



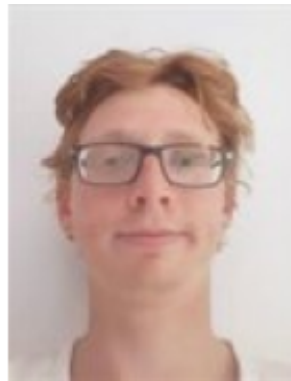
Danmarks
Tekniske
Universitet

62531 udviklingsmetoder til it-system
62532 versionsstyring og testmetoder
02312 indledende programmering

Gruppe 23



Hans Christian
Leth-Nissen
s205435



Henrik Lynggaard
Skindhøj
s205464



Kasper Falch
Skov
s205429



Christoffer Fink
s205449



Fillip Igor Meyer
Clausen
s205468

Henrik Lynggaard Skindhøj, s205464
Hans Christian Leth-Nissen, s205435
Filip Igor Meyer Clausen, s205468

Christoffer Fink, s205449
Kasper Falch Skov, s205429

Github repo:

https://github.com/Kageschwump/23_del3

Timeregnskab

Kasper Falch Skov, s205429	1 timer
Christoffer Fink, s205449	1 timer
Filip Igor Meyer Clausen, s205469	1 timer
Henrik Lynggaard Skindhøj, s205464	1 timer
Hans Christian Leth-Nissen, s 205435	1 timer

Abstract

Indholdsfortegnelse

1. Indledning


I opgavebeskrivelsen har vi fået til opgave at lave et Monopoly Junior spil. Vi har fået udleveret reglerne til spillet, og ud over dette har vi fået mere eller mindre frie rammer til at vurdere hvad vi mener der er det vigtigste i reglerne. I projektet vil vi udarbejde vores krav efter F.U.R.P.S modellen. Efterfølgende vil vi udarbejde nogle use cases, som vi beskriver fully dressed. Herefter vil vi finde frem til nogle passende klasser, som vil være relevante for programmet, og ud fra disse vil vi opstille en række artefakter. Vi vil også lave en række tests, som kunden kan køre, så vi sikrer os, at programmet kører som det skal. Vi har fået udleveret en GUI, som kunden gerne vil have vi bruger i spillet, så den gør vi selvfølgelig også brug af.


2. Kravspecifikation

Vi har valgt at dele vores krav op i FURPS-princippet

Functionality:

Alle krav som ligger her under functionality, har vi lagt her, fordi vi mener at disse krav har noget at gøre med hvordan spillet skal opføre sig, når det er færdigudviklet.

- F1: Spillet skal kunne spilles mellem 2-4 spillere.
- F2: Spilleren skal kunne lande på et felt og fortsætte fra det felt næste gang det er deres tur.
- F3: Den yngste spiller starter.
- F4: Alle spillere starter på feltet "Start"
- F5: Spillerne skal rykke sig med uret
- F6: Spilleren skal altid rykke fremad, aldrig tilbage.
- F7: En spiller rykker det antal felter, som terningen viser i øjne, efter de har slået med den.
- F8: Hver gang man passerer eller lander på start, modtager man 2 Monopoly W-skejs. 
- F9: Alle spillere går på skift, så der bliver skiftet tur, hver gang en spiller har rykket et antal felter
- F10: Hvis en spiller lander på et ledigt felt, **skal** personen betale banken det beløb der står på feltet, og placere et solgt skilt på feltet.
- F11: Hvis en spiller lander på et felt, der er ejet af en anden spiller, skal de betale husleje til ejeren af feltet. Huslejen er det beløb der står skrevet på feltet.
- F12: Hvis en spiller lander på et felt som spilleren selv ejer, sker der ikke noget
- F13: Hvis en spiller ejer begge ejendomme i samme farve er huslejen det dobbelte af det beløb, som står på feltet.
- F14: Hvis en spiller lander på et "CHANCE" felt, skal de tage det øverste af 20 chance-kort fra bunken og følge instruktionerne, herefter skal de placere kortet tilbage nederst i bunken
- F15: Hvis en spiller lander på et "GÅ I FÆNGSEL" felt, skal spilleren gå til "FÆNGSEL" feltet, hvor de ikke modtager 2 skejs for at passere start. I starten af den fængslede spillers næste tur, skal de betale 1 skejs, eller bruge "Du løslades uden omkostninger"-kortet hvis spilleren har det. Derefter skal spilleren kaste med terningen, og rykke normalt. En spiller kan godt modtage husleje, mens de er i fængsel.
- F16: Hvis en spiller lander på et "PÅ BESØG" felt, sker der ikke noget
- F17: Hvis en spiller lander på et "GRATIS PARKERING" felt, sker der ikke noget
- F18: Hvis en spiller ikke har nok penge til at betale husleje, købe en ejendom, som personen lander på, eller betale afgiften fra et chancekort, er personen gået fallit, og spillet slutter.
- F19: Den spiller der har flest penge, når spillet er slut, vinder.
- F20: Hvis 2 spillere har samme antal penge efter spillet er slut, optælles værdien af ejendomme og lægges til pengebeholdningen.

F21: Hver spiller får tildelt et hvis antal , når spillet starter. Antallet er bestemt ud fra mængden af spillere

Usability:

Usability beskriver for det meste en blanding af dokumentation, æstetisk og responsivt design.

U1: Der skal være tydelig sammenhæng mellem beskrivelser og diagrammer, samt implementering af kode.

U2: Projektet skal udarbejdes efter GRASP-mønstre.

Reliability:

Reliability krav er krav der har noget at gøre med om programmet er stabilt, altså at det ikke crasher på underlige tidspunkter. Samtidig sikre vi også at programmet kører som det skal uden diverse bugs.

R1: Der skal laves mindst 3 testcases med tilhørende fremgangsmåder, testprocedure og testrapporter.

R2: Der skal laves mindst én JUnit test til centrale metoder.

R3: Der skal laves mindst en brugertest. Testen skal udføres med en der ikke har kendskab til kodning.

Performance:

DTU's maskiner skal kunne køre kører programmet. Her tester vi programmets helhed ved indførelse af Junit tests, samt kørsel på DTU's maskiner. På DTU's maskiner tids tester vi vores kode.

P1: Spillet skal køre uden bemærkelsesværdige forsinkelser, på DTUs maskiner

Supportability:

Under supportability har vi valgt at stille en række krav, som vi mener gør det nemmere for kunden eller andre spiludviklere at følge vores projekt. Dette gør at kunden kan følge udviklingen af projektet, og hvis spillet skulle outsources til en anden udvikler, vil det også være nemmere at få et overblik over koden.

S1: Projektet skal udarbejdes i Git, da kunden gerne vil kunne følge med i udviklingen af spillet.

S2: Projektet skal indeholde en beskrivelse af hvordan man importerer projektet fra git til IntelliJ, og hvordan man derefter kører programmet.

S3: Koden skal committes hver gang en delopgave er løst, eller mindst en gang i timen

S4: Hvert commit skal indeholde en kort beskrivelse, der forklarer, hvad der er lavet.

3. Analyse


Vi har diskuteret nogle af parametrene i gruppen, og vi har derfor valgt at udelade "Hvem er din brik?"-figurkort, det er kortene under chancekort.pdf, side to, der tilhører de forskellige figurers beskrivelser. Det har vi valgt at gøre fordi vi ikke kommer til at spille med figurer som i eksemplet på Junior Matador, men i stedet kommer til at kunne vælge mellem fire forskellige slags køretøjer(GUI-cars).

I vores kode har vi oprettet en Controller-mappe og Model-mappe, der indeholder alle vores klasser.

Controller-mappen indeholder vores Handler-klasser, *PlayerHandler*, *ChanceCardHandler*, *GuiHandler* og *PlayerHandler*. Det er alle klasser der er med til at styre spillets gang.

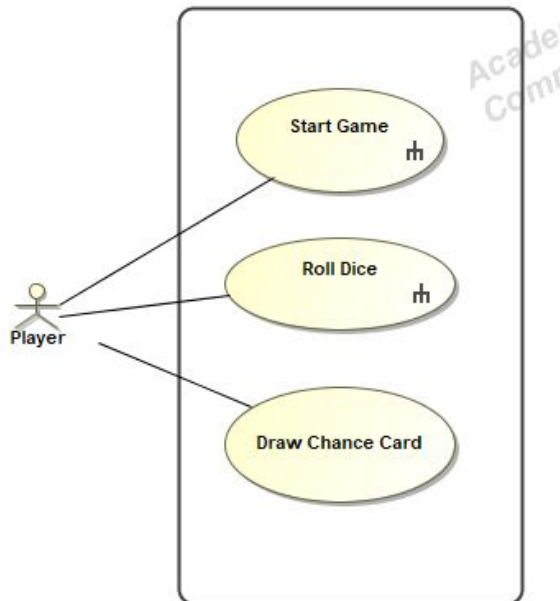
I Model-mappen, har vi oprettet endnu en mappe der hedder *Squaretypes*. Det skyldes at felterne på vores spillebræt har forskellige priser, egenskaber, og farver, for at nævne nogle få, alt efter om det er et *ChanceCardSquare*, *Property*, *ParkingSquare*, *JailSquare*, *VisitJailSquare* eller *StartSquare* man lander på.

Model-mappen indeholder også de resterende klasser der har betydning for hvordan spillet kommer til at se ud. *Account*, *ChanceCard*, *Dice*, *GameBoard*, *GameSquare* og *RuleSet*. *Account*-klassen er spillerens balance, *ChanceCard*-klassen indeholder de 20 forskellige chancekort, *Dice*-klassen er vores terning til spillet, *GameBoard*-klassen indeholder vores GUI-spillebræt, *GameSquare*-klassen er den overordnede klasse for de forskellige felter, og deres størrelser på spillebrættet, de enkelte forskellige spillefelter er defineret i *Squaretypes*-mappen. *RuleSet*-klassen er selve reglerne for spillets gang, eksempelvis når

en spiller lander eller passerer start, skal spilleren modtage 2 , samt hvis en spiller lander på en ejendom der er ejet af en anden spiller, skal spilleren betale husleje, svarende til prisen på ejendommen.

Use case diagram

Use cases



Vi mener ikke der er brug for mere end tre usecases når man snakker om et brætspil. Det eneste vi har brug for er at starte spillet, at slå med terningerne, og at trække et kort, når man lander på et chance-felt. Ud over dette er spillet rimelig selvkørende i og med at det følger et sæt regler, som allerede er defineret. Nedenunder har opstillet vores tre use cases fully dressed.

Use case section:	Beskrivelse
Navn	UC 1.1 - Start game
Scope	System
Level	User Goal
Primær aktør	Player
Preconditions	Spillet er ikke allerede igang
Postconditions	Motherlode er sat op, og klar til at spille
Main Success Scenario	1. Spiller: Kører programmet 2. System: Spørger om antal spillere 3. Player: Indtaster antal spillere 4. System: Opretter spillere 5. System: Beder om navne

Henrik Lynggaard Skindhøj, s205464
Hans Christian Leth-Nissen, s205435
Filip Igor Meyer Clausen, s205468

Christoffer Fink, s205449
Kasper Falch Skov, s205429

	6. player: Indtaster navne 7. System: Beder om alder 8. Player: Indtaster alder 9. System: Starter spillet
Extensions	NaN
Frequency of occurrence	Hver gang der skal startes et nyt spil

Use case section:	Beskrivelse
Navn	UC 1.2 - Roll dice
Scope	System
Level	User Goal
Primær aktør	Player
Preconditions	Motherlode er oprettet/startet
Postconditions	Der er rullet en terning
Main Success Scenario	1. System: Venter på at spilleren klikker 'roll' 2. Player: Klikker på roll, og kaster terningen 3. System: Genererer et tal fra 1-6, lægger dem sammen, og rykker det antal øjne på brættet. 4. System: Printer feltets tilhørende tekst ud 5. System: Prompter en ny spiller til at kaste med terningerne
Extensions	4.1.1 Spilleren: Lander på en

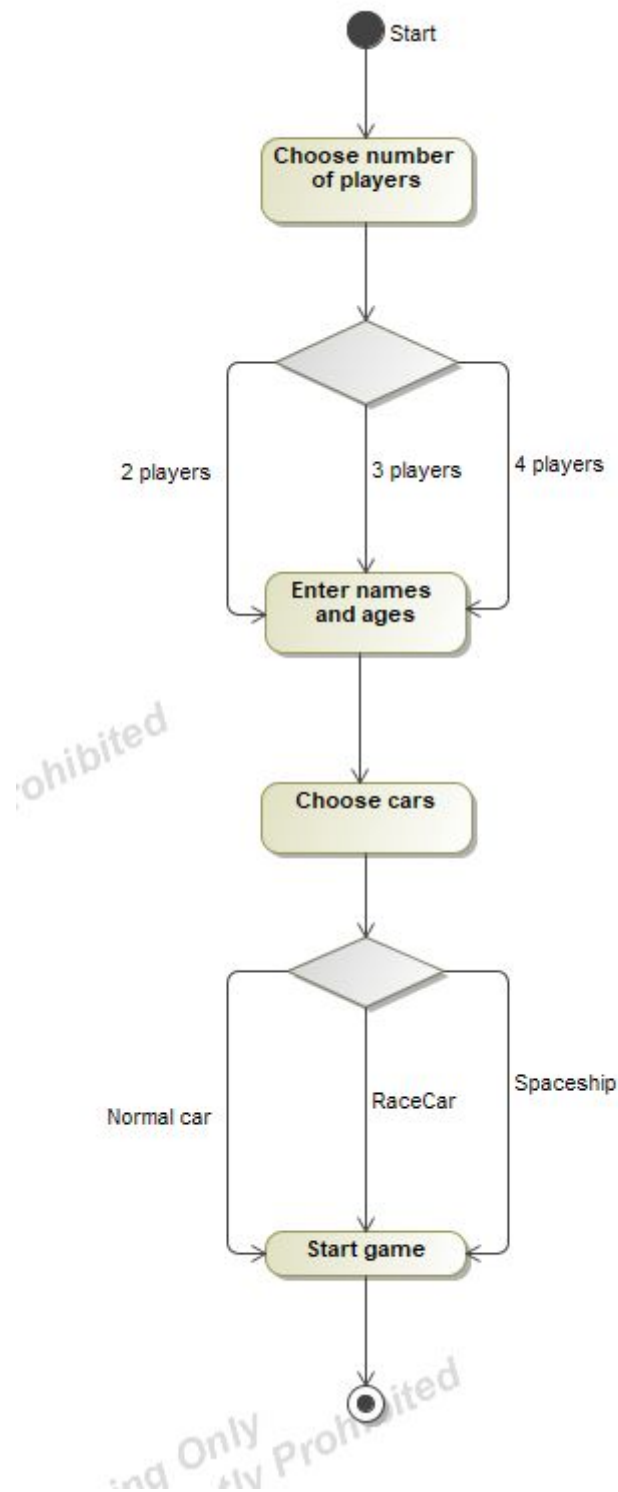
	<p>'prison-square'</p> <p>4.1.2 System: Flytter spilleren til fængslet</p> <p>4.2.1 Spilleren: Lander på en 'chancecard-square'</p> <p>4.2.2 system: Prompter UC 1.3</p> <p>4.3.1 Spilleren: Passerer start</p> <p>4.3.2 system: Tildeler spilleren 2 monopoly money</p> <p>4.4.1 Spiller: Lander på en anden spillers 'property-square'</p> <p>4.4.2 System: Sender penge til spilleren der ejer 'property-square'</p> <p>4.5.1: Player: Lander på en 'property-square' der ikke er ejet af en spiller</p> <p>4.5.2: Player: Køber 'property-squaren'</p> <p>4.5.3 System: Tildeler 'property-squaren' til spilleren</p> <p>4.6.1 Spiller: Lander på en 'visit-jail-square'</p> <p>4.6.2 System: Går videre til næste spillers tur</p> <p>4.7.1 Spiller: Lander på en 'parking-square'</p> <p>4.7.2 System: Går videre til næste spillers tur</p>
Frequency of occurrence	Hver gang en spiller skal kaste en terning, altså hver tur

Henrik Lynggaard Skindhøj, s205464
Hans Christian Leth-Nissen, s205435
Filip Igor Meyer Clausen, s205468

Christoffer Fink, s205449
Kasper Falch Skov, s205429

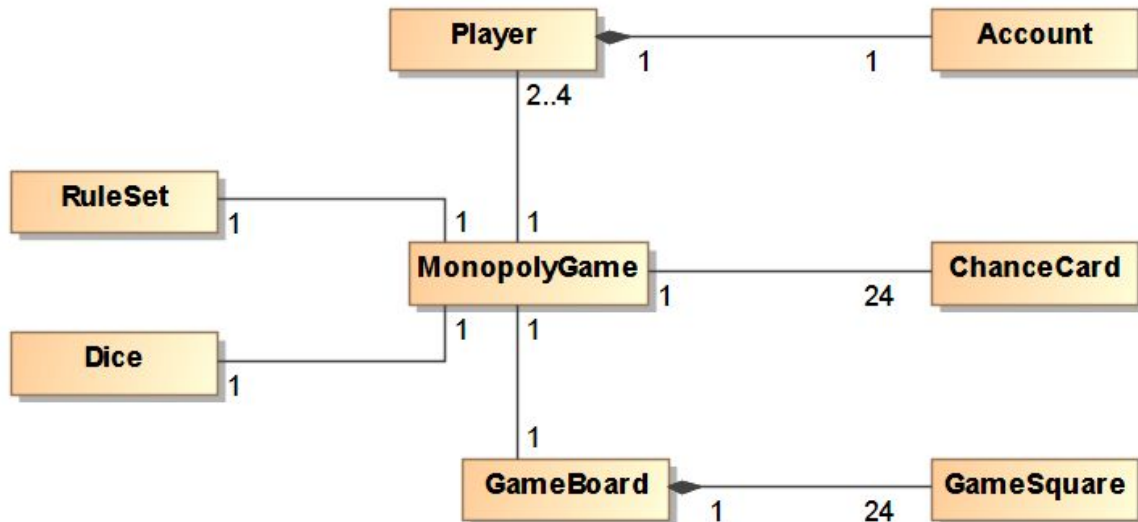
Use case section:	Beskrivelse
Navn	UC 1.3 - Draw card
Scope	System
Level	User Goal
Primær aktør	Player
Preconditions	En player lander på et chance-felt
Postconditions	Chancekortets betingelser er udført
Main Success Scenario	1. System: Trækker et kort for spilleren 2. System: Printer chance-kortets indhold 3. System: Udfører kortets betingelser
Extensions	NaN
Frequency of occurrence	Hver gang en player lander på et chance-kort

Activity diagram



Dette diagram er et aktivitet flow når man starter spillet. Vores spil starter med en menu, hvor der kun er 3 valg. Alle valg omhandler antallet af spillere, hvilket leder brugeren til en ny menu prompt, hvor spillerne vælger navn og alder. Hvis spillere har samme alder, så vælger den en tilfældig starter. Efter vælger spillerne biler og deres farve.

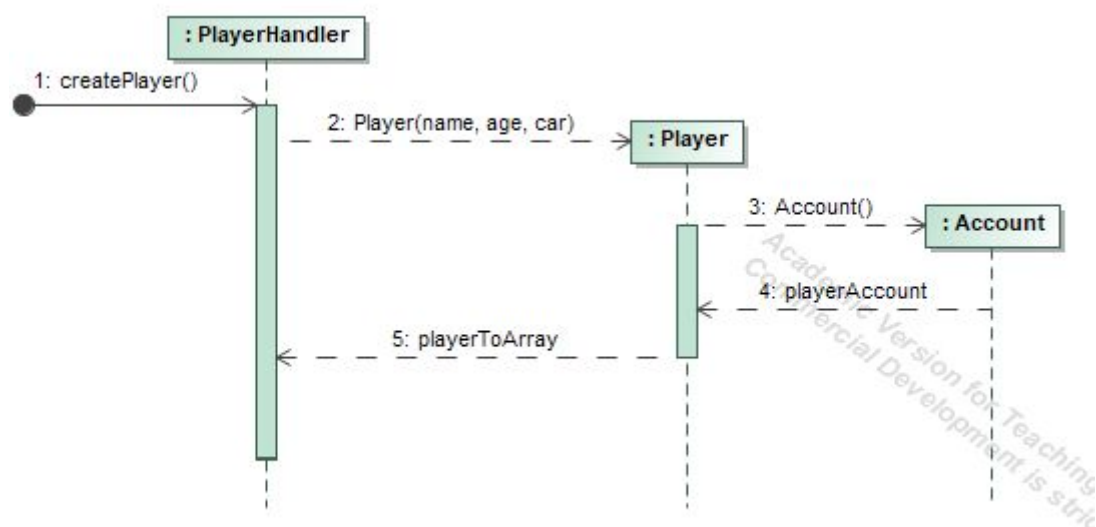
Domænemodel



Figur 1:

Vores domænemodel forsøger at beskrive spillet og associationerne. I vores tilfælde har vi valgt at angive "MonopolyGame" som en klasse. Vi har i domænemodelen også inkluderet multipliciteten imellem vores klasser, for eksempel har 1 GameBoard 24 GameSquares. Vores tidligere fejl var at inkorporere software klasser i domænemodellen.

System Sekvensdiagram

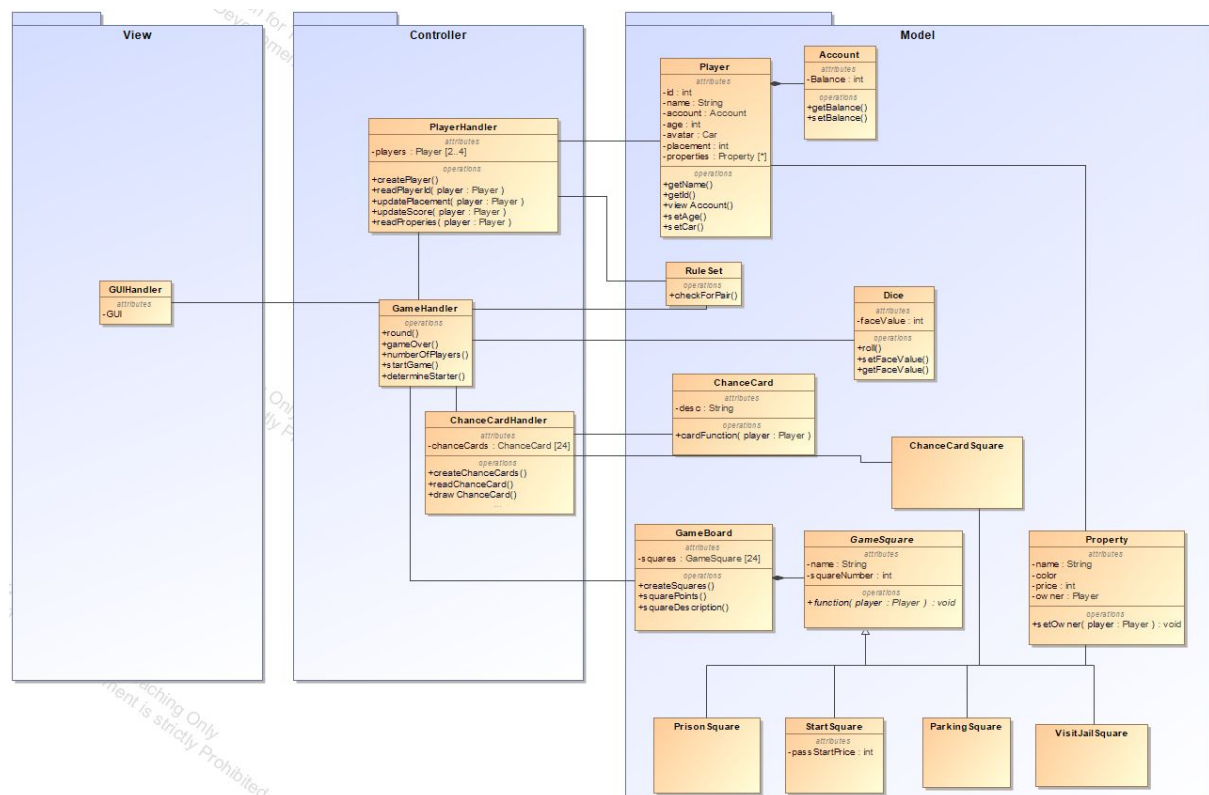


Figur 2:

Vores createPlayer metode er essentiel, da den opretter flere objekter. Både player og account er oprettet. Disse objekter SKAL have navn, alder og en bil, som avatar på gameboardet. Player klassen har også default værdier. Placering er en default værdi, da alle spillere starter på samme sted.

5. Design

Design Klassediagram



Figur 3:

I vores design klassediagram, kan det ses hvordan vores klasser interagerer med hinanden, i form af deres relationer, metoder og attributter. Vi har udarbejdet vores klassediagram ud fra rapportens analysedel.

Sekvensdiagram

GRASP mønstre

Vi har udarbejdet vores projekt med brug af GRASP principperne.

Creator:

Henrik Lynggaard Skindhøj, s205464
Hans Christian Leth-Nissen, s205435
Filip Igor Meyer Clausen, s205468

Christoffer Fink, s205449
Kasper Falch Skov, s205429

Information expert:

Controller:

Associationsproblemer:

Vores *GameSquare* henter informationer fra *ChanceCardSquare* og *Property*, da en række af chancekortene har en funktion der hedder ryk til et felt af en bestemt farve, hvis ejendommen er ledig får spilleren den. Hvis den ikke er ledig skal spilleren betale husleje, det vil sige at chancekortet skal kunne trække de informationer et sted fra, og det gør den fra *Property*. Chancekortet trækkes og læses ved hjælp af *ChanceCardSquare*, der læser kortet.

High cohesion:

Vi har i vores design klassediagram udarbejdet vores klasser med den tanke om at overholde high cohesion. Dette har vi opnået ved at sørge for at vores klassers metoder ikke krydser nogle andre klassers metoder, dette har vi sørget for ved give hver af vores klasser et veldefineret ansvarsområde, og sørget for at vi ikke skabte nogle klasser der ikke var behov for, eller var overflødige.

Low coupling:

Vi har i projekt vores forsøgt at skabe et stykke software, som opfylder low coupling princippet. Dette vil sige at vi har sørget for at de klasser vi har skabt, har lav afhængighed af hinanden, og ingen unødvendige associationer. Dette gør at vores klasser bliver nemmere at forstå, og læse, og samtidig giver det mulighed for at ændre på nogle klasser, senere hen i projektet, uden at der skal laves store ændringer på mange forskellige klasser, da hver klasse har lav afhængighed af de andre.

Polymorphism

6. Implementering

- Lav passende konstruktører.

- Lav passende get og set metoder.

- Lav passende toString metoder.

- Lav en klasse *GameBoard* der kan indeholde alle felterne i et array.

Henrik Lynggaard Skindhøj, s205464
Hans Christian Leth-Nissen, s205435
Filip Igor Meyer Clausen, s205468

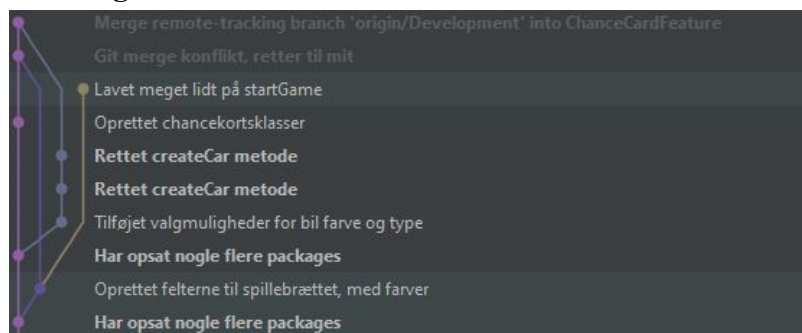
Christoffer Fink, s205449
Kasper Falch Skov, s205429

Tilføj en toString metode der udskriver alle felterne i arrayet.
Lav det spil kunden har bedt om med de klasser I nu har.
Benyt GUI'en. Gui' skal importeres fra Maven: [Maven repository](#)

7. Dokumentation

Forklar hvad arv er.
Forklar hvad abstract betyder.
Fortæl hvad det hedder hvis alle fieldklasserne har en landOnField metode der gør noget forskelligt.
Dokumentation for test med screenshots.
Dokumentation for overholdt GRASP.

Git merge konflikt:



Projektet havde en del merge konflikter. Da vi hver især programmeret på samme tid og committede. Vi brugte IntelliJ til opstilling af forskellen mellem vores branches og commits, hvor IntelliJ spurgte om prioritering af kode.

Ved merge konflikten holde vi gruppemøde og gennemgik vores ændringer. Dette viste sig at være en effektiv problemløsning. Alle gruppemedlemmernes kode blev implementeret uden behov for revert.

8. Test

Code coverage

9. Projektplanlægning

Henrik Lynggaard Skindhøj, s205464
Hans Christian Leth-Nissen, s205435
Filip Igor Meyer Clausen, s205468

Christoffer Fink, s205449
Kasper Falch Skov, s205429

Versionsstyring

Konfigurationsstyring

10. Konklusion

Bibliografi

Programmerings bogen

Lewis, J.. & Loftus, W.. (2017). *Java Software Solutions: Foundations of Program Design* (9.. udg.). Pearson Education Limited.

UML bogen

Larman, C.. (2004). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development* (3.. udg.). Pearson Education.

Vi har i vores projekt genbrugt nogle dele af rapporten og koden fra CDIO 2.

Bilag