

Resumé

This paper describes our way from idea to finished matador project. It will show how we through analyzing the problem of creating a digital matador game comes up with a solution that works. The matador game is an old board game from 1903. For first time in 1936 the first paper edition was published. In the following the architecture of the system is described in various diagrams. All codings is done in the compiler program eclipse and the finished game can be found together with this report.

Indhold

1	Indledning (134228)	1
2	Baggrund og motivation (134226)	2
3	Problemformulering (134218)	2
4	Proces (123744 ,134228)	3
5	Krav og Analyse	4
5.1	Kravspecifikation (134218)	4
5.2	Use-case (134218)	5
5.3	Tilstands diagram (134226)	8
5.4	Domænemodel (134226, 134223)	8
5.5	Navneordsanalyse (130714)	8
5.6	Risikoanalyse (130714)	9
5.7	System sekvens diagram (134226)	9
5.8	Database (130714, 134226, 134228)	11
6	Implementation	15
6.1	Software dokumentation (134223)	15
6.2	Sekvens diagram (134218)	15
6.3	Design klassediagram (134218)	15
7	Test (134223, 134228)	16
7.1	Strategi for test	16
7.2	Testen	16
7.3	Test konklusion	19
8	Brugervejledning (134223)	20
8.1	Opstart spillet	20

8.2	Oversigt over spillet	20
9	Konklusion (130714)	22
A	Appendices	23

Figurer

1.1	Matador spil	1
4.2	Tidsplan	3
5.3	Use-case 'Oprette spil'	5
5.4	Use-case 'Køb grund'	5
5.5	System sekvens diagram: Opret spil	10
5.6	System sekvens diagram: Spillet	10
5.7	Normalisering af databasen	12
5.8	ER-diagram	13
A.9	Tilstands diagram	24
A.10	Domæne model	25
A.11	Sekvens diagram: Game	26
A.12	Sekvens diagram: PlayerMove	27
A.13	Sekvens diagram: InstanceOfShipping	28
A.14	Design klasse diagram	29
A.15	Test: Delete Parent row	30
A.16	Test: Update Parent row	31
A.17	Test: Update Child row	32

Tabeller

1	Use-case beskrivelse af 'Oprette et spil'	6
2	Use-case beskrivelse af 'Køb en grund'	7
3	Relation skema	14

1 Indledning (134228)

Matador har igennem flere årtier været en fast bestanddel i de fleste hjem. Et brætspil med rødder helt tilbage til 1903, hvor 'The Landlord's Game' så lyset for første gang. I 1936 designede C. Drechsler Papirvarefabrik den danske udgave af dette brætspil, Matador, der ved C. Drechsler død i begyndelsen af 1970'erne blev solgt til Brio, som de fleste af os i dag nok forbinder med Matador. Vi vil i denne opgave designe et program der kan simulere et Matador spil, og give spillerne samme fornemmelse som et alm. spil Matador. Her skal programmet indeholde det velkendte spillebræt, alle de velkendte funktioner og spilleregler matador består af. Udover de fornævnte ting, skal brugerne have mulighed for at gemme og indlæse spillet igen, da et typisk spil Matador ofte tager lang og spille færdig.



Figur 1.1: Matador spil

2 Baggrund og motivation (134226)

Under kurserne 02327 og 02362 skal vi udarbejde et matadorspil. Det skal så vidt som muligt fungere på samme måde som det originale spil, Kravspecifikationerne er derfor reglerne til matador. Vi har valgt at gøre projektet an, som var det en bestillings opgave fra en kunde.

Vi vil under denne opgave bygge videre på den viden vil tilegnede os i kurserne 02313 (Udviklingsmetoder til IT-systemer) og 02314 (Indledende programmering). Dvs. dette projekt er bygget op fra bunden, men bygger på de principper vi lærte i de fornævnte kurser. Forskellen er nu at vi skal inddrage database kurset, hvilket også afspejler sig i programmet som er udviklet i forbindelse med dette kursus.

Alle tabeller vil blive lavet i MySQL og derefter skal vores kode kunne samarbejde med databasen for at få spillet oppe og køre, til dette benyttes JDBC.

Det forventes, at læseren af denne rapport har grundlæggende kendskab til programmering i Java, samt grundlæggende kendskab til database implementering, dette sikre at læseren vil få det optimale ud af denne rapport.

3 Problemformulering (134218)

Vi ønsker at designe et program der kan simulere et Matador spil. Vores krav til spillet er at det skal bygges op ved hjælp af 3-lags modellen og det skal være muligt at gemme og hente spillet, for det kan være muligt at gemme spillet, skal vi have en database.

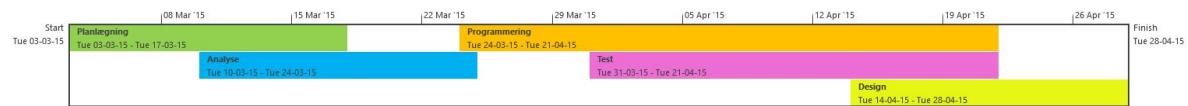
Derfor har vi følgende problemstillinger vi står med:

- Analysere vores problem vha. UML
- Bygge koden op med 3-lags model
- Tilkoble en database, der gør det muligt at gemme/loade spillet

4 Proces (123744 ,134228)

Vi har til dette projekt valgt at arbejde efter OOAD metoden Unified Process (UP). Denne metode består typisk af fire faser:

- **Inception:** Første fase i metoden består af forberedelse af programmet. Her er det vigtigt at identificerer nøglefunktionerne i det færdige program, og lave use-case beskrivelse til disse, samt en risikoanalyse.
- **Elaboration:** Der skal designes use-cases, gennemse risikoanalyse og beslutte det videre forløb med udgangspunkt i use-casene, og udarbejde projektplan over forløbet.
- **Construction:** I denne del konstrueres programmet, dvs. funktionerne udvikles og testes, samt dokumentation for koden laves. Her vurderes det samtidigt om programmet lever op til brugerens krav, og om programmet er tilfredsstillende.
- **Transition:** Her leveres det færdige program til brugeren, og eventuelle problemer med det leveret system løses. Nedenfor ses



Figur 4.2: Tidsplan

Metoden er iterativ, og projektet bliver dermed delt op i passende iterationer. Varigheden af disse iterationer bestemmes efter kompleksiteten af programmet, men det vigtige ved at benytte disse iterationer, er at man ved hver endt iteration har et "færdigt" program, og ved hver ny iteration tilføjer til dette program.

Metoden er use-case drevet, dvs. de udarbejdede use-cases udgør kernen i udviklingen af programmet. Når en iteration påbegyndes, vælges der så en eller flere use-cases der tages udgangspunkt i. Hvilket kan være en ulempe, da der kræves så meget dokumentation i form af sekvensdiagrammer og det kan være tidskrævende og resultere i, at ændringer løbende koster meget tid. Ide Up er låst fast i nogle i sine faser, er det ikke nemt at lave store ændringer, når man først er færdig med en fase. Fordelene er selvfølgelig de reviews der er undervejs.

Metoden er risiko drevet, og selve risikoanalysen foretages tidligt i projektforløbet, og ud fra denne analyse vælges hvilke use-cases man vil arbejde med, og hvornår i forløbet de forskellige use-cases skal udarbejdes for at eliminere de største risici tidligt i forløbet.

5 Krav og Analyse

Arkitekturen af spillet vil blive fastslået i dette afsnit, gennem analyse af kravene for systemet. For at demonstrere og dokumentere dette, bliver der udformet diverse diagrammer ud fra kravene.

5.1 Kravspecifikation (134218)

Vi har valgt at lave en kravspecifikation over de funktionelle og non-funktionelle krav. Det gør vi for, at sikre os at vi har forstået opgaven og samtidig diskutere de “løse” ender. Til at lave kravspecifikationen benytter vi os af FURPS, der er en model. Grunden til vi benytter denne model, er at vi på den måde sikre os, at gennemgå de forskellige typer krav, et softwareprogram har. FURPS står for følgende fem ting:

5.1.1 Funktionalitet

- Spillet skal fungere som et Matadorspil
- Man skal være mellem 2-6 spillere
- Det skal være muligt at gemme spillet og indlæse det senere
- Der skal benyttes en GUI til spillet
- Spiller skal kunne pantsætte egne grund
- Spiller skal kunne købe grunde, der ikke er købt

5.1.2 Usability

- Det skal være forståeligt, det vil sige, at det skal være muligt for alle der ved hvordan man benytter en computer, at spille Matador spillet, så vidt man kender Matador reglerne

5.1.3 Reliability

- Det forventes altid at være muligt at spille matador, så længe man har en computer at spille det på, også selvom man ikke er på internettet. Dog kræver gem/hent spil at der er forbindelse
- Hvis spillet crasher, vil det være muligt at hente spillet igen

5.1.4 Performance

- Spillet skal kunne køre uden yderligere forsinkelser, der vil generere spilleren

5.1.5 Supportability

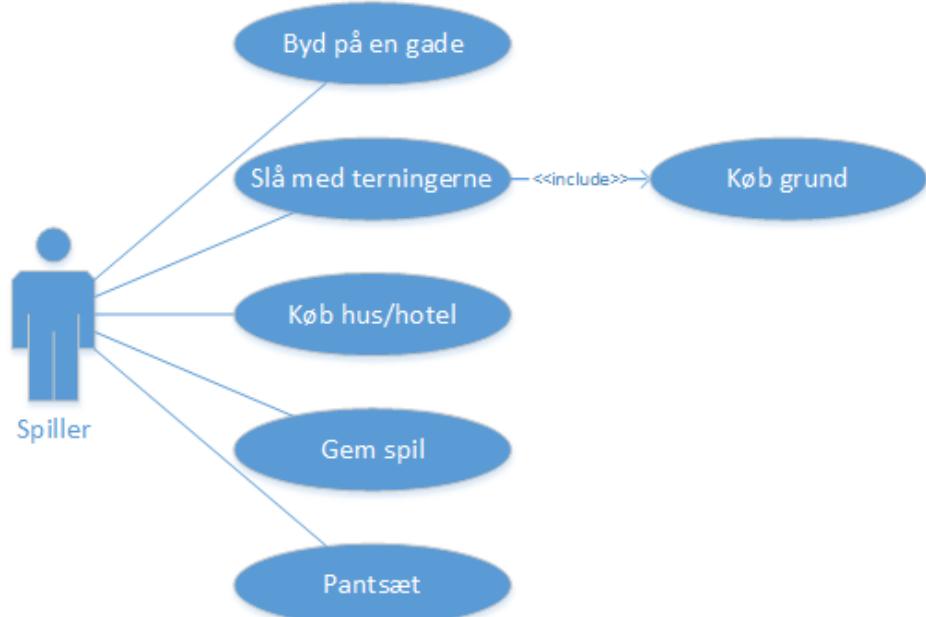
- Spillet skal kunne køre på windows, os og linux, så længe man har installeret Java

5.2 Use-case (134218)

Vi har valgt at lave use-cases over, hvilke muligheder man har i spillet, både inden man er inde i spil og inden man slår et slag. Det har vi lavet for at give et overskueligt billede over, hvilke muligheder spilleren kan foretage sig i matador spillet.



Figur 5.3: Use-case 'Oprette spil'



Figur 5.4: Use-case 'Køb grund'

5.2.1 Fully use-case beskrivelser

Use-case	Oprette et spil
Scope	Matador
Level	User-goal
Primary actor	Spiller
Stakeholders and interests	Denne use-case er henvendt til spillere, der har lyst til at spille Matador.
Preconditions	Denne use-case skal bruges til at starte et spil Matador.
Main succes scenario	<ol style="list-style-type: none"> 1. Spilleren starter programmet 2. Spilleren vælger 'Nyt spil' 3. Spilleren vælger 'antal spillere' 4. Programmet godkender antal spillere 5. Spilleren navngiver spillerne 6. Programmet er nu klar til at starte
Extensions	Hvis der er valgt et forkert antal spillere, skal dette gøres om, dvs. der vælges andet end 2-6 spillere. Spillet tillader ikke indtastninger udover det tilladte.

Tabel 1: Use-case beskrivelse af 'Oprette et spil'

Use-case		Købe en grund
Scope		Matador
Level		User-goal
Primary actor		Spiller
Stakeholders and interests		Denne use-case er henvendt til spillere, der gerne vil købe en grund.
Preconditions		Use-casen skal bruges til at købe grunde i Matador spiller.
Main succes scenario		<ol style="list-style-type: none"> 1. Spilleren slår med terningerne 2. Spilleren rykker antal øjne frem 3. Programmet kontrollerer hvilken type felt spilleren er landet på 4. Programmet kontrollerer om grunden er ejet 5. Spilleren vælger at købe grunden 6. Programmet kontrollerer om spilleren har nok penge til grunden 7. Spilleren ejer grunden
Extensions		Hvis spilleren ikke har nok penge til grunden, får spilleren besked om dette. Hvis grunden er ejet kan det ikke lade sig gøre at købe grunden, og spilleren skal bestale leje.

Tabel 2: Use-case beskrivelse af 'Køb en grund'

5.3 Tilstands diagram (134226)

Tilstandsdiagrammer benyttes til at vise et model elements livscyklus. Det kan være mange ting. Til Matadorspillet, har gruppen udarbejdet tilstandsdiagrammet ud fra forskellige use-cases og klasser som modelementer. Diagrammet A.9 indeholder mange tilstande, disse tilstande opnås ved de forskellige eksterne hændelser, eksempelvis ved hjælp af relationer mellem de forskellige tilstande, så ændre det pågældende model elements tilstand. Disse relationer, der igangsættes af hændelserne, kaldes transitioner.

Som nævnt er der udarbejdet et tilstandsdiagram, og diagrammet A.9 viser hvordan spillet startes op og hvilke muligheder man har i de forskellige hændelser. For gennem matador spillet kan man komme i forskellige tilstande, og det er disse hændelser som gør at man kommer i de forskellige tilstande.

Et eksempel kunne være hvis der ses på diagrammet A.9, og i tilfælde af at det er en spillers tur, hændelsen er nu at man skal slå med terningerne og det medføre at man lander på et felt og dermed ændres tilstanden, herfra kan man lande på feltet for lykke kort og der er flere muligheder for forskellige hændelser, og dermed at ende i forskellige tilstande, både positive og negative, eksempelvis hvis man trækker et kort hvor man skal rykke frem til start og modtage 4000 og en anden mulighed kunne være at trække et kort hvorpå man skal i fængsel.

(Se appendix A.9 for Tilstands diagram)

5.4 Domænemodel (134226, 134223)

Domænemodellen i appendix A.10 viser et billede af den virkelige verden. Vi har forsøgt at gøre systemet simpelt og vise, hvilke elementer der hører til hinanden. Vores system er opdelt efter forskellige rollefordelinger, og hver felt har sin rolle. Modellen A.10 viser den overordnede struktur af Matador spillet, man kan se hvilke elementer der bliver benyttet og hvad deres relation til hinanden er. Som det ses har vi Board som er tilknyttet Field og Controller. Field er alle de typer af felter som man kan lande på (40 felter). Controller er knyttet til User og Dice, hvilket er vores terninger hvorfra vi får den samlede sum af de to terninger som giver os svar på hvilket felt vi lander på. Vi kan se at hver User har 1 konto, 1 bil og mulighed for at kaste med 2 terninger.

(Se appendix A.10 for Domænemodel)

5.5 Navneordsanalyse (130714)

For at skabe overblik og give inspiration til hvilke klassenavne og attributnavne vi kan have med i vores program, benyttes der en navneordsanalyse. I denne kigges der blandt andet på problemformuleringen og kravspecifikationen, for at finde relevante navneord.

- Spiller
- Terning

- Balance
- TUI
- Felt
- Penge beholdning
- Spilleplade

5.6 Risikoanalyse (130714)

I dette projekt har vi valgt at lave en analyse som har til hensigt at identificere hvilke risici der er forbundet til projektet, og deres sandsynlighed og eventuelle konsekvenser. I denne risikoanalyse er der valgt bottom up metoden, hvilket vil sige at projektets forskellige dele gennemgås en efter en, og der undersøges hvad der kan gå galt / hvilke udfordringer der er i dette.

Programmet:

Selve matadorspillet bliver kodet i java/eclipse som synkroniseres ved hjælp af github. Dette kan der være en risiko ved da der kan ske synkroniseringsfejl hvis flere gruppemedlemmer redigere i samme klasse på samme tid.

Der kan også opstå fejl i mangel på kommunikation hvis et gruppemedlem ændre noget i koden uden at fortælle det til gruppen hvorefter en anden ændre det til noget tredje. På den måde kan der opstå forvirring om hvad der står i koden og fejl/grim kode kan derved opstå.

Databasen:

Selve databaseserveren køre på en computer/server stående hjemme hos joakim. Risikoen ved dette er, at hvis denne går ned, mister strømmen eller på anden måde stopper med at virke, kan programmet ikke gemme eller loade spillet, og væsentlig funktionalitet går derved tabt.

En anden risici ved dette er også at hvis noget går galt med serveren kan vi ikke fixe det ved mindre vi er fysisk tilstede. Dette kan langsomtliggøre udviklingsarbejdet.

5.7 System sekvens diagram (134226)

Her ses et SSD for en spiller/Bruger. System sekvens diagrammet er interaktionen mellem bruger og systemet, det giver et overblik over spillets forløb, derfor vises kun det, en spiller har mulighed for. SSD diagrammet skal kunne passe sammen med Use-casene, derfor ses 2 SSD diagrammer. Det første over hvordan man starter spillet og det næste over spillets forløb.

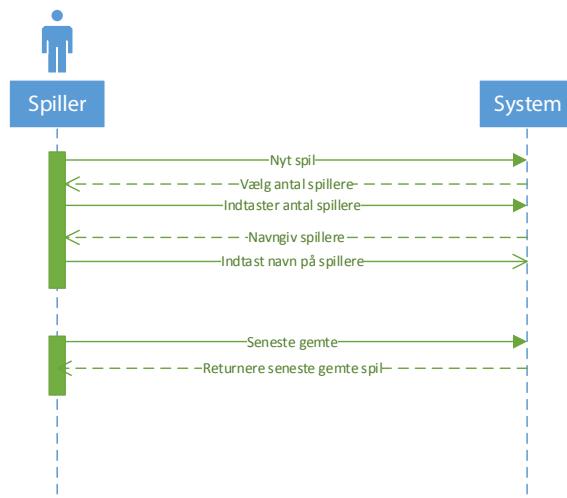
På det første system sekvens diagram kan vi se at en spiller skal starte spillet, hvis der vælges et nyt spil, beder systemet ham om at indtaste antal spillere , da spillereglerne er 2-6 spillere. Indtaster man et tal mellem 2-6, beder systemet om spillernes navne og derefter beder den om at kaste med terningerne og dermed starte spillet.

Andet system sekvens diagram er en fortsættelse af det første hvor spillet er startet. Her er vist alt det en spiller har mulighed for under spillet. Hvor man kaster med terninger,

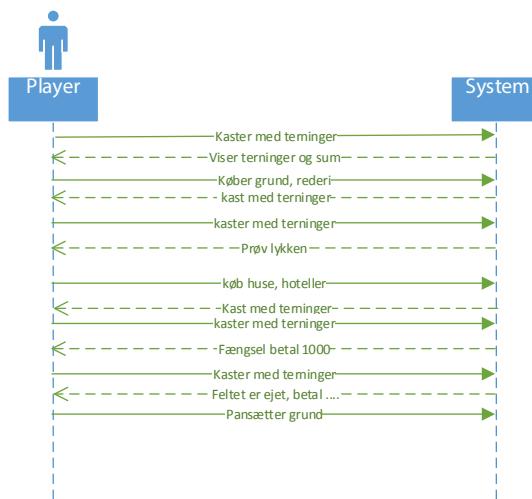
derefter viser systemet antal øjne på terningerne og i tilfælde af at man lander på et ikke ejet felt giver systemet spilleren mulighed for enten at købe grund'en eller sende turen videre til næste spiller.

Som vi kan se er der grunde man ikke kan købe såsom 'Prøv lykken'. Her trækker man et kort, hvor systemet så derefter fortæller hvad man skal. Lander man på fængsel feltet skal man slå to ens med terningerne, lykkes dette ikke inden for tre runder køber man sig ud.

I tilfælde af at man køber alle grunde i samme farve så kan vi se at systemet giver mulighed for at bygge huse og hoteller. Hvis man lander på et ejet felt beder systemet spilleren om at betale leje til spilleren der ejer feltet.



Figur 5.5: System sekvens diagram: Opret spil



Figur 5.6: System sekvens diagram: Spillet

5.8 Database (130714, 134226, 134228)

Vores database er opsat på en server der kan tilgås online ved hjælp af brguernavn og kode der står i databaseklasse. I databasen har vi oprettet en user med userNumber 0. Denne user fungere som ”banken” og er lavet for at alle felter har en ”ejer” og dermed bliver gemt ordentligt i databasen.

5.8.1 Normalisering (130714, 134226)

Når der designes en logisk database, kan man bruge normalisering til at designe en effektivt og robust database. Normalisering i 3.nf sørger for at man ikke har redundans og opdateringsanomalier, som man ellers ville få, hvis man ikke fulgte normaliseringsreglerne. Ved benyttelsen af normaliserings reglerne har vi opnået sikre opdatering anomalier, samt undgået redundans.

- **Første normalform:** Opnås ved at fjerne gentagne grupper, relateret data sættes i en tabel for sig selv, og dermed identificere de relateret datas primary keys
- **Anden normalform:** Opnås ved at oprette tabeller som kan bruges ved flere poster, hvor de relaterede data er foreign keys
- **Tredje normalform:** Opnås ved fjernelse af alle de felter der ikke afhænger af nøgler

Måden vi har sikret at vores database model er på tredje normalform, er ved at køre de tre normalformer igennem slavisk. Som der også nævnes tidligere, er normalformerne blevet brugt til at kvalitetssikre vores database model. Heriblandt sikre at der ikke var nogen gentagne grupper, og at tabellerne kunne bruges ved flere poster og det sidste at vi fjernede alle tabellerne som ikke var afhængige af nogen nøgler.

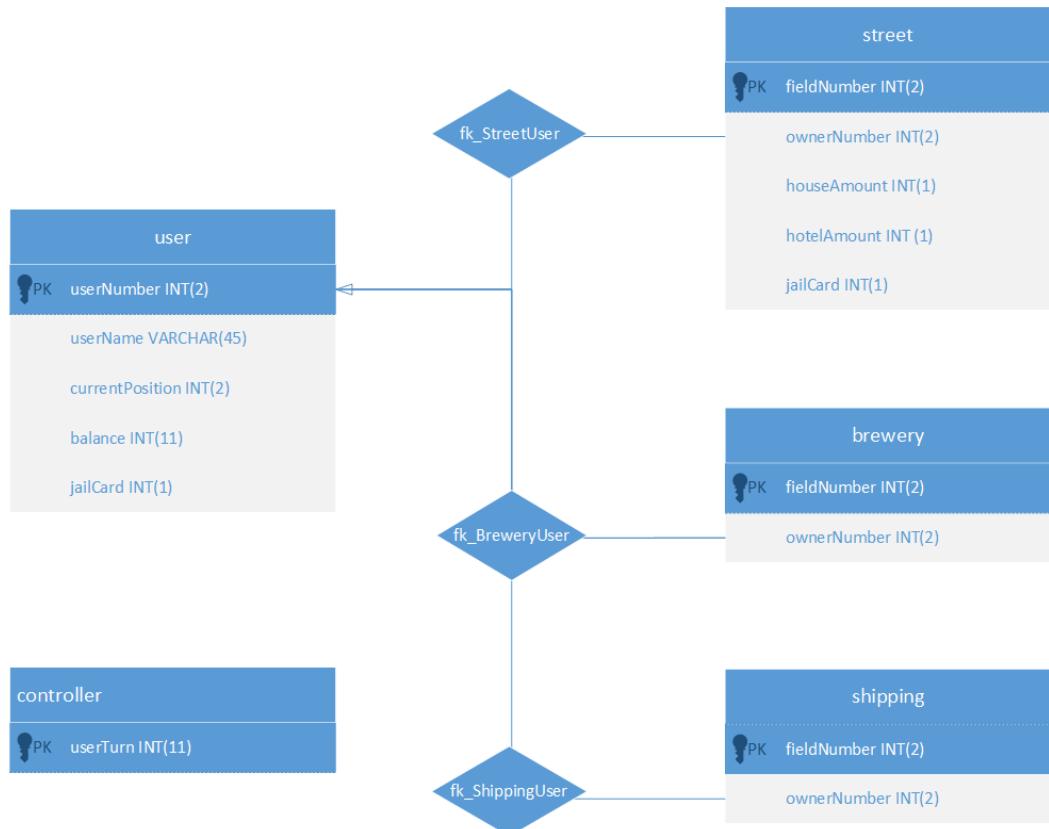
I nedestående diagram kan normaliseringen af vores database ses, fra ingen normalisering hvor alle elementerne står i samme tabel, til de på 3. normalform er fordelt ud til flere forskellige tabeller for at undgå dårligt databasedesign som beskrevet ovenfor.

Matador:					
fieldnumber	ownerNumber	houseAmount	hotelAmount	usernumber	userName
currentPositio	balance	jailCard	userTurn		
1. Normalform					
User:					
usernumber	userName	currentPositio	balance	jailCard	userTurn
street					
fieldnumber	houseAmount	hotelAmount	ownerNumber		
2. Normalform					
User:					
usernumber	userName	currentPositio	balance	jailCard	userTurn
Street					
fieldnumber	houseAmount	hotelAmount	ownerNumber		
3. Normalform					
User:					
usernumber	userName	currentPositio	balance	jailCard	
Street:					
fieldnumber	houseAmount	hotelAmount	ownerNumber		
Shipping:					
fieldnumber	ownerNumber				
Brewery:					
fieldNumber	ownerNumber				
Controller:					
userTurn					

Figur 5.7: Normalisering af databasen

5.8.2 ER-Diagram (130714, 134226, 134228)

Et ER-diagram minder meget om domænemodellen, da diagrammet bruges til at skabe et visuelt overblik over data vi ønsker at gemme i databasen, og giver et indblik i relationerne mellem entiteterne. Diagrammet er i høj grad blevet benyttet som et værktøj når vi skulle udvikle designet af databasen, og implementere den i programmet.



Figur 5.8: ER-diagram

5.8.3 Relation skema (134228)

Vi konstruerer vores logiske model med det formål at afspejle databasens struktur. Vi udleder relationerne ud fra vores E/R-model, og validerer disse relationer med normalisering, og på den måde konstruerer vores logiske model. Formålet er at optimerer den relationelle model, ved at minimal resundans, og dermed give brugeren en database der fylder lidt, nem at hente data fra og nem at vedligeholde.

User PRIMARY KEY	(userNumber, userName, currentPosition, balance, jailCard) userNumber
Controller PRIMARY KEY	(userTurn) userTurn
Street PRIMARY KEY FOREIGN KEY	(fieldNumber, ownerNumber, houseAmount, hotelAmount) fieldNumber ownerNumber REFERENCES User (userNumber) ON DELETE CASCADE, ON UPDATE CASCADE
Shipping PRIMARY KEY FOREIGN KEY	(fieldNumber, ownerNumber) fieldNumber ownerNumber REFERENCES User (userNumber) ON DELETE CASCADE, ON UPDATE CASCADE
Brewery PRIMARY KEY FOREIGN KEY	(fieldNumber, ownerNumber) fieldNumber ownerNumber REFERENCES User (userNumber) ON DELETE CASCADE, ON UPDATE CASCADE

Tabel 3: Relation skema

6 Implementation

6.1 Software dokumentation (134223)

Programmet er blevet dokumenteret ved hjælp af Javas indbyggede dokumentation system, Java doc.

Dette er blevet gjort ved at tilføje Java doc kommentarer til alle public metoder i koden, og private i controller klassen. Grunden til at der ikke er tilføjet kommentarer til de generelle private metoder skyldes at disse oftest blot er en specificering af beskrivelsen som allerede står i public metoderne.

Udover dokumentation for de enkelte metoder er der også tilføjet forfattere på hver klasse, hvilket er i overens med hvem der har stået for at lave denne.

6.2 Sekvens diagram (134218)

Vi har lavet et sekvensdiagram over sekvensen, hvor man slår et slag samt køber feltet man lander på. I dette tilfælde lander man på et felt af typen shipping. Ved hjælp af sekvensdiagrammet A.11 viser vi, hvilke metoder man skal bruge, for at slå et slag og derefter købe et felt af typen shipping.

Måden det er lavet på er at vi har lavet referencer, for at prøve at gøre det mere simpelt og overskueligt. Dog har vi ikke lavet alle referencerne, kun dem der benyttes, når man køber et felt af typen shipping. Det vil sige at vi benytter os af referencerne playerMove A.12 og instanceOfShipping A.13.

(Se Appendix A.11, A.12 og A.13 for Sekvens diagrammer)

6.3 Design klassediagram (134218)

Design klassediagram er lavet til at give et overblik over vores klasser, for det færdige program og hvordan de hænger sammen. Klassediagram er en visuel oversigt som viser associationer mellem klasserne og hvordan det teknisk hænger sammen, ment på en mere objektorienteret måde. Design klassediagrammet A.14 giver et struktureret overblik over deres nedarvninger, altså hvis en klasse er subklasse af en anden. Derudover viser diagrammet også klassernes navn, deres variable samt deres metoder.

(Se appendix A.14 for Design klasse diagram)

7 Test (134223, 134228)

7.1 Strategi for test

7.1.1 White-box test

Formålet med at udføre whitebox-test er at få gennemløbet alt kode i programmet. For at få gennemløbet alt kode, har vi valgt vi at gøre dette som en user-test. Dvs. vi kører programmet igennem ved at starte et nyt spil, hvor vi vil udnytte vores kendskab til koden, og på den måde gennemløbe så meget af koden som muligt.

7.1.2 Black-box test

Ved en black-box test behandler vi programmet, som navnet antyder, som en sort boks hvor vi ingen kendskab har til selve systemet. Dvs. at testen foregår ved at give programmet en række data, og derefter analysere det output programmet givet. Denne test anses for at være en succes hvis programmet giver det forventede output. Vi har valgt at lave fire black-box tests:

- Spillerens konto
- Spillerens position
- Feltets ejer
- Køb af hus

7.1.3 Database test

Formålet med at teste databasen, er at sikre vi har et stabilt og veldesignet database-design, der lever op til de krav og analyser vi har lavet. Vi ønsker altså at teste om databasen er oprettet rigtigt, og FOREIGN KEYS og PRIMARY KEYS er defineret rigtigt. I vores test har vi fokuseret på hvor vidt vores FOREIGN KEYS er defineret rigtigt, og opfylder de krav vi har stillet i analysen. Vi har valgt at lave testen direkte i MySQL.

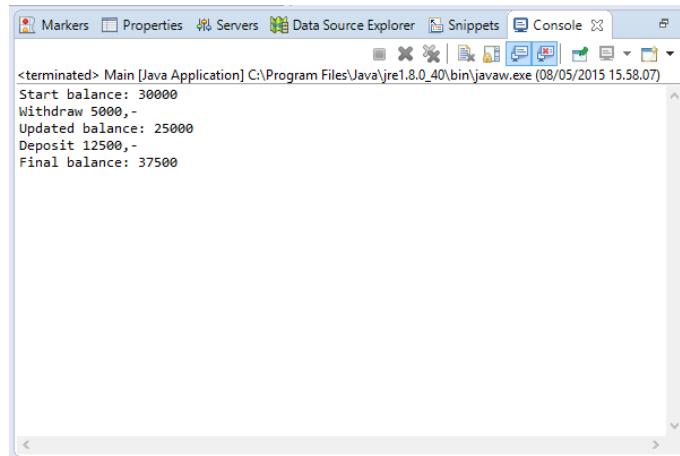
7.2 Testen

7.2.1 White-box test

Selve testen foregik ved at vi tre fra gruppen satte os og startede et spil. Vi benyttede her vores kendskab til koden og gennemgik så meget som muligt. Når problemer opstod gik vi direkte i koden og debuggede for at finde det præcise sted koden slog fejl, og dermed løse problemerne.

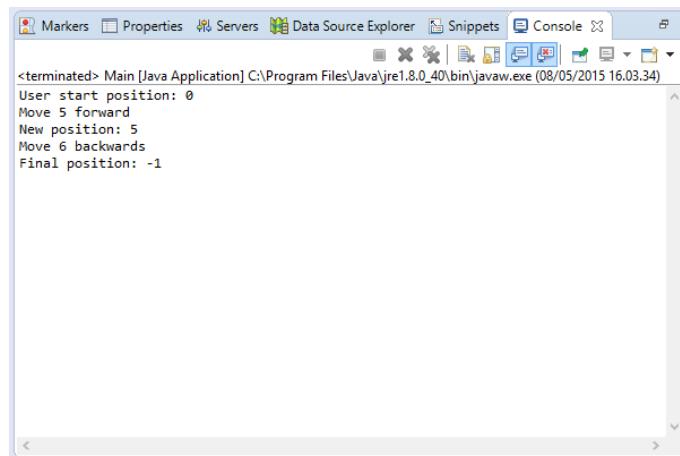
7.2.2 Black-box test

Spillerens konto: Testen blev udført ved at først trække penge fra spillerens konto, og efterfølgende sætte penge på spillerens konto.



A screenshot of an IDE's Console tab. The output window shows the following text:
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_40\bin\javaw.exe (08/05/2015 15.58.07)
Start balance: 30000
Withdraw 5000,-
Updated balance: 25000
Deposit 12500,-
Final balance: 37500

Spillerens position: Testen blev udført ved at først at rykke spilleren nogle positioner frem, og derefter nogle positioner tilbage.



A screenshot of an IDE's Console tab. The output window shows the following text:
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_40\bin\javaw.exe (08/05/2015 16.03.34)
User start position: 0
Move 5 forward
New position: 5
Move 6 backwards
Final position: -1

Feltets ejer: Testen blev udført ved først at kontrollere feltets nuværende ejer. Derefter satte vi 'Tester' som ejer, og kontrollerede igen. Til sidst fjernede vi ejer igen, og kontrollerede igen ejer af feltet.

```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_40\bin\javaw.exe (08/05/2015 16.23.04)
Hvidovrevej Owned by: null
Setting owner to Tester
Hvidovrevej Owned by: Tester
Remove owner
Hvidovrevej Owned by: null
```

Køb af hus: Testen blev udført ved at sætte 'Tester' til at købe hus uden at eje alle felter i den pågældende farve. Antallet af huse kontrolleres inden testen fortsætter.

Herefter blev 'Tester' ejer af alle felter i farven, og sat til at købe hus igen. Antallet af huse kontrolleres igen.

```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_40\bin\javaw.exe (08/05/2015 16.33.53)
Tester owns field 2, tries to buy house
Rødovrevej has 0 houses
Set Tester to own field 4, which are blue,
and it allows him to buy houses, since he own all blue fields
Buy house
Rødovrevej has 1 houses
```

7.2.3 Database test

Delete Parent row: Testen udføres ved at slette en række i User tabellen, og se hvilken effekt det har på 'Child rows' i Street tabellen. (Appendix A.15)

Update Parent row: Testen udføres ved at opdatere en række i User tabellen, og se hvilken effekt det har på 'Child rows' i Street tabellen. (Appendix A.16)

Update Child row: Testen udføres ved at opdatere en række i Street tabellen, og se hvilken effekt dette har. (Appendix A.17)

7.3 Test konklusion

7.3.1 White-box test

Problem:	Når spillet forsøges gemt anden gang opstår der fejl ved 'Duplicate Primary Keys'. Problemet opstår kun når spillet er gemt, lukket ned, åbnet igen og hentet, og så efterfølgende forsøges gemt igen. Kan sagtens eksekvære når der gemmes flere gange i samme spil.
Løsning:	Problemet lå i at når spillet blev hentet, var der i koden glemt en tæller for Shipping. Dvs. spillet hentede det første Shipping felt ud 4 gange. Problemet opstod så når spillet skulle gemme det samme felt 4 gange, da PRIMARY KEY så selvfølgelig også vil være den samme, og dermed opstår 'Duplicate Primary Keys'.
Problem:	Når seneste gemte spil hentes ned, giver spillet forkert lejepris på Shipping og Brewery. Når spillerne lander på en af felterne skal der betales lejepris som om at alle felter af typen Shipping eller Brewery er ejet af den samme spiller.
Løsning:	Vi kunne ikke lokalisere hvor i koden dette problem præcis opstår, og dermed endnu ikke fundet en løsning

7.3.2 Black-box test

Spillerens konto: Testen var vellykket da vi fik det forventede output.

Spillerens position: Testen var delvist vellykket da vi ved at rykker spilleren fremad fik det forventet output. Ved at rykke spilleren baglæns havde vi forventet en fejl ville opstå, da det ikke burde være muligt for spilleren at rykke baglæns.

Feltets ejer: Testen var vellykket, med forventet output da ny ejer blev registreret.

Køb af hus: Testen var vellykket, da det ikke var muligt for 'Tester' at købe hus uden at eje alle grunde i den pågældende farve.

Da 'Tester' var ejer kunne han købe huse og de blev registreret.

7.3.3 Database test

Delete Parent row: Testen var vellykket, da vi ved at slette en række i User 'Parent row' fik samme resultat i 'Child row'. Dvs. de rækker i Street hvor User med userNumber 1 stod som owner, blev slettet.

Update Parent row: Testen var vellykket, da vi her så at 'Child row' i Street blev opdateret sammen med dens 'Parent row' i User.

Update Child row: Testen var vellykket, da det ikke var muligt for os at opdatere i en 'Child row'.

8 Brugervejledning (134223)

8.1 Opstart spillet

Spillet startes op, enten via et IDE program (så som Eclipse), eller ved at åbne Java Archive filen.

Herefter har kan der vælges mellem at lave et nyt spil eller hente det seneste gemte spil.

Nyt spil

Når der startes et nyt spil bedes brugeren at vælge hvor mange spillere der skal være med i det nye spil.

Der kan vælges at være mellem to og seks spillere, begge tal inkluderet.

Der angives herefter navne på det valgte antal deltagere.

Herefter er spillet begyndt, og den spiller nummer et starter.

Gemt spil

Det sidste gemte spil bliver hentet fra databasen og gjort klar til at begynde igen.

8.2 Oversigt over spillet

Formålet med spillet er at blive den rigeste spiller, og eneste som ikke er gået i fallit.

Dette gøres hovedsageligt ved at købe og sælge grunde, i ens egen fordel.

Disse grunde kan købes af banken, hvis man lander på feltet, og feltet ikke ejes af en anden spiller.

Hvis feltet ejes af en anden spiller skal denne betales i forhold til feltets lejepris.

Lejesummen kan forhøjes ved at købe huse og hoteller på ens grunde.

Ud over grunde findes der også andre felter, så som prøv lykken som giver en mulighed for at trække et kort, hvis ordre må efterkommes.

Det ender også sommetider med at spillere bliver sat i fængsel.

Matador kan spilles af 2 til 6 personer, der hver starter med 30.000 kr.

8.2.1 Felter

Herunder er en kort beskrivelse for de forskellige grunde og hvad deres rolle er i spillet.

Der findes hovedsageligt to forskellige slags felter: Felter der kan ejes, og felter der ikke kan ejes.

8.2.1.1 Felter der kan ejes

Disse slags felter kan købes, sælges og pantsættes til banken, men derudover kan de også sælges til andre spillere.

Priserne på grundende kan aflæses på brættet.

Når en grund ejes og en anden bruger rammer dette felt, skal denne bruger betale husleje til ejeren af feltet.

Grunde

Ud over den generelle beskrivelse ovenfor kan huslejen forhøjes ved enten at eje alle felter af samme farve, eller ved at købe huse eller hoteller.

Bryggerier

Bryggeriers husleje beregnes ved at multiplicere brugerens terningeslag med 100, eller 200 hvis ejeren har skødet til begge bryggerier.

Rederier

Rederiers husleje beregnes ud fra hvor mange rederier ejeren ejer.

1 rederi: 500 kr.	2 rederier 1000 kr.	3 rederier 2000 kr.	4 rederier 4000 kr.	
-------------------	---------------------	---------------------	---------------------	--

8.2.1.2 Felter der ikke kan ejes

Prøv-lykken

Disse felter betyder at brugeren tager et kort fra lykke-kort bunken, hvor de skal udfører det som står beskrevet.

Fængsel

Der findes to forskellige slags fængsel felter. Det ene er ”På besøg” som fungere som et helle felt, altså intet sker.

Det andet felt er ”De fængsles” hvor brugeren bliver sat i fængsel, ”På besøg”-feltet, og bliver der indtil der slås 2-ens. Brugeren modtager ikke 4000 kr. for at passere start.

Dette er undtaget hvis brugeren enten har et ”Kom ud af fængsel”-kort, eller har efter tredje runde og han stadig ikke er ude.

Hvis det sidste sker, skal brugeren betale 1000 kr. og rykker det antal felter som hans sidste slag viste.

Start

Start er hvor alle brugere starter deres spil, og derudover når denne passerer betales 4000 kr. til brugeren. Hvis man lander på feltet sker der ingenting.

Skatter

Der findes to forskellige skatte felter, det ene er indkomstskatten hvor der betales 4000 kr., eller 10% af brugerens saldo.

Den anden er ekstra-ordinær statsskat, hvor brugeren betaler 2000 kr.

Parkering

Dette felt er helle, og intet sker her på.

8.2.2 Huse og hoteller

Det er muligt at købe huse og hoteller på grunde, hvis man ejer alle af samme farve. Huse skal bygges jævnt over alle grundene af samme farve, dette vil sige at for at købe nummer to hus på et felt, skal alle andre felter have et hus. For at kunne bygge et hotel skal der have fire huse på de andre grunde.

Der kan kun bygges et hotel på hver grund, og der kan ikke placeres huse ved siden af et hotel.

8.2.3 Pantsæt

Det er kun muligt at pantsætte grunde, som er ubebygget. Har man allerede bygninger på grunden skal disse sælges først.

Ved pantsætning får brugeren halvdelen af grundens pris, men har ikke krav på at få husleje hvis nogle lander på grunden.

Brugeren kan genkøbe grunden for samme pris som han fik udleveret, da grunden blev pantsat.

8.2.4 Indbyrdes handel

Brugere kan bytte grunde, bryggerier og rederier mellem hinanden. Dette kan gøres ved at en bruger byder på en andens bruger grund, hvorefter den anden bruger skal godtage buddet.

Grunde hvorpå der er huse eller hoteller kan ikke byttes, og disse skal sælges først. Dog kan grunde der er pantsat godt sælges, men køberen skal genkøbe grunden for at kunne kræve husleje.

9 Konklusion (130714)

I denne opgave havde vi problemformuleringen at vi ønskede at lave en simulering af et matadorspil, der kunne gemmes i en database.

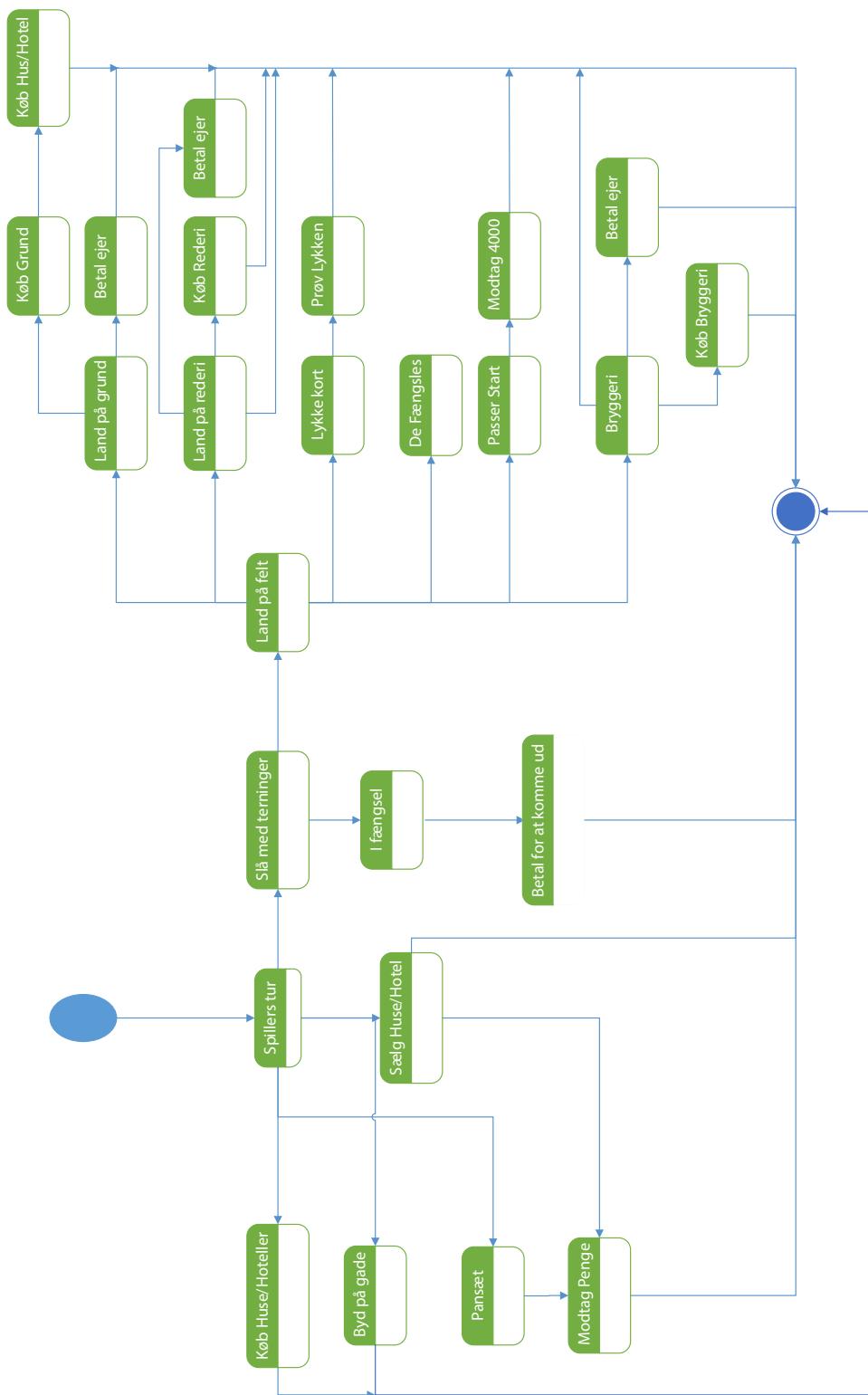
Vi har løst opgaven ved først at analysere problemet med fremstillingen af forskellige modeller, og benytte analysemetoder. Vi kom frem til et program kodet i java, som indeholdt alt funktionalitet og kontakten til databasen.

Databasedelen løste vi ved at benytte Oracle's MySQL workbench som vi kørte på vores egen server.

Det færdige program virkede efter hensigten og vi fik derfor opfyldt vores problemformulering. Det kan gennemføre et spil Matador, som hvis man sad med selve bræt-spillet, og spillet kan gemmes, hvis man ønsker at fortsætte på et andet tidspunkt.

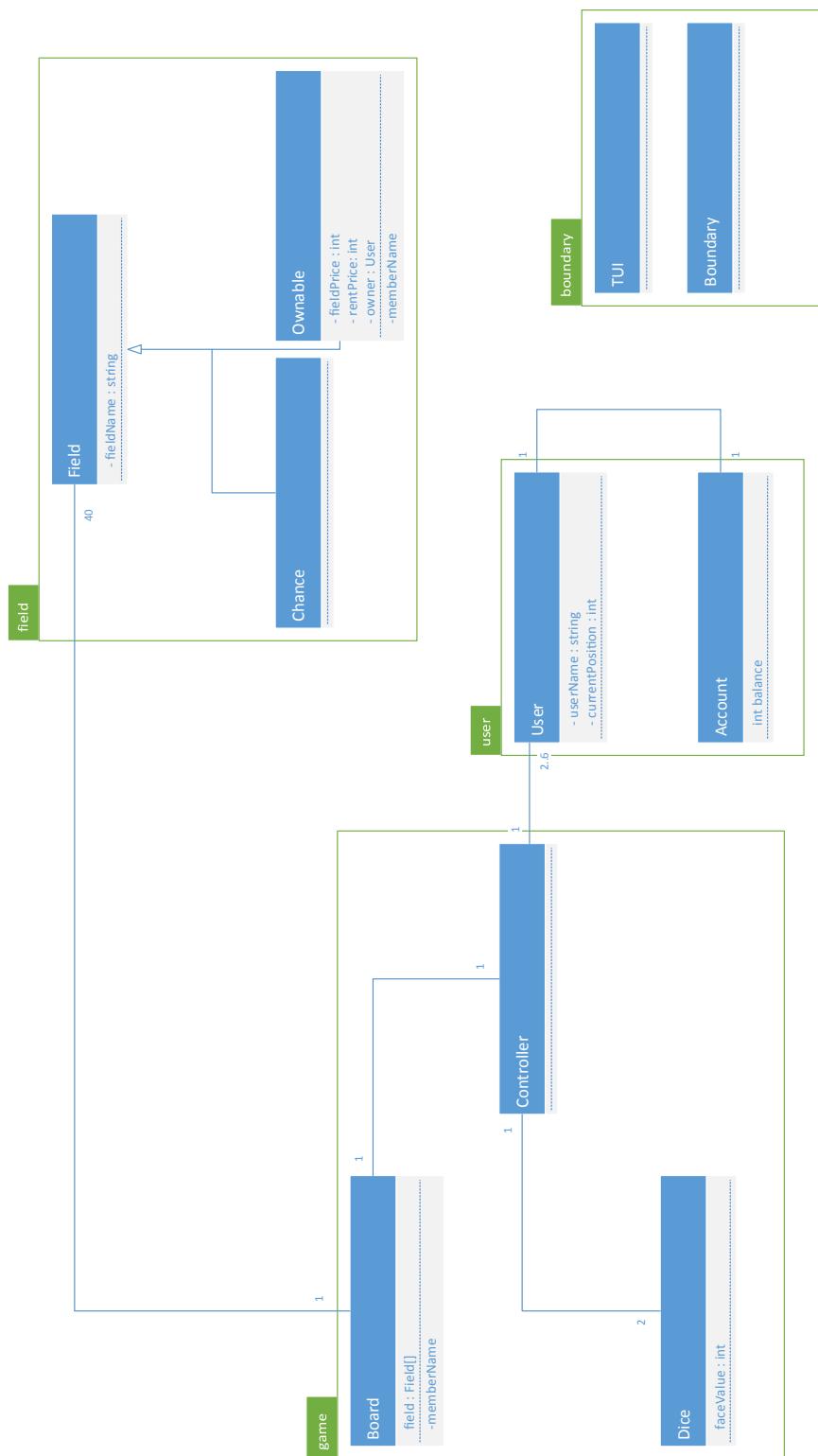
A Appendices

Tilstandsdiagram



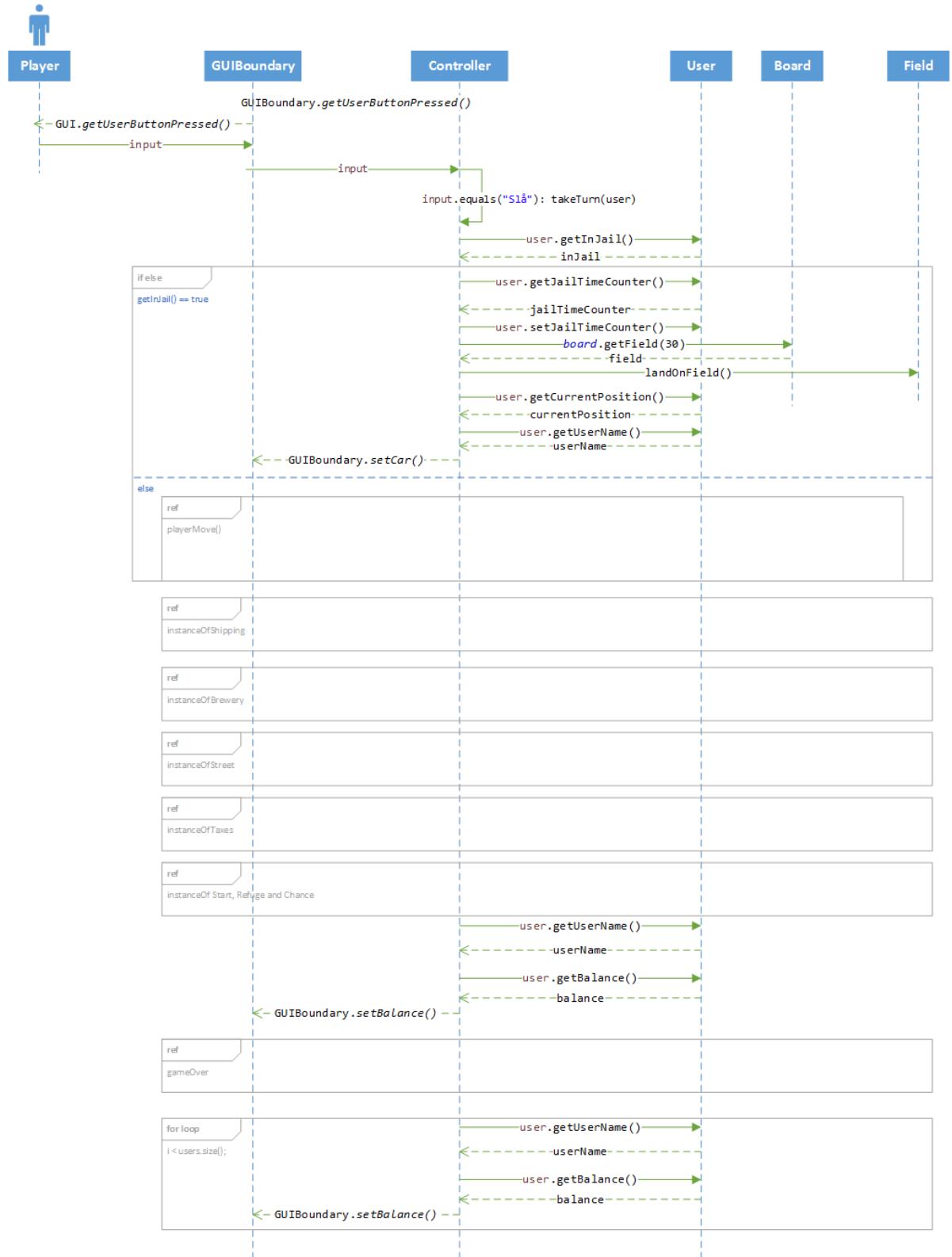
Figur A.9: Tilstands diagram

Domænemodel



Figur A.10: Domæne model

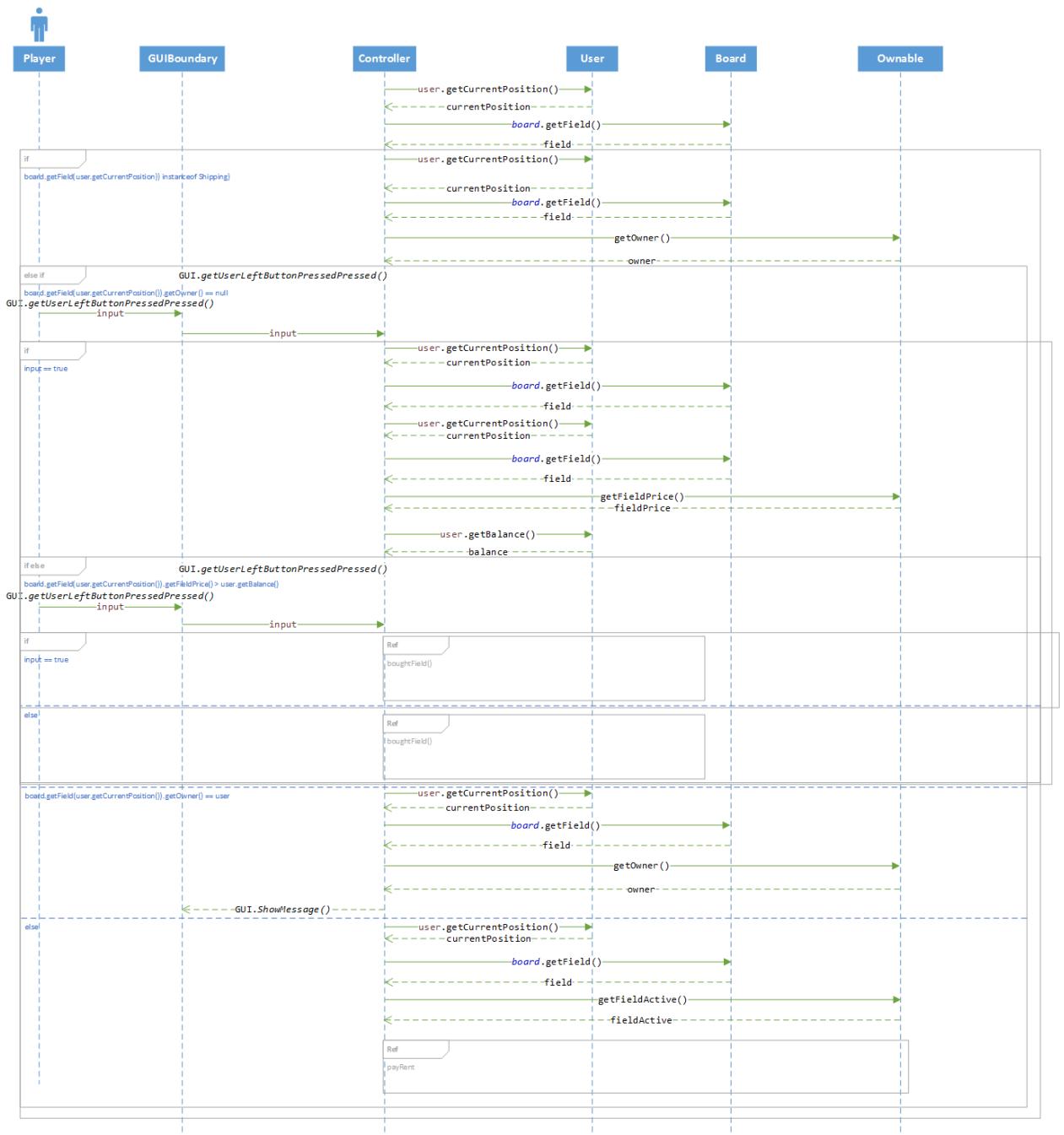
Sekvens diagram



Figur A.11: Sekvens diagram: Game

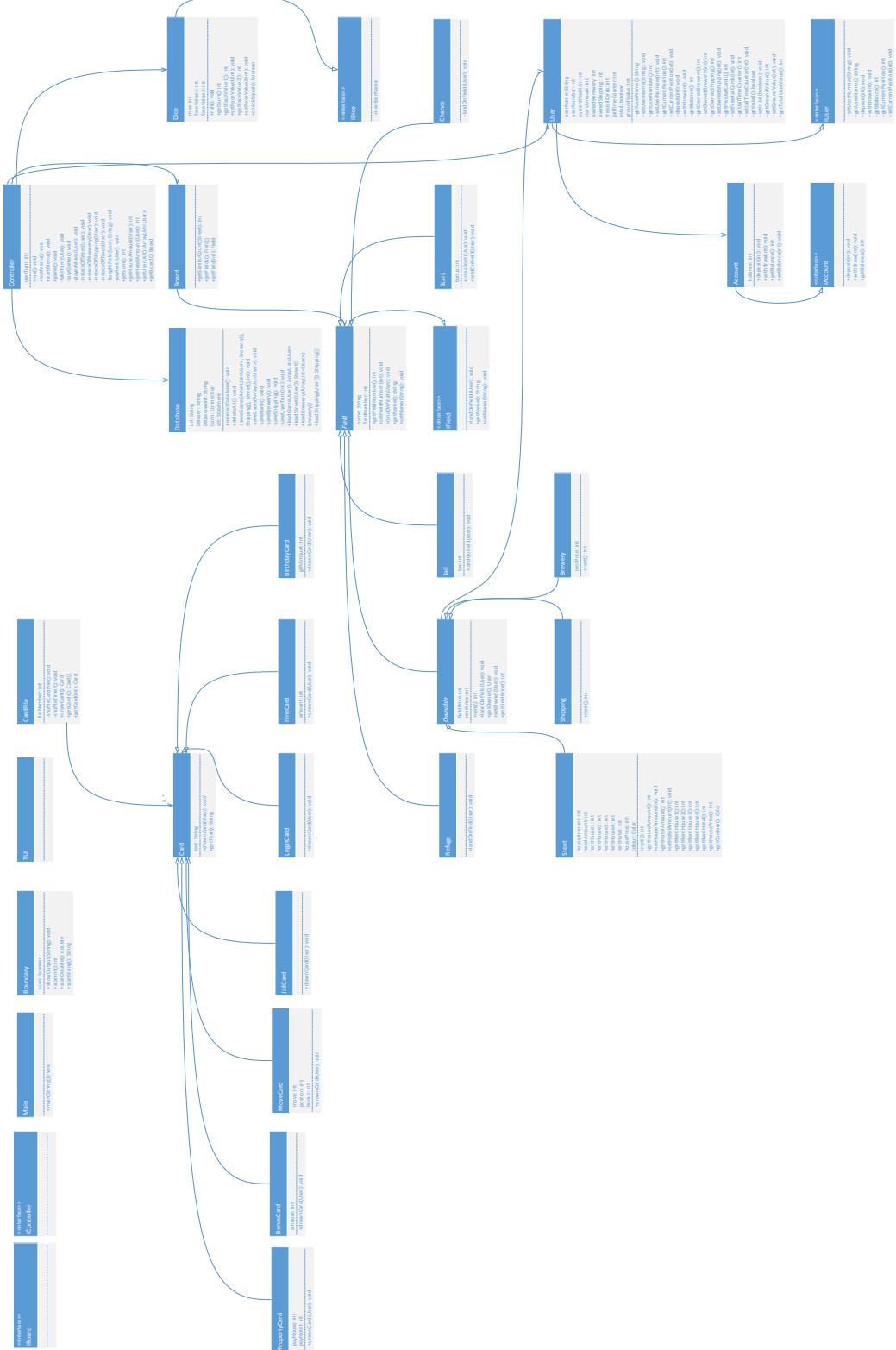


Figur A.12: Sekvensdiagramm: PlayerMove



Figur A.13: Sekvens diagram: InstanceOfShipping

Design klasse diagram



Figur A.14: Design klasse diagram

Database test

User					Street				
	userNumber	userName	currentPosition	balance	jailCard	fieldNumber	ownerNumber	houseAmount	hotelAmount
▶	0	Bank	0	0	NULL	2	0	0	0
	1	Andreas	18	20400	0	4	0	0	0
	2	Jokke	20	27600	0	7	0	0	0
	3	Bjarke	11	26000	0	9	1	0	0
						10	2	0	0
						12	0	0	0
						14	0	0	0
						15	0	0	0
						17	0	0	0
						19	1	0	0
						20	0	0	0
						22	0	0	0
						24	0	0	0
						25	0	0	0
						27	0	0	0
						28	0	0	0
						30	0	0	0
						32	0	0	0
						33	0	0	0
						35	0	0	0
						38	0	0	0
						40	0	0	0

```
DELETE FROM user WHERE userNumber = 1;
```

	userNumber	userName	currentPosition	balance	jailCard	fieldNumber	ownerNumber	houseAmount	hotelAmount
▶	0	Bank	0	0	NULL	2	0	0	0
	2	Jokke	20	27600	0	4	0	0	0
	3	Bjarke	11	26000	0	7	0	0	0
						10	2	0	0
						12	0	0	0
						14	0	0	0
						15	0	0	0
						17	0	0	0
						20	0	0	0
						22	0	0	0
						24	0	0	0
						25	0	0	0
						27	0	0	0
						28	0	0	0
						30	0	0	0
						32	0	0	0
						33	0	0	0
						35	0	0	0
						38	0	0	0
						40	0	0	0

Figur A.15: Test: Delete Parent row

User						Street				
	userNumber	userName	currentPosition	balance	jailCard		fieldNumber	ownerNumber	houseAmount	hotelAmount
▶	0	Bank	0	0	NULL	▶	2	0	0	0
	1	Andreas	18	20400	0		4	0	0	0
	2	Jokke	27	22400	0		7	0	0	0
	3	Bjarke	11	26000	0		9	1	0	0
							10	2	0	0
							12	0	0	0
							14	0	0	0
							15	0	0	0
							17	0	0	0
							19	1	0	0
							20	0	0	0
							22	0	0	0
							24	0	0	0
							25	0	0	0
							27	0	0	0
							28	2	0	0
							30	0	0	0
							32	0	0	0
							33	0	0	0
							35	0	0	0
							38	0	0	0
							40	0	0	0

```
UPDATE user SET userNumber = 4 WHERE userNumber = 1;
```

	userNumber	userName	currentPosition	balance	jailCard		fieldNumber	ownerNumber	houseAmount	hotelAmount
▶	0	Bank	0	0	NULL	▶	2	0	0	0
	2	Jokke	27	22400	0		4	0	0	0
	3	Bjarke	11	26000	0		7	0	0	0
	4	Andreas	18	20400	0		9	4	0	0
							10	2	0	0
							12	0	0	0
							14	0	0	0
							15	0	0	0
							17	0	0	0
							19	4	0	0
							20	0	0	0
							22	0	0	0
							24	0	0	0
							25	0	0	0
							27	0	0	0
							28	2	0	0
							30	0	0	0
							32	0	0	0
							33	0	0	0
							35	0	0	0
							38	0	0	0
							40	0	0	0

Figur A.16: Test: Update Parent row

User					Street				
	userNumber	userName	currentPosition	balance	jailCard	fieldNumber	ownerNumber	houseAmount	hotelAmount
►	0	Bank	0	0	NULL	2	0	0	0
	1	Andreas	18	20400	0	4	0	0	0
	2	Jokke	27	22400	0	7	0	0	0
	3	Bjarke	11	26000	0	9	1	0	0
						10	2	0	0
						12	0	0	0
						14	0	0	0
						15	0	0	0
						17	0	0	0
						19	1	0	0
						20	0	0	0
						22	0	0	0
						24	0	0	0
						25	0	0	0
						27	0	0	0
						28	2	0	0
						30	0	0	0
						32	0	0	0
						33	0	0	0
						35	0	0	0
						38	0	0	0
						40	0	0	0

```
UPDATE street SET ownerNumber = 4 WHERE ownerNumber = 1;
```

✖ 19 16:42:17 UPDATE street SET ownerNumber = 4 WHERE ownerNumber = 1
Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails ('matador`.`street', CONSTRAINT `fk_StreetUser` FOREIGN KEY (`ownerNumber`) REFERENCES `user` (`userNumber`) ON DELETE CASCADE ON UPDATE CASCADE)

Figur A.17: Test: Update Child row