

---

# CDIO - 3

---

*Et simpelt brætspil for to spillere til seks spillere*

Af gruppe 33



Simon Engquist  
s143233



Arvid Langsø  
s144265



Mikkel Lund  
s165238



Jeppe Nielsen  
s093905



Mads Stege  
s165243

---

*Danmarks Tekniske Universitet  
DTU Compute*

25. november 2016 - Kl. 23:59

Side antal (med bilag): 60

Kurser: 02312, 02313, 02315

**Tidsplan for CDIO del 3 - 2016**

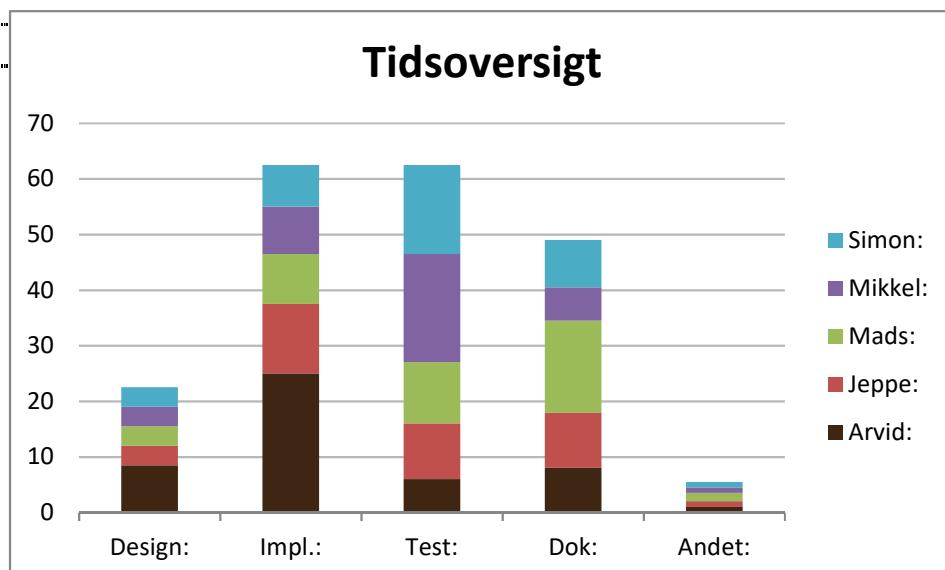
Dato	Deltager	Design:	Impl.:	Test	Dok.:	Andet:	I alt:
2016-11-07	Arvid	1,5	0	0	0	0	1,5
	Jeppe	1,5	0	0	0	0	1,5
	Mads	1,5	0	0	0	0	1,5
	Mikkel	1,5	0	0	0	0	1,5
	Simon	1,5	0	0	0	0	1,5
2016-11-08	Arvid	2	0	0	0	0	2
	Jeppe	2	0	0	0	0	2
	Mads	2	0	0	0	0	2
	Mikkel	2	0	0	0	0	2
	Simon	2	0	0	0	0	2
2016-11-10	Arvid	0	6	0	0	0	6
	Jeppe	0	6	0	0	0	6
	Mads	0	5	0	1	0	6
	Mikkel	0	6	0	0	0	6
	Simon	0	6	0	0	0	6
2016-11-12	Arvid	0	4	1	0	0	5
	Mikkel	0	4	0	0	0	0
2016-11-14	Arvid	0	1,5	0	0	0	0
	Jeppe	0	0	0	0	0	0
	Mads	0	0	0	1	0	1
	Mikkel	0	0	0	0	0	0
	Simon	0	0	0	0	0	0
2016-11-15	Arvid	0	1	0	0	0	1
	Jeppe	0	1	0	0	0	1
	Mads	0	1	0	0	0	1
	Mikkel	0	1	0	0	0	1
	Simon	0	0	0	0	0	0
2016-11-17	Arvid	0	5	1	0	1	7
	Jeppe	0	0	6	0	1	7
	Mads	0	0	5,5	0	1,5	7
	Mikkel	0	0	6	0	1	7
	Simon	0	0	6	0	1	7
2016-11-18	Arvid	0	1,5	0	0	0	1,5
	Jeppe	0	1,5	0	0	0	1,5
	Mads	0	1,5	0	0	0	1,5
	Mikkel	0	1,5	0	0	0	1,5
	Simon	0	1,5	0	0	0	1,5
2016-11-19	Arvid	1	4	0	0	0	5
	Jeppe	0	0	1,5	0	0	1,5
	Mads	0	0	2	0,5	0	2,5
	Mikkel	0	0	0	0	0	0
	Simon	0	0	0	0	0	0

CDIO3 Tidsplan

2016-11-21	Arvid	1	2	0	0	0	3
	Jeppe	0	0	2,5	0	0	2,5
	Mads	0	0	1,5	1	0	2,5
	Mikkel	0	0	2,5	0	0	2,5
	Simon	0	0	2,5	0	0	2,5
2016-11-22	Arvid	0	0	0	0	0	0
	Jeppe	0	0	0	0	0	0
	Mads	0	0	0	0	0	0
	Mikkel	0	0	0	0	0	0
	Simon	0	0	0	0	0	0
2016-11-23	Arvid	3	0	1	0	0	4
	Jeppe	0	0	0	0	0	0
	Mads	0	0	2	2	0	4
	Mikkel	0	0	6	0	0	6
	Simon	0	0	4	0	0	4
2016-11-24	Arvid	0	0	3	4	0	7
	Jeppe	0	0	0	7	0	7
	Mads	0	0	0	7	0	7
	Mikkel	0	0	5	2	0	7
	Simon	0	0	3,5	3,5	0	7
2016-11-25	Arvid	0	0	0	4	0	4
	Jeppe	0	0	0	3	0	3
	Mads	0	0	0	4	0	4
	Mikkel	0	0	0	4	0	4
	Simon	0	0	0	4	0	4
	Sum	22,5	59,5	62,5	48	5,5	176,5

### CDIO3 Tidsplan

Tidsskema (Gruppemedlem):	Design:	Impl.:	Test:	Dok:	Andet:	I alt:
Arvid:	8,5	25	6	8	1	48,5
Jeppe:	3,5	12,5	10	10	1	37
Mads:	3,5	9	11	16,5	1,5	41,5
Mikkel:	3,5	8,5	19,5	6	1	38,5
Simon:	3,5	7,5	16	8,5	1	36,5



# **Indholdfortegnelse**

<b>1 Abstract</b>	<b>6</b>
<b>2 Indledning</b>	<b>6</b>
<b>3 Problemformulering</b>	<b>6</b>
<b>4 Krav Specifikation</b>	<b>7</b>
<b>5 Design</b>	<b>11</b>
<b>6 Implementering</b>	<b>37</b>
6.1 Minimumskrav og installationsguide . . . . .	38
<b>7 Test</b>	<b>40</b>
7.1 System test . . . . .	43
7.2 Acceptance test . . . . .	44
<b>8 Forbedringsforslag</b>	<b>50</b>
<b>9 Konklusion</b>	<b>51</b>
<b>10 Bilag</b>	<b>52</b>
10.1 Udvidet Systemtest . . . . .	52
10.1.1 Test: S01 . . . . .	52
10.1.2 Test: S02 . . . . .	53
10.1.3 Test: S03 . . . . .	53
10.1.4 Test: S04 . . . . .	54
10.1.5 Test: S05 . . . . .	55
10.1.6 Test: S06 . . . . .	56
10.1.7 Test: S07 . . . . .	56
10.1.8 S08 . . . . .	57
10.2 KlasseDiagram1 . . . . .	57
10.3 KlasseDiagram2 . . . . .	59

# 1 Abstract

We have been tasked with creating a semi-complete Monopoly game. With 21 individual fields, various effects and up to six players, this assignment has been a bit of an undertaking. The project builds upon experience from previous projects, and even uses a lot of the same classes - with some modifications where needed.

The report shows how UML and UP has been used to design the software. This makes the process easier and saves time in the development process. The program has been written in the Object Oriented language JAVA, following the principles of GRASP and FURPS+. The program uses inheritance to create the 21 individual fields on the board, making the process much better and easier to read and compile.

Our finished program surpasses all of our previous projects, both in size, as well as level of code. The code has been written with expandability in mind, and potential for even more future improvements. This is very important for our 3-week period in January. The program itself is very user friendly and the game is fun to play.

## 2 Indledning

Vores forløb og arbejde med CDIO-projekterne har introduceret os til et stadigt mere avanceret spil. Vi startede ud med et simpelt terningespil, og skal nu skabe et næsten funktionelt Matador spil. Vores CDIO3 projekt skal indeholde 21 felter, heriblandt fem forskellige undertyper, mulighed for ejerskab, og skattekalkulation.

Vi anvender segmenter fra de tidligere projekter, bl.a. en Account klasse fra CDIO2, og en DiceCup med to Die-objekter i, fra CDIO1.

Til at skabe vores projekt, vil vi anvende en GUI, skrevet af DTU, som vi modificerer. Al spillogik og beregning står vi selv for.

Vi laver al kodning i Java, vha. Eclipse v. 4.6.1. Vi anvender GitHub til at lagre vores kode og agere som versionskontrol.

## 3 Problemformulering

Målet i dette projekt er at lave koden til et mere avanceret brætspil. Kunden forventer et brætspil for to til seks personer. Antallet af spillere skal kunne indstilles. Kunden kræver at dette produkt har flere felter end de tidligere versioner, samt at der er en decideret brugerflade. Kunden har derfor udarbejdet en liste over felterne som spillerne skal kunne lande på, samt de muligheder spillerne skal have ifb. med disse. Alle disse skal implementeres i spillet.

En af spillerne starter spillet ved at slå med terningerne. Alt efter terningernes værdier, lander spilleren på det relevante felt. Feltet kan have en effekt på spillerens pengebeholdning, og det videre spil. Der er 5 typer felter, 3 af dem kan ejes. De resterende felter har blot en effekt på

spilleren. Det skal være muligt for spilleren at købe feltet, hvis feltet kan ejes, og kræve leje af grunden af de andre spillere. Spillets GUI skal præsentere et stykke tekst når spillerne lander på et felt, som kort beskriver handlingen, og hvordan dette præcist påvirker spilleren. Efter spillerens tur er færdig, skal spillet sende turen videre til den næste spiller.

Spillerne skal starte med en pengebeholdning på 30.000, og spillet skal slutte når kun én spiller står tilbage, mens de andre er gået bankerot.

Spillet skal let kunne oversættes til andre sprog, og det skal være muligt at skifte til andre terninger.

Kunden forventer samtidig at der efterfølgende skal kunne arbejdes videre på systemet, og at systemet er grundigt testet. Der skal være dokumentation for disse tests, samt mulighed for at kunden selv kan gennemgå disse. Kunden forventer til sidst en beskrivelse af minimumskravene til systemet.

## 4 Krav Specifikation

Vi har lavet en liste over kravene til produktet, både ud fra kundens vision og de krav som vi selv har opstillet. Vi har sorteret dem efter FURPS+'s model:

- **Functionality** (Funktionalitet)
- **Usability** (Brugbarhed)
- **Reliability** (Pålidelighed)
- **Performance** (Ydelse)
- **Supportability** (Supportering)
- +
  - Design constraints (Design begrænsninger)
  - Implementation constraints (Implementations begrænsninger)
  - Interface constraints (Grænseflade begrænsninger)
  - Physical constraints (Fysiske begrænsninger)
  - Legal constraints (Lovmæssige begrænsninger)

## Funktionalitet:

- F1: Spillet skal kunne køres på DTU's Windows Databarer.
- F2: Der skal være 2-6 spillere.
  - F2.1: Spillerne skal indtaste hvor mange spillere der skal være med i starten af spillet.
- F3: Spillerne skal skrive deres navne ind i starten af spillet.
- F4: Spillerne skal starte med 30.000 hver.
- F5: Spillerne skal starte udenfor brættet og begynder deres tur fra felt nummer 1.
- F6: Spillerne skal slå med to seks sidde terninger, og gå det antal felter frem som sammen af terningerne viser.
- F7: Spilleren skal kunne lande på et felt og fortsætte derfra i næste slag.
  - F7.1: Spillerne skal have mulighed for at vælge hvad han vil gøre på feltet hvis dette er relevant.
- F8: Spillet slutter når alle på nær en spiller er bankerot.
- F9: Spillerne skal gå bankerot når deres pengebeholdning er mindre end 0.
- F10: Hvis en spiller skal betale en afgift og ikke har nok penge, går han bankerot, taber spillet og der skal ske følgende:
  - F10.1: Spilleren overfører de resterende penge til ejeren af grunden.
  - F10.2: Spillerens grunde bliver frigivet så andre spillere kan købe dem, hvis de lander på dem.
- F11: De felter der *kan* ejes kan kun ejes af *en* spiller.
- F12: Der skal være fem typer felter:
  - F12.1.: Territory:
    - \* F12.1.1: Skal kunne købes når man lander på det hvis ingen andre spillere ejer det.
    - \* F12.1.2: Hvis feltet er ejet af en anden spiller, skal spilleren der landede på feltet betale leje til ejeren.
  - F12.2: Refuge
    - \* F12.2.1: Når man lander på Refuge skal man have udbetalt en bonus.
  - F12.3: Labor camp
    - \* F12.3.1: Feltet skal kunne købes når man lander på det hvis ingen andre spillere ejer det.

- \* F12.3.2: Hvis feltet er ejet af en anden spiller skal spilleren slå med terningene og gange summen med 100. Dette tal skal så ganges med antallet af Labor Camps med den samme ejer. Summen af dette skal betales til ejeren af Labor Campen.
- F12.4: Tax
  - \* F12.4.1: Spilleren skal betale 10% af sin samlede formue eller et fast beløb.
- F12.5: Fleet
  - \* F12.5.1: Feltet skal kunne købes når man lander på det hvis ingen ejer det.
  - \* F12.5.2: Hvis feltet er ejet af en anden spiller skal spilleren betale en afgift afhængig hvor mange Fleets ejeren af feltet har. Afgiften er beskrevet herunder:
    - F12.5.2.1: 1 Fleet: 500
    - F12.5.2.2: 2 Fleets: 1000
    - F12.5.2.3: 3 Fleets: 2000
    - F12.5.2.4: 4 Fleets: 4000

<b>Feltlist</b>	<b>Felttype</b>	<b>Effect</b>	<b>Price</b>
Tribe Encampment	Territory	Rent 100	Price 1000
Crater	Territory	Rent 300	Price 1500
Mountain	Territory	Rent 500	Price 2000
Cold Desert	Territory	Rent 700	Price 3000
Black cave	Territory	Rent 1000	Price 4000
The Werewall	Territory	Rent 1300	Price 4300
Mountain village	Territory	Rent 1600	Price 4750
South Citadel	Territory	Rent 2000	Price 5000
Palace gates	Territory	Rent 2600	Price 5500
Tower	Territory	Rent 3200	Price 6000
Castle	Territory	Rent 4000	Price 8000
Walled city	Refuge	Receive 5000	NaN
Monastery	Refuge	Receive 500	NaN
Huts in the mountain	Labor camp	Pay 100 x dice	Price 2500
The pit	Labor camp	Pay 100 x dice	Price 2500
Goldmine	Tax	Pay 2000	or 10% of total assets
Caravan	Tax	Pay 4000	or 10% of total assets
Second Sail	Fleet	Pay 500-4000	Price 4000
Sea Grover	Fleet	Pay 500-4000	Price 4000
The Buccaneers	Fleet	Pay 500-4000	Price 4000
Privateer armade	Fleet	Pay 500-4000	Price 4000

## **Brugbarhed:**

- B1: Spillets GUI skal præsentere en liste over spillets regler, når spillet starter op.
  - B1.1: Disse spilregler skal være logiske og lette at forstå.
- B2: Spilleren skal hele tiden kunne se sin penge beholdning.

## **Pålidelighed:**

- P1: Programmet skal kunne håndtere forkerte brugerinputs, og give en fejlmeddeelse tilbage til brugeren.
  - P1.1: Programmet skal kunne kende forskel på korrekte og forkerte brugerinputs, og behandle dem derefter.
  - P1.2: Programmet skal håndtere forkerte brugerinputs, og bede om et nyt, korrekt, input.
- P2: Programmet skal være fair for alle, således at alle spillere har lige stor chance for at vinde. Derudover skal det være en tilfældig person som udvælges af spillet som begynder.

## **Ydelse:**

- Y1: Der må ikke være bemærkelsesværdige forsinkelser. Programmet skal maksimalt bruge 0,5 sekunder på alle beregninger ved spil.
- Y2: Programmet skal starte korrekt og virke korrekt i 99% af alle spil-kørsler på en Windows databar-maskine.

## **Supportering:**

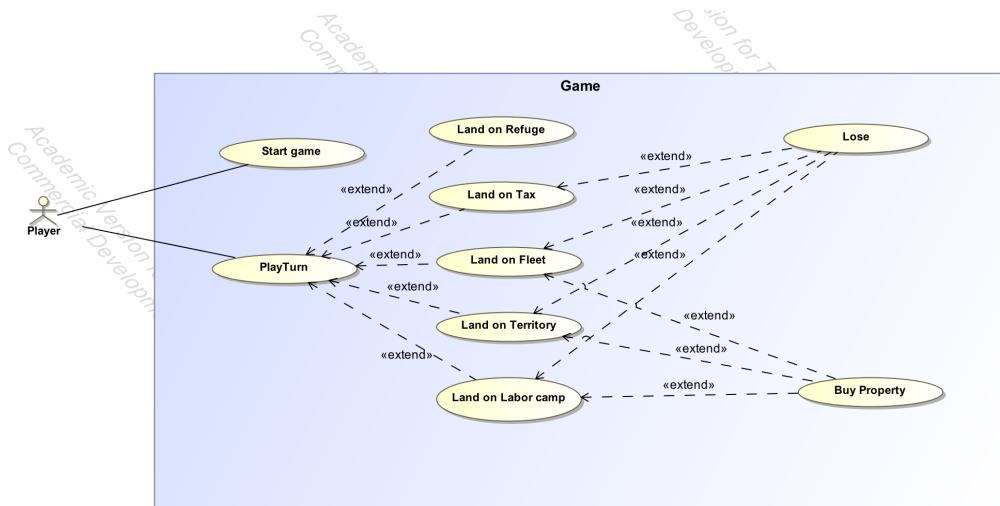
- S1: Spillet skal være nemt at oversætte.
- S2: Spillets kode skal være nemt tilgængeligt og overskueligt.
- S3: Spillets kode skal indeholde en passende mængde kommentarer således andre kan tilgå og forstå koden.

**+:**

- +1: Spillet skal kodes i Java, vil som minimum kræve en skærmopløsning på 800x600 pga. GUI'en.
- +2: Spillet skal ikke fylde mere end 50MB på en hukommelsesenhed.

## 5 Design

### Use-case diagram



Figur 3: Use-Case diagram for programmet.

### Use-case beskrivelser

Vi har valgt kun at lave en Fully dressed usecase. Denne Fully dressed usecase er for land on fleet. Vi har dog lavet beskrivelser af samtlige usecase. Alle beskrivelserne er på engelsk da vi har lavet design processen på engelsk.

#### Use-case: Play turn.

1. Player presses “roll dice” and the round starts.

#### Use-case: Buy field.

1. Player tries to buy field.
  - (a) If the player has enough money he buys the field.
  - (b) Otherwise the turn ends.

#### Use-case: LaborCamp.

1. Player landed on Labor camp field.
  - (a) If the Labor camp is owned by another player.

- i. The players rolls the dice and have to pay 100 times the amount he rolled.
  - A. If the player can afford it he pays the amount to the owner.
  - B. Otherwise he loses.
- (b) If the labor camp is not owned, the player can buy it if he wants to and has enough money.

### **Use-case: Refuge.**

1. Player landed on Refuge field.
2. The player gets a bonus.

### **Use-case: Tax.**

1. Player landed on a Tax field.
2. The player now has two options.
  - (a) Pay 10% of the players own fortune.
  - (b) Pay a specific amount depending on the tax field.
3. If the player can't afford any of the two options, the players loses and is out of the game.

### **Use-case: Territory.**

1. Player landed on Territory field.
2. If the Territory is owned by another player.
  - (a) The player has to pay rent to the owner.
  - (b) If he can't pay he loses.
3. If the territory is not owned, the player can buy it if he wants to and has enough money.

### **Use-case: Lose**

1. The player pays his remaining money to whom he owes.
  - (a) If the player lands on tax, the players character leaves the board.
  - (b) If the player landed on a Fleet, Territory or Labor Camp, an amount of money from the players funds is given to the owner of that field.
2. The paying - and broke - player then release all of their properties, so other players may buy them again.

3. The player has now lost the game, and is no longer participating.

### **Use-case: Fleet**

Følgende er en fully dressed use case for fleet.

**Primary Actor:** Player.

**Preconditions:** The player has rolled the dice and landed on a fleet field.

**Success Guarantee (Postconditions):** The player either pay rent, buy the field or chose to do nothing.

#### **Main Success Scenario:**

1. Player lands on a Fleet field.
2. If the fleet is owned by another player, the player has to pay rent to that player.
  - (a) If the player can afford to pay rent, he has to pay the following:
    - i. If the owner owns only one fleet, the players pays 500.
    - ii. If the owner has two fleets, the players pays 1000.
    - iii. If the owner has three fleets, the players pays 2000.
    - iv. If the owner has all four fleets, the players pays 4000.
  - (b) The players turn is over.

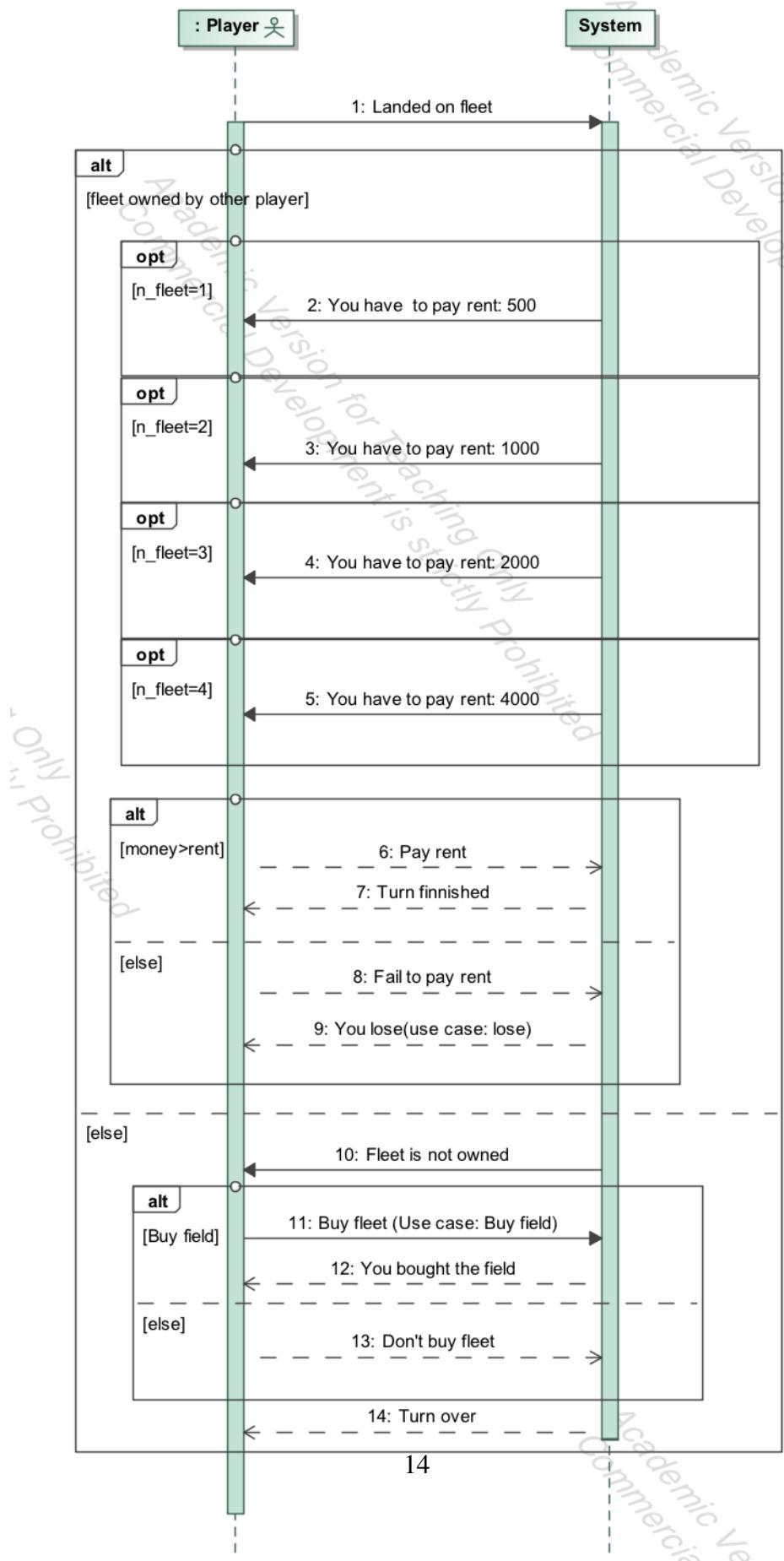
#### **Alternative Flows:**

1. (a) Player lands on a Fleet field.
  - (b) If the fleet is owned by another player, the player has to pay rent to that player.
  - (c) If the player can't afford to pay the rent, he loses.
2. (a) Player lands on a Fleet field.
  - (b) If the fleet is not owned by another player. The player can buy it.
  - (c) If the player wants to buy it, he has to pay the price of the property.
3. (a) Player lands on a Fleet field.
  - (b) If the fleet is not owned by another player. The player can choose to buy it.
  - (c) If the player doesn't want to buy the field, the players turn is over.

#### **Special Requirements:**

The text that describes what happens needs to be understandable, so that the user understands what is happening.

## System Sekvensdiagram

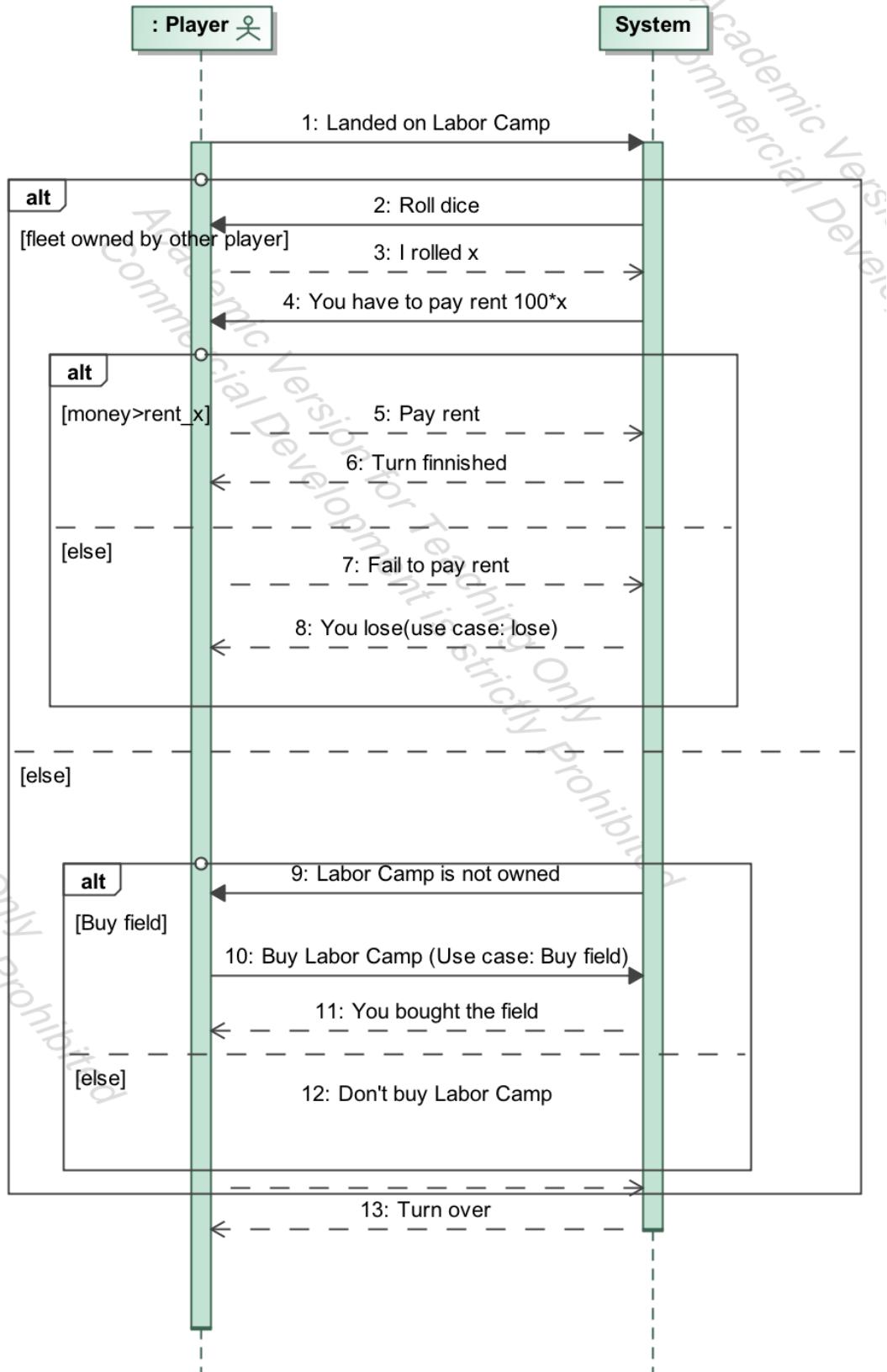


Figur 4: System sekvensdiagram for land on fleet field

**Figur 4 beskrivelse:**

Spilleren har slået med terningerne og er landet på et Fleet felt. Hvis Fleet feltet er ejet af en anden spiller, skal han betale et beløb til den spiller, afhængig af hvor mange Fleet felter han ejer. Han betaler 500, 1000, 2000 eller 4000 for henholdsvis et, to, tre eller fire ejet Fleet felter. Hvis han har råd til at betale, betaler han lejen, og slutter hans tur. Hvis han ikke har råd så taber han spillet.

Hvis Fleet feltet ikke er ejet kan spilleren vælge at købe den.

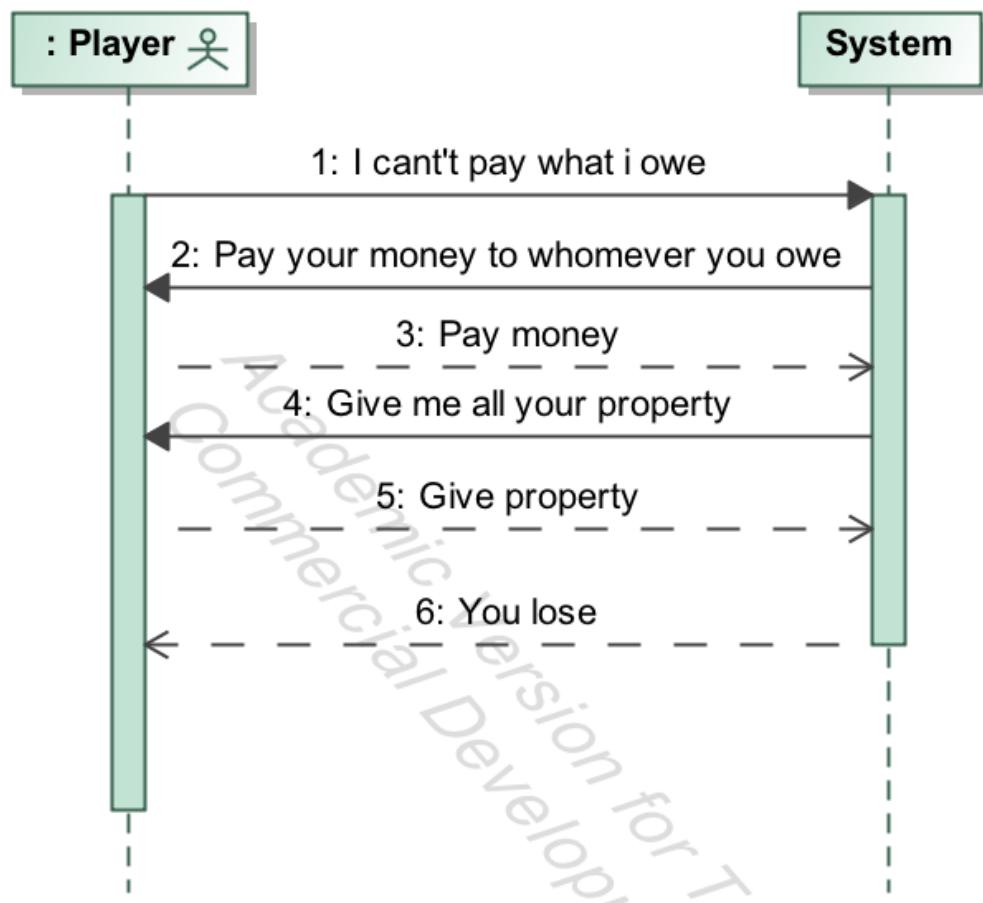


Figur 5: System Sekvendiagramm for land on Labor Camp field

### Figur 5 beskrivelse:

Spilleren lander på Labor camp. Hvis den er ejet af en anden spiller skal han slå med terningerne og betale 100 gange terningernes værdi til ejeren. Hvis ejeren har to Labor Camps skal spilleren betale det dobbelte til ejeren. Har spilleren ikke råd til at betale, taber han.

Spilleren kan købe grunden hvis den ikke er ejet.



Figur 6: System Sekvensdiagram for Lose.

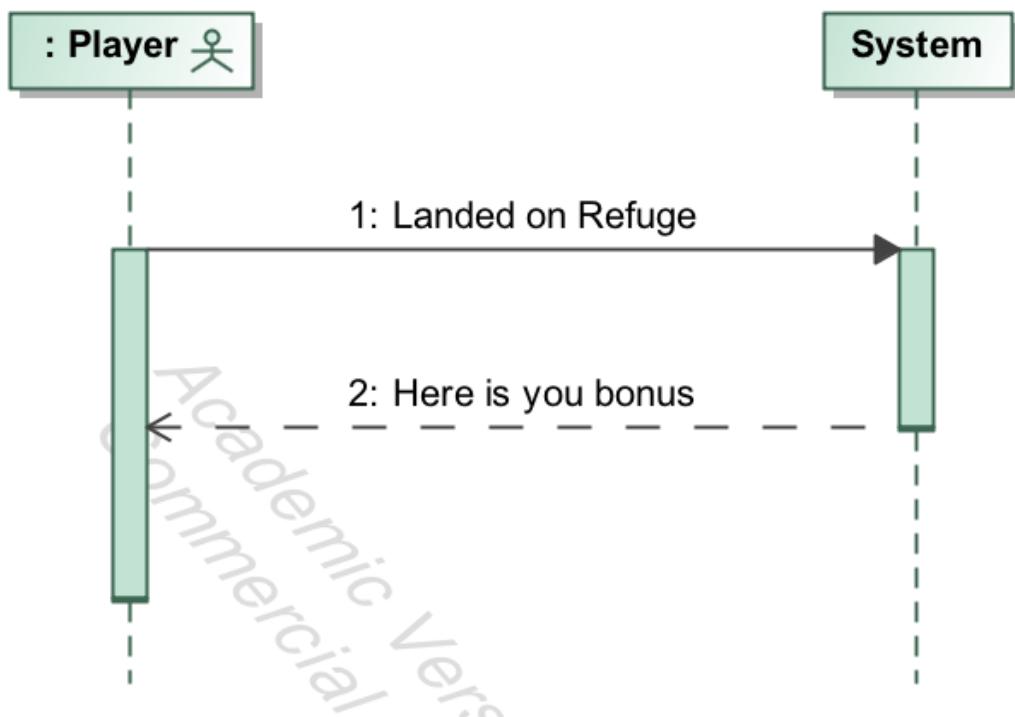
### Figur 6 beskrivelse:

Denne use-case beskriver forløbet hvis en spiller ikke har penge nok til at betale den gæld vedkommende skylder. Spilleren fortæller spillet 'Jeg kan ikke betale hele den gæld jeg skylder'.

Spillet svarer igen i en besked til spilleren at spilleren skal betale den mængde penge som spilleren ejer. Det bemærkes her at spillerens pengemængde er mindre end det beløb vedkommende skylder.

Spillerens resterende pengebeholdning overføres til ejeren af feltet, og spillerens pengebeholdning når 0. Fordi spilleren skylder flere penge end vedkommende kan betale, giver spillet spilleren besked på at spilleren skal overdrage samtlige ejendomme tilbage til spilbrættet.

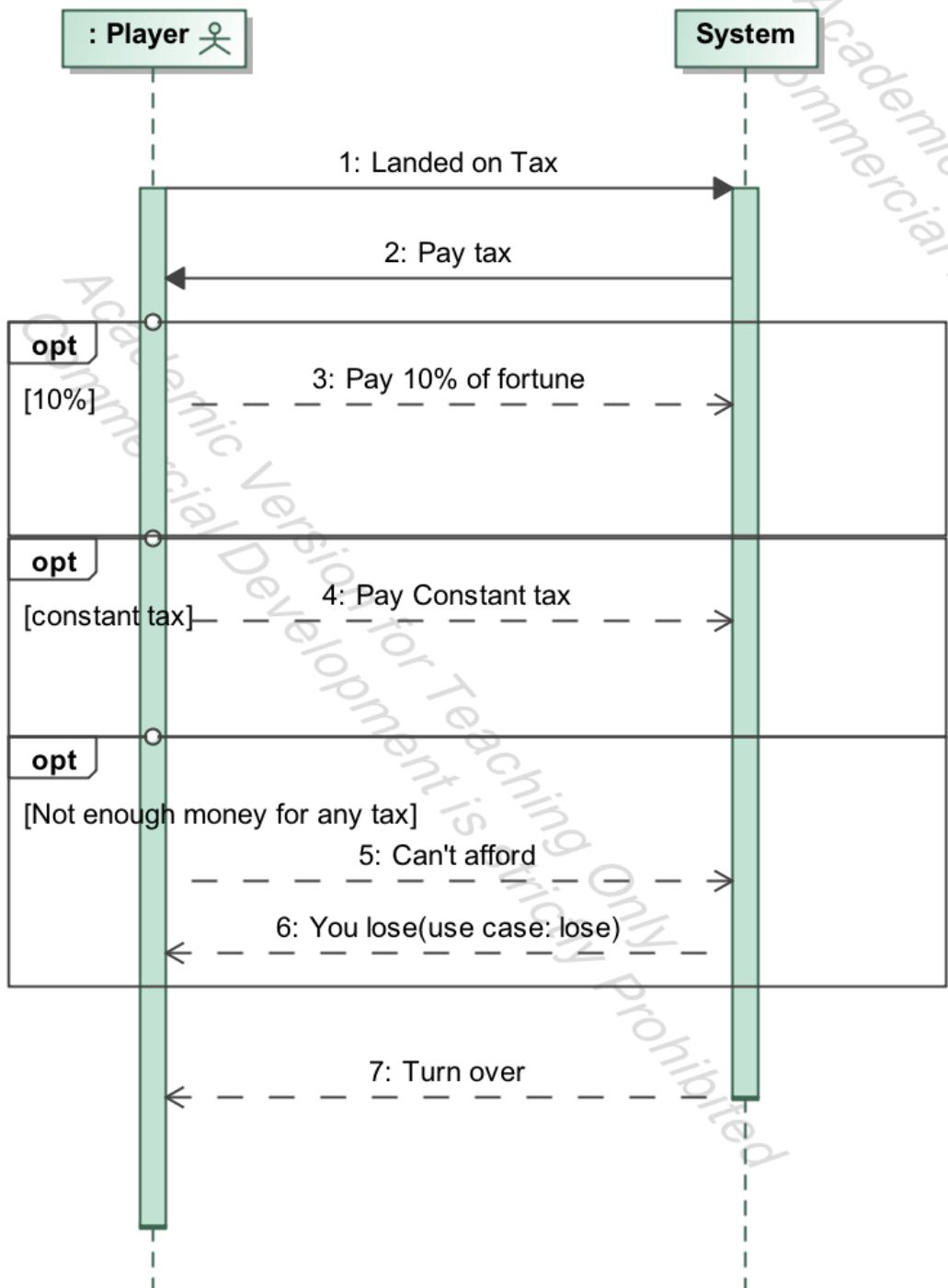
Spilleren overfører samtlige ejendomme tilbage spilbrættet, hvor grundene sættes til salg igen. Spilleren udgår fra spillet.



Figur 7: System sekvendiagram for land on refuge field

#### **Figur 7 beskrivelse:**

Dette use-case diagram beskriver det korte forløb, når en spiller lander på et 'Refuge' felt. Spillet registrerer hvilket et af de to Refuge-felter som spilleren har landet på, og tildeler spilleren en tilførsel af penge. I diagrammet er det beskrevet som en 'bonus' til spillerens pengebeholdning, da der ikke forekommer andre påvirkninger på spilleren.



Figur 8: System Sekvendiagram for land on tax field

#### Figur 8 beskrivelse:

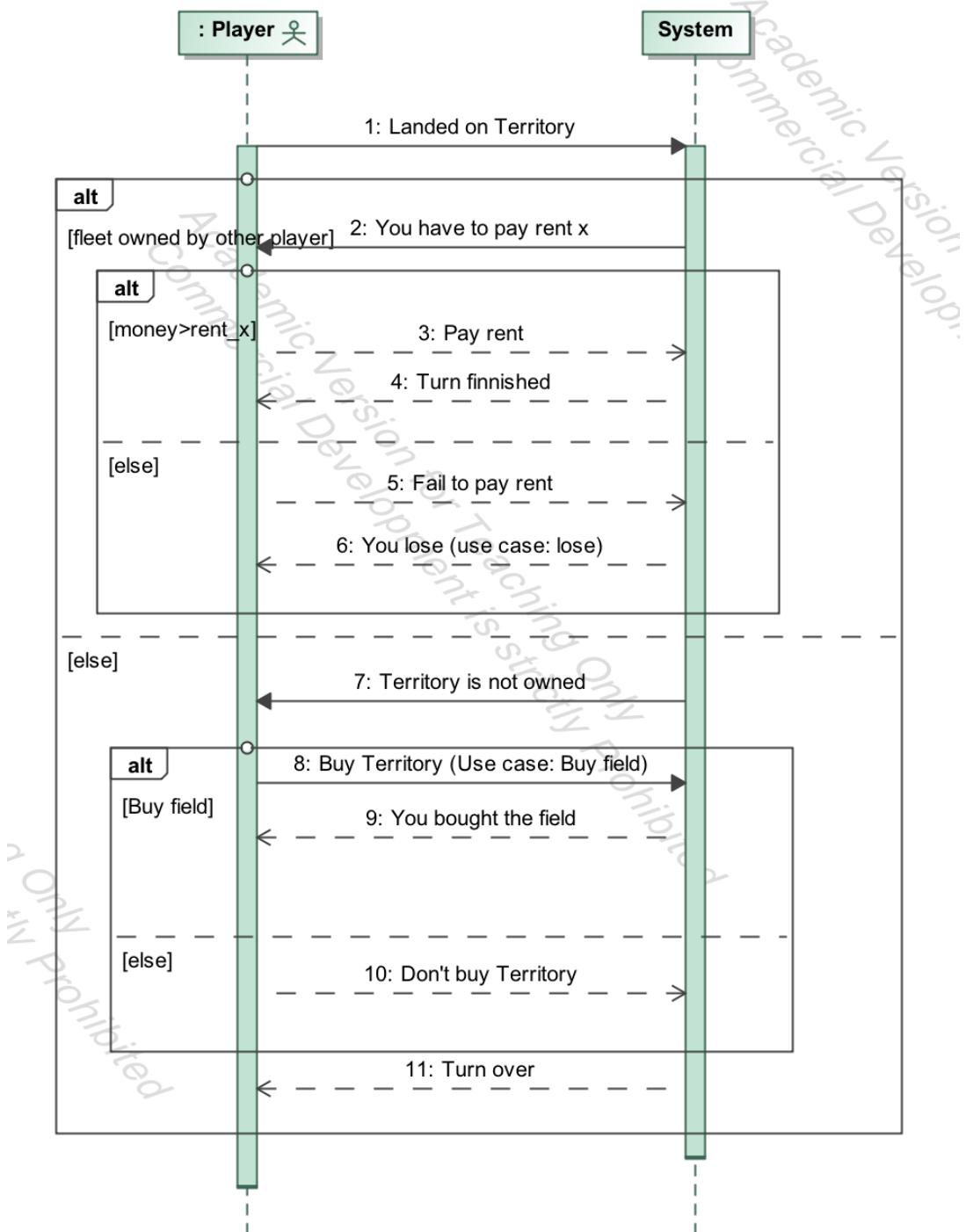
Dette diagram beskriver forløbet når en spiller lander på et 'Tax' felt. Alt efter spillerens valg, og spillerens daværende pengebeholdning, håndterer spillet spillets videre forløb.

I det første scenario vælger spilleren at betale 10% af sin formue. Spillet beregner en værdi svarrende til 10% af spillerens formue, og fratrækker dette fra spillerens pengebeholdning.

Spillet sender derefter turen videre til den næste spiller.

I det andet scenario vælger spilleren at betale en i forvejen fastsat pris for at lande på 'Tax'-feltet. Det nøjagtige beløb bestemmes ud fra hvilket et af Tax-felterne som spilleren lander på. Dette beløb fratrækkes spillerens pengebeholdning, og spillet sender turen videre til den næste spiller.

I det tredje og sidste scenario har spilleren ikke en stor nok pengebeholdning til at betale det fastsatte beløb, eller 10% af sin formue, i.e., spillerens pengebeholdning er 0. Spilleren fortæller spillet at vedkommende ikke kan betale beløbet. Spillet arbejder derefter ud fra samme use-case, som hvis spilleren skyldte flere penge, end vedkommende kan betale - Se use-case: Lose. Spilleren ud går fra spillet, og spillet sender turen videre til den næste spiller.



Figur 9: System sekvensdiagram for Land on Territory Field

### Figur 9 beskrivelse:

SSD diagrammet for at lande på et 'Territory' kan beskrives i detaljer således:

Overordnet set er der to hovedmuligheder:

- Enten er territoriet ejet af en anden spiller.

- Eller også er territoriet *ikke* ejet af en anden spiller.

Hvis spiller A lander på et Territory ejet af en anden spiller, spiller B, kan der ske to ting.

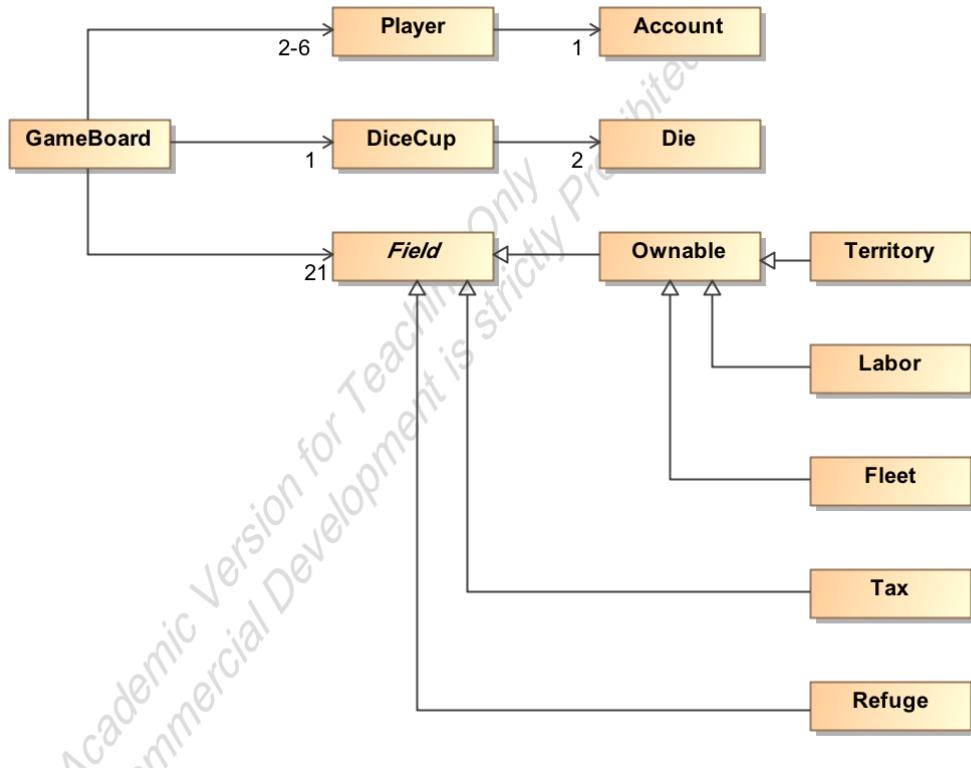
Enten så betaler spiller A for leje af grunden, en overførsel af penge fra den ene pengebeholdning til den anden, og turen afsluttes.

Hvis spiller A ikke kan betale lejen på B's Territory, skal spiller A overdrage sin resterende pengebeholdning til Territory ejer, spiller B. Præcis på samme fremgangsmåde som i den tidligere use-case: Lose. Spiller A ud går fra spillet, og spillet sætter spiller A's ejendomme til salg. Turen går videre til den næste spiller.

Spiller A lander på et Territory som ikke har nogen ejer, har vedkommende to muligheder. De kan enten vælge at købe grunden. I så fald tilføjes grunden til spillerens liste af grunde, spillerens pengepulje fratrækkes grundens pris og grunden sættes til ikke længere at være 'Ownable'. Spiller A kan nu opkræve leje for sin grund.

Spiller A kan dog også vælge at lade være med at købe grunden, og lade turen gå videre til den næste spiller.

## Domænemodel



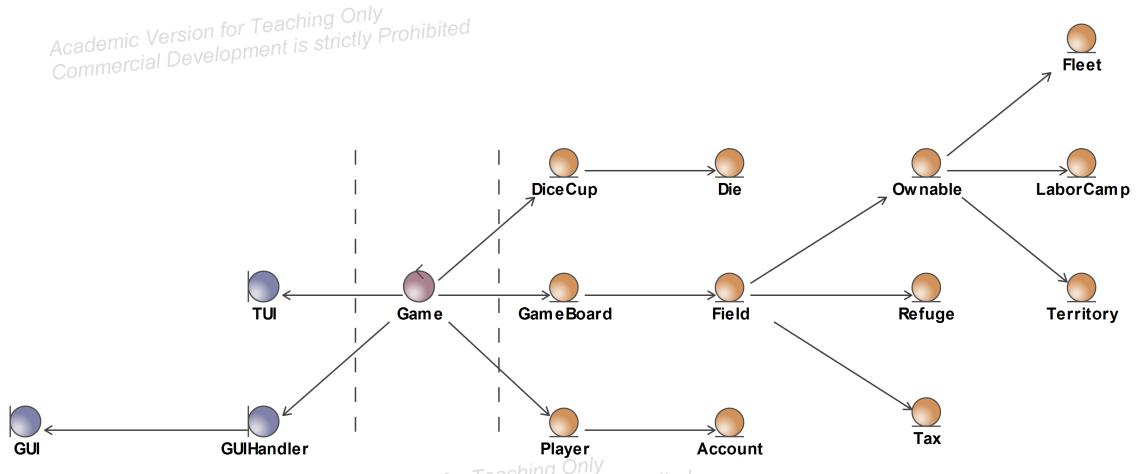
Figur 10: Domæne model

Vi har her modelleret hvordan brætspillet kunne se ud i virkeligheden. Vi har et spillebræt, og på det bræt har vi fem felter af forskellige typer. To af disse er bare ordinære felter, der har en

effekt på spilleren når spilleren lander på dem. De tre andre er felter som kan ejes. Spilleren kan altså købe disse felter og andre spillere der lander på dem skal så betale spilleren penge. Vi opretter desuden et par Dice, som vi forestiller os ligger i en digital DiceCup. Til slut opretter vi selvfølgelig også et par spillere, og vi forestiller os at de hver især har en Account-kasse, hvor deres pengebeholdning ligger.

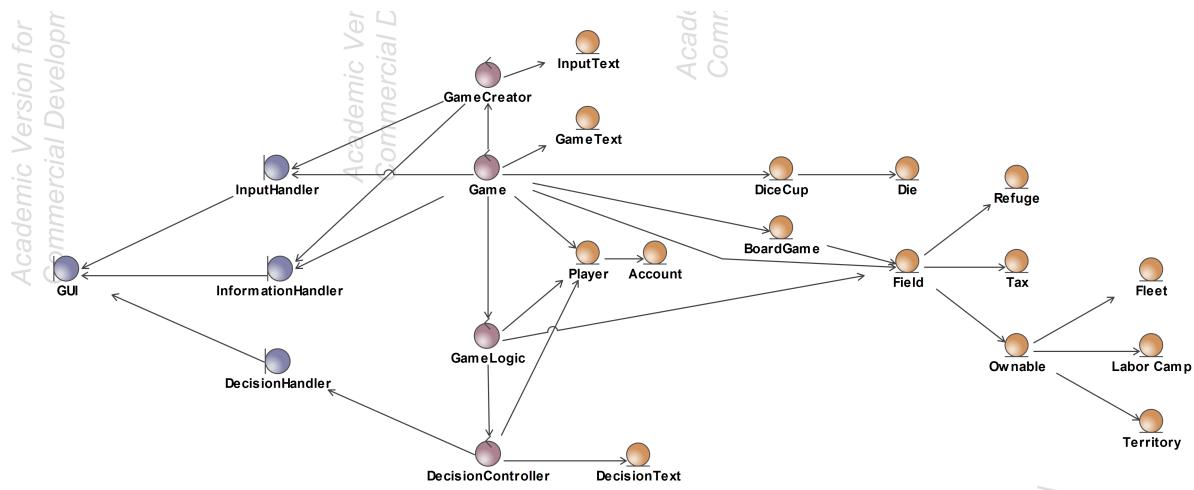
## BCE-model

Vi har lavet to iterationer af BCE modellen. Dette har vi gjort da vi ikke mente den første var god nok. Dette er den originale model.



Figur 11: Original BCE model

Originalt besluttede vi os for kun at have en Controller. Det var Game der styrede det hele. Men det medførte at både Game og GUIHandleren blev meget store og til dels ikke overholdt GRASP. Vi valgte derfor at dele ansvaret lidt ud. Den nye model endte med at se sådan ud:



Figur 12: Nye BCE model

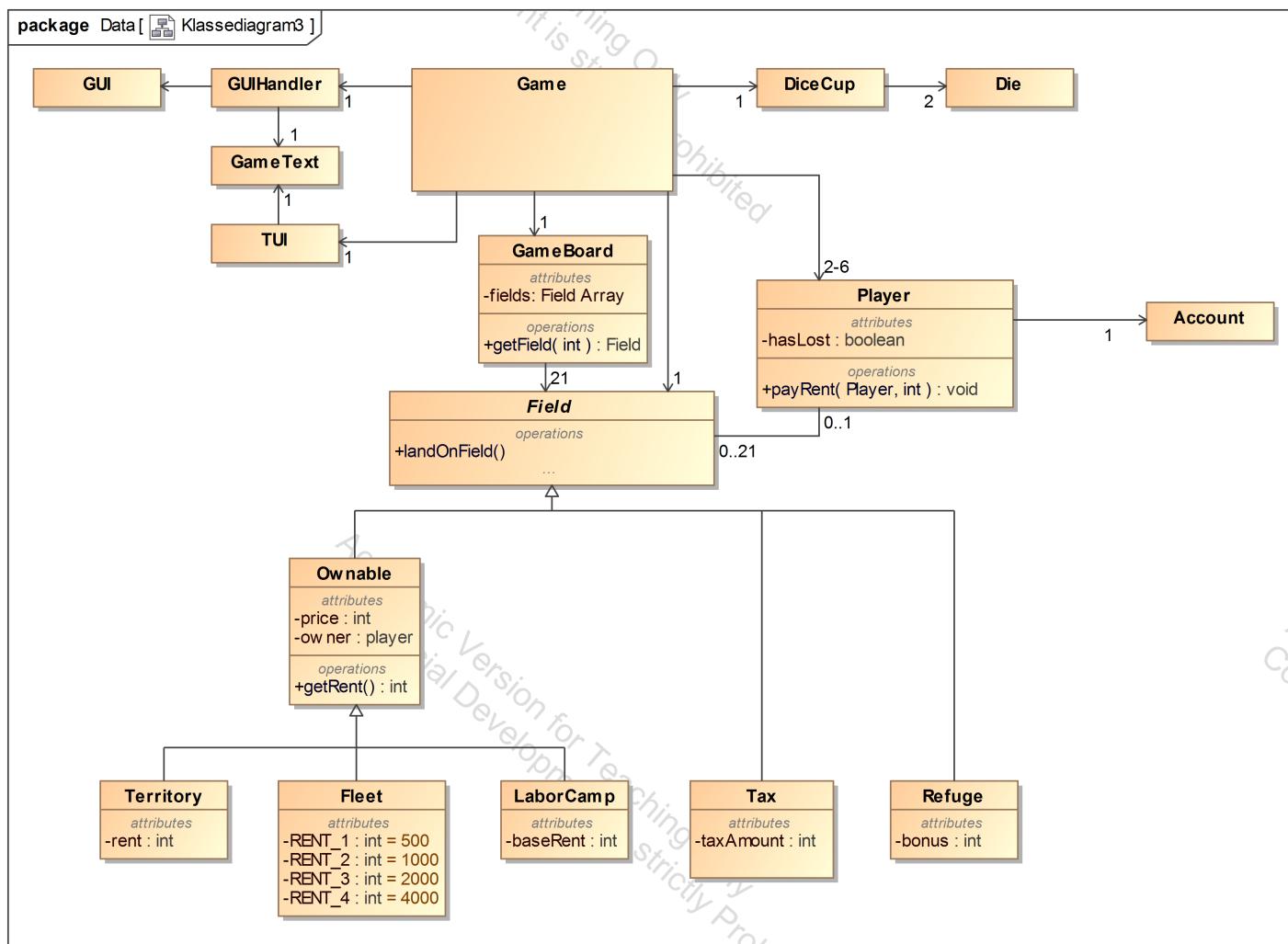
Det kan ses her at det er delt lidt mere op. Det betyder desværre os at vi har fået lidt højere kobling, men vi mener det har resulteret i en lidt højere Cohesion også. Det har desuden gjort især GUIHandler meget mere overskuelig, når dens ansvar er lagt ud på 3 forskellige klasser. Game er desuden blevet delt op i fire Controllers.

GameCreator der har ansvaret for at oprette spillet, og hente spillernavne. Game selv har ansvaret for basal information til spilleren. Det inkluderer hvor de er landet for eksempel.

GameLogic styre hvad der sker når en spiller lander på et felt. GameLogic kan desuden kalde DecisionController, hvis den skal bruge noget information fra spilleren. f.eks. hvis den vil vide om spilleren vil købe en grund.

## **Klassediagram.**

I projektet er der blevet itereret en del, derfor er her vist et originalt klasse diagram fra analyse fasen uden metoder. Det kan give et overblik over den generelle funktionalitet. Vi har lavet to klasse diagrammer før dette. De kan findes i bilagene, på figur 38 og figur 39.



Figur 13: Originalt klasse diagram fra analyse fasen. Her uden metoder.

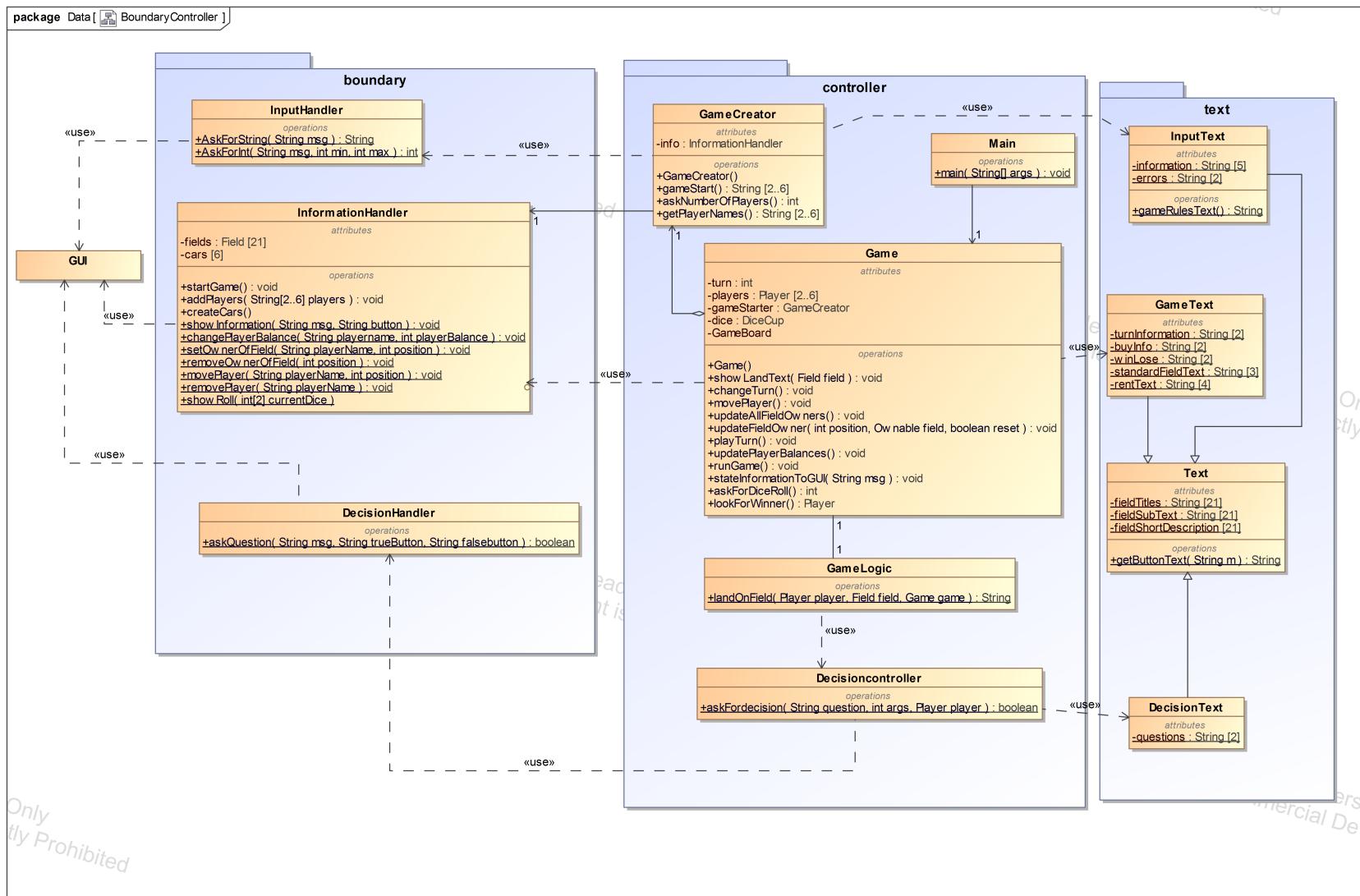
Det kan her ses hvordan de forskellige objekter hænger sammen. Det vigtigste er at kigge på Boundary og Controller delen, da det er her der blevet lavet mange ændringer.

Blandt andet har vi kun én Controller som vi har kaldt Game. Det gjorde at game havde meget ansvar og klassen blev dermed meget stor. Derfor bliver den senere delt op. Det samme gælder GUIHandleren. Den har alt for meget Information og alt for mange metoder i denne udgave af diagrammet.

Det er ikke helt tydeligt i diagrammet, men det blev klart i koden. Vi har derfor valgt at splitte Game op i flere Handlers. Vi har desuden valgt at tage TUIens funktioner og implementere i GUI. Dermed er vi kommet frem til et nyt klasse diagram. Det giver os en lidt højere kobling, tilgengæld får vi også en højere - og bedre - cohesion.

Det skal siges at der mangler en association mellem Player og Player selv. Dette gælder både for dette og næste diagram.

Da klasse diagrammet er så stort, har vi valgt at lave flere små diagrammer. Den fulde udgave kan findes på Github i .PNG format hvis man vil se hele sammenhængen. Da klassediagrammet er blevet ændret markant fra analyse fasen, beskrives her de største ændringer, hvilket ligger i Boundary og Controller området:



Figur 14: Diagrammet viser hvordan Boundary og Controller kommunikere.

Overstående diagrammer viser hvordan pakkerne Boundary, Controller og Text bruger hinanden.

Ideen er at Boundary pakken bruges til at kommunikere med GUI'en. Der er tre klasser i Boundary pakken.

InputHandleren er den klasse der har ansvaret for at bede om input fra brugeren. Den har to metoder, en der beder om en Integer, og en der beder om en String.

Pakken Text indeholder alt det tekster der i spillet. Controllerne bruger dem til at hente den tekst der matcher det der skal skrives ud.

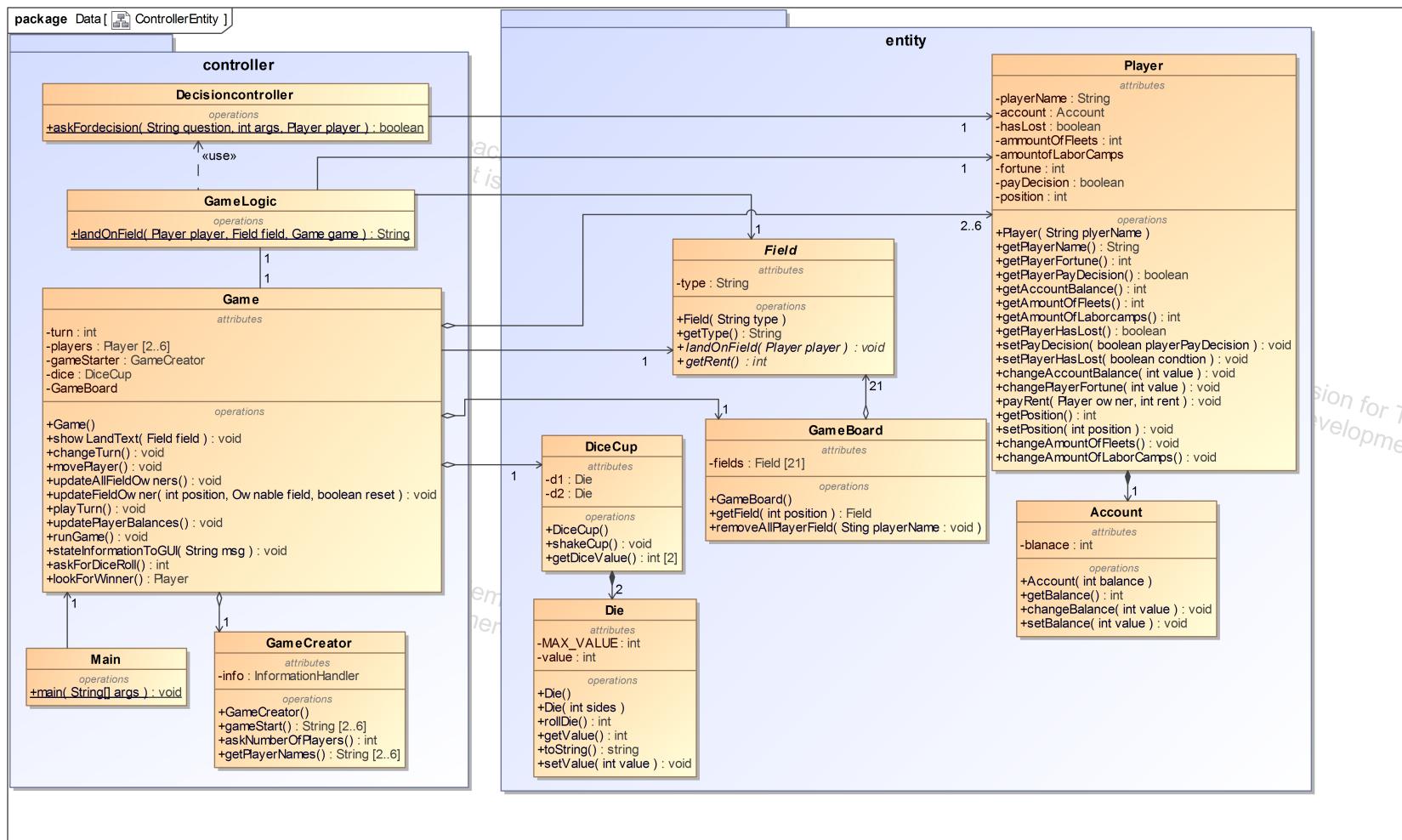
InputText, GameText og DecisionText arver alle sammen fra Text klassen. Denne indeholder felternes tekster og tekster til knapper. De andre tekster klasser er specialiserede i en bestemt type tekst. Eksempelvis bruges InputText når brugeren skal indtaste et input. Al tekst der relaterer til det ligger i InputText.

Til sidst har vi Controller pakken. Den indeholder alle spillets Controllers. Main Controlleren eneste funktion er at oprette et game objekt og kalde metoden runGame().

GameCreator formål er at oprette GUI og hente spillernes navne via InputHandleren. Den sender dernæst spillernavnene tilbage til Game.

Game styrer spillet generelt. Den printer alt hvad der sker og sørger for at rykker spillerene efter hvad de slår.

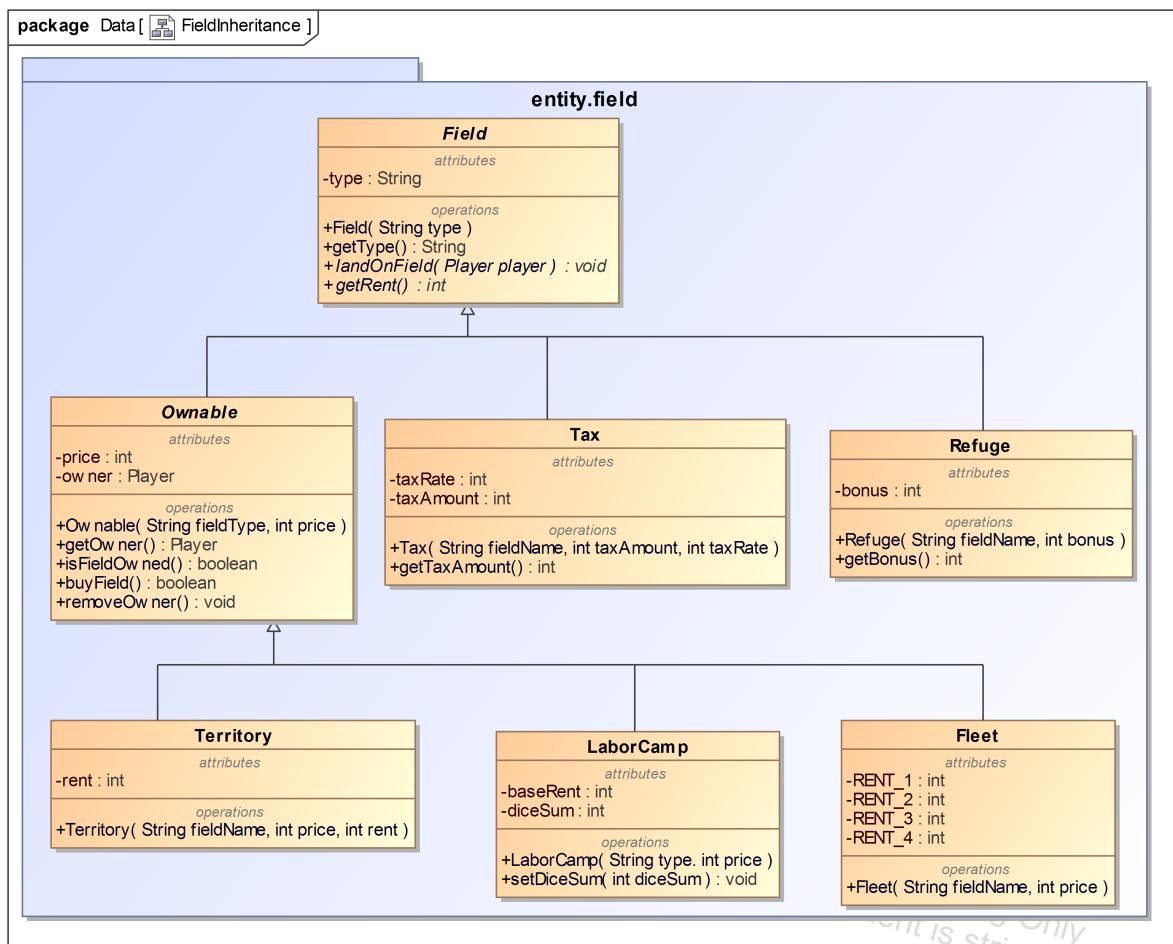
Gamelogic har ansvaret for at udføre felternes funktioner. Det vil sige købe grunde, og betale leje(rent). Hvis den har brug for beslutninger fra Spilleren, kalder den Decision Controlleren som gennem DecisionHandleren stiller spørgsmål til spilleren på GUI.



Figur 15: Diagrammet viser hvordan Controllere og entity kommunikere.

Det kan ses her hvordan vi har implementeret de forskellige Entities. Vi kan se hvordan game har alle Entities, mens de andre Controllere bruger de objekter de får af Game eventuelt gennem andre controllerer. f. eks. giver game GameLogic en Player og et Field som den behandler. Player og DiceCup har begge en komposit klasse. Player har en Account som kun den kan tilgå, og man skal gå igennem Player for at ændre på Account. Tilsvarende har DiceCup to terninger, som kun kan bruges gennem DiceCup.

Field i dette diagram er en abstrakt klasse, dvs at man ikke kan oprette objekter af typer Field, i virkeligheden er Field en masse forskellige felter der arver fra Field. Arv betyder at Klasserne arver metoder og instansvariabler fra klassen som de ”extender” fra. Til gengæld kan man override metoder, det vil sige at man ændre på metoden i klassen som har arvet den. Dermed kan man kalde den samme metode på objekterne, men hvor de har forskellige resultat på objekter af forskellige typer. Hvilke felter der arver fra Field kan vi se på understående diagram:



Figur 16: Diagrammet viser hvordan klasserne arver fra Field

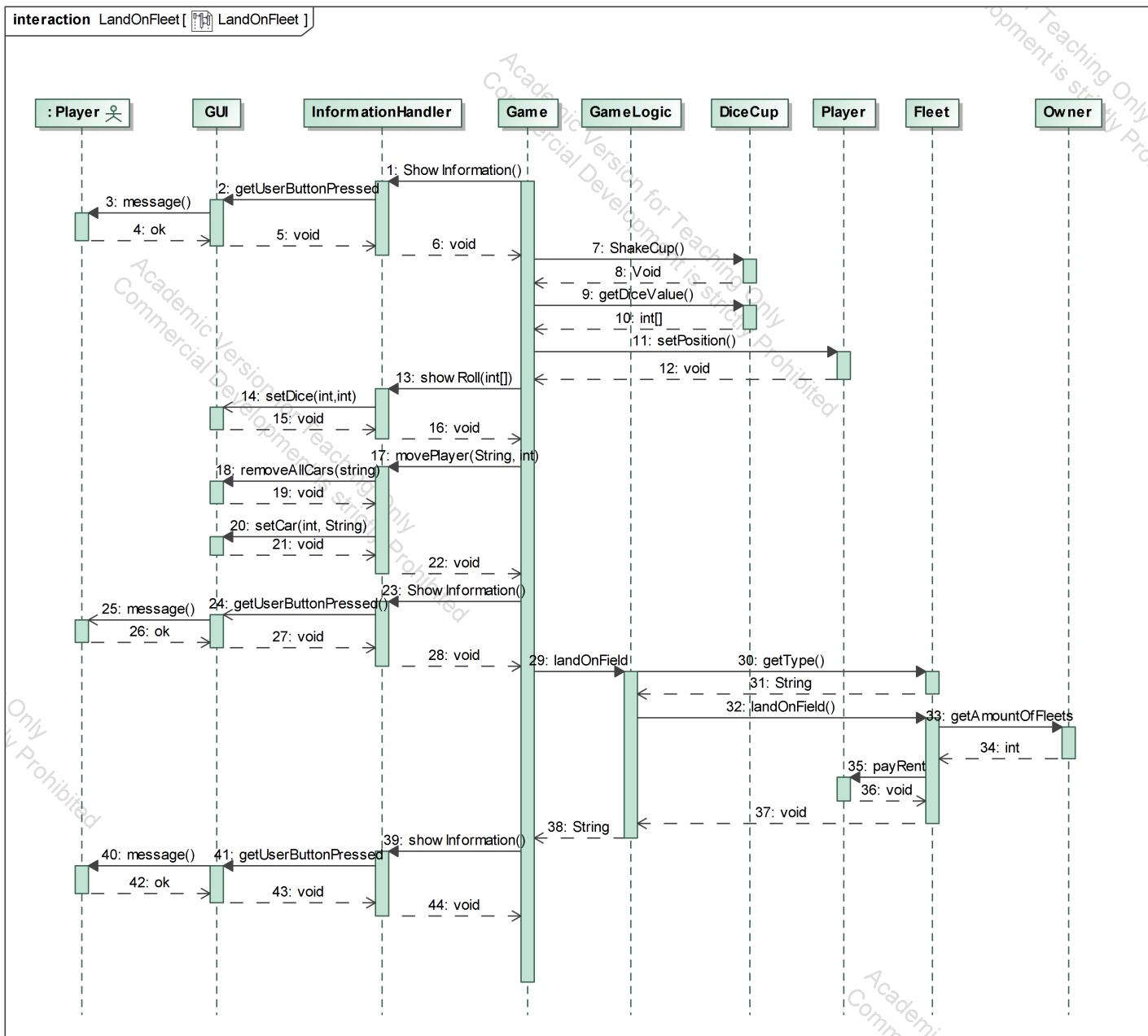
Her kan det ses at Field har en type, som beskriver hvilket slags felt den er. På den måde slipper man for at bruge instanceof(), hvilket man godt vil undgå. Fælles for alle felterne er landOnField metoden. Denne udfører forskellige ting, afhængig af hvilket felt der er tale om. Som udgangspunkt er det dog altid noget med enten at betale penge eller modtage penge.

`getRent()` metoden henter den mængde penge man skal betale. `Ownable` har desuden en `Owner`, og en metode `buyField()` der gør det muligt at købe feltet `Labor Camp`. `Labor Camp` har en speciel metode, `setDiceSum`, der indeholder hvad spilleren slår efter de er landet på feltet. Denne værdi bruges til at udregne hvad de skal betale til ejeren af feltet. Denne kaldes som det først i `landOnField`, hvis `Labor Campen` *er ejet*.

## **Design Sekvensdiagram**

Vi har valgt kun at lave ét Design sekvens diagram, her for landOnFleet.





Figur 17: Diagrammet viser hvordan landOnFleet use casen bliver udført i spillet.

Det kan ses her at spilleren starter sin tur. Dernæst trykker han ”ok” for at slå. Så kalder Game en DiceCup, der slår med terningerne og henter terningeslagene frem. Spillerens position på brættet opdateres, så spilleren kan se hvor han er landet. Dernæst får han en besked om hvad det er for et felt han er landet på, og hvad der ske på det her felt.

Da der er tale om et Fleet felt, får han at vide at han skal betale et beløb til spilleren der ejer Fleeten. Denne pris er afhængig af hvor mange Fleets ejeren af feltet ejer.

Når spilleren trykker ok, kalder game gamelogic. Denne klasse sørger for at betale spilleren. Den henter feltypen, og dernæst kalder den metoden landOnField på feltet. Metoden sørger for at spilleren betaler sin leje til ejeren og hans tur er slut.

## 6 Implementering

### Kodestruktur

Java-projektet er struktureret efter BCE-modellen, hvilket vil sige at de forskellige klasser er opdelt i forskellige pakker, Entities, Controllers og Boundaries. Dette betyder at man let kan udskifte dele af koden (en eller flere klasser), uden at skulle ændre i de øvrige klasser. Eksempelvis kan man udskifte spillets logik (GameLogic-klassen) med en anden spil logik. Dette kunne f.eks. være, hvis man ønskede at spillet skulle have nogle andre spilleregler.

Klasserne i Entity-pakkerne er de klasser der repræsenterer virkelige objekter (f.eks. GameBoard-klassen). Disse klasser indeholder data om objekter, og metoder, der håndterer data og gør det let at få adgang til disse. Derudover er der en entity pakke ’text’, som indeholder forskellige klasser med spillets tekster. Dette er gjort således, så spillets tekster let kan redigeres (både skifte sprog, eller ændre indhold).

Klasserne i Controller-pakken er de klasser der arbejder logisk med dataene (f.eks. GameLogic, som styrer spillets regler).

Klasserne i Boundary-pakken arbejder med den udleverede GUI. Hvilket vil sige, at klasserne står for at opdatere de grafiske elementer i bruger-interfacet (f.eks står InformationHandler blandt andet for at oprette spillebrættet grafisk).

### GRASP

I forrige afsnit blev der beskrevet hvordan koden blev struktureret efter BCE-modellen og hvordan dele af koden på baggrund af dette let kunne udskiftes. En anden udviklingsmetode, som har hjulpet med til dette er GRASP.

GRASP (General Responsibility Assignment Software Patterns), der hænger meget sammen med BCE-modellen, er en måde at dele koden op i ansvarsområder. Blandt andet styres ’boundaries’ og ’entities’ af flere ’controllers’. Hver controller har sit eget ansvar. Det vil sige, at de har forskellige opgaver. De fleste af ’controllers’ i programmet er, ude over at være ’controller’, også både ’information experts’ og ’creators’. At være ’creator’ betyder blandt andet at have ansvaret for at oprette et eller flere objekter. F.eks. opretter GameBoard 21 Field objekter. Ved at uddelegerer ansvaret i koden til forskellige klasser, kan man opnå lav kobling (low

coupling). Dette betyder for programmet, at det er meget lettere at tilføje, fjerne og redigere features. I selve koden betyder lav kobling, at når noget skal ændres i f.eks. Player-klassen, så skal det kun ændres i selve klassen, hvor hvis koblingen havde været høj, så havde man også skulle ændre i alle de klasser, som afhæng af Player-klassen. Med lav kobling kan vi altså ændre i Player-klassen, uden at skabe store konflikter andre steder i koden. Selvom koden er skrevet med det formål, at have lav kobling har man alligevel været nødt til at tilføje en ekstra kobling et par steder, blandt andet mellem Game og Field, fordi Game bruger Field-objekterne så ofte. Dette gør at man kan undgå at skulle tilgå felterne igennem GameBoard, hvilket ville føre til meget besvær. Ovenstående er alt sammen med til at opretholde high-cohesion, hvilket betyder at hver enkelt klasse er fokuseret, håndterbar og forståelig.

## 6.1 Minimumskrav og installationsguide

Spillet kan med sin meget simple grafiske brugerflade køre på størstedelen af moderne computersystemer. Nedenfor er en liste over hardware- og softwarespecifikationerne, som kan kører spillet 100%:

- **Hardware:**

- Systemets interne hardware skal have en regnekraft sammenlignelig med computerne i DTU's Databarer.
- Tastatur.
- Skærm m. minimumsopløsning på 800x600.
- Computermus.

- **Software:**

- Styresystem:
  - \* Windows 8.1 eller nyere.
- Softwarepakker:
  - \* Eclipse 4.6.1
  - \* Java 8.0 - <https://www.java.com/en/download/>

Minimumskravet at spillet kan køre på Windows er opstillet af den grund at det er det samme styresystem som DTU's Databarer kører med. Spillet er derfor ikke blevet testet på OSX eller en Linux Distro.

## **Installationsguide (vha. GitHub og Eclipse, og kørsel via .bat fil):**

Installationsguiden beskriver hvordan programmet indlæses fra GitHub til Eclipse's IDE, samt hvordan man kan starte programmet vha. den medfølgende .bat fil.

- **Eclipse og GitHub:**

For at indlæse, gennemse og teste programmets kode kræves der en IDE. Man har i dette projekt anvendt Eclipse Neon 4.6.1, som kan hentes på:

<http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/neon1a>

Koden kan læses på flere andre IDE'er, men som udgangspunkt vil denne guide antage at Eclipse anvendes. For at hente repositoriet, skal du gå ind på:

[https://github.com/ArvidLangsoe/33\\_CDIO3.](https://github.com/ArvidLangsoe/33_CDIO3)

Følg derefter instruktionerne nedenfor:

- **Trin 1:** Tryk på den store grønne knap i midten til højre af skærmen, med teksten 'Clone or Download', og tryk igen på det lille 'Copy to Clipboard' ikon.
- **Trin 2:** Start dit Eclipse program, og tryk på feltet der hedder 'File', som typisk ligger i øverste venstre side af skærmen.
- **Trin 3:** Vælg 'Import'. Dette åbner et vindue, med et søgefelt. Indtast 'GIT' i feltet, og vælg 'Projects From Git'. Alternativt, kan man også finde valgmuligheden inde under 'Git' folderen i vinduet.
- **Trin 4:** Tryk på 'Next', og marker derefter valgmuligheden 'Clone URI'. Da vi *har* kopieret linket i **Trin 1**, skulle Eclipse indsætte de korrekte links i felterne. Såfremt Eclipse ikke gør dette, skal der stå:
  - URL: [https://github.com/ArvidLangsoe/33\\_CDIO3.git](https://github.com/ArvidLangsoe/33_CDIO3.git)
  - Host: GitHub.com
  - Repository Path: /ArvidLangsoe/33\_CDIO3.git
- **Trin 5:** Indtast brugernavn (email) og kodeord til din GitHub bruger. Tryk derefter på 'Next'.
- **Trin 6:** Marker alle branches, 'HEAD', 'boundary.redesign', 'format', 'implement', 'master' og 'test'. Tryk derefter på 'Next'.
- **Trin 7:** Vælg et directory på din computer hvor du gemmer dit projekt, og vælg 'master' som din 'Initial branch'.
- **Trin 8:** Du mangler nu kun at vælge en 'Wizard' for at importere koden. Eclipse vælger selv at importere eksisterende Eclipse projekter. Lad dette felt forblive markeret, og tryk på 'Next'.
- **Trin 9:** Tryk på 'Finish'. Du har nu importeret projektet ind i dit Eclipse program, og kan vælge imellem de forskellige branches, samt de enkelte foldere i projektet.

- **Trin 10:** Åben projektet ved at dobbeltklikke på mappen i 'Package Explorer', og vælg 'src'.
- **Trin 11:** Tryk på mappen der hedder 'controller', og åben 'Main.java'. 'Main' er hovedfilen, som starter vores program.
- **Trin 12:** Efter at have åbnet Main.java, skal du trykke på den lille grønne pil, i den øverste bjælke i Eclipse, kaldet 'Run'. Dette starter programmet.
- **.bat fil kørsel:** Åben mappen '33\_CDIO3', og dobbelt-klik på mappen der hedder 'Installation'. Det er vigtigt at du henter både 33\_CDIO3.jar og CDIO3\_WIN.bat på din computer. Dobbelt-klik derefter på filen der hedder 'CDIO3\_WIN.bat' for at starte spillet.

Såfremt spillet ikke starter, kan du prøve følgende fejlløsninger:

- **For GitHub/Eclipse segmentet:**

- Opdater/Geninstaller Eclipse Neon 4.6.1 og Java 8.0. Begge programmer er *nødvendige* for at køre spillet via Eclipse.
- Fjern dit lokale repository og importer projektet fra [https://github.com/ArvidLangsoe/33\\_CDIO3](https://github.com/ArvidLangsoe/33_CDIO3) igen. Se ovenstående guide for hjælp med dette.
- Se efter opdateringer til dit systems software, i.e. Windowsopdateringer eller nye softwarepakker til dit grafikkort.

- **For .bat fil segmentet:**

- Opdater/geninstaller din Java-software, og genstart dit system. Du skal bruge Java 8.0 for at kunne køre spillet.
- Hent spil-mappen fra CampusNet eller GitHub repositoriet igen, og forsøg at køre spillet igen.
- Genstart din computer, og prøv igen.

## 7 Test

Igennem programmeringsfasen er spillets kode blevet testet for at sikre at alt virkede som det skulle. Koden er testet med flere testmetoder ved hjælp af det indbyggede testing framework, JUnit, i Eclipse. Spillet er også testet ved at gennemkøre programmet i en systemtest flere gange, for at se om det fungerede som det skulle.

Der er lavet JUnit tests for alle de vigtige metoder i Entity klasserne i programmet. Der testes blandt andet om de forskellige felter har de rigtige renteværdier mv.

I JUnit testene blev de forskellige funktionaliteter i programmet gennemprøvet. Funktionaliteterne i programmet er udviklet på baggrund af kundens opstillede krav. Til at få et overblik over projektkravene og om kravene er opfyldt i programmet, så opstilles der en traceabilitymatrice.

Figur (18) viser traceabilitymatricen for projektet, hvor de forskellige testcases står horisontalt (S01, S02, ..) og projektkravene står vertikalt (F1, F2, ..).

	F1	F2	F2.1	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F12.1	F12.2	F12.3	F12.4	F12.5	B1
S01	X	X	X																
S02				X	X													X	
S03						X	X	X											
S04												X			X				
S05							X					X				X			
S06							X					X	X			X		X	
S07												X						X	
S08									X	X	X	X	X						

Figur 18: Skema der sammenligner test med krav.

I traceabilitymatricen ses det, at alle projektkravene som vi mener kan testes er blevet testet.

Se nedenfor for en uddybende beskrivelse af tre testcases for tre System tests, der ligger flere mindre udførte test i senere afsnit:

Test case ID	S01
Summary	In this test we test the start of the game. We test if the player can enter the amount of players.
Requirements	F1, F2, F2.1
Preconditions	The game has been started.
Test procedure	Enter the amount of players that wants to play. Then enter that amount of player names.
Test data	None
Expected result	All 6 players appear on the screen.
Actual result	All 6 players appear on the screen.
Status	Successful
Tested by	Simon
Date	24-11-2016
Test environment	Windows Databar at DTU

Figur 19: Formel test af S01.

Test case ID	S04
Summary	In this test we test the labor camp field.
Requirements	F1,F12,F12.3
Preconditions	Player 1 owns two labor camp fields. Player 2 lands on one of them.
Test procedure	Test that when player 2 lands on the field he is informed where he landed. Then he should be prompted to roll the dice. Then he should pay the owner player 1. Now player 1 should land on his own fields and he should get a message he has done so. Nothing else should happen after that.
Test data	The players should have enough money left to pay eachother.
Expected result	Player 2 pays player 1, and player 1 is not told he has to pay himself.
Actual result	Player 2 test is good, but wrong message is send to player 1. However it has no influence on game or his money pool.
Status	Partially succesful.
Tested by	Simon
Date	24-11-2016
Test environment	Windows Databar at DTU

Figur 20: Formel test af S02.

Test case ID	S06
Summary	In this test we test that you can buy ownable fields.
Requirements	F1, F7, F12, F12.1, F12.3, F12.5
Preconditions	The game has been started, and the players has entered their names.. The player has sufficient cash to buy a field, and land on a ownable field.
Test procedure	The player get a message that they landed on a ownable field and are asked if they want to buy it. The player accepts and he pays money.
Test data	None
Expected result	The player gets a message that they have bought the field and a colored border appear around the field he bought.
Actual result	The player gets a message that they have bought the field and a colored border appear around the field he bought.
Status	Succesful
Tested by	Simon
Date	24-11-2016
Test environment	Windows Databar at DTU

Figur 21: Formel test af S03.

Før afleveringen af projektet, er programmet testet på en DTU databarenes computere.

Da systemtestene dækker over alle de krav der umiddelbart kan testes, har vi valgt ikke at

inkludere Junit test beskrivelser. Hvis man er interesseret i disse, kan de findes i testklasserne på github.

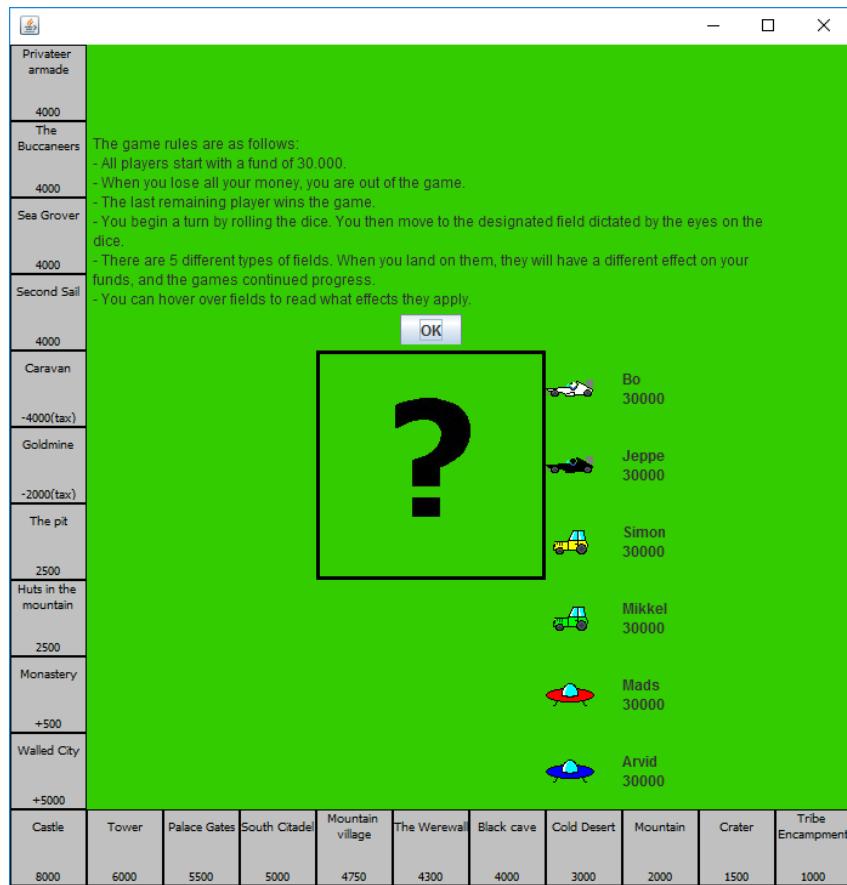
## 7.1 System test

Vi har vedlagt en et bilag med alle system test med billeder og beskrivelser. Her er nogle korte beskrivelser for testene.

- **S01:** Tester om man kan spille mellem 2-6 spillere og at de kan vælge navne.  
**Krav:** F1, F2.
- **S02:** Tester at to spiller ikke kan hedde det samme, og at der kommer beskeder der beskriver spillets start.  
**Krav:** F3, F4.
- **S03:** Tester at spilleren kan slå med terningerne og rykke rundt på pladen efter hvad han har slået.  
**Krav:** F5, F6, F7.
- **S04:** Tester at spilleren kan lande på en ejet Labor Camp og betale ejeren efter hvad han slår. Tester desuden at spilleren ikke kan betale sig selv.  
**Krav:** F12.3.
- **S05:** Tester Tax feltet.  
**Krav:** F7, F12.4.
- **S06:** Tester at man kan købe felter.  
**Krav:** F12.1.1, F12.3.1,F12.5.1, F7.
- **S07:** Tester at fleet felterne fungerer korrekt.  
**Krav:** F12.5.
- **S08:** Tester at spillere kan vinde og tabe spillet.  
**Krav:** F7, F8.

## 7.2 Acceptance test

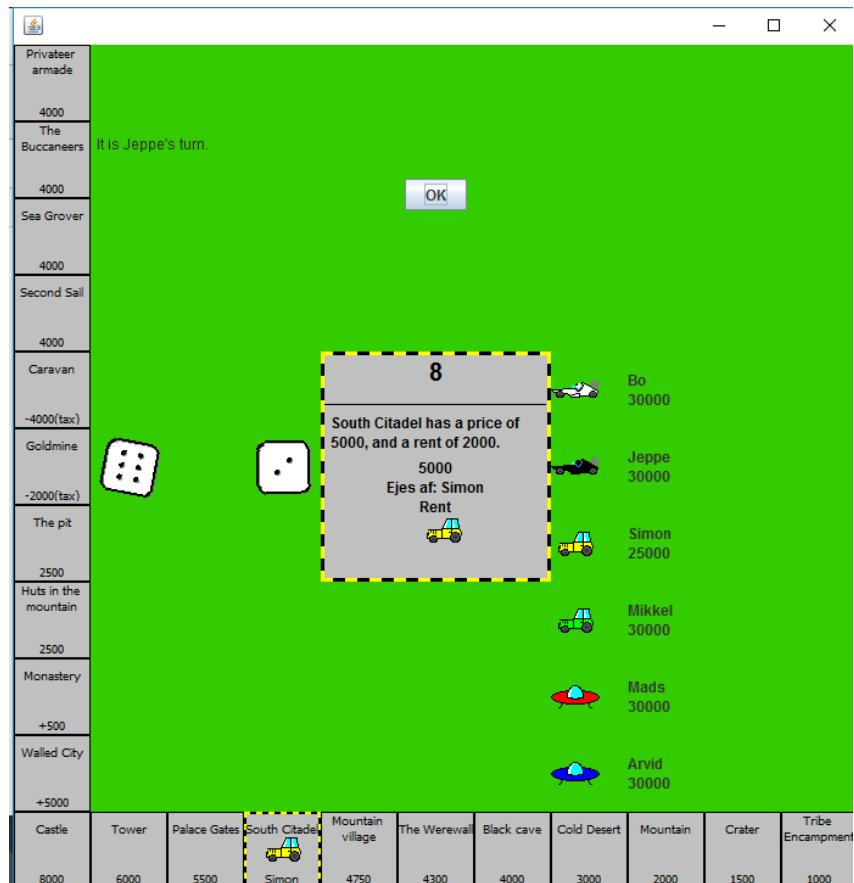
Når spillet startes, kan en af spillerne vælge hvor mange spillere der deltager i spillet. Der kan vælges mellem 2-6 spillere, hvor der herefter skal angives spillernes navne. Efter spillernes navne er indtastet, bliver spillets regler introduceret. Der klikkes på 'Ok' og spillet vælger herefter en tilfældig spiller der starter.



Figur 22: Spillet er startet, og der er indtastet antallet af spillere og deres navne.

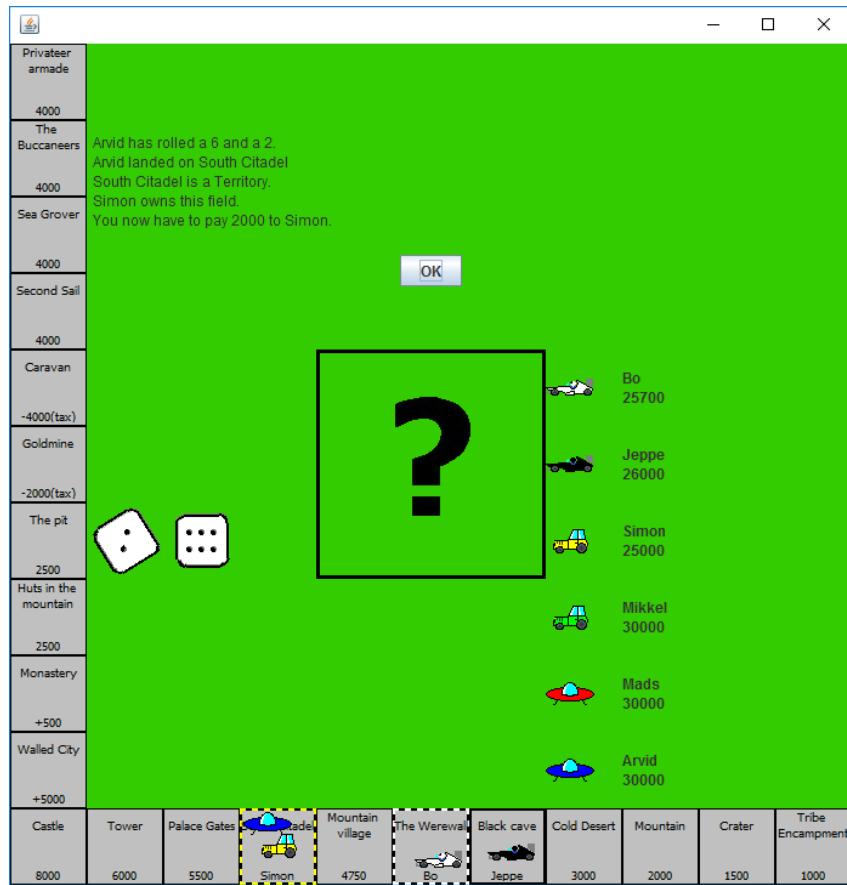
Det ses her at alle spillerne starter med en pengesum på 30.000.

Startspilleren slår med terningerne, og lander på et felt. I dette tilfælde lander han på et 'Territory' felt og har så muligheden for at købe feltet. Feltet købes, og ejerens farver kan nu ses på brættet rundt omkring det købte felt. Det ses også at spillerens næste tur vil starte fra det felt han landede på i forrige runde.



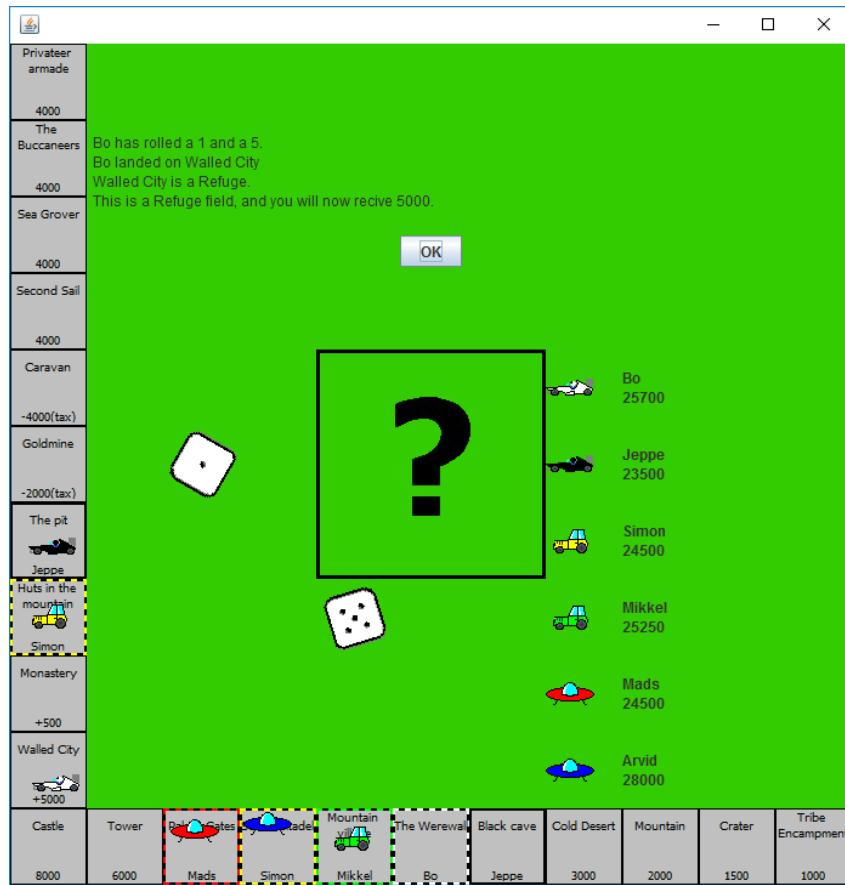
Figur 23: Simon har købt felt 'South Citadel'. Hans pengebeholdning er ændret og han er nu ejeren af feltet.

Spillet køres igennem et par gange, og en spiller lander nu på et 'Territory' felt der er ejet af en anden spiller.



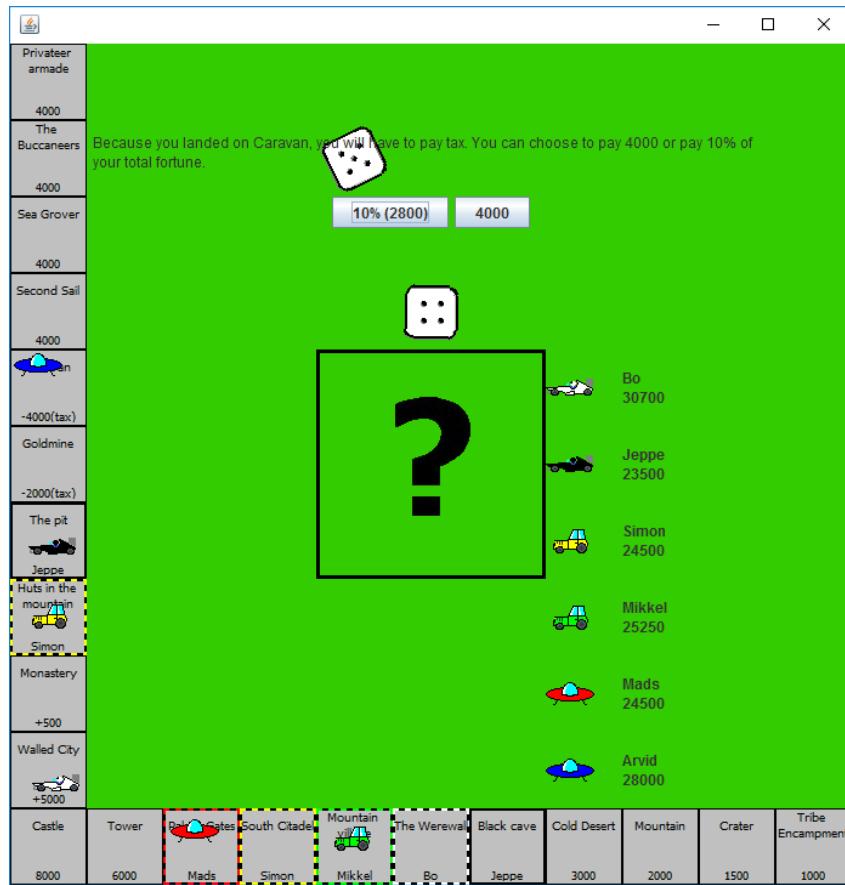
Figur 24: Arvid lander på 'South Citadel' felt som er ejet af Simon. Arvid betaler renten og deres pengebeholdning ændres efter der klikkes på 'Ok'.

Spillet kører videre, og en spiller lander på et 'Refuge' felt. Spilleren får ikke muligheden for at købe feltet, men får derimod en bonus.



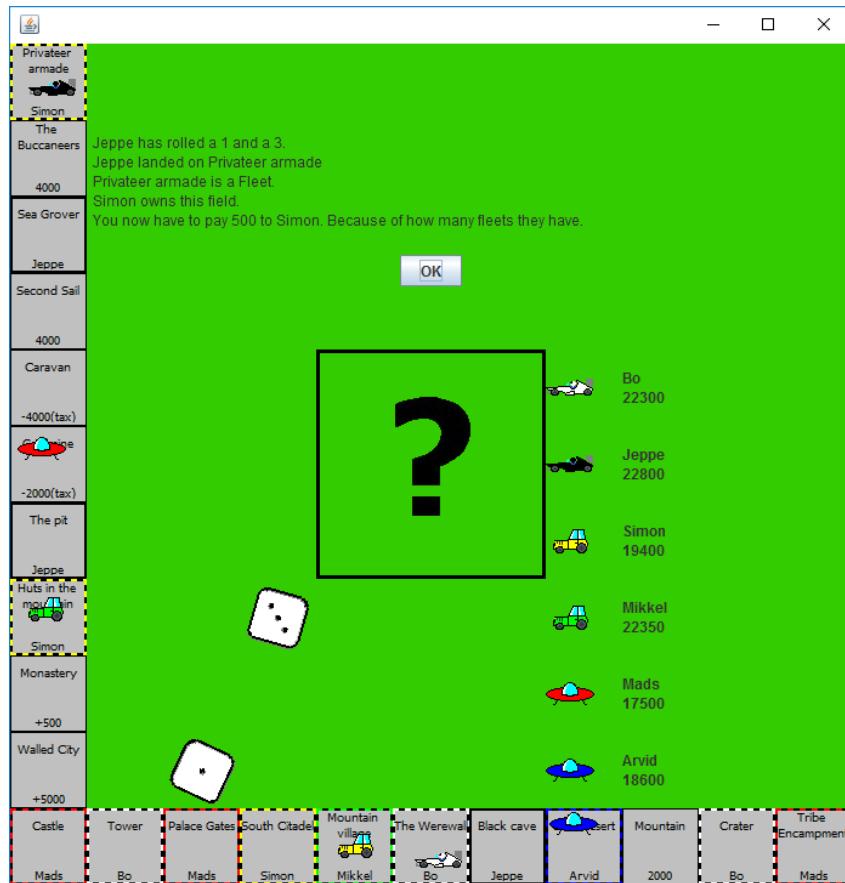
Figur 25: Bo lander på 'Walled City' og får en bonus på 5000 penge.

Der spilles videre og en spiller lander nu på et 'Tax' felt, hvor spilleren får muligheden for at betale 10% af sin formue eller et fast rente beløb.



Figur 26: Arvid lander på 'Caravan' og skal nu betale skat. Han får muligheden for at betale 10% af sin formue (2800) eller et fast skattebeløb på 4000.

Der spilles videre. Nu lander en spiller på et 'Fleet' felt der er ejet af en anden spiller. Fleet feltets rentebeløbet er afhængigt af hvor mange af Fleet der er ejet af den samme ejer.

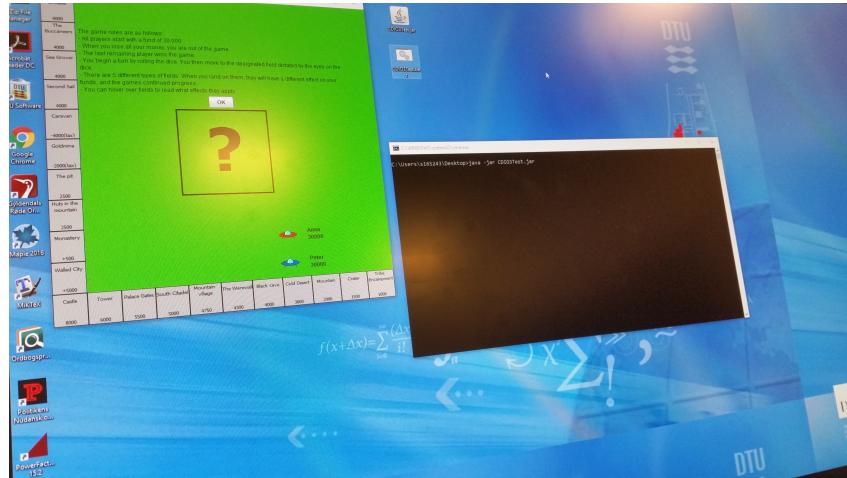


Figur 27: Jeppe lander på 'Privateer armade' som er ejet af Simon. Da Simon kun ejer ét Fleet felt, skal Jeppe kun betale den mindste rente værdi på 500.

Det ses tydeligt via spillets GUI hvilke af spillerne som ejer hvilke af brættets felter, samt hvad hvert felt koster. Fordi brættet hele tiden viser hvilken figur hver spiller har, elimineres risikoen for forvirring, da spillerne hele tiden kan bekraefte deres figur og felter.

Vi testede vores program på en af DTUs databars computere. Dette var for at sikre at vores krav om at spillet kunne køre på vores minimum-system krav.

Torsdag den 24. november, 2016, blev den endelige version af vores 'Rodatam'-spil testet, og kørte uden problemer. Vores medfølgende .bat fil blev også testet, og virker som den skal.



Figur 28: Dokumentation af test-kørslen. På billede ses to spillere, Anna og Peter spille. Spillets introduktion kan ses tydeligt, mens spillets .JAR og .bat fil kan ses på computerens skrivebord

## 8 Forbedringsforslag

- **Vis spillernes formue i GUI'en:**

Det er på nuværende tidspunkt ikke muligt for spilleren at se sin samlede formue i spillet. Dette kan virke irriterende for spilleren, og det kunne være smart at vise dem deres formue. Dette ville gøre det lettere for spillerne at følge med i deres samlede værdi.

- **Vis hvor mange Fleets og Labor Camps felter en spiller ejer, i ren tekst:**

Teksten er skrevet generelt for felterne i Fleet og labor camp. Det vil altså sige at spillerne ikke får at vide præcis hvor mange Fleets en spiller har når en spiller lander på et fleet-felt. Spilleren får blot en række mulige beløb at vide, som denne skal betale.

- **Bedre navngivning af klasser, primært Controller navne:**

Navngivningen af klasserne for Controllere og Boundary kan virke forvirrende og man bør ændre det så det er mere tydeligt hvad klassernes funktioner er. Eksempelvis findes en klasse kaldet GameCreator, men dens job er at oprette GUI'en, hvilket er misvisende. Dette bør ændres i fremtidige versioner af programmet.

- **Bedre BCE model:**

Selvom man har prøvet at følge principperne i BCE, kan Controller Boundary relationen virke lidt vag. Det kunne være godt at have en Controller og Boundary for hver use case, eventuelt slå et par stykker af dem sammen. Generelt set ville det gøre koden bedre og give højere Cohesion.

## 9 Konklusion

I dette projekt har vi skabt et mild udgave af matador hvor der er 2-6 spillere, der rykker brikker rundt på brættet og lander på diverse felter. Nogle af felterne kan ejes, og felterne kan have effekter på spillerne når de lander på dem. Det er blevet lavet sådan at når en spiller står tilbage alene har den spiller vundet, og spillerne taber når deres konto når under 0.

Vi har lavet en godt stykke kode via Eclipse. Vi har formået at skabe et spil med en flot GUI der er let forståelig for spillerne. Koden er bygget op efter BCE's principper og vi opretholder GRASP bedst muligt i koden. Derved har vi fået lavet et stykke kode som er nem at ændre, både for os selv, og andre interesseret.

GitHub har især vist sig værdigfuld i vores arbejde med projektet, og har gjort implementeringen og tests meget nemme at præsentere og hente ned.

Vi har sørget for at spillet er nemt at oversætte ved at have separate klasser der indeholder alle tekst meddelelser. Dermed kan man hurtigt ændre disse hvis man gerne vil oversætte spillet til et andet sprog.

Der er lavet mange forskellige test. Blandet andet er der lavet JUnit test for samtlige Entity klasser for at se om de gør det korrekte. Der er desuden lavet grundige tests i GUI'en for at se at spillet gør de rigtige ting på de rigtige tidspunkt.

Alt i alt er projektet gået godt og vi har lavet et fuldt funktionelt spil.

# 10 Bilag

## 10.1 Udvidet Systemtest

### 10.1.1 Test: S01

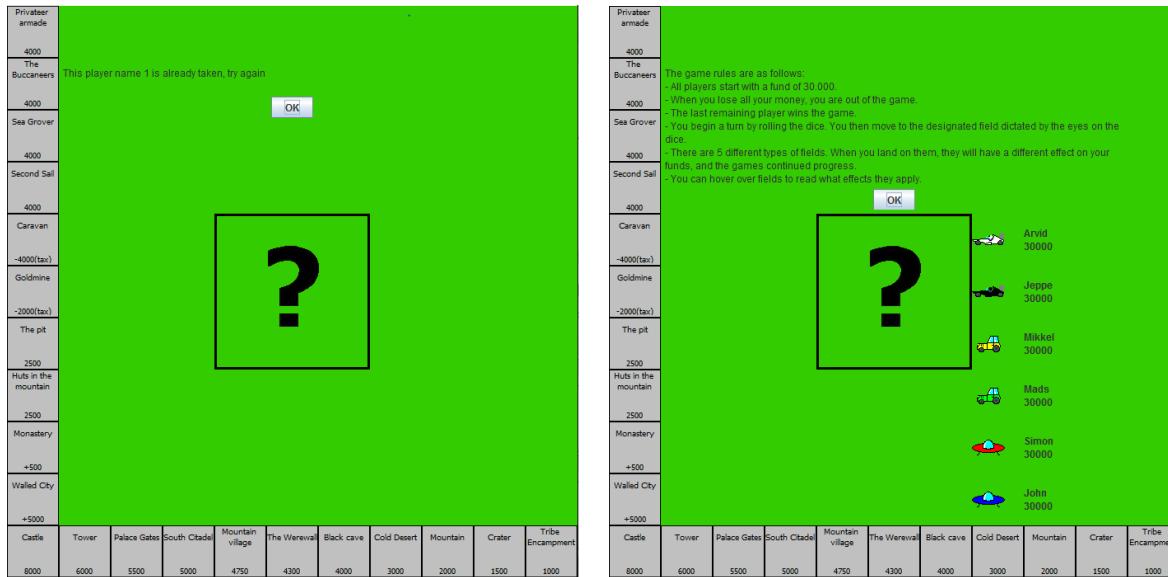
The screenshot shows a game setup screen. On the left, there is a vertical list of items with their prices: Privateer armade (4000), The Buccaneers (4000), Sea Grover (4000), Second Sail (4000), Caravan (-4000(tax)), Goldmine (-2000(tax)), The pit (2500), Huts in the mountain (2500), Monastery (+500), and Walled City (+5000). Below this list are numerical values for various resources: Castle (8000), Tower (6000), Palace Gates (5500), South Citade (5000), Mountain village (4750), The Werewal (4300), Black cave (4000), Cold Desert (3000), Mountain (2000), Crater (1500), and Tribe Encampment (1000). In the center, there is a large black square containing a white question mark. To the right of the square is a text input field containing the number "6" and an "OK" button. Above the input field, a message says "Enter the number of players who wants to play. The number must be between 2 and 6." On the far right, another part of the interface shows a list of items with their prices: Privateer armade (4000), The Buccaneers (4000), Sea Grover (4000), Second Sail (4000), Caravan (-4000(tax)), Goldmine (-2000(tax)), The pit (2500), Huts in the mountain (2500), Monastery (+500), and Walled City (+5000). Below this list are numerical values for various resources: Castle (8000), Tower (6000), Palace Gates (5500), South Citade (5000), Mountain village (4750), The Werewal (4300), Black cave (4000), Cold Desert (3000), Mountain (2000), Crater (1500), and Tribe Encampment (1000). In the center of this right section is a black square containing the number "2". To the left of the square, a message says "Please enter the name of player 1". Below the message is a text input field containing the name "John" and an "OK" button.

(a) Vælger antal spillere 2-6 (i dette tilfælde her seks).

(b) skriver spiller navnene

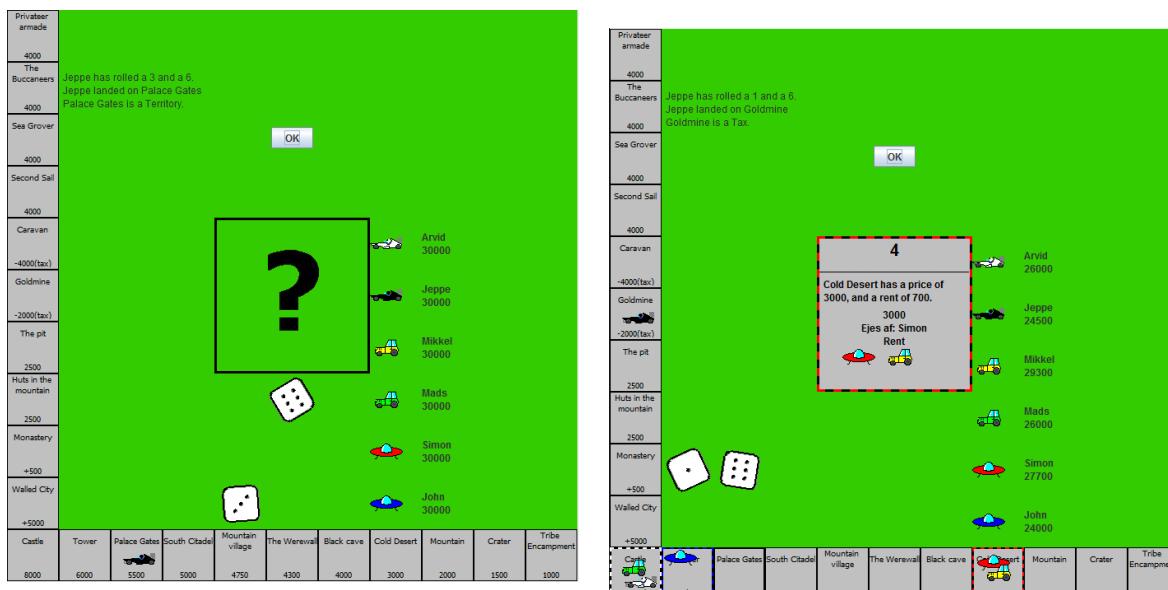
Figur 29: Før selve spillet starter vælges spillermængden (2-6) og disses navne. Hvis man vælger hvad som helst andet end 2, 3, 4, 5 eller 6 som antal spiller vil ”ok”knappen forblive grå. F1(+2)

### 10.1.2 Test: S02



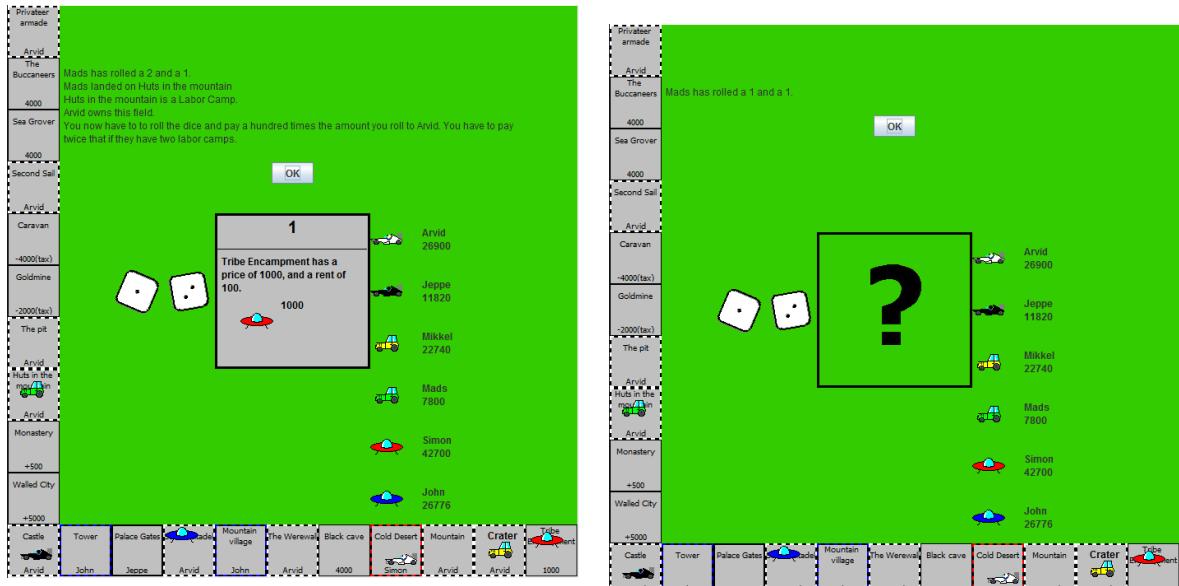
Figur 30: Prøver at taste to identiske navne, men det kan man ikke. Taster seks reelle navne og starter spillet med en pengesum på 30.000 til hver spiller. F3(+4)

### 10.1.3 Test: S03

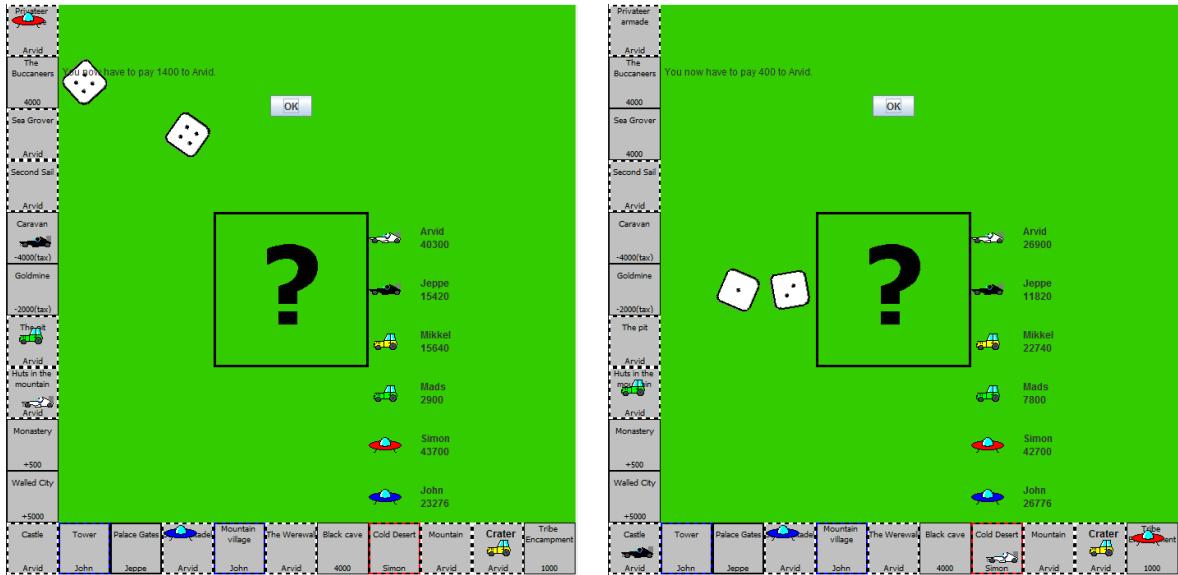


Figur 31: Spilleren 'Jeppe' spiller sit først slag og starter derfor fra et start felt som ikke ses (F5). I 'Jeppe's anden tur slår han 1+6 og derfor ender på felt 16=7+9, (F5,F6,F7).

#### 10.1.4 Test: S04

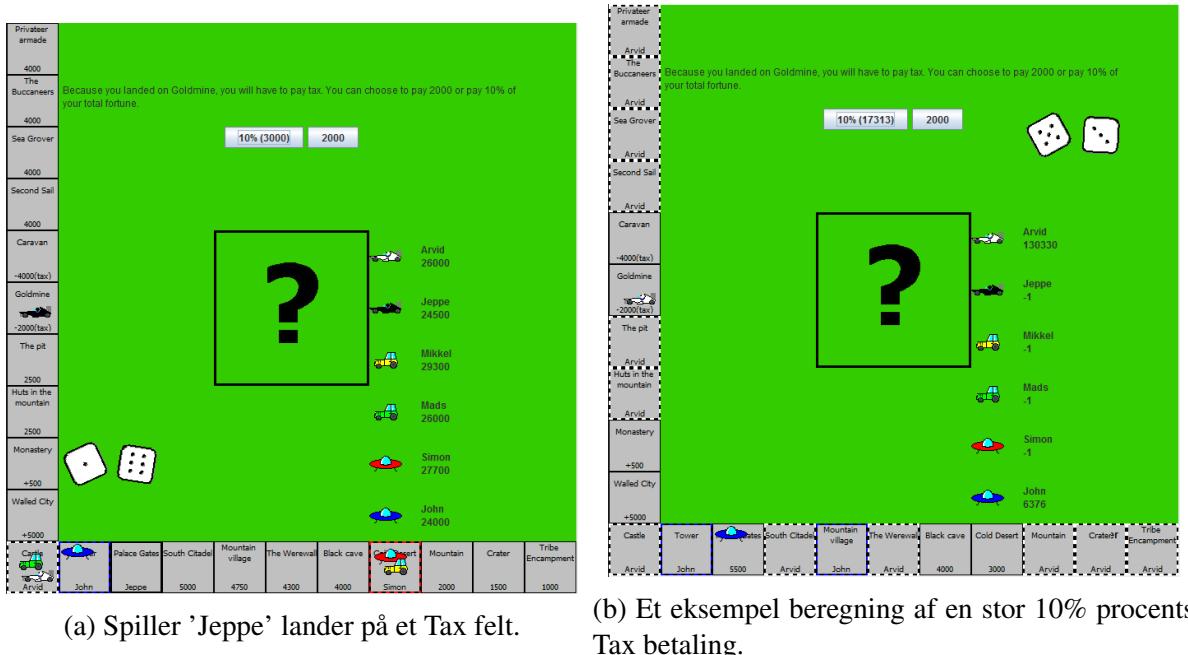


Figur 32: Spiller 'Mads' lander på en af 'Arvid's' to Labor Camp felter, og skal derfor betale  $100 * (\text{et terninge slag med } 2 \text{ terninger}) * 2$  (fordi 'Arvid' har to labor camps), (F12,3)



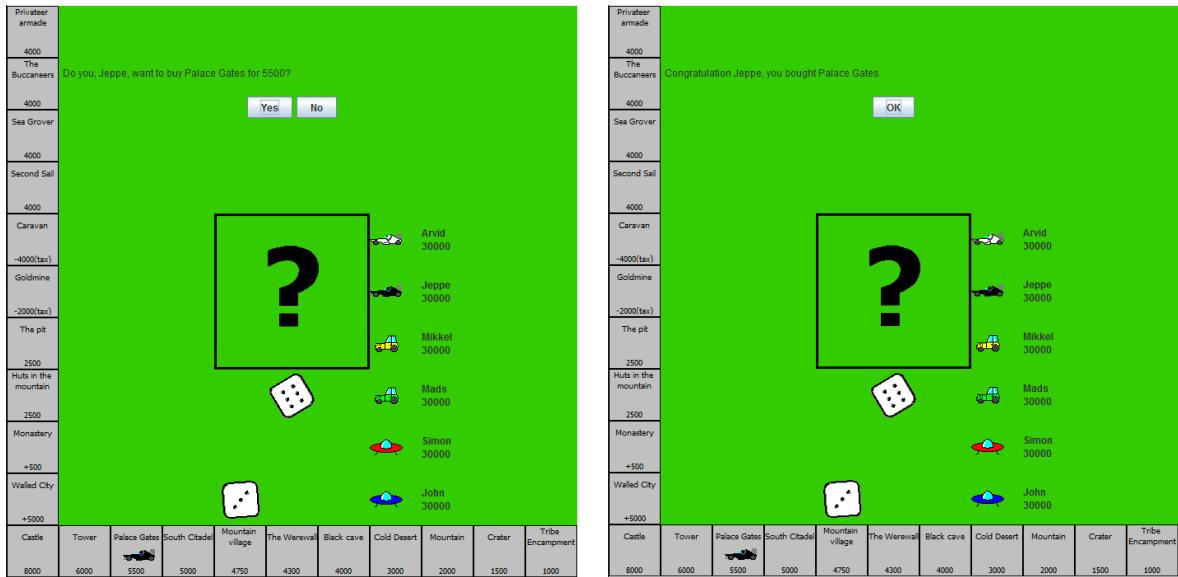
Figur 33: Afslutter ovenstående betaling mellem 'Mads' og 'Arvid' (F12,3). Under test-fasen fandt vi en fejl i spillet hvor 'Arvid' skal betale sig selv for at lande på sine egne Labor Camp felter. Dette problem løses af sig selv, da 'Arvid' betaler 'Arvid' lige så meget som 'Arvid' mister. Det har altså ingen betydning i dette tilfælde for 'Arvid's pengebeholdning. Der sker altså intet andet end at der bliver brugt lidt tid på den visuelle repræsentation. Det er en visuel fejl som evt. kan rettes i en senere version.

### 10.1.5 Test: S05



Figur 34: Spiller 'Jeppe' vælger imellem hvilken Tax betalingsmetode han skal vælge (F12.4, F7).

### 10.1.6 Test: S06

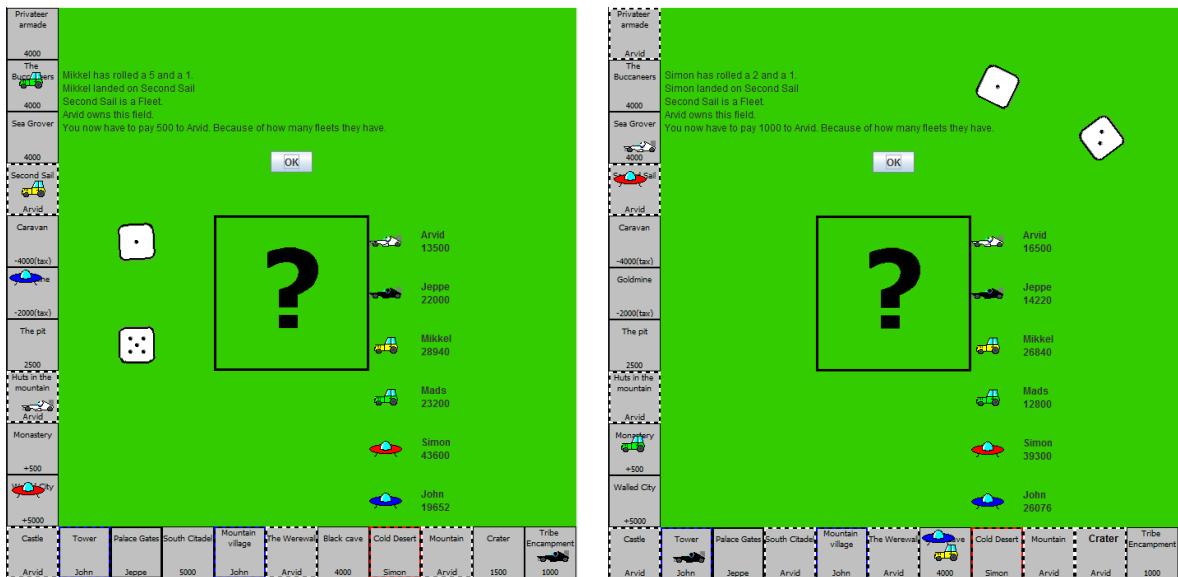


(a) 'Jeppe' vælger at købe et felt.

(b) 'Jeppe' har købt et felt.

Figur 35: Spiller 'Jeppe' vælger at købe et felt, og gennemfører overførslen (F12.1.1, F12.3.1, F12.5.1, F7).

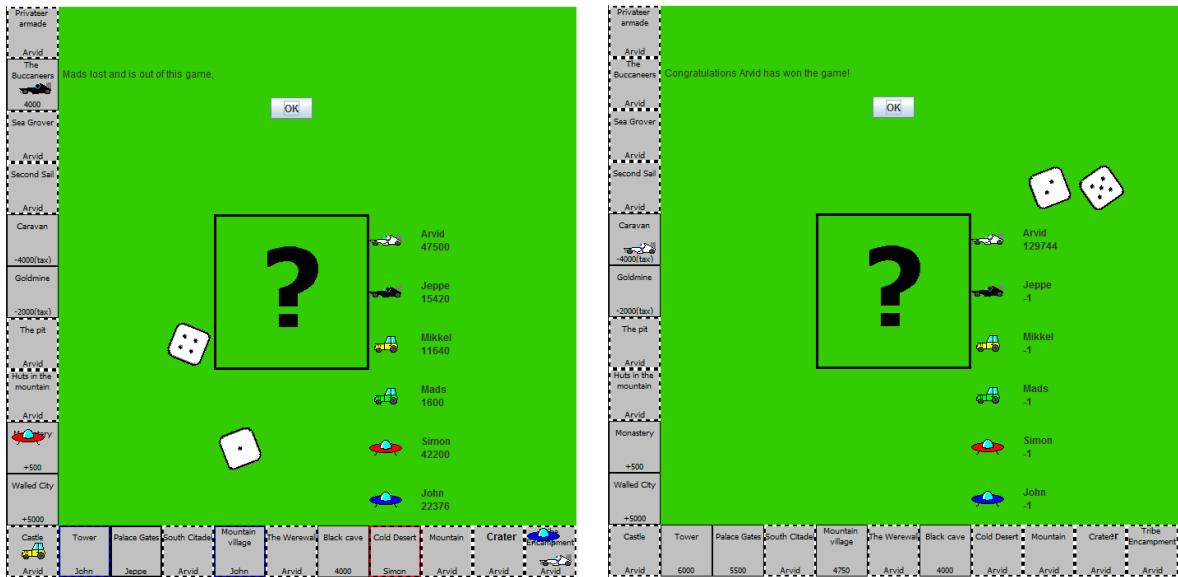
### 10.1.7 Test: S07



(a) 'Mikkel' lander på et Fleet felt, hvor ejeren har 1 Fleet.  
(b) 'Simon' lander på et Fleet felt, hvor ejeren har 2 Fleets

Figur 36: 'Mikkel' lander på et Fleet felt, hvor ejeren ('Arvid') har et Fleet felt. 'Simon' lander på et Fleet felt, hvor ejeren (Arvid) har 2 Fleets (F12.5.2).

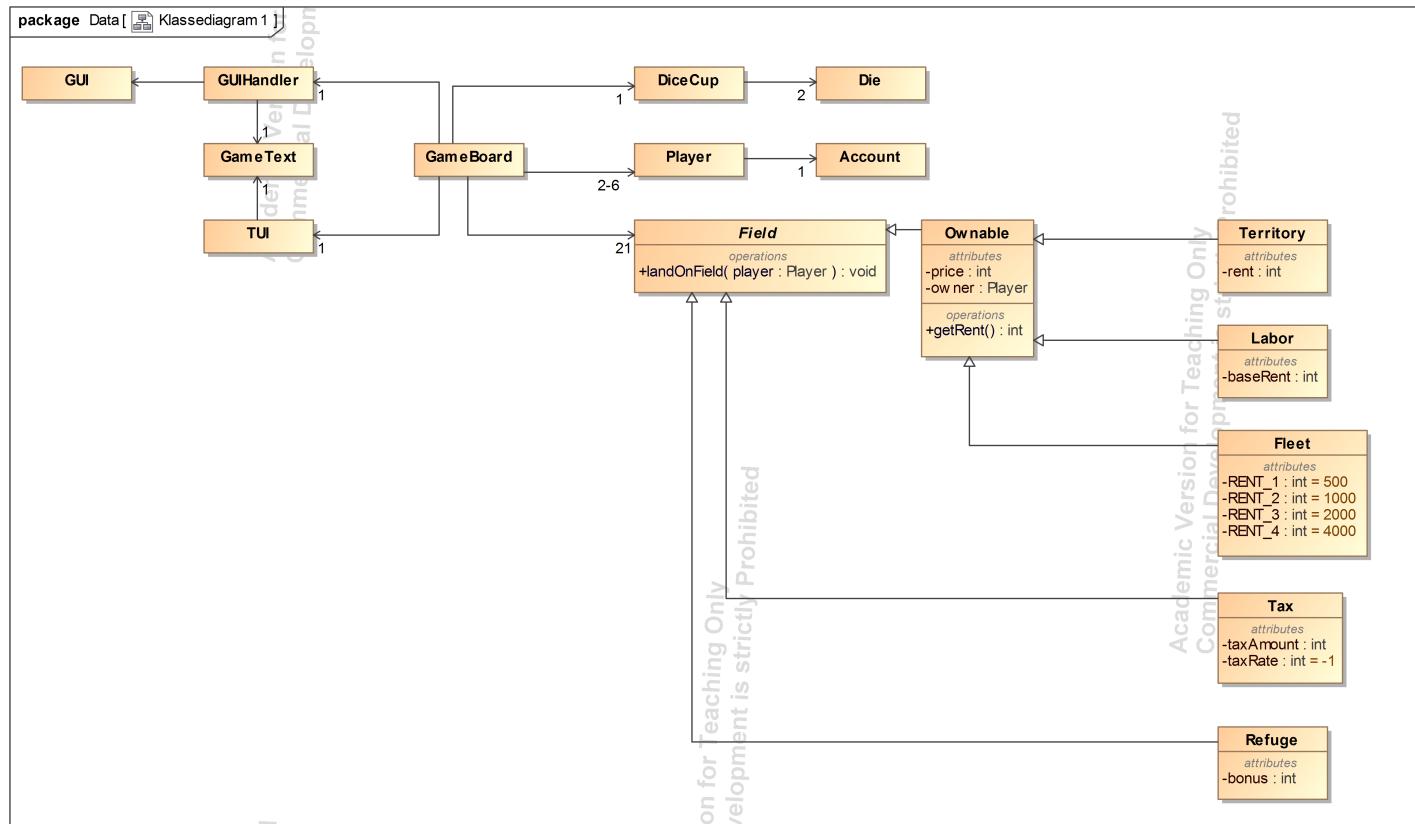
### 10.1.8 S08



(a) 'Mads' går bankerot og derved er ude af spillet. (b) 'Arvid' er sidste spiller tilbage, han vinder.

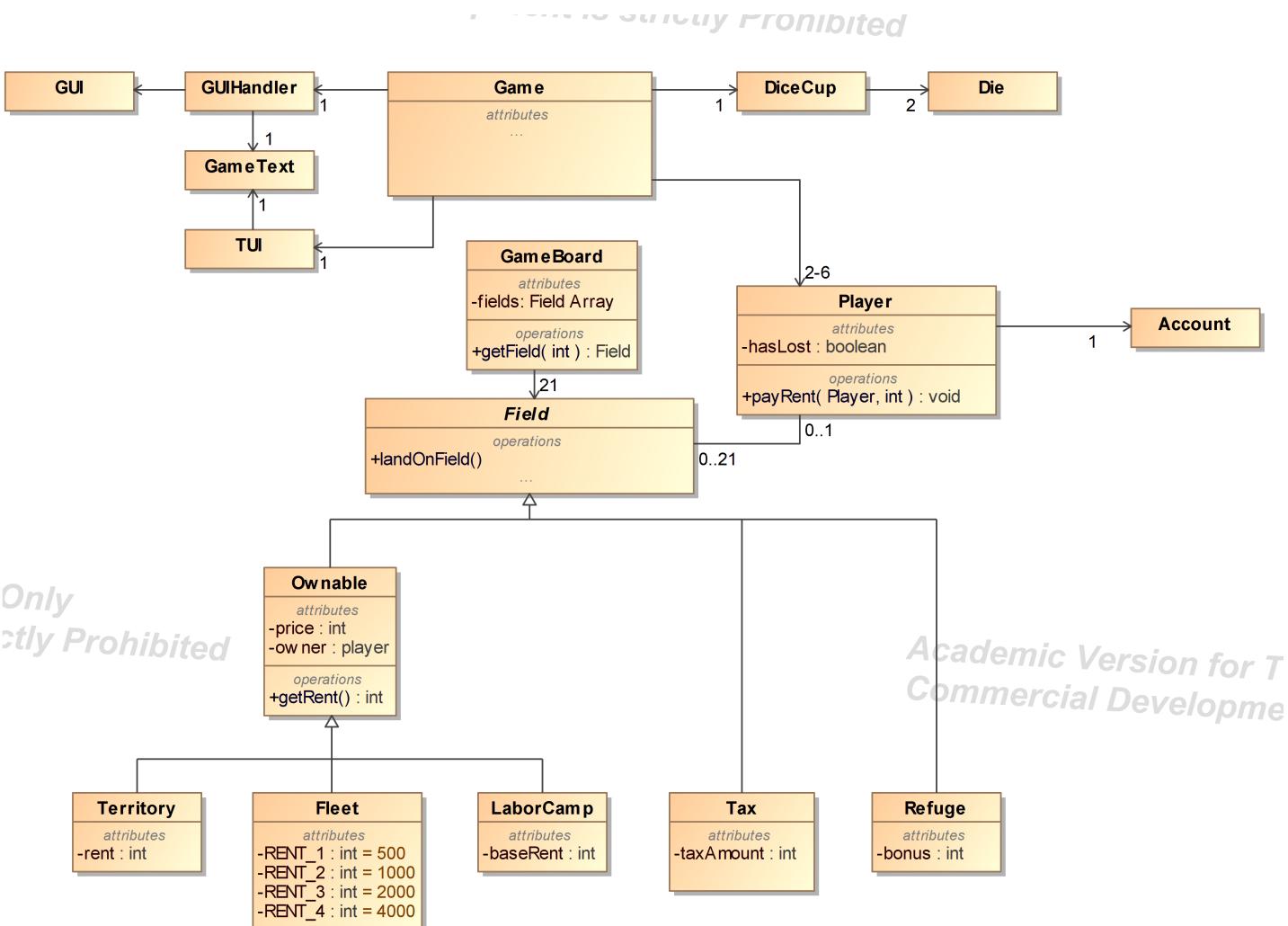
Figur 37: 'Mads' går bankerot og derved er ude af spillet. 'Arvid' er den sidste spiller som ikke er gået bankerot, det betyder at han vinder. (F7, F8)

## 10.2 KlasseDiagram1



Figur 38: Det første Diagram, der mangler nogle pile.

### **10.3 KlasseDiagram2**



Figur 39: Det første Diagram, der mangler nogle pile.