# CDIO Final

# Monopoly

Gruppe nr: 12

Afleveringsfrist: 21/01/2018

02312 Indledende programmering E18

Gruppemedlemmer:


Rasmus Traub Nielsen
s185101

Jens Daniel Kramhøft
s175445

Michael Lund Jarberg
s185091

Nicolai Nisbeth
s175565

Tanja Sølvsten
s185089

Marina Schmidt Malling
s185104


Link til GitHub repository: https://github.com/jarberg/12_3uger

I alt 21 normal sider

# Timeregnskab

| | Analyse | Design | Implementation | Dokumentation |
|---|---|---|---|---|
| Jens Daniel Kramhøft | 9½ | 12 | 64 | 20 |
| Michael Lund Jarberg | 9 | 6 | 72½ | 16 |
| Marina Schmidt Malling | 10½ | 6 | 62½ | 17 |
| Nicolai Nisbeth | 11 | 13 | 68 | 16 |
| Tanja Sølvsten | 10½ | 7 | 49½ | 17 |
| Rasmus Traub Nielsen | 9½ | 6 | 49 | 7 |

**Indholdsfortegnelse**

**Timeregnskab** **1**

**Resumé** **4**

**Indledning** **5**

**1. Analyse** **6**
1.1 Vision 6
1.2 Kravspecifikation 6
MoSCoW 6
1.3 Use cases 10
1.3.1 Use case diagram 10
1.3.3 Use case beskrivelser 10
1.4 System sekvens diagrammer 14
1.5 Domænemodel 17

**2. Design** **18**
2.1 Designklassediagrammer 18
2.1.1 Visit Field Diagram 18
2.1.2 Chancekort Diagram 20
2.1.3 Player-Field relation Diagram 21
2.1.4 Localisation Diagram 22
2.2 Singleton Implementering 22
2.3 Sekvensdiagrammer 23
2.4 Arkitektur 25
2.5 GRASP 26

**3. Implementering** **29**
3.1 Arrays 29
3.2 Overloading 29
3.3 Køb af bygning 30

**4. Test** **31**
4.1 Struktur 32
4.2 Testmetoder 33
4.2.1 Testmiljø 33
4.3 Test cases 33
4.4 Kode dækning 36
4.5 Traceability Matrix 37
4.6 Brugertest 37

**5. Konfiguration** **39**
5.1 Versionsstyring 39
5.1.1 Maven 42
5.1.2 Java 42

**6. Projektforløb** **43**

**7. Konklusion** **44**

**8. Bilag** **45**

8.1 Supplerende specifikation 45

8.2 Use cases 45

8.3 Timeregnskab 49

# Resumé

Det følgende er en rapport over udviklingen af IOOuterActives "Matadorspil", med opdeling i de overordnede emner, analyse, design, implementering og test. Rapporten beskriver de grundlæggende udviklingsmetoder og software-principper bag implementeringen af spillet, der er bygget ud fra kundens vision om det klassiske spil. Kravene har dannet fundamentet for spillets udvikling. Herefter følger en grundig gennemgang af hvordan kunden kan installere, tilføje og vedligeholde spillet efter overførsel fra udviklingerne.

# Indledning

I denne rapport vil IOOuterActives version af det fulde Matador spil blive præsenteret. Rapporten vil belyse de vigtigste elementer for hvordan spillet er blevet udviklet. Projektet er blevet lavet ud fra software-principper. Først pensles kundens vision og krav ud. De bliver brugt til at skabe de use cases, der er fundamentet for hele projektet. Disse use cases bliver brugt til at designe domænemodellen og systemsekvensdiagrammerne, som giver en visuel forståelse for projektet. I implementering vil rapporten fremhæve de vigtigste elementer i programmets kode og til sidst er der en gennemgang af hvordan projektet er blevet afprøvet og hvordan hele projektforløbet har været.

# 1. Analyse

I følgende analyse tages der udgangspunkt i kundens vision, som i enighed med udviklerne bliver yderligere specificeret i kravspecifikationen.

## 1.1 Vision

IOOuterActive har fået en sidste opgave, hvor der skal udvikles en digital version af det fulde Matadorspil. Denne opgave er blevet udført på baggrund af kundens vision og projektlederens bemærkninger. Spillet er en forlængelse af opgaverne som IOOuterActive har leveret i de forrige uger, og reflektere derfor mange af de samme krav og visioner som kunden tidligere har givet udtryk for. Reglerne og opsætningen der bliver brugt i det interaktive spil er blevet hentet fra det originale brætspil. Kunden har overladt os en vurderingen af, hvilke der er de vigtigste elementer i spillet og disse bliver reflekteret i den næste paragraf, kravspecifikation.

## 1.2 Kravspecifikation

Kravene prioriteres alt efter vigtigheden af deres implementering. Vigtigheden bestemmes ud fra deres effekt på andre krav og ud fra hvor essentielt kravet er for spillet. Det er ud fra kundens vision ikke er nødvendigt for spillets funktionalitet at have samtlige krav og regler implementeret. Til prioriteringen anvendes MoSCoW metoden. Alle ikke-funktionelle krav prioriteres ikke efter MoSCoW metoden og fremgår i stedet i projektets supplerende specifikation, i bilag 8.1.

### MoSCoW

**Must have**

Liste over krav der **skal** implementeres.

- **M1:** Systemet skal spilles af 3-6 spillere.
- **M2:** Systemet skal lade spillerne vælge mellem 6 køretøjer.
- **M3:** Systemet skal lade spillernes figurer starte på “START”- feltet.

- **M4:** Systemet skal give spillerne en startkapital på 1500 kr.
- **M5:** Systemet skal lade spilleren kaste med terningen og få en sum.
- **M6:** Systemet skal kun rykke spilleren med uret det antal felter ternings sum giver.
- **M7:** Systemet skal lade spilleren modtager 200 kr. fra banken hvis denne spiller lander på/passere start-feltet.
- **M8:** Systemet skal lade ejendoms grunde starte uden ejer.
- **M9:** Systemet skal lade spilleren købe et ledigt felt når spilleren lander på feltet.
- **M10:** Systemet skal kræve husleje, hvis en spiller lander på et felt ejet af en anden spiller.
- **M11:** Systemet skal indikere hvilke spillere der ejer grundene.
- **M12:** Systemet skal lade en spiller rykke ud fra fængsel hvis spilleren betaler 50 kr.
- **M13:** Systemet skal eliminere en spiller, hvis spilleren skylder flere penge end spilleren har mulighed for at frembringe fra sine værdier.
- **M14:** Systemet skal vælge vinderen som den sidste spiller tilbage når alle andre er gået fallit.

**Should have**

Liste over krav der **burde** implementeres.

- **S1:** Systemet skal give en spiller en ekstra tur hvis spilleren slår to ens.
- **S2:** Systemet skal give spillerne mulighed for at udveksle fire huse til et hotel der koster fem gange så meget som et hus.
- **S3:** Systemet skal indikere om en grund er pantsat.
- **S4:** Systemet skal kræve dobbelt leje fra en grund hvis ejeren af grunden ejer alle grunde i samme farve.
- **S5:** Systemet skal lade spillerne pantsætte ubebyggede grunde til banken.
- **S6:** Systemet skal give spillerne mulighed for at købe deres pantsatte grunde tilbage med 10 procent i rente.
- **S7:** Systemet skal ikke kræve leje af pantsatte grunde når andre spillere lander på dem.

- **S8:** Systemet skal undlade at give spillere penge for at passere start når de sættes i fængsel.
- **S9:** Systemet skal lade en spiller rykke ud fra fængsel hvis spilleren har et friheds-kort.
- **S10:** Systemet skal lade en spiller rykke ud fra fængsel hvis spilleren slår 2 ens og får også et ekstra kast.
- **S11:** Systemet skal lade en spiller rykke ud fra fængsel hvis spilleren har været i fængsel over 3 omgange. Ved tredje kast der ikke er 2 ens skal spilleren betale 50 kr. i bøde samt rykke antal øjne.
- **S12:** Systemet skal lade spilleren trække det øverste kort i 'prøv lykke' bunken, hvorefter teksten på kortet udføres af spilleren.
- **S13:** Systemet skal lægge kortet som er blevet trukket tilbage nederst i bunken.
- **S14:** Systemet skal blande bunken med chancekort ved spillets start.
- **S15:** Systemet skal kræve enten 200 kr penge plus 10% af spillerens værdier fra spilleren når spilleren lander på indkomstskat feltet, efter valg af spilleren.
- **S16:** Systemet skal lade spillere bygge huse på deres ejede felter, hvor det så er givet at spilleren ejer alle felter af samme type, og at de andre felter af den type har en eller færre antal huse forskel fra de andre felter.
- **S17:** Systemet skal lade spillere bygge hoteller for 5*huspris på deres ejede felter, givet af spilleren ejer alle felter af den type, og at de andre felter af samme type har 4 huse eller et hotel.
- **S18:** Systemet skal sætte en spillers resterende grunde på auktion, når spilleren går fallit.

**Could have**

Liste over krav der **kunne** implementeres.

- **C1:** System skal smide en spiller i fængsel hvis spilleren har slået 2 ens 3 gange i træk.
- **C2:** Systemet skal lade en grund gå på auktion, hvis en spiller vælger ikke at købe den.
- **C3:** Systemet skal give spillere mulighed for at sælge bygninger til banken.

- **C4:** Systemet skal lade spillere sælge deres bygninger til halv pris.

**Won't have (this time)**

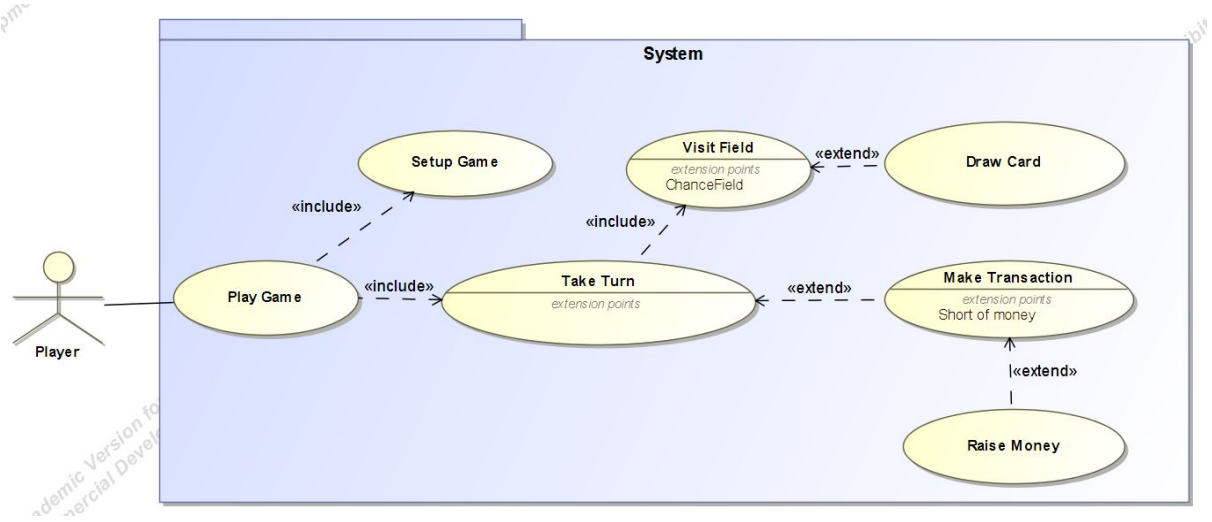
Der er ikke en liste over krav som **ikke** implementeres. Der er ikke benyttet nogle won't krav i dette projekt.

## 1.3 Use cases

Systemets use cases bliver defineret ud fra kundens vision og den udarbejdede kravspecifikation.

### 1.3.1 Use case diagram

Spillets primære og eneste aktør er "Player" som har relation til systemets kritiske use case "Play Game", se figur 1.1. "Play Game" indeholder sub-use cases som hver er ansvarlig for mindre dele af systemets funktionalitet. Fordelen ved at opdele "Play Game" i sub-use cases er en bedre fordeling af ansvar for at opnå high cohesion.




Figur 1.1

### 1.3.3 Use case beskrivelser

Samtlige use cases er beskrevet fully-dressed. Use cases "Play Game", "Setup Game", og "Take Turn" fremgår nedenstående i samme rækkefølge som de er opremset. De resterende use cases er i bilag 8.2.

Use casen **"Play Game"** er ansvarlig for eksekveringen af spillet.

| **Use case:** Play Game |
|---|
| **ID:** UC1 |
| **Kort beskrivelse:** Spillet spilles indtil vinder findes |
| **Primær aktør:** Spiller |
| **Precondition:** 3-6 spillere ønsker at spille spillet |
| **Main flow:**<br>1. include( Setup Game ).<br>2. Så længe spillet har mere end én spiller tilbage<br>2.1 Systemet beder spiller om at starte sin tur<br>2.2 include( Take Turn )<br>3. Systemet kårer vinderen |
| **Postcondition:** En vinder er fundet og spillet er afsluttet |
| **Alternative flows:** |

Use casen **"Setup Game"** er ansvarlig for at klargøre spillet.

| **Use case:** Setup Game |
|---|
| **ID:** UC2 |
| **Kort beskrivelse:** Spiller starter spillet og spillet klargøres |
| **Primær aktør:** Spiller |
| **Precondition:** Systemet er startet. |
| **Main flow:**<br>1. Systemet beder Spiller om at vælge sprog.<br>2. Spiller vælger spillets sprog.<br>3. Systemet beder Spiller om at angive antal af spillere.<br>4. Spiller vælger antallet af spillere.<br>5. For hver spiller<br>5.1 Systemet beder spiller om at indtaste deres navn.<br>5.2 Spiller indtaster deres navn.<br>6. For hver spiller<br>6.1 Systemet beder spiller om at vælge spillebrik.<br>6.2 Spiller vælger spillebrik.<br>7. Systemet opretter spillepladen med spillere. |

| |
|---|
| **Postcondition:** Spillet er klargjort |
| **Alternative flows:** |

Use casen **"Take Turn"** er ansvarlig for eksekveringen af en spillers tur.

| **Use case:** Take Turn |
|---|
| **ID:** UC3 |
| **Kort beskrivelse:** Den aktive spiller kaster med terningerne. Systemet rykker brikken og effekten af det pågældende felt udføres. |
| **Primær aktør:** Spiller |
| **Precondition:** Det er en spillers tur |
| **Main flow:**<br>1. Systemet kaster med terningerne.<br>2. Systemet rykker spilleren til et felt ud fra terningens værdi.<br>3. Hvis Spiller har passeret start så<br>3.1 Systemet giver Spiller 200 kr.<br>3.1.1 include( Make Transaction )<br>4. include( Visit Field )<br>5. Systemet viser tur muligheder<br>6. Spilleren vælger mulighed "Afslut Tur"<br>7. Turen slutter. |
| **Postcondition:** Ny spillers tur. |
| **Alternative flows:**<br>1a. Spiller er i fængsel<br>1.Systemet viser tre valgmuligheder.<br>2. Spiller udløser et 'Get out of Jail' kort.<br>2a. Spiller betaler 50 kr.<br>2b. Spiller slår efter 2 ens med terninger.<br>3. Spiller fortsætter sin tur terninger.<br>3a. Spiller kom ikke ud fængslet og slutter tur. |

6.a Spiller vælger mulighed “Saelg frihedskort”

1. Systemet betaler spilleren for kortet
2. Systemet fratager spilleren kortet

6.b Spiller vælger muligheden "Byt Grund"

1. Systemet spørger spiller om byttehandlens detaljer
2. Spiller angiver byttehandlen
3. Systemet gennemfører byttet hvis det blev godtaget af Spiller.
    3a. Spiller accepterede ikke byttet.

6.c Spiller vælger aktionen “Pantsaet Grund”

1 Spiller vælger det felt de vil pantsætte

2 Include( Make Transaction )

6.c Spiller vælger aktionen “Koeb pantsat grund tilbage”

1 Spiller vælger det felt Spiller vil købe

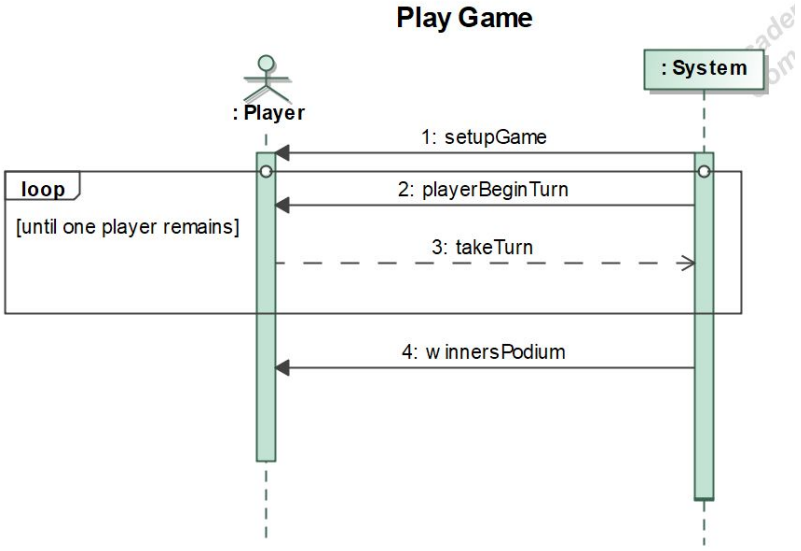
2 Include( Make Transaction )

6d. Spiller vælger mulighed "Koeb bygning"

1. Spiller angiver hvilken bygning
2. Include( Make Transaction )

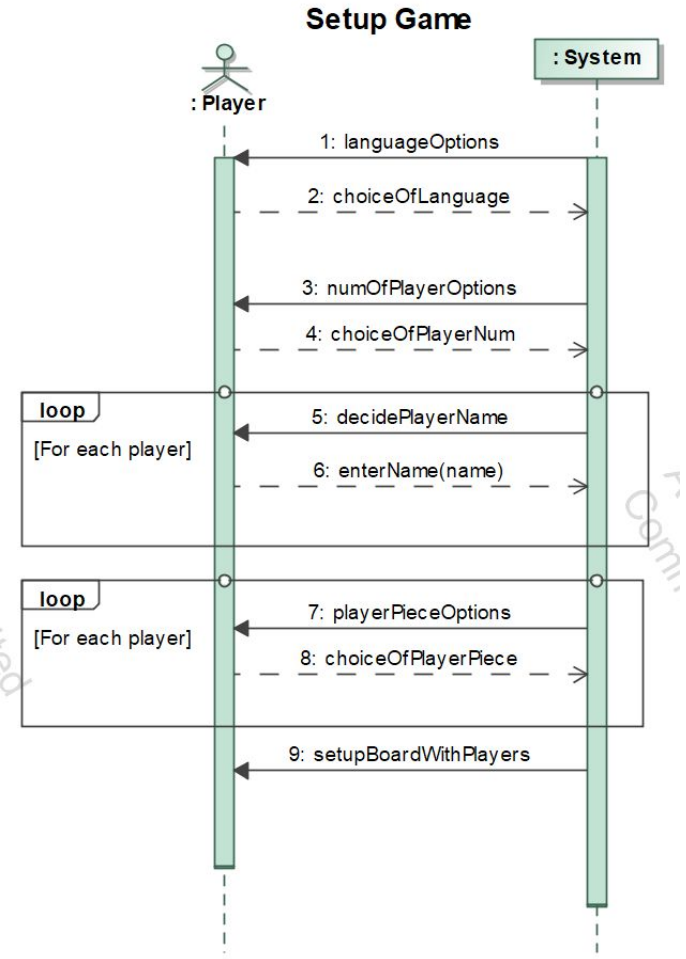
## 1.4 System sekvens diagrammer

I **"Play Game"** beder systemet spillerne om at fuldføre **"Setup Game"** og skiftevis **"Take Turn"** indtil der kun er én spiller tilbage i spillet, se figur 1.2. Afslutningsvis kåres den tilbageblivende spiller som vinder.



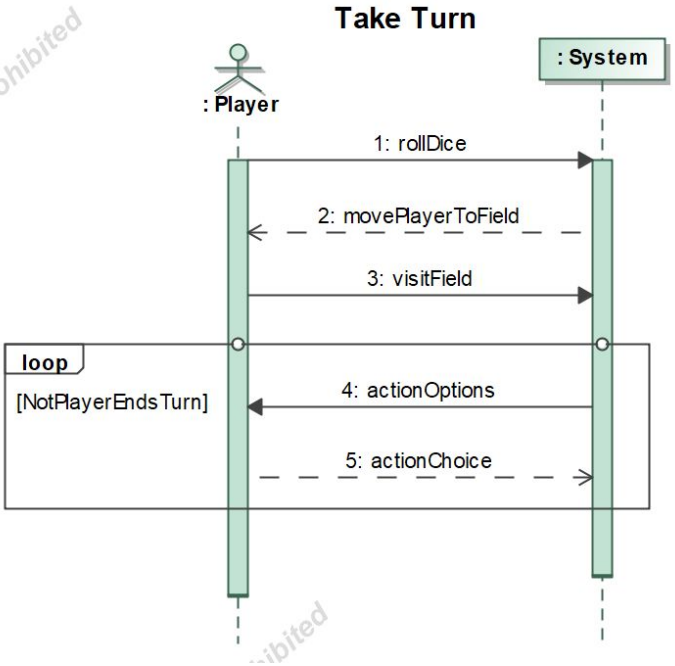

Figur 1.2

I **"Setup Game"** beder systemet spillerne om at specificere hvilke sprog de ønsker spillet skal være i, og antallet af deltagende spillere, se figur 1.3. For hver spiller angives spillerens navn og valg af spillebrik. Afslutningsvis opstiller systemet spillebrættet og placerer spillerne på startfeltet.




Figur 1.3

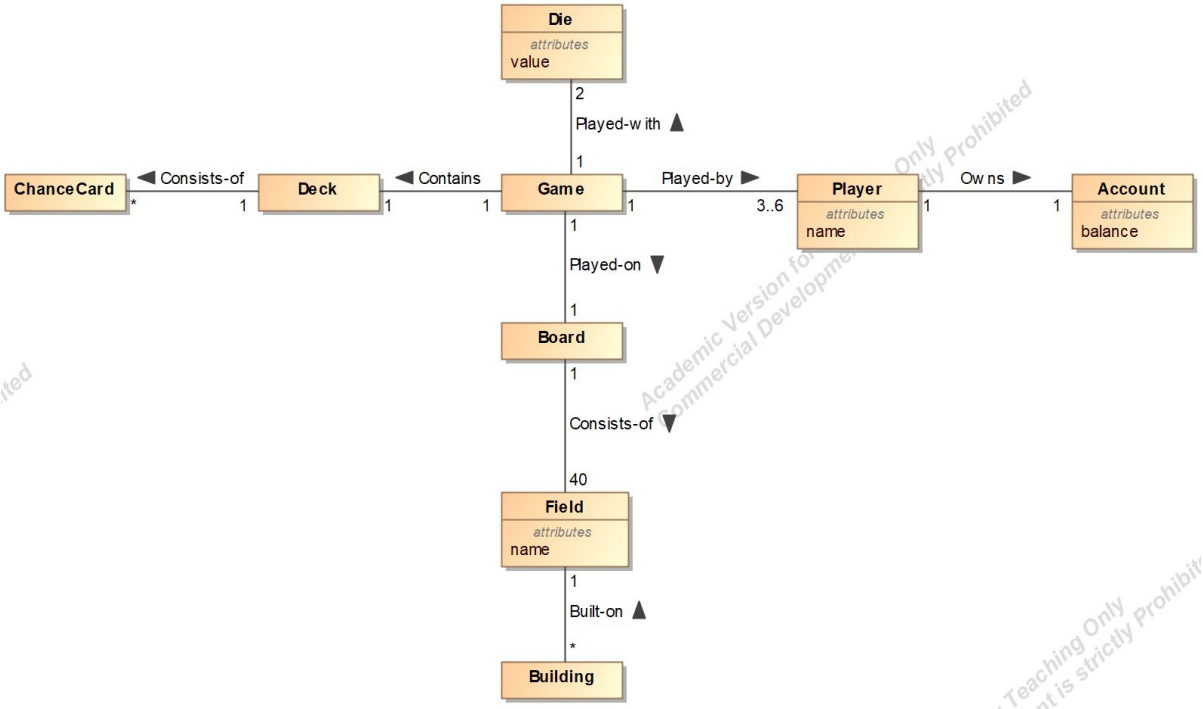
I **"Take Turn"** beder systemet spilleren om at slå med terningerne og terningernes sum angiver, hvor mange felter spilleren rykkes frem, se figur 1.4. Effekten af feltet, der landes på, udføres og spillerens balance påvirkes. Ved turens afslutning bliver spilleren præsenteret for de aktioner, som spilleren har adgang til på det nuværende tidspunkt i spillet. Når spilleren ønsker at afslutte sin tur vælges "End turn".




Figur 1.4

## 1.5 Domænemodel

Domænemodellen består af koncepter og deres indbyrdes relationer i den virkelige, ikke-software orienterede, verden. Koncepterne er blevet identificeret ud fra navneord i use case beskrivelserne og har til formål at agere som inspiration til designet af software arkitekturen.




Figur 1.5 Relation mellem Field og Player

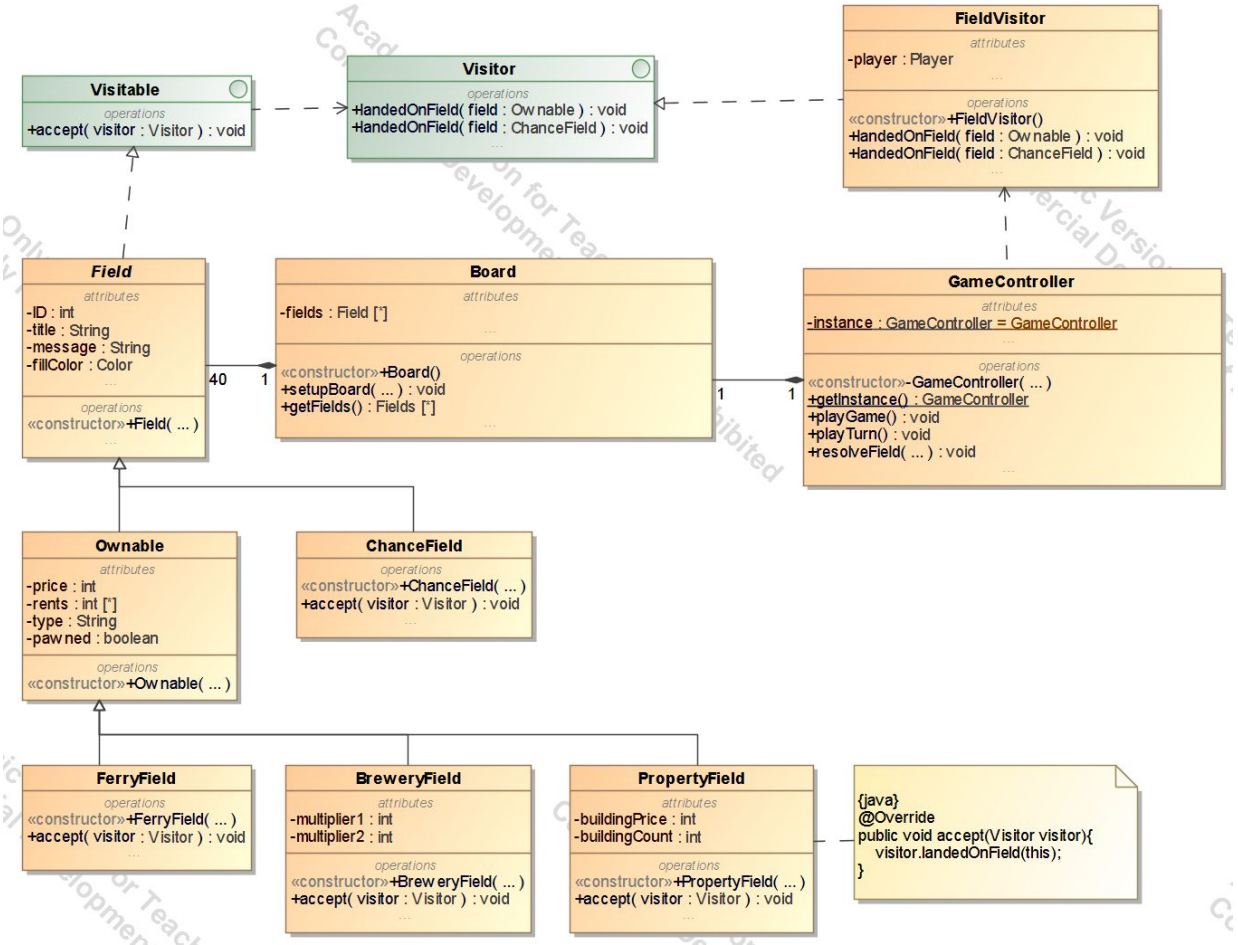
# 2. Design

Designet er opbygget ud fra analysen, GRASP-mønstre, design patterns og møder med kunden og ekstern projektleder.

## 2.1 Designklassediagrammer

I projektets forløb er der udviklet flere små designklassediagrammer med udgangspunkt i use cases, for lettere at danne forståelse for en specifik del af designet. Hver klassediagram indeholder kun de vigtigste klasser, væsentligste attributter og de operationer klassernes kernefunktionalitet er bygget op omkring.

### 2.1.1 Visit Field Diagram

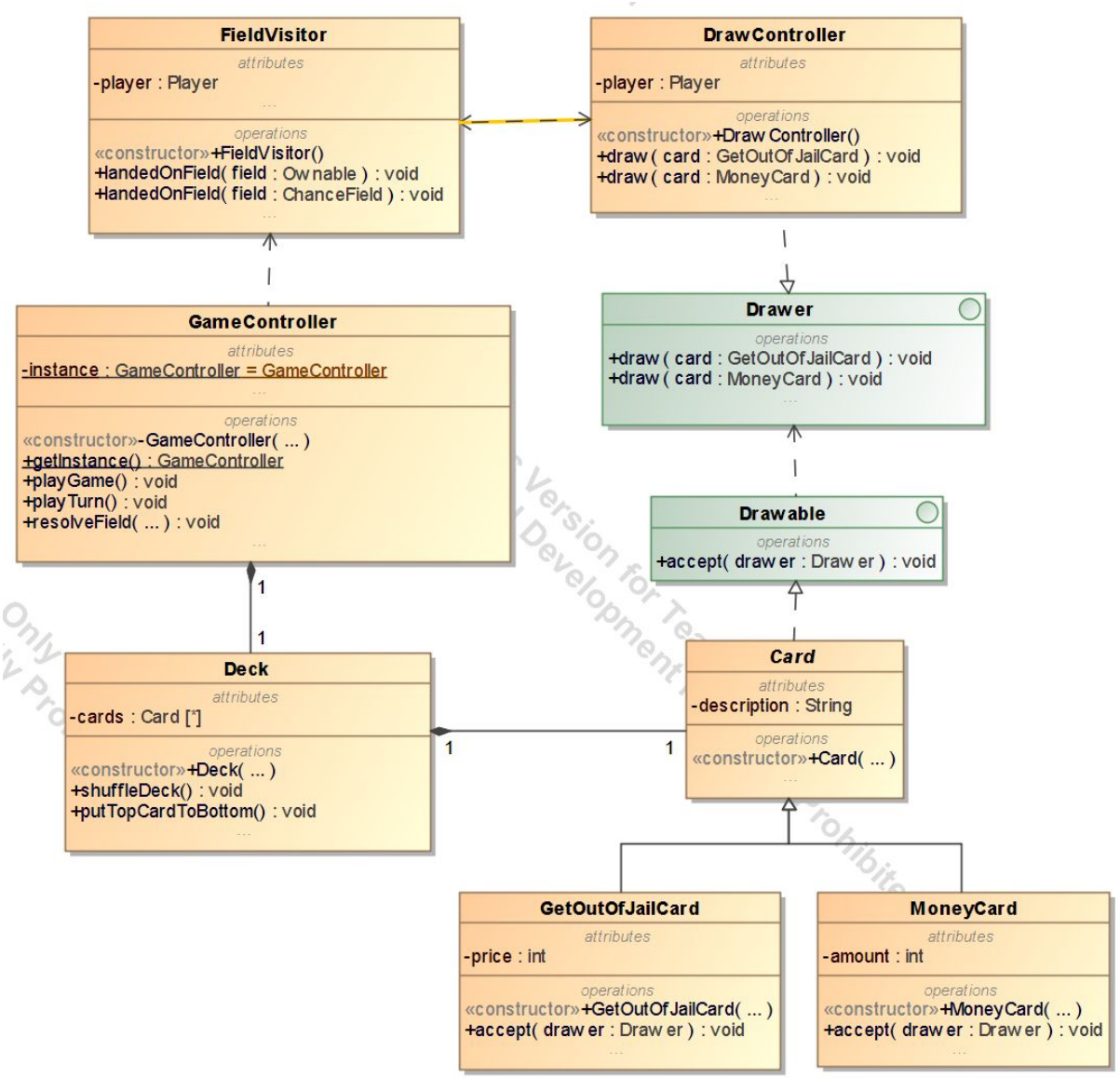


UC6: Visit Field[1]

I designet af den centrale del af Matadorprojektet: felterne og den logik, der udføres når man lander på dem, anvendes der visitor pattern. Visitor pattern gør det muligt at have én klasse med ansvar for alle de forskellige typers logik, i stedet for at have en controller klasse for hver felt type. Det gør designet mere overskueligt og fleksibelt. I designet er der anvendt polymorfi og nedarvning af de forskellige slags felter, der alle er en subklasse af den abstrakte "Field" klasse for at undgå redundans af kode. "Field" klassen implementerer et interface "Visitable". Det tvinger alle Fields til at implementere "Accept" metoden, der kan gøres som vist i noten ved "PropertyField" klassen. Felterne ved at de skal acceptere en klasse der implementerer et andet interface "Visitor" som parameter, men de ved ikke at den konkrete implementation af "Visitor", "FieldVisitor", er en del af Controller pakken. Det er i den, at logikken sker, når en spiller lander på et felt. Visitor pattern gør det enkelt for en Controller at implementerer funktionaliteten, når en spiller lander på et felt. Blot lave en instans af "Visitor" med den spiller hvis tur det er, og bed feltet der landes på om at acceptere spilleren.

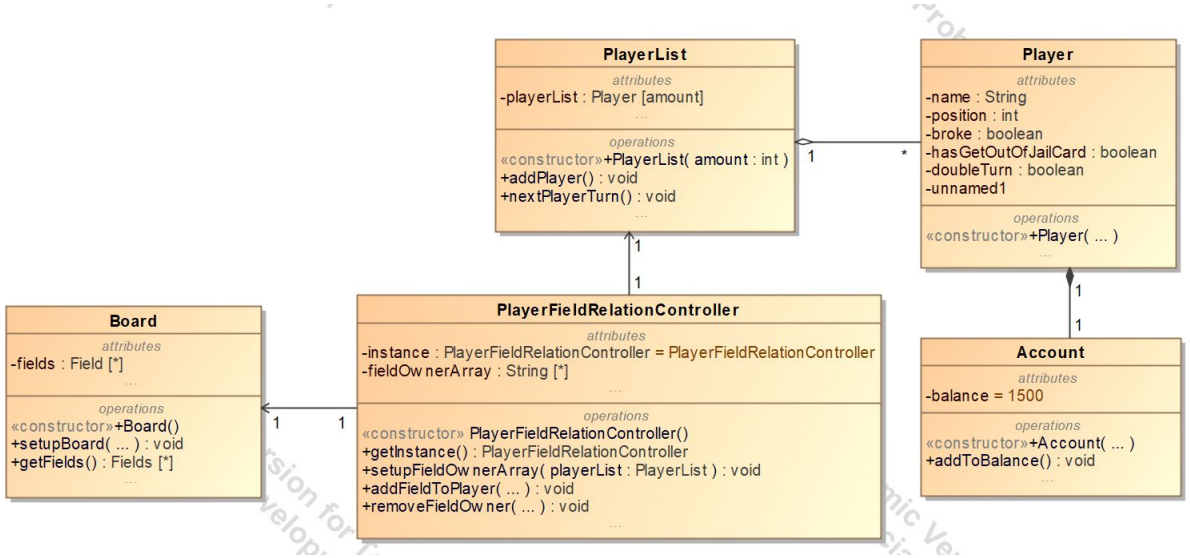
[1] Dele taget fra gruppe 16 CDIO 3

### 2.1.2 Chancekort Diagram



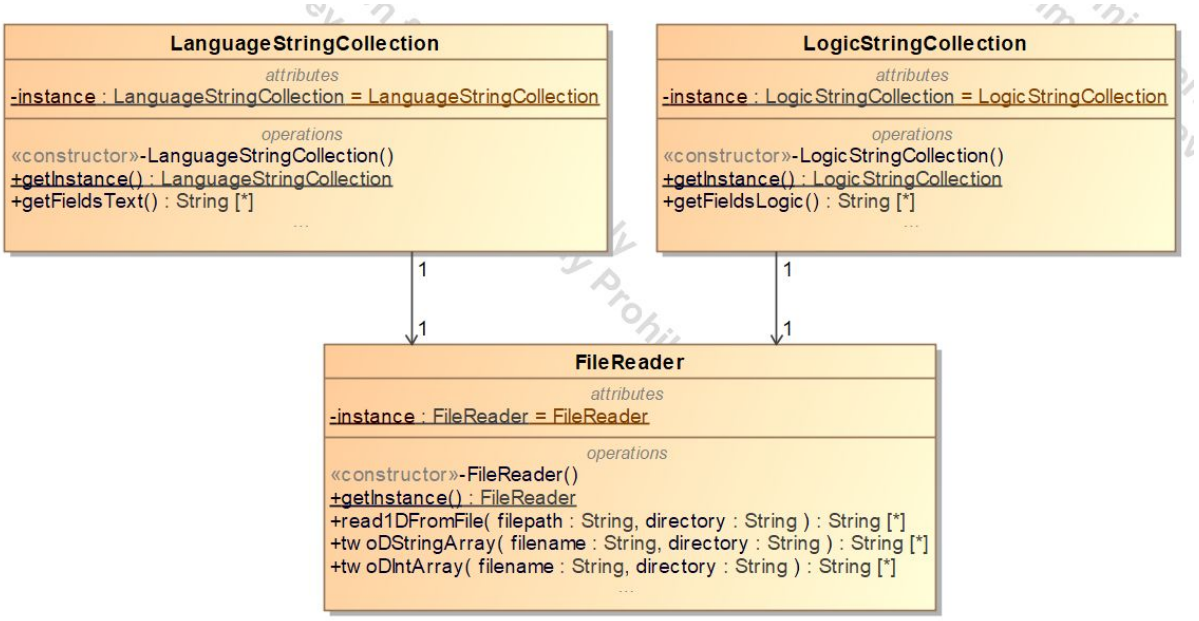

I tilfælde af at spilleren lander på et ChanceField bliver der lavet en instans af "DrawController" med samme spiller, og det trukket chancekort accepterer spilleren. Derfor er det nødvendigt for "FieldVisitor" at have kendskab til "DrawController", da en spiller kun kan trække et chancekort når der landes på et chance felt. Skulle et nyt type felt eller chancekort blive tilføjet til spillet, så kræver det at feltet implementerer den samme linje metode, at en tilsvarende metode tilføjes til "Visitor" interfacet, og at "FieldVisitor" så definerer logikken for feltet.

### 2.1.3 Player-Field relation Diagram




I Matador er det essentielt at spillerne kan købe felter. Der er undgået en dobbeltkobling direkte fra spiller til board ved at have en "PlayerFieldRelationController" klasse som har kendskab til både "Board" og "PlayerList", og fungerer som bindeled imellem dem. Til at holde styr på hvilke felter en spiller ejer, har "PlayerFieldRelationController" et todimensionelt String array med navnet 'fieldOwnerArray'. Arrayet er bygget op ligesom et HashMap, hvor nøglen er spillerens navn af typen String og værdien er et array af felt id'er af typen String.

### 2.1.4 Localisation Diagram




Spillet er designet således at koden er uafhængig af matador-spillets sprog og basale spilværdier. Ved spillets start bliver brugeren bedt om at vælge et sprog, Dansk eller Engelsk som default. Sprogvalget er stien til en af underpakkerne, Dansk og Engelsk, der ligger i ressourcepakken. Begge underpakker indeholder tekstfiler med beskrivelser af chancekort, felter og beskeder til spillerne. Dette gør det nemt for kunden at tilføje yderligere sprog til spillet. Ved tilføjelse af et tredje sprog kræver det blot en pakke med sprogets navn, som indeholder oversatte tekst filer. Det samme er tilfældet for felterne og chance kortenes værdier, som kræver at man ændrer i tekst filerne under logikpakken hvis man eksempelvis ønsker andre priser på felterne. Både Logic- og LanguageCollection klasserne er singletons, da der aldrig skal være mere end en af hver, samt for at undgå at indlæse den samme tekst flere gange.

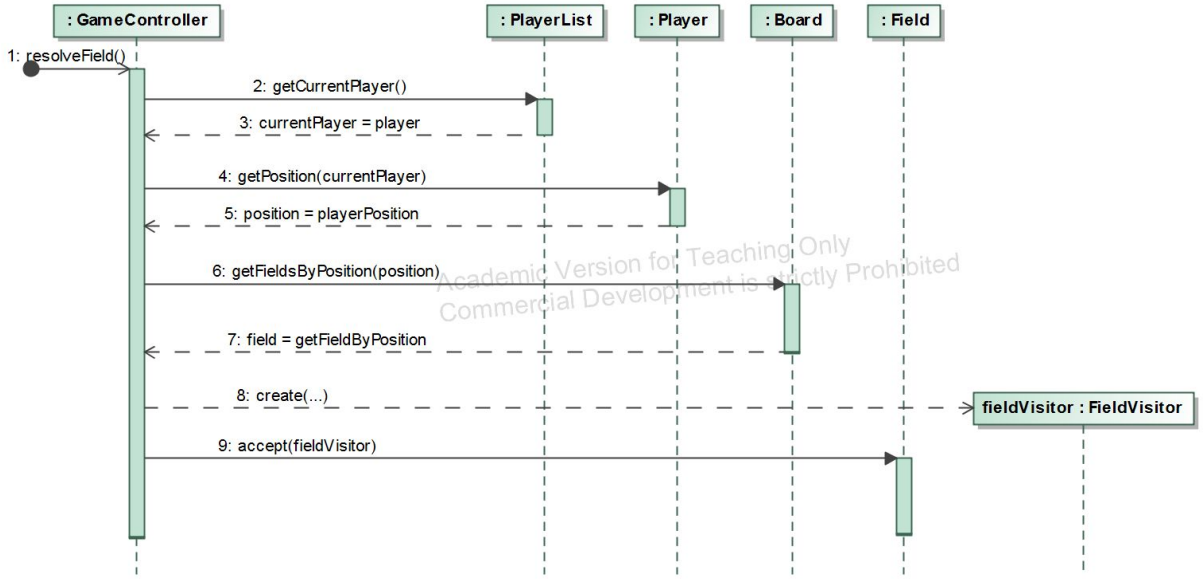
## 2.2 Singleton Implementering

Vi har anvendt Singleton Pattern for at sikre os at der kun bliver oprettet ét objekt af de klasser, hvor oprettelse af flere instanser af klassen ville resultere i redundans eller fejl. Dette

gør sig fx gældende i "ViewController"-klassen der kontrollerer brugergrænsefladen, da det altid er den samme brugergrænseflade der skal vises noget på.

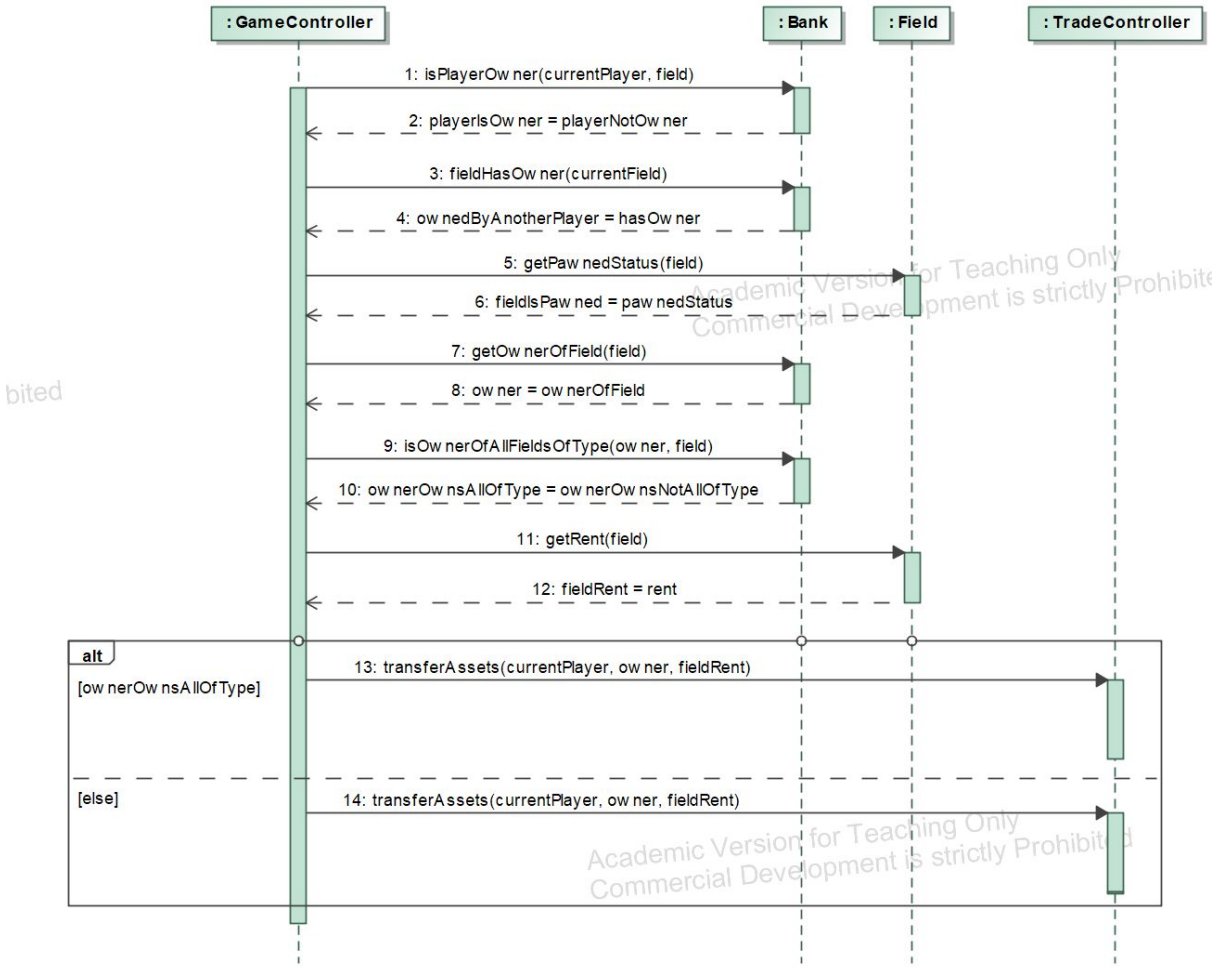
## 2.3 Sekvensdiagrammer

I dette projekt er spillets gang opdelt i to sub-usecases, "Setup Game" og "Take Turn", som begge er del af den kritiske use case Play Game. I det nedenstående sekvensdiagram antages det at "SetupGame" allerede er færdigkørt og første spiller er landet på et felt.




Metoden "ResolveField" kaldes i "GameControlleren" hvorpå den nuværende spiller hentes fra "PlayerList" objektet. Spillerens nuværende position hentes fra spiller objektet og "GameControlleren" finder det felt som er tilsvarende til spillerens position i "Board" objektet. Et nyt "FieldVisitor" objekt skabes som modtager feltet gennem den implementerede abstrakte metode 'accept' som efter felt typen kalder den givne metode for feltet.

Dette diagram antager at en spiller er landet på et "PropertyField" der er ejet af en anden spiller.



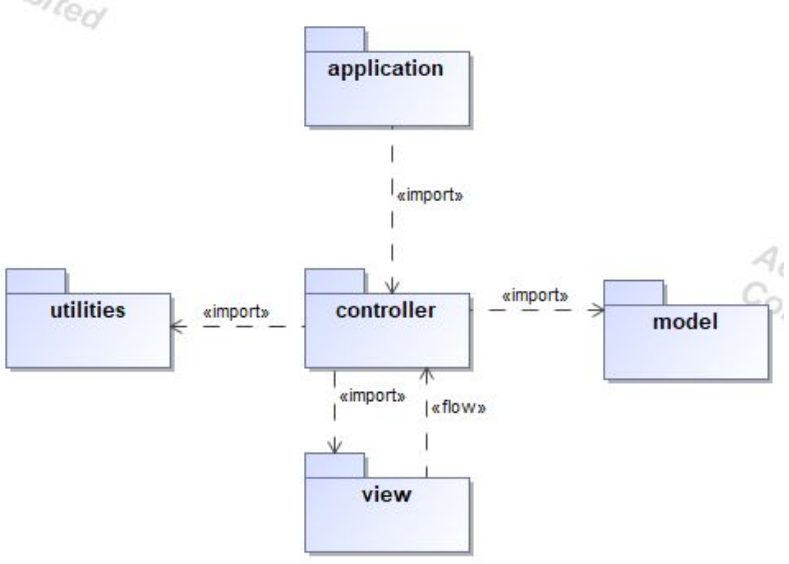

"GameControlleren" checker gennem "PlayerFieldRelation" klassen om det felt den nuværende spiller står på ejes, om der er en anden spiller som ejer feltet, om feltet er pantsat, hvem ejeren er, og om ejeren ejer alle felter af samme type.

"GameControlleren" ved nu, at det er en anden spiller som ejer feltet og at spilleren ikke har pantsat det. Der overføres feltets husleje fra den nuværende spilleren til ejeren hvor lejen er baseret på om ejeren ejer alle felter af samme type.

## 2.4 Arkitektur

Dette projekt anvender arkitekturmønsteret Model View Controller (MVC) ved at opdele de fleste software-klasser i pakker med tilsvarende navne. 'Model' pakken omhandler de objekter der repræsenterer dele af spillet. 'View' delen af projektet består af de visuelle elementer som brugeren interagerer med. I dette projekt optræder den udleverede grafiske brugergrænseflade "MatadorGUI" som en outsourced view-pakke, der kommunikeres med gennem en controller. 'Controller' pakken består af opstilling og logikken af spillet. Det vil sige at den opretter, læser, og manipulerer data fra 'Model' pakken, samt at den kommunikerer med vores 'View' gennem en "ViewController" for at opdatere hvad brugeren kan se, eller få at vide hvad brugeren har givet som input.

Opdelingen af pakkerne er til for at opnå designprincipperne low coupling og high cohesion af projektets pakker. Yderligere indeholder projektet en 'utilities' pakke der bruges til at læse informationen spillet opstilles med fra tekstfiler, og en 'application' pakke der står for at starte spillet.



## 2.5 GRASP

### 2.5.1 Low Coupling

Projektet er opbygget med lav kobling i sinde. Dette ses tydeligst i de mere enkle klasser i model pakken som "Account", der kun skabes og anvendes af Player objekter, "DieSet" som udelukkende anvendes i "GameController", samt i de størrer klasser som "DrawControlleren" der kun bruges af "FieldVisitor" klassen som også kun bruges af "GameControlleren". Den abstracte "Card" klasse som bliver skabt og delt igennem et Deck objekt. "DieSet" der bliver skabt, håndterer og deler dens information med andre klasser af "GameControlleren". Singleton klassen "FileReader", der står for alt indlæsning af filer, bruges kun af Logic- og Language-controllerne, der hver holder på den indlæste tekst til videre brug af andre klasser. "ViewControlleren" er det eneste led mellem de andre controllere og den externe view pakke 'MatadorGUI'.

### 2.5.2 High Cohesion

Projektet, som er opbygget efter objektorientede principper, og ved brug af MVC arkitektur opdeling, er klasserne blevet inddelt i pakker med klasser, der tilhører de bestemte pakketyper. "ViewController" håndterer alt det, som der bliver vist via brugergrænsefladen. Det vil sige at den har med alle brugergrænsefladem etoderne at gøre, og at det kun er den klasse som holder styr på de metoder. Det er "PlayerFieldRelationController" klassen som håndterer alt som har med at gøre med hvem der ejer hvilke felter. Den har kun de metoder som har noget med felternes status at gøre. "Player" klassen er den eneste som kender til metoderne i "Account" klassen, altså spillerens pengebeholdning, og alle metoder som har noget med spilleren at gøre. Det er "FieldVisitor" klassen der har logikken som ved hvad der præcist skal ske på det bestemte felt som spilleren er landet på, ved hjælp af diverse 'landOnField' metoder. "DrawController" klassen er den som håndterer chance-kortenes metoder efter samme princip. Det er så hver enkelt chancekortklasse har funktionerne, og en samlet klasse har metoderne til hver enkelte chancekort. "FileReader" klassen er specifikt den som håndterer hvordan .txt filer bliver læst og hentet til de tilhørende klasser. Det er vedrørende hvordan de forskellige sprog-filer læses, og hvordan logik-filerne til chancekortene bliver læst.

### 2.5.3 Creator

“Player” er den klasse som har det overordnede ansvar for at oprette “Account” klassen, som er den klasse der gør det muligt for en spiller at holde styr på sine penge. Det er ”Board” klassen som opretter ”Fields” klassen, da uden ”Field” klassen ville der ikke eksistere nogen felter på boardet. ”Deck” klassen er den som opretter ”Card” klassen, fordi det er ”Deck” klassen som indeholder ”Card” klassen, så for at indeholde noget, så bliver ”Card” klassen oprettet. Det er ”ViewControlleren” som kreerer GUI playeren, da det er ”ViewControlleren” som holder styr på brugergrænsefladen og dermed opretter en GUI player så det er muligt for spilleren at se sin brik i spillet.

### 2.5.4 Controller

Spillets logik og mere komplekse sammenfletninger af klasser er opdelt i specialiserede controllers ud fra principperne, information expert, high cohesion, creator og i sammenhæng med usecases. ”GameControlleren” håndterer usecases som ‘SetupGame’, ‘PlayGame’ og ‘TakeTurn’ som er sammensætninger af de andre controllerer og tilhørende model klasser. ”ViewControlleren” indeholder udelukkende metoder til opdateringer af GUI og Brugerinput. ”FieldVisitor” håndterer alle felt usecases ”DrawControlleren” håndterer alle kort usecases. Spiller transaktioner og felt ejerskab er i det analoge matador spil styret af en spiller som er deklareret bank fra spillets start. Da dette er unødvendigt at implementere i vores spil er banken splittet op i to controllerer med hvert deres ansvars område. ”PlayerFieldRelationControlleren” håndterer Players ejede felter. ”TradeControlleren” indeholder alle spillets transaction metoder og kald dertil.

### 2.5.5 Information Expert

”PlayerFieldRelationControlleren” er et særligt eksempel på vores tilgang til information expert. Overordnet har denne controller ansvaret for at holde styr på hvilke felter ejes af hvilken spiller. Controlleren har også ansvaret for at administrere hvilke felter der har huse. hvor mange huse de har og om en spiller ejer alle af samme type felt som eksempler. Klassen har for det meste brug for mindst et Player object for at kunne søge efter ejerskab men kender også til board og felter hvor relationen mellem ejer og felt opbevares i et String Array. ”Board” og ”Deck” klasserne er eksperter når det komme til Fields og Cards. De kender kun

til LanguageCollection og LogicCollection klasserne ud over deres egne pakkeklasser, da de også er Creators af de samme og har brug for at få tekst og logic fra filerne.

### 2.5.6 Polymorphism

Dette koncept er står frem i, som eks, board pakken med felterner til spillebrættet. Alle felter er underklasser af ”Field” klassen, der indeholder felternes fælles attributter, ID, Title Subtitle, Message, fieldColor. samt getter metoder til disse. Som nævnt i forklaring af projektets design nedarver de forskellige felttyper fra den abstrakte klasse ‘Field’. Dette tillader at give de forskellige typer felter deres egne attributter og metoder der er relevante for deres brug i spillet. Udover det er der dog også fælles egenskaber mellem felterne BreweryField, FerryField, og PropertyField. Derfor er de alle subklasser af den abstrakte klasse Ownable.  Ownable og Field klasserne er abstrakte fordi de ikke skal kunne instantieres - og de ville også skulle implementere den abstrakte metode ‘accept(Visitor visitor)’.Måden en spiller lander og løser et felts effect er sat op med interface klassen “Visitable “ som Field klassen implementere så alle underklasserne af Fields kan modtage et object “her en spiller” som implementere interface klassen “Visitor”.

# 3. Implementering

Projektet er efter projektlederen ønsker implementeret i Java 1.8 - altså Java med sprogniveau 8.

## 3.1 Arrays

Projektlederen har bedt om en implementering der ikke er afhængig af Java's egen gennemtestede, udbredte og praktiske API i form af Collections som 'ArrayList' eller 'HashMap'. Gennem projektet er der derfor anvendt Arrays, og det har været nødvendigt at lave metoder som den følgende, der tilføjer et element til et Array.

```
private String[] addToStringArray(String[] array, String newString){
  String[] newArray = new String[array.length + 1];
  for (int i = 0; i < array.length; i++) {
     newArray[i] = array[i];
  }
  newArray[array.length] = newString;
  return newArray;
}
```

Den ovenstående metode modtager to parametre - en reference til et Array af referencer til String objekter, samt en reference til et String objekt der skal tilføjes. Et nyt tomt String Array med plads til én String-reference mere en det modtagne Array bliver initialiseret, og et for loop itererer over det gamle Array og manuelt kopierer indholdet over til det nye. Den String-reference der skulle tilføjes til Arrayet bliver sat en på den ekstra plads, der på daværende tidspunkt er null, og det nye Array returneres. Til arbejde med Arrays er metoder som denne, og tilsvarende metoder der fjerner et element, meget nyttige. Det havde også været muligt at lave en metode der gav mulighed for at tilføje en dynamisk mængde String referencer ved at ændre metodens andet parameter fra `String newString` til `String... newStrings`, og lave en mindre omskrivelse af koden for at gøre den afhængig af længden på Arrayet 'newStrings'.

## 3.2 Overloading

Projektet anvender en praktisk implementering af overloading af metoder. Vi har anvendt en Visitor-pattern til at lande på felter og trække kort, hvilket resulterer i at et felts effekt bliver

løst på de nedenstående to enkelte linjer. FieldVisitoren's konstruktør tager en række information om den nuværende tilstand af spillet som parametre, der her er forkortet til "...".

```
FieldVisitor fieldVisitor = new FieldVisitor( ... );
currentField.accept(fieldVisitor);
```

Dette får feltet 'currentField' til at kalde den overloadede metode 'landOnField' med sig selv, og dermed også sin type, som parameter.

```
@Override
public void accept(Visitor visitor){
  visitor.landOnField(this);
}
```

Metoden har derefter forskellig logik alt efter hvilken felttype 'this' betegner. Det sikres at FieldVisitoren har implementeret alle disse metoder ved at klassen FieldVisitor implementerer interfacet Visitor.

```
void landOnField(ChanceField field);
void landOnField(GoToJailField field);
void landOnField(JailField field);
void landOnField(ParkingField field);
void landOnField(Ownable field);
void landOnField(TaxField field);
void landOnField(StartField field);
```

Her forstås et interface som en kontrakt, der garanterer at alle klasser der følger den, altså alle klasser der implementerer interfacet, har implementeret alle dets metoder, der alle er abstrakte.

## 3.3 Køb af bygning

Metoden der bliver kørt når en spiller vælger at bygge en bygning er god til at beskrive hvordan køb og handel er implementeret i spillet. Metoden ligger i "TradeController" og ser således ud:

```
public void buyBuilding(Player player, PropertyField field){
  while(player.getBalance() < field.getBuildingPrice()){
    raiseMoney(player);
    if(player.getBrokeStatus())
      break;
  }
  if(!player.getBrokeStatus()){
    transferAssets(player, -field.getBuildingPrice());
    field.addBuilding();
    viewController.addBuilding(field);
```

```
    }
}
```

Metoden tager en Player og et PropertyField som parametre. Hvis spilleren ikke har penge nok til at købe bygningen, bliver metoden 'raiseMoney' kaldt med spilleren som parameter indtil spilleren har penge nok, eller går fallit og breaker ud af while loopet. Hvis spilleren derefter ikke er gået fallit, betyder det, at spilleren har penge nok til at købe bygning, og spilleren trækkes penge fra og bygningen tilføjes til feltet.

# 4. Test

For at kunne udlevere et projekt som lever op til kundens krav, er programmet blevet afprøvet ved white box testing, i form af JUnit 4 tests. Der er blevet lagt fokus på de mest centrale metoder i koden, og code coverage er blevet tilset for hvert test. For at vise samhørighed i testene, er navnene på testene konsekvent blevet kaldt for 'should-' og derefter navnet på den funktion der bliver testet. Dette gør det lettere for udvikleren og kunden at se hvilke funktioner der bliver testet og det viser en sammenhæng mellem alle test klasserne. Dette gør det også lettere for kunden at gentage testene. Testene er så vidt muligt blevet lavet så det er muligt at benytte testene uden at bruge specifikke inputs. For eksempel ses der forneden testen til 'shouldMoveAmount()' metoden.

```
@Test
public void shouldMoveAmount() {

    int moveAmount = 5;
    MoveAmountCard card = new MoveAmountCard("desc", moveAmount);

    Player player = playerList.getCurrentPlayer();
    int playerOldPosition = player.getPosition();

    drawController.draw(card);

    assertEquals(moveAmount+playerOldPosition, player.getPosition());

}
```

Der bliver testet om spilleren bliver rykket et vis antal, når de trækker et bestemt chancekort. Testen er blevet opstillet så det er let at ændre på hvor mange skridt spilleren skal rykke.

## 4.1 Struktur

For hvert test er der blevet benyttet @Before , setUp() og @After, tearDown() metoder. @Before, setUp() er blevet brugt, da der i programmet findes flere tests, som benytter sig af at de samme objekter er blevet oprettet for at kunne implementeres. Ligeledes er der blevet brugt en @After, tearDown() metode, som frigiver @Before metoden, når hvert test er blevet kørt.

```
@Before
public void setUp(){
  gameCon = GameController.getSingleInstance();
  viewController = new ViewControllerStub();

  gameCon.setViewController(viewController);
  tradeController = TradeController.getSingleInstance();
  tradeController.setViewController(viewController);

  languageCollection = LanguageStringCollection.getSingleInstance();
  logicCollection = LogicStringCollection.getSingleInstance();
  fileReader = FileReader.getSingleInstance();
}
```

```
@After
public void tearDown(){
  languageCollection = null;
  logicCollection = null;
  tradeController = null;
  gameCon = null;
  viewController = null;
  fieldVisitor = null;
  drawController = null;
  fileReader = null;
}
```

Et eksempel af dette kan ses foroven, hvor @Before og @After er blevet benyttet i ”GameControllerTest” klassen. Her bliver de specifikke objekter som skal bruges i testene oprettes via @Before, setUp() og derefter sat til ‘null’ for at frigive dem igen, via @After, tearDown() metoden.

## 4.2 Testmetoder

### 4.2.1 Testmiljø

For kunne teste elementer af spillets gang automatisk med unit tests, har det været nødvendigt at opstille et testmiljø, hvor brugergrænsefladen ikke bliver oprettet. Det har vi opnået ved at lave en stub af vores ViewController, der ikke viser noget til brugeren og returnerer pålidelige værdier når den bliver spurgt om input. Metoderne som klasserne har brug for bliver defineret i et interface, og både ”ViewControlleren” og ”ViewControllerStub” implementerer det. Stubben bliver sat som ”ViewController” for de forskellige klasser når der skal testes.

## 4.3 Test cases

Der er opstillet tre formelle test cases som har til formål at dobbelttjekke de krav som har været vanskelige at teste. Det skyldes at variablernes tilstand afhænger af en lang række bruger valg gennem den grafiske brugergrænseflade. Derfor er der valgt at udføre manuelle tests som gennemgår et basis-scenarie for de udvalgte must-have krav.

| | |
|---|---|
| TestCase id | TC01 |
| Resume | Systemet sætter spillebræt, spillere og start kriterier op efter brugerens valg. |
| Krav | M1, M2, M3, M4, M8 |
| Prækondition | Nyt spil start. |
| Postkondition | Første spillers tur |
| Test procedure | 1. Spiller vælger sprog<br>2. Spiller vælger enten 3,4,5 eller 6. antal af spillerer.<br>3. Gentages for alle spillere<br>a. Spiller indtaster navn<br>4. Gentages for alle spillere<br>a. Spiller vælger ønsket farve og køretøj |
| Test Data | Sprog: Danish<br>Antal spillere: 3 |

| | |
|---|---|
| | Navne: Christian, Stig, Morten<br>Christian: Den cyan UFO<br>Stig: Den roede racerbil<br>Morten: Den orange traktor |
| Forventet resultat | Alle Spillere med valgte navne og køretøjer står klar ved startfeltet på brættet med valgt sprog og en balance på 1500 |
| Faktisk resultat | Christian, Den cyan UFO, 1500<br>Stig, Den roede racerbil, 1500<br>Morten, Den orange traktor, 1500 |
| Status | Bestået |
| Tested af | Nicolai Nisbeth |
| Dato | 18.01.2019 |
| Testmiljø | IntelliJ IDEA 2018.2.3 (Ultimate Edition)<br>Java 8<br>på Windows 10 standard edition |

| | |
|---|---|
| TestCase id | TC02 |
| Resume | Første spiller kaster terning, rykker antal øjne frem, lander på ejendoms felt og løser feltet. |
| Krav | M5, M6, M8, M9, M11 |
| Prækondition | TC01 |
| Postkondition | næste spillers tur |
| Test procedure | 1. Spiller kaster med terning ved sin turs start.<br>2. Systemet rykker antal øjne frem på brættet<br>3. Spiller lander på ejendomsfelt<br>4. Systemet checker om feltet er ledigt<br>5. Systemet spørger spilleren om denne vil købe feltet kun hvis feltet er ledigt.<br>6. spilleren vælger at købe feltet<br>7. Systemet tager feltet pris fra spillerens konto og sætter spilleren som ejer visuelt ved at ændre kanten áf feltet til spillerens farve |
| Test Data | Spilleren slår en 3'er og lander på Hvidovre.<br>Spilleren køber feltet og balancen ændres 1400. |

| | |
|---|---|
| Forventet resultat | spillers konto fratrækkes felt pris og feltet sættes til at ejes af spilleren |
| Faktisk resultat | spillers konto fratrækkes felt pris og feltet sættes til at ejes af spilleren |
| Status | Bestået |
| Tested af | Michael Jarberg |
| Dato | 18.01.2019 |
| Testmiljø | IntelliJ IDEA 2018.2.3 (Ultimate Edition)<br>Java 8<br>på Windows 10 standard edition |

| | |
|---|---|
| TestCase id | TC03 |
| Resume | En spiller lander på en anden spillers ejendom, kan ikke betale og er erklæret fallit. |
| Krav | M7, M10, M11, M12, M13, M14 |
| Prækondition | kun 2 Spillere tilbage hvor en er ved at gå fallit. |
| Postkondition | vinder er fundet, spillet er slut |
| Test procedure | 1. Spilleren kaster terninger<br>2. Spilleren lander på den anden spillers felt<br>3. Spilleren kan ikke betale lejen<br>4. Spilleren erklæres fallit<br>5. Systemet |
| Test Data | |
| Forventet resultat | den anden spiller erklæres vinderen af spillet. |
| Faktisk resultat | den anden spiller erklæres vinderen af spillet. |
| Status | Bestået |
| Tested af | Michael Jarberg |
| Dato | 18.01.2019 |

| | |
|---|---|
| Testmiljø | IntelliJ IDEA 2018.2.3 (Ultimate Edition)<br>Java 8<br>på Windows 10 standard edition |

Hermed er spillets basale funktionalitet testet, da spillerne kan igangsætte spillet, gennemføre en tur og løse feltet samt spille indtil en spiller går fallit og vinderen kåres.

## 4.4 Kode dækning

For at give en indikation, om hvorvidt det implementerede system er af ordentlig kvalitet, er der blevet udført kode dækning på alle pakkerne. Kode dækning angiver hvor mange linjer af kode der udføres, når JUnit testsene bliver benyttet. Da ovenstående tests blot er et udpluk vil den nedenstående kode dækning ikke belyse summen af disse tests dækningsgrad, men af alle unit tests vi har lavet. Dog har vi valgt ikke at lave test cases for trivielle metoder som getters og setters, og det vil have konsekvens for vores dækningsgrad, da alle statements i kildekoden tjekkes.

| Pakke | Klasse | Metode | Linje |
|---|---|---|---|
| Model | 100% (27/27) | 90% (118/130) | 95% (352/369) |
| Controller | 100% (6/6) | 62% (89/143) | 51% (587/1132) |
| Utilities | 100% (3/3) | 96% (31/32) | 98% (75/76) |

Det ses at både koden i model-, controller og utilities pakkerne har en acceptabelt dækningsgrad. Det har betydet at implementeringen af nye funktionaliteter ikke har resulteret i behovet for en masse manuelle tests og en bekræftelse på at metoderne fungerer efter hensigten.

## 4.5 Traceability Matrix

For at sikre at alle spillets must-have krav har en tilhørende test case opstilles en traceability matrix. Ud fra matricen aflæses det at must-have kravene mindst har bestået en test case og dermed implementeret i systemet.

| TC | M1 | M2 | M3 | M4 | M5 | M6 | M7 | M8 | M9 | M10 | M11 | M12 | M13 | M14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TC1 | x | x | x | x | | | | x | | | | | | |
| TC2 | | | | | x | x | | x | x | | x | | | |
| TC3 | | | | | | | x | | | x | | x | x | x |

## 4.6 Brugertest

Med fokus på spillets brugervenlighed, har vi valgt at lave en brugertest. De tre brugere der blev bedt om at teste spillet havde ingen kendskab til udviklingen af softwaren. Testen blev udført ved at testbrugerne fik en beskrivelse af en opgave de skulle løse. De løste opgaven mens de blev observeret. Den observerende var under testen opmærksom på at holde sig i baggrunden og ikke komme med råd til, hvordan opgaven skulle løses. Test-brugerne blev bedt om at tænke højt og fortælle deres tanker.

Opgaven de blev stillet var: "Åben spillet. Opret jer som spillere i spillet ved at vælge sprog, indtaste jeres navne og vælge spillebrik. Følg spillets instruktioner og spil på skift indtil en af jer vinder."

*Testen gav følgende input til forbedring af brugervenligheden via brugerspørgsmål:*

**Hvordan oplevede i brugervenligheden? Var programmet let eller svært at bruge og forstå?**

- Der var fra spillets start program instruktioner til at hjælpe brugeren undervejs i spillet.

**Hvordan oplevede i programmet visuelt? Var det opstillet på en overskuelig måde?**

- Ja, det var opstillet på en overskuelig måde. Men helt grafisk kan det godt gøres mere lækkert.
- Mangler noget farve i midten af spillet. ja overskueligt

**Oplevede i nogle mindre fejl i programmet? Og i så fald hvilke?**

- Spillet brød sammen da en spiller forsøgte at bytte en grund med en anden spiller. Vores navne i spillet var "1", "2" og "3".

**Hvordan oplevede i underholdningsniveauet? Var det sjov?**

- Ja, meget godt når man får en ejerskabsfølelse på spillet, når man selv vælger bil, navn, og selv kan vælge muligheder på sin tur.

**Yderligere konstruktiv kritik og forslag til ændringer?**

- I stedet for scroll down menuer kan der anvendes knapper.
- Eventuelt have en restriktion for valg af navne.

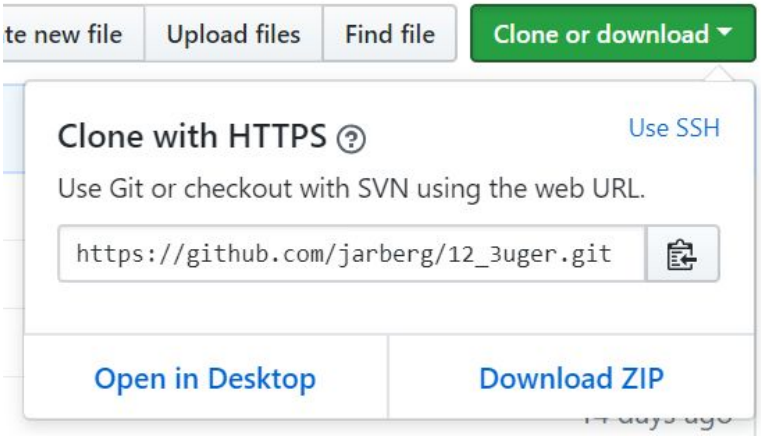
Brugernes feedback medførte en ændring til længden af en spillers navn. Det er ikke længere muligt at have et navn med færre en 3 bogstaver. Da en spiller med eksempelvis navnet "1", kan forveksles med felternes id i programmet.

# 5. Konfiguration

De udviklingsplatforme der er blevet brugt til at udforme vores program er MagicDraw, som der er blevet brugt til at vise hvordan koden hænger sammen, og IntelliJ, som er blevet brugt som IDE, Integrated Development environment, til vores program. Produktionsplatformene er både IOS systemer og Windows 10 operativsystemer. Derudover er der også blevet brugt 'matadorgui.jar' biblioteket, som er hentet fra maven. Der er intet ekstra som skal hentes, da alt ligger i selve programmet.

## 5.1 Versionsstyring[2]

Som standard, får kunden et link til siden hvor kildekoden er placeret i et git repository: https://github.com/jarberg/12_3uger . Dernæst for at kunne få fat i vores kode, så findes der to metoder for at få det hentet ned til intelliJ.

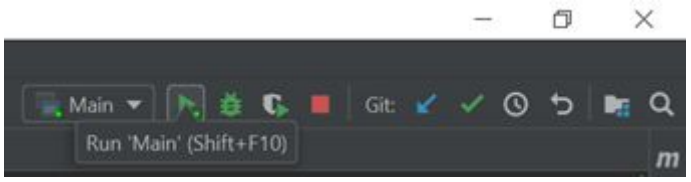



1) Man trykker på "Clone or download" knappen og downloader en zip-fil.
2) Åbner mappen hvor zip-filen er i.
3) Højreklikker på zip-filen og klikker på "Udpak alle…"
4) Dernæst dukker der et pop op vindue frem.
   a) Den nye fil skal gemmes i samme mappe, som selve installationen af IntelliJ, eller vil kildekoden ikke fungerer når man kører selve programmet.
5) Herefter åbner man IntelliJ, hvor man så trykker på import og finder-

[2] Afsnit taget fra gruppe 12's CDIO 3.

den nye udpakkede fil.

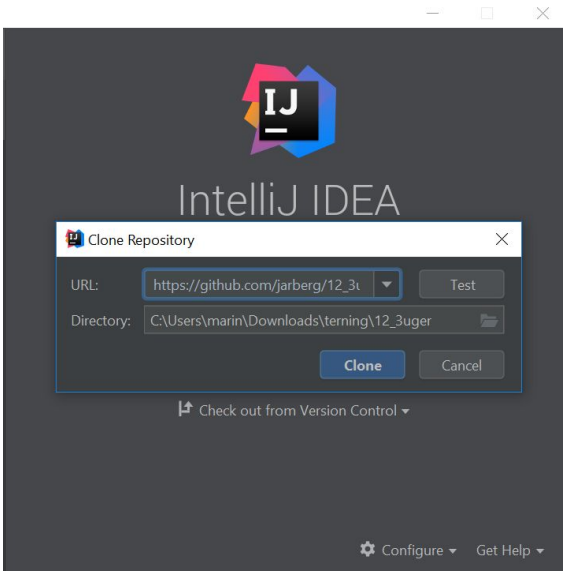

6) Til sidst for at kunne køre kildekoden, så skal man åbne Main klassen ude i venstre side, og klikke på den grønne kør knap oppe i højre siden af fanen.




Hvis der er problemer med den metode hvilket man kan se inden kildekoden køres (hvis der mangler en blå cirkel foran klasse navnet ude i venstre side kan den ikke køres) findes der heldigvis en anden metode for at få kildekoden ind i IntelliJ:

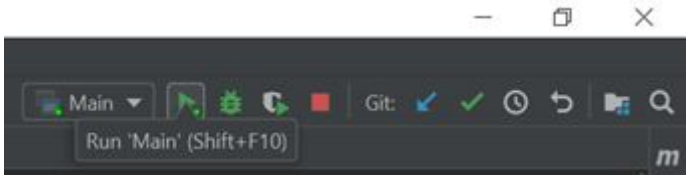
1) I stedet for at downloade en zip-fil, så kopierer man linket i stedet.
2) Herefter åbner man IntelliJ, og trykker på “Check out from Version Control” →”Git”


3) Hvor man så indsætter det kopieret link.



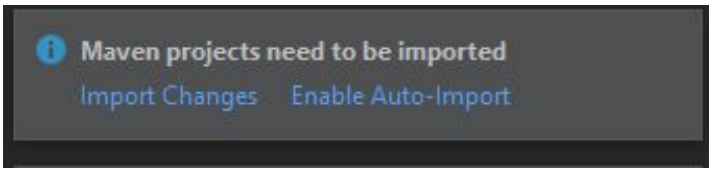

4) Herefter burde kildekoden komme frem og virke ligesom beskrevet i den anden metode.

5) Til sidst for at kunne køre kildekoden, så skal man åbne Main Klassen ude i venstre side, og klikke på den grønne kør knappen oppe i højre siden af fanen.



### *5.1.1 Maven*[3]

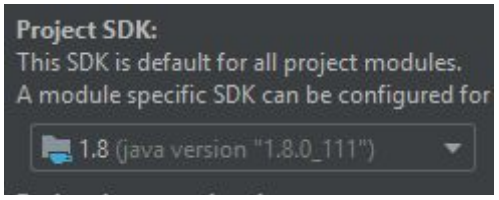
Når intellij og github er synced vil der poppe en besked som denne op i intellij




Da projektet anvender maven til at holde styr på projektets dependencies som GUI og Junit skal autoimport slås til ellers vil brugeren skulle importere alle fremtidige ændringer manuelt.

### *5.1.2 Java*[4]

projektet er bygget med Javas JDK version 8 som Maven er sat op til at compile og outpute fra.




[3] Taget fra gruppe 16's CDIO 3.
[4] Taget fra gruppe 16's CDIO 3.

# 6. Projektforløb

I dette projekt blev der udviklet et interaktivt Matador spil. Spillet er en forlængelse af de tidligere CDIO opgaver som er blevet stillet til IOOuterActive. I den forbindelse kan dette projekt ses som et iterativt forløb, da det er de samme grundlæggende discipliner der er blevet benyttet i dette projekt.

Projektet blev startet ved at få dannet et overblik over kravene fra IOOuterActive, så alle gruppens medlemmer kunne blive enige om, hvilke krav der som minimum skulle udføres ud fra kundens vision. Fra disse krav er der lavet use case diagrammer og en domænemodel. Disse krav, use cases, og domænemodel blev præsenteret til kunden og projektlederen ved første statusmøde. Mødet blev holdt for at sørge for at, vores projektgruppe og kunden var enig om den vision kunden havde til projektet.

Herfra blev der skabt et pakke diagram, som gjorde det muligt for gruppen at opdele projektet i forskellig ansvarsområder, i forhold til implementeringen. Under hele forløbet blev GRASP principperne benyttet. Der blev oprettet et repository på GitHub, så gruppens medlemmer kunne arbejde parallelt med hinanden. Herfra gik gruppen i fuld sving med klassediagrammer, sekvensdiagrammer og implementering.

Til anden statusmøde skulle der være lavet en tidsplan over hvor meget af projektet der var blevet fuldført, og hvor mange timers implementering der manglede for projektets afslutning på daværende tidspunkt. Desværre fik vi ikke lavet et tidstabel og vores præsentation var derfor ikke særligt vellykket. Det gjorde dog at gruppen konsekvent benyttede et tidstabel i den resterende del af projektforløbet, som gjorde at vi bedre kunne kommunikere hvilke mangler der var tilbage i projektet.

Projektgruppen mødtes hver dag i 3-ugers forløbet, hvilket gjorde det lettere at diskutere nøjagtigt hvordan hele gruppen opfattede kundens vision, og konstant videreudvikle på de iterative trin.

# 7. Konklusion

Projektets produkt burde leve op til kundens vision om en digital implementering af det klassiske Matador spil. Takket være en stor fælles indsats er alle de krav som er blevet opstillet, er blevet implementeret i programmet. Det har været til stor fordel at vi har arbejdet på lignende projekter før dette projekt, da vi har lært meget fra foregående projekter, som vi har kunne bruge og implementere i dette sidste Matador program. Dog på trods af en stor indsats kunne nogle af udviklings principperne have været fulgt bedre.

# 8. Bilag

## 8.1 Supplerende specifikation

**Ikke-funktionelle krav**

**IF1:** Systemet skal udvikles i Java.

**IF2:** Systemet skal udvikles i IntelliJ.

**IF3:** Systemet skal kunne spilles på Windows på databarene på DTU.

**IF4:** Systemet skal kunne spilles uden bemærkelsesværdige forsinkelser.

**IF5:** Systemet skal være let oversat.

**IF6:** Systemet skal være digitalt og dermed ignoreres proceduren for spillets fysiske opsætning samt restriktioner.

## 8.2 Use cases

| **Use case:** Make Transaction |
|---|
| **ID:** UC4 |
| **Kort beskrivelse:** En handel. |
| **Primær aktør:** Spiller |
| **Precondition:** Det er en spillers tur |
| **Main flow:**<br>1. Hvis nødvendigt include( Raise Money )<br>2. Systemet viser besked om handel.<br>3. Spiller modtager 200 kr. for at passere start. |
| **Postcondition:** Transaktionen er gennemført |
| **Alternative flows:**<br>3.a Spiller indgår i en bygnings handel<br>1. Systemet trækker værdien fra Spillers balance<br>2. Systemet tilføjer bygningen til grunden |

3.b Spiller har købt en grund.

1. Systemet trækker værdien fra Spillers balance
2. Systemet tilføjer grunden til Spiller

3.c Spiller har pantsat en grund.

1. Systemet tilføjer værdien til Spillers balance
2. Systemet pantsætter grunden

3.d Spiller har byttet en grund

1. Systemet skifter ejer på feltet

| **Use case:** Raise Money |
|---|
| **ID:** UC5 |
| **Kort beskrivelse:** Spiller anskaffer penge ved salg af ejendomme. |
| **Primær aktør:** Spiller |
| **Precondition:** Spiller deltager i en handel hvori Spiller ikke har nok penge. |
| **Main flow:**<br>1. Systemet giver spilleren salgsmuligheder<br>2. Spiller vælger at sælge ejendomme<br>3. Spiller vælger en ejendom at sælge<br>4. Systemet giver Spilleren penge for ejendommen ud fra dens værdi( Make Transaction )<br>5. Spiller har nok penge. |
| **Postcondition:** Spiller har nok penge |
| **Alternative flows:**<br>2.a Spilleren har ingen ejendomme<br>1. Spiller er gået fallit<br>2. Systemet sætter Spillers ejendomme på auktion<br><br>5a. Spiller har ikke nok penge<br>Gentag fra trin 1 |

| **Use case:** Visit Field |
| --- |
| **ID:** UC6 |
| **Kort beskrivelse:** Spiller lander på felt og feltets effekt udføres. |
| **Primær aktør:** Spiller |
| **Precondition:** Spiller er ved at lande på et felt |
| **Main flow:**<br>1. Spiller er landet på et PropertyField<br>2. Systemet informerer om feltet og spørger Spiller om de vil købe feltet<br>3. Spiller køber felt fra banken include( Make Transaction ) |
| **Postcondition:** Spilleren har fri tur og mulighed for at give turen videre |
| **Alternative flows:**<br>1a Spiller lander på startfelt<br>1. Spiller bliver informeret om feltet<br><br>1b. Spiller lander på et ChanceField<br>1. Spiller bliver informeret om feltet<br>2. Include( Draw Card )<br><br>1c. Spiller lander på et JailField<br>1. Spiller bliver informeret om at de bare er på besøg<br><br>1d. Spiller lander på et GoToJailField<br>1. Spiller bliver informeret om feltet<br>2. Systemet sætter Spiller i fængsel<br><br>1e. Spiller lander på et ParkingField<br>1. Spiller bliver informeret om feltet.<br><br>1f. Spiller lander på et TaxField<br>1. Systemet informerer spiller om feltet |

2. Spiller vælger betalingsform include( Make Transaction )

2b. Spiller ejer selv feltet

1. Systemet informerer Spiller om feltet

2c. Felt er ejet af anden spiller

1. Include( Make Transaction )

| **Use case:** Draw Card |
|---|
| **ID:** UC7 |
| **Kort beskrivelse:** Spiller trækker et chancekort og dets effekt udføres.. |
| **Primær aktør:** Spiller |
| **Precondition:** Spiller har landet på et "Prøv Lykken" felt. |
| **Main flow:**<br>1. Spiller har trukket et MoneyCard.<br>2. Systemet informerer Spiller om kortet.<br>3. Include( Make Transaction ) |
| **Postcondition:** Spilleren har fri tur og mulighed for at give turen videre |
| **Alternative flows:**<br>1a. Spiller har trukket et GetOutOfJailCard<br>1. Systemet informerer spiller om kortet<br>2. Spiller får tildelt kortet<br><br>1b. Spiller har trukket et MonopolyJackpotCard<br>1. Systemet undersøger om Spillers er kvalificeret til legatet.<br>2. Spiller er kvalificeret og modtager legatet include( Make Transaction )<br>2b. Spiller er ikke kvalificeret<br><br>1c. Spiller har trukket et MoveCard<br>1. Systemet informerer Spiller om kortet<br>2. Systemet rykker Spiller.<br>3. Spiller modtager penge hvis Spiller har passeret start |

3a. Spiller modtager ikke penge hvis Spiller har passeret start

1d. Spiller har trukket et PayForBuildingsCard
1. Systemet informerer Spiller om kortet
2. Spiller betaler for hvert hus og hotel Spiller ejer include( Make Transaction )

1e. Spiller har trukket et TeleportAndPayDoubleCard
1. Systemet informerer Spiller om kortet.
2. Spiller rykker frem til nærmeste dampskibsselskab
3. Spiller betaler dobbelt leje til ejeren include( Make Transaction )
3a. Spiller ejer selv feltet og betaler intet
3b. Spiller får mulighed for at købe feltet fordi ingen ejer det include( Make Transaction )

1f. Spiller har trukket et TeleportCard
1. Systemet informerer Spiller om kortet
2. Spiller bliver sat direkte i fængsel.

1g. Birthday Card
1. Systemet informerer Spiller om kortet
2. Spiller modtager penge fra de andre spillere include( Make Transaction )

## 8.3 Timeregnskab

| Jens Daniel Kramhøft | Analyse | Design | Implementation | Dokumentation |
|---|---|---|---|---|
| 7/1/2019 | 4½ | 2 | | 1 |
| 8/1/2019 | 5 | 1 | | 1 |
| 9/1/2019 | | 2 | 6 | |
| 10/1/2019 | | 3 | 4 | |
| 11/1/2019 | | 2 | 4 | |
| 13/1/2019 | | | 1 | |
| 14/1/2019 | | 3 | 3 | 1 |
| 15/1/2019 | | 1 | 5 | |
| 16/1/2019 | | | 12 | |

| | | | | |
|---|---|---|---|---|
| 17/1/2019 | | | 12 | |
| 18/1/2019 | | | 11 | |
| 19/1/2019 | | | 2 | 2 |
| 20/1/2019 | | | 4 | 9 |

| Michael Lund Jarberg | Analyse | Design | Implementation | Dokumentation |
|---|---|---|---|---|
| 7/1/2019 | 5 | 2 | | 2 |
| 8/1/2019 | 4 | 1 | | 2 |
| 9/1/2019 | | 2 | 5-6 | 1 |
| 10/1/2019 | | | 7 | |
| 11/1/2019 | | | 7 | |
| 12/1/2019 | | | 2 | |
| 13/1/2019 | | | 1 | |
| 14/1/2019 | | | 7 | |
| 15/1/2019 | | | 7 | |
| 16/1/2019 | | | 10 | |
| 17/1/2019 | | | 11 | |
| 18/1/2019 | | | 11 | |
| 19/1/2019 | | | | |
| 20/1/2019 | | | 2 | 11 |

| Marina Schmidt Malling | Analyse | Design | Implementation | Dokumentation |
|---|---|---|---|---|
| 7/1/2019 | 4½ | 2 | | 1 |
| 8/1/2019 | 4 | 1 | | 2 |
| 9/1/2019 | | 2 | 5 | |
| 10/1/2019 | | 1 | 6 | |
| 11/1/2019 | 2 | | 4 | 1 |

| | | | | |
|---|---|---|---|---|
| 12/1/2019 | | | 2 | |
| 13/1/2019 | | | | |
| 14/1/2019 | | | 6½ | |
| 15/1/2019 | | | 5 | 1 |
| 16/1/2019 | | | 9 | |
| 17/1/2019 | | | 10 | |
| 18/1/2019 | | | 10 | |
| 19/1/2019 | | | 5 | 1 |
| 20/1/2019 | | | | 11 |

| Nicolai Nisbeth | Analyse | Design | Implementation | Dokumentation |
|---|---|---|---|---|
| 7/1/2019 | 5 | 2 | | 2 |
| 8/1/2019 | 4 | 1 | | 2 |
| 9/1/2019 | | 2 | 5 | |
| 10/1/2019 | | 3 | 5 | |
| 11/1/2019 | 2 | 1 | 2 | 1 |
| 12/1/2019 | | | 5 | |
| 13/1/2019 | | | 2 | |
| 14/1/2019 | | | | 2 |
| 15/1/2019 | | | 5 | |
| 16/1/2019 | | | 12 | |
| 17/1/2019 | | | 12 | |
| 18/1/2019 | | | 12 | |
| 19/1/2019 | | | 8 | |
| 20/1/2019 | | 4 | | 9 |

| Tanja Sølvsten | Analyse | Design | Implementation | Dokumentation |
|---|---|---|---|---|
| 7/1/2019 | 4½ | 2 | | 1 |

| | | | | |
|---|---|---|---|---|
| 8/1/2019 | 4 | 1 | | 2 |
| 9/1/2019 | | 2 | 5 | |
| 10/1/2019 | | | 6 | |
| 11/1/2019 | 2 | | 4 | 1 |
| 12/1/2019 | | | | |
| 13/1/2019 | | | | |
| 14/1/2019 | | 2 | 4 | |
| 15/1/2019 | | | 6 | 1 |
| 16/1/2019 | | | 8 | |
| 17/1/2019 | | | 7 | 1 |
| 18/1/2019 | | | 9 | |
| 19/1/2019 | | | | 1 |
| 20/1/2019 | | | ½ | 10 |

| Rasmus Traub Nielsen | Analyse | Design | Implementation | Dokumentation |
|---|---|---|---|---|
| 7/1/2019 | 4½ | 2 | | 1 |
| 8/1/2019 | 4 | 1 | | 2 |
| 9/1/2019 | | 2 | 5 | |
| 10/1/2019 | 1 | 1 | 6 | |
| 11/1/2019 | | | 4 | |
| 12/1/2019 | | | | |
| 13/1/2019 | | | | |
| 14/1/2019 | | | 5 | |
| 15/1/2019 | | | 3 | |
| 16/1/2019 | | | 8 | |
| 17/1/2019 | | | 3 | |
| 18/1/2019 | | | 10 | |

| | | | | |
|---|---|---|---|---|
| 19/1/2019 | | | 4 | 2 |
| 20/1/2019 | | | 1 | 2 |