

CDIO Final
‘Matador’

Udviklingsmetoder til IT-systemer 02313, Softwareteknologi

Gruppe Nr.: 31

Afleveringsfrist: Mandag 16/01-2016 12:00

Institut: DTU Compute

Vejleder: Mads Nyborg

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

Denne rapport indeholder 89 sider eks. denne side.



Jia Hao Johnny Chen, s165543



Christopher David Carlson Chytræus, s165230



Bertram Christian Henning, s153538



Jonathan Yngve Friis, s165213



Thomas Kristian Lorentzen, s154424

Abstract

The project described in this report is a 'Matador' game based partially on our previous CDIO projects. For this project, we have created several small controllers interacting with different classes and their inherited subclasses. All the controllers have been directly interacting with one major GUIcontroller, which has been interacting with the GUI itself. The project has been designed to be able to run on all data bars on DTU, which has been successfully tested. To make our development process easier, we have been modeling our concepts with different UML diagrams for both the classes and their methods. Additionally, we have been running JUnit Tests for all relevant methods for the most important classes, as well as creating formal test cases for certain scenarios not as easily tested by code. The purpose with these tests and test cases is to ensure our project did not have any game breaking flaws. The primary focus(es) of this report is a documentation of our UML diagrams, Use Cases, Test Cases and explanations of our theoretical terms. Lastly, we can conclude that our project has been very successful.

Abstract	2
Timeregnskab	4
Indledning.....	5
Emne.....	5
Formål.....	5
Kravspecifikationer	5
Functional:	5
Usability:	6
Reliability:	6
Performance:	7
Implementation:	7
Navneord- og udsagnsords analyse.....	7
Navneord	7
Udsagnsord.....	8
Teori.....	8
UP	8
GRASP	9
Metode	9
Hovedafsnit.....	10
Domænenomodel	10

CRC Kort.....	11
Use case diagram.....	14
Use cases	14
Systems sekvensdiagrammer	24
SSD for use case 1 og 2:	24
SSD for use case 8:.....	25
SSD for use case 3:.....	26
BCE-model	27
Design klassediagram	28
Design sekvensdiagram	29
Kodedel.....	30
Kode for landPåFelt:	30
Kode for handel af ejendomme:.....	32
Kode for Chancekort:.....	34
Test	35
Traceability matrix.....	35
Formelle test cases:	36
JUnit-test	63
Konfiguration	65
Guide til importering af git repository:	65
Konklusion	67
Overordnet konklusion.....	67
Produktorienteret konklusion	67
Forslag til videre arbejde og forbedringer.....	68
Litteraturliste	68
Bilag	69

Timeregnskab

CDIO Final							
Timeregnskab	Uge 01						
Dato	Deltager	Design	Impl.	Test	Dok.	Andet	I Alt
02-03-04-05-06/01/2017	Bertram	7	12		4		23
02-03-04-05-06/01/2017	Christopher	10	7		7		24
02-03-04-05-06/01/2017	Johnny	9	6		9		24
02-03-04-05-06/01/2017	Jonathan	9	8		7		24
02-03-04-05-06/01/2017	Thomas	10	6		8		24
	Sum	45	39		35		119

CDIO Final							
Timeregnskab	Uge 02						
Dato	Deltager	Design	Impl.	Test	Dok.	Andet	I Alt
09-10-11-12-13/01/2017	Bertram	5	11	5	8		29
09-10-11-12-13/01/2017	Christopher	7	4	6	9		28
09-10-11-12-13/01/2017	Johnny	5	4	6	10		25

09-10-11-12-13/01/2017	Jonathan	7	5	5	9		26
09-10-11-12-13/01/2017	Thomas	5	4	5	11		25
	Sum	29	28	27	47		131

Indledning

Emne

At tage udgangspunkt i Matador og videreudvikle vores brætspil fra CDIO-3.

Formål

Formålet med projektet er at inddrage hvad vi har lært fra de tidligere CDIO opgaver til at udvikle et Matador spil. Spillet kan spilles af tre til seks spillere, som bevæger sig rundt på et spillebræt bestående af 40 felter. Rapporten indeholder en analyserende del bestående af navne- og udsagnsord analyse, diagrammer og use cases, hvilket hjælper os med at danne et bedre overblik over hvordan vores proces kommer til at se ud. Udover dette hjælper det os også meget med at nedbryde vores problemstillinger og diagrammer til funktionel kode.

Kravspecifikationer

Functional:

1. Systemet skal kunne simulere et slag med to 6-sidede terninger
2. Systemet skal kunne flytte en brugers brik til det rigtige felt alt efter summen af terningslaget
3. Systemet skal kunne holde styr på en spillers pengebeholdning
4. Systemet skal sørge for at ændre brugerens pengebeholdning alt efter hvilket felt han/hun lander på og overføre penge til en anden spiller hvis feltet allerede er ejet.
5. Systemet skal vise en tekst, der omhandler det felt man holder musen over
6. Systemet skal give spilleren mulighed for at vælge om han/hun vil købe et felt, som han/hun er landet på.
7. Systemet skal have felter med unikke egenskaber og navne. Se bilag #1
8. Systemet skal ved start tildele alle spillere 30.000 kr.

9. Systemet skal placere samtlige spillere ved 'start' feltet, når spillet startes
10. Systemet skal give spillerne 4000 kr hver gang de har passeret eller landet på start feltet.
11. Systemet skal give en spiller et ekstra kast, hvis spilleren slår to éns.
12. Systemet skal sende en spiller i fængsel, hvis spilleren slår to éns tre gange i træk.
13. Systemet skal lade en fængslet spiller slå et enkelt kast med terningerne. I tilfældet af to éns er spilleren sluppet fri og spilleren tildeles et ekstra kast.
14. Systemet skal trække 1000 kr. fra en fængslet spiller i tilfælde af, at spilleren ikke formår at slå to éns i tredje forsøg på at komme ud af fængslet og derefter skal systemet give spilleren et kast mere.
15. Systemet skal give en spiller, som er i fængsel, mulighed for at betale en bøde på 1000 kr. for at blive sat fri. Dette skal ske inden spilleren kaster med terningerne.
16. Systemet skal lade spillere trække "prøv lykken kort" hvis man lander på "prøv lykken" felt.
17. Systemet skal lade lejen fordoble i tilfælde af spilleren ejer samtlige ens grunde (farve).
18. Systemet skal kunne lade en spiller sælge huse/hoteller til ejede skøder for halvdelen af den oprindelige pris.
19. Systemet skal kunne lade en spiller købe huse til ejede skøder såfremt spilleren ejer samtlige skøder af den bestemte farve.
20. Systemet skal kunne lade en spiller købe hoteller til ejede skøder såfremt spilleren ejer samtlige skøder af den bestemte farve og har maks antal huse på hver skøde af samme farve
21. Systemet skal kun lade en spiller købe et hus eller hotel på en grund hvis han har mindst lige så mange huse på de andre grunde i samme gruppe.
22. Systemet skal trække 10% af en spillers samlede værdier eller 4000 kr fra en spillers pengebeholdning, hvis spilleren lander på indkomstskat-feltet.
23. Systemet skal kunne erklære en spiller, ude af spillet, hvis spilleren skylder mere end hvad spilleren ejer.
24. Spillet slutter, når der kun er en enkelt spiller, som ikke er gået fallit.

Usability:

25. Systemet skal være et spil mellem tre til seks brugere på samme computer.

Reliability:

26. Systemet skal have lige stor chance for at slå 1-6 med terningerne, som med en fysisk terning.
27. Systemet skal kunne fungere fejlfrit i minimum én time.

Performance:

- 28. Systemet skal kunne bruges på maskinerne i data-barerne på DTU.
- 29. Systemet skal kunne vise resultat af kastet inden for 0.6 sekunder.

Implementation:

- 30. Systemet skal udvikles i Eclipse.
- 31. Systemet skal bruge UTF-8 som tegnsæt.
- 32. Systemet skal kodes i Java.

Navneord- og udsagnsords analyse

Ud fra vores navneord- og udsagnsords analyse har vi fået lavet diverse klasser og metoder. Vores navneords analyse bruges til at finde kandidater til klasser og attributter til vores program. Dette gøres ved at tjekke de mest relevante navneord ud i de funktionelle krav, og så lave en liste. F.eks. er klassen spiller lavet ud fra navneordet "brik". Udsagnsord Analysen bruges til at finde kandidater til metoder i klasserne. Et eksempel på dette er udsagnsordet "simulere", som man kan lave metoden "RulTerning" ud fra.

Navneord

Terninger (to seks-sidede)

Felt

Pengebeholdning

(Brugers) Brik

Tekst (der omhandler feltet)

Mulighed (for at købe felt)

(Unikke) Egenskaber og navne.

30.000 kr.

(Højeste første) Slag

Skøder

Fængsel

Indkomstskat-feltet

(En) Leje

Udsagnsord

Simulere (et slag)

Flytte (en spillers brik)

Holde styr på (pengebeholdning)

Ændre (brugerens pengebeholdning)

Overføre (penge)

Udskrive (tekst som) omhandler (et felt)

Sætte (en skøde på auktion)

Give (spilleren 4000 kroner ved start)

Betale (bøde på 1000 kr for at komme ud af fængsel)

Sælge (huse og hoteller)

Erklære (spiller fallit)

Pantsætte (sine skøder)

Teori

UP

Vi har benyttet Unified Process til at dele projektet på op i flere overskuelige faser. Inception Fasen har ikke fyldt meget, da spil-idéen allerede er givet til os. Ud fra spil-idéen (Matador) har vi skitseret kravspecifikationer, hvor vi har udarbejdet og finpudset det i løbet af elaborations-fasen. I denne fase har vi også udarbejdet diverse diagrammer og modeller, som beskriver hvordan spilsystemet i Matador er opbygget. I konstruktionsfasen har vi kodet spillet ud fra designet i elaborations-fasen. Til sidst har vi, i transitionsfasen, implementeret vores Matador spil på et af DTU's databar og afprøvet spillet.

GRASP

Creator:

I vores spil system er det klassen SpilController, som er vores creator. Den opretter Matador spillet og henter metoder fra de andre klasser. Det her at der bliver oprette objekter af de andre klasser og det her selve "spillet" foregår.

Controller:

Vores controllers er følgende: BankController, LandPåFeltController, PrøvlykkenController og SpilController. Control klasserne har ansvar diverse system handlinger i spillet. De skal kunne håndtere de forskellige use cases korrekt, som der kan opstå i spillet.

Information Expert:

I vores spil system er der flere der kan være Informations Expert. Vi har valgt vores til at være FeltBeskrivelser. Det har vi gjort fordi det er den klasse som indeholder navne, værdier, leje og farve for alle felterne, hvilket er essentielt at vide i vores spil system. Der er flere kandidater til at være information expert som f. eks. LykkeBeskrivelser, Spiller og Konto. De indeholder hver især information, men det er i mindre omfang i forhold til FeltBeskrivelser.

Low Coupling:

Vi har prøvet så godt vi kunne med at overholde low coupling ved at sørge for at så få klasser snakker med hinanden som muligt. Vi har ikke helt overholdt det da der er en del klasser der snakket med SpillerListe. LandPåFelt-, Bank-, Spil- og PrøvLykkenControlller snakker også en del indbyrdes. Grunden til dette er at vi ikke har kunne se en bedre løsning på problemet.

High Cohesion:

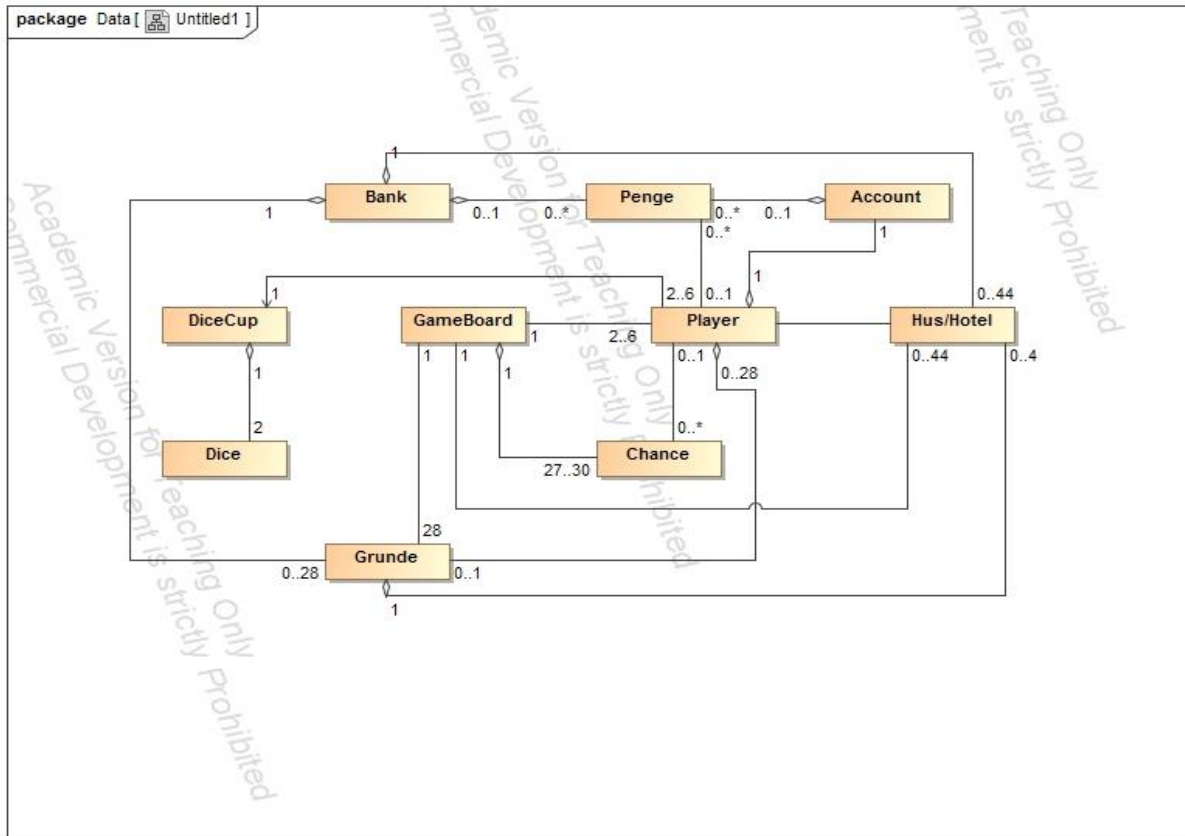
Vi har også her prøvet så godt som vi kunne at overholde high cohesion. Vi har altså prøvet at holde klasserne så "fokuserede", som muligt. D.v.s at vi har prøvet at have så få funktioner i hver klasse, som muligt så hver klasse har så lidt at lave som muligt. Klasser som f. eks BankControlleren har mange metoder, og man kunne måske have fordelt dem bedre ud. F. eks så har BankControlleren metoder der omhandler tapperier og rederier. Udover dette har vi overholdt high cohesion, da vi syntes at vores klasser har de funktioner, som de har brug for og kun dem.

Metode

For at udvikle vores Matador spil har vi inddraget metoder fra vores tidligere CDIO opgaver. Vi har brugt samme process til at nedbryde spillets design og krav til kode. I designprocessen startede vi ud med at lave diverse diagrammer til at få overblik og strukturere arkitekturen i spillet. Derefter har vi kodet spillet ud fra designprocessen.

Hovedafsnit

Domænemodel



Her ses vores domænemodel. I denne model visualiseres, hvordan de virkelighedsnære elementer i vores program hænger sammen. Der vises også kvantiteten af de forskellige elementer, og hvorvidt de aggregerer eller ej. F.eks. er der mellem tre og seks spillere som interagerer med DiceCup klassen. Ligeledes er der kun en Dicecup, som så interagerer med de forskellige players på skift alt efter hvis tur det er. DiceCup aggregerer dice klassen, da den bruger elementer fra dice klassen. Derfor hænger de meget sammen. Igen ses der også, at der er 2 dice objekter som interagerer med en Dicecup.

Vi har mellem GameBoard og Chance skrevet at antal af Chance er 27..30, men vi har ændret dette tal til 21. Da vi i det færdige spil kun har 21 prøv lykken kort.

CRC Kort

I vores CRC-Kort opstiller vi de forskellige klasser i forhold til hvilke ansvar de skal have (dette bliver deres metoder) samt hvem de samarbejder med i systemet. Dette bliver de klasser som den respektive klasse interagerer med. Vi har i dette eksempel taget vores controllere til at lave CRC kort for i stedet for alle klasserne, da controllerne er mere omfattende.

Bank controller:

Responsibilities	Collaborators
-KøbHus -Sælghus -KøbEjendom -FjernSpiller	-SpillerListe -SpilleBræt -Grund -Felt -GUIController

PrøvLykken Controller:

Responsibilities	Collaborators
-BlandKort -TrækKort -PrøvLykken	-Spiller -LykkeBeskrivelser -SpilController -GUIController -SpillerListe

LandPåFelt Controller:

Responsibilities	Collaborators
-LandPåFelt	-BankController -GUIController -Spiller -Felt -Ejendom -Grund

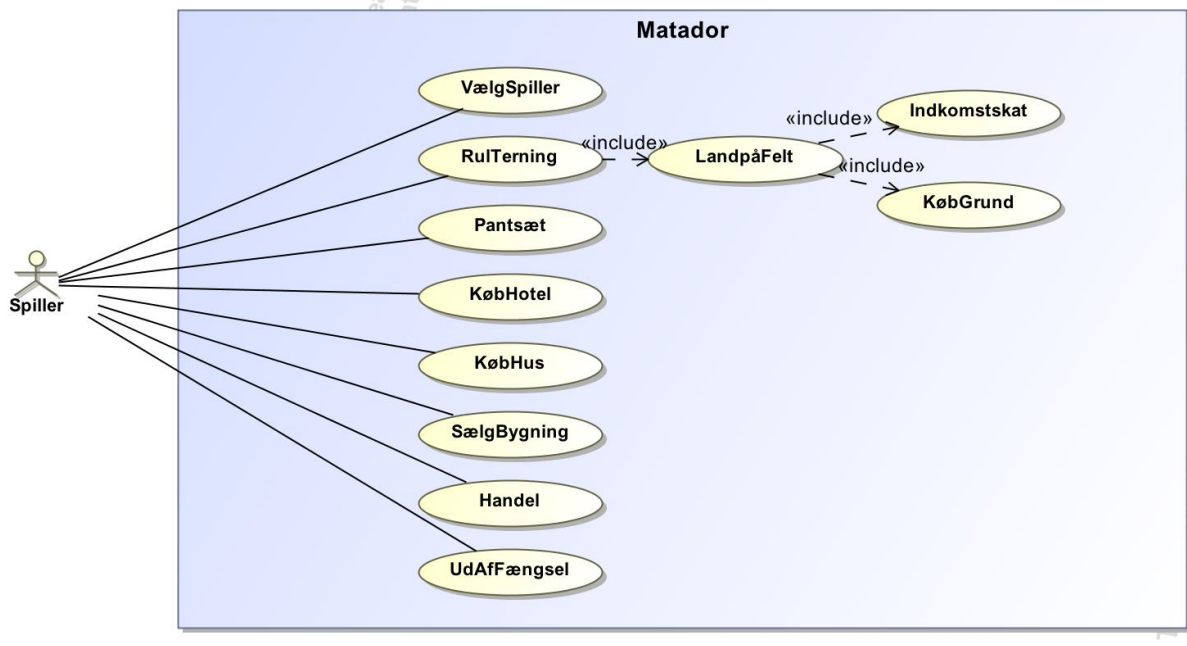
Spil Controller:

Responsibilities	Collaborators
-StartGame -TilføjSpillere	-GUIController -LandPåFeltController -BankController -SpillerListe -Raflebæger -Main

GUI Controller:

Responsibilities	Collaborators
<ul style="list-style-type: none">-CreateBoard-TilføjSpiller-fjernBil-flytBil-visPengeBeholdning-visBesked-SpørgSandtFalsk-SpørgString-vælgString-fjernEjer-visprøvLykkenKort	<ul style="list-style-type: none">-SpilController-BankController-LandPåFeltController-PrøvLykkenController

Use case diagram



I diagrammet ovenover ses vores use case diagram. Altså kan man se hvad en spiller kan i vores spil. F. eks kan en spiller vælge antal spillere og deres navne, dette er beskrevet med use casen VælgSpiller. Da dette diagram blev lavet i starten af produktionen er der en af use casene, som ikke er blevet realiseret. Dette er Pantsæt. Denne use case har vi ikke implementeret da vi ikke havde tid til det. Vi valgte at prioritere Handel, da vi mente at det var en vigtigere del af spillet og at spillet ville blive for kedeligt uden en sådan funktion. Nedenfor ses “fully dressed” use case beskrivelser af alle use casene, selv Pantsæt.

Use cases

Use Case: RollTheDice
ID: 1
Brief Description: The player rolls two dice, and moves accordingly to the sum of the two rolled values.
Primary actor: The current player.

Preconditions: It is the current player's turn.

Main flow:

The player rolls the dice

the player moves the number of fields ahead equivalent to the sum of the values of the dice.

Postconditions: The player has ended his turn.

Secondary and tertiary flows:

Two equal values:

The player rolls two equal values

The player moves the number of fields ahead equivalent to the sum of dice values

The player gets another turn

The same procedure is repeated.

Two equal values thrice in a row:

The player rolls two equal values thrice

The player ends up in jail

Use Case: Indkomstskat

Brief Description: The player lands on a field where he/she either has to pay ten percent of their fortune or 4000 in cash. alternately, they land on a field where they have to pay 2000, with no

option to pay ten percent.
Primary actors: The player and the computer.
Precondition: The current player has rolled the dice
<p>Main flow:</p> <p>The player rolls the dice</p> <p>The player lands on a tax field</p> <p>The player chooses to either pay ten percent of their fortune or 4000 KR</p> <p>The system subtracts the value from the player's account</p> <p>Turn is ended</p>
Postcondition: The player's turn has ended, and he has a certain amount less of money than before

Use Case: KøbGrund
ID: 2
Brief description: When a player lands on a property not already owned he/she has the choice of buying that property
<p>Primary actors:</p> <p>A player</p> <p>The bank</p>

Preconditions: The player must land on a property not already owned by other players and further have the necessary amount of money
Main flow: The player rolls the dice The player lands on a vacant property The player chooses to either buy the property or let it go on auction If the player has the necessary money and chooses to buy the property the player must pay the bank the correct amount, according to the title deed The player receives the title deed for the property
Postconditions: The player successfully paid the bank and received the title deed

Use Case: KøbHus
ID: 3
Brief Description: It's the player's turn and he wants to buy a house on one of his owned properties.
Primary actor: Current player
Preconditions: The player owns the property on which he wants to buy a house and the rest of the properties with the same color, and has sufficient funds
Main flow: The player chooses to buy a house on the property he owns

The system gives the player a house on the property and subtracts funds from the player equivalent to a fifth of the price for a hotel.
Postconditions: The player now owns a house on the property on which he purchased a house.

Use Case: KøbHotel
ID: 4
Brief description: Its the players turn and the player wants to buy a hotel on one of his properties
Primary actors: A player
Preconditions: Have 4 houses on all of the same type ground and sufficient funds
Main flow: The player chooses to buy a hotel System gives player a hotel and takes the 4 houses back.
Postconditions: Owns a hotel and loses 4 houses on the ground. Correct amount of money is taken from the player.

Use Case: Pantsætning
ID: 5
Brief description: A Player can choose(On his/her turn only) to mortgage a property in order to receive a fixed amount of money from the bank, depending on the worth of the property, that is

being mortgaged.
<p>Primary actors:</p> <p>A player</p> <p>The bank</p>
Preconditions: The Player owns a property and it is the player's turn.
<p>Main flow:</p> <p>The Player chooses to mortgage his property.</p> <p>The system gives money to the player and mortgages the property.</p>
Postconditions: The system successfully grants the player the correct amount of money, and mortgages the player's property. The system also ensures that if another player lands on the mortgaged property, that player doesn't pay rent.
Comments: This use case is not an option, since we decided not to include it, due to lack of time.

Use Case: VælgSpiller
ID: 6
Brief description: When the game starts the system asks how many players are playing and what their names are. The players then type in a number and their names.
<p>Primary actors:</p> <p>All the players</p>

Preconditions: A number of players must be decided and their names known.
<p>Main flow:</p> <p>The system starts and ask the players how many are playing.</p> <p>The players type in a number.</p> <p>The system ask for the name of each player.</p> <p>The players types in their names.</p>
Postconditions: The system creates the correct number of players and gives them the correct names.

Use Case: SælgBygning
ID: 7
Brief description: At the start of a player's turn he/she has the possibility of selling currently built buildings for half of the original price
<p>Primary actors:</p> <p>A player</p> <p>The bank</p>
Preconditions: It's the player's turn and the player currently owns one or more houses and or a hotel
Main flow:

<p>It's the player's turn</p> <p>The player sells a house and or a hotel to the bank</p> <p>The player receives half of the money he/she originally paid for the buildings</p> <p>The bank receives the house and or hotel</p>
<p>Postconditions: The player received the correct amount of money and the bank received the buildings that are now for sale again</p>

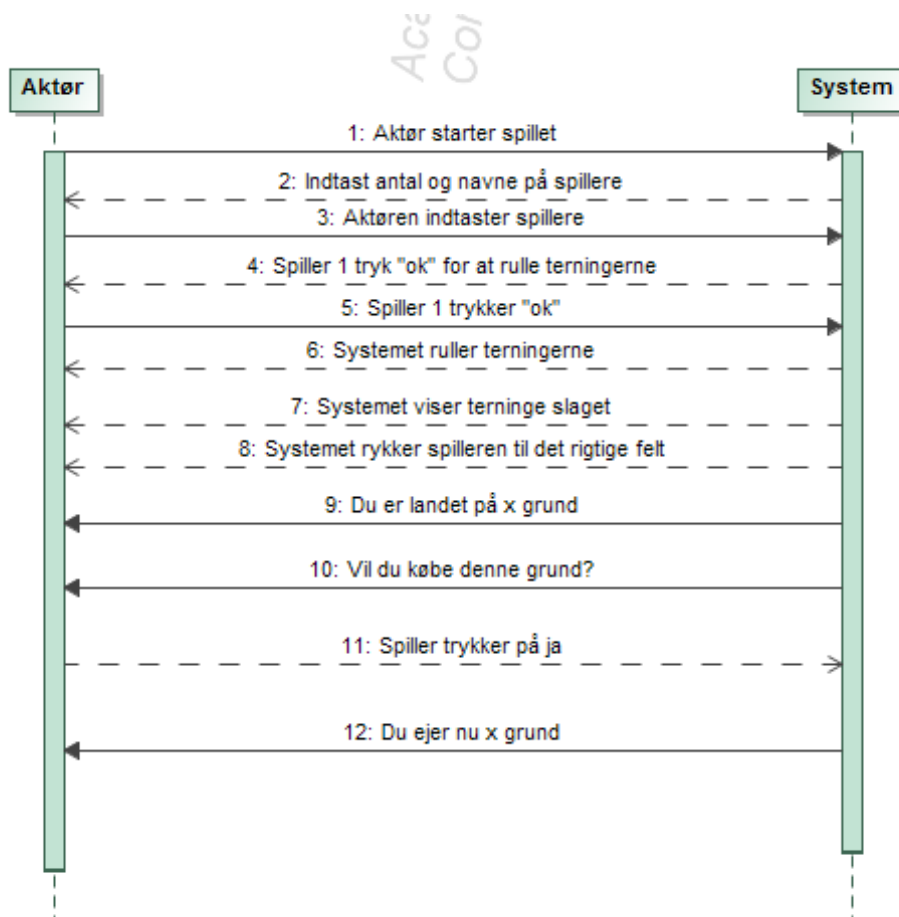
Use Case: UdafFængsel
ID: 8
Brief Description: The player gets out of jail by one of three possible ways other than the default.
Primary actor: The player
Pre-condition: The player is in jail, but has three ways to get out of jail. If the player uses the lucky card, they got it in an earlier round.
<p>Main flow:</p> <p>Use of "Prøv lykken" card:</p> <p>The player gets in jail</p> <p>The player uses "prøv lykken" earned in a previous round</p>

<p>Two equal values:</p> <p>The player is in jail.</p> <p>The player rolls the dice one to three times according to how many rounds they have been in jail without rolling two of the same values. The rolled value is two of the same.</p> <p>The player is out of jail</p> <p>The players moves according to his roll</p> <p>The player has finished his turn</p>
<p>Paying 1000:</p> <p>The player is in jail</p> <p>The player rolls thrice over three turns without getting two of the same values in one roll.</p> <p>The player pays 1000 to get out of jail</p> <p>The player is out of jail</p> <p>The player has finished his/her turn.</p>
<p>Postconditions: The player is out of jail.</p>
<p>Comments: Getting out of jail with a “Prøv lykken kort” is not possible since we decided not to implement this function due to lack of time.</p>

Use Case: Handel
ID: 9
Brief Description: When it is a player's turn he/she can choose to trade, properties for money, with another player. The price for the property is to be discussed face to face.
Primary actor: Player 1 and Player 2
Precondition: Player 1 owns a property and it's player 2's turn.
<p>Main flow:</p> <p>It's player 2's turn</p> <p>Player 2 chooses the trade option from a dropdown menu.</p> <p>The system asks player 1 how much he wants for the property.</p> <p>Player 1 types in the amount he wants from it.</p> <p>Player 2 accepts the amount and pays player 1.</p> <p>The system sets player 2 as owner of the property.</p>
Postconditions: Player 2 is now the owner of the property and the system successfully transferred the correct amount of money from player 2 to player 1.

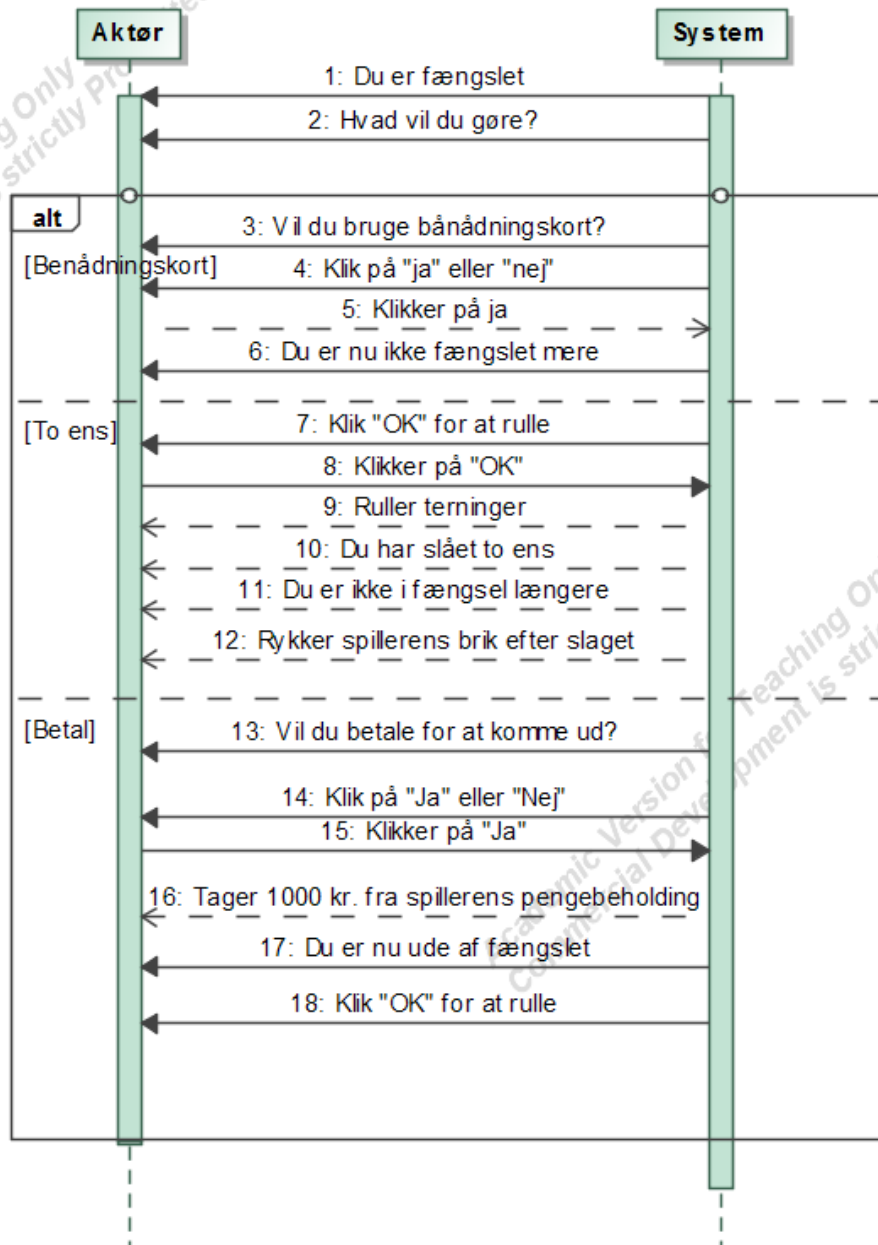
Systems sekvensdiagrammer

SSD for use case 1 og 2:



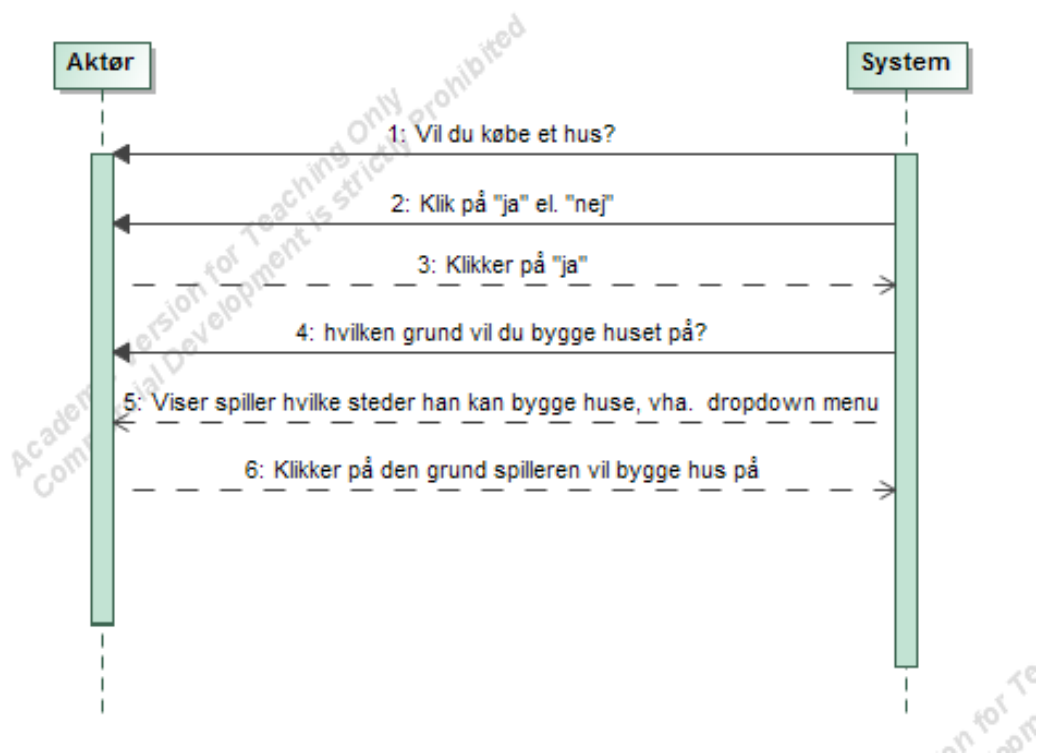
I vores SSD ser vi hvordan en bruger interagerer med systemet, når de ruller en terning. De ikke-stiplede linjer viser, hvilke handlinger, som foretages af aktøren, og de stiplede linjer viser så hvilke aktioner/svar, som systemet sender tilbage til aktøren. Dette modelleres for et helt use cases varighed.

SSD for use case 8:



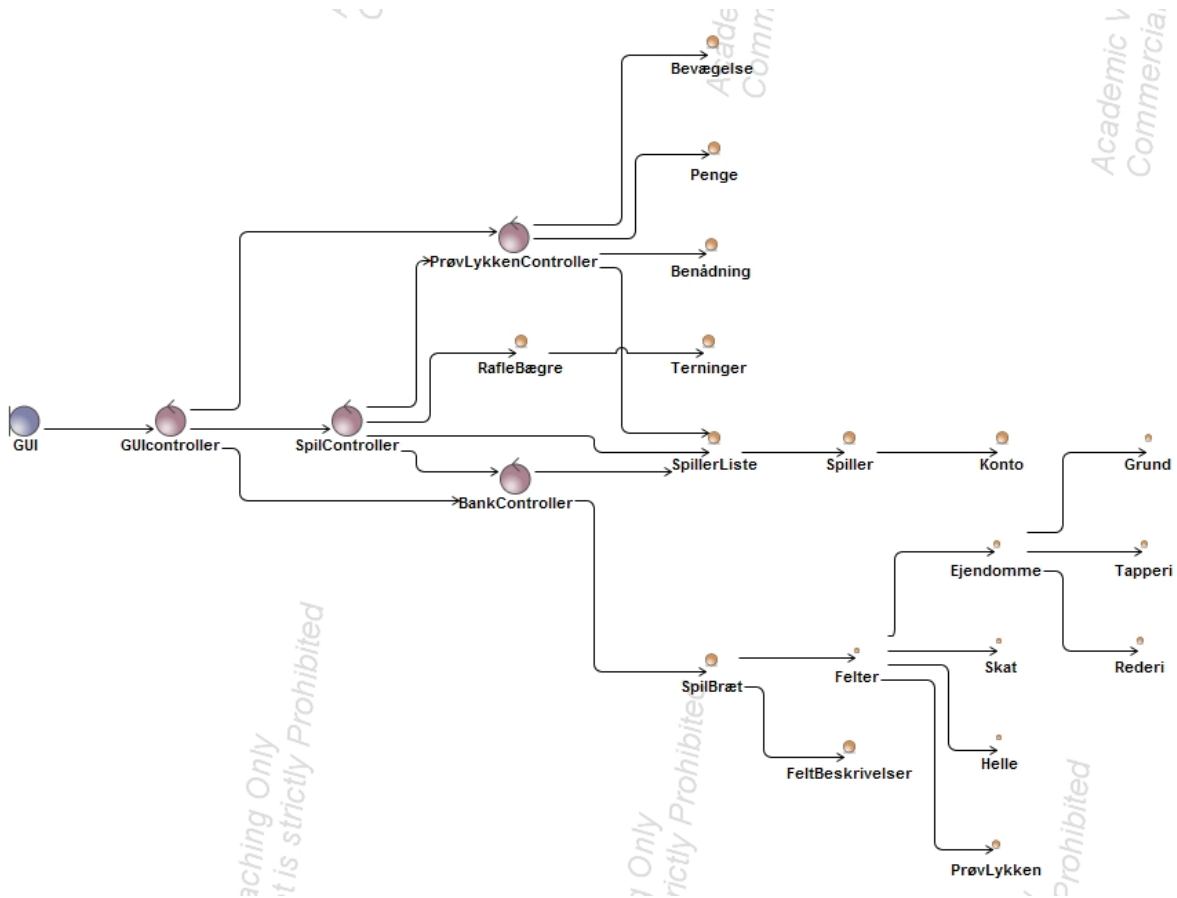
Systemsekvensdiagram for use case 8. I dette Use Case har brugeren op til flere muligheder for at komme ud af fængsel, og de forskellige scenarier opstilles i forskellige loops, som er uafhængige af hinanden. Vi har ikke nået at implementere benådning kort fordi vi har prioriteret andre ting.

SSD for use case 3:



I dette system sekvens diagram ses det hvordan GUI'en og brugeren interagerer med hinanden i brugsscenariet for at købe et hus i matadorspillet. I dette Use Case bruger GUI'en drop down menuen hvor der er mulighed for at vælge ja eller nej.

BCE-model

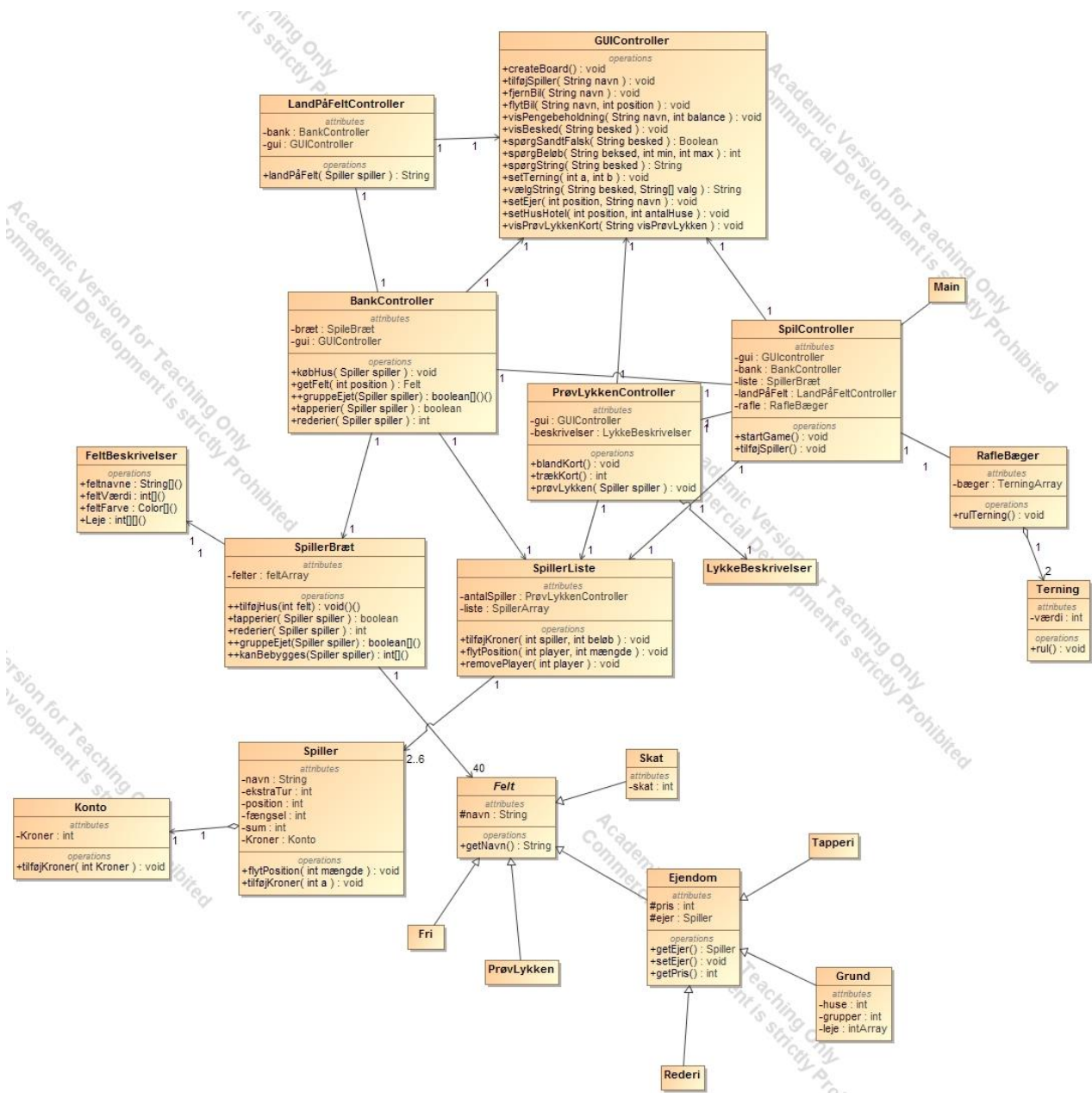


I vores BCE model ser vi, hvordan de forskellige klasser interagerer med hinanden, og hvilke funktioner de har i det overordnede system. De er i BCE delt op i Boundary, Control og Entity. Vores Boundary er sat til at være vores GUI, da det er den primære klasse i vores system som brugeren interagerer med. Vores GUI controller, samt de andre controller-klasser for visse dele af spillet, er sat som vores controllere i BCE modellen, da det er dem som bruger information fra vores entities. Vores GUI-controller er i bredere forstand den overordnede controller end de andre, da det er den som bruger de forskellige klasser (og controllere) til at interagere med vores GUI. Vores entities her er de forskellige objekter, som controllerne bruger til at interagere med systemets GUI. Nogle entities, såsom felt, nedarver forskellige typer af objektet, f.eks. ejendomme, skat, helle eller PrøvLykken.

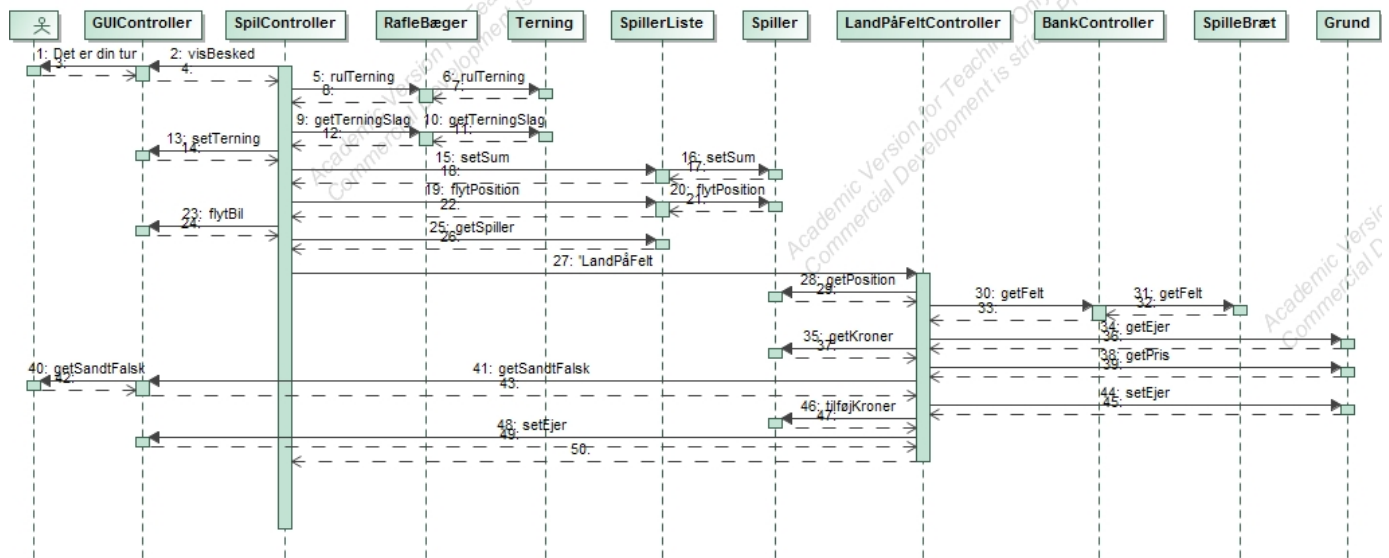
Klasserne Bevægelse, Penge og Benådning er blevet samlet til klassen LykkeBeskrivelser, som er en entity. Helle klassen er blevet omdøbt til Fri.

Design klassediagram

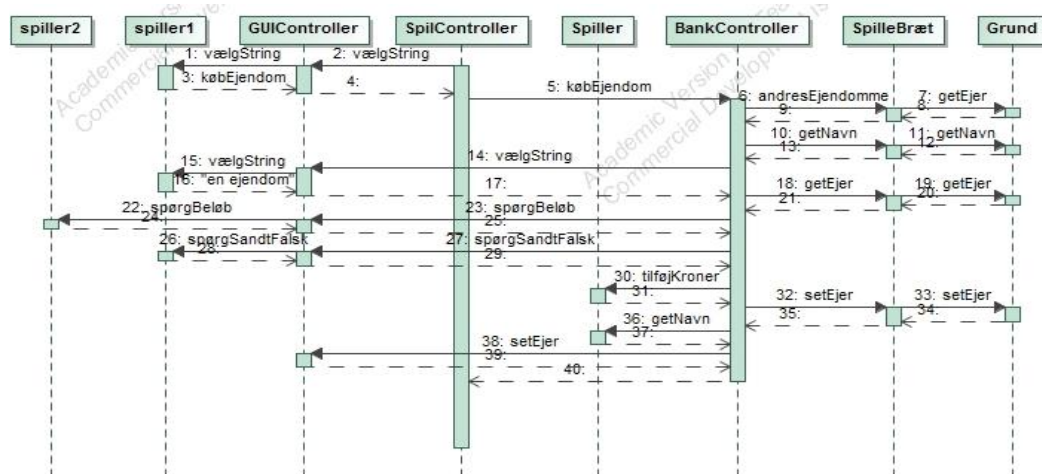
I vores design klassediagram ser vi sammenhængen mellem vores klasser. Der vises i disse klasser de forskellige attributter, metoder samt deres tilgængelighed (hvorvidt de er public, private eller protected). Med diagrammet får man samtidigt et bedre billede af, hvilke klasser (samt tilhørende operationer og attributter) der nedarves, og hvilke kontrollere de interagerer med. Ved at kigge på diagrammet kan man se, at den mest centrale controller er GUIController, da det er den som interagerer med flest klasser. Diagrammet viser også, hvilke klasser der aggregeres med andre. Heriblandt er der Raflebæger klassen og Terning klassen, da de bruger informationer fra hinanden til at fungere optimalt.



Design sekvensdiagram



I dette diagram ses hvordan de forskellige klasser interagerer med hinanden under en spillers runde. De stiplede linjer indikerer, når en klasse sender et respons til den klasse, som havde sendt en metode til en klasse. Rækkefølgen af metode kaldene er angivet kronologisk fra toppen af og nedad med nummerering af rækkefølgen. Ud fra diagrammet fremgår det tydeligt, at SpilController er den mest centrale klasse sammen med LandPåFeltController.



Dette design sekvens diagram viser hvad der sker når en spiller gerne vil købe en ejendom af en anden spiller. Det starter med at man vælger en ejendom på en liste over andre spilleres ejendomme, så spørger den ejeren af den valgte ejendom om hvor meget han skal have for ejendommen, og så spørger den spilleren om han vil betale prisen. Hvis han betaler prisen bliver han sat som ejeren af grunden.

Kodedel

Kode for landPåFelt:

```
public String landPåFelt(Spiller spiller) {
    int position = spiller.getPosition();
    Felt felt = bank.getFelt(position);
    if (felt.getClass().getSimpleName().equals("Fri")) {
        switch (spiller.getPosition()) {
            case 0:
                gui.visBesked("Du landede på Start og modtager 4000 kr!");
                break;
            case 10:
                gui.visBesked("Du er på besøg i fængslet.");
                break;
            case 20:
                gui.visBesked("Gratis Parkering.");
                break;
            case 30:
                gui.visBesked("Politiet fangede dem, de fængsles.");
                spiller.setPosition(10);
                gui.flytBil(spiller.getNavn(), spiller.getPosition());
                spiller.setFængsel(1);
                break;
        }
    } else if (felt.getClass().getSimpleName().equals("Skat")) {
        if (spiller.getPosition() == 4) {
            int tiProcent;
            tiProcent = spiller.getKroner() / 10;
            String[] valg = {"4000 kr.", tiProcent + " kr."};
            if (gui.vælgString("Hvad vil du helst betale?", valg).equals("4000 kr. ")) {
                spiller.tilføjKroner(-4000);
            } else {
                spiller.tilføjKroner(-tiProcent);
            }
        } else {
            gui.visBesked("Du skal betale 2000 kr. i skat");
            spiller.tilføjKroner(-2000);
        }
    } else if (felt.getClass().getSimpleName().equals("PrøvLykken")) {
        gui.visBesked("Du er landet på et prøv lykken felt, prøv igen imorgen");
        prøvLykken.prøvLykken(spiller);
    } else {
        Ejendom felt1 = (Ejendom) felt;
        if (felt1.getEjer() == null && spiller.getKroner() > felt1.getPris()) {
            // Lets the player buy the field if there is no owner and the
            // player has enough coins
            if (gui.spørgSandtFalsk("Vil du købe " + felt1.getNavn() + " for " + felt1.getPris() + " kr?")) {
                felt1.setEjer(spiller);
                spiller.tilføjKroner(-felt1.getPris());
                gui.setEjer(position, spiller.getNavn());
            }
        }
    }
}
```

```

    }
} else if (felt1.getEjer() == null) {
    gui.visBesked("Du har ikke nok penge til at købe denne ejendom.");
} else if (felt1.getEjer().equals(spiller)) {
    gui.visBesked("Du har landet på din egen ejendom.");
} else {
    if (felt.getClass().getSimpleName().equals("Tapperi")) {
        int a = 0;
        if(bank.tapperier(spiller)){
            a = spiller.getSum()*200;
        } else {
            a = spiller.getSum()*100;
        }
        felt1.getEjer().tilføjKroner(a);
        spiller.tilføjKroner(-a);
    } else if (felt.getClass().getSimpleName().equals("Rederi")) {
        int rederier = bank.rederier(felt1.getEjer());
        int a = 250 * (int) Math.pow(2, rederier);
        felt1.getEjer().tilføjKroner(a);
        spiller.tilføjKroner(-a);
    } else {
        Grund felt2 = (Grund) felt1;
        if(bank.grupperEjet(spiller)[position/5] && felt2.getHuse() == 0){
            int a = felt2.getLeje()*2;
            felt2.getEjer().tilføjKroner(a);
            spiller.tilføjKroner(-a);
        } else {
            int a = felt2.getLeje();
            felt2.getEjer().tilføjKroner(a);
            spiller.tilføjKroner(-a);
        }
    }
}
}
}

```

Ovenover ses en metode fra klassen LandPåFeltController. Denne metode står for hvad der sker når en spiller lander på et felt på brættet. Først oprettes der en int kaldet position, denne int er altså den position spilleren har på spillebrættet, som et heltal. Derefter har vi en if-sætning, som deler alle felterne op i forskellige kategorier: Fri, Skat og Prøvlykken. Disse kategorier repræsenterer de forskellige former for felter, som kan findes på et matador spillebræt. Systemet spørger herefter et objekt af Felt-klassen felt om hvilket felt der er på den position spilleren har. Hvis feltet er et fristed og tilhører klassen Fri, køres en switch case, med de forskellige frie felter og ved hver case sendes en besked til GUI'en. Hvis feltet tilhører et skatfelt, køres en if-sætning, som deler de to skatfelter op, da skat felterne ikke er ens. Hvis feltet, som spilleren er landet på tilhører klassen PrøvLykken, udføres et metodekald fra klassen PrøvLykkenController der sørger for at spilleren trækker et prøvlykken kort. Hvis felter spilleren lander på ikke er en af ovenstående kategorier, så må det altså tilhøre en klasse, som kan ejes, altså en grund. Nu køres så en if-lykke, hvor if-statementet tjekker om grunden allerede er ejet og om spilleren har råd til grunden. Hvis begge betingelserne er opfyldt så sendes en besked til GUI'en, om at den skal spørge spilleren om han/hun vil købe grunden. Hvis grunden ikke er ejet, men spilleren ikke har råd, sendes en besked om at spilleren ikke har penge nok til at købe grunden. Hvis feltet allerede ejet, men af spilleren, som er landet på feltet, sendes også en besked til GUI'en, men en besked om at spilleren allerede ejer grunden. Hvis intet af ovenstående passer, så må feltet altså være ejet af en anden person og derfor skal spilleren, som landede

på feltet, betale leje til ejeren. Her opstilles en if-lykke, som sørger for at trække det rigtige beløb fra spilleren alt efter hvilken type felt spilleren er landet på. Hvis det er et tapperi trækkes enten 100 gange tal-slaget eller 200 gange tal-slaget. Hvis det er et rederi trækkes 2 opløftet i antal rederier, som ejeren af rederiet har. Hvis det ikke er et rederi eller et tapperi, må det være en helt almindelig grund og derfor tjekkes det om ejeren af grunden har alle de andre af samme farve og om der er huse. Information om lejen har ejendommen fået fra FeltBeskrivelser klassen.

Alt dette sker i denne metode. Så hvorfor lige highlighte den? Jo fordi den spiller en vital rolle i forhold til spillet og det er derfor vigtigt at den virker korrekt. landPåFelt metoden er ret lang og kan virke uoverskuelig, derfor burde vi nok til en anden gang have delt den op i mindre bidder. F. eks en metode til hver type af felt. Grunden til at vi valgte at lave det i en controller var at i den tidligere CDIO opgave havde vi en landPåFelt metode inde i hver felt klasse, og det betød at de ikke rigtigt kunne tale med spilleren og de andre felter uden at man mistede den lave coupling, derfor valgte vi at prøve denne struktur. Vi er ikke rigtigt tilfreds med nogen af de to metoder, fordi de begge to giver lavere cohesion og højere coupling på deres egen måde, men vi kan ikke umiddelbart se en bedre måde at gøre det på.

Kode for handel af ejendomme:

```
public void købEjendom(Spiller spiller){
    int[] andresEjendomme = bræt.andresEjendomme(spiller);
    if (andresEjendomme.length == 0) {
        gui.visBesked("Der er ikke nogle ejendomme du kan købe.");
    } else {
        String[] kanKøbe = new String[andresEjendomme.length];
        for (int i = 0; i < andresEjendomme.length; i++) {
            kanKøbe[i] = bræt.getNavn(andresEjendomme[i]);
        }
        String valg = gui.vælgString("Hvilken ejendom vil du prøve at købe?", kanKøbe);
        int a = 1;
        for (int i = 0; i < kanKøbe.length; i++){
            if(valg.equals(kanKøbe[i])){
                a = i;
            }
        }
        int pris = gui.spørgBeløb(bræt.getEjer(andresEjendomme[a])+ " , hvor meget skal du have for " + valg + "?", 0, 100000);

        if (spiller.getKroner() < pris){
            gui.visBesked("Det har " + spiller + " ikke råd til at betale.");
        } else {
            if (gui.spørgSandtFalsk(spiller + " , vil du betale " + pris + " for " + valg + "?")){
                bræt.getEjer(andresEjendomme[a]).tilføjKroner(pris);
                spiller.tilføjKroner(-pris);
                bræt.setEjer(andresEjendomme[a], spiller);
                gui.setEjer(andresEjendomme[a], spiller.getNavn());
            }
        }
    }
}
```

Ovenover ses metoden, som sørger for use case 9, handel. Først oprettes et int array andresEjendomme, som henter et array fra en metode i Spillebræt klassen, som indeholder alle de grunde hvor der er en ejer,

som ikke er den spiller hvis tur det er. Der påbegyndes nu en if-lykke, som tjekker hvor mange og hvilke ejendomme der kan købes.

Hvis der ikke er nogle grunde, som kan købes vises der en besked hvori der står netop dette.

Ellers laves der et String array kanKøbe, som har længden af det andresEjendomme. Der laves så en for-lykke der kører alle de mulige ejendomme igennem, og tilføjer deres navne til String arrayet kanKøbe.

Derefter sendes der et valg til spilleren gennem GUI'en, hvor spilleren får en drop down menu med alle de valgmuligheder han har, fra kanKøbe arrayet. Vi har så en for-lykke der kører alle mulighederne igennem og tjekker hvilken spilleren har valgt. Nu får vi så GUI'en til at spørge ejeren af ejendommen spilleren prøver at købe, hvad han skal have for sin ejendom og han bedes indtaste et tal mellem 0-100000. Systemet tjekker nu om spilleren har råd til at betale det ejeren vil have for sin ejendom.

Hvis ikke sendes en besked om at spilleren ikke har råd.

Hvis spilleren derimod har råd til at købe ejendommen bedes han klikke ja til tilbuddet i GUI'en. Når spilleren har trykket ja, så trækkes prisen for ejendommen fra spillerens beholdning og tilføjes til den gamle ejers beholdning. Til sidst ændres ejeren til køberen.

Vi ville showcase denne del af kode, da vi mener det at kunne handle mellem hinanden er et vigtigt punkt i matador spillet og at et matador spil uden handel ikke er et rigtigt matador spil. Denne måde at gøre det på er også smart da udover at handlen sker i koden, har vi også handel face-to-face, da prisen diskuteres indbyrdes mellem spillerne.

Kode for Chancekort:

```
24
25  /**
26   * Blander chancekortene
27   */
28  public void blandKort() {
29      for (int i = 0; i < dæk.length; i++) {
30          float f = (float) Math.random() * dæk.length;
31          int a = (int) f;
32
33          int temp = dæk[i];
34          dæk[i] = dæk[a];
35          dæk[a] = temp;
36      }
37  }
38
39  /**
40   * Lader en spiller trække et chance-kort
41   * @return
42   */
43  public int trækKort() {
44      int temp = dæk[0];
45      for (int i = 0; i < dæk.length - 1; i++) {
46          dæk[i] = dæk[i + 1];
47      }
48      dæk[dæk.length - 1] = temp;
49      return temp;
50  }
```

I den første metode blandes kortene i bunken af chancekort. Metoden kører alle 21 kort i dækket gennem for-lykken og bruger Math.random metoden til at blande kortene. Den anden metode lader en spiller trække et chancekort fra bunken. Først oprettes variablen temp og sættes lig dæk-arrayet. Så kører samtlige værdier i arrayet igennem en for-lykke og lægger 1 til i-værdien. Altså trækker man det "øverste" kort i bunken først, kortet under næste gang osv. Til sidst returneres temp værdien, som så er det valgte chancekort.

Test

Traceability matrix

	TC1	TC2	TC3	TC4	TC5	TC6	TC7	TC8	TC9	TC10	TC11	TC12	TC13	TC14	TC15	TC16	TC17	TC18	TC19	TC20	TC21	JUnit1	JUnit2	JUnit3	JUnit4	JUnit5	JUnit6	JUnit7	JUnit8	JUnit9	JUnit10
K1	x																														
K2		x																													
K3																															
K4			x																												
K5				x																											
K6					x																										
K7						x																									
K8																															
K9							x																								
K10																															
K11								x																							
K12									x																						
K13										x																					
K14											x																				
K15														x																	
K16															x																
K17																															
K18																															
K19																x															
K20																	x														
K21													x																		
K22																															
K23																															
K24																	x														
K25																		x													
K26																															
K27																															
K28																															
K29																															

Formelle test cases:

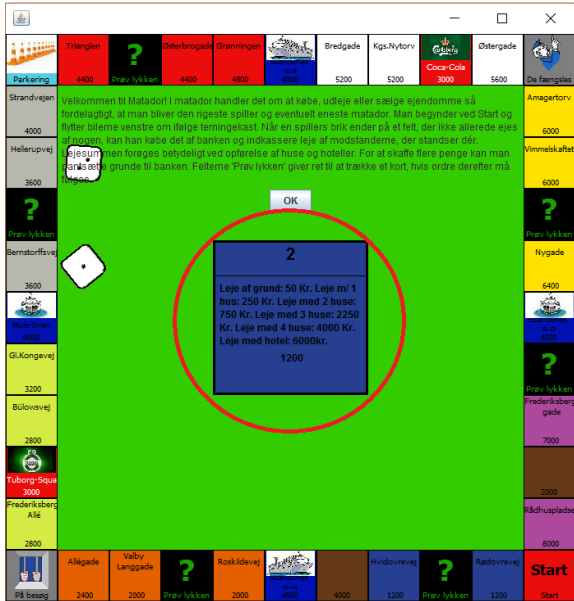
Testcase ID	TC01
Summary	Tests dice rolling
Requirements	R1
Preconditions	Player has clicked “ok” to roll the dice
Postconditions	The dice rolls
Test Procedure	Start the game Roll the dice Check if the dice have rolled
Test Data	Dice roll eg. value = 9
Expected Result	The system rolls 9
Actual Result	The system rolled 9
Status	Passed
Tested by	Johnny Chen
Date	12-01-2017
Date environment	Eclipse Neon 4.6.0 for Windows 10 Home

Testcase ID	TC02
Summary	The player moves to the correct field according to the dice roll
Requirements	R2
Preconditions	The player has rolled the dice
Postconditions	The player has moved to the correct field
Test Procedure	<p>Start the game</p> <p>Roll the dice</p> <p>Check if you have been moved to the correct field</p>
Test Data	Dice roll value = 5, Field 5 = Scandlines Helsingør-Helsingborg
Expected Result	The system moves the player to the correct field
Actual Result	The system moved the player to the correct field
Status	Passed
Tested by	Johnny Chen
Date	12-01-2017
Date environment	Eclipse Neon 4.6.0 for Windows 10 Home

Testcase ID	TC03
-------------	------


Summary	The system shall manage the account according to which field the player lands on and transfer correctly to other players if field is owned.
Requirements	R4
Preconditions	The game has started. Player 2 owns the property "Hvidovrevej".
Postconditions	The player who landed on the owned field has transferred the correct amount of kroner to the owner.
Test Procedure	Player 1 rolls the dice Player 1 lands on a owned property. The system subtracts the correct amount of kroner from player 1 to the owner.
Test Data	Dice roll value = 3, Field 3 = Hvidovrevej (Owned by player 2).
Expected Result	The system subtracts 50 kr. from player 1 and gives it to player 2.
Actual Result	Player 1's account lost 50 kr. and player 2 gained 50 kr.
Status	Passed
Tested by	Johnny Chen and Jonathan Friis
Date	13-01-2017
Date environment	Eclipse Neon 4.6.0 for Windows 10 Home

Testcase ID	TC04
Summary	The system shall show a message when the player hovers the mouse over a field.
Requirements	R5
Preconditions	The game has started.
Postconditions	A message shows.
Test Procedure	<p>Hover the mouse over a field(In this case "Rødovrevej")</p> <p>The system shows a message</p>
Test Data	Look at attachment
Expected Result	<p>The system shows the correct description of the field:</p> <p>"Leje af grund: 50 kr."</p> <p>"Leje m/1 hus: 250 kr."</p> <p>"Leje m/2 huse: 750 kr."</p> <p>"Leje m/3 huse: 2250 kr."</p> <p>"Leje m/4 huse: 4000 kr."</p> <p>"Leje med hotel: 6000 kr."</p>
Actual Result	<p>The systems shows:</p> <p>"Leje af grund: 50 kr."</p> <p>"Leje m/1 hus: 250 kr."</p>


	<p>“Leje m/2 huse: 750 kr.”</p> <p>“Leje m/3 huse: 2250 kr.”</p> <p>“Leje m/4 huse: 4000 kr.”</p> <p>“Leje med hotel: 6000 kr.”</p>
Status	Passed
Tested by	Johnny Chen and Jonathan Friis
Date	13-01-2017
Date environment	Eclipse Neon 4.6.0 for Windows 10 Home
Attachment	

Testcase ID	TC05
Summary	The system shall give a player the opportunity to choose between to buy

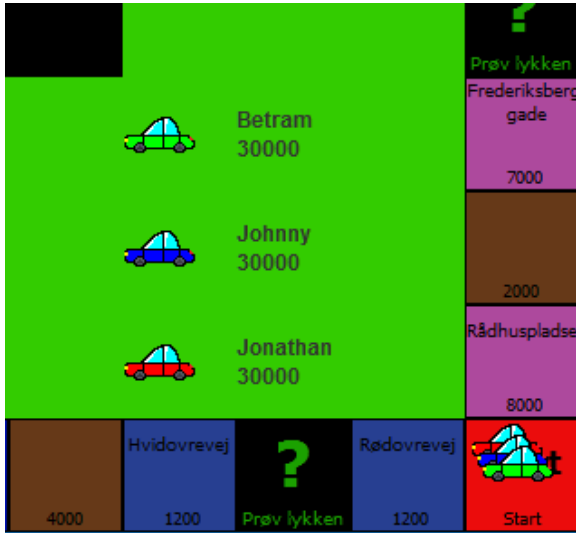
	or not to buy, when the player lands on a property.
Requirements	R6
Preconditions	The game has started. The player lands on an unowned property.
Postconditions	The system gives the player a choice.
Test Procedure	The player rolls the dice. The players lands on an unowned property. The system gives the player a choice.
Test Data	Dice roll value = 6. Field 6 = "Roskildevej".
Expected Result	The system gives the player a choice to buy or not to buy
Actual Result	The system shows: "Vil du købe Scandlines H-H for 4000 kr?" "Ja eller nej"
Status	Passed
Tested by	Johnny Chen and Jonathan Friis
Date	13-01-2017
Date environment	Eclipse Neon 4.6.0 for Windows 10 Home

Attachment	
------------	--

Testcase ID	TC06
Summary	The system creates a board with unique field names and values.
Requirements	R7
Preconditions	None
Postconditions	The system creates a board with unique field name and values.
Test Procedure	<p>The player starts the game.</p> <p>The system creates the board.</p>
Test Data	None
Expected Result	The system creates a board with unique field names and values.
Actual Result	The system created a board with unique field names and values.
Status	Passed
Tested by	Johnny Chen and Jonathan Friis
Date	13-01-2017

Date environment	Eclipse Neon 4.6.0 for Windows 10 Home
Attachment	

Testcase ID	TC07
Summary	The system places every player on “Start” when the game starts.
Requirements	R9
Preconditions	The number and names of the players has been chosen.
Postconditions	The game sets the players cars on start.
Test Procedure	<p>Start the game.</p> <p>The system sets all the player cars on the “Start” field.</p>
Test Data	Player number = 3.
Expected Result	The system places the player cars on the start field.

Actual Result	The system placed the player cars on the start field.
Status	Passed
Tested by	Jonathan Friis
Date	13-01-2017
Date environment	Eclipse Neon 4.6.0 for Windows 10 Home
Attachment	

Testcase ID	TC08
Summary	The system shall give a player an extra turn if he/she rolls the dice so that each die has the same value.
Requirements	R11
Preconditions	The game has started

	It's the player's turn
Postconditions	The system gives the player another turn.
Test Procedure	<p>The players rolls the dice</p> <p>The roll has the value 3 and 3.</p> <p>The system moves the player and awards the player another turn.</p>
Test Data	Dice roll values: 3 and 3.
Expected Result	The system moves the player and gives him/her another turn
Actual Result	The system moved the player and gave the him/her another turn
Status	Passed
Tested by	Jonathan Friis
Date	15-01-2017
Date environment	Eclipse Neon 4.6.0 for Windows 10 Home

Testcase ID	TC09
Summary	The system shall send a player to jail if he/she rolls to equal value dice, three times in a row.
Requirements	R12

Preconditions	<p>The game has started</p> <p>The player has rolled two equal value dice rolls two times in a row.</p>
Postconditions	The player is in jail
Test Procedure	<p>The player rolls the dice</p> <p>The player rolled 3 and 3.</p> <p>The system puts the player in jail</p>
Test Data	Dice roll value = 3 and 3.
Expected Result	The system jails the player.
Actual Result	The system jailed the player
Status	Passed
Tested by	Jonathan Friis
Date	15-01-2017
Date environment	Eclipse Neon 4.6.0 for Windows 10 Home


Testcase ID	TC10
Summary	The system shall release a player from jail if he/she rolls the dice and rolls two equal value dice.

Requirements	R13
Preconditions	<p>The game has started.</p> <p>The player is in jail.</p> <p>It's the player's turn.</p>
Postconditions	The game releases the player from jail and gives the player another turn.
Test Procedure	<p>The player rolls the dice</p> <p>The player rolls 5 and 5.</p> <p>The system releases the player from jail.</p> <p>The player is given another turn.</p>
Test Data	Dice roll value = 5 and 5.
Expected Result	The system releases the player and gives him/her another turn.
Actual Result	The system released the player and gave him/her another turn.
Status	Passed
Tested by	Jonathan Friis
Date	13-01-2017
Date environment	Eclipse Neon 4.6.0 for Windows 10 Home

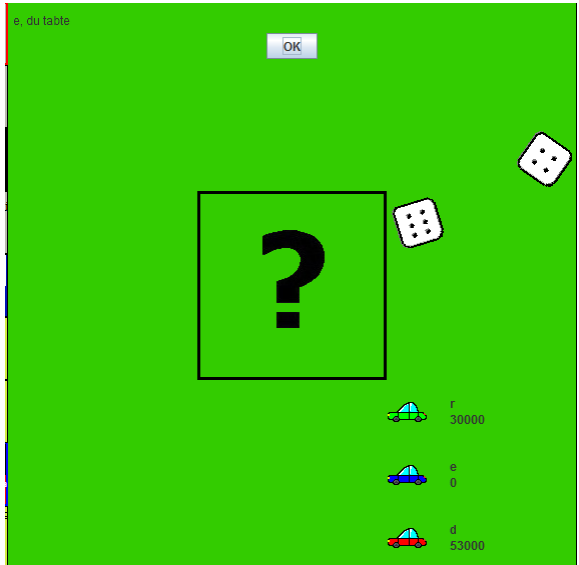
Testcase ID	TC11
Summary	If a player can't roll two equal value dice within three turns, the system takes 1000 kr from the player., releases the player and give the player another turn.
Requirements	R14
Preconditions	The player has been two turns in jail.
Postconditions	The game releases the player from jail, takes 1000 kr. from the player and gives the player another turn.
Test Procedure	<p>The player rolls the dice.</p> <p>The player doesn't roll two equal value dice.</p> <p>The system takes 1000 kr. from the player.</p> <p>The system releases the player from jail and gives the player another turn.</p>
Test Data	None
Expected Result	The system releases the player, takes 1000 kr. from the player and gives him/her another turn.
Actual Result	The system released the player, took 1000 kr. and gave him/her another turn.
Status	Passed
Tested by	Jonathan Friis

Date	13-01-2017
Date environment	Eclipse Neon 4.6.0 for Windows 10 Home

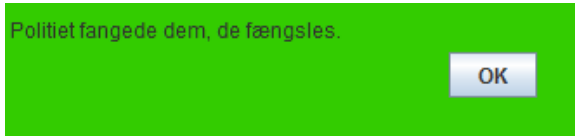
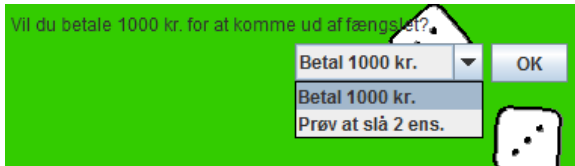
Testcase ID	TC12
Summary	The system shall only allow a player to build a house/hotel on a property if the player has at least as many houses on the other properties of the same group(color).
Requirements	R21
Preconditions	<p>The player has all properties of one group.</p> <p>The player has build one house on all except one property.</p> <p>It's the player's turn.</p>
Postconditions	The system only allows the player to build on properties that has at least as many house as the others.
Test Procedure	<p>The player presses "Køb hus"</p> <p>The system shows a dropdown menu of properties the player can build houses on.</p> <p>The players presses the property he/she wants to build a house on.</p>
Test Data	None
Expected Result	The system shows a drop down menu where the player can choose between properties where he/she is allowed to build houses.

Actual Result	The system showed the player a drop down menu where he/she could choose where to build houses, and it only showed the allowed properties.
Status	Passed
Tested by	Jonathan Friis
Date	13-01-2017
Date environment	Eclipse Neon 4.6.0 for Windows 10 Home
Attachment	

Testcase ID	TC13
Summary	The system shall declare a player broke/ lost the game, when a player reaches zero and can't pay his dept.
Requirements	R23
Preconditions	<p>The player doesn't have any houses sell.</p> <p>The player has 10 kr.</p>

Postconditions	The system declares the player broke/lost the game
Test Procedure	<p>The player rolls the dice</p> <p>The player lands on an owned property.</p> <p>The system declares the player broke</p>
Test Data	None
Expected Result	The system writes a message to the broke player.
Actual Result	The system writes: “(Name of player), du tabte.”
Status	Passed
Tested by	Jonathan Friis
Date	13-01-2017
Date environment	Eclipse Neon 4.6.0 for Windows 10 Home
Attachment	

Testcase ID	TC14
Summary	The system gives a jailed player a choice between paying 1000 kr. or rolling the dice (hoping for a dice roll with the same value for each dice) as a means for escaping jail.
Requirements	R15
Preconditions	The game has started The player is in jail.
Postconditions	The system gives the player a choice.
Test Procedure 1	The rolls the dice The player landed on jail The other players take their turn. The jailed player is given a choice between paying or rolling.
Test Data	None
Expected Result	The player is given a choice between paying or rolling to escape jail.
Actual Result	The system shows: "Politiet fangede dem, de fængsles" "Vil du betale 1000 kr. for at komme ud af fængslet?" "Betal 1000 kr."

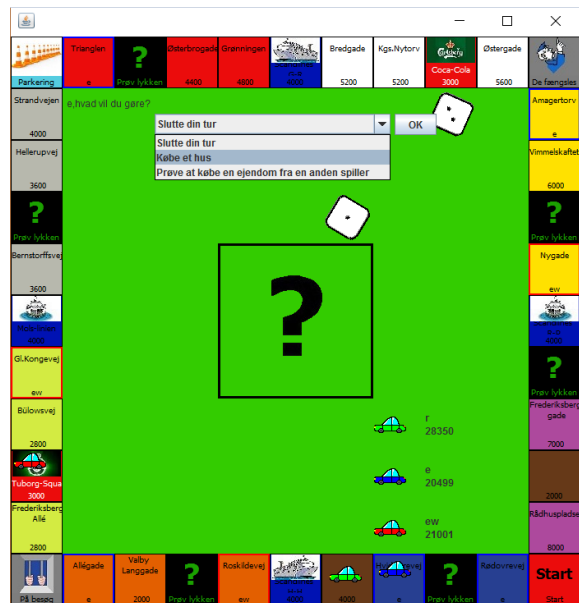
	"Prøv at slå 2 ens"
Status	Passed
Tested by	Johnny Chen and Jonathan Friis
Date	13-01-2017
Date environment	Eclipse Neon 4.6.0 for Windows 10 Home
Attachment	 

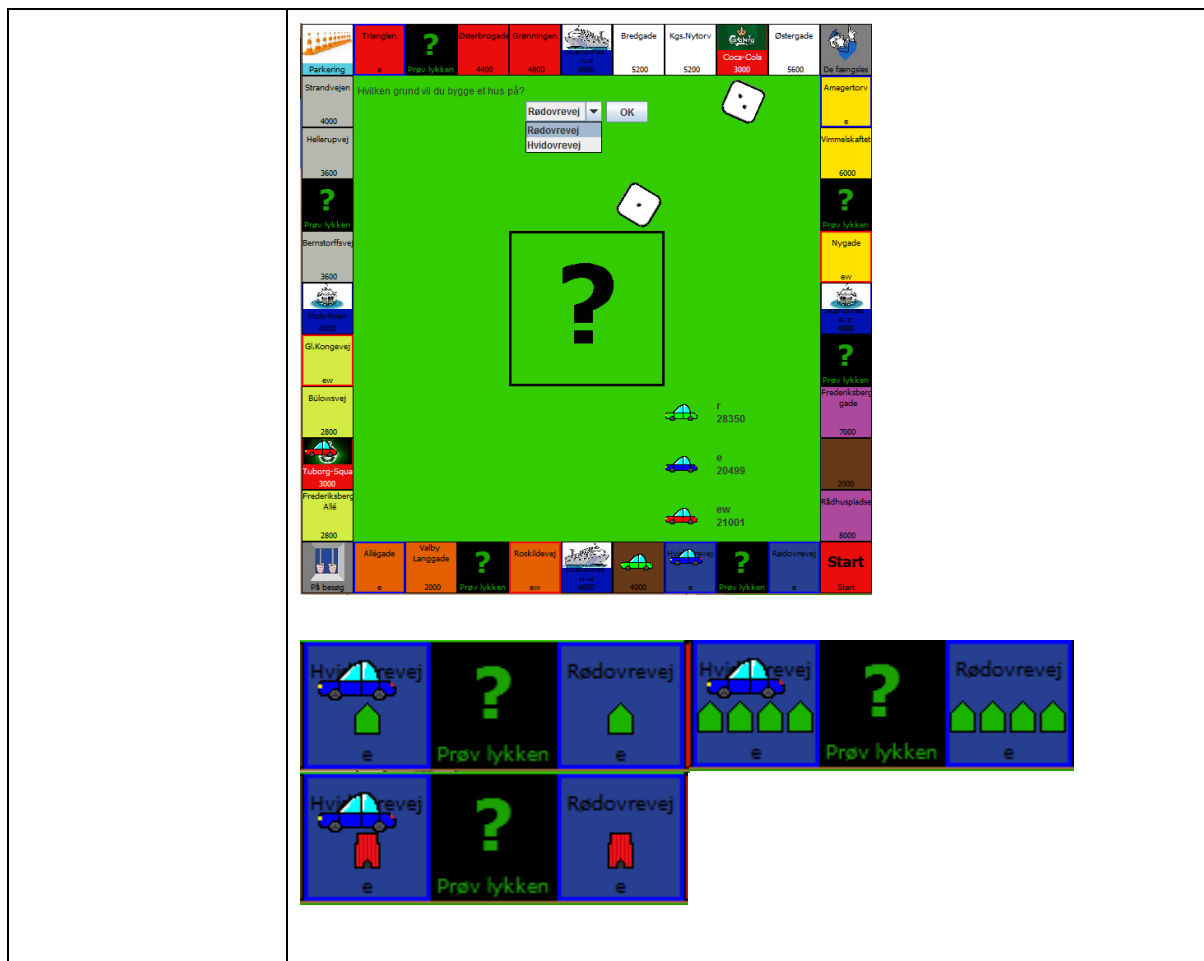
Testcase ID	TC15
Summary	The system shall give the player a "Prøv lykken kort" card if he lands on a "Prøv lykken" field.
Requirements	R16
Preconditions	<p>The game has started.</p> <p>The player has rolled a dice value, that will move him to a "Prøv lykken" field.</p>

Postconditions	The player is shown the “Prøv lykken kort” card.
Test Procedure	<p>The players rolls the dice</p> <p>The player lands on a “Prøv lykken kort” field.</p> <p>The system shows the player the card he/she draws.</p>
Test Data	Dice roll value = 2, Field 2 = “Prøv lykken kort” field.
Expected Result	The player is shown a “Prøv lykken kort” card.
Actual Result	<p>The system shows:</p> <p>“Eftergivelse af Kvartals skat: Modtag 3000 kr.”</p>
Status	Passed
Tested by	Johnny Chen and Jonathan Friis
Date	13-01-2017
Date environment	Eclipse Neon 4.6.0 for Windows 10 Home

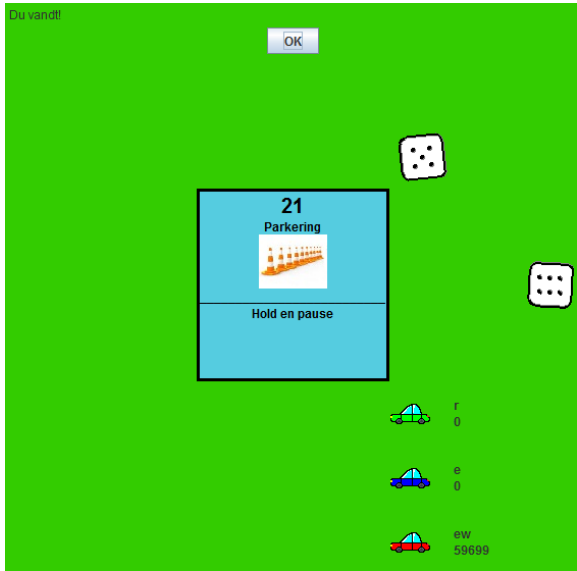
Attachment	
------------	--

Testcase ID	TC16
Summary	The system shall let the player buy houses/hotels on a property, if the player owns all properties of the same group(color).
Requirements	R19, R20
Preconditions	<p>The game has started</p> <p>The player owns all properties of a group.</p>
Postconditions	The system gives the player the choice to buy houses/hotels (Our game registrations 5 houses as a hotel).
Test Procedure	<p>It's the player's turn</p> <p>The system gives the player the choice to buy houses on his properties.</p>
Test Data	None

Expected Result	The systems gives the player the choice to buy a house
Actual Result	The system shows: "Købe et hus"
Status	Passed
Tested by	Johnny Chen and Jonathan Friis
Date	13-01-2017
Date environment	Eclipse Neon 4.6.0 for Windows 10 Home
Attachment	 <p>The screenshot shows the Monopoly game board with a green background. A central dialog box is open, displaying the text "Købe et hus" (Buy a house). The dialog box has a title bar "Slutte din tur" (End your turn) and a list of options: "Slutte din tur", "Købe et hus", and "Prøve at købe en ejendom fra en anden spiller". The "Købe et hus" option is selected. The game board shows various properties with their respective costs and icons. The player's token is a white car.</p>



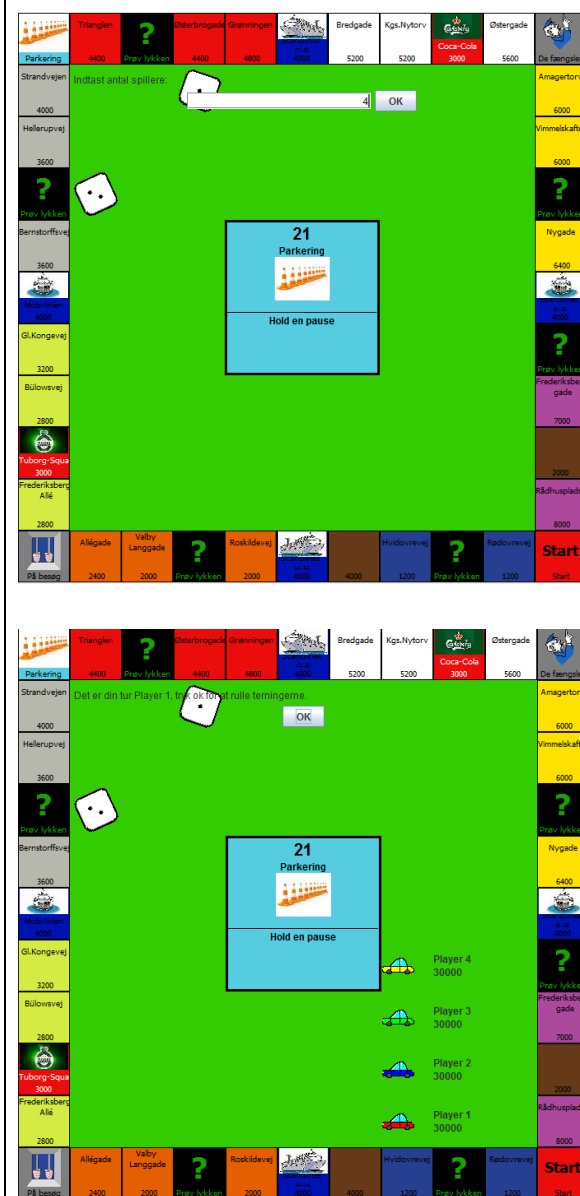
Testcase ID	TC17
Summary	The game will end when there is only one player left, who is not declared broke.
Requirements	R24
Preconditions	All players are broke except one player.
Postconditions	The lone survivor wins the game.
Test Procedure	Play the game until only one player remains

	The system announces the winner.
Test Data	None
Expected Result	The system announces the player who won.
Actual Result	The system displays following message: “(The players name) du vandt!”
Status	Passed
Tested by	Johnny Chen and Jonathan Friis
Date	13-01-2017
Date environment	Eclipse Neon 4.6.0 for Windows 10 Home
Attachment	

Testcase ID	TC18
-------------	------

Summary	The system shall be a game between 3-6 players on one computer
Requirements	R25
Preconditions	None
Postconditions	The system creates the correct number of players
Test Procedure	Start the game Choose between 3-6 players Type in name of players
Test Data	Number of players = 4
Expected Result	The system creates 4 players
Actual Result	The system created 4 players
Status	Passed
Tested by	Johnny Chen and Jonathan Friis
Date	13-01-2017
Date environment	Eclipse Neon 4.6.0 for Windows 10 Home

Attachment



Testcase ID	TC19
Summary	The system shall run without any errors for an hour
Requirements	R27
Preconditions	The players have started the game

Postconditions	The game ran successfully for an hour.
Test Procedure	Start the game Play the game for an hour
Test Data	None
Expected Result	The game runs for an hour without errors
Actual Result	The game ran for an hour without errors
Status	Passed
Tested by	Johnny Chen and Jonathan Friis
Date	15-01-2017
Date environment	Eclipse Neon 4.6.0 for Windows 10 Home

Testcase ID	TC20
Summary	The system shall be playable on the computers in the databars at DTU.
Requirements	R28
Preconditions	On a computer in a databar at DTU.
Postconditions	The game runs

Test Procedure	<p>Pull the game into eclipse.</p> <p>Run the main class.</p> <p>The computer runs the game with no problems.</p>
Test Data	None
Expected Result	The computer runs the game.
Actual Result	The computer ran the game.
Status	Passed
Tested by	Johnny Chen and Jonathan Friis
Date	13-01-2017
Date environment	Eclipse Neon 4.6.0 for Windows 10 Home

Testcase ID	TC21
Summary	The system shall show thee the dice roll value within 0.6 seconds
Requirements	R29
Preconditions	The player has started the game
Postconditions	The system shows the dice value to the player within 0.6 seconds

Test Procedure	Start the game Roll the dice
Test Data	Dice value = 6
Expected Result	The system shows dice value 6 within 0.6 seconds
Actual Result	The system showed value 6 within 0.6 seconds
Status	Passed
Tested by	Johnny Chen and Jonathan Friis
Date	13-01-2017
Date environment	Eclipse Neon 4.6.0 for Windows 10 Home

JUnit-test

JUnit 1 testAddCoins:

I denne JUnit Test tester man funktionen for at tilføje penge til den nuværende spillers konto. Der sættes en expected int for forløbet man regner med at have. Derefter ændres der et par gange på beløbet, hvor der trækkes og lægges penge til og fra en spillers saldo, som pr. automatik er sat til at starte som 30000. actual int værdien sættes til at være funktionen for at få den nuværende saldo. Ved JUnit test ses det, at denne test er succesfuld.

JUnit 2 testAccount:

I denne JUnit Test tester man saldoen på spillerens konto ved at sætte den forventede værdi til at være 30000 kr. actual værdien er her værdien når man bruger get'eren til den nuværende spillers konto. Denne test var også vellykket.

JUnit 3 testFlytPosition:

I Denne JUnit Test afprøver man spillets funktion for at rykke en spiller frem. Dette gør man ved at sætte en expected position, som i vores tilfælde er 8. Derefter Bruger vi funktionen til at rykke 8 felter frem. Vores int actual er den nuværende position for spilleren efter at have rykket de 8 felter. Denne test virkede fint, og derfor kan koden konkluderes som vellykket.

JUnit 4 testStartPosition:

I denne JUnit Test afprøver vi, om det at passere eller lande på startfeltet giver spilleren 4000 kroner. Måden vi gør det på er ved at tage sætte vores forventede værdi (int expected) til at være 34000, dvs. 4000 oveni ens startkonto, som man får ved at passere start feltet. Den faktiske værdi sættes til getKroner for test spilleren efter man har rykket spilleren 42 felter (med det tal er man sikker på man har passeret feltet. Vores test har været positiv, så vi kan konkludere, at vores eksperiment har været succesfuldt.

JUnit 5 testTerning:

I denne JUnit Test ser man på, om terningen slår en sum mellem 2 og 12 i 1000 slag. Man sætter Int expected til at være 1000, og actual starter som 0. Man kører så en for løkke til at tjekke om terningen fungerer efter design. For hvert slag hvis sum er indenfor det rigtige tal lægges der et tal oven i actual indtil 1000. Hvis det lykkes, er det et succesfuldt program. Programmet har fungeret efter design, da testen var positiv.

JUnit 6 testRoll:

I denne test afprøver man hvor pålidelig terningen er i forhold til at slå de forskellige værdier med jævn fordeling. Man opsætter seks forskellige værdier, som hver har sit switch case. For hver gang der bliver rullet en af de seks værdier, tælles det op for den bestemte værdi. Der rulles 60000 gange med terningerne. Der testes for hver mulig værdi om der er omkring 1/6 af hver værdi rullet, med forbehold for afvigelse på 4 procent både positivt og negativt. Hvis det kriterie er opfyldt for alle værdier, er testen vellykket. Det har testen været i vores eksempel. Derfor virker koden fint.

JUnit 7 testFireTusind

Her tester vi om spilleren har mulighed for at vælge at betale 4000, når spilleren er landet på 'skat'-feltet. Først flyttes spilleren til position 4 som er skattefeltet. Den forventede pengebeholdning sættes så til 26000 og så trækkes der 4000 fra spillerens pengebeholdning. Den aktuelle pengebeholdning sættes til .getKroner metoden og assertEquals tester om de to værdier er ens.

JUnit 8 testTiProcent

Her tester vi om spilleren får muligheden for at betale 10% af sin pengebeholdning i stedet for 4000 kr. Først oprettes tiProcent-variablen og spilleren flyttes så til felt 4, som er skattefeltet. tiProcent variablen

sættes til `getKroner / 10`, som er de 10% af den samlede pengebeholdning. Den forventede pengebeholdning er altså 27000. Så trækkes 'tiProcent' vha. `tilføjKroner` metoden og den faktiske pengebeholdning sættes som `actual`. `AssertEquals` tester så de to værdier og ser om de er ens.

JUnit 9 TestSælgHus

Her tester vi om spilleren får halvdelen af de penge tilbage når han sælger et hus. Først lader vi spilleren være ejeren af grundene, så sætter vi huse på grundene, og så får vi spilleren til at sælge 2 forskellige huse og checker om han har fået den rigtige mængde kroner tilbage.

JUnit 10 TestLandPåEjetGruppe

Her tester vi om systemet opkæver den rigtige mængde penge når en spiller lander på et felt hvor ejeren ejer hele gruppen af grunde. Først får vi spiller 1 til at lande på en grund ejet af spiller 2, så giver vi spiller 2 ejerskab over alle grunde i den gruppe, og så får vi spiller 1 til at lande på grunden igen og checker om han har betalt 3 gange lejen i alt.

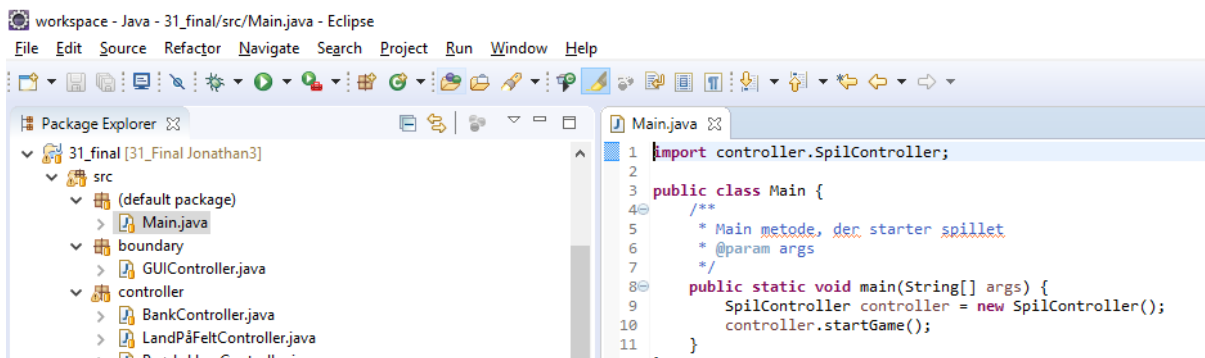
Konfiguration

Programmet kræver en computer med java 1.7 eller nyere installeret og eclipse installeret.

Guide til importering af git repository:

1. Start browser
2. Gå ind på siden: https://github.com/BertramHenning/31_Final
3. Klik på "clone or download" og kopier derefter linket
4. Åben eclipse
5. Klik på "file" og vælg "import"
6. Vælg "Git-> Projects from Git"
7. Tryk "next"
8. Klik på "clone URL"

9. Klik på “next”
10. Klik på “next” igen
11. Vælg sti at gemme det på
12. Klik på “next”
13. Vælg “import existing eclipse projects” og tryk “next”
14. Sørg for at project mappen har et flueben ud for sig eller klik i den tomme kasse
15. Tryk “finish”
16. Åben projektet i eclipse
17. Åben “src”
18. Åben pakken “(Default Package)”
19. Åben Main.java
20. Klik på den grønne knap med den hvide pil i.



Konklusion

Overordnet konklusion

Ud fra vores Matador projekt kan vi konkludere, at vores produkt stemmer overens med de krav vi har formuleret ud fra kundens vision og dermed er et vellykket projekt.

Produktorienteret konklusion

Til at udvikle dette produkt har vi delvist brugt viden fra vores tidligere CDIO opgaver til at anskaffe os ideer og mønstre til en effektiv udviklingsproces. Vores UML diagrammer og use cases har i høj grad hjulpet os med at få et mere overskueligt blik over hvordan vores endelige produkt har skulle se ud. For at kunne gøre arbejdsfordelingen mere effektiv har vi ved brug af GitHub kunne fusionere vores individuelle dele som vi hver især har bidraget med til det endelige projekt. Ved brug af test cases og JUnit tests har vi kunnet sikre os, at koden har fungeret som den skulle, og at der ikke er eventuelle fejl. Hermed kan vi fra et produktorienteret synspunkt konkludere, at vores projekt er blevet udviklet effektivt og succesfuldt.

Forslag til videre arbejde og forbedringer

- Pantsætning
- Auktion
- Flere prøv lykken kort
- Giv op mulighed

Litteraturliste

CDIO-3

Larman, Craig: "Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development", tredje udgave, Addison Wesley Professional, Oktober 20 2004.

<https://www.campusnet.dtu.dk/cnnet/filessharing/SADownload.aspx?FileId=4298361&FolderId=1026792&ElementId=522003> - 02313, Udviklingsmetoder til IT-systemer. Lektion 06: Use case realisering, design.

Forfatter: Ian Bridgwood, Henrik Bechmann - Afdeling for informatik, DTU Diplom. Sidst besøgt: 15-01-16

<https://www.campusnet.dtu.dk/cnnet/filessharing/SADownload.aspx?FileId=4291632&FolderId=1025146&ElementId=522003> - 02313, Udviklingsmetoder til IT-systemer. Lektion 05: Analyse, UML notation.

Forfatter: Henrik Bechmann - Afdeling for informatik, DTU Diplom. Sidst besøgt: 15-01-16

<https://www.campusnet.dtu.dk/cnnet/filessharing/SADownload.aspx?FileId=4285254&FolderId=1023805&ElementId=522003> - 02313, Udviklingsmetoder til IT-systemer. Lektion 04: Use case modellering. Forfatter:

Henrik Bechmann - Afdeling for informatik, DTU Diplom. Sidst besøgt: 15-01-16

<https://www.campusnet.dtu.dk/cnnet/filessharing/SADownload.aspx?FileId=4277841&FolderId=1021827&ElementId=522003> - 02313, Udviklingsmetoder til IT-systemer. Lektion 03: Rapportskrivning. Forfatter:

Henrik Bechmann, Henrik Tange - Afdeling for informatik, DTU Diplom. Sidst besøgt: 15-01-16

Bilag

Bilag 1:

Oversigt over alle felterne og deres skøder i matador spillet.

Startfelt: Modtag 4000 kr.

Rødovrevej:

Farve	Blå
Pris	1200 kr.
Leje	50 kr.
Leje m/ 1 hus	250 kr.
Leje m/ 2 hus	750 kr.
Leje m/ 3 hus	2250 kr.
Leje m/ 4 hus	4000 kr.
Leje m/ hotel	6000 kr.
Hus pris	1000 kr.
Hotel pris	1000 kr. + 4 huse

Prøv lykken felt: Træk et prøv lykken kort.

Hvidovrevej:

Farve	Blå
Pris	1200 kr.
Leje	50 kr.
Leje m/ 1 hus	250 kr.
Leje m/ 2 hus	750 kr.
Leje m/ 3 hus	2250 kr.
Leje m/ 4 hus	4000 kr.
Leje m/ hotel	6000 kr.
Hus pris	1000 kr.
Hotel pris	1000 kr. + 4 huse

Indkomst skat felt: Betal 4000 kr. eller 10% af pengebeholdning.

Scandlines Helsingør-Helsingborg:

Type	Rederi
Pris	4000 kr.
Leje	500 kr.
Leje hvis 2 rederier ejes	1000 kr.

Leje hvis 3 rederier ejes	2000 kr.
Leje hvis 4 rederier ejes	4000 kr.
Pantsætnings Værdi	2000 kr.

Roskildevej:

Farve	Orange
Pris	2000 kr.
Leje	100 kr.
Leje m/ 1 hus	600 kr.
Leje m/ 2 hus	1800 kr.
Leje m/ 3 hus	5400 kr..
Leje m/ 4 hus	8000 kr.
Leje m/ hotel	11000 kr.
Hus pris	1000 kr.
Hotel pris	1000 kr. + 4 huse
Pantsætnings Værdi	1000 kr.

Prøv lykken felt:

Træk et prøv lykken kort.

Valby Langgade:

Farve	Orange
Pris	2000 kr.
Leje	100 kr.
Leje m/ 1 hus	600 kr.
Leje m/ 2 hus	1800 kr.
Leje m/ 3 hus	5400 kr..
Leje m/ 4 hus	8000 kr.
Leje m/ hotel	11000 kr.
Hus pris	1000 kr.
Hotel pris	1000 kr. + 4 huse
Pantsætnings Værdi	1000 kr.

Allégade:

Farve	Orange
Pris	2400 kr.

Leje	150 kr.
Leje m/ 1 hus	800 kr.
Leje m/ 2 hus	2000 kr.
Leje m/ 3 hus	6000 kr..
Leje m/ 4 hus	9000 kr.
Leje m/ hotel	12000 kr.
Hus pris	1000 kr.
Hotel pris	1000 kr. + 4 huse
Pantsætnings Værdi	1200 kr.

Fængsel: På besøg

Frederiksberg Allé:

Farve	Grøn
Pris	2800 kr.
Leje	200 kr.
Leje m/ 1 hus	1000 kr.
Leje m/ 2 hus	3000 kr.

Leje m/ 3 hus	9000 kr..
Leje m/ 4 hus	12500 kr.
Leje m/ hotel	15000 kr.
Hus pris	2000 kr.
Hotel pris	2000 kr. + 4 huse
Pantsætnings Værdi	1400 kr.

Tuborg squash:

Type	Tapperi
Pris	3000 kr.
Leje	100 gange så meget som terningeslaget
Leje hvis Coca-cola ejes	200 gange så meget som terningeslaget
Pantsætnings Værdi	1500 kr.

Bülowsvej:

Farve	Grøn
Pris	2800 kr.

Leje	200 kr.
Leje m/ 1 hus	1000 kr.
Leje m/ 2 hus	3000 kr.
Leje m/ 3 hus	9000 kr..
Leje m/ 4 hus	12500 kr.
Leje m/ hotel	15000 kr.
Hus pris	2000 kr.
Hotel pris	2000 kr. + 4 huse
Pantsætnings Værdi	1400 kr.

Gl. Kongevej:

Farve	Grøn
Pris	3200 kr.
Leje	250 kr.
Leje m/ 1 hus	1250 kr.
Leje m/ 2 hus	3750 kr.

Leje m/ 3 hus	10000 kr..
Leje m/ 4 hus	14000 kr.
Leje m/ hotel	18000 kr.
Hus pris	2000 kr.
Hotel pris	2000 kr. + 4 huse
Pantsætnings Værdi	1600 kr.

Mols-Linien:

Type	Rederi
Pris	4000 kr.
Leje	500 kr.
Leje hvis 2 rederier ejes	1000 kr.
Leje hvis 3 rederier ejes	2000 kr.
Leje hvis 4 rederier ejes	4000 kr.
Pantsætnings Værdi	2000 kr.

Bernstorffsvej:

Farve	Grå
Pris	3600 kr.
Leje	300 kr.
Leje m/ 1 hus	1400 kr.
Leje m/ 2 hus	4000 kr.
Leje m/ 3 hus	11000 kr..
Leje m/ 4 hus	15000 kr.
Leje m/ hotel	19000 kr.
Hus pris	2000 kr.
Hotel pris	2000 kr. + 4 huse
Pantsætnings Værdi	1800 kr.

Prøv lykken felt: Træk et prøv lykken kort.

Hellerupvej:

Farve	Grå
Pris	3600 kr.
Leje	300 kr.

Leje m/ 1 hus	1400 kr.
Leje m/ 2 hus	4000 kr.
Leje m/ 3 hus	11000 kr..
Leje m/ 4 hus	15000 kr.
Leje m/ hotel	19000 kr.
Hus pris	2000 kr.
Hotel pris	2000 kr. + 4 huse
Pantsætnings Værdi	1800 kr.

Strandvejen:

Farve	Grå
Pris	4000 kr.
Leje	350 kr.
Leje m/ 1 hus	1600 kr.
Leje m/ 2 hus	4400 kr.
Leje m/ 3 hus	12000 kr..

Leje m/ 4 hus	16000 kr.
Leje m/ hotel	20000 kr.
Hus pris	2000 kr.
Hotel pris	2000 kr. + 4 huse
Pantsætnings Værdi	2000 kr.

Parkerings felt: Fristed indtil næste tur

Trianglen:

Farve	Rød
Pris	4400 kr.
Leje	350 kr.
Leje m/ 1 hus	1800 kr.
Leje m/ 2 hus	5000 kr.
Leje m/ 3 hus	14000 kr..
Leje m/ 4 hus	17500 kr.
Leje m/ hotel	21000 kr.
Hus pris	3000 kr.

Hotel pris	3000 kr. + 4 huse
Pantsætnings Værdi	2200 kr.

Prøv lykken felt: Træk et prøv lykken kort.

Østerbrogade:

Farve	Rød
Pris	4400 kr.
Leje	350 kr.
Leje m/ 1 hus	1800 kr.
Leje m/ 2 hus	5000 kr.
Leje m/ 3 hus	14000 kr..
Leje m/ 4 hus	17500 kr.
Leje m/ hotel	21000 kr.
Hus pris	3000 kr.
Hotel pris	3000 kr. + 4 huse
Pantsætnings Værdi	2200 kr.

Grønningen:

Farve	Rød
Pris	4800 kr.
Leje	400 kr.
Leje m/ 1 hus	2000 kr.
Leje m/ 2 hus	6000 kr.
Leje m/ 3 hus	15000 kr..
Leje m/ 4 hus	18000 kr.
Leje m/ hotel	22000 kr.
Hus pris	3000 kr.
Hotel pris	3000 kr. + 4 huse
Pantsætnings Værdi	2400 kr.

Scandlines Gedser-Rostock:

Type	Rederi
Pris	4000 kr.
Leje	500 kr.

Leje hvis 2 rederier ejes	1000 kr.
Leje hvis 3 rederier ejes	2000 kr.
Leje hvis 4 rederier ejes	4000 kr.
Pantsætnings Værdi	2000 kr.

Bredgade:

Farve	Hvid
Pris	5200 kr.
Leje	450 kr.
Leje m/ 1 hus	2200 kr.
Leje m/ 2 hus	6600 kr.
Leje m/ 3 hus	16000 kr..
Leje m/ 4 hus	19500 kr.
Leje m/ hotel	23000 kr.
Hus pris	3000 kr.
Hotel pris	3000 kr. + 4 huse

Pantsætnings Værdi	2600 kr.
--------------------	----------

Kgs. Nytorv:

Farve	Hvid
Pris	5200 kr.
Leje	450 kr.
Leje m/ 1 hus	2200 kr.
Leje m/ 2 hus	6600 kr.
Leje m/ 3 hus	16000 kr..
Leje m/ 4 hus	19500 kr.
Leje m/ hotel	23000 kr.
Hus pris	3000 kr.
Hotel pris	3000 kr. + 4 huse
Pantsætnings Værdi	2600 kr.

Coca-cola felt:

Type	Tapperi
------	---------

Pris	3000 kr.
Leje	100 gange så meget som terningeslaget
Leje hvis Tuborg Squash ejes	200 gange så meget som terningeslaget
Pantsætnings Værdi	1500 kr.

Østergade:

Farve	Hvid
Pris	5600 kr.
Leje	500 kr.
Leje m/ 1 hus	2400 kr.
Leje m/ 2 hus	7200 kr.
Leje m/ 3 hus	17000 kr..
Leje m/ 4 hus	20500 kr.
Leje m/ hotel	24000 kr.
Hus pris	3000 kr.
Hotel pris	3000 kr. + 4 huse

Pantsætnings Værdi	2800 kr.
--------------------	----------

Fængsels felt: Fanget her indtil man har opfyldt kravene for at komme ud(se kravspecifikation).

Amagertorv:

Farve	Gul
Pris	6000 kr.
Leje	550 kr.
Leje m/ 1 hus	2600 kr.
Leje m/ 2 hus	7800 kr.
Leje m/ 3 hus	18000 kr..
Leje m/ 4 hus	22000 kr.
Leje m/ hotel	25000 kr.
Hus pris	4000 kr.
Hotel pris	4000 kr. + 4 huse
Pantsætnings Værdi	3000 kr.

Vimmelskftet:

Farve	Gul
Pris	6000 kr.
Leje	550 kr.
Leje m/ 1 hus	2600 kr.
Leje m/ 2 hus	7800 kr.
Leje m/ 3 hus	18000 kr..
Leje m/ 4 hus	22000 kr.
Leje m/ hotel	25000 kr.
Hus pris	4000 kr.
Hotel pris	4000 kr. + 4 huse
Pantsætnings Værdi	3000 kr.

Prøv lykken felt: Træk et prøv lykken kort.

Nygade:

Farve	Gul
Pris	6400 kr.
Leje	600 kr.

Leje m/ 1 hus	3000 kr.
Leje m/ 2 hus	9000 kr.
Leje m/ 3 hus	20000 kr..
Leje m/ 4 hus	24000 kr.
Leje m/ hotel	28000 kr.
Hus pris	4000 kr.
Hotel pris	4000 kr. + 4 huse
Pantsætnings Værdi	3200 kr.

Scandlines Rødby-Puttgarden:

Type	Rederi
Pris	4000 kr.
Leje	500 kr.
Leje hvis 2 rederier ejes	1000 kr.
Leje hvis 3 rederier ejes	2000 kr.
Leje hvis 4 rederier ejes	4000 kr.

Pantsætnings Værdi	2000 kr.
--------------------	----------

Prøv lykken felt: Træk et prøv lykken kort.

Frederiksberggade:

Farve	Lilla
Pris	7000 kr.
Leje	700 kr.
Leje m/ 1 hus	3500 kr.
Leje m/ 2 hus	10000 kr.
Leje m/ 3 hus	22000 kr..
Leje m/ 4 hus	26000 kr.
Leje m/ hotel	30000 kr.
Hus pris	4000 kr.
Hotel pris	4000 kr. + 4 huse
Pantsætnings Værdi	3500 kr.

Ekstraordinær statsskat felt: Betal 2000 kr. til banken.

Rådhuspladsen:

Farve	Lilla
Pris	8000 kr.
Leje	1000 kr.
Leje m/ 1 hus	4000 kr.
Leje m/ 2 hus	12000 kr.
Leje m/ 3 hus	28000 kr..
Leje m/ 4 hus	34000 kr.
Leje m/ hotel	40000 kr.
Hus pris	4000 kr.
Hotel pris	4000 kr. + 4 huse
Pantsætnings Værdi	4000 kr.

Bilag 2: Prøv lykken kort.

Oversigt over de forskellige prøv lykken kort og deres funktioner.

Der er 21 kort i alt.

Når et kort er trukket og brugt lægges det nederst i bunken med de resterende "Prøv Lykken Kort".

Modtag penge (af banken) kort (12):

Eftergivelse af Kvartals skat: Modtag 3000 kr.

Præmieobligation udtrækkelse (2): Modtag 1000 kr.

Gageforhøjelse: Modtag 1000 kr.

Aktieudbytte (3): Modtag 1000 kr.

Møbel auktion: Modtag 1000 kr.

Klasselotteriet (2): Modtag 500 kr.

Tipning: Modtag 1000 kr.

Avl fra nyttehaven: Modtag 200 kr.

Betal penge kort (9):

Reparation af vogn (2): Betal 3000 kr.

Rødt lys bøde: Betal 1000 kr.

2 kasser øl: Betal 200 kr.

Parkeringsbøde: Betal 200 kr.

Bilforsikring: Betal 1000 kr.

Nye dæk: Betal 1000 kr.

Tandlægeregning: Betal 2000 kr.

Cigaret told: Betal 200 kr.