

G15 CDIO delopgave 3

Indledende Programmering

02312-14

Udviklingsmetoder til IT-Systemer

02313

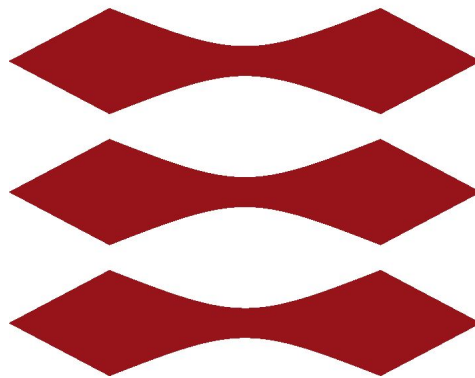
Versionsstyring og Testmetoder

02315

Afleveringsfrist: 29. november 2019

[Github Group 15 Repository](#)

DTU



Gruppe 15



Christian
s184140



Kamilia
s195466



Mina
s195381



Oliver
s176352



Sara
s195388



Tobias
s195458

Abstract (Kamilia)

This assignment is a combination of three courses, “Udviklingsmetoder til IT systemer”, “Indledende programmering” and “Versionsstyring og Testmetoder”. With the applications, MagicDraw, Github and IntelliJ, we have the ability of utilizing these to our main task, which is to produce a replica of a junior monopoly game, which needs to fulfill the specific requirements given from a customer. This assignment includes various diagrams as well as the coding behind the junior monopoly game. Our aim is to create a monopoly game that can be played by two to four people, where it’s possible to earn money, buy properties and much more. The game only has a positive outcome or a negative outcome of going fallit. In the end it is concluded, that a proper functioning replica of a junior monopoly game can be programmed by using different components from the three courses.

Indholdsfortegnelse:

1. Indledning (Sara og Kamilia)	4
2. Analyse	5
2.1 Krav (Christian)	5
2.2 Use cases (Tobias)	7
2.3 Domæne diagram (Tobias, Christian & Oliver)	8
2.4 SystemSekvensdiagram (Sara)	9
3. Design	10
3.1 Klassesdiagram (Tobias, Christian og Oliver)	10
3.2 Sekvensdiagram (Sara)	11
3.3 GRASP - mønstre (Kamilia og Christian)	12
4. Implementering (Mina, Oliver og Kamilia)	13
4.1 Koden (Interessant eksempel)	13
4.2 Dokumentation	15
5. Test	17
5.1 Testcases (Tobias og Christian)	17
5.2 JUnit test (Mina & Christian)	19
5.3 Brugertest (Tobias og Sara)	20
5.4 Databaser test (Sara)	21
6. Projektplanlægning	22
6.1 Tidsplan (Tobias)	22
6.2 Projektforløb (Sara)	22
7. Konfigurationsstyring (Oliver)	23
7.1 Udviklingsmiljø	23
7.2 Produktionsmiljø	24
7.3 Versionering	24
8. Konklusion (Mina)	25
9. Bilag	26
Bilag 1: chancekort	26
Bilag 2: Testcases Data	28
Bilag 3: Unified process for timeregnskab	31

Timeregnskab (Tobias)

dato	Krav og Analyse	Design	Implementering	Test
11/11/2019	3	0	1	0
12/11/2019	5	2	0	0
13/11/2019	0	1	0	0
14/11/2019	3	0	0	0
15/11/2019	1	2	2	0
16/11/2019	5	1	0	0
17/11/2019	0	4	4	2
18/11/2019	2	0	3	0
19/11/2019	4	1	12	0
20/11/2019	2	7	0	0
21/11/2019	0	2	4	0
22/11/2019	2	3	6	2
23/11/2019	0	1	0	4
24/11/2019	1	5	4	1
25/11/2019	0	0	1	5
26/11/2019	0	1	0	3
27/11/2019	1	0	2	8
28/11/2019	0	0	0	1
29/11/2019	1	0	1	0
Timer i alt	30	30	40	26

Navn	Krav og Analyse	Design	Implementering	Test	I alt
Christian	7	6	8	6	22
Kamilia	5	4	4	4	17
Mina	4	8	4	2	18
Sara	6	6	4	3	19
Oliver	5	6	15	4	27
Tobias	6	4	6	7	23
Timer i alt	30	30	40	26	126

1. Indledning (Sara og Kamilia)

Som videre arbejde til de to første opgaver, er vi blevet bedt om at lave et fuldendt matador spil. Programmet skal være mellem 2-4 spillere, hvor man kan lande på forskellige felter, trække chancekort og hvorefter balancen ændrer sig herefter. Hvert felt skal have forskellige positive eller negative effekter for scoren, hvilket også gør sig gældende for chance kortene.

Til at udarbejde produktet, har vi gjort brug af IntelliJ, GitHub og Magicdraw til hhv. at programmere, versionere og lave diagrammer og modeller. Valget af programmer gør, at vi alle kan arbejde selvstændigt samtidig med at man deler det med hinanden.

Produktet som er en replica af et junior matador spil er lavet med udgangspunkt i de forskellige krav og designet ud fra vores domæne diagram. Derudover har vi videreudviklet programmet ud fra vores use cases og de dele af programmet som vi i forvejen havde arbejdet med i de foregående opgaver.

2. Analyse

2.1 Krav (Christian)

Kundens vision:

Kunden ønsker et system, som kan køre på Windows maskinerne i data barerne på DTU. Spillet er en replikation af klassikeren Monopoly Junior. Spillet skal foregå mellem to til fire spillere.

Spillerne bevæger sig i ring på en decideret spilleplade, hvor forskellige felter afgøre spillernes skæbne. De overordnede felter er startfeltet, grunde, fængsel, på besøg fængsel, gratis parkering (neutral vente plads) og chancekort. Spillet indeholder 14 forskellige chancekort, som kan give fordel eller ulempe til spilleren, kan findes i bilag 1.

Spillerne skiftes til at kaste med to terninger, hvilket afgør, hvor langt de skal bevæge sig på spillepladen. Når en spiller lander på en grund, som er til salg, skal spilleren købe grunden. Grunde udløser en afgift til andre spilleren, som betales til grundejeren når en spiller lander på det pågældende felt. Spillet vindes ved en spiller går fallit, fordi de ikke kan betale deres afgift eller ikke betale for den grund de er landet på. Herefter optælle de resterende spiller deres penge, hvor den rigeste vinder.

Kravspecifikation:

Funktionelle krav:

- Spillet skal være mellem to til fire personer
- Spillerne skiftes til at kaste to terninger
- Turen skifter hver gang en spiller har kastet de to terninger.
- Startpenge:
 - 2 personer: 20 monopoly dollar hver
 - 3 personer: 18 monopoly dollar hver
 - 4 personer: 16 monopoly dollar hver
- Spillerne bevæger sig skiftevis rundt i ring på spillebrættet
- Spillebrættet består af 24 felter.
- Spillerne rykker skiftevis rundt, i en længde bestemt af summen af de to terninger.
- En spiller rykker altid frem aldrig tilbage.
- Spillebrættet består af følgende felter: start, ejendomme, chancekort, på besøg i fængsel og i fængsel.

- Når spilleren lander eller passerer start modtages 1 monopoly dollar
- Hvis en spiller lander på en ledig ejendom, skal spilleren købe grunden til den angivet pris i bunden af feltet. Køb foregår ved en betaling til banken.
- Hvis en spiller lander på en ikke ledig ejendom, skal spilleren betale ejer med den angivet pris i bunden af feltet.
- Hvis en spiller lander på et felt de selv ejer skal de ikke gøre noget.
- Ved hvert kast udskrives en tekst angående det aktuelle felt spilleren lander på. I tilfælde af en afgift eller køb af grund ændres spillerens penge balance.
- Spillet stopper når en spiller går fallit.
- Spiller mister penge ved køb af ejendom, afgift fra chancekort eller lande på andres ejendom.
- Når et spil stopper, optælles de resterende spillers penge. Den rigeste spiller vinder.
- Hvis en spiller lander på et chancekort, skal de tage en af de 14 chancekort. Aktionen præsenteres ved en tekst, hvorefter aktionen påføres spilleren.
- Hvis en spiller rammer feltet "Går i Fængsel" rykkes spilleren direkte til fængslet. Spilleren modtager ikke 1 Monopoly Dollar for at krydse start. Runden efter skal spilleren betale 1 Monopoly Dollar eller benytte "Du løslades uden omkostninger"-kortet, hvis indebærer ejer dette chancekort. Herefter kastes terninger som normalt. Det er muligt at modtage husleje når man er i fængsel
- Spillerne må ikke have det samme navn

Ikke funktionelle krav:

- Spillet skal kunne tilgås via maskinerne i DTU's databarer, uden bemærkelsesværdige forsinkelser.
- Der skal udarbejdes en beskrivelse af minimumskrav samt vejledning i hvordan kildekoden compiles, installeres og afvikles. Dette inkluderer en beskrivelse af hvordan koden importeres fra et git repository.

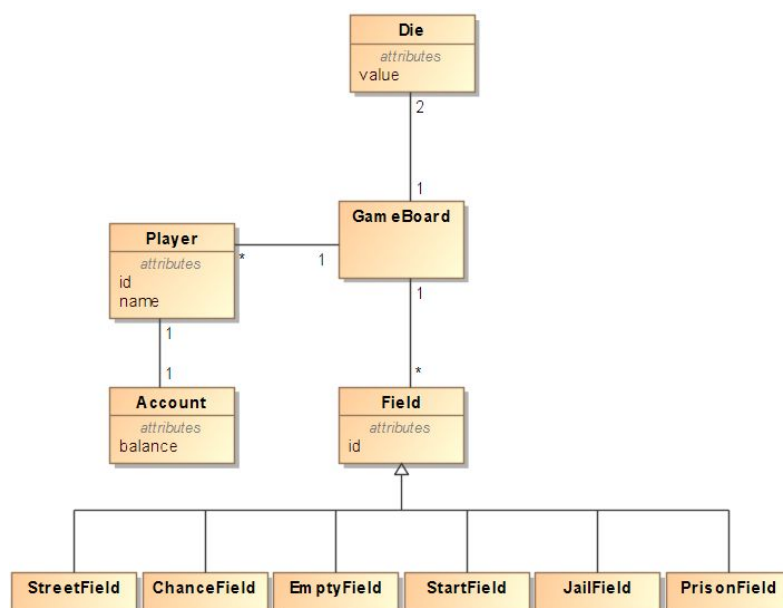
2.2 Use cases (Tobias)

Ligesom de forrige spil har spilleren kun mulighed for at slå med terningerne. Hver gang han har slået skal og lander på en grund han enten betale for at købe et felt eller give penge til spilleren der ejer feltet. Spilleren kan også lande på felter med chancekort eller fængsel men der bliver spillerens handlinger også styret af systemet. Der er også et pause felt der ikke gør noget og hvis spilleren lander på et felt han allerede ejer sker der heller ikke noget.

Brief Use Case	
UC1: Spil Monopoly Junior Man spiller 2-4 spillere på en plade med en bestemt mængde felter. I spillet rykker hver spiller rundt hvor de enten får eller mister penge. Spillet er slut når en spiller er gået bankerot og den af de resterende spillere der har flest penge har så vundet.	
Casual Use Case - UC1: Spil Monopoly Junior	
Spillet starter med aktørerne angiver antallet af spillere samt deres individuelle navne. Mængden af spillere går fra to til fire og afgør de individuelles startkapital. Hertil slår systemet så med terningen på hver spillers tur og flytter spilleren til det rigtige felt. På hvert felt skal spilleren købe feltet, give penge til ejeren eller trække et chancekort. Chancekort har mulighed for at give eller tage penge samt uddele frikort til at undgå fængsel. Spillet er slut ved at en af spillerne går bankerot og ender med en balance på 0. Hertil er vinderen den spiller tilbage med flest penge.	
Fully dressed	
Use Case Navn	UC1: Spil Monopoly Junior
Scope	Spil Monopoly Junior
Level	User goal - Spille Monopoly
Primær aktør	Spiller
Interessenter	Ingen
Preconditions:	Spillerne initialiseres med deres individuelle navne og passende startkapital afhængig af antal spillere.
Postconditions/ Success Guarantee	En spiller er gået bankerot og en vinder er fundet
Hoved Succes scenarie	<ol style="list-style-type: none">1. Spiller i fængsel betaler kaution2. Spiller kaster med terning3. Spiller rykker til nyt felt4. Ny spillers tur5. Trin 1-4 gentages indtil en spiller er bankerot6. Spilleren med flest penge tilbage har vundet

Extensions	<p>1a. Spiller betaler for at komme ud af fængslet</p> <p>1b. Spiller bruger sit frikort og undgår at betale</p> <p>3a. Spilleren lander på felt med ejendom</p> <p>1a. Køber ejendommen</p> <p>1b. Betaler penge til ejeren</p> <p>3b. Spilleren lander på felt med chancekort</p> <p>1a. Spiller modtager penge eller mister penge</p> <p>1b. Spiller modtager frikort til fængsel</p> <p>1c. Spiller rykker til en ejendom og modtager grunden gratis eller betaler leje hvis grunden ejes af en anden spiller.</p> <p>3c. Spilleren lander på gå til fængsel og ryger direkte i fængsel</p> <p>3d. Spiller lander på gratis parkering eller besøg fængsel og slutter sin tur</p> <p>3e. Spiller passerer start og modtager 1 monopoly dollar</p> <p>6a. Hvis en spiller er gået bankerot og to eller flere spillere har samme højeste antal penge er der flere vindere</p>
Frequency of Occurrence:	2-4 gange per runde afhængig af antal spillere.

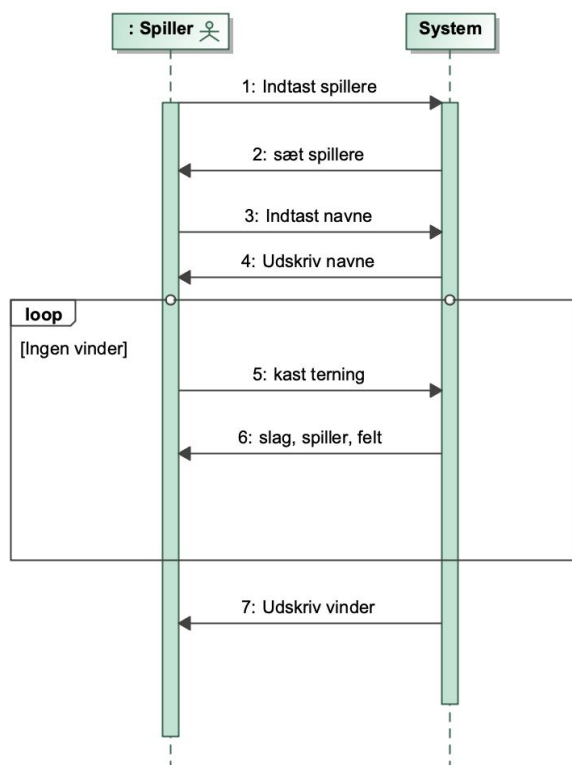
2.3 Domæne diagram (Tobias, Christian & Oliver)



Figur: 2.4.1 Domæne Diagram

Vores domæne diagram er opsat meget ligesom det forrige projekt da vi også tager meget udgangspunkt i det og bruger klasserne derfra til spillet. De seneste tilføjelser er gameboard og field samt de forskellige typer af fields som arver fra field. Dette er er hvad vi mener burde dække alle de behov vi har ud fra vores krav om som burde beskrive vores produkt bedst muligt. Denne opsætning burde også give plads til tilføjelser og ændringer når vi arbejder med selve koden.

2.4 SystemSekvensdiagram (Sara)



Figur: 2.4.1 SystemSekvensdiagram

Da vores use cases er meget simpel, har vi valgt at lave vores systemsekvensdiagram over hele spillet. Vores matadorspil kører mellem en spiller og spillet. Til at starte med, skal spillerne vælge antal spillere, og derefter skrive sit navn for at blive sendt videre til spillepladent. Herefter returnere spillet antal spillere og navnene. Spillerne skal kaste terninger, og derefter returnere spillet resultatet og den samlede score ud fra hvilke felter spilleren lander på ud fra terningernes øjne. Det kan af fase 5-6 ses, at hvis ikke der er defineret en grænse for en vinder, kører disse i et loop. Til sidst udskriver spillet en vinder.

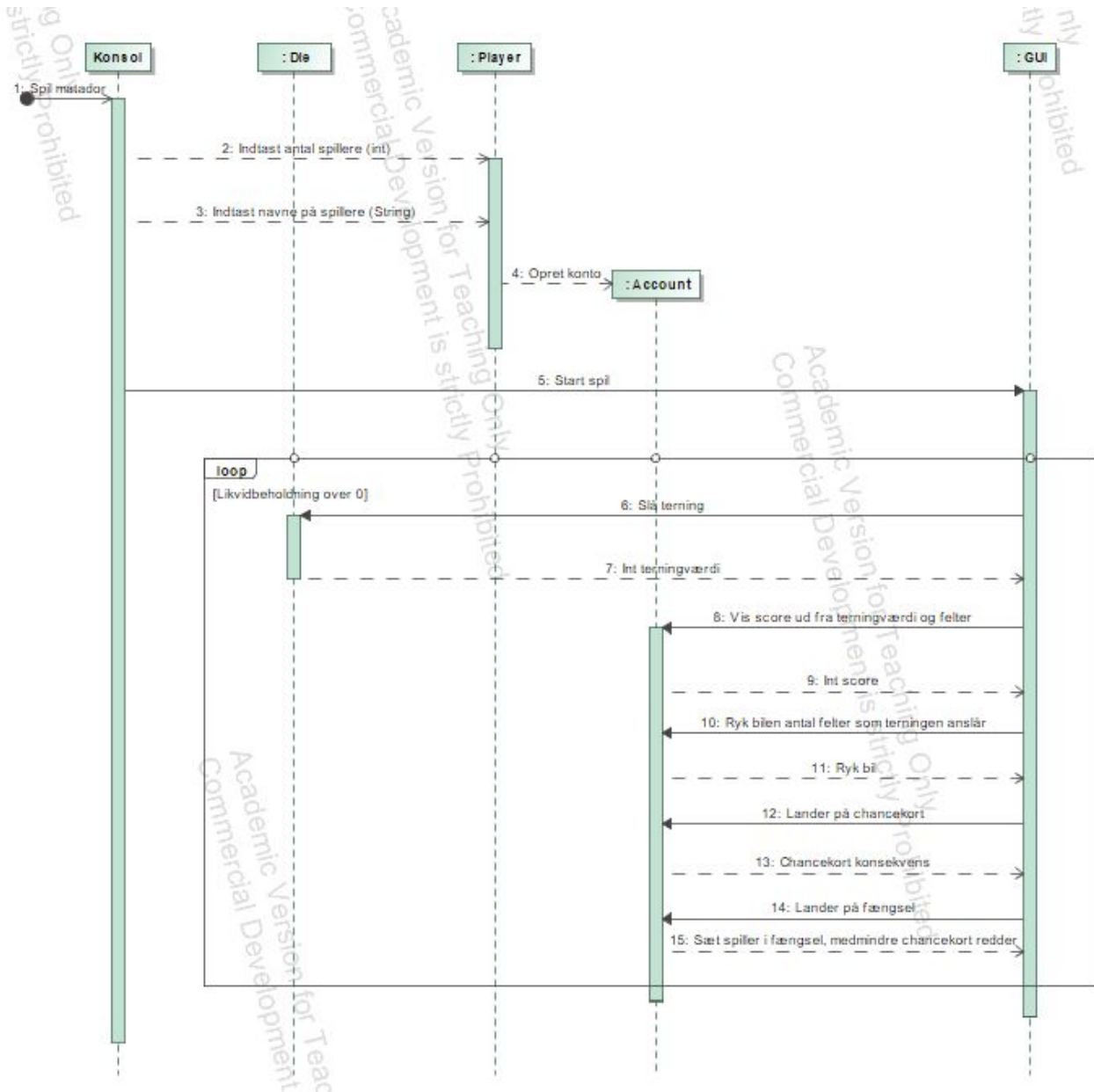
3.1 Klassediagram (Tobias, Christian og Oliver)

Grundene til dette er hvordan vi skal kommunikere med vores GUI og den måde vi så har været nødt til at sætte vores klasser op. De klasser vi havde fra det gamle projekt ligner stadig meget det samme med nogle få nødvendige tilføjelser til at kommunikere med de nye klasser vi har tilføjet.



side 10 ud af “antal” 32

3.2 Sekvensdiagram (Sara)



figur: 3.2.1 Klassediagram

Vi har samlet vores sekvensdiagram over hele vores projekt, da det synes at give det bedste overblik. Her kan man se forbindelserne mellem konsollen, terningerne, spillerne, kontoen og selve spillet. Når man slår med terningen, vil man kunne se terninge værdi, balance og evt chancekort og fængsel. Man kan også se sin bil flytte sig rundt på pladen. Igen har vi et loop, men denne gang opløftet af vores grænse på en likvid beholdning på under 0 point, hvor en spiller har vundet.

3.3 GRASP - mønstre (Kamilia og Christian)

GRASP som står for General Responsibility Assignment Software Patterns bruges til objekt orienteret design til at allokere ansvar til klasser.

Creator

Objekt-skabelse af en af de hyppigste aktiviteter i et objektorienteret system. En Creator angiver, hvem der skal være ansvarlig for skabelsen af en ny instans af en klasse. Dette afgøres ud fra interaktionen og associationen mellem objekterne. Eksempelvis har vi en SpilButik med dens Spil. Spilbutik kender til Spil, men ikke omvendt. Her er det altså SpilButik der initialiserer Spil objekter i sin egen klasse. SpilButik er altså Creator.

Information Expert

Et objekt bliver tildelt det ansvar den har information til at opfylde. Antag vi ønsker at købe alle spillene i SpilButik. Her vil SpilButik være Information Expert, da den kan give os den ønskede efterspørgelse, da den har den rigtige information. Vinder-klassen og taber-klassen indeholder informationer som de selv skal anvende, derfor beskrives de også som information expert klassen.

Controller

Controlleren tildeler ansvar til håndtering af inputs fra en brugergrænseflade. Altså hvem der videre kommunikere ændringer i modellen til et eventuelt view lag. Der er ofte det antal Controller, som der er use case scenarier.

Polymorphism

Polymorfi omhandler håndtering af relaterede klasser. Polymorfi kan opstå når en metode går igen i flere klassen, men med små variationer. Det sker ved initialisere en super-klasser der indeholder en abstrakt metoden fåRumfang. Alle subklasser skal nu arve denne metode. Sub-klasserne kunne eksempelvis være pyramide og kugle.

Low Coupling

Her er der fokus på lav afhængighed mellem klasser. Dermed kan man isolere ændringer, nemmere forståelige design og nemmere at genbruge.

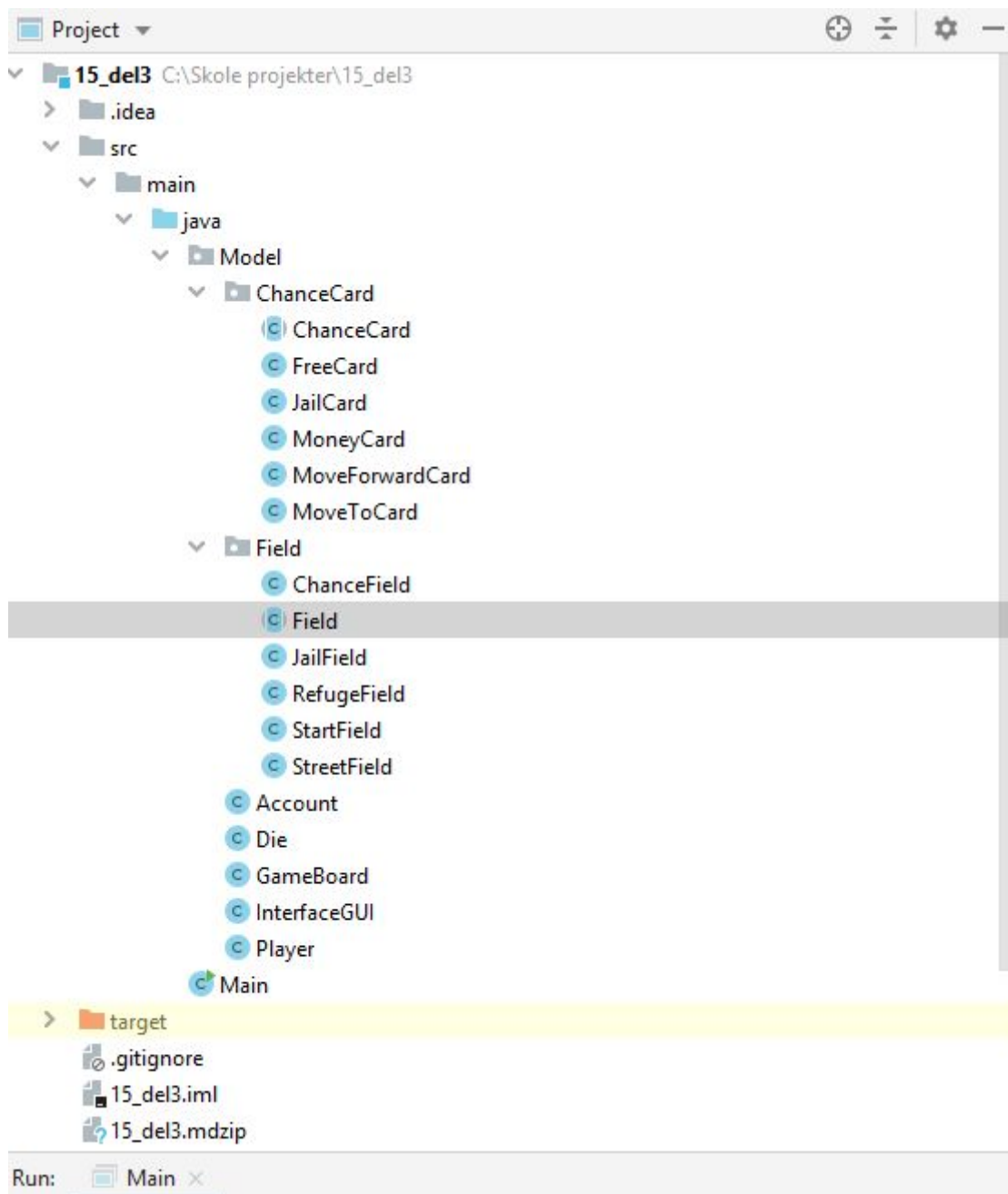
High Cohesion

High Cohesion forlænger Low coupling. Her tildeles tydeligt og specifikt ansvar til de enkelte klasser. Dette er med til at skabe Low coupling

4. Implementering (Mina, Oliver og Kamilia)

4.1 Kodens (Interessant eksempel)

Projektet struktur ser sådan her ud



På billedet ses det at vi i model package har diverse klasser og herinde er der også ChanceCard package og Field package. Disse har en overordnet klasse, henholdsvis ChanceCard og Field, som er abstrakte og arves fra de andre filer i deres package. Main klassen er kun for “boot” af programmet og starter altså kun GameBoard op.

En interessant eksempel kan ses ved at kigge på ChanceCard klassen som er abstrakt og to klasser der arver derfra.

```
package Model.ChanceCard;

abstract public class ChanceCard {
    private String name;
    public ChanceCard(String name) { this.name = name; }
    public String getName() { return name; }

    protected int getRandomInt(int min, int max) { return (int) Math.floor(Math.random() * (max - min + 1)) + min; }
}
```

Ovenover ses ChanceCard klassen som er abstrakt og altså kun bruges til arv. ChanceCard har en constructor som skal arves når klassen bliver arvet. Det vil altså sige en constructor kan fungere forskelligt i henholdsvis MoneyCard og MoveToCard klassen men constructoren i ChanceCard klassen skal kaldes fra MoneyCard og MoveToCard klassen.

```
package Model.ChanceCard;

import ...

public class MoneyCard extends ChanceCard {
    private int income;

    public MoneyCard(String name, int income) {
        super(name);
        this.income = income;
    }

    public void playerGetIncome(Player player){
        player.getAccount().setBalance(player.getAccount().getBalance() + income);
        InterfaceGUI.setGuiPlayerBalance(player);
    }

    public void playerGetIncomeFromOther(Player player, Player[] players){
        for(Player otherPlayer : players){
            otherPlayer.getAccount().setBalance(otherPlayer.getAccount().getBalance() - income);
            player.getAccount().setBalance(player.getAccount().getBalance() + income);
            InterfaceGUI.setGuiPlayerBalance(otherPlayer);
        }
        InterfaceGUI.setGuiPlayerBalance(player);
    }
}
```

MoneyCard som ses her arver altså fra ChanceCard og man kan se hvordan dens constructor er udvidet med også at sætte income. Det er altså fordi når man trækker et MoneyCard så skal spilleren have ændret sin balance.

Et andet kort er MoveToCard

```
package Model.ChanceCard;

import ...

public class MoveToCard extends ChanceCard {
    private int position;
    public MoveToCard(String name, int position) {
        super(name);
        this.position = position;
    }

    public void movePlayerToPosition(Player player, Field[] fields) { player.setFieldPosition(position, fields); }
}
```

Her ses det hvordan MoveToCard også har udvidet ChanceCard men denne klasse sætte i stedet for positionen til noget og det er altså fordi at en spiller skal have ændret sin position når de trække denne type af kort.

4.2 Dokumentation

Dokumentationen for dette projekt beskriver nogle bestemte begreber vi er blevet bedt om at uddybe og forklare. Hertil er der også noget dokumentation for vores test og hvordan vi har fulgt GRASP principperne i koden og produktet.

Arv

Arv betyder, at man kan udlede en ny klasse fra en allerede eksisterende klasse. Den eksisterende klasse kaldes “parent class”, “super class” eller “base class”, og den udledte klasse kaldes “child class” eller “subclass”. Som navnet hentyder, nedarver “child class” fra “parent class”, den nye klasse arver de metoder og data som allerede findes i den eksisterende klasse. Det er muligt at ændre den nedarvede klasse ved at tilføje nye variabler og metoder, og ved at tilpasse de eksisterende. I java bruges det reservede ord “extends” til at etablere et nedarvnings forhold.

Typisk kan man se om man kan bruge arv, ved brug af “er” reglen. Hvis man kan sige at den nedarvede klasse “er” en type/form af den eksisterende klasse, er der nok tale om arv. Man kan fx. sige at en bil *er* en slags køretøj. Dermed har klassen “bil” nedarvet fra klassen “køretøj”.

Abstract

En klasse kan også defineres som værende abstrakt, dvs at der ikke kan initialiseres et nyt objekt af den klasse. En klasse indeholdende abstrakte metoder, er abstrakt, og kan ikke instantieres. I et

UML diagram vil navnet på den abstrakte klasse stå i kursiv. Abstrakte klasser bliver oftest brugt for nedarvning. Det kan for eksempel være klassen “køretøj” som nævnt tidligere. Det giver mening at en bil arve for et køretøj men det giver knap så meget mening hvis man kunne lave et objekt på klassen køretøj, da et køretøj i sig selv ikke afgør om hvorvidt det er en “bil” eller et andet køretøj.

Hvad det hedder hvis alle fieldklasserne har en `landOnField` metode der gør noget forskelligt.

Det hedder polymorfi og betegner når en klasse “bil” og en klasse “motorcykel” begge arver metoden “drive” men gør noget forskelligt. Fx kunne det være at når bilen eller motorcyklen skulle køre at de accelereret med forskellige størrelser.

Dokumentation for overholdt GRASP.

```
public void setupPlayers() {
    // Initialisering af spillere
    Scanner input = new Scanner(System.in);
    int totalPlayers;
    while (true) {
        try {
            System.out.println("Indtast antal spillere mellem 2 og 4:");
            totalPlayers = input.nextInt();
            if (totalPlayers > 1 && totalPlayers < 5) break;
        } catch (Exception e) {
            input.nextLine(); // Fordi nextInt ikke kalder \n som er ny line, derfor bliver denne automatisk kaldt
        }
    }
    input.nextLine(); // Fordi nextInt ikke kalder \n som er ny line, derfor bliver denne automatisk kaldt
    players = new Player[totalPlayers];
    for (int i = 0; i < totalPlayers; i++) {
        System.out.println("Indtast spiller nr. " + (i + 1));
        String playerName = input.nextLine();
        boolean exist = false;
        for (Player p : players) {
            if (p != null && p.getName().equals(playerName)) {
                exist = true;
            }
        }
        if (!exist) {
            Account account = new Account( balance: 20);
            if (totalPlayers == 3) account = new Account( balance: 18);
            else if (totalPlayers == 4) account = new Account( balance: 16);
            players[i] = new Player(playerName, account);
        } else {
            i--;
            System.out.println("Navnet eksisterer allerede.");
        }
    }
}
```

Figur: 4.2.1 Screenshot af Gameboard

Her kan det ses, at Gameboard er under Control klassen men også creator klassen, da den fordeler arbejdet men også oprettelsen af de forskellige objekter.

5. Test

5.1 Testcases (Tobias og Christian)

Vi har lavet nogle test cases for initialiseringen af spillet, om logikken i spillet virker og om interaktion med felterne er som de skal være. Alle test data kan findes under bilag 2: testcases og indeholder alt test-dataen for de forskellige testcases. vores krav er udvalgt med udgangspunkt i at dække mest muligt af programmet.

Test case ID	TC01
Summary	Test for initialisering af spil
Requirements	<ul style="list-style-type: none">- Spillerne må ikke have det samme navn- Spillet skal være mellem to til fire personer- Startpenge: 2 personer: 20\$, 3 personer: 18\$, 4 personer: 16\$.
Preconditions	Programmet er installeret korrekt
Postconditions	Spillet kører
Test procedure	<ol style="list-style-type: none">1. Indsæt værdier for antal spillere2. Indsæt værdier for spillernavne3. Checker startværdier
Test data	Antal spillere: (1, -5, test, "j%a2k75, 65413959325021391359, 3) Spillernavne(for 4 spillere): (-0,(Ingenting), "j%a2k75,(Ingenting),test) Startværdier: (2,a,b),(3,a,b,c),(4,a,b,c,d)
Expected result	Spillet ville køre uden nogen fejl og med de rigtige startværdier
Actual result	Ingen fejl hændte og spillet kørte med de rigtige startværdier
Status	Success
Tested by	Tobias Kristensen
Date	25-11-2019
Test environment	IntelliJ IDEA 2019.2.3 (Ultimate Edition) Build #UI-192.6262.58 built on August 20, 2019 on Windows 10

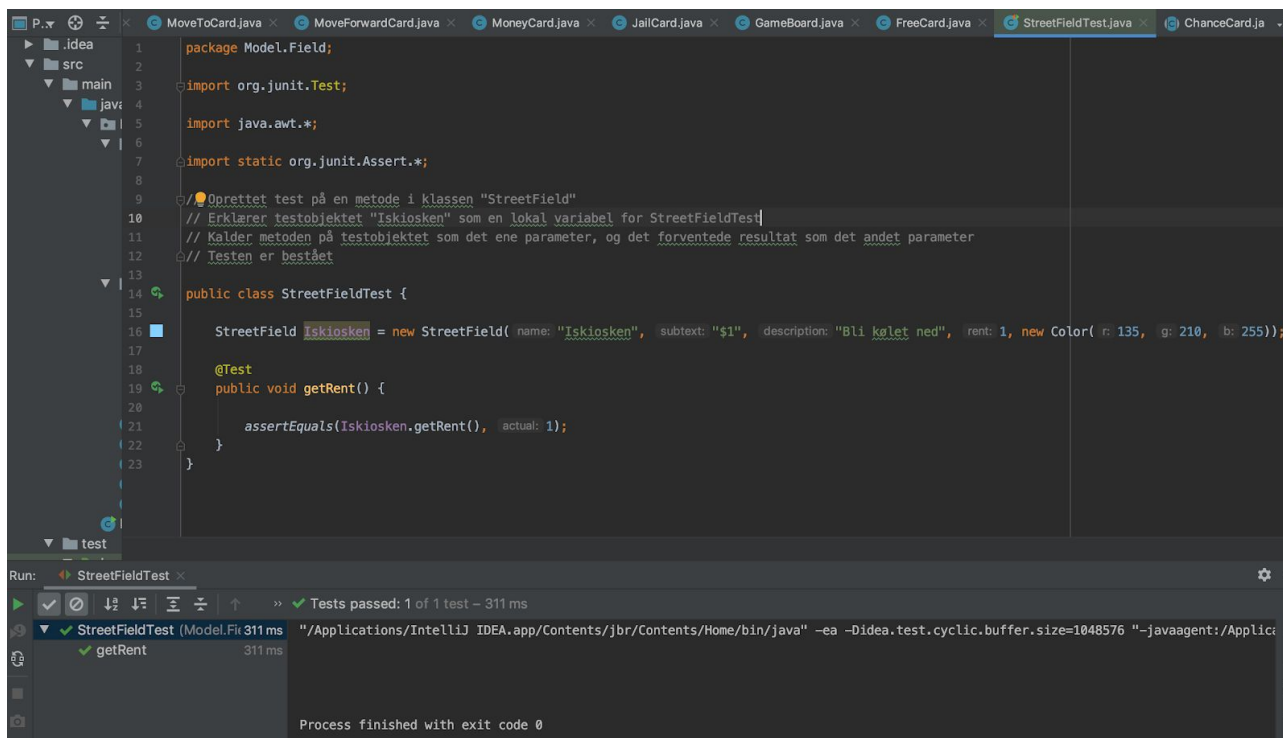
Test case ID	TC02
Summary	Test af logik
Requirements	<ul style="list-style-type: none">- Turen skifter hver gang en spiller har kastet de to terninger.- Spillet stopper når en spiller går fallit.
Preconditions	Spillere er indtastet og programmet er startet
Postconditions	Ingen fejl er opstået og spillet er enten færdigt eller stadig i gang

Test procedure	1. Tryk på kast terning og observer at turen skifter 2. Tryk på kast terning og observer spillet stopper hvis en spiller går fallit
Test data	Museklik
Expected result	Spillet ville køre uden nogen fejl
Actual result	Ingen fejl hændte og spillet kørte uden fejl
Status	Success
Tested by	Tobias Kristensen
Date	25-11-2019
Test environment	IntelliJ IDEA 2019.2.3 (Ultimate Edition) Build #UI-192.6262.58 built on August 20, 2019 on Windows 10

Test case ID	TC03
Summary	Test af interaktion med felter
Requirements	- Når spilleren lander eller passerer start modtages 1 monopoly dollar - Spiller mister penge ved køb af ejendom, afgift fra chancekort eller lande på andres ejendom.
Preconditions	Spillere er indtastet og programmet er startet
Postconditions	Ingen fejl er opstået og spillet er enten færdigt eller stadig igang
Test procedure	1. Tryk på kast terning og observer om penge modtages når start passeres 2. Tryk på kast terning og observer spillere mister penge når de skal
Test data	Museklik
Expected result	Spillet ville køre uden nogen fejl
Actual result	Ingen fejl hændte og spillet kørte uden fejl
Status	Success
Tested by	Tobias Kristensen
Date	25-11-2019
Test environment	IntelliJ IDEA 2019.2.3 (Ultimate Edition) Build #UI-192.6262.58 built on August 20, 2019 on Windows 10

5.2 JUnit test (Mina & Christian)

JUnit er en automatisk test, hvor man tester på metoder i en specifik klasser. Der oprettes en test klasser, hvori en testmetode konstrueres. I vores tilfælde ønskede vi at teste for korrekt leje på den pågældende grund. Det testmetode `getRent`, benyttede sig af `assertEquals`, hvor den benyttede sig af objektet ved navn `Iskiosken`, som havde lejen 1. Testen blev verificeret.



```
1 package Model.Field;
2
3 import org.junit.Test;
4
5 import java.awt.*;
6
7 import static org.junit.Assert.*;
8
9 // Oprettest test på en metode i klassen "StreetField"
10 // Erklærer testobjektet "Iskiosken" som en lokal variabel for StreetFieldTest
11 // Kaldet metoden på testobjektet som det ene parameter, og det forventede resultat som det andet parameter
12 // Testen er bestået
13
14 public class StreetFieldTest {
15
16     StreetField Iskiosken = new StreetField( name: "Iskiosken", subtext: "$1", description: "Bli kølet ned", rent: 1, new Color( r: 135, g: 210, b: 255));
17
18     @Test
19     public void getRent() {
20
21         assertEquals(Iskiosken.getRent(), 1);
22     }
23 }
```

Run: StreetFieldTest x

Tests passed: 1 of 1 test - 311 ms

StreetFieldTest (Model.Fik 311 ms) - /Applications/IntelliJ IDEA.app/Contents/jbr/Contents/Home/bin/java" -ea -Didea.test.cyclic.buffer.size=1048576 "-javaagent:/Appli

getRent 311 ms

Process finished with exit code 0

Vi synes, ikke vi havde andre relevante- eller bedre metoder at JUnit-teste, da `getRent` var en af de få metoder der returnerer noget.

Test Case ID	TC04
Referat	Automatiseret test af <code>getRent</code> .
Krav	Skal returnere værdien af rent på det pågældende felt (huslejen på feltet).
Betingelser før	Der findes ingen data om huslejen på det pågældende felt.
Betingelser efter	Huslejen på feltet er returneret.
Testens fremgangsmåde	<ul style="list-style-type: none">· Der oprettes en JUnit test case.· Testen køres i IntelliJ.
Test data	<code>getRent</code> på <code>Iskiosken</code> = 1
Forventet resultat	1
Faktisk resultat	1

Status	Godkendt
Testet af	Mina Ahmadi
Dato	26-11-2019
Testmiljø	IntelliJ IDEA 2019.2.1 på macOS Mojave vers. 10.14

5.3 Brugertest (Tobias og Sara)

Vi har valgt at lave en brugertest der tager udgangspunkt i svartiderne for vores program. Der er ikke så meget at teste omkring brugervenligheden da vi kun har en knap i spillet efter at spillerne har indtastet deres navne i konsollen.

Det mindst brugervenlige er når brugerne skal starte spillet og bruge konsollen da man ikke bare kører en .exe fil men kører programmet inde fra IntelliJ. Det gør at brugeren skal vide hvor man starter spillet og hvor man indtaster navnene. Vi antager dog at brugeren kan finde ud af disse to ting og herefter er der ikke nogen steder at brugeren kan komme på vildspor. Vi har også sørget for at brugeren ikke kan skrive forkerte navne ind og at det ikke kan forårsage fejl i spillet i vores test-cases.

Under observation af en testperson, som ingen kendskab har til at kode, indmaden eller meningen med projektet, blev følgende spørgsmål besvaret.

Nedenfor er en oversigt over de stillede spørgsmål og dertilhørende svar:

Spørgsmål:	Svar:
Hvad synes du om programmets svartider?	De synes at være meget normale. Dog går der noget tid fra man har tastet navne ind til spillepladen kommer frem, hvilket jeg blev lidt forvirret over.
Synes du det kører hurtigst på spillepladen eller i konsollen?	Konsollen synes at svare hurtigst, men det var sjovere at være inde på spillepladen.
Ville du fravælge spillet pga. svartiderne?	Nej det ville jeg bestemt ikke. Så lange svartider er der slet ikke.

Flere reaktioner fra testpersonen under spillets gang:

- En smule ensformigt og kedeligt i længden.
- Ærgerligt at man ikke kan se hvem der ejer hvad, og på den måde bytte rundt, købe osv.
- Mange chancekort funktioner, som gør spillet en smule sjovere.

5.4 Databar test (Sara)

Eftersom en af kravspecifikationerne er, at programmet skal kunne køre på en computerne på DTU's databarer, er det testet hvorvidt dette er opfyldt. For at tjekke for det, er følgende steps udført:

1. Installer programmet gennem Github eller ved brug af USB stik.
2. Eksekver programmet med filnavnet 15_DEL3.JAR
3. Spillet startes op, hvorefter spilleren indtaster antal spillere og navne
4. Spillet kan nu spilles

Gennemføres de 4 steps, så kan det ud fra databar testen konkluderes, at kravspecifikationen er opfyldt.

6. Projektplanlægning

6.1 Tidsplan (Tobias)

Tidsplanen er lavet med udgangspunkt i at lave alt det forberedende med krav og analyse og så bevæge sig hen imod designet og koden. Essentielt følge Unified process så meget som muligt. Det vises og er beskrevet i projektførløbet hvor meget vi har fulgt den her tidsplan og hvor meget vi har fulgt unified process. Tidsplanen lavede vi ret tidligt så vi har ikke fulgt den til punkt og prikke men fokuseret mere på at lave en fyldestgørende projekt og ikke læne os op af den hvis den ikke passede ind med vores arbejdsprocess og arbejdsfordeling.

Aktivitet	Underaktivitet	11/11	12/11	13/11	14/11	15/11	16/11	17/11	18/11	19/11	20/11	21/11	22/11	23/11	24/11	25/11	26/11	27/11	28/11	29/11
Indledning	Abstract																			
	Indledning																			
Krav og analyse	Kravliste																			
	Use Case diagram																			
	Domæne diagram																			
	Systemsekvensdiagram																			
Design	Designklassediagram																			
	Sekvensdiagram																			
	GRASP-mønstre																			
Implementering	Kode eksempel																			
Test	Testcases																			
	JUnit test																			
	Brugertest																			
Konklusion																				
Kode skrivning	GUI																			
	Klasser																			
	UI																			
	Konfiguration																			

6.2 Projektførløb (Sara)

Da vi denne gang har haft længere tid til opgaven, gjorde vi det ikke klart fra dagen vi fik opgaven om hvornår vi skulle mødes os. Henover den første weekend, aftalte vi at pigerne kiggede på koden, mens drengene kiggede på diagrammerne. Vi fik en dag fri fra vores undervisningen, og brugte denne sammen til at sidde og lave på koden og opgaven. Hertil fik vi koden færdig en af de efterfølgende dage, så vi kunne gå i gang med at lave selve rapporten. Vi opdelte nogle af diagrammerne og test, og efter weekenden aftalte vi at mødes så vi kunne hjælpe hinanden i mål, læse og se hvad vi havde lavet, og samle rapporten til en helhed. Alt dette har også været med henblik på unified process og vi har prøvet at følge den process eller have den i baghovedet i løbet af projektet. Dette kan ses under bilag 2 der viser et diagram over projektførløbet og den tid vi har brugt på de forskellige dele af projektet

7. Konfigurationsstyring (Oliver)

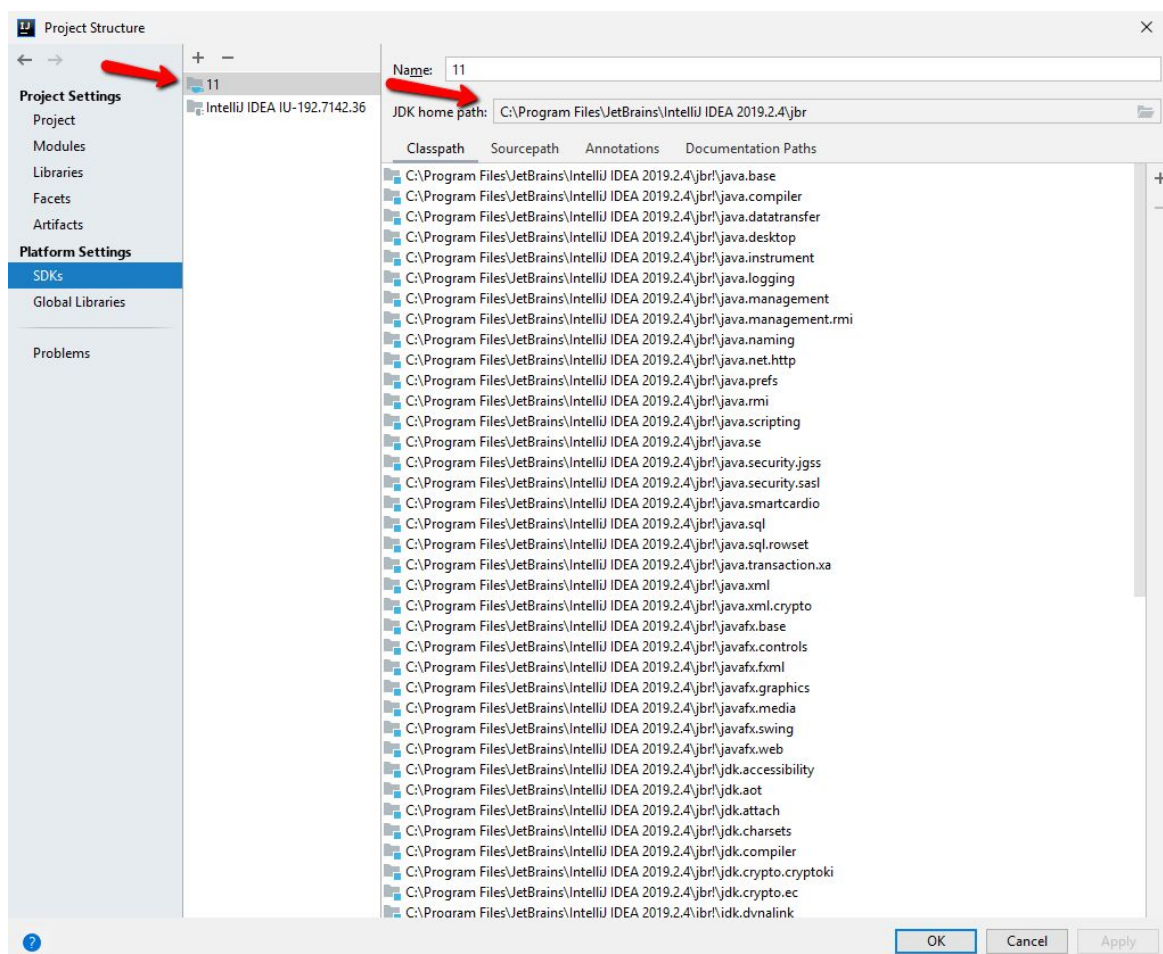
7.1 Udviklingsmiljø

Før at hente og køre projektet skal du bruge.

- IntelliJ IDEA 2019.2.4 (Ultimate edition)
- Java SDK 11.0.0
- Git

På IntelliJ's startside vælges “Check out from Version Control” -> “Git” og skriv

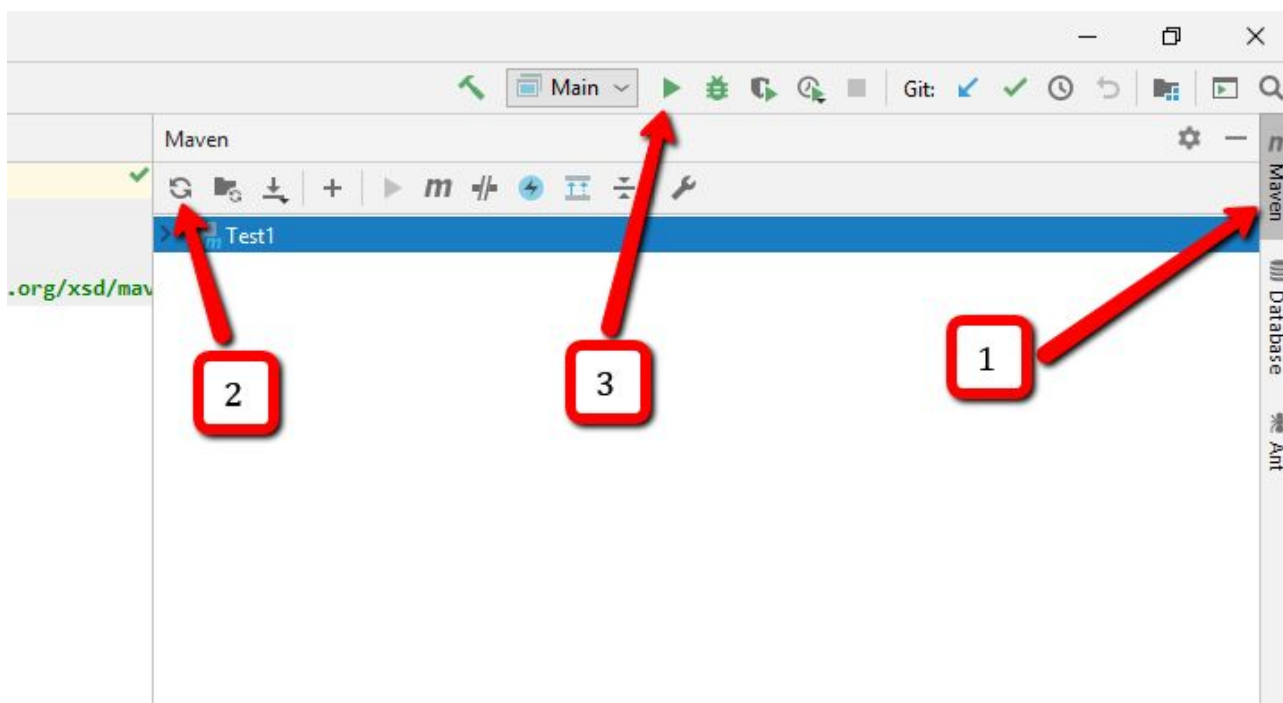
https://github.com/sarakarkov/15_del3 i URL og hent. Når programmet er åben så vælges File -> Project Structure. I menuen vælges SDKs.



Det er vigtigt at Name er præcist 11 og det peger på en SDK der er version 11.

Klik Ok.

Før du kører programmet skal pom.xml hente det eksterne bibliotek matadorgui:3.1.1. Dette hentes ved hjælp af Maven. Maven er et framework til at hente eksterne biblioteker og det kan også bruges til at udgive egne biblioteker.



Klik på Maven oppe i højre side af IntelliJ (1) og vælg derefter “Sync” (2) vist på billedet. Nu burde matadorgui:3.1.1 hentes til projektet og efterfølgende kan det køres ved at klikke på den grønne pil (3).

7.2 Produktionsmiljø

For at køre programmet i dette miljø skal du blot hente java

- Java 11

Kør .java filen og programmet kører.

7.3 Versionering

Under udviklingen af programmet har vi benyttet version styring, herunder Git, til at udvikle produktet. Vores repository ligger som nævnt tidligere her https://github.com/sarakarkov/15_de13.

Programmet kan hentes til IntelliJ på flere forskellige måder.

- 1) Det kan hentes på IntelliJ startside ved at klikke på Check out from Version Control -> Git
- 2) Hvis man har et andet program kører kan man klikke oppe i top menuen VCS -> Check out from Version Control -> Git
- 3) Det kan hentes ved at åbne en command prompt, cmd, og skrive `git clone https://USERNAME:PASSWORD@github.com/sarakarkov/15_de13`
Erstat USERNAME med brugernavn og PASSWORD med kodeord.

Alle disse kræver at Git er installeret.

8. Konklusion (Mina)

Under arbejdet af denne rapport samt vores projekt, har vi gjort brug af metoder fra de tre kurser vi har haft i dette semester, samt taget forbehold for de fejl og misforståelser der kan opstå under implementering af et projekt. Ved brug af GitHub har vi oprettet et fælles repository, hvor alle gruppens medlemmer har haft mulighed for at versionere og holde sig opdateret på ændringer i koden i IntelliJ samt ændringer i diagrammerne i MagicDraw. Vi har fået opfyldt kravene vi fik stillet af kunden, og er nået til ende med de hensigter vi havde ved projektets start.

Indsatsen og samarbejdet i gruppen har fungeret godt, og vi har gentagende gange lagt tid til at vi kan mødes og arbejde på projektet i fællesskab, samt få hjælp på tværs af hinanden. Vi har denne gang oprettet diagrammer og modeller før vi er gået i gang med kodningen af projektet, hvilket har fungeret meget godt.

Vi kan konkludere, at vi vha. et Github Repository, diagrammer, modeller, en fuldendt rapport samt et kodet projekt, har fået kodet et junior matador spil som opfylder kundens vision og forespørgsler.

9. Bilag

Bilag 1: chancekort

Gule er ikke inkluderet i vores spil men er en del af det originale matador junior

CHANCE 1:

Giv dette kort til **BILEN**, og tag et chancekort mere.

BIL: på din næste tur skal du drøne frem til et hvilket som helst ledigt felt og købet det.

Hvis der ikke er nogen ledige felter skal du købe et fra en anden spiller!

CHANCE 2:

Ryk frem til start Modtag M2

CHANCE 3:

Ryk op til 5 felter frem.

CHANCE 4:

GRATIS FELT! Ryk frem til et **orange** felt.

Hvis det er ledigt, får du det **GRATIS!** Ellers skal du **BETALE** leje til ejeren.

CHANCE 5:

Ryk 1 felt frem eller tag et chancekort mere

CHANCE 6:

Giv dette kort til **SKIBET**, og tag et chancekort mere.

SKIB: På din **næste tur** skal du sejle frem til et **hvilket som helst** ledigt felt og købe det.

Hvis der ikke er nogen ledige felter skal du købe et fra en anden spiller!

CHANCE 7:

Du har spist for meget slik. **BETAL** M2 til banken

CHANCE 8:

GRATIS FELT! Ryk frem til et **orange** eller **grønt** felt. Hvis det er ledigt, får du det

GRATIS! Ellers skal du **BETALE** leje til ejeren.

CHANCE 9:

GRATIS FELT! Ryk frem til et **lyseblåt** felt.

Hvis det er ledigt, får du det **GRATIS!** Ellers skal du **BETALE** leje til ejeren.

CHANCE 10:

Du løslades uden omkostninger Behold dette kort, indtil du får brug for det

CHANCE 11:

Ryk frem til **Strandpromenaden**

CHANCE 12:

Giv dette kort til **KATTEN** og tag et chancekort mere.

KAT: på din **næste tur** skal du liste dig hen på et **hvilket som helst** ledigt felt og købe det.

Hvis der ikke er nogen ledige felter, skal du købe et fra en anden spiller

CHANCE 13:

Giv dette kort til **HUNDEN**, og tag et chancekort mere.

HUND: På din **næste tur** skal du hoppe hen på et **hvilket som helst** ledigt felt og købe det.

Hvis der ikke er nogen ledige felter, skal du købe et fra en anden spiller.

CHANCE 14:

Det er din fødselsdag! Alle giver dig M1

TILLYKKE MED FØDSELSDAGEN!

CHANCE 15:

GRATIS FELT! Ryk frem til et **pink** eller **mørkeblåt** felt.

Hvis det er ledigt, får du det **GRATIS!** Ellers skal du **BETALE** leje til ejeren.

CHANCE 16:

Du har lavet alle dine lektier **MODTAG** M3 fra banken

CHANCE 17:

GRATIS FELT! Ryk frem til et **rødt** felt.

Hvis det er ledigt får du det **GRATIS** Ellers skal du **BETALE** leje til ejeren

CHANCE 18: GRATIS FELT!

Ryk frem til skaterparken for at lave det perfekte grind! Hvis ingen ejer den får du den gratis ellers skal du betale ejeren.

Bilag 2: Testcases Data

TC1: Test for initialisering af spil

TC1.1:

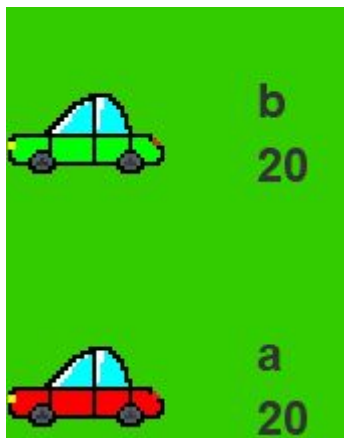
```
Indtast antal spillere mellem 2 og 4:  
1  
Indtast antal spillere mellem 2 og 4:  
-5  
Indtast antal spillere mellem 2 og 4:  
test  
Indtast antal spillere mellem 2 og 4:  
"j%a2k75  
Indtast antal spillere mellem 2 og 4:  
65413959325021391359  
Indtast antal spillere mellem 2 og 4:  
3  
Indtast spiller nr. 1
```

TC1.2:

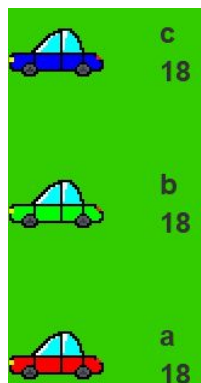
```
Indtast antal spillere mellem 2 og 4:  
4  
Indtast spiller nr. 1  
-0  
Indtast spiller nr. 2  
  
Indtast spiller nr. 3  
"j%a2k75  
Indtast spiller nr. 4  
  
Navnet eksisterer allerede.  
Indtast spiller nr. 4  
test  
Spillet begynder. GUI'en bliver åbnet.
```

TC1.3:

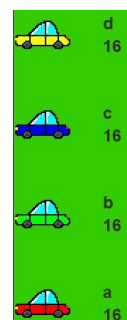
```
Indtast antal spillere mellem 2 og 4:  
2  
Indtast spiller nr. 1  
a  
Indtast spiller nr. 2  
b  
Spillet begynder. GUI'en bliver åbnet.
```



```
Indtast antal spillere mellem 2 og 4:  
3  
Indtast spiller nr. 1  
a  
Indtast spiller nr. 2  
b  
Indtast spiller nr. 3  
c  
Spillet begynder. GUI'en bliver åbnet.
```

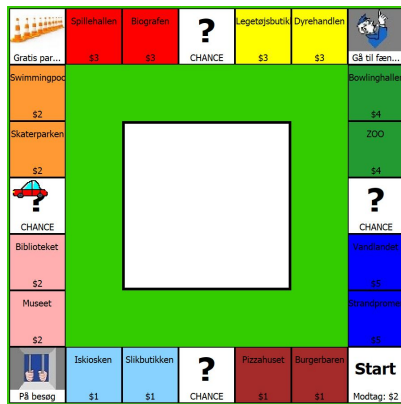


```
Indtast antal spillere mellem 2 og 4:  
4  
Indtast spiller nr. 1  
a  
Indtast spiller nr. 2  
b  
Indtast spiller nr. 3  
c  
Indtast spiller nr. 4  
d  
Spillet begynder. GUI'en bliver åbnet.
```

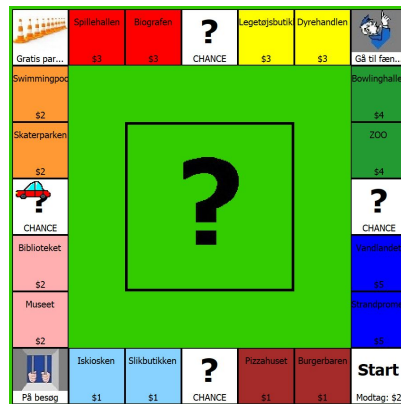


TC2: Test af logik

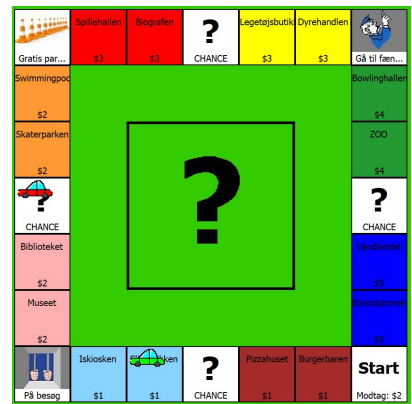
TC2.1



Spiller a's tur er slut

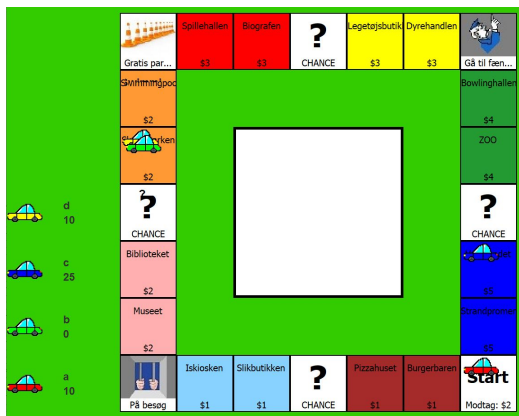


Nu er det b's tur

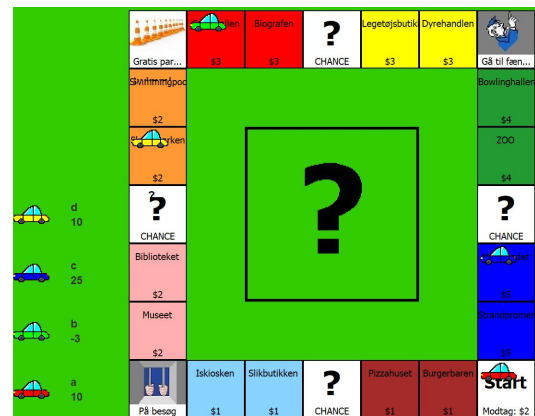


Spiller b landede på slikbutikken og købte den

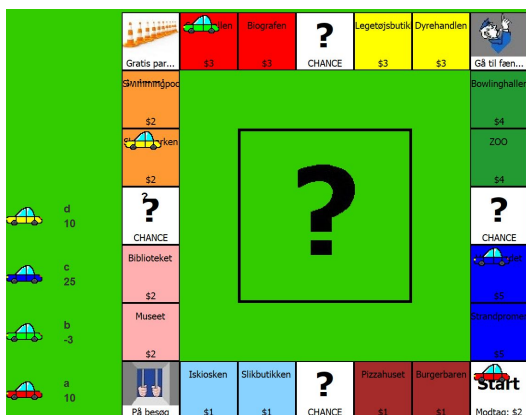
TC2.2



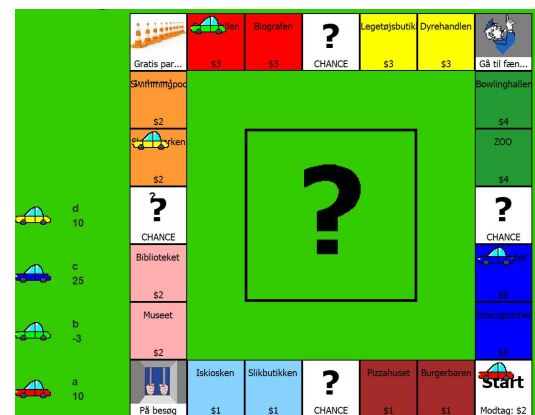
Nu det spiller b's tur



Spiller b landede på spillehallen og købte den



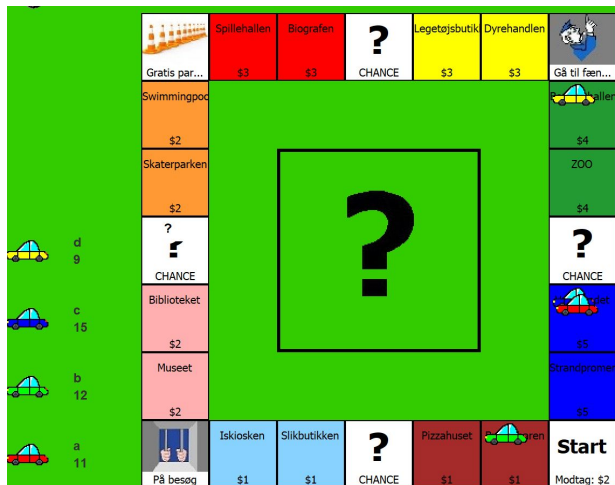
Spillet slutter da b har tabt. vinderen, den rigeste spiller, er a



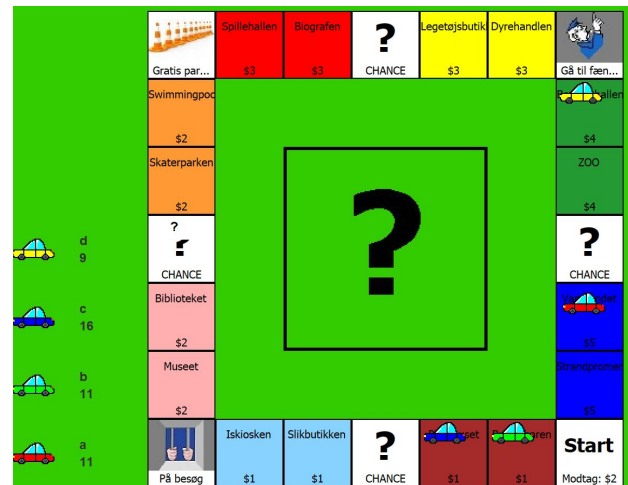
spiller b's tur er slut

TC3: Test af interaktion med felter

TC3.1

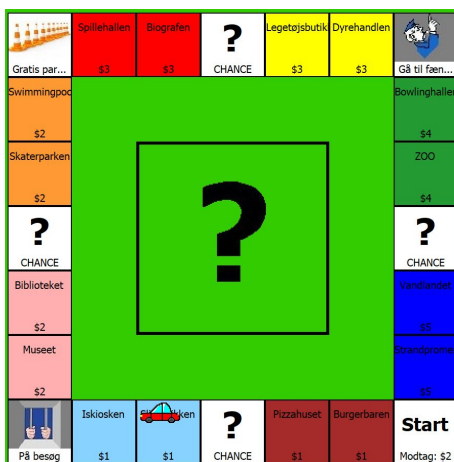


Nu er det Spiller c's tur

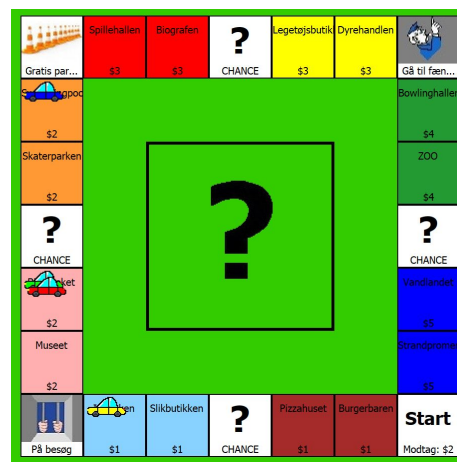


Spiller c passerer start og modtager \$1

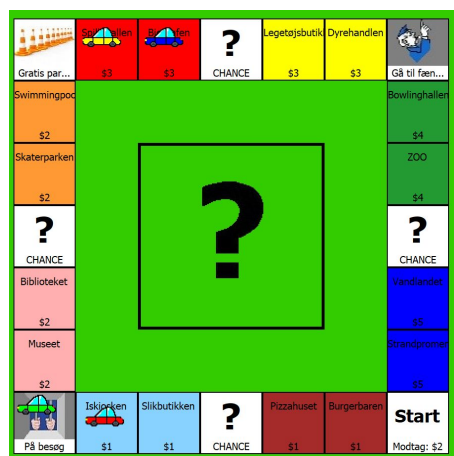
TC3.2



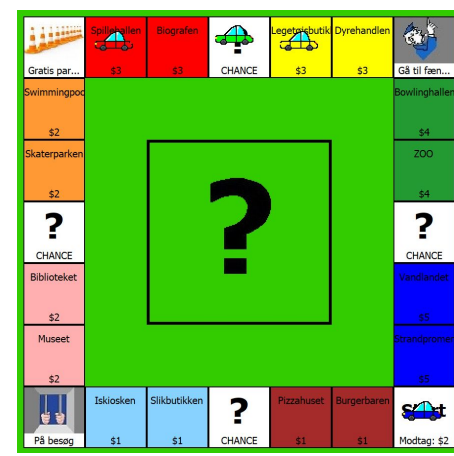
Spiller a landede på slikbutikken og købte den og mistede \$1



spiller a landede på biblioteket og betalte leje til b



Spiller b ryger i fængsel uden at modtage penge og betaler 1\$ næste runde



Spiller b har fået penge fra de andre spillere

Bilag 3: Unified process for timeregnskab

