Indledende programmering (02312) – Udviklingsmetoder til IT-systemer (62531) – Versionsstyring og testmetoder (62532)

CDIO DELOPGAVE 3

Gruppe 23

Tobias Henriksen - s211615@student.dtu.dk

Christiaan Vink - s215832@student.dtu.dk

Espen Ulff-Møller - s215831@student.dtu.dk

Jacob Pilegaard Justesen - s164958@student.dtu.dk

Afleveringsfrist: 26.11.2021

Timeregnskab

Tid (oprundet i timer)	Krav og analyse	Design	Implementering	Test	Programmering
Tobias	3	1	0	0	3
Christiaan	8	1	1	2	1
Espen	4	3	3	2	13
Jacob	0	0	0	0	0

Resumé

This report revolves around the project of creating a digital Monopoly junior. The report consists of 5 main chapters, analysis, Design, Implementation, Test, and Configuration in which we have developed given artifacts recommended by the project leader.

Indholdsfortegnelse

1. Indledning

Der er endnu en opgave fra kunden som vi, IOOuterActive, skal laves. Denne her gang handler det om spillet 'Monopoly Junior'. Kunden har haft det fornemmelse at denne opgave er større end de andre, og derfor har de givet os mulighed for at undlade nogle af reglerne og prioritere de vigtige funktioner af spillet, hvis det bliver for meget arbejde. Ligesom sidste gang har vi i starten af rapporten analyseret de krav og den vision som er givet af kunden til vores program. Fra det har vi udarbejdet forskellige diagrammer og modeller som hjælper os til at visualisere processen. Udover det har vi også taget hensyn til GRASP-mønstret i vores udvikling af projektet.

2. Krav

For at lave programmet som kunden ønsker sig, har vi opstillet et antal krav, hvorefter vi har inddelt disse krave i funktionelle og non-funktionelle krav. Det gør vi for at specificere de vigtigste elementer af visionen af kunden. Herefter har vi lavet en prioriteringsliste af kravene ud fra MoSCow metoden.

2.1 Kravspecificering

K1: Spillet slutter når 1 af spillerne ikke har flere penge tilbage.
K2: Der skal være en spilleplade med sine særlige felter.
K3: Der skal være 'Chance' kort som har forskellige funktioner når de er trykket.
K4: 'Chance' kortene har deres plads på spilleplade.
K5: Der skal være billetluger til hver spiller.
K6: Hver spiller har en pengebeholdning.
K7: Hver spiller starter med 5x $1, 4x $2, 3x $3, 2x $4 og 1x $5 = $35
K8: Systemet får rolle som 'banken' og skal handler penge handlinger gennem spillet.
K9: Hver spiller skal vælge en farve, til deres brik.
K10: Man starter på 'Gå!' med ens brik.
K11: Hver brik er en bil.
K12: Spillet forløber ved at tage tur, og flyt ens bil det antal felter som terningen viser.
K13: Spiller som kaster det højeste i starten begynder.
K14: Der skal være felter, som når man lander på dem, så skal man betale banken derefter ejer man felten.
K15: Hvis man lander på en felt, ejet af en spiller, betale man spiller.
K16: Man betale dobbelt hvis spilleren ejer flere af det samme felt.
K17: Der skal være et sted på spilleplade hvor man lægger 'Små penge'.
K18: Feltet 'Toiletter' er et neutralt felt.

K19: Ved feltet 'Tage til toiletter' betaler man $3 til 'Små penge' og flytter bilen til 'Toiletter' feltet.
K20: Det skal være en felt på den modsatte side af GO, hvor man betaler $2 til 'Små penge'.
K21: Hvis man går forbi 'Gå!' Feltet får man $2 fra banken.
K22: Der skal være en 'Gå!' Felt.
K23: Der skal være et felt 'Jernbane', som lad spilleren rulle igen.
K24: Der skal være et felt 'Jackpot!' som give spiller der lande på det alt nuværende 'Små Penge' på bordet.
K25: Skal være nemt at oversætte til andres sprog.
K26: Skal anvendes GUI til spillet

Uspecificeret krav

UK1: Hvor mange af hver felt skal der være, er der en maksimum, minimum?

UK2: Hvor mange Chancekort skal der være?

UK3: Er der et særligt antal billetluger?

UK4: Kan banken løbe tør for penge?

UK5: Hvilke spiller er efter spiller der starter?

UK6: Hvad for en terning rulle man med?

UK7: Hvis der er 2+ spiller, hvem vinder så hvis spillet er slut?

2.1.1 Funktionelle krav

K1: Spillet slutter når 1 af spillerne har ikke flere penge tilbage.

K5: Der skal være billetluger til hver spiller.

K6: Hver spiller har en pengebeholdning.

K7: Hver spiller starter med 5x $1, 4x $2, 3x $3, 2x $4 og 1x $5

K8: 1 spiller får rolle som 'banken' og skal handler penge handlinger gennem spillet.

K9: Hver spiller skal vælge en farve, til deres brik.

K10: Man starter på 'Gå!' med ens brik.

K12: Spillet forløber ved at tage tur, og flyt ens bil det antal felter som terningen viser.

K14: Der skal være felter, som når man lander på dem, så skal man betale banken derefter ejer man felten.

K15: Hvis man lander på en felt, ejet af en spiller, betale man spiller.

K22: Der skal være en 'Gå!' Felt.

2.1.2 Non-Funktionelle krav

K3: Der skal være 'Chance' kort som har forskellige funktioner når de er trykket.

K2: Der skal være en spilleplade med sine særlige felter.

K4: 'Chance' kortene har deres plads på spilleplade.

K11: Hver brik er en bil.

K13: Spiller som kaster det højeste i starten begynder.

K16: Man betale dobbelt hvis spilleren ejer flere af det samme felt.

K17: Der skal være et sted på spilleplade hvor man lægger 'Små penge'.

K18: Feltet 'Toiletter' er et neutralt felt.

K19: Ved feltet 'Tage til toiletter' betaler man $3 til 'Små penge' og flytter bilen til 'Toiletter' feltet.

K20: Det skal være en felt på den modsatte side af GO, hvor man betaler $2 til 'Små penge'.

K21: Hvis man går forbi 'Gå!' Feltet får man $2 fra banken.

K23: Der skal være et felt 'Jernbane', som lad spilleren rulle igen.

K24: Der skal være et felt 'Jackpot!' som give spiller der lande på det alt nuværende 'Små Penge' på bordet.

K25: Skal være nemt at oversætte til andres sprog.

2.2 Prioritering af krav - MoSCoW

Must have:

K1: Spillet slutter når 1 af spillerne har ikke flere penge tilbage.
K5: Der skal være billetluger til hver spiller.
K6: Hver spiller har en pengebeholdning.
K7: Hver spiller starter med 5x $1, 4x $2, 3x $3, 2x $4 og 1x $5
K8: 1 spiller får rolle som 'banken' og skal handler penge handlinger gennem spillet.
K9: Hver spiller skal vælge en farve, til deres brik.
K10: Man starter på 'Gå!' med ens brik.
K12: Spillet forløber ved at tage tur, og flyt ens bil det antal felter som terningen viser.
K14: Der skal være felter, som når man lander på dem, så skal man betale banken derefter ejer man felten.
K15: Hvis man lander på en felt, ejet af en spiller, betale man spiller.
K22: Der skal være en 'Gå!' Felt.
K26: Skal anvendes GUI til spillet

Should have:

K3: Der skal være 'Chance' kort som har forskellige funktioner når de er trykket.
K2: Der skal være en spilleplade med sine særlige felter.
K13: Spiller som kaster det højeste i starten begynder.
K16: Man betale dobbelt hvis spilleren ejer flere af det samme felt.
K20: Det skal være en felt på den modsatte side af GO, hvor man betaler $2 til 'Små penge'.

Could have:

K18: Feltet 'Toiletter' er et neutralt felt.
K19: Ved feltet 'Tage til toiletter' betaler man $3 til 'Små penge' og flytter bilen til 'Toiletter' feltet.
K21: Hvis man går forbi 'Gå!' Feltet får man $2 fra banken.
K23: Der skal være et felt 'Jernbane', som lad spilleren rulle igen.
K24: Der skal være et felt 'Jackpot!' som give spiller der lande på det alt nuværende 'Små Penge' på bordet.

Would be nice to have:

K11: Hver brik er en bil.

K17: Der skal være et sted på spilleplade hvor man lægger 'Små penge'.

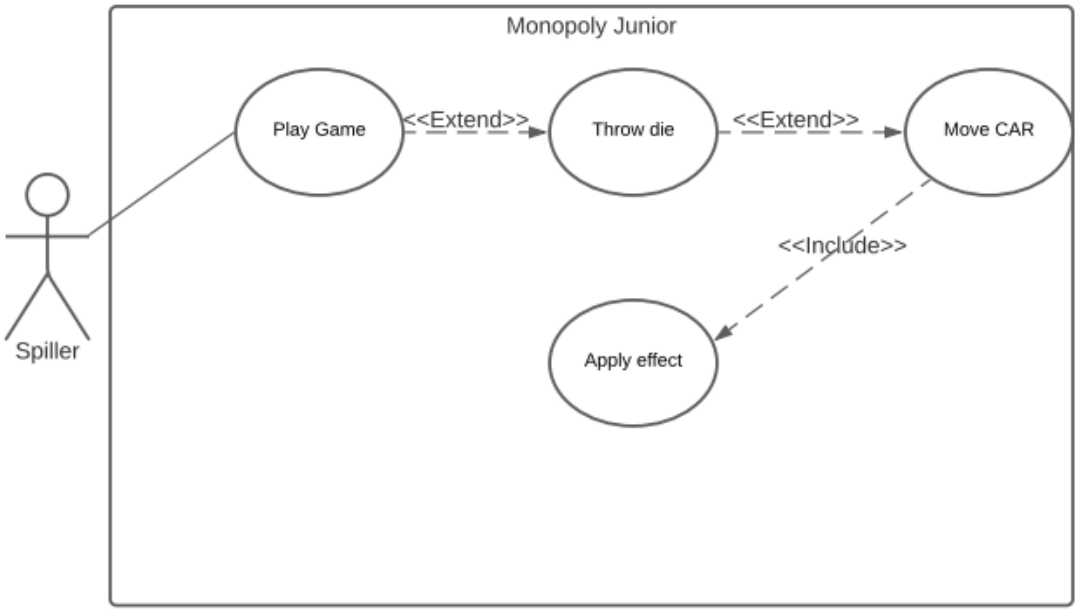
K4: 'Chance' kortene har deres plads på spilleplade.

K25: Skal være nemt at oversætte til andres sprog.

3. Analyse

3.1 Use case Diagram

I vores use case beskrivelse, viser vi hvordan spiller interagerer med funktionen 'Play round'. Derudover illustreres det også samspillet mellem disse klasser. Vi har valgt at lad Bank være en del af systemet og ikke en af spillerne, derfor er Bank ikke en Aktør, men en funktion. Spiller pleje også at har 'Association' med 'Balance' og 'Banken' men fordi spillet er digitalt, og banken er en aktør, har vi valgt at det er nogle som vil foregå automatisk.



3.2 Use case beskrivelse

Use case Diagrammet har forskellige uses cases som man kan beskriver, vi har valgt at beskrive dem som vil har mest indflydelse på det Monopoly Junior program. Dem som har kommet i betragtning til det er: Field effect, Play round og Balance.

3.2.1 Fully dressed case beskrivelse

UC1

Navn på Use case	Apply effect
Scope	Hvert felt har en effekt, som gør at spillers brik skal rykke sig eller at spillerne får/taber nogle er deres penge.
Level	User level Goal
Primary Aktør	Spiller
Preconditions	Effekt fra et felt Spiller Terning

	Pengebeholdning (balance)
Stakeholder and interest	**Spiller** skal kunne se hvad for en effekt det felt de lander på har på deres 'balance' eller brikkens position. **Move Field** det kræves at "Move Field" fungerer for at "Field Effect" skal kunne fungere.
Postconditions	Spiller vil blive tildelt feltets effekt.
Scenarie	Der bliver kastet en terning og Spillers brik vil rykke til det tildelte felt. Der vil der være en effekt som kan gøre at spiller skal rykke sin brik igen eller at de skal give penge til banken, en anden spiller eller skal lægge det i 'små penge' bunken. Der er også den mulighed at spiller får penge af banken eller fra 'Små penge' bunken.
Basic Flow	1. Der er 24 antal felter, med egen navn og effect. 2. Feltet som siger 'Gå' er stedet hvor spillerne vil starte. Dermed har det også den effekt at når man kommer forbi felten så få man $2. 3. Felten med 'Jernbane' vil lade spilleren give mulighed for at kaste igen. 4. Felten 'Tage til toiletter' har den effekt at man skal give $3 til 'Små penge' beholdning. Man bliver også send til felten 'toilet'. 5. Der er 24 felter som er spillepladens 'forlystelser', for disse felter skal man betale et beløb til banken, herefter ejer man det. Hvis man lander på det felt mens det er ejet af en anden spiller, så skal man betale det beløb man har købt det for til ejeren. 6. Der er et felt på den modsatte side af 'Gå!' hvor man skal betaler $2. 7. Felter Jackpot har den effekt at man får alt nuværende $ der ligger på 'Små penge'

3.2.2 Brief Use Use case beskrivelse

UC2

Navn på Use case	Play Game
Scope	Spille en runde
Level	User level Goal
Primary Aktør	Spiller

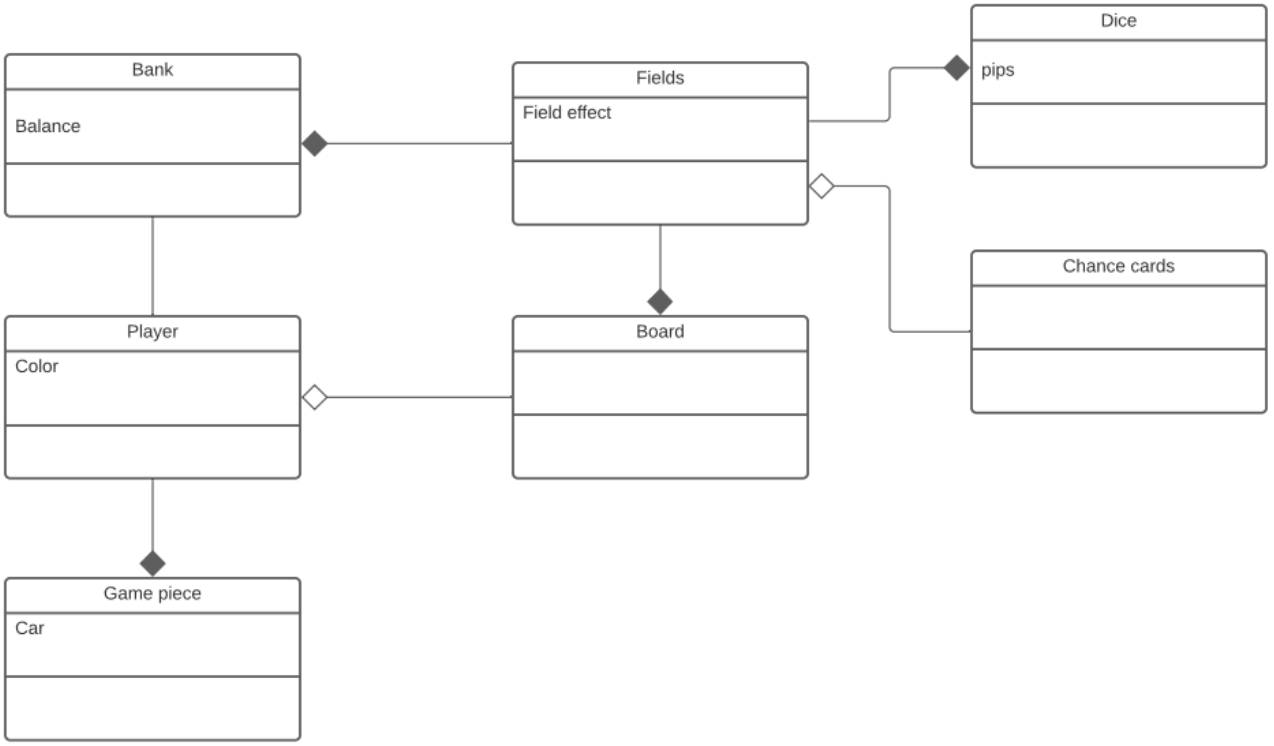
Preconditions	mindst 2 spillerne 1 terning første 'play round' starter alle spillerne med 5x $1, 4x $2, 3x $3, 2x $4 og 1x $5
Postconditions	En af spillerne har tabt alt sin penge
Scenarie	Slå terning, ryk felt, få penge, tab penge og få ejendom til spillepladens seværdigheder
Basic Flow	1. Spiller der begynder kaster en terning. 2. Antal givne øje på terning vil flytte spillers brik det antal felter. 3. Feltet vil har et effekt som vil ændre spillers pengebeholdning eller/og flytte deres brik. 4. Det næste spiller vil nu gøre det sammen 5. Man gentager 'Play round' indtil en af spillerne har ikke flere $ tilbage.

3.3 Domænemodel

Domænemodel til Junior Monopoly spillet illustrerer de forskellige klasser samt hvordan de spiller sammen. Denne domænemodel er udformet på baggrund af vores Use Case og indeholder de klasser som er nødvendige for et fungerende system.

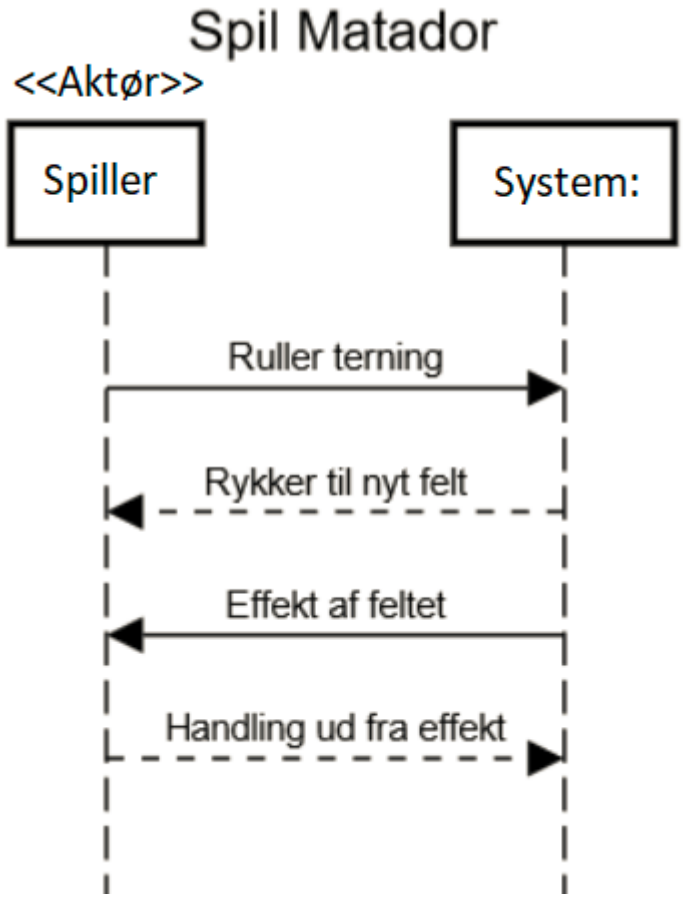
klasser i domænemodellen:

- Bank
- Fields
- Dice
- Chance Card
- Board
- Player
- Game Piece



3.4 Systemsekvensdiagram

System Sekvensdiagrammet beskriver adfærden mellem aktør (brugeren) og system. adfærden mellem bruger og system bliver beskrevet med en lifeline som beskriver objekternes levetid. derfor bliver der beskrevet en tidslinje for interaktionerne mellem de to. Dette gøres for at give os udvikler en hurtig forståelse for den information og metoder som skal kaldes på.

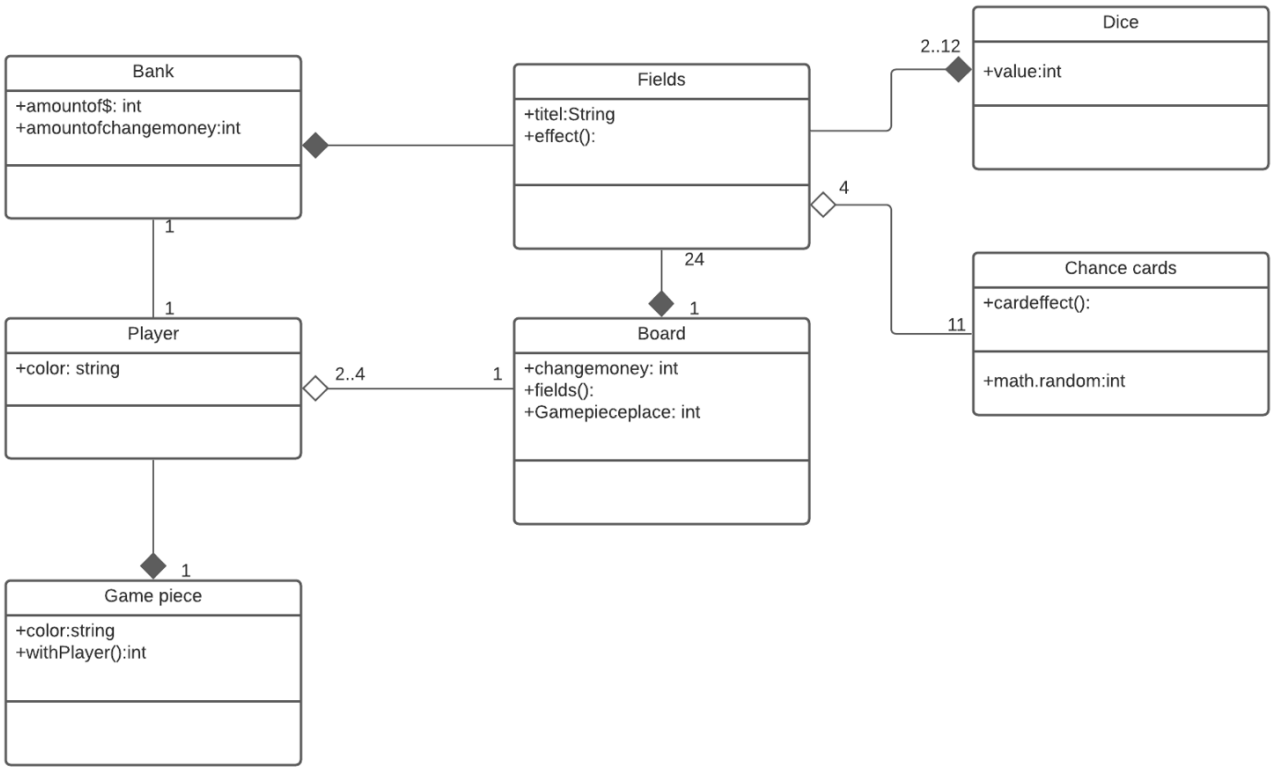

Spil Matador
<<Aktør>>
Spiller
System:
Ruller terning
Rykker til nyt felt
Effekt af feltet
Handling ud fra effekt

4. Design

I dette kapitel er der blever moduleret et design klassediagram for en forståelse for systemets opbygning på en fagsproglig og detaljeret baggrund. Derudover er der også blevet udarbejdet et sekvensdiagram til beskrivelse af samspillet mellem de forskellige klasser for at de udfører deres funktionalitet.

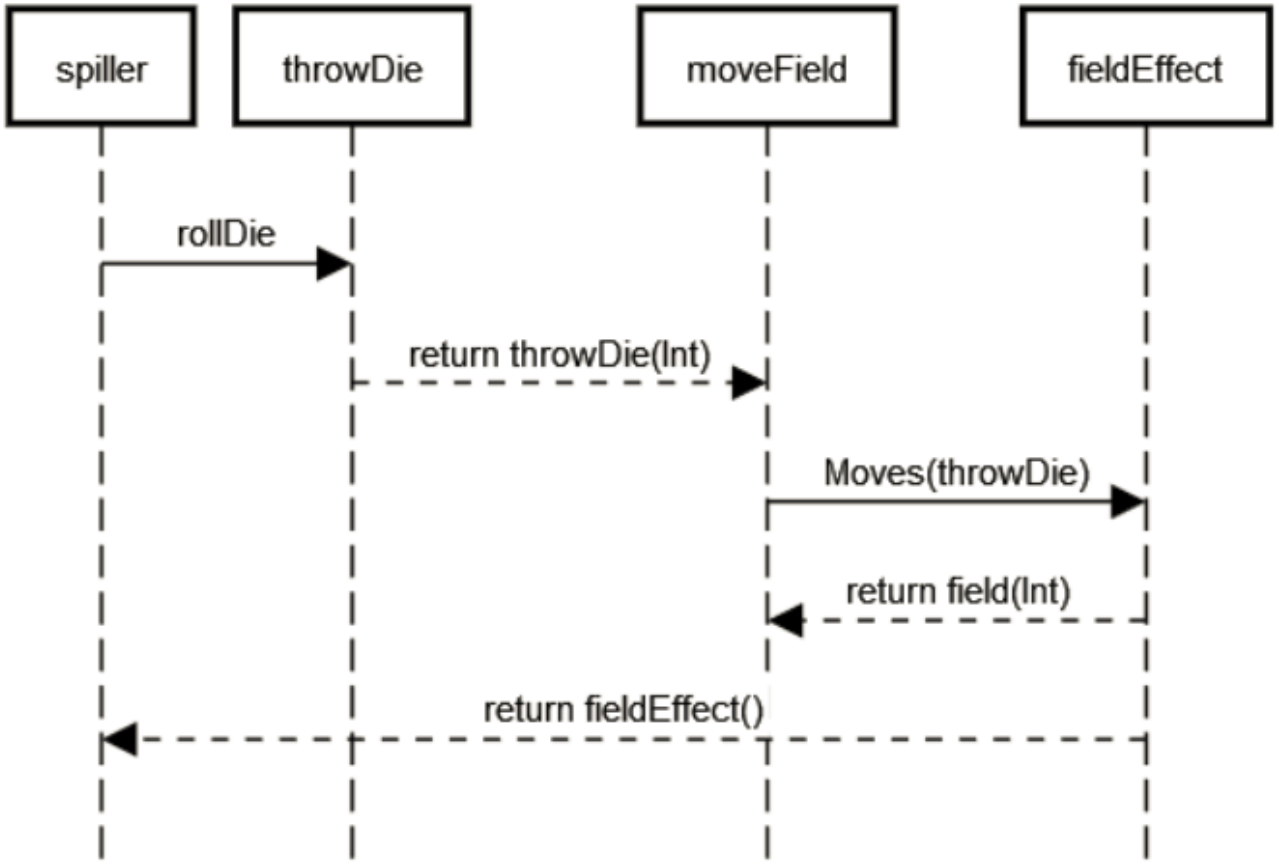
4.1 Design Klassediagram

Dette diagram giver en forståelse og overblik af systemets sammensætning og implementering. mellem hver klasse bliver beskrevet deres association samt deres attributter og operationer. Dette gøres for at implementeringsfasen nemmest gøres og for at udviklingsteamet får en dybere forståelse for programmets opbygning.



4.2 Sekvensdiagram

Sekvensdiagrammet beskriver adfærden i rækkefølge mellem de forskellige klasser over tid. Dette diagram er ikke fyldestgørende for det endelige produkt idet det kun beskriver det man kan kalde "bevægelse" hvori **Bank klassen** mangler for at kunne beskrive den endelige adfærd mellem klasserne.



5. Implementering

Her tager vi vores spiller klasse som vores primære eksempel.

- ☑ ~~Lav passende konstruktører.~~
  Her har vi et eksempel på en konstruktør. Den har to konstruktions metoder for at gøre det nemmere at teste spillere til fremtidige tests.

```java
public class Player {
    public final String name;
    private int money = Config.START_MONEY;

    private int prevPosition = 0;
    private int position = 0;
    public boolean inJail = false;

    public boolean outOfJailFree = false;
    public int jailTime = 0;

    public boolean bankrupt = false;


    public Player(String name) {
        this.name = name;
    }

    public Player(String name, int money) {
        this.name = name;
        this.money = money;
    }
```

- [x] ~~Lav passende get og set metoder.~~

  Her har vi en håndfuld eksempler på getter/setters. De er sorteret så de funktioner der arbejder med det samme står sammen. Vi har valgt kun at have en getMoney, da vi kun har brug for at tjekke værdien f.eks for at undgå spilleren går i minus når de køber boder, skal se om spillet er slut eller hvem der har vundet.

```java
public int getMoney() { return money; }

public void addMoney(int value) { this.money += value; }

public int getPrevPosition() { return prevPosition; }

public int getPosition() { return position; }

public void setPosition(int position) {
    this.prevPosition = this.position;
    this.position = position;
}
```

☐ Lav passende toString metoder.
Dette har vi ikke nået da vi nedprioriterede sprogkravet.

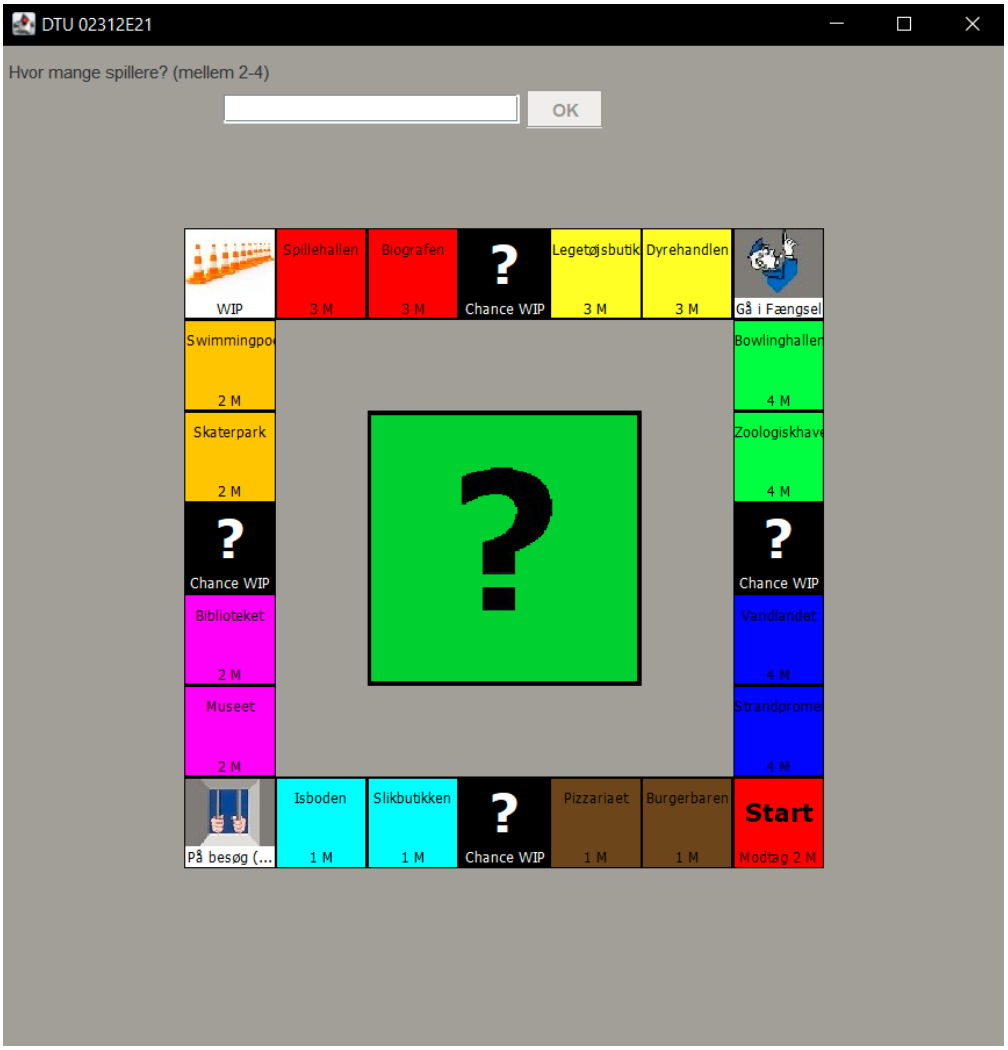
☑ ~~Lav en klasse GameBoard der kan indeholde alle felterne i et array.~~

```java
public class FieldFactory {
    public static Field[] MakeFields() {

        return new Field[]{
                new StartField( name: "Start", subtext: "Modtag 2 M"),
                new PropertyField( name: "Burgerbaren", subtext: "1 M", price: 1, rent: 1, new Color( r: 102, g: 70, b: 23)),
                new PropertyField( name: "Pizzariaet", subtext: "1 M", price: 1, rent: 1, new Color( r: 102, g: 70, b: 23)),
                new ChanceField( name: "?", subtext: "Chance WIP"),
                new PropertyField( name: "Slikbutikken", subtext: "1 M", price: 1, rent: 1, Color.CYAN),
                new PropertyField( name: "Isboden", subtext: "1 M", price: 1, rent: 1, Color.CYAN),
                new JailField( name: "I fængsel", subtext: "På besøg (WIP)"),
                new PropertyField( name: "Museet", subtext: "2 M", price: 2, rent: 2, Color.MAGENTA),
                new PropertyField( name: "Biblioteket", subtext: "2 M", price: 2, rent: 2, Color.MAGENTA),
                new ChanceField( name: "?", subtext: "Chance WIP"),
                new PropertyField( name: "Skaterpark", subtext: "2 M", price: 2, rent: 2, Color.ORANGE),
                new PropertyField( name: "Swimmingpoolen", subtext: "2 M", price: 2, rent: 2, Color.ORANGE),
                new LooseChangeField( name: "Gevinsten", subtext: "WIP"),
                new PropertyField( name: "Spillehallen", subtext: "3 M", price: 3, rent: 3, Color.RED),
                new PropertyField( name: "Biografen", subtext: "3 M", price: 3, rent: 3, Color.RED),
                new ChanceField( name: "?", subtext: "Chance WIP"),
                new PropertyField( name: "Legetøjsbutikken", subtext: "3 M", price: 3, rent: 3, Color.YELLOW),
                new PropertyField( name: "Dyrehandlen", subtext: "3 M", price: 3, rent: 3, Color.YELLOW),
                new GoToJailField( name: "Gå i Fængsel", subtext: "Gå i Fængsel"),
                new PropertyField( name: "Bowlinghallen", subtext: "4 M", price: 4, rent: 4, Color.GREEN),
                new PropertyField( name: "Zoologiskhave", subtext: "4 M", price: 4, rent: 4, Color.GREEN),
                new ChanceField( name: "?", subtext: "Chance WIP"),
                new PropertyField( name: "Vandlandet", subtext: "4 M", price: 4, rent: 4, Color.BLUE),
                new PropertyField( name: "Strandpromenaden", subtext: "4 M", price: 4, rent: 4, Color.BLUE),
        };
```

- [ ] Tilføj en toString metode der udskriver alle felterne i arrayet
  Som beskrevet før blev dette nedprioriteret.

- [x] ~~Benyt GUI'en. Gui' skal importeres fra Maven: Maven repository~~
  Vi har her et billede af vores spilbræt. Vi har valgt at ændre en håndfuld farver, da vi fandt dem ubehagelige. Vi har f.eks ændret baggrundsfarven til en grå, da den lyse kraftige grønne skar meget i øjnene.



6. Dokumentation

6.1 Arv

I java kan en subklasse arve attributter og metoder fra en superklasse ved brug af extends. Subklassen kan bruge de metoder superklassen har, og vil have superklassen attributter. Subklassen kan derudover have sine egne ekstra attributter og metoder. En final klasse kan ikke være en superklasse.

6.2 Abstract betydning

Man kan have abstract klasser og metoder. Abstract klasser behøver ikke at have abstract metoder, men abstract metoder skal være i abstract klasser. Abstract klasser kan ikke instantieres og abstract metoder er ikke implementeret. Det kan være praktisk at bruge abstracte superklasser hvis man har mange klasser der ligner hinanden så sub klasserne kan arve de metoder de skal bruge subklasser skal dog implementere de abstrakte metoder de arver.

6.3 Klasser med samme metode med forskellige effekter

Det er en abstrakt klasse, som bliver defineret med metoder og variabler af en klasse som har flere abstrakt metoder. Man bruge abstrakt klasse for at have en mere rent kode, det er ikke nødvendige at have det. Hvis en laver en klasse abstrakt, så vil den ikke kunne instantieres. Det hjælpe med at afskærme koden så at den ikke bliver brugt hvor den ikke skulle. For en abstrakt klasse til at virke, har den bruge for subklasse som definerer attributterne videre, sådan at de individuelt kan instantieres.

7. Test

7.1 Test cases

TESTCASE ID	TC01
BESKRIVELSE	Test om hver spiller starter på det samme felt.
Krav	K10: Man starter på 'Gå!' med ens brik
Test procedure	Ser ved spilstart, hvis brikkerne stå på 'Gå!'. At efter man rulle terning, at brikkerne rykker sig lige så langt væk fra starten som terninger kaster højt.
Test data	"Player" felt status ved start.
Forventet resultat	At bilerne flytter det ønskede antal felter.
Faktisk resultat	Brikkerne flyttet, 5 og 7 felter væg fra start efter det første terning kast af 5 og 7.
Status	Pass
Testet af	Christiaan
Dato	29/11/2021
Testmiljø	IntelliJ IDEA 2021.2.1 (Ultimate Edition) Build #IU-212.5080.55, built on August 24, 2021 Runtime version: 11.0.11+9-b1504.16 amd64 VM: OpenJDK 64-Bit Server VM by JetBrains s.r.o. GC: G1 Young Generation, G1 Old Generation Memory: 1024M Cores: 8 Kotlin: 212-1.5.10-release-IJ5080.55

TESTCASE ID	TC02
BESKRIVELSE	Test om spillet faktisk slutter når en af spillerne har ikke flere penge tillbage.
Krav	K1: Spillet slutter når 1 af spillerne ikke har flere penge tilbage.
Test procedure	Køre spillet forskellige gang ved 2, 3 og 4 spiller, indtil det slutter.
Test data	Spil slut ved en pengebeholdning af 0.

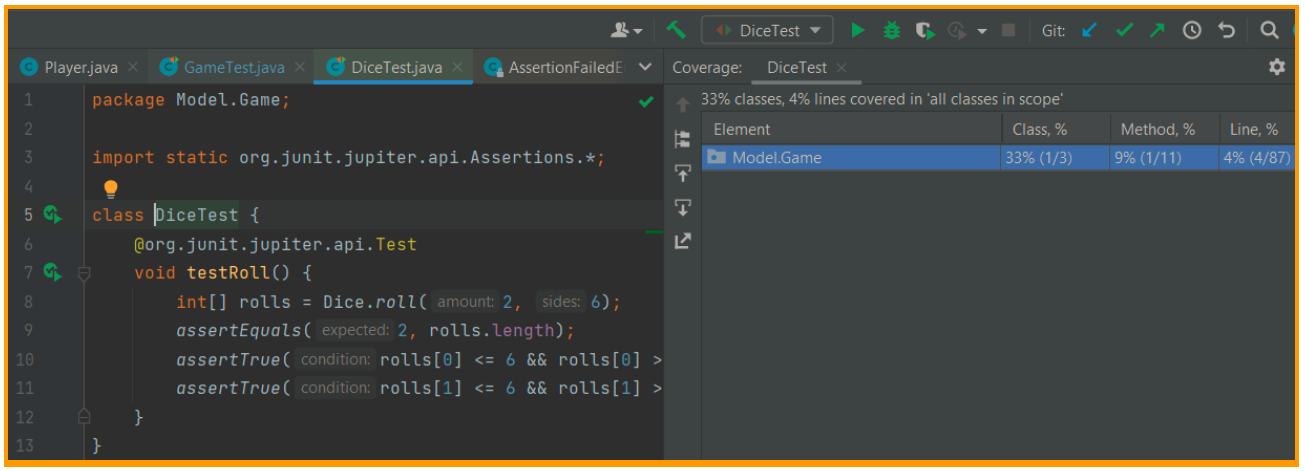
Forventet resultat	At spillet slutter når en af spillerne får 0 point.
Faktisk resultat	Spillet slutter kun hvis Spiller 1 havde minuspoint, og så kørt det en ekstra runde. Spillet køre for evigt hvis spiller 1 får for mange felter.
Status	Failed
Testet af	Christiaan
Dato	29/11/2021
Testmiljø	IntelliJ IDEA 2021.2.1 (Ultimate Edition) Build #IU-212.5080.55, built on August 24, 2021 Runtime version: 11.0.11+9-b1504.16 amd64 VM: OpenJDK 64-Bit Server VM by JetBrains s.r.o. GC: G1 Young Generation, G1 Old Generation Memory: 1024M Cores: 8 Kotlin: 212-1.5.10-release-IJ5080.55

TESTCASE ID	**TC03**
BESKRIVELSE	Test om felterne, som man ejer, virker rigtigt, ift. til penge overføring.
Krav	K15: Hvis man lander på en felt, ejet af en spiller, betale man spiller.
Test procedure	Kør spillet, og ser hvis de felter som er købt, og hvor en anden spiller lander på, faktisk laver den penge overføring den skulle. Køre spillet i en omgang, indtil det er slut.
Test data	“effekt” af bygning felterne.
Forventet resultat	At felterne som er ejet, overføre det ønsket mængde af penge fra den spiller som lander på det felt der er ejet.
Faktisk resultat	Spillerne får det rigtige mængde af penge når en spiller lander på deres felt.
Status	Pass
Testet af	Christiaan
Dato	29/11/2021
Testmiljø	IntelliJ IDEA 2021.2.1 (Ultimate Edition) Build #IU-212.5080.55, built on August 24, 2021 Runtime version: 11.0.11+9-b1504.16 amd64 VM: OpenJDK 64-Bit Server VM by JetBrains s.r.o.

	GC: G1 Young Generation, G1 Old Generation Memory: 1024M Cores: 8 Kotlin: 212-1.5.10-release-IJ5080.55

7.2 Junit Test

Vi har kørt en "DiceTest" i programmet ved at bruge Junit. Resultatet kan man ser nedenunder med code coverage.



7.3 Brugertest

For denne test har vi en bruger, som ikke finde ud at kode. Til det har vi spurgt en af vores familie medlemmer til at spille spillet.

Der er blevet testet følgende krav: K26: Skal anvendes GUI til spillet

Vi fik følgende feedback:

- Det er sjovt, at terningerne lander forskellige steder hver gang.
- Pludselig dukkede der en bil mere op, så vi blev 3 spillere i stedet for 2.
- Det er lidt svært at se, hvilken brik der rykker og hvorhen, fordi de rykker så hurtigt.
- Hvad er chance wip? Hvis det er kortene i midten, så kan man ikke vende dem.
- Spillet er lidt småt, det er lidt svært at se skriftstørrelsen.
- Nogle gange når man slår med terningerne, så lander de indenunder knappen, så man ikke kan se,

- hvad man har slået. Bilen lander også overskriften på brættet, så jeg ikke kan se, hvilken bod jeg er i gang med at købe.
- Det er godt at der kommer en blå/rød kant rundt om de boder, man har købt, så man kan se, hvem der ejer dem.
- Synes, bilerne er søde.

8. Konfigurationsstyring

Der er blevet brugt Git til versionsstyring af programmet. Programmet er blevet udviklet i version 2021.2.1 i intelliJ IDEA.

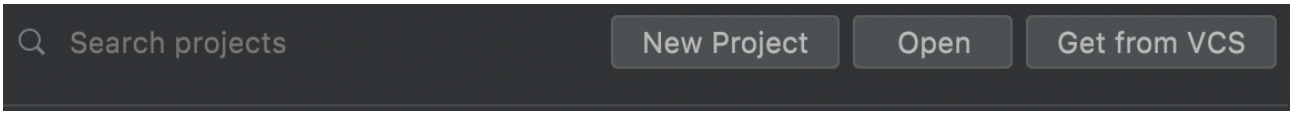
Spillet kan findes på denne URL: https://github.com/EspenUlff/CDIO3_23.git

Bemærk: For at kunne køre spillet skal version 2021.2.1 (eller nyere) af intelliJ IDEA downloades samt

8.1 Download og byg spil

spillet hentes ved brug af linket og åbnes gennem intelliJ

Åbn intelliJ → Click "Get from VCS"



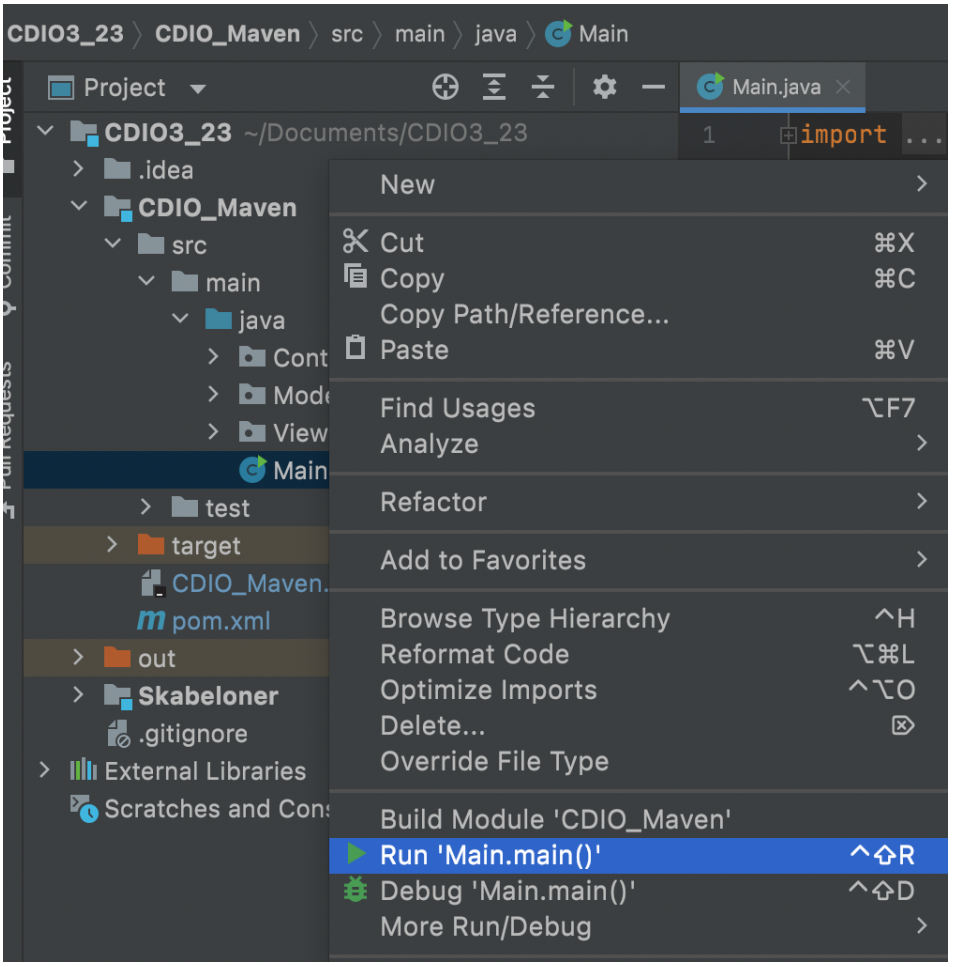
Sæt Version control as "Git" (som vist nedenunder) og kopier URL linket ind i boksen under

Efter programmet åbnes i intelliJ trykker man på den grønne hammer for at bygge programmet. herefter er spillet klar til brug.

Gå under følgende mapper for at køre spillet

CDIO_Maven → src → Main → java → klik "Run" Main.main

Se billedet under for vejledning



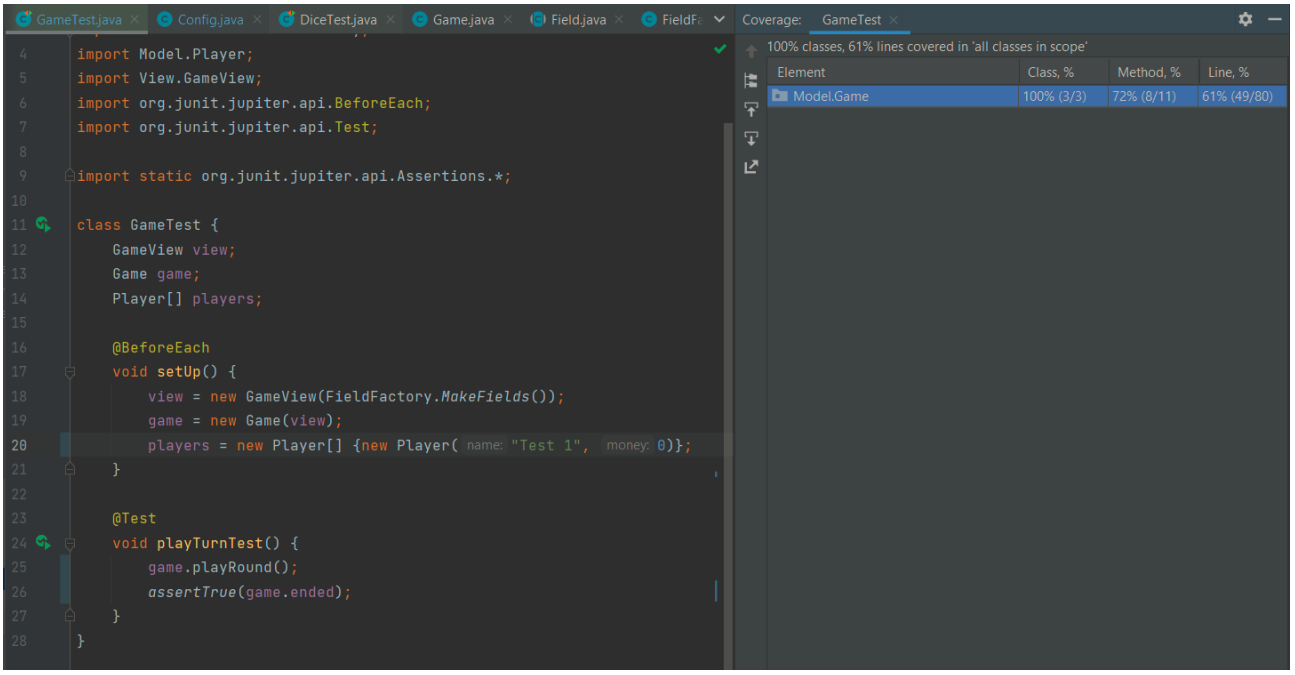
9. *Konklusion*

Som forventet fik vi (IOOuterActive) et større projekt end sidste gang, og vi har derfor måtte prioritere funktioner og krav for at kunne overholde tidsfristen. Vi kommet frem til et produkt som er et program som kan køre, og gøre brug af GUI´en , selvom at der er stadig nogle enkelte fejl. Vi har desværre ikke nået at bruge GRASP mønstret og toString metoden. Vi kan ud fra dette konkludere er vi er kommet frem til et tilfredsstillende produkt som overholder de givne krav.

Litteraturliste:

Bøger:

1. Larman Craig , CL. (2004). *Applying UML and Patterns : An introduction to object-Oriented Analysis and Design and Iterative Development* (3. udg.). John Wait.
2. Gamma Erich, G.E., Helm Richard, R.H., Johnson Ralph, R.J. & Vlissides John, J.V. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series .
3. Lewis John, L.J. & Loftus William, L.W. (2018). *Java TM Software Solutions : Foundations of Program Design* (9. udg.). Pearson Education.


GameTest.java
Config.java
DiceTest.java
Game.java
Field.java
FieldF
import Model.Player;
import View.GameView;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
class GameTest {
GameView view;
Game game;
Player[] players;
@BeforeEach
void setUp() {
view = new GameView(FieldFactory.MakeFields());
game = new Game(view);
players = new Player[] {new Player(name: "Test 1", money: 0)};
}
@Test
void playTurnTest() {
game.playRound();
assertTrue(game.ended);
}
}
Coverage: GameTest
100% classes, 61% lines covered in 'all classes in scope'
Element
Class, %
Method, %
Line, %
Model.Game
100% (3/3)
72% (8/11)
61% (49/80)