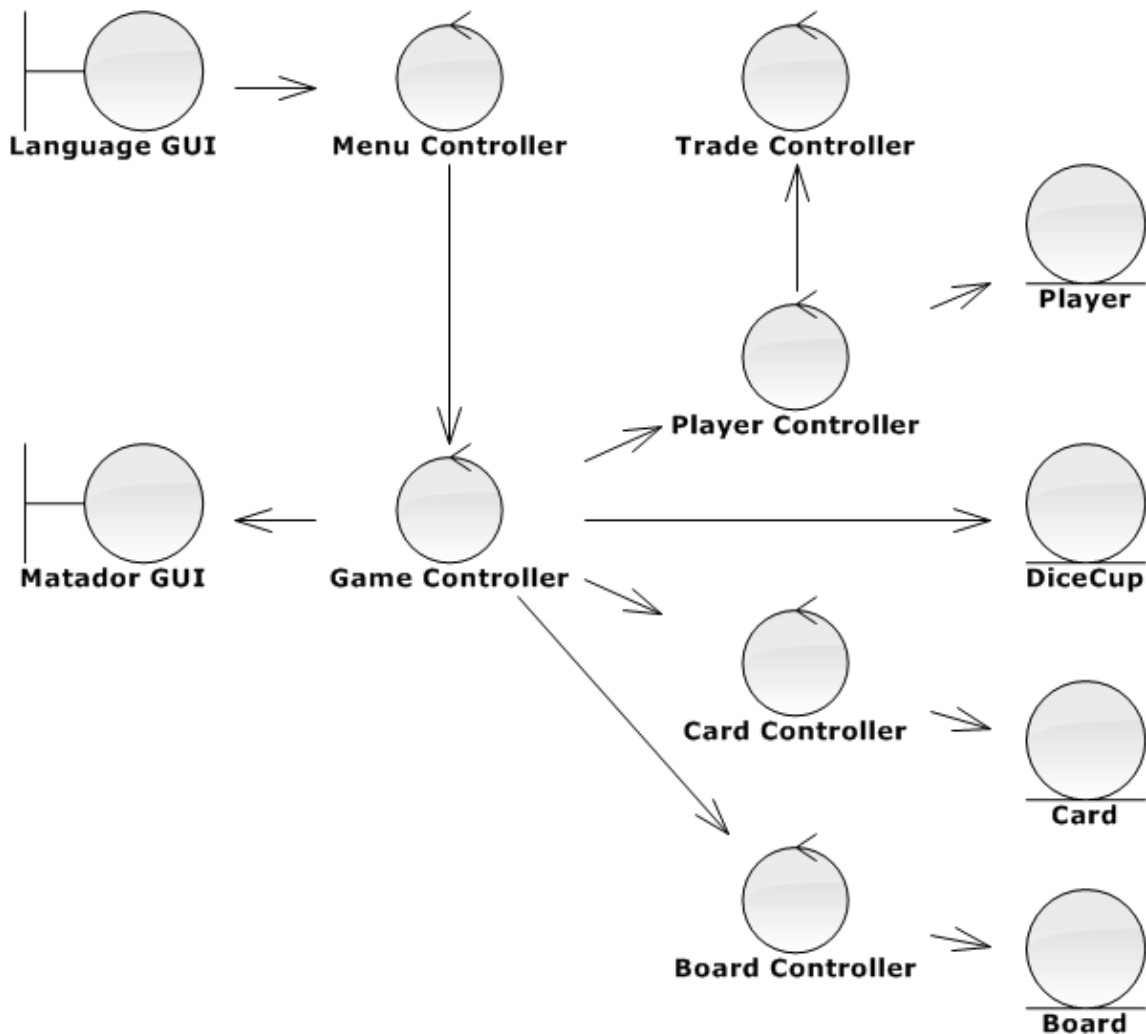
Design Patterns

I det nærværende projekt har vi forsøgt at implementere flere forskellige patterns for at opnå en kodebase med høj reusability, reliability og supportability. Vi har i anden række forsøgt at opnå en god usability for brugeren med et konsistent og forståeligt interface. Vi har ikke i væsentligt omfang koncentreret os om performance, da vores program ikke kan forventes at øges væsentligt i kompleksitet og næppe skal afvikles på platforme hvor ressourcerne ikke er rigelige.

BCE pattern

I den forrige opgave blev vi bedt om at implementere en polymorf metode LandOnField på vores felter. Det gav en kodemæssigt simpel løsning på de meget forskelligartede felters funktion. Set fra designperspektiv, bryder det dog med BCE pattern, idet felterne, som må betragtes som entities også får en controller funktion. I delopgave 3 havde felterne således direkte interaktion med GUI'en. I større projekter er det en risikabel løsning, da ændringer i GUI'ens funktion kan skabe problemer i alle klasser, der tilgår den direkte. I det givne projekt, hvor GUI'en ikke ændres under udviklingen, vil det næppe blive det store problem, men for øvelsens skyld, valgte vi at forsøge at overholde BCE patternet.

En løsning kunne være at udspalte entities fra felterne, således at hver feltklasse får sin egen controller og entity. Det giver dog hurtigt et virvar af klasser med meget lidt kode i. Vi valgte i stedet at beholde data i vores Field klasser og samle de forskellige felters metoder i en BoardController. Vores felter har således ingen interaktion med vores boundary. Vi har således opnået overholdelse af BCE, men på bekostning af lidt mere kompliceret kode. Den lidt specielle GUI's funktioner er forsøgt indkapslet ved indirektion - således at kun decoratoren tilgår funktioner i GUI'en - det gør forhåbentlig skift til en anden gui lettere.



BCE diagram. Bemærk de to forskelligrettede pile fra GUI'erne. Language GUI aktiverer en listener i Menu Controller, Matador-GUI'en bliver (som en TUI) spurgt om et input fra brugeren.

I vores BCE diagram ses opbygningen af vores program i Boundaries, Controllers og Entities. Player Controller, Card Controller og Board Controller interagerer med hyppige use relations, ligesom de interagerer med Matador GUI'en. Trade controller er sub-controller for indbyrdes handel mellem spillerne og får sine parametre fra PlayerControllen og interagerer ligeledes med Matador GUI'en.

GRASP pattern

Vi har forsøgt at arbejde med 'responsibility driven design' og implementeret Controller, Creator, Information Expert, Low Coupling, High Cohesion og Polymorphism.

High Cohesion og *Low Coupling* er forsøgt opnået for at sikre bedre supportability og reusability. Højest Coupling ses fra GameController, der interagerer med alle

subcontrollers, GUI'en og DiceCup. Coupling kunne godt mindskes for GameController, men det ville skabe højere Coupling i andre klasser eller nødvendiggøre flere klasser (eks. i form af controllers) og ville muligvis skabe lavere Cohesion, idet nye klassers ansvarsområde risikerer at blive svært at definere. Vi har vurderet at den nuværende kompleksitet er acceptabel og samtidig sikrer en høj Cohesion idet ansvarsområderne i vores øjne er logisk fordelt.

Information Expert er søgt implementeret ved at de objekter der naturligt kender informationen og anvender den, selv varetager den. Felternes ejerskab er en særlig udfordring, da felterne har behov for at vide, hvem der ejer dem (eks for at betale leje) og spillerne har behov for at vide hvilke felter de ejer, (for at kunne bygge huse og handle indbyrdes). For at undgå at informationen ligger to steder og desuden en dobbelt coupling mellem Fields og Players, hvor felterne kender deres ejer og spillerne kender deres felter, har vi valgt at indføre statiske hjælpermetoder til at tilgå data fra spillebrættet. Således forespørger GameController BoardController, hver gang et ejerskab skal bruges og BoardController genererer så de relevante data. Vi har for at undgå at data på boardet ændres uden en direkte reference til boardController, kun implementeret statiske *getter* metoder.

Creator er søgt overholdt, idet GameController opretter egne subcontrollers, der i anden række opretter de entities de skal bruge - GameController opretter Players, CardController Cards, BoardController Board og så fremdeles.

Controller pattern er implementeret ved at al control vedrørende en given use case er samlet i en controller. Således håndteres det at rykke rundt på brættet af vores GameController. Selve sub use casen 'Land on field' håndteres af BoardControlleren. Sub use casen 'Player acts' varetages af GameController, der igen har en TradeController til at håndtere al handel mellem spillere. Håndtering af 'Draw Chance Card' foregår i CardController.

Polymorphism. En del af polymorfien er fjernet fra felterne, sammenlignet med 3. delprojekt. Som beskrevet ovenfor, er det for at undgå interaktion mellem entities og boundary. Vi genererer dog stadig en decoratorMessage, der sammensættes ved nedarvning. Field genererer beskeden om hvilket felt spilleren er landet på og subklasserne tilføjer feltspecifikke beskeder.

Singleton Pattern

Raflebægeret (DiceCup) bruges både til at slå for at rykke og til at bestemme hvor meget leje en spiller skal betale for at lande på bryggerierne. Da der kun er et raflebæger i spillet, har vi valgt at gøre det til en singleton, der så kan tilgås globalt. Vi undgår dermed at skulle passe referencen hver gang spilleren lander på et felt. Vi bruger en 'eager loading' singleton - idet instantieringen af DiceCup må forventes at ske på få millisekunder. DiceCup singleton'en skaber dog (i større projekter) det problem, at det kan være svært at afgøre hvilke klasser der anvender singleton'en, idet alle klasser kan

tilgå den med `public static getInstance()`². Den først beskrevne metode er sikrere, men tungere kodemæssigt.

Hjælper klasser

Enkelte metoder tilhører ikke logisk en klasse og finder anvendelighed mange steder i koden. Vi har eksempelvis brugt String arrays i stort omfang, da vi er bundet af ikke at måtte bruge avancerede typer som eks. ArrayList. En meget hyppigt anvendt funktion er at lægge to String Arrays sammen eller lægge en String til et array, hvorfor vi har oprettet en hjælperklasse StringTools, med to statiske metoder til netop dette.

En anden ikke klassespecifik metoder er blanding af objekterne i et array - eks til at blande prøv lykken kortene. Det kunne tænkes at man også ville blande skøder, biler eller noget helt andet i en fremtidig udgave af matador, hvorfor vi har lagt metoden i en helper klasse Shuffler.

Implementering

- “Udvalgte komplekse kodedele forklares
 - Forklaringer i tekst
 - Forklaringer i relevante diagrammer”

MenuController

Facade controller, der sørger for at vælge sprog og delegerer ansvaret videre GameController. Har sin egen boundary i form af en JFrame³.

GameController

Håndterer opsætning af spillet og det overordnede flow gennem spillernes tur.

Instantierer subcontrollerne - BoardController og CardController samt dicecup og Decorator. Herefter bruges setupGame() til at vise GUI'en med felter fra boardController og til at bede playerController om at oprette et player[].

Hovedloopet i spillet, runGame() looper indtil der kun er en spiller tilbage. Er spillerne ikke gået fallit får de enten en playerTurn(...) eller en jailTurn(...), hvis de er i spillet.

PlayerTurn(...) håndterer spillets main use case og enkelte ekstra regler, der logisk hører til her. Der slås med terninger og ansvaret delegeres til playerController, der varetager at flytte spilleren og tilhørende subuse cases. Spilleren tilbydes at købe huse og handle og ønsker han det delegeres ansvaret til playerController.

² Singleton pattern. (2014, January 16). In *Wikipedia, The Free Encyclopedia*. Retrieved 17:48, January 17, 2014, from http://en.wikipedia.org/w/index.php?title=Singleton_pattern&oldid=590910949

³How to Make Dialogs - Oracle Documentation. Retrieved January 16 2014, from <http://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html>

TwoOfAKindCheck() kontrollerer om spilleren slår to ens og opdaterer spillerens tæller for to ens. Slår spilleren to ens 3 gange i træk, sættes hans inJail = true.

JailTurn(...) håndterer fængselsreglerne og tilbyder spilleren at købe sig ud, slå efter to ens eller bruge en benådning. Efter 3 forsøg tvinges spilleren til at købe sig ud. Kommer spilleren ud køres afterJailTurn(...), der er en speciel udgave af en normal tur, hvor terningslaget fra fængslet bruges.

PlayerController:

Vores constructor i PlayerControlleren tager en int startMoney og en int totalfields.

Den første metode, vi støder på er move(...)-metoden. Den er simpel, den tager en player, en sum, som spilleren skal flyttes, og en decorator. Spilleren bliver derefter flyttet den pågældende sum frem eller tilbage. Hvis spilleren passerer start, bliver passStart() kaldt, som giver spilleren startpengene. Move har også et lille regnestykke, som sørger for, at spilleren kommer rundt på pladen, og ikke lander uden for boardet.

Derefter kommer moveTo(...) metoden, som flytter en spiller til et bestemt felt. Her har vi brugt overload for at løse problemet med kortet, der beder en om at flytte til nærmeste rederi og betale dobbelt takst. Den ene moveTo holder en rent modifier ekstra, som bliver ganget på taksten.

Den normale moveTo() har også en int til sidst, men den er sat til 1, så der ikke bliver ændret i regnestykket.

Move metoderne delegerer til sidst ansvaret til boardControlleren - boardController.landOnField(...).

Efter moveTo() kommer moveToJail(). Den flytter en spiller i fængsel, og giver ikke spilleren startpenge, hvis han passerer start. Den spiller spillerens fængselsstatus til 1 (som betyder at spilleren er hans første tur i træk i spældet) og sætter twoOfAKindCount lig med 0.

payDebt(...) er en metode til at inddrive gæld. Den kaldes med en debitor, kreditor, decorator, en besked og en gældsstørrelse (se kode nedenfor). Gælden forsøges inddrevet i loop, der kører så længe gælden ikke er betalt. I loopet er et try-catch statement, der forsøger at trække beløbet på spillerens konto. Mislykkes det at inddrive gælden kaster account en InsufficientFundsException (IFE). IFE'en igangsætter et forsøg på at rejse pengene. Har spilleren grunde, der ikke er pantsat, tvinges han til at handle eller pantsætte, har han ikke flere ikke pantsatte grunde, tilbydes han også at give op - "Bankrupt", hvorefter playerController afvikler spilleren via playerController.hostileTakeOver(...) og while loopet brydes hvorefter debtPaid returneres til den kaldende funktion.

```

94  public boolean payDebt(Player debtor, Player creditor, Decorator decorator, String[] msg,
    int debt) {
95      boolean debtPaid = false;
96      while (!debtPaid){
97          decorator.showMessage(msg);
98          try {
99              debtor.getAccount().withdraw(debt);
100             if (creditor != null) {
101                 creditor.getAccount().deposit(debt); //Should be handled in a seperate
statement - future
102             }
103             debtPaid = true;
104         } catch (InsufficientFundsException e) {
105             //insufficient funds to pay player
106             System.out.println("Insufficient funds for transaction");
107             //If player has any unpawned fields, he is forced to pawn them
108             if(BoardController.hasAnyUnPawnedFields(debtor)){
109                 String[] msg1 = new String[]{"YouMustAtLeastPawnAllFields"};
110                 decorator.showMessage(msg1);
111                 handleInsufficientFunds(debtor, debt, decorator);
112             } else {
113                 //Else he can choose between trying to raise money
114                 String[] messageString = new String[] {"TradeOrGoBroke"};
115                 String[] buttons = new String[] {"Trade", "Bankrupt"};
116                 int selection = decorator.getUserButtonPressed(messageString, buttons);
117                 if (selection == 0){
118                     handleInsufficientFunds(debtor, debt, decorator);
119                 } else {
120                     hostileTakeOver(creditor, debtor);
121                     break;
122                 }
123             }
124         } catch (IllegalAmountException e) {
125             System.err.println("IllegalAmount LandOnOwnable");
126             e.printStackTrace();
127             break;
128         }
129     }
130     return debtPaid;
131 }

```

PayDebt(). Se teksten ovenfor.

handleInsufficientFunds(...) er en metode, der bliver kaldt, hvis en spillers konto bliver forsøgt overtrukket. Den giver pågældende spiller mulighed for at rejse de penge han mangler. Metoden bliver kaldt hvis en spiller vil købe huse eller en grund han ikke har råd til. Metoden returnerer en boolean som giver den kaldende funktion mulighed for at lave forskellige aktioner alt efter om spilleren får betalt eller ej. Metoden bliver også brugt i payDebt metoden som er en slags udvidelse, da denne metode bliver kaldt hvis en spiller skal tvinges til at betale, på grund af leje eller "Prøv lykken" kort.

PlayerController.java

```
93     public boolean handleInsufficientFunds(Player player, int amount,
    Decorator decorator) {
94         String[] msg1 = new String[]{"YouMustPawnTrade"};
95         String[] msg2 = new String[]{"TradeOrNot"};
96         String[] opt0 = new String[]{"Trade", "Cancel"};
97         while(player.getAccount().getBalance() < amount){
98             String[] msg0 = new String[]{"NotEnoughMoneyYouNeed"+
    Integer.toString(amount)};
99             decorator.showMessage(StringTools.add(msg0, msg1));
100             if(decorator.getUserButtonPressed(msg2, opt0) == 0){
101                 tradeController.trade(player, decorator, this);
102             }
103             else{
104                 break;
105             }
106             decorator.updatePlayer(player);
107         }
108         return player.getAccount().getBalance() > amount;
```

handleInsufficientFunds(). Så længe det ikke er lykkedes at rejse beløbet (amount) spørges spilleren om han vil handle. Vil han det, delegeres ansvaret til tradeController, ellers brydes loopet.

getPlayersLeft forklarer lidt sig selv. Den kører gennem spillerne og tjekker om de er broke.

Metoden hostileTakeOver tager to spillere. En kreditor og en debitor. Hvis en spiller er gået fallit mens han var i gæld til en anden spiller bliver resten af hans penge, og hans resterende grunde overdraget til kreditoren. Hvis han er gået fallit til banken bliver hans grunde nulstillet.

playerSetup(...) køres ved spillets start og sikrer at spillerne adspørges om antal spillere og navne. nameCheck(...) bruges til at sikre at spillerne ikke bruger ens navne, da GUI'en ikke understøtter dette.

TradeController

Denne klasse er en af de helt store da den kontrollerer al form for handel i spillet. Vi har implementeret: jævnt køb og salg af huse, man skal eje alle grunde i en gruppe for at bygge huse, man kan sælge sine grunde til andre spillere, man kan pantsætte (kun hvis alle grunde i samme gruppe ikke har huse) og ophæve pantsætning. Alle disse funktioner bliver styret af TradeControlleren. Fordi vi ikke er tilladt at bruge arrayList er klassen

også blevet en smule mere omfangsrig. Når man spiller spillet har man to valgmuligheder med hensyn til handel: huse eller grunde. Hvis man vælger handel med grunde, kalder man `trade()` og passer den aktive spiller, `decoratoren` og `playerControlleren` med. `trade()` kører nogle tjek med hjælp fra nogle hjælpe metoder, som afgør hvilke muligheder spilleren har. Hvis spilleren ikke har nogle grunde får han det straks at vide, og hvis han har pantsat alle sine grunde får han også det at vide, og får derved kun mulighed for at sælge eller ophæve pantsætning af sine grunde. `trade()` sender ansvaret videre til enten `sell()`, `pawn()` eller `unPawn()` alt efter hvad spilleren vælger, disse sørger så for resten af handels forløbet på den måde er `trade()` kun en slags menu eller mellem led. For brugervenlighed har vi også sørget for at de lister som kommer frem når en spiller skal sælge, pantsætte eller ophæve pantsætning, kun viser de felter som opfylder kravene for de forskellige valg. Det har vi formået at lave med flere metoder i `TradeControlleren`, som tjekker for alle de forskellige ting. En af udfordringerne ved kun at bruge "almindelige" arrays var at man blev først nødt til at tælle hvor mange pladser arrayet skulle have, før man kunne fylde dem op, hvis et array

```

372     private Street[] getBuyBuildingFields(Street[] buildableFields){
373
374         Street[] buyHouseFields = new Street[buildableFields.length];
375         for(int i = 0; i < buildableFields.length; i++){
376             boolean legitField = true;
377             if(buildableFields[i].getBuildings() < Street.MAX_NUMBER_OF_BUILDINGS){
378                 for(int j = 0; j < buildableFields.length; j++){
379                     if(buildableFields[i].getGroup().equals(buildableFields[j].getGroup())
380                        && !buildableFields[i].equals(buildableFields[j])){
381                         if(buildableFields[i].getBuildings() >
382                            buildableFields[j].getBuildings()){
383                             legitField = false;
384                             break;
385                         }
386                     }
387                 }
388             }
389             else {
390                 System.out.println("LegitField = false");
391                 legitField = false;
392             }
393             if(legitField){
394                 buyHouseFields[i] = buildableFields[i];
395             }
396         }
397         return removeNullStreets(buyHouseFields);
398     }

```

getBuyBuildingFields(): et eksempel på et af de mere komplicerede tjeks. Denne metode finder de grunde hvor der kan bygges huse. Den tjekker bl.a. for antal bygninger på hver enkel grund, og for antal bygninger på grunde i samme gruppe

indeholdt null fik vi en null pointer exception. Nogle metoder med simple tjeks tæller først, og derefter fylder array'et op. Senere fandt vi en løsning med hjælp af en ekstra metode som fjernede alle null'er som den fandt, derved skulle listen bare være lang nok til at starte. Denne metode kunne bruges flere steder, så det forkortede vores kode en hel del i TradeControlleren. Buildings() fungerer meget som trade(), den kalder alle metoder med hensyn til bygninger, og ligesom trade() bliver også her kun vist de felter som spilleren kan bygge på eller som han kan sælge af. på forrige side er et eksempel af et tjek for grunde hvor der kan bygges huse. Før metoden bliver kaldt, har en anden metode fundet alle de grupper spilleren ejer, af typen Street, som er de eneste hvor der kan bygges huse, dvs at grunde i en gruppe, hvis ejer ikke er den samme for alle, ikke er med. Den store for loop iterer derfor kun over de felter hvor der kan være en mulighed for at der kan bygges huse. For hver af disse grunde bliver der først tjekket for antal huse, hvis grunden har max antal huse (dvs 5 som svarer til et hotel) bliver den ikke tilføjet listen. Hvis grunden har mindre end 5 huse, må alle buildableFields gennemgås og sammenlignes. Hvis en grund af samme gruppe har et mindre antal huse, bliver aktuel grund ikke tilføjet listen. Da alle grunde er kørt igennem bliver null's fjernet fra listen og derefter returneret.

BoardController

BoardControlleren har som eneste attribut et board, der er gjort static, så vi kan have et sæt statiske hjælpermetoder - getFieldNumber(Field), getFieldsbyPlayer(Player), getNumberOfFleets(Player) og hasAnyUnPawndFields(Player). Disse metoder kan tilgås globalt og omsætter enten et Field til et feltnummer, genererer en liste af felter fra en Player, eller kontrollerer om en spiller har felter der ikke er pansat. I en evt. fremtidig refactoring, vil vi forsøge at slippe af med det globale scope, men det kræver en del omskrivning og tiden tillod det ikke.

BoardControlleren har som nævnt overtaget de polymorphe felters landOnField metode. Denne gang har vi i stedet konstrueret en LandOnFieldMetode(...) i boardControlleren. Denne metode undersøger feltets type med 'instanceof' og kører herefter den tilhørende kode, enten umiddelbart eller for Tax og Ownables tilfælde i private metoder for at bevare overblikket. Er feltet et Refuge gives der blot besked, er det et GoToJail, bedes playercontrolleren om at flytte spilleren til fængslet, er det et Chance, delegeres opgaven til cardControlleren.

LandOnOwnable(...) afvikler tre forskellige kodebidder alt efter ejeren af feltet. 1) Er feltet ikke ejet, tilbydes spilleren at købe feltet i den private metode OfferOwnableToPlayer(...). 2) Er det ejet af en anden spiller tvinges han til at betale lejen for feltet. 3) Er det ejet af spilleren selv vises blot en besked om fred og ingen fare. Når feltet er ejet af en anden spiller afgøres lejen ved at adspørge feltet. Herefter forsøges lejen inddrevet via playerControlleren med metoden payDebt(...). Mislykkes det at

inddrive gælden gives spilleren besked og afvikles af playerControllens metode - `hostileTakeOver(...)`.

`LandOnTax(...)` afgør først om Tax feltet er af typen hvor der skal betales et fast beløb, eller om man kan vælge at betale en procentsats i stedet. Er der et valg adspørges spilleren. Herefter forsøges det at inddrive gælden via `playerController.payDebt(...)`. Går spilleren fallit gives han besked og spilleren afvikles via `playerController.hostileTakeOver(...)`, der dog kaldes uden en creditor, hvilket signalerer at spillerens aktiver skal overgå til banken.

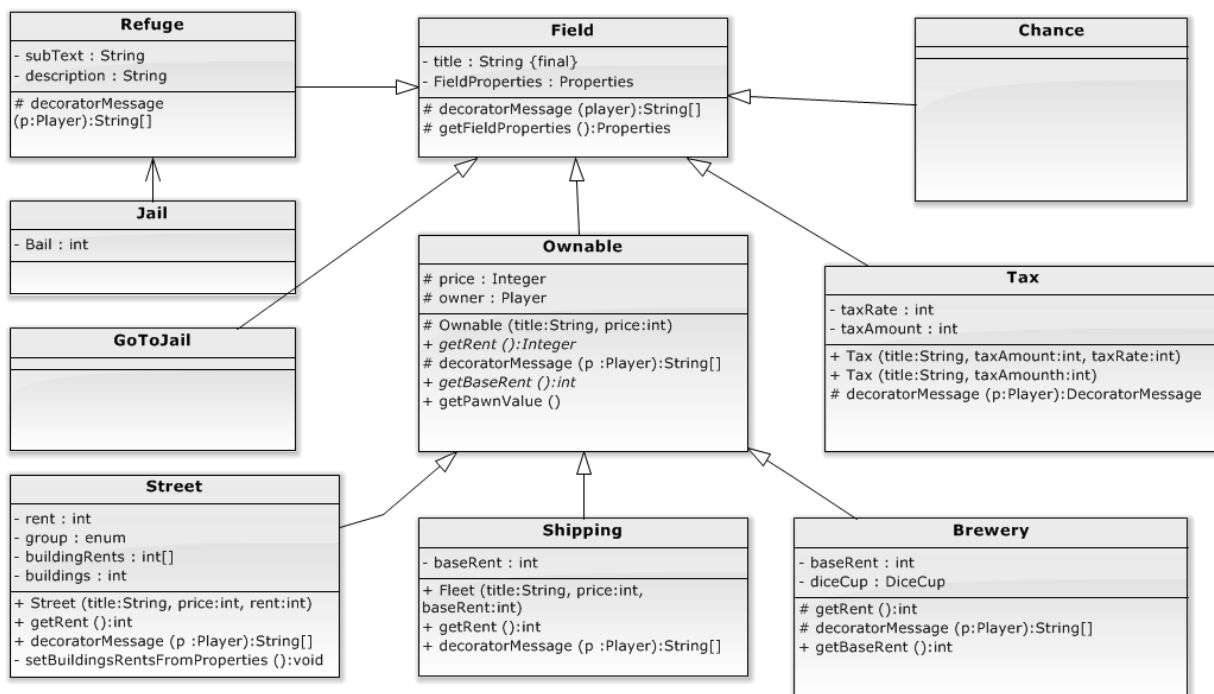
`offerOwnableToPlayer(...)` tilbyder spilleren at købe feltet. Siger han ja køres en try catch blok der forsøget at trække pengene - lykkes det updateres feltet med ejer og brættet opdateres. Er feltet et Fleet opdateres alle felter for at sikre at alle flåder viser den korrekte leje. Mislykkes det at trække pengene sørger en `InsufficientFundsException` for at spilleren får mulighed for at rejse pengene ved at handle - medmindre han ingen felter ejer. Lykkes det ikke at rejse penge, brydes loopet.

Board

Board søger fortrinsvis for at holde et array af Fields. Den har en enkelt hjælpermetode, `getFieldByNumber`, der omsætter et feltnummer til et felt. Desuden har den en `loadPropertiesFile()` metode, der er planlagt til at kunne læse felterne fra en properties fil i en fremtidig version. Constructoren fylder selv arrayet med standard matadorfelter, hvis man kalder den med int'en 40.

Field klasser

Field klasserne anvender arv på samme måde som i delprojekt 3, med den undtagelse at controllerlogikken som nævnt er flyttet til BoardController (Se diagram nedenfor). Field superklassen har en metode `getFieldProperties`, der sørger for at Properties filen, der indeholder data om felterne kun læses en gang. Filen bruges af subclasserne til at konfigurere felterne. Arv og polymorfi anvendes i metoden `decoratorMessage(...)` til at generere en `String[]` der vises til spilleren, når han lander på feltet.



CardController:

CardControlleren er, som navnet hentyder, den controller, der styrer operationen af kortene i spillet. I constructeren bliver kortbunken oprettet i en Card-array, som derefter bliver blandet af vores Shuffler klasse.

Metoden getNextCard giver os det næste kort i decket, og bruger moveCards metoden til at flytte kortene i bunken og smide det brugte kort bagerst. Der er dog en undtagelse, som er, når man trækker et fængselskort. Når der trækkes et fængselskort bruger vi 'instanceof', som er en slags pseudo-polymorphi til at genkende, at det er et fængselskort, og derefter flytte det ind i et jailCard-array, og sende spilleren oplysningen om, at han har et fængselskort. Dette gør vi, fordi at når man spiller normalt matador, tager man jo fængselskortet ud af bunken, og smider det ikke i igen, før det er blevet brugt. Vi blander heller ikke bunken, efter at den er blevet kørt igennem, men lader den bare køre rundt, da reglerne ikke siger noget om, at de skal blandes, men blot at et kort skal nedest i bunken, efter at det er blevet brugt.

Til at løse problemet med kortene, hvor den pågældende spiller skal betale et beløb for hvert hus og hotel, han ejer, har vi lavet en metode, der hedder getBuildingExpenses. Den tager en player, en int houseExpense, en int hotelExpense, og en boardController, og kører så igennem alle spillerens fields, bruger igen instanceof til at finde hans gader, og adderer alle hans huse, og hoteller. Vi har valgt at gøre så hotellet tæller som et femte hus, så hvis huse == 5 på et felt tæller det som et hotel, og ikke huse. Sidenhen bliver regnestykket lagt sammen, og vi kan i det pågældende kort bestemme hvad omkostning per hus og per hotel skal være.

Vores største metode i CardController er den, som hedder drawCard. Først kører vi getNextCard. Herefter har vi brugt instanceof rigtig meget, for at bestemme hvilket kort, der er blevet trukket, og hvad der skal ske når det pågældende kort bliver trukket.

Kort Klasserne:

Vi har brugt arv til kort klasserne.

Klassen Card er en abstrakt klasse, hvis constructor holder en String cardDescription. Den er superklasse til alle de andre kortklasser.

De forskellige kortklasser holder i virkeligheden næsten ingen information. Blot en cardDescription, og så- an på hviken kortklasse, der er tale om, så en int eller to. Kortenes logik ligger i CardControlleren, der så bruger kortenes informationer og manipulerer med dem.

Test og kvalitetssikring

Vi har løbende gennem opbygning af programmet unit testet delelementer med 'test drivers' i form af lokale main metoder, der tester for klassernes egne funktioner. Disse er stadig at finde i klasserne og kan tilrettes til at teste delelementer af koden efter behov. Vi har forsøgt at prøve at teste de branches der er i alle klasser, således at al kode bør være eksekveret mindst en gang. De mest komplekse og vigtigste funktioner er testet med data der modsvare de mulige ækvivalensklasser.

Venner og familie har gennemført brugertests. Her blev vi opmærksomme på at fornemmelsen af at slå med terninger er vigtig for nogle brugere, hvorfor vi tilføjede 'pynt' - så det gav en fornemmelse af at man slår med terningerne.

Systematiske JUnit tests har vi ikke kunnet nå at gennemføre, da vi har været få personer til at løfte arbejdsbyrden (jvf. timeregnskabet).

I bilagene under test scenarier kan nogle af nogle scenarierne ses gennemløbet. Tax felterne er afprøvet både med tilstrækkelig og utilstrækkelige penge. Streets er afprøvet for om de kan købes, betales leje på og bygges huse på samt om de kan pantsættes, sælges og hæves pantsætningen samt sælge huse. De øvrige felter er testet for om de returnerer de rette svar ved hver metode.

Chancekortene er afprøvet ved at lade en spiller lande på et chance felt og få alle de prædefinerede kort efter tur. Player er allerede grundigt gennemtestet i tidligere delopgaver - ligeledes diceCup.

Controllers er testet med test drivers og test entities. Resterne kan som nævnt ses i klasserne.

Kvalitet - 'FURPS+'

Vi har forsøgt at vurdere vores spil ud fra FURPS+

- **Functionality:** Vi har lavet et funktionelt spil, som simulerer næsten alle regler i Matador. Undtaget er auktioner, at sælge benådningskortet og Matadorlegatet.
- **Usability:** Spillet er brugervenligt, og der behøves ikke at læses nogen brugervejledning for at kunne gennemføre spillet. Spillet giver beskeder, der tillader en spiller med kendskab til Matador at spille spillet uden yderligere instruktion. Interfacet kan være lidt bøvet at klikke sig igennem, men alle valgmuligheder optræder på rigtige tidspunkter
- **Reliability:** Gennem brugertests og Unit tests, har vi elimineret så mange fejl som muligt. Controllernes mest komplekse funktioner er testet gennem Unittests. I vores sidste iteration har vi forsøgt at afprøve alle tænkelige branches og spillet kører uden crashes. Vi har således et relativt pålideligt spil.
- **Performance:** Dette har ikke været relevant faktor i vores spil. Spillet bruger så få ressourcer at vi har ikke haft brug for køretids-analyser og optimering, spillet optager heller ikke mærkbar disk-space.
- **Supportability:** Vi har gjort en hel del ud af at fremtidssikre koden, så vi kan genbruge så meget som muligt af koden senere. For eksempel har vi undgået "hard coding" i videst muligt omfang, og al tekst i spillet bliver gennem identifiere oversat, ved hjælp af decorator og properties filer, til meningsfyldt tekst. Properties filerne indeholder al tekst, og det gør det nemt at have overblik og ændre på teksten, samt at udvide til flere sprog. Alle klasser er søgt holdt med lav kobling og høj cohesion, således at delementerne kan genbruges og rettes med færrest mulige konsekvenser i andre klasser.
- **+ (Plusset):** Spillet kræver at computeren har og kan køre en opdateret version af java 1.6 og at mus og tastatur er tilkoblet. Vil man have temasangen med, kræves der også højtalere. Ellers er der ikke nogle nævneværdige krav som spillet har til computeren.

Konklusion

Alt i alt er det lykkedes os at udvikle et Matadorspil med et avanceret handelssystem, der tillader spillerne at rejse penge, når behovet er der - i tråd med reglen 'spillerne kan handle når som helst'. Det er lykkedes os at anvende mange af de patterns og designmetoder vi har lært om, og vi har anvendt alle kodeteknikker fra undervisningen og tilføjet lidt ekstra.

Projektet har lidt under manglende arbejdskraft (se timeskema) og således har vi måttet gå på kompromis - Dette er gået mest ud over testing, der vil få mere opmærksomhed i en fremtidig iteration.

Vi har forsøgt os med nye patterns og strategier, og har således forsøgt at overholde BCE-patternet strengere end sidst. Vi har eksperimenteret med Singleton pattern og har anvendt subclassede exceptions til at håndtere gældsspørgsmålet. Desuden har vi haft stort fokus på high cohesion og low coupling og har refactoreret mange gange for at opnå at kode ligger i de logiske klasser og at kode ikke går igen flere steder - for at øge supportability.

Alt dette har efter vores overbevisning givet os betydeligt flere værktøjer at arbejde med og øget vores evner som udviklere væsentligt, hvilket også ses i det færdige resultat.

Bilag

Kildekode

Kildekoden er vedlagt i mappen kildekode i projektet. Kodens organisering fremgår af projekttræet i src mappen og er som følger

(default package)

 Main

game

 Account

 Board

 BoardController

 Decorator

 DiceCup

 Die

 GameController

 MainMenuController

 Player

 PlayerController

 StringTools

 TradeController

 Translator

game.cards

 Card

 CardController

 GetOutOfJailCard

 GotoJailCard

 MoveFieldsCard

 MoveToCard

 PayCard

 PayForBuildingsCard

 ReceiveCard

 ReceiveFromPlayersCard

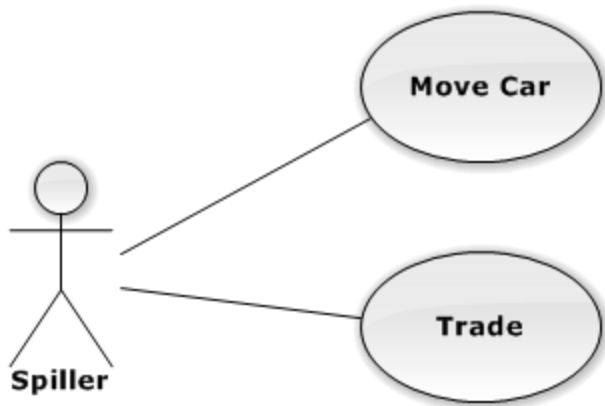
 Shuffler

game.fields

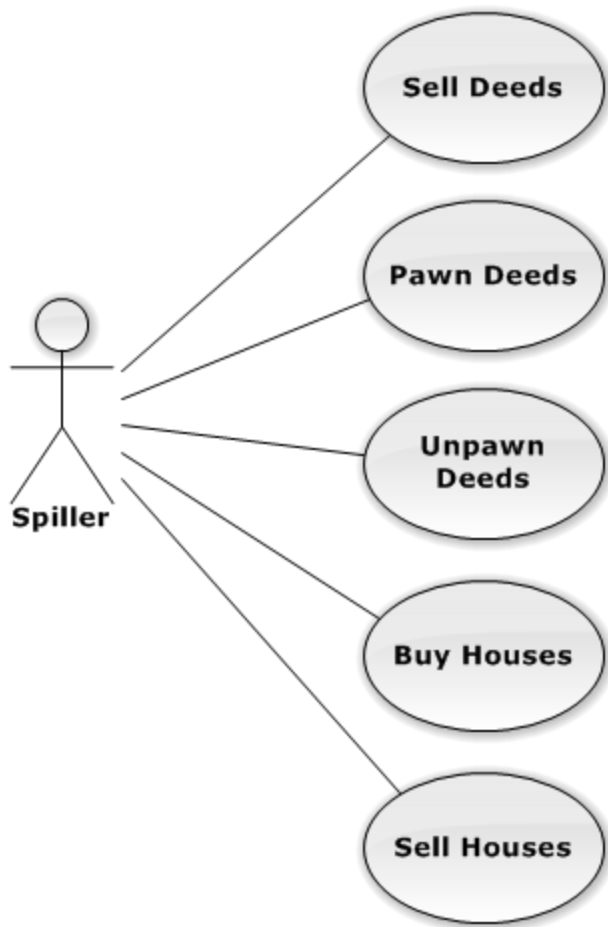
 Brewery

Chance
Field
GotoJail
Jail
Ownable
Refuge
Shipping
Street
Tax
test
TestGUI
cards.properties
danish.properties
english.properties
faroese.properties
Field.properties

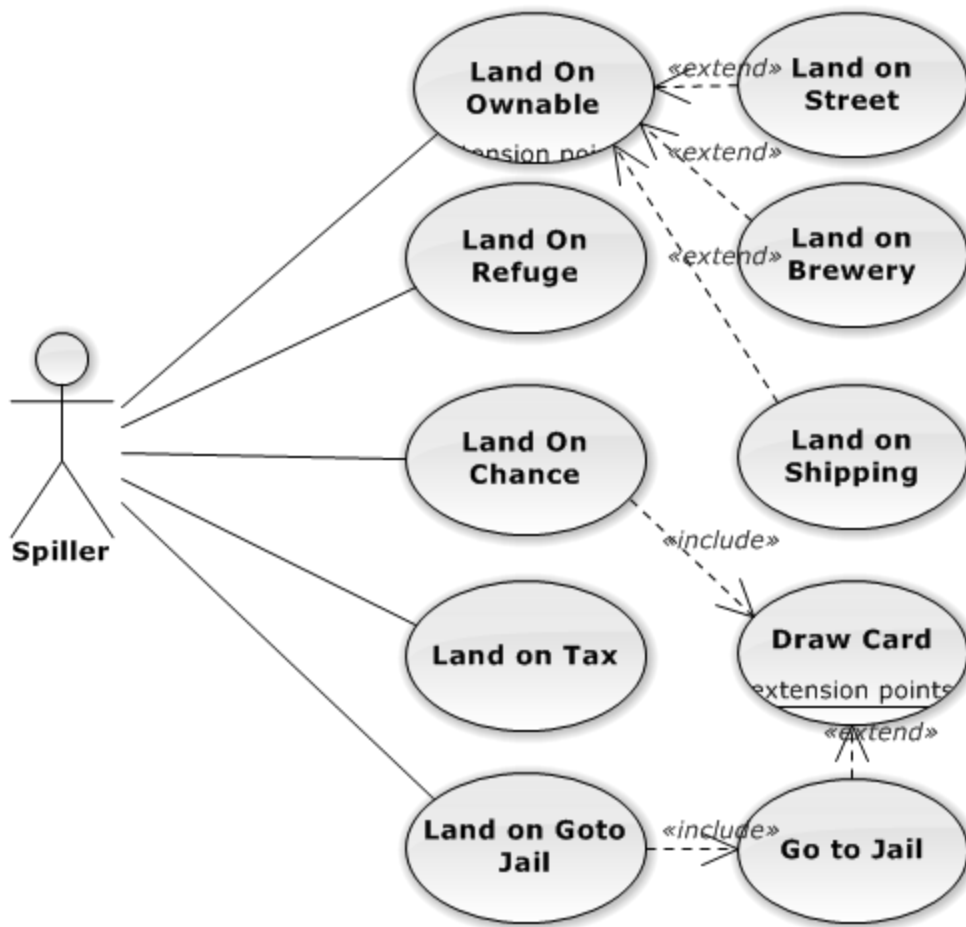
Sub Use Case Diagrammer



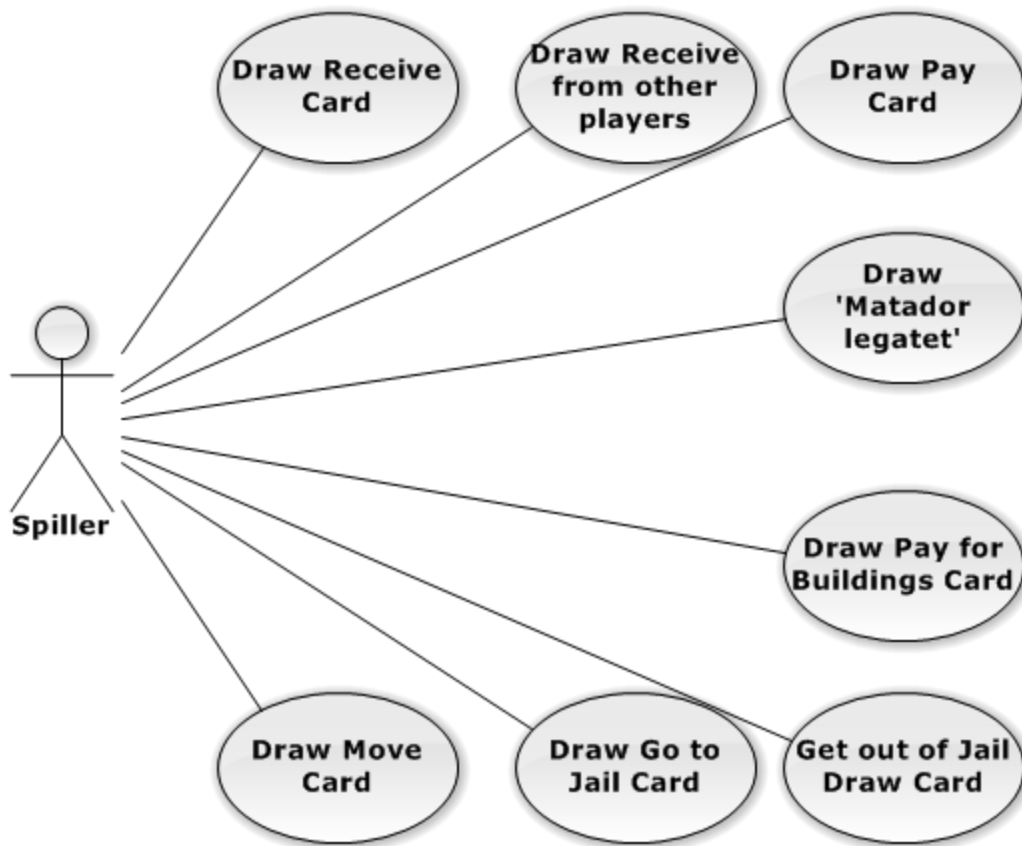
Sub Use Case Player Acts.



Sub Use Case Player Trades.





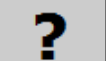
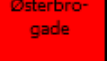
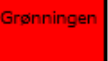

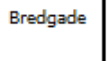
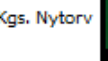
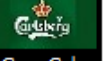
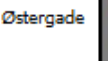



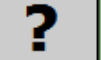


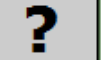





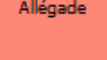
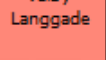
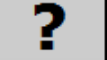
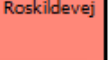


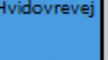
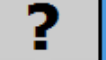
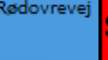

Sub Use Case Land On Field. Bemærk at sub use casen er en include for Go to jail og en extend på Land on Chance.


















Sub Use Case Draw Card.

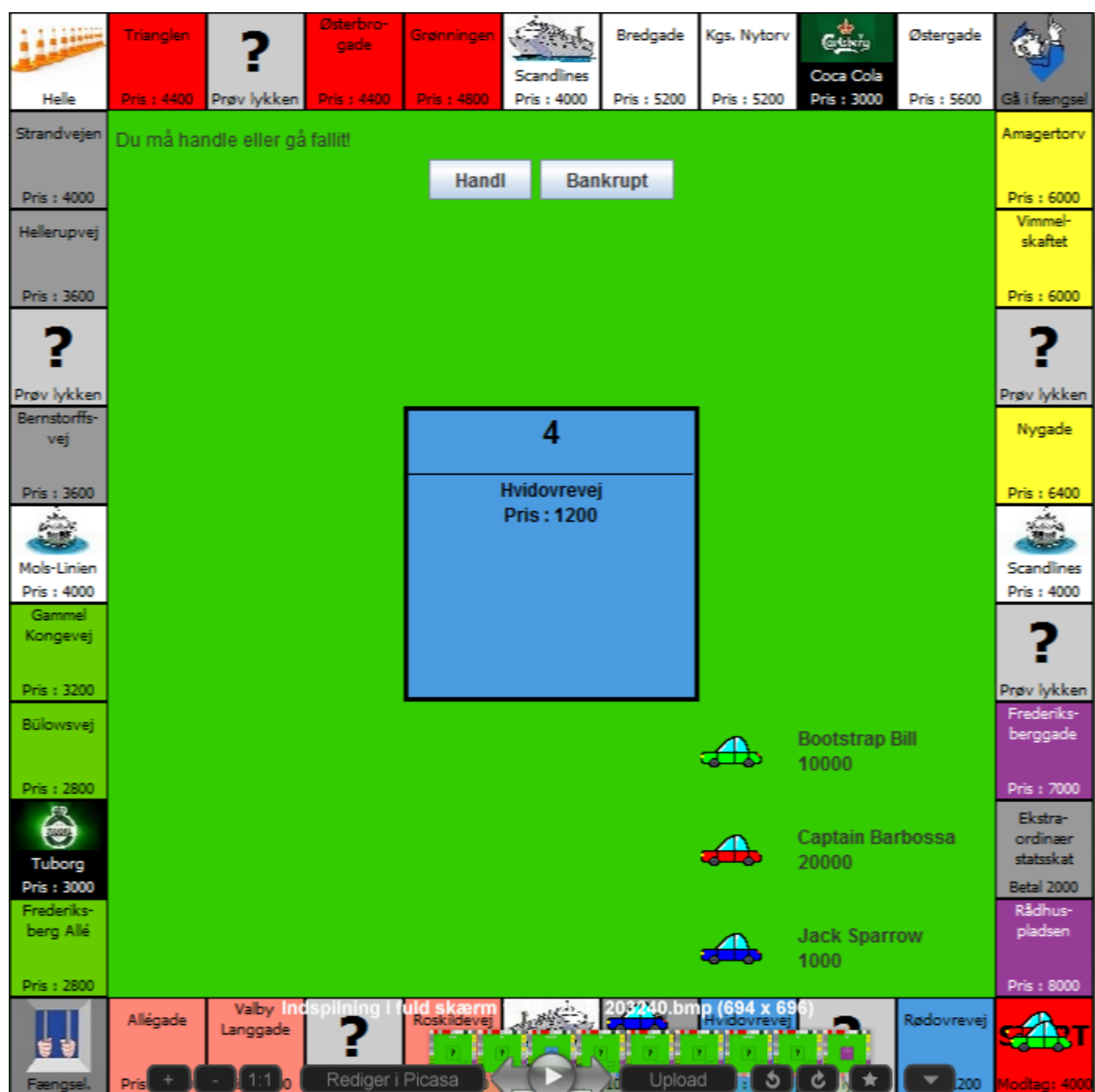
Testdokumentation

Scenarie 'Gå fallit'















 Helle	 Pris : 4400	 Prøv lykken	 Pris : 4400	 Pris : 4800	 Pris : 4000	 Pris : 5200	 Pris : 5200	 Pris : 3000	 Pris : 5600	 Gå i fængsel
Strandvejen Pris : 4000	<p>Jack Sparrow, du er landet på Betal indkomstskat. Du skal betale skat: 4000 or 10%</p> <div> <input type="text" value="4000"/> <input type="button" value="OK"/> </div>									Amagertorv Pris : 6000
Hellerupvej Pris : 3600										Vimmel-skaftet Pris : 6000
 Prøv lykken										 Prøv lykken
Bernstorffs-vej Pris : 3600										Nygade Pris : 6400
 Pris : 4000										 Pris : 4000
Gammel Kongevej Pris : 3200										 Prøv lykken
Bülowsvej Pris : 2800	 Bootstrap Bill 10000									Frederiks-berggade Pris : 7000
 Pris : 3000	 Captain Barbossa 20000									Ekstra-ordinær statsskat Betal 2000
Frederiks-berg Allé Pris : 2800	 Jack Sparrow 1000									Rådhus-pladsen Pris : 8000
 Fængsel	 Pris : 2400	 Pris : 2000	 Prøv lykken	 Pris : 2000	 Pris : 4000	 10% el. 4000	 Pris : 1200	 Prøv lykken	 Pris : 1200	 Modtag: 4000

Spilleren lander på Tax

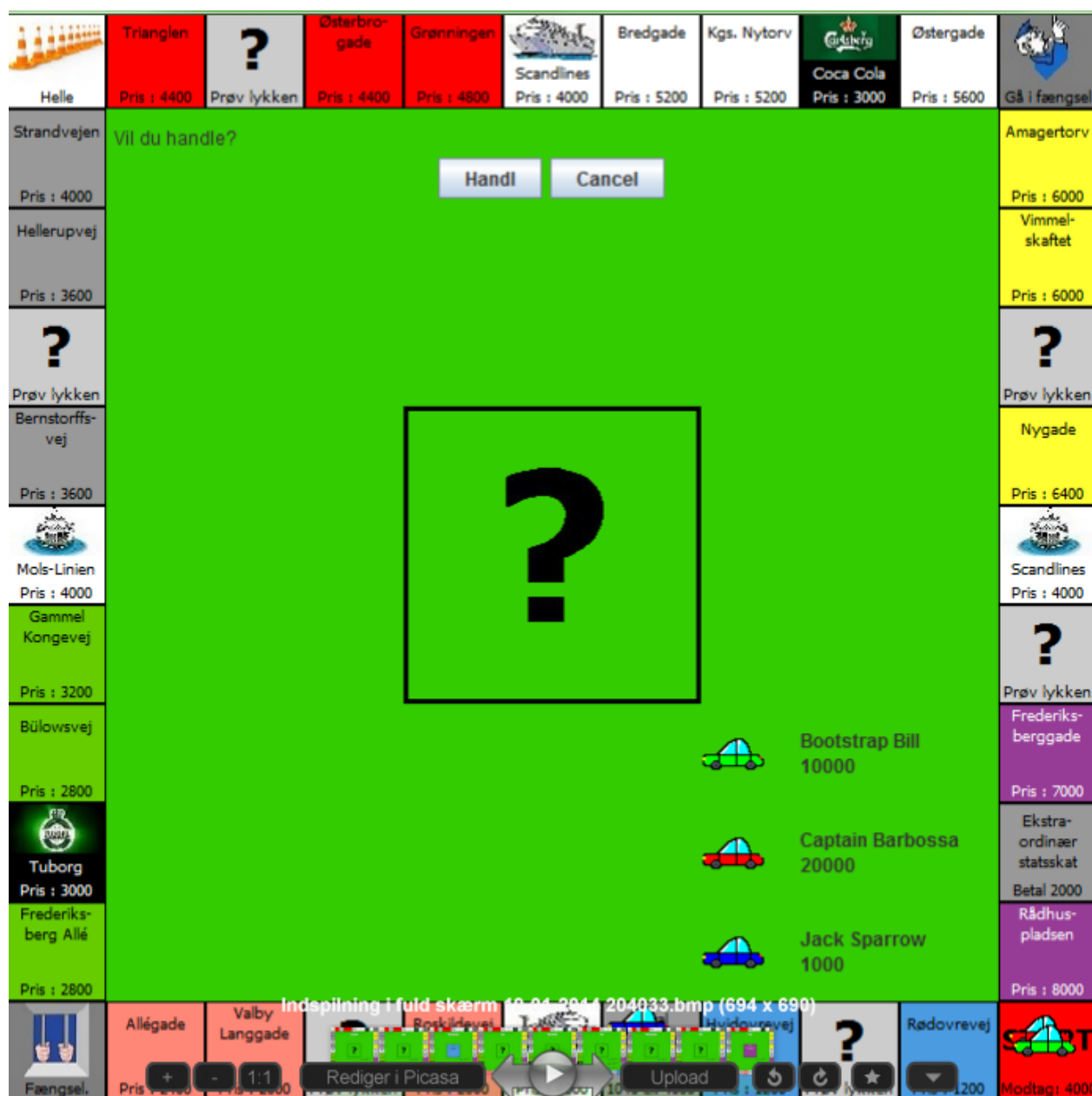
	Trianglen Pris : 4400	? Prøv lykken	Østerbro- gade Pris : 4400	Grønningen Pris : 4800	 Scandlines Pris : 4000	Bredgade Pris : 5200	Kgs. Nytorv Pris : 5200	 Coca Cola Pris : 3000	Østergade Pris : 5600	 Gå i fængsel
Strandvejen Pris : 4000	<div> <p>. Du skal betale skat 4000</p> <p>OK</p>  <div>  Bootstrap Bill 10000  Captain Barbossa 20000  Jack Sparrow 1000 </div> </div>									Amagertorv Pris : 6000
Hellerupvej Pris : 3600										Vimmel- skaffet Pris : 6000
? Prøv lykken										? Prøv lykken
Bernstorffs- vej Pris : 3600										Nygade Pris : 6400
 Mols-Linien Pris : 4000										 Scandlines Pris : 4000
Gammel Kongevej Pris : 3200										? Prøv lykken
Bülowsvej Pris : 2800										Frederiks- berggade Pris : 7000
 Tuborg Pris : 3000										Ekstra- ordinaer statsskat Betal 2000
Frederiks- berg Allé Pris : 2800										Rådhus- pladsen Pris : 8000
 Fængsel	Allégade Pris : 2400	Valby Langgade Pris : 2000	? Prøv lykken	Roskildevej Pris : 2000	 Scandlines Pris : 4000	 Indkomst- skat 10% el. 4000	Hvidovrevej Pris : 1200	? Prøv lykken	Rødovrevej Pris : 1200	 SZRT Modtag: 4000















Tilbydes at handle eller gå fallit


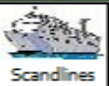


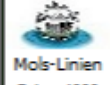

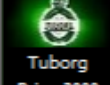


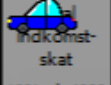

	Trianglen Pris : 4400	? Prøv lykken	Østerbro- gade Pris : 4400	Grønningen Pris : 4800	 Scandlines Pris : 4000	Bredgade Pris : 5200	Kgs. Nytorv Pris : 5200	 Coca Cola Pris : 3000	Østergade Pris : 5600	 Gå i fængsel
Strandvejen Pris : 4000	Du har ikke nok penge, du mangler: 4000Du bliver nødt til at handle/pantsætte. <div>OK</div>									Amagertorv Pris : 6000
Hellerupvej Pris : 3600	<div>?</div>									Vimmel- skaffet Pris : 6000
? Prøv lykken										? Prøv lykken
Bernstorffs- vej Pris : 3600										Nygade Pris : 6400
 Mols-Linien Pris : 4000										 Scandlines Pris : 4000
Gammel Kongevej Pris : 3200										? Prøv lykken
Bülowsvej Pris : 2800										Frederiks- berggade Pris : 7000
 Tuborg Pris : 3000										Ekstra- ordinær statsskat Betal 2000
Frederiks- berg Allé Pris : 2800										Rådhus- pladsen Pris : 8000
<div><div> Bootstrap Bill 10000</div><div> Captain Barbossa 20000</div><div> Jack Sparrow 1000</div></div>										
 Fængsel.	Allégade Pris : 2400	Valby Langgade Pris : 2000	? Prøv lykken	Roskildevej Pris : 2000	 Scandlines Pris : 4000	 10% el. 4000	Hvidovrevej Pris : 1200	? Prøv lykken	Rødovrevej Pris : 1200	 Modtag: 4000

Besked om hvor mange penge der mangler


































	Triangeln	?	Østerbro-gade	Grønningen		Bredgade	Kgs. Nytorv		Østergade	
Helle	Pris : 4400	Prøv lykken	Pris : 4400	Pris : 4800	Scandlines Pris : 4000	Pris : 5200	Pris : 5200	Coca Cola Pris : 3000	Pris : 5600	Gå i fængsel
Strandvejen	Du har ikke nogen felter at handle med.									Amagertorv
Pris : 4000	OK									Pris : 6000
Hellerupvej										Vimmel-skaftet
Pris : 3600										Pris : 6000
?										?
Prøv lykken										Prøv lykken
Bernstorffs-vej										Nygade
Pris : 3600										Pris : 6400
										
Mols-Linien Pris : 4000										Scandlines Pris : 4000
Gammel Kongevej										?
Pris : 3200										Prøv lykken
Bülowsvej										Frederiks-berggade
Pris : 2800										Pris : 7000
										Ekstra-ordinær statsskat
Tuborg Pris : 3000										Betal 2000
Frederiks-berg Allé										Rådhus-pladsen
Pris : 2800										Pris : 8000
	Allégade	Valby Langgade	?	Roskildevej			Hvidovrevej	?	Rødovrevej	
Fængsel	Pris : 2400	Pris : 2000	Prøv lykken	Pris : 2000	Scandlines Pris : 4000	Indkomst-skat 10% el. 4000	Pris : 1200	Prøv lykken	Pris : 1200	Modtag : 4000

Ingen felter at handle med

	Triangeln Pris : 4400	? Prøv lykken	Østerbro- gade Pris : 4400	Grønningen Pris : 4800	 Scandlines Pris : 4000	Bredgade Pris : 5200	Kgs. Nytorv Pris : 5200	 Coca Cola Pris : 3000	Østergade Pris : 5600	 Gå i fængsel
Strandvejen Pris : 4000	<div>Du må handle eller gå fallit!</div> <div> <div>Handl</div> <div>Bankrupt</div> </div> <div>?</div> <div> <div>Bootstrap Bill 10000</div> <div>Captain Barbossa 20000</div> <div>Jack Sparrow 1000</div> </div>									Amagertorv Pris : 6000
Hellerupvej Pris : 3600										Vimmel- skaftet Pris : 6000
? Prøv lykken										? Prøv lykken
Bernstorffs- vej Pris : 3600										Nygade Pris : 6400
 Mols-Linien Pris : 4000										 Scandlines Pris : 4000
Gammel Kongevej Pris : 3200										? Prøv lykken
Bülowsvej Pris : 2800										Frederiks- berggade Pris : 7000
 Tuborg Pris : 3000										Ekstra- ordinær statsskat Betal 2000
Frederiks- berg Allé Pris : 2800										Rådhus- pladsen Pris : 8000
 Fængsel	Allégade Pris : 2400	Valby Langgade Pris : 2000	? Prøv lykken	Roskildevej Pris : 2000	 Scandlines Pris : 4000	 Indkomst- skat 10% el. 4000	Hvidovrevej Pris : 1200	? Prøv lykken	Rødovrevej Pris : 1200	 Modtag: 4000













Må vælge fallit

	Triangeln		Østerbro-gade	Grønningen		Bredgade	Kgs. Nytorv		Østergade	
Helle	Pris : 4400	Prøv lykken	Pris : 4400	Pris : 4800	Scandlines Pris : 4000	Pris : 5200	Pris : 5200	Coca Cola Pris : 3000	Pris : 5600	Gå i fængsel
Strandvejen	<div>Du er gået fallit!</div> <div>OK</div> <div></div> <div>  Bootstrap Bill 10000  Captain Barbossa 20000  Jack Sparrow 1000 </div>									Amagertorv
Pris : 4000										Pris : 6000
Hellerupvej										Vimmel-skaftet
Pris : 3600										Pris : 6000
										
Prøv lykken										Prøv lykken
Bernstorffs-vej										Nygade
Pris : 3600										Pris : 6400
										
Mols-Linien Pris : 4000										Scandlines Pris : 4000
Gammel Kongevej										
Pris : 3200										Prøv lykken
Bülowsvej										Frederiks-berggade
Pris : 2800										Pris : 7000
										Ekstra-ordinær statsskat
Tuborg Pris : 3000										Betal 2000
Frederiks-berg Allé										Rådhus-pladsen
Pris : 2800										Pris : 8000
	Allégade	Valby Langgade		Roskildevej			Hvidovrevej		Rødovrevej	
Fængsel.	Pris : 2400	Pris : 2000	Prøv lykken	Pris : 2000	Scandlines Pris : 4000	Indkomst-skat 10% el. 4000	Pris : 1200	Prøv lykken	Pris : 1200	Modtag: 4000

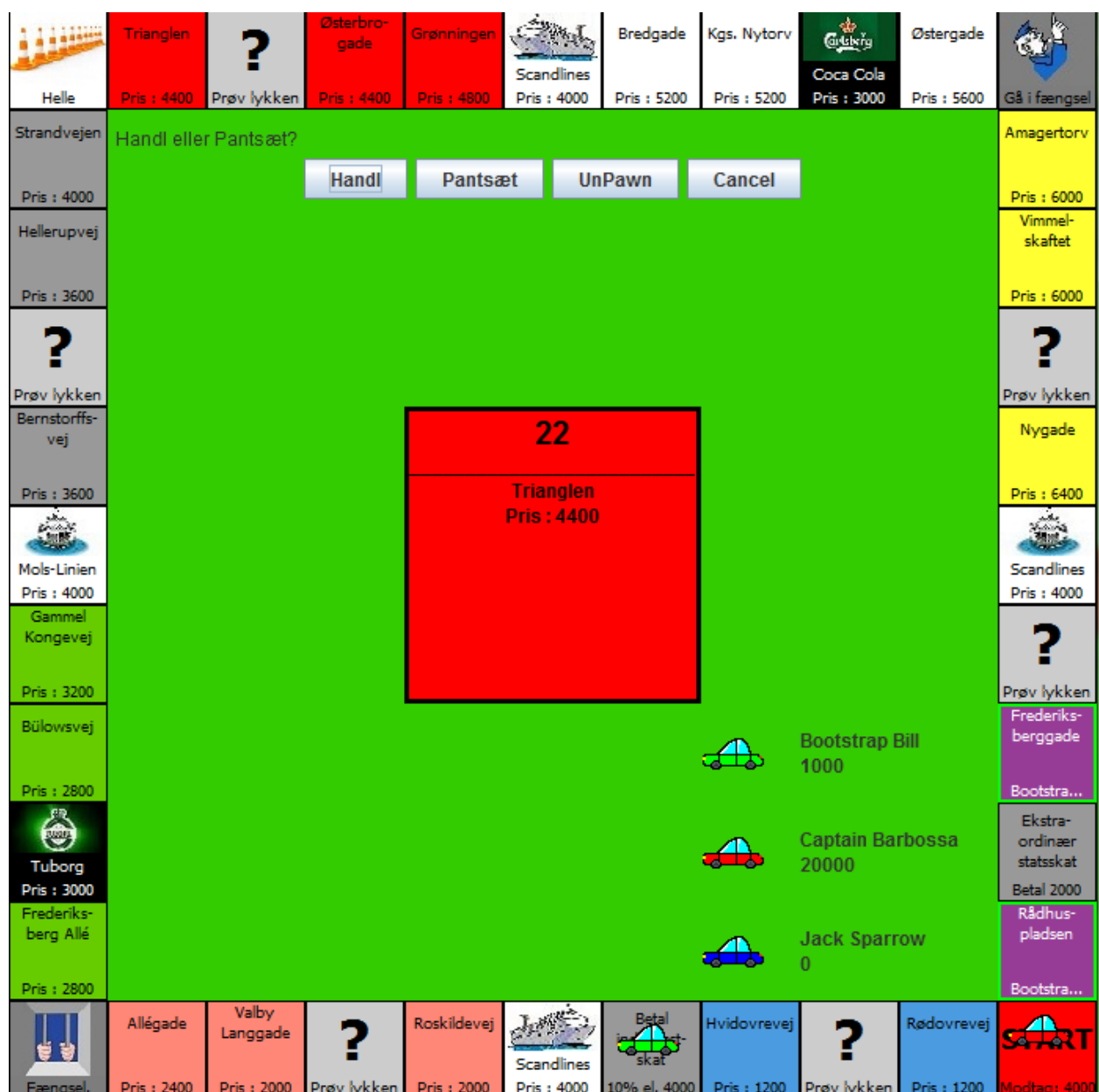
	Trianglen Pris : 4400	? Prøv lykken	Østerbro- gade Pris : 4400	Grønningen Pris : 4800	 Scandlines Pris : 4000	Bredgade Pris : 5200	Kgs. Nytorv Pris : 5200	 Coca Cola Pris : 3000	Østergade Pris : 5600	 Gå i fængsel
Strandvejen Pris : 4000	<div>Captain Barbossa- Slå med terninger!</div> <div>OK</div> <div>?</div> <div>Bootstrap Bill 10000</div> <div>Captain Barbossa 20000</div> <div>Jack Sparrow 0</div>									Amagertorv Pris : 6000
Hellerupvej Pris : 3600										Vimmel- skaftet Pris : 6000
? Prøv lykken										? Prøv lykken
Bernstorffs- vej Pris : 3600										Nygade Pris : 6400
 Mols-Linien Pris : 4000										 Scandlines Pris : 4000
Gammel Kongevej Pris : 3200										? Prøv lykken
Bülowsvej Pris : 2800										Frederiks- berggade Pris : 7000
 Tuborg Pris : 3000										Ekstra- ordinær statsskat Betal 2000
Frederiks- berg Allé Pris : 2800										Rådhus- pladsen Pris : 8000
 Fængsel	Allégade Pris : 2400	Valby Langgade Pris : 2000	? Prøv lykken	Roskildevej Pris : 2000	 Scandlines Pris : 4000	Betal indkomst- skat 10% el. 4000	Hvidovrevej Pris : 1200	? Prøv lykken	Rødovrevej Pris : 1200	 Modtag: 4000

Jack Sparrow ryger ud af spillet og mister alle penge. Bilen fjernes korrekt.
















Scenarie 'Undgå fallit'













	Triangeln Pris : 4400	? Prøv lykken	Østerbro- gade Pris : 4400	Grønningen Pris : 4800	 Scandlines Pris : 4000	Bredgade Pris : 5200	Kgs. Nytorv Pris : 5200	 Coca Cola Pris : 3000	Østergade Pris : 5600	 Gå i fængsel
Strandvejen Pris : 4000	Du er nødsaget til at pantsætte til du har rejst beløbet, eller må gå fallit. <input type="button" value="OK"/>									Amagertorv Pris : 6000
Hellerupvej Pris : 3600										Vimmel- skaffet Pris : 6000
? Prøv lykken										? Prøv lykken
Bernstorffs- vej Pris : 3600										Nygade Pris : 6400
 Mols-Linien Pris : 4000										 Scandlines Pris : 4000
Gammel Kongevej Pris : 3200										? Prøv lykken
Bülowsvej Pris : 2800										Frederiks- berggade Bootstra...
 Tuborg Pris : 3000										Ekstra- ordinær statsskat Betal 2000
Frederiks- berg Allé Pris : 2800										Rådhus- pladsen Bootstra...
 Fængsel	Allégade Pris : 2400	Valby Langgade Pris : 2000	? Prøv lykken	Roskildevej Pris : 2000	 Scandlines Pris : 4000	 Betal 10% el. 4000	Hvidovrevej Pris : 1200	? Prøv lykken	Rødovrevej Pris : 1200	 Modtag : 4000

Bootstrap bill er også uheldig at lande på Tax















Men tilbydes at handle eller pantsætte

	Triangeln	?	Østerbro-gade	Grønningen		Bredgade	Kgs. Nytorv		Østergade							
Helle	Pris : 4400	Prøv lykken	Pris : 4400	Pris : 4800	Scandlines Pris : 4000	Pris : 5200	Pris : 5200	Coca Cola Pris : 3000	Pris : 5600	Gå i fængsel						
Strandvejen	Vælg et felt som du vil pantsætte:									Amagertorv						
Pris : 4000	<div><div>Cancel</div><div>▼</div><div>OK</div></div>									Pris : 6000						
Hellerupvej	<div><div>Cancel</div><div>Frederiksberggade</div><div>Rådhuspladsen</div></div>									Vimmel-skaftet						
Pris : 3600										Pris : 6000						
?										?						
Prøv lykken										Prøv lykken						
Bernstorffs-vej										Nygade						
Pris : 3600										Pris : 6400						
	Mols-Linien	<div><div>?</div></div>								Scandlines Pris : 4000						
Pris : 4000																
Gammel Kongevej																
Pris : 3200																
Bülowsvej																
Pris : 2800																
	Tuborg	<div><div> Bootstrap Bill 1000</div><div> Captain Barbossa 20000</div><div> Jack Sparrow 0</div></div>								Frederiks-berggade						
Pris : 3000																
Frederiks-berg Allé																
Pris : 2800																
	Allégade	Valby Langgade	?	Roskildevej			Hvidovrevej	?	Rødovrevej							
Fængsel.	Pris : 2400	Pris : 2000	Prøv lykken	Pris : 2000	Scandlines Pris : 4000	10% el. 4000	Pris : 1200	Prøv lykken	Pris : 1200	Modtag: 4000						






















 Helle Pris : 4400	Trianglen Pris : 4400	? Prøv lykken	Østerbro- gade Pris : 4400	Grønningen Pris : 4800	 Scandlines Pris : 4000	Bredgade Pris : 5200	Kgs. Nytorv Pris : 5200	 Coca Cola Pris : 3000	Østergade Pris : 5600	 Gå i fængsel
Strandvejen Pris : 4000	Du skal betale skat: 4000 <input type="button" value="OK"/>									Amagertorv Pris : 6000
Hellerupvej Pris : 3600										Vimmel- skaflet Pris : 6000
? Prøv lykken										? Prøv lykken
Bernstorffs- vej Pris : 3600										Nygade Pris : 6400
 Mols-Linien Pris : 4000										 Scandlines Pris : 4000
Gammel Kongevej Pris : 3200										? Prøv lykken
Bülowsvej Pris : 2800										Frederiksberg Pawned
 Tuborg Pris : 3000										Bootstra... Ekstra- ordinaer statsskat Betal 2000
Frederiks- berg Allé Pris : 2800										Rådhus- pladsen Bootstra...
 Fængsel	Allégade Pris : 2400	Valby Langgade Pris : 2000	? Prøv lykken	Roskildevej Pris : 2000	 Scandlines Pris : 4000	 Betal skat 10% el. 4000	Hvidovrevej Pris : 1200	? Prøv lykken	Rødovrevej Pris : 1200	 Modtag: 4000

Men klarer sig ved at pantsætte Frederiksberggade












	Trianglen Pris : 4400	? Prøv lykken	Østerbro- gade Pris : 4400	Grønningen Pris : 4800	 Scandlines Pris : 4000	Bredgade Pris : 5200	Kgs. Nytorv Pris : 5200	 Coca Cola Pris : 3000	Østergade Pris : 5600	 Gå i fængsel
Strandvejen Pris : 4000	Du er landet på dit eget felt - Pyha!									Amagertorv Pris : 6000
Hellerupvej Pris : 3600										Vimmel- skaffet Pris : 6000
? Prøv lykken	<div> 21 Helle  </div> <div> . Tag en pause </div>									? Prøv lykken
Bernstorffs- vej Pris : 3600										Nygade Pris : 6400
 Mols-Linien Pris : 4000										 Scandlines Pris : 4000
Gammel Kongevej Pris : 3200										? Prøv lykken
Bülowsvej Pris : 2800										Frederiksberg Pawned Bootstra...
 Tuborg Pris : 3000										Ekstra- ordinær statsskat Betal 2000
Frederiks- berg Allé Pris : 2800										Rådhus- pladsen Bootstra...
 Fængsel	Allégade Pris : 2400	Valby Langgade Pris : 2000	? Prøv lykken	Roskildevej Pris : 2000	 Scandlines Pris : 4000	Betal indkomst- skat 10% el. 4000	Hvidovrevej Pris : 1200	? Prøv lykken	Rødovrevej Pris : 1200	 Modtag: 4000

Pengene overføres korrekt

Scenarie 'Vind Spillet'




 Helle Pris : 4400	Triangeln Pris : 4400	 Prøv lykken	Østerbro- gade Pris : 4400	Grønningen Pris : 4800	 Scandlines Pris : 4000	Bredgade Pris : 5200	Kgs. Nytorv Pris : 5200	 Coca Cola Pris : 3000	Østergade Pris : 5600	 Gå i fængsel
Strandvejen Pris : 4000	Du må handle eller gå fallit! <div> <input type="button" value="Handl"/> <input type="button" value="Bankrupt"/> </div> <div>  </div> <div>  Bootstrap Bill 500  Captain Barbossa 20000  Jack Sparrow 0 </div>									Amagertorv Pris : 6000
Hellerupvej Pris : 3600										Vimmel- skaftet Pris : 6000
 Prøv lykken										 Prøv lykken
Bernstorffs- vej Pris : 3600										Nygade Pris : 6400
 Mols-Linien Pris : 4000										 Scandlines Pris : 4000
Gammel Kongevej Pris : 3200										 Prøv lykken
Bülowsvej Pris : 2800										Frederiksberg Pawnd Bootstra...
 Tuborg Pris : 3000										Ekstra- ordinær statsskat Betal 2000
Frederiks- berg Allé Pris : 2800										Rådhuspladse Pawnd Bootstra...
 Fængsel.	Allégade Pris : 2400	Valby Langgade Pris : 2000	 Prøv lykken	Roskildevej Pris : 2000	 Scandlines Pris : 4000	 Betal skat 10% el. 4000	Hvidovrevej Pris : 1200	 Prøv lykken	Rødovrevej Pris : 1200	 Modtag: 4000

Bootstrap Bill er desværre så uheldig (vi snyder lidt med terningerne) at lande 3 gange på betalindkomstskat og går også fallit













	Trianglen Pris : 4400	? Prøv lykken	Østerbro- gade Pris : 4400	Grønningen Pris : 4800	 Scandlines Pris : 4000	Bredgade Pris : 5200	Kgs. Nytorv Pris : 5200	 Coca Cola Pris : 3000	Østergade Pris : 5600	 Gå i fængsel
Strandvejen Pris : 4000	<div>Tillykke, Captain Barbossa- Du vandt!!!</div> <div>OK</div> <div>22</div> <div>Trianglen Pris : 4400</div> <div> Bootstrap Bill 0</div> <div> Captain Barbossa 20000</div> <div> Jack Sparrow 0</div>									Amagertorv Pris : 6000
Hellerupvej Pris : 3600										Vimmel- skaftet Pris : 6000
? Prøv lykken										? Prøv lykken
Bernstorffs- vej Pris : 3600										Nygade Pris : 6400
 Mols-Linien Pris : 4000										 Scandlines Pris : 4000
Gammel Kongevej Pris : 3200										? Prøv lykken
Bülowsvej Pris : 2800										Frederiks- berggade Pris : 7000
 Tuborg Pris : 3000										Ekstra- ordinær statsskat Betal 2000
Frederiks- berg Allé Pris : 2800										Rådhus- pladsen Pris : 8000
										Allégade Pris : 2400

Da der kun er en aktiv spiller tilbage, udråbes vinderen!

Scenarie 'Køb grund'





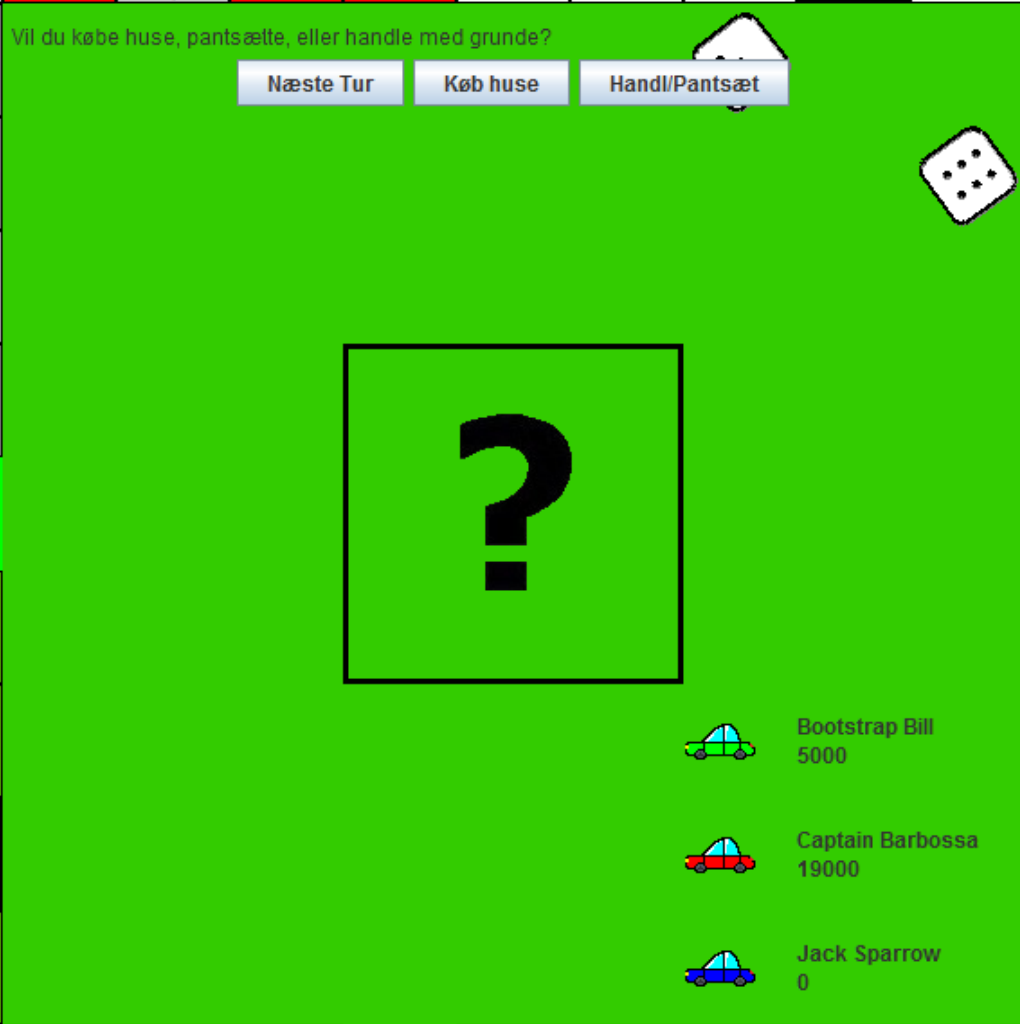









 Helle Pris : 4400	Triangeln Pris : 4400	? Prøv lykken	Østerbro- gade Pris : 4400	Grønningen Pris : 4800	 Scandlines Pris : 4000	Bredgade Pris : 5200	Kgs. Nytorv Pris : 5200	 Coca Cola Pris : 3000	Østergade Pris : 5600	 Gå i fængsel
Strandvejen Pris : 4000	Bootstrap Bill, du er landet på Roskildevej, der koster 2000og indbringer 100i leje.. Vil du købe feltet?									Amagertorv Pris : 6000
Hellerupvej Pris : 3600	<input type="button" value="Ja"/> <input type="button" value="Nej"/>									Vimmel- skaflet Pris : 6000
? Prøv lykken	<div> <div>21 Helle</div>  <div>. Tag en pause</div> </div>									? Prøv lykken
Bernstorffs- vej Pris : 3600										Nygade Pris : 6400
 Mols-Linien Pris : 4000										 Scandlines Pris : 4000
Gammel Kongevej Pris : 3200										? Prøv lykken
Bülowsvej Pris : 2800										Fredenks- berggade Pris : 7000
 Tuborg Pris : 3000	 Bootstrap Bill 10000  Captain Barbossa 20000  Jack Sparrow 0									Ekstra- ordinaer statsskat Betal 2000
Frederiks- berg Allé Pris : 2800										Rådhus- pladsen Pris : 8000
 Fængsel	Allégade Pris : 2400	Valby Langgade Pris : 2000	? Prøv lykken	 Roskildevej Pris : 2000	 Scandlines Pris : 4000	Betal indkomst- skat 10% el. 4000	Hvidovrevej Pris : 1200	? Prøv lykken	Rødovrevej Pris : 1200	 Modtag: 4000

Bootstrap Bill lander på Roskildevej og tilbydes feltet

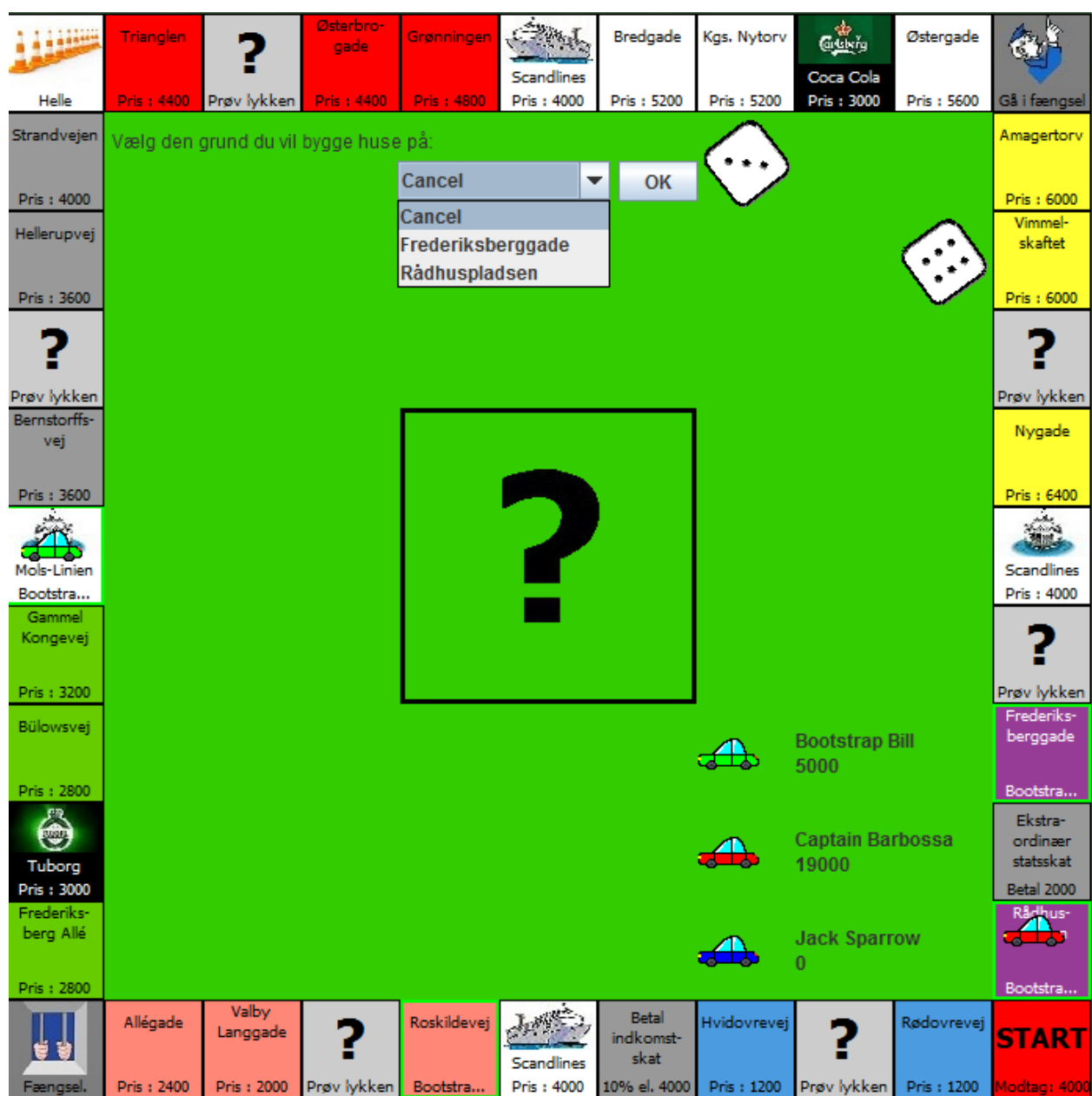
	Triangeln	?	Østerbro-gade	Grønningen		Bredgade	Kgs. Nytorv		Østergade	
Helle	Pris : 4400	Prøv lykken	Pris : 4400	Pris : 4800	Scandlines Pris : 4000	Pris : 5200	Pris : 5200	Coca Cola Pris : 3000	Pris : 5600	Gå i fængsel
Strandvejen	Captain Barbossa- Slå med terninger!									Amagertorv
Pris : 4000										Pris : 6000
Hellerupvej										Vimmel-skaftet
Pris : 3600										Pris : 6000
?										?
Prøv lykken										Prøv lykken
Bernstorffs-vej										Nygade
Pris : 3600										Pris : 6400
										
Mols-Linien Pris : 4000										Scandlines Pris : 4000
Gammel Kongevej										?
Pris : 3200										Prøv lykken
Bülowsvej										Frederiks-berggade
Pris : 2800										Pris : 7000
										Ekstra-ordinær statsskat
Tuborg Pris : 3000										Betal 2000
Frederiks-berg Allé										Rådhus-pladsen
Pris : 2800										Pris : 8000
	Allégade	Valby Langgade	?			Betal indkomst-skat	Hvidovrevej	?	Rødovrevej	
Fænøse	Pris : 2400	Pris : 2000	Prøv lykken	Roskildevej Bootstrap...	Scandlines Pris : 4000	10% el. 4000	Pris : 1200	Prøv lykken	Pris : 1200	Modtag 4000

Købet gennemføres korrekt

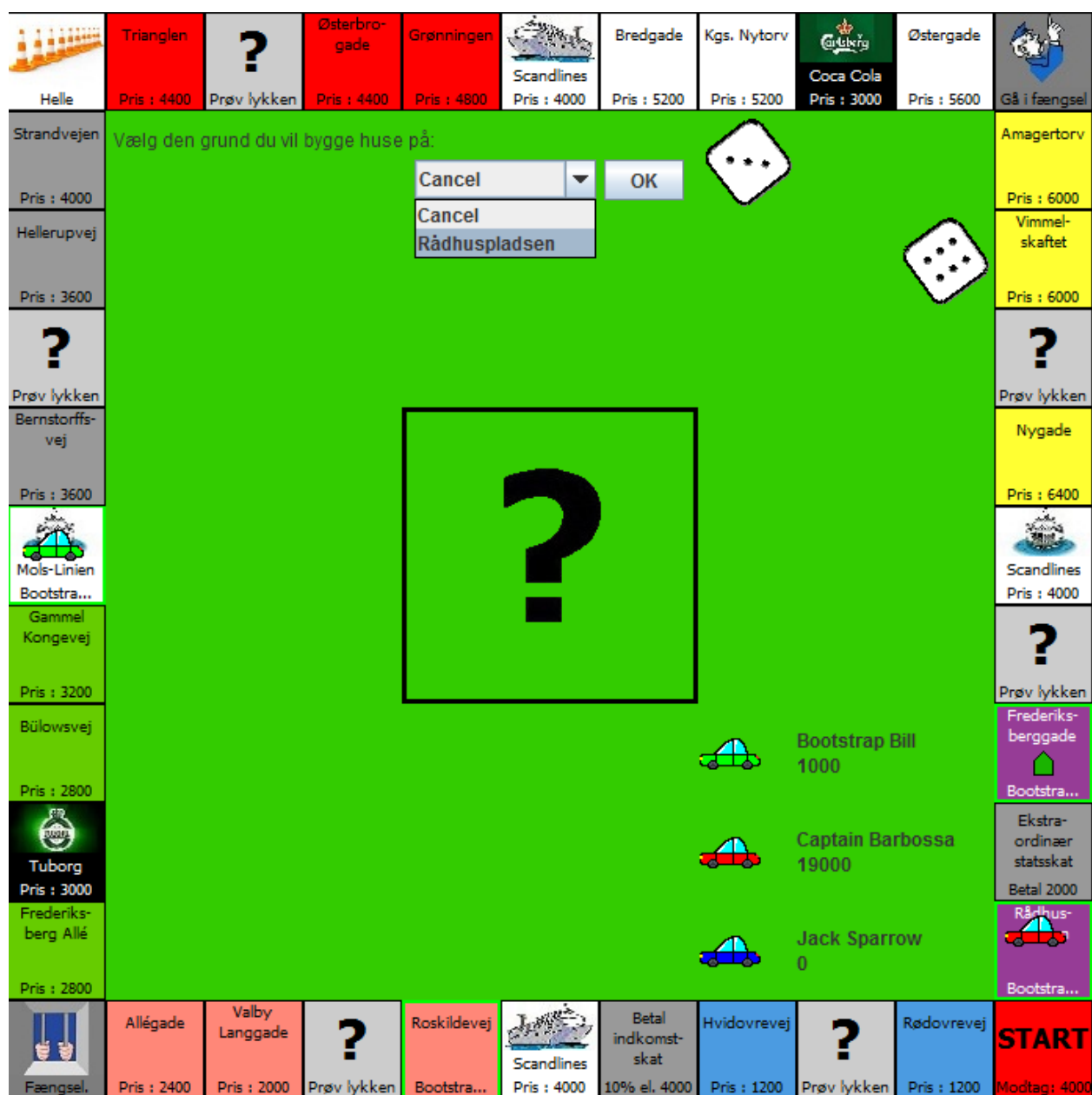
Scenarie køb og salg af huse

 Helle	Triangeln Pris : 4400	? Prøv lykken	Østerbro- gade Pris : 4400	Grønningen Pris : 4800	 Scandlines Pris : 4000	Bredgade Pris : 5200	Kgs. Nytorv Pris : 5200	 Coca Cola Pris : 3000	Østergade Pris : 5600	 Gå i fængsel
Strandvejen Pris : 4000	<p>Vil du købe huse, pantsætte, eller handle med grunde?</p> <p>Næste Tur Køb huse Handl/Pantsæt</p>									Amagertorv Pris : 6000
Hellerupvej Pris : 3600										Vimmel- skaffet Pris : 6000
? Prøv lykken										? Prøv lykken
Bernstorffs- vej Pris : 3600										Nygade Pris : 6400
 Mols-Linien Bootstra...										 Scandlines Pris : 4000
Gammel Kongevej Pris : 3200	 Bootstrap Bill 5000  Captain Barbossa 19000  Jack Sparrow 0									? Prøv lykken
Bülowsvej Pris : 2800										Frederiks- berggade Bootstra...
 Tuborg Pris : 3000										Ekstra- ordinær statsskat Betal 2000
Frederiks- berg Allé Pris : 2800										 Rådhus- Bootstra...
 Fængsel.	Allégade Pris : 2400	Valby Langgade Pris : 2000	? Prøv lykken	Roskildevej Bootstra...	 Scandlines Pris : 4000	Betal indkomst- skat 10% el. 4000	Hvidovrevej Pris : 1200	? Prøv lykken	Rødovrevej Pris : 1200	START Modtag: 4000

Bootstrap Bill er nu så heldig at eje alle grunde i en farve så han tilbydes at købe huse



Han får korrekt kun lov til at bygge på de grunde der er i lilla gruppe (ikke Roskildevej)



Efter huset er købt er det korrekt kun tilladt at bygge på Rådhuspladsen

Litteraturliste

- Larman, Craig (2005) [2004]. *Applying UML and Patterns – An Introduction to Object-Oriented Analysis and Design and Iterative Development* (3rd ed.). New Jersey: Prentice Hall. ISBN 0-13-148906-2.
- Java Software Solutions. Pearson Education, Limited (Juni 2011) John Lewis og William Loftus ISBN-10: 0273760181 ISBN-13: 9780273760184
- How to Make Dialogs - Oracle Documentation. Retrieved January 16 2014, from <http://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html>
- Singleton pattern. (2014, January 16). In *Wikipedia, The Free Encyclopedia*. Retrieved 17:48, January 17, 2014, from http://en.wikipedia.org/w/index.php?title=Singleton_pattern&oldid=590910949