Final MonopolySpil

Danmarks Tekniske Universitet

Gruppe 11

Bashar Bdewi

(s183356)

Derar Almosawe

(s183356)

Sulaiman Kasas

(s195462)

Marah Marak

(s182964)

Link til GitHub-repo:

https://github.com/Dalmos87/11_final.git

Rapport i kurset Indledende programmering 02312

Danmarks Tekniske Universitet

Afleveringsfrist: 20. Januar 2020

Indhold

1. Timeregnskab

	Bashar	Derar	Sulaiman	Marah
06-01	4	4	4	2
07-01	5	5	5	5
08/01	4	3	4	4
09/01	4	4	4	4
10/01	4	5	5	3.5
11/01	3	3	2	2
13/01	2	1.5	1.5	1.5
14/01	6.5	6.5	6	4
15/01	1.5	3	3	3
16/01	3.5	3	3	2.5
18/01	4	4	4	4
19/01	4	4.5	4.5	4

2. Abstract

This report describes the work done on the CDIO project part 4 by group 11, which is a software project written in the Java language. The project is a" Monopoly-game played by three to six players. First, the requirements of the product are described, mainly through a use case analysis of the final product. By analyzing use cases like" Start game", and" landed on property "as well as the product description of the real "Monopoly-game", the group has described the requirements of the system and divided them in two categories: nonfunctional and functional requirements. The requirements were reconsidered throughout the whole project. After this comes the analysis-section, describing the domain-model of the system, mainly through domain-classdiagram. This diagram shows the central classes of a Monopoly-game such as" Monopolyspil"," Spiller", "Spilleplade"and "Felt". Then comes the Design-section describing the process of planning how to make the software classes and methods based on the domain model. This results in a very detalied classdiagram of the software classes. The last part of the report contains notes on the

Project management as well as the tests that were made to check the liability system. There were made several tests, and these were divided into two categories: Automated and manual tests. In the end it is concluded, that the project does not meet up to all the important requirements.

3. Indledning

Vi er blevet stillet til opgave at udarbejde et Matador-program i Java. Opgaven kommer som en naturlig forlængelse af vores tidligere CDIO-opgaver, hvor vi har udviklet forskellige stykker software, som kan bruges i forbindelse med udviklingen af dette program, f.eks. de virtuelle terninger og spiller-klassen. Derudover anses dette projekt for at være den afsluttende projekt på vores første semester inden gruppen i januar skal op til eksamen. Formålet med spillet er at købe, udleje eller sælge ejendomme så fordelagtigt, at man bliver den rigeste spiller og dermed spillets eneste matador. Spillet har nogle overordnet meget tydelige og entydige regler, som på visse punkter gør det ideelt projekt til at lære at programmere. Til gengæld er det også et temmelig omfattende spil med f.eks. det store antal chancekort og felter. Noget af det sværeste ved projektet har derfor vist sig at være, hvordan systemets struktur skulle planlægges og opbygges på en hensigtsmæssig måde, så den både er forståelig, overskuelig, velfungerende og evt. kan genbruges.

4. Krav

Det er en del af systemudviklerens opgave at finde ud af, hvad brugerne ønsker at det nye system kan. Krav er en specifikation af hvad der skal implementeres. En kravspecifikation opdeles efter funktionelle og ikke funktionelle krav.

MoSCoW (Must have, Should have, Could have og Want to have), bruges til at lave en liste over funktionelle krav.

FURPS+ (Funktionelle, Usability, Reliability, Performance, Supportability osv.) bruges til at lave en list over ikke funktionelle krav.

4.1 Ikke funktionelle krav

ID	Beskrivelse
KIF1	Systemet skal have en hurtig responstid.
KIF2	Systemet skal være let for brugeren at bruge.
KIF3	Programmet skal være nemt at teste.
KIF4	Programmet skal være nemt at vedligeholde.

KIF5	Programmet skal ikke kræve høj computer performance.
KIF6	Programmet skal nemt kunne oversættes til et andet
KIF7	Programmet skal kunne genbruges.
KIF8	Koden skal kunne genbruges

4.2 Funktionelle Krav

ID	Beskrivelse
KF1	Et spil mellem 3-6 spiller efter behov.
KF2	Man kaster med to sekssidede terning og rykker sin bil det antal øjne man slår rundt på brættet i urets retning.
KF3	Alle spiller starter med 30000 kr.
KF4	Spillet skal havde en spilleplade med 40 forskellige felter
KF5	Alle spillere starter på startfeltet.
KF6	Hver gang man passere start får du 4000kr.
KF7	Land på et felt. Hvis feltet er ledigt, systemet beder om du vil købe det • Hvis ja, Betal banken, det beløb der står på feltet. Banken retunerer feltets skød. Placer dit navn på feltet, så man kan se man ejer det. • Hvis nej, sætter baken feltets skød på auktion og de andre spiller kan byde på det. Højste byder modtager skødet. Hvis feltet er ejet af en anden spiller, betales den husleje der står på skødet til ejeren. Hvis feltet er ejet af spilleren, sker der ingenting. Hvis en anden spiller ejer alle ejendomme i samme farve, skal man betale det dobbelte i huslejen. Hvis en spiller lander på et felt med bygninger eller hotel, der ejet af en anden spiller skal spilleren betale det beløb der står på skødet. Skylder en spiller mere, end han ejer, skal han overdrage alt til sin kreditor efter at have solgt eventuelle bygninger tilbage til banken, og han må derefter udgå af spillet. Er det banken, der er kreditor, sælger bankøren straks modtagne grunde på auktion. Hvis man lander på et felt med " prøv lykken kort" på, skal man trække et chancekort og gør hvad der står på det. Der er 32 prøv lykken kort. • Hvis man trækker kortet, "De fængsles" ophæves reglen om at man modtager 4000 kr. for at passerer start, i starten af de næste tur, skal man betale 1000 kr. eller bruge "Du løslades uden omkostninger" - kortet, der kan trækkes og gemmes fra prøv lykken kortene. Kast derefter terningen og ryk frem som normalt. Man kan godt modtage husleje mens du er i fængsel. • Gratis parking feltet, man skal ikke betale for at stå der.

KF8	Spillet slutter ved at der er kun en spiller tilbage.

5. Analyse

5.1 Use cases

ID	Beskrivelse
UC1	**Start spil** Det valgte antal spillere indtaster deres navne, vælger en farve til deres bil og derefter får de udleveret 30.000 hver.
UC2	**Land på et felt** Spilleren rykker deres bil frem til et felt baseret på et terningekast og baseret på tidligere placering. 1.1 Hvis feltet er ledigt, systemet beder om du vil købe det 1.1.1 Hvis ja, Betal til banken, det beløb der står på feltet. Banken retunerer feltets skød. Placer dit navn på feltet, så man kan se man ejer det. Pengene bliver trukket fra spillerens pung af systemet. 1.1.2 Hvis nej, sætter baken feltets skød på auktion og de andre spiller kan byde på det. Højste byder modtager skødet. 1.2 Hvis feltet er ejet af spilleren, sker der ingenting. 1.3 feltet er ejet af en anden spiller 1.3.1 Hvis en anden spiller ejer ejendomen, betales den husleje der står på skødet til ejeren. 1.3.2 Hvis en anden spiller ejer alle ejendomme i samme farve, skal man betale det dobbelte i huslejen. 1.3.3 Hvis en spiller lander på et felt med bygninger eller hotel, der ejet af en anden spiller skal spilleren betale det beløb der står på skødet. 1.4 Hvis man lander på et felt med " prøv lykken kort" på, skal man trække et prøv lykken kort og gør hvad der står på det. Der er 32 prøv lykken kort. 1.4.1 Hvis man trækker kortet, "De fængsles" ophæves reglen om at man modtager 4000 kr. for at passerer start, i starten af de næste tur, skal man betale 1000 kr. eller

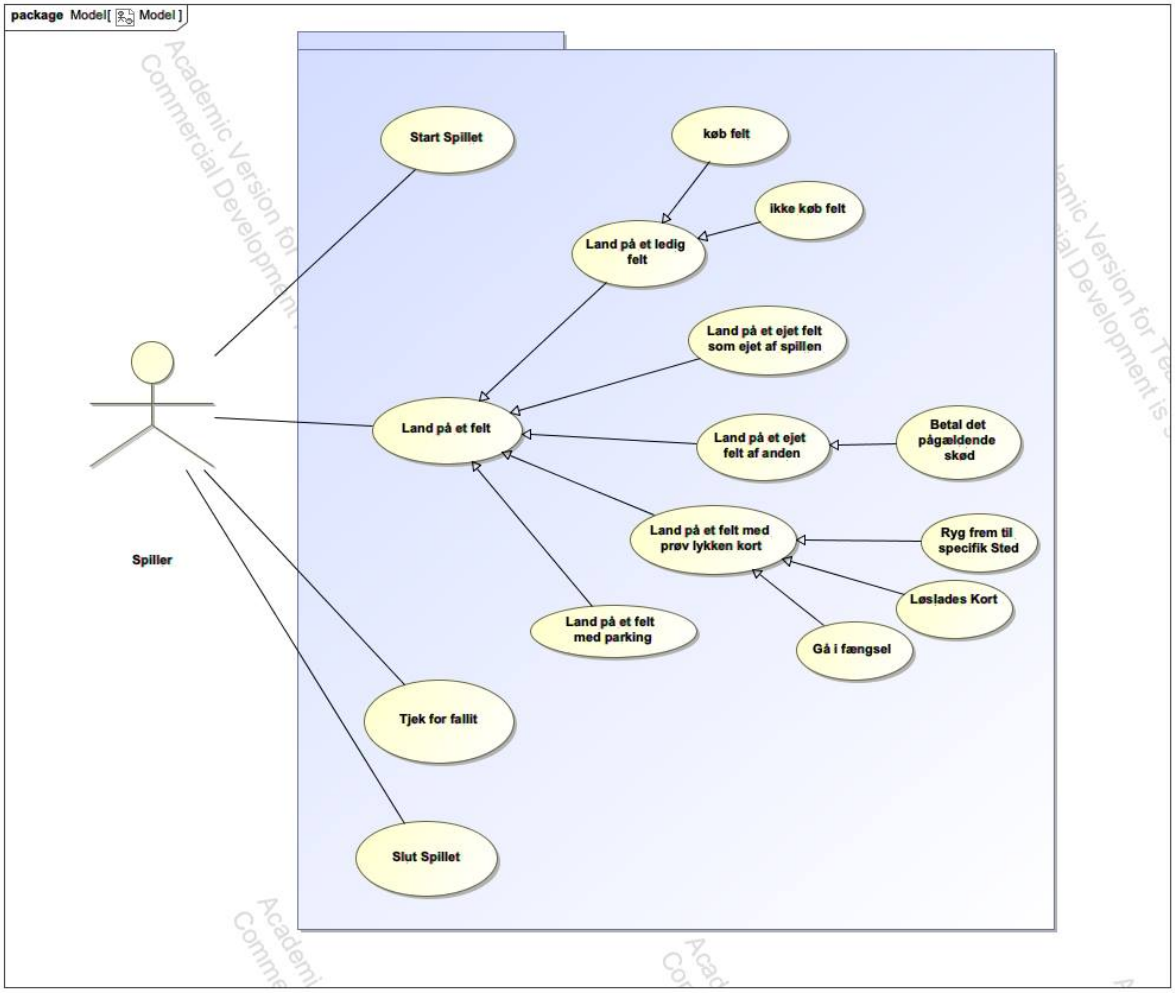
	bruge "Du løslades uden omkostninger" - kortet, der kan trækkes og gemmes fra prøv lykken kortene. Kast derefter terningen og ryk frem som normalt. Man kan godt modtage husleje mens du er i fængsel. 7.7 Gratis parking feltet, man skal ikke betale for at stå der.
UC3	**Tjek for fallit** 1. Der tjekkes, om spilleren skal betale mere end spilleren ejer. 2. Hvis spilleren har nok penge til at betale, gør han det 3. Hvis ikke spilleren har nok penge til at betale, betaler han det sidste han har, og han må derefter udgå af spillet. 4. Resten af spillerne fortsætter med at spille
UC4	**Slut spillet** Spillet slutter ved at der er kun en spiller tilbage. Ellers spillet sluttes manuelt.

5.2 Fully dressed use case

Fully Dressed version af UC2, Land på et felt

Use case Section	Comment
Use case navn	Land på et felt
Primær aktør	Player Spillerne spiller spillet
Pre-conditions	Spilleren rykker sin bil frem til et felt baseret på et terningekast og baseret på tidligere placering
Succeskriterier/ postconditions	Spilleren har enten købt et felts skød, betalt leje, eller er skødet blevet sat på auktion og muligvis blevet solgt til en anden spiller, trukket et prøv lykken kort, eller er spilleren sat i fængsel
Main flow	1. Land på et ledigt felt • Systemet spørger om spilleren vil købe feltets skød. Hvis ja, så betaler spilleren prisen for skødet og bliveren ejeren af skødets felt. Hvis nej, sættes skødet på auktion så de andre spillere kan byde på

	det. Skødet går til den højeste byder. Hvis ingen byder på det, så kan feltet først blive købt eller sat på auktion indtil en spiller lander på feltet igen. 2. Land på et felt ejet af dig selv • Hvis en spiller lander på et felt som er ejet af personen selv, sker der ingenting 3. Land på et felt ejet af en anden spiller • Hvis en anden spiller ejer ejendommen, betales den husleje der står på feltets skød til ejeren • Hvis en anden spiller ejer alle ejendomme i samme farve, skal man betale det dobbelte af huslejen. • Hvis en spiller lander på et felt med bygninger eller hotel der er ejet af en anden spiller, skal spilleren betale det beløb der står på feltets skød. 4. Land på et felt med prøv lykken kort • Hvis man lander på et felt med prøv lykken kort på, skal man trække et prøv lykken kort og gøre hvad der står på det. Der er 32 prøv lykken kort • Hvis man trækker kortet "De fængsles" ophæves reglen om at man modtager 4000kr for at passerer start, i starten af spillerens næste tur, skal man betale 1000kr. eller bruge "du løslades uden omkostninger" kortet der kan trækkes og gemmes fra prøv lykken kort bunken. Kast derefter terningen og ryk som normalt. Man kan godt modtage husleje mens man er i fængsel • Ryk frem til et specifikt felt • Du løslades uden omkostninger kortet bruges til at komme ud af fængsel. 5. Land på et felt med gratis parkering • Lander en spiller på gratis parkering feltet er man i helle og der sker ingenting 6. Land på betal indkomstskatfeltet • Lander en spiller på indkomstskatfeltet hæves der automatisk 2000kr fra spillerens pung 7. Land på ekstraordinær statsskatfeltet • Lander en spiller på ekstraordinær statsskatfeltet hæves der automatisk 1000kr fra spillerens pung



Figur 1: Use case Diagram

5.3 Aktører

● Spiller

- Ønsker at spille spillet

Styrer alt input som start af spillet, valg af farver, valg af antal af spillere, at kaste terningen

5.4 Navneord Analyse

Udfra vores usecases beskrivelser kan vi analysere os frem til når nøgleord vi skal bruge fremadrettet i rapport. De forskellige navneord er understreget i sektionen usecases. De forskellige navneord er følgende:

- Bil
- Terningkast
- Prøv lykken kort
- ejendom
- felt
- konto
- bank
- De fængsel
- parkering
- skød
- Gratis parking
- Start
- Indkomstskat
- StatsSkat
- PåBesøg

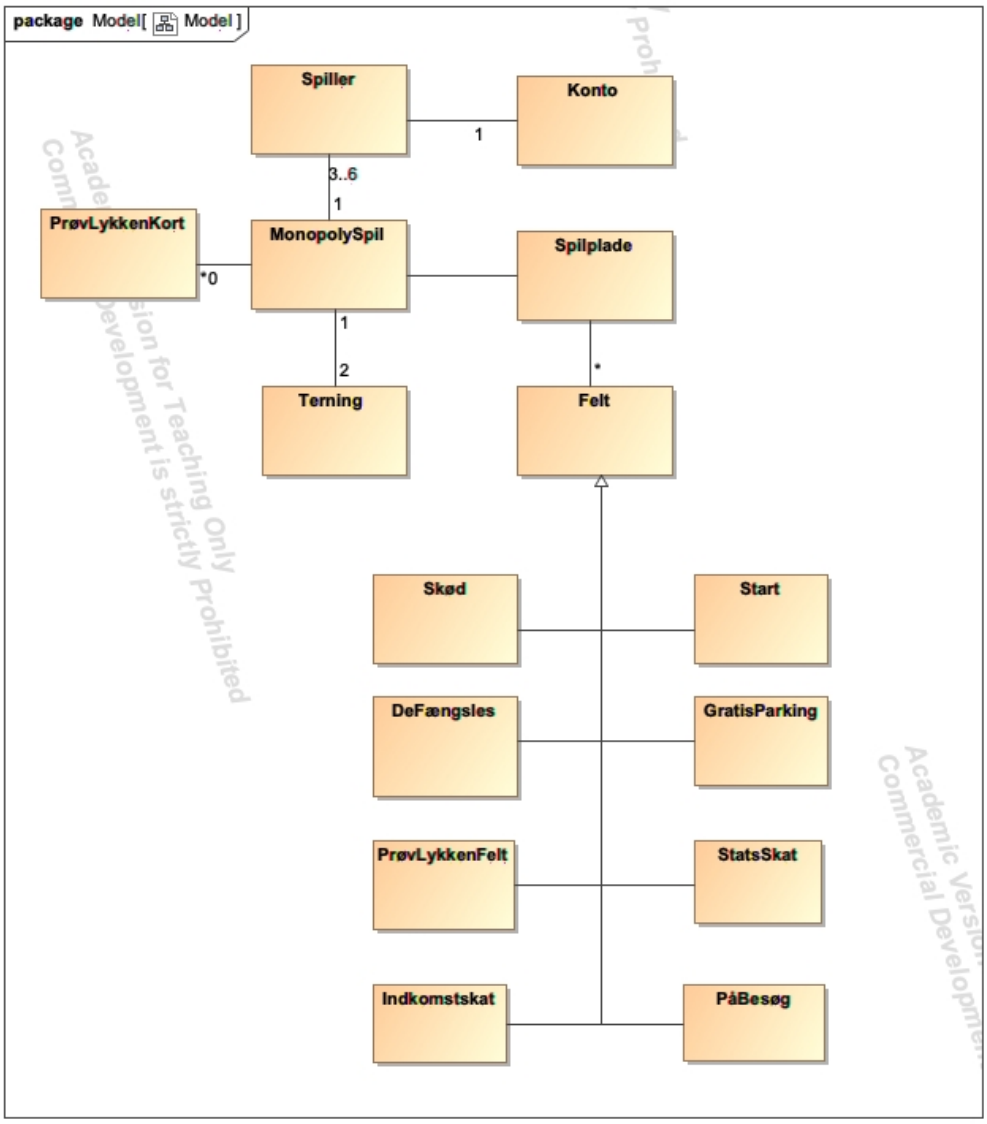
5.5 Nødvendige klasser

Ud Fra analyse af vores use cases har vi konkluderet at vi skal bruge følgende klasser

- Main
- MonopolySpil
- Terning
- PrøvLykkenKort
- Spilplade
- Felt
  - Start
  - GratisParking
  - PrøvLykkenFelt
  - Skød
  - DeFængsles
  - StatsSkat
  - Indkomstskat
  - PåBesøg

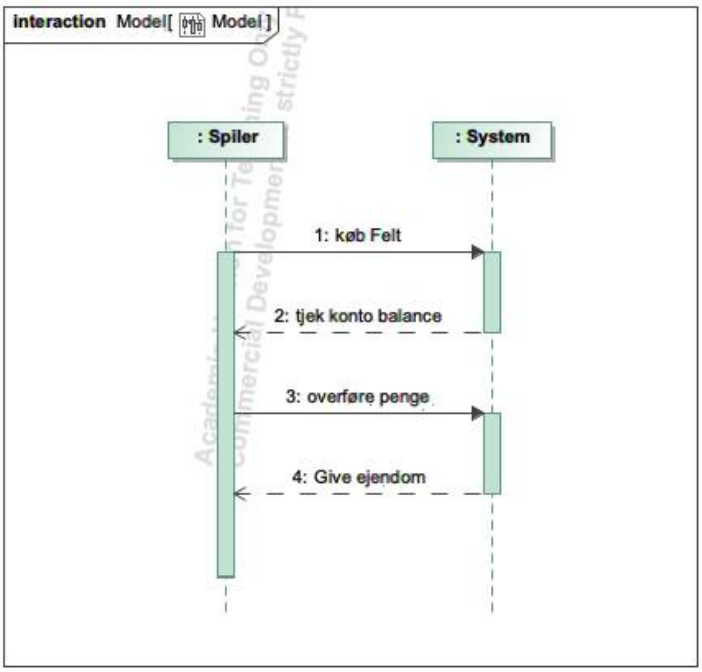
- Pung
- Spiller

5.6 Domæne klassediagram



Figur 2: Domæne Klassediagram

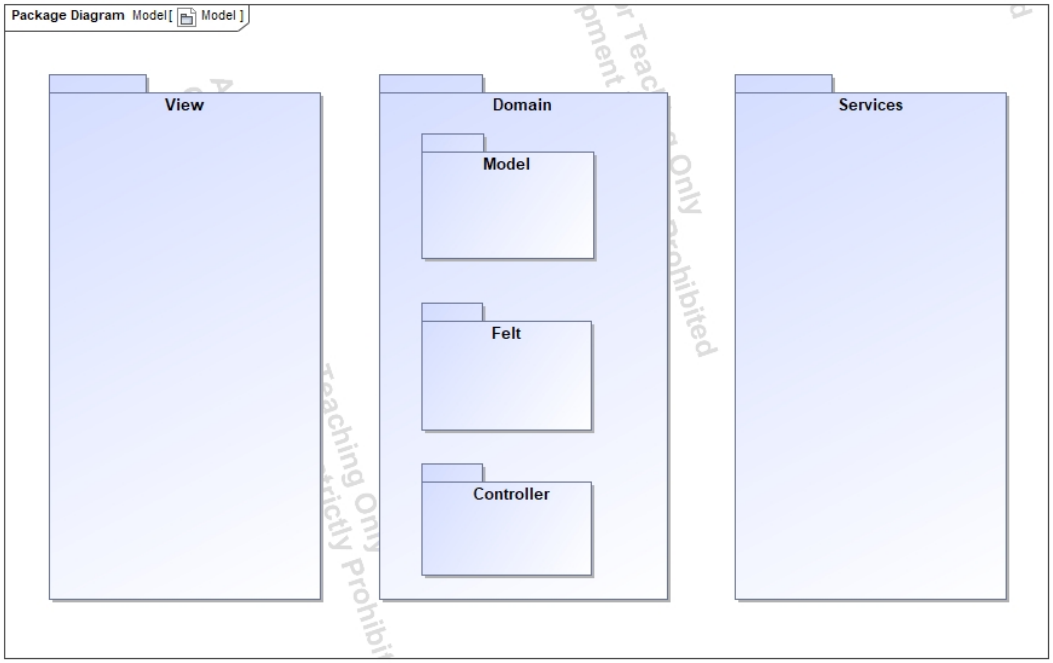
5.7 Systemsekvensdiagram



Figur 3: SystemsekvensDiagram

SystemsekvensDiagram over forløbet, når en spiller skal købe en ejendom

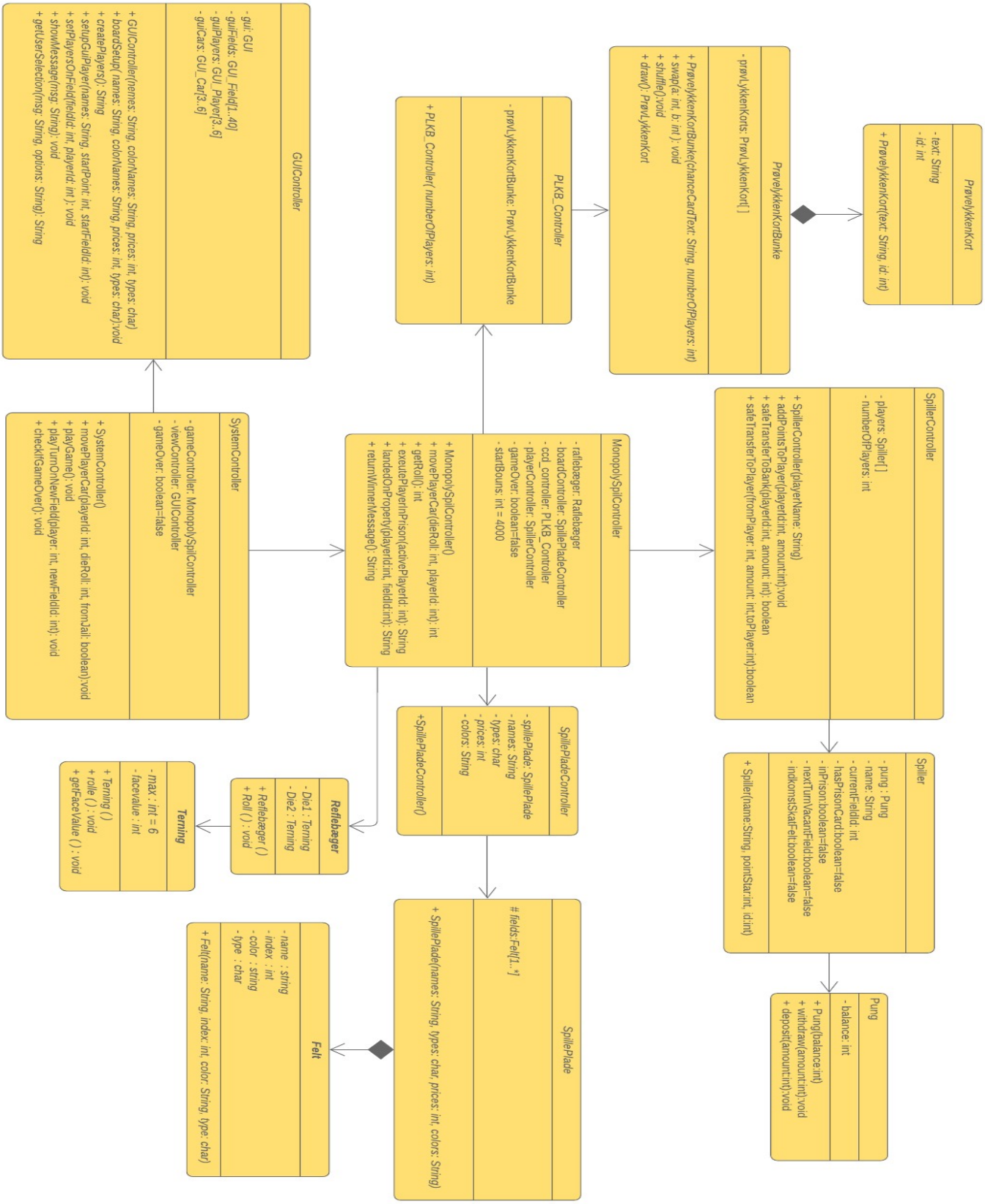
5.8 Pakke Diagram



Figur 4: PakkeDiagram

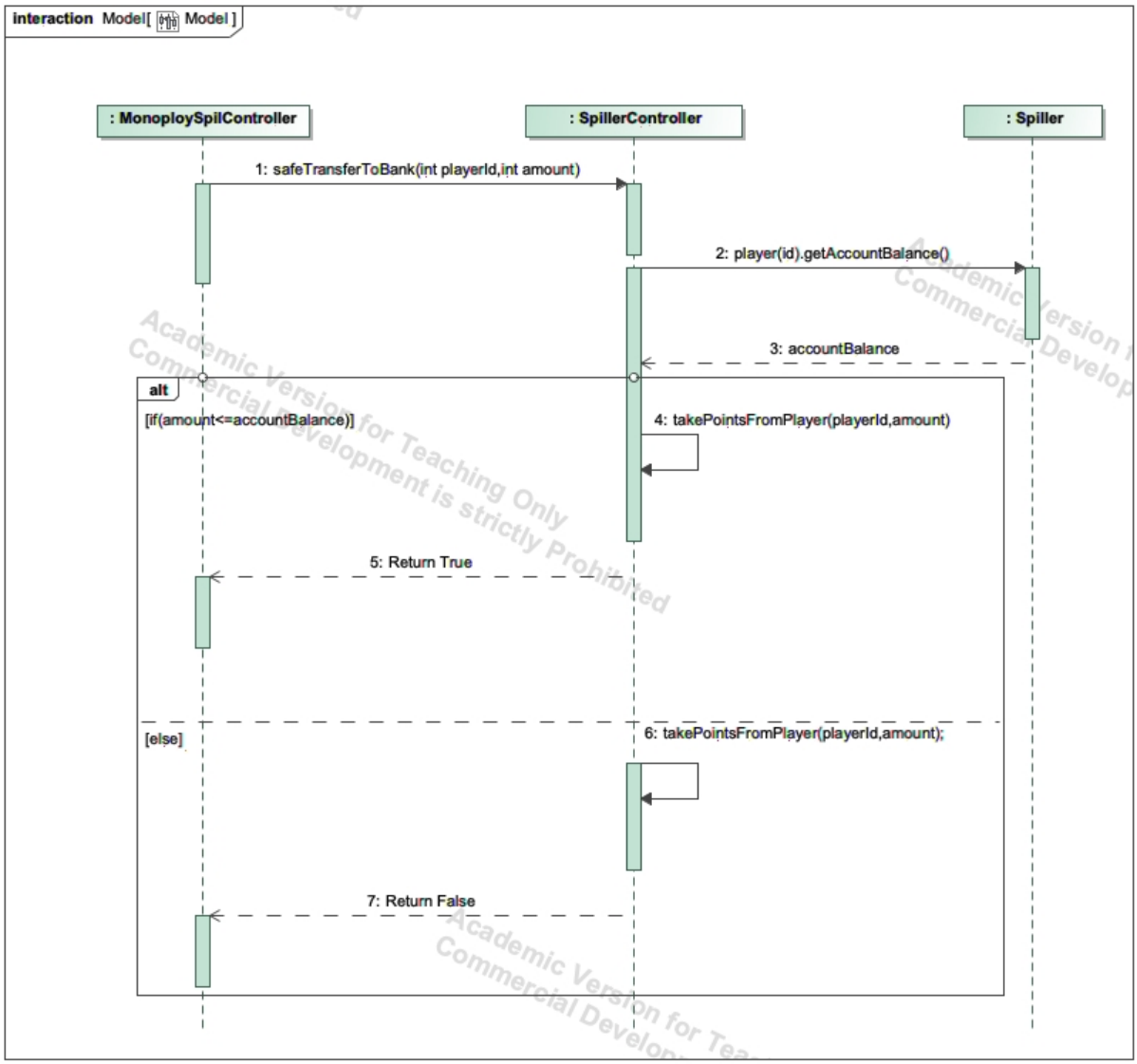
6. Design

6.1 Klassediagram



Figur 5: DesignKlassediagram

6.2 Sekvensdiagram

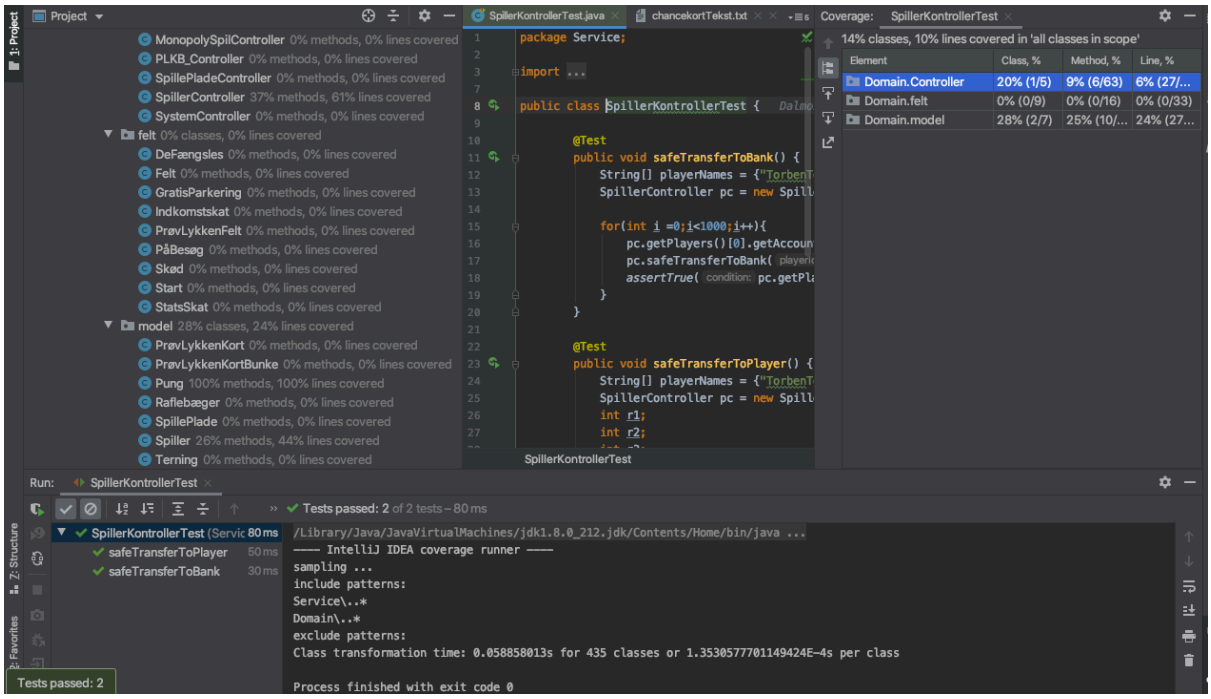


Figur 6: Sekvensdiagram over metoden safeTransferToBank()

7. Test

7.1 Automatiserede tests

Vi har lavet 2 automatiserede JUnit-tests, som tester 2 centrale metoder i SpilleKontroller-klassen, hhv. safeTransferToBank() og safeTransferToPlayer. Disse er opsat på en måde, så de kan køres fra klassen SpillerKontrollerTest i test-pakken og programmet, og vil udprinte i konsollen, om de er bestående. Metoden safeTransferToBank() tager 2 int-argumenter. Den ene er et id på den spiller, der skal lave overførslen, og den anden er beløbet, der skal overføres. Metoden skal gerne sikre, at spillerens balance aldrig bliver negativ, men at spilleren blot betaler sine sidste penge, hvis spilleren går fallit. Testen af safeTransferToBank() tester, at lige meget hvilken (ikke-negativ) balance en spiller har, vil balancen stadig være ikke-negativ efter at safeTransferToBank() er blevet kaldt på den spiller med et vilkårligt positivt eller negativt beløb.



Figur 7: Dokumentation af code coverage i vores JUnit test.

Metoden safeTransferToBank() tager 3 int-argumenter. 2 af dem er id på spillerne, overførslen skal ske imellem, og den sidste er beløbet, der skal overføres. Metoden skal gerne sikre, at spillerens balance ikke bliver negativ, men at spilleren blot betaler sine sidste penge, hvis spilleren går fallit. Testen af safeTransferToPlayer() tester, at lige meget hvilke (ikke-negative) balancer de to spillere har, vil deres balancer stadig være ikke-negative efter at safeTransferToBank() er blevet kaldt mellem dem med et vilkårligt positivt beløb. Det vil aldrig være relevant at kalde den metode med et negativt beløb, da man i så fald kunne kalde den ”den modsatte vej altså. Vi har af flere omgange gennemført disse tests og altid bestået dem, som de er i deres nuværende form. I figur 7 ses et screenshot fra Intellij efter at de to tests er gennemført med code-coverage. På billedet ses code coverage for de to tests, dvs. det vises hvor mange procent af systemets klasser og linjer, der har været aktiveret under eksekveringen i systemets forskellige pakker. Vi testede overførselsfunktioner i SpillerKontroller-klassen. Derfor giver det god mening, at kun har været i SpillerKontroller, Pung og Spiller at der er blevet eksekveret kode, som det også fremgår af billedet. Ideelt skulle vi gerne have unit-tests til at teste størstedelen af programmet igennem. Kravet til dette projekt var dog blot at lave en enkelt unit-test. Havde der været mere tid, havde vi lavet nogle flere og mere gennemgående test.

7.2 Manuelle tests

Vi har også udarbejdet 3 testcases med tilhørende testrapporter. Disse kan ses i figur 8 og 9. Disse er tests af funktionaliteten af større dele af programmet, end det er muligt at teste med unit-tests. Unittests virker nemlig rigtig dårligt til at teste en brugergrænseflade, da opførslen af en brugergrænseflade er svær at teste automatiseret. Vores testcases indeholder derfor ”instruktioner” som en bruger kan læse, så brugeren ved hvordan testen skal udføres. Brugeren kan så verificere eller falsificeres testen, og de kan reproduceres af forskellige brugere.

7.3 Brugertests

For at teste brugervenligheden og funktionaliteten af vores program, har vi også lavet en brugertest. I brugertesten har vi fået en person, som ikke har kendskab til programmet til at forsøge at spille spillet. Brugeren blev bedt om at ”tænke højt ”mens hun spillede, og en udvikler (Sulaiman) tog noter

Test case ID	TC1
Beskrivelse	Test navneinputtet i spillets opsætning
Preconditions	Spillet er startet og har budt velkommen
Postconditions	Systemet kender spillernes navne og den rækkefølge, deres ture skal afvikles

Procedure	Det valgte antal spillere indtaster deres navne, vælger en farve til deres bil og derefter får de udleveret 30.000 hver.
Forventet resultat	Turene blev afviklet korrekt
Egentligt resultat	Turene blev afviklet korrekt
Status	Er bestået
Testet at	Derar
Dato	19.01.2020
Test enviroment	Intellij IDEA 2019.3.1(Ultimate Edition) Build: 193.5662.53 built on 18 December 2019 On Windows 10 Home Edition

Figur 8: Test case 1: Test navneinputtet i spillets opsætning

Test case ID	TC2
Beskrivelse	Test af ejeren af en ejendom vises på brættet
Preconditions	Spillet er initialiseret med 3-6 spillere
Postconditions	Spillet er i gang
Procedure	1. Start spillet op og angiv spillernes navne 2. Tryk på spillets ejendomme, dvs. de farvede felter og læs at de alle ”ingen ejer” har 3. Når nogen har købt en ejendom, tryk igen på feltet på brættet 4. Tjek at den korrekte spiller er angivet som ejer
Forventet resultat	Feltet ejer er nu opdateret med det rigtige navn
Egentligt resultat	Feltet ejer er nu opdateret med det rigtige navn
Status	Er bestået
Testet at	Bashar
Dato	19.01.2020
Test enviroment	Intellij IDEA 2019.3.1(Ultimate Edition) Build: 193.5662.53 built on 18 December 2019 On Windows 10 Home Edition

Figur 9: Test case 2: Test at ejeren af en ejendom vises på brættet

8. Projektplanlægning

Efter at vi havde sat os ind i hvordan Matador spilles, gik vi gang med at lave vores klassediagram og arbejdede med at finde en god struktur i vores system. Vi fik arbejdet os frem til en god skabelon til et klassediagram, hvor der var lav kobling og høj samhørighed

mellem klasserne. Dette har vi brugt som generel pejling i løbet af hele projektet for, hvordan vi gerne ville strukturere systemet. Vi begyndte derefter at skrive koden ud fra ”bottom up-metoden hvor vi lavede de mest uafhængige klasser først, fordi vi derfor havde større frihed til at ændre i vores struktur hvis vi skulle blive klogere undervejs. Med uafhængige klasser mener vi f.eks. Terning, Felt, SpillePlade osv., altså klasser, der ikke styrer ret mange andre klasser. Efter det begyndte vi langsomt at skrive funktionaliteten i vores controller-klasser. Vi fokuserede på at implementere en funktion af gangen, fx opsætningen af spillebrættet, og samtidig undervejs verificerede og testede at det implementerede virkede som vi ønskede.

9. Konfigurationsstyring

Til konfigurationsstyring af vores projekt har vi benyttet os af Maven sammen med IntelliJ for at sikre en god og ensartet konfiguration på tværs af alle udviklere. Ved hjælp af Maven har vi importeret JUnit version 4.12 samt Matador GUI version 3.1.7. De to biblioteker bliver automatisk importeret i IntelliJ og dermed har vi sikret os at alle udviklere benytter sig af samme version af de eksterne biblioteker. Med Maven har vi samtidig også mulighed for nemt at opdaterer hvilken version vi benytter os af, skulle dette blive nødvendigt.

Importering af Git-Reppository i IntelliJ

I IntelliJ kan man importere projeket direkte fra GitHub på følgende måde: File → New → Project from Version Control → Git Derefter indsætter man følgende link og klikker på ”Clone”: **https://github.com/Dalmos87/11_final.git** Efter at IntelliJ har hentet projektet fra GitHub skal man klikke på ”Auto import” i den boksen der popper op nede i højre hjørne. Derefter vil Maven automatisk importere de forskellige biblioteker.

10. Konklusion

Vores projekt lever ikke helt op til kravlisten, som vi selv definerede i starten af projektet. Vi har lavet flere forskellige tests, som alle viser positive resultater. Hvis der havde været mere tid til rådighed, havde det helt sikkert været relevant at lave nogle flere og mere uddybende tests til simpelthen at tjekke mere af programmet. Desuden har vi udeladt at implementere noget funktionalitet fra det officielle Matador på grund af tidsmangel og manglende gruppemedlemmer da vi kun er 4 personer i vores gruppe.

11. Bilag

11.1 Ordbog

Følgende er en oversigt over de vigtigste begreber der bruges i projektet.

Private Hvis en metode eller et felt er private, kan det kun tilgås fra den klasse, det er defineret i.

Public Hvis en metode eller et felt er public, kan det tilgås fra alle steder.

get-metode En public metode der returnerer værdien af en klasses felt, som ellers var private.

set-metode En public metode der ændrer værdien af en klasses felt, som ellers var private.

GIT Versioneringsprogrammet, der bruges til dette projekt.

GitHub Online service hvor GIT-projekter kan tilgås.

Nedarving En klasse kan være nedarvet fra en anden klasse. Det medfører at subklassen ”arver”superklassens metoder. Eksempelvis nedarver alle vores felt-klasser fra den abstrakte felt-klasse.

Abstrakt En abstrakt klasse er en klasse, der ikke kan oprettes objekter af. Dens formål er derfor egentlig at man skal kunne nedarve andre klasser fra den. Eksempelvis er der i vores spil på intet tidspunkt instantieret et objekt af Felt-klassen, mens der er instantieret mange Property-objekter.

Property nedarver, som beskrevet i klassediagrammet, nemlig fra Field.

Polymorfi Polymorfi betyder at flere subklasser, der nedarver fra den samme super-klasse kan have en metode, der hedder det samme, men gør noget forskelligt i hver subklasse. Vi har i vores projekt valgt at bruge polymorfi et enkelt sted, nemlig til getPrice()-metoden. Den returnerer for et propertyfelt ejendommens pris, mens den for andre typer felter returnerer et negativt tal, som indikerer at feltet ikke kan købes.