

Indledende programmering, Udvikling til IT-systemer og Versionsstyring og testmetoder

---

CDIO 3  
Gruppe 12

---



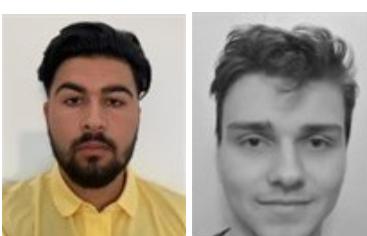
Vincent Buchholz | s205355

Tobias Stærmose | s205356



Nicole M. B. Oler | s205351

Oliver B. Klenum | s193625



Haider A. Akbar | s205334

Sebastian J. Pedersen | s205335

27. november 2020

## Abstract

This assignment is the last of our CDIO project, which is based upon knowledge from three different classes. The purpose of this assignment is to program a junior monopoly game, as we know it from the physical game with a gameboard, players etc.

We have used a GUI and IntelliJ as programs to develop the game, furthermore the clients requirements have been essential for the development. We have tested our program in different ways to ensure that the game is working as it is supposed to.

Furthermore, we have used a git repository to make the assignment, as this makes it possible for us to programme in the same file/document and programme whenever it is possible for us individually.

## Timeregnskab

Her sættes ind antal timer hver gang man har arbejdet på CDIO:

Dato	10/11	12/11	19/11	20/11	24/11	26/11	27/11
Vincent	2	4	3	4	5	3	3
Tobias	2	4	3	4	3	3	2
Haider	2	4	3	4	3	3	2
Nicole	2	4	3	4	3	3	2
Sebastian	2	4	3	4	3	3	2
Oliver	2	4	3	4	3	3	2

## Indholdsfortegnelse

1 Indledning	4
2 Krav	4
3 Analyse & design	6
5 Dokumentation	11
6 Test	12
7 Versionsstyring	15
8 Konfigurationsstyring	15
9 Projektplanlægning	17
10 Konklusion	17
11 Litteratur- og kildefortegnelse	17

## 1 Indledning

Denne opgave bygger på det vi har lært i kurserne Indledende programmering, Versionsstyring og testmetoder og Udviklingsmetoder til IT-systemer. I dette projekt skal vi udvikle et Monopoly Junior spil til IOOuterActive. Dette spil bygger på de forrige spil som vi også har udviklet til IOOuterActive, dog er der en reel spilleplade, samt rigtige spillefelter med. Vi har derfor i denne opgave lavet og designet et Monopoly Junior spil. Spillet kan spilles af 2-4 personer, og vindes af den person der har flest M (penge), når en person går fallit.

## 2 Krav

I forbindelse med programmering af Monopoly Junior har vi sammen med kunden, samt spillets regler, udarbejdet følgende krav til det færdige spil.

### Funktionelle krav:

k.1.1. Spillet skal indeholde:

2-4 spillere.

24 felter.

24 chancekort. (*obs. krav ikke opfyldt*)

k.1.2. Spillere skal modtage penge på deres konto ved spillets start:

k.1.2.1. Ved 2 spillere skal hver spiller modtage **16M**

k.1.2.2. Ved 3 spillere skal hver spiller modtage **18M**

k.1.2.3. Ved 4 spillere skal hver spiller modtage **20M**

k.1.3. Den yngste deltager skal starte spillet. Derefter roteres der mellem spillerne.

k.1.4. Alle spillere starter på feltet “**Start**”.

k.1.5. Spilleren, hvis tur det er, slår med en 6 sidet terning.

k.1.6. Spillerne rykker frem på brættet i urets retning.

k.1.7. Antal af terningens øjne skal afgøre hvor mange felter frem man rykker.

k.1.8. Spillerne starter deres tur fra det felt vedkommende sidst landede på.

k.1.9. Hver gang man lander eller passere feltet “**Start**” bliver **2M** overført til den givne spillers konto.

k.1.10. Hvert ledigt felt har en pris og en husleje. (*Se bilag 1*).

k.1.11. Når en spiller lander på et felt skal spilleren:

- k.1.11.1. **Chance felt:** Spilleren modtager et chancekort og påvirkes afhængig af chance kortet. (*Se bilag 2*).
- k.1.11.2. **Ledigt felt:** Spilleren skal købe det. Beløbet bliver trukket fra spillerens konto, afhængig af feltets pris, og feltet bliver nu ejet af den givne spiller.
- k.1.11.3. **Ejet felt:** Spilleren skal betale husleje til feltets ejer. Feltets husleje bliver trukket fra spillerens konto og overført til feltets ejers konto.
- k.1.11.4. **Dobbelts ejet felt:** Spilleren skal betale det dobbelte i husleje til feltets ejer, hvis ejeren har 2 felter i samme "farve".
- k.1.11.5. **"Gå i fængsel":** Spilleren rykkes til feltet "fængsel". Man modtager ikke **2M** ved passering af start. I starten af næste tur skal man betale **1M** eller bruge "Du løslades uden omkostninger" - chance kortet (*ikke opfyldt*). Herefter får du et terningekast og fortsætter som normalt.
- k.1.11.6. **Gratis parkering:** Ingen ændring.
- k.1.11.7. **På besøg:** Ingen ændring.
- k.1.11.8. **Felt man selv ejer:** Ingen ændring
- k.1.12. Spillet slutter, når en spiller går fallit, dvs. ikke kan betale husleje, købe ledigt felt eller betale afgift fra et chancekort.
- k.1.13. Vinderen af spillet er den spiller med flest **M** på sin konto ved spillets afslutning. I tilfælde hvor flere spillere har samme antal **M** på deres konti, er vinderen spilleren med størst ejendomsværdi (*ikke opfyldt*).

### **Usability krav:**

- k.2.1. Krav til brug af GUI som er angivet i opgavebeskrivelsen.

### **Performance krav:**

- k.3.1. Spillet skal kunne starte op på mindre end 5 sekunder.  
k.3.2. Spillet skal have en responstid på mindre end 1 sekunder.

### **Supportability krav:**

- k.4.1. Mindst tre testcases med tilhørende fremgangsmåde/testprocedure og testrapporter.  
k.4.2. Mindst en Junit test til centrale metoder  
k.4.3. Mindst en brugertest. Brugeren må ikke have erfaring med programmering.  
k.4.4. Koden skal indeholde relevante kommentarer.  
k.4.5. Koden skal dokumenteres ved brug af GIT.

**Minimum systemkrav:**

- k.5.1. Windows 10
- k.5.2. macOS
- k.5.3. Java 8, JDK 14, I intellij Java Version 14.0.2 (SDK)
- k.5.4. HDD 500 MB
- k.5.5. 2 GB RAM
- k.5.6. CPU 1 ghz

**Fysiske krav:**

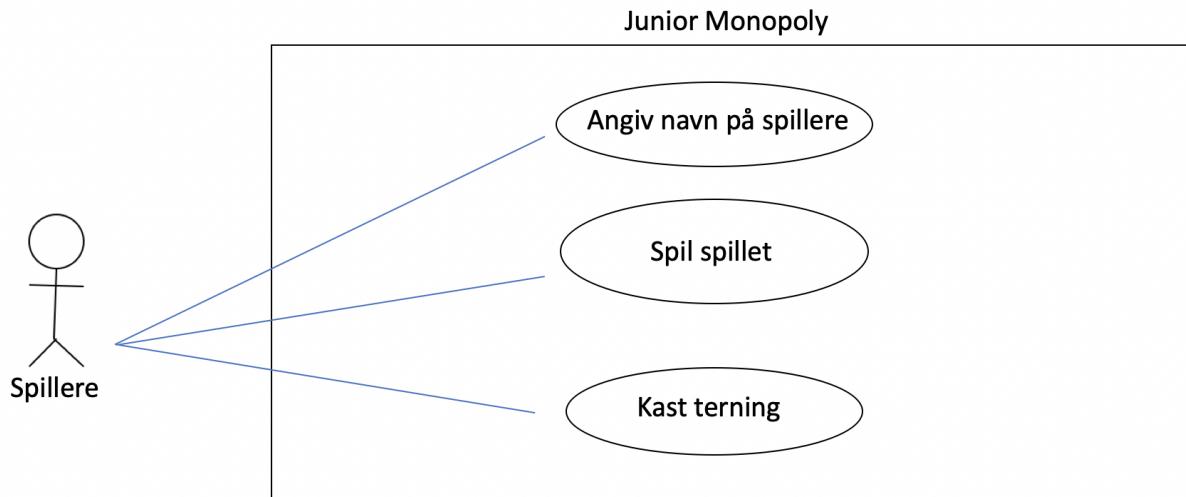
- k.6.1. Spillet skal kunne køre på DTU's databarer.

3 Analyse & design

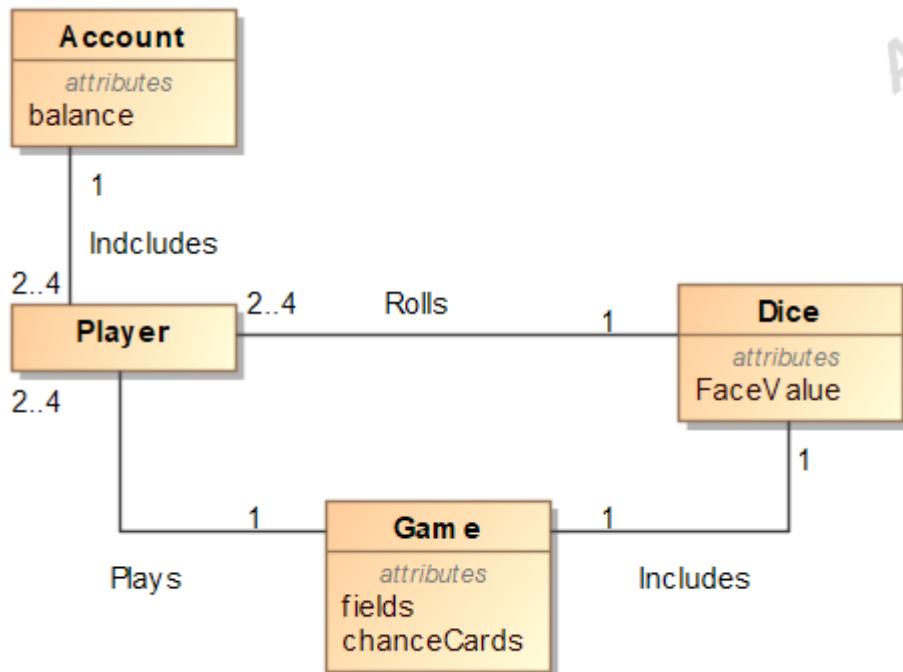
3.1 Use cases i Junior Monopoly

1. Angiv navne
2. Kast terning
3. Spil spillet

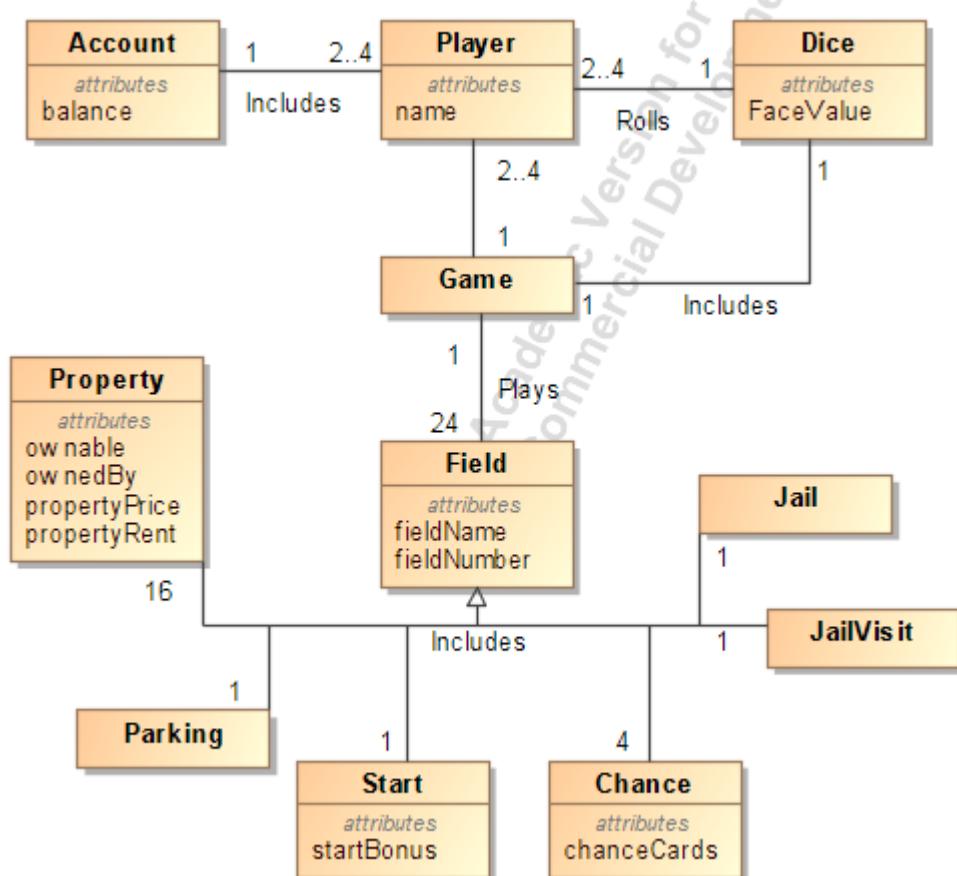
Nedenfor ses use case diagrammet for Junior Monopoly spillet. Der er 3 centrale use cases:



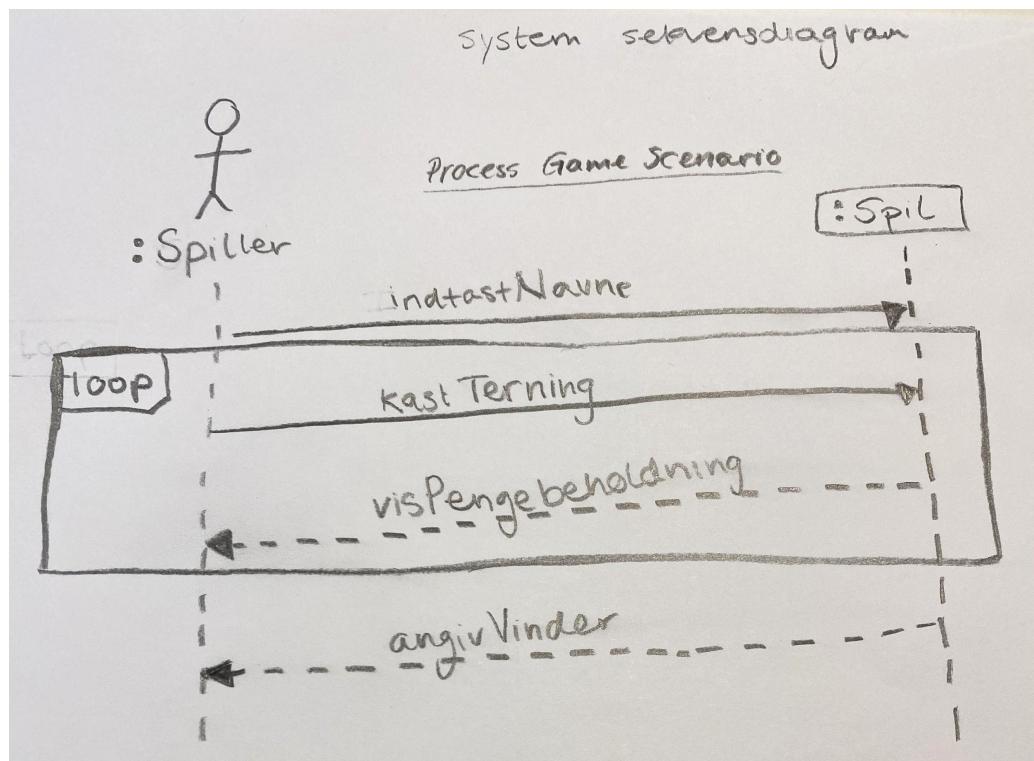
Domænemodellen for spillet er skitseret nedenfor:



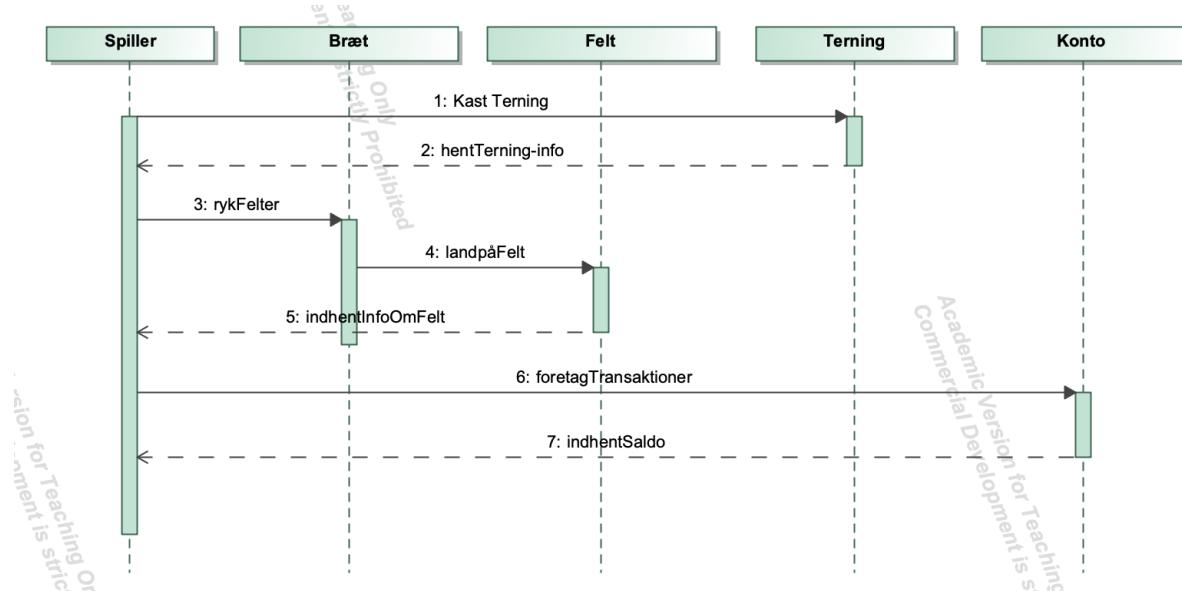
Udvidet domænemodel:



Her ses vores systemsekvensdiagram:



3.2 Her ses sekvensdiagrammet:



### 3.3 Casual beskrivelser af Use Cases

**Use case:** Angiv Navne

**Main success scenario:** Brugerne, som ønsker, at spille spillet angiver navne på spillere.

**Alternate scenario:** Brugerne skriver andet end deres navn.

**Use case:** Kast terningen

**Main success scenario:** Systemet kaster terningerne, efter at spillere har angivet dette skal ske.

**Alternate scenario:** Systemet slår ikke terningerne.

**Use Case:** Spil Monopoly Junior

**Fully Dressed**

**Use Case UC1: Play Monopoly Junior**

**Scope:** Monopoly Junior Game

**Primary Actor:** Players

**Stakeholders and Interests:**

- The players want an easy to play Monopoly game that suits the younger audience. Therefore the application shall be fun, simple and quick so the gameplay is smooth and easy to understand.
- The customer wants a fast, responsive and working Monopoly Game that includes the ground rules for a monopoly game with a suitable GUI.

**Preconditions:** The players must enter their player names to start the game.

**Success Guarantee (or Postconditions):** Game starts after Player Names are set. The balance is recorded and saved throughout the game. Game ends when a player can't pay their debt. Player with the highest balance wins.

**Main Success Scenario (or Basic Flow):**

1. Customer starts Monopoly Junior
2. Players enter their names.
3. Starting balance is deposited to the players
4. Game starts and the players are placed on the starting field.
5. First player rolls the dice and actions vary depending on where you land. (There are 24 fields that each have different actions that can affect your balances both ways.
6. If you land on an available field you must buy it if you have enough balance. When you buy it an indicator will show that you own it.
7. If the field is owned by another player you must pay rent to them according to the field's determined rent.
8. If a player can't afford rent, the game stops and the player with the highest balance wins.

**Extensions (or Alternative Flows):**

- a. If you land on a “chance field” the game will proceed with the statement written on the card.
- b. If a player owns both fields in the same colorgroup the rent you own doubles.
- c. If you land on the “go to jail” field, you have to move your character to the “jail” field.
- d. To escape from jail you must use 1 coin or a chance card that lets you out. Game continues afterwards.

**Special Requirements:**

- Playable on machines with an operating system that has a JVM installed.  
Executable from IDE or CommandPrompt.

**Frequency of Occurrence:** Game can be run at any time.

**Open Issues:**

Elements of the game should be adjusted according to the audience playing it.

Anvendelse af GRASP-mønstre:

Vi har benyttet os af GRASP-mønstrene “low coupling” og “high cohesion” i klasserne “Die” og “Account”. Begge disse klasser bruger ikke metoder eller henter information fra andre klasser. Derudover er de fokuseret, som betyder at klasserne kun står for deres egne ansvarsområder. I dette tilfælde betyder det at klassen “Die” er eneansvarlig for at oprette og udføre terninge elementer og indeholder kun metoder til dette formål. På samme måde er klassen “Account” eneansvarlig for at oprette og håndtere spillernes konti. Dette resulterer i at ændringer i disse klasser har lav til ingen påvirkning på andre klasser, hvilket resultere i færre fejl. Derudover gør det muligt for klasserne let at blive genbrugt i andre programmer.

4 Implementering

Nogle af de overvejelser vi har taget i betragtning under implementeringen af koden, har været at gruppere de forskellige funktioner under diverse java klasser. Når klasserne er opdelt, giver det en bedre uafhængighed mellem klasserne (Low coupling). Klasserne er gennemarbejdet på den måde at de kan implementeres i klassen “Main”. De arbejder på den måde sammen om at eksekvere det “færdige produkt” (High Cohesion).

Derudover har diff-listen også bidraget til at se ændringer i de forskellige commits, henholdsvis hvad der er blevet tilføjet og slettet.

## 5 Dokumentation

### 5.1 Arv

Klasser i objektorienteret analyse kan arve eller nedarve egenskaber fra andre klasser. Hvis en klasse skal kunne næsten det samme som en anden klasse i ens program, så kan man sørge for, at den ”arver” dens egenskaber og funktionalitet. Super-klassen er den der nedarves fra og sub-klassen er den der nedarves til.

### 5.2 Abstract

En abstrakt klasse, er defineret som en klasse som ikke kan instantieres men kan fungere som en subklasse. De kan levere funktioner til ”forældre” klassen og kan til fordel benyttes hvis du skal flette det sammen med klasser som har relevans

### 5.3

Hvis alle fieldklasserne har en landOnField metode der gør noget forskelligt, så er der tale om override polymorfi. Det har vi dog ikke implementeret i vores Monopoly Junior spil.

### 5.4 Dokumentation til brug af GRASP-mønstre:

```
4  public class Account {
5      private int balance;
6
7      public int deposit(int amountToDeposit) {
8          this.balance = this.balance + amountToDeposit;
9          return balance;
10     }
11
12     public int withdraw(int amountToWithdraw) {
13         this.balance = this.balance - amountToWithdraw;
14         return this.balance;
15     }
16
17     public int getBalance() {
18         return balance;
19     }
20
21     public int setBalance(int balance) {
22         this.balance = balance;
23         return this.balance;
24     }
25
26     public String toString() {
27         return String.valueOf(this.balance);
28     }
29 }
```

```

3 import java.util.Random;
4
5
6     public class Die {
7         Random rand = new Random();
8
9         private int die;
10
11
12         public void roll() {
13             this.die = rand.nextInt( bound: 6)+1;
14         }
15
16
17         public int getDie(){
18             return die;
19         }
20
21         public String toString() {
22             return String.valueOf(die);
23         }
24     }

```

## 6 Test

### 6.1 Junit Tests

I denne test af die klassen, har vi testet om man kan stole på at terningen virker og er tilfældig. Ved hjælp af junit test har vi rullet terningen 1000 gange og den har vist en værdi mellem 1 & 6 hver gang. Desuden har vi printet antal gange terningen har landet på de forskellige værdier. Da der er ingen store afvigelser kan vi bekræfte at terningen virker.

The screenshot shows an IDE interface with several windows:

- Project View:** Shows the project structure under 'CDIO3' with packages 'src' and 'test' containing various Java files like 'Fields.java', 'AccountTest2.java', and 'DieTest.java'.
- Code Editor:** Displays the 'DieTest.java' file with the following code:

```

import ...
class DieTest {
    @Test
    void roll() {
        int one = 0;
        int two = 0;
        int three = 0;
        int four = 0;
        int five = 0;
        int six = 6;
        for (int i = 0; i < 1000; i++) {
            Die DieTest = new Die();
            DieTest.roll();
            System.out.print(DieTest.getDie() + " ");
            assertTrue( condition: 1 <= DieTest.getDie()
        }
    }
}

```
- Coverage Analysis:** A 'Coverage' window titled 'DieTest' showing coverage details:

Element	Class, %	Method, %	Line, %
apple	25%	(1/4)	10% (2/19)
CDIO3	25%	(1/4)	10% (2/19)
com			
gui_codebehind			
gui_fields			
gui_resources			
gui_tests			
images			
java			
javax			
jdk			
junit			
META-INF			
netscape			
org			
sun			
toolbarButtonGra...			
- Run View:** Shows the 'Run' tab with 'DieTest' selected. The 'Test Results' section displays the output of the test, showing counts for each die value from 1 to 6.

I den første test af vores Account klasse, har vi testet om vores deposit metode virker og er pålidelig. Så vi har valgt at køre en junit test hvor vi indsætter 1500M til vores balance i et loop 1000 gange. Det lykkedes og vi får en ny balance på 1500M

```

import ...

class AccountTest {
    @Test
    void deposit() {
        for (int i = 0; i < 1000; i++) {
            Account accountTest = new Account();
            System.out.println("Starting Balance " + accountTest.getBalance());
            accountTest.deposit(amountToDeposit);
            System.out.println("Balance after Deposit " + accountTest.getBalance());
        }
        int expectedBalance = 1500;
        assertEquals(expectedBalance, accountTest.getBalance());
    }
}

```

Coverage: AccountTest

Element	Class, %	Method, %	Line, %
apple			
CDIO3	25%	(1/4)	15% (3/19) 0% (5/8...)
com			
gui_codebehind			
gui_fields			
gui_main			
gui_resources			
gui_tests			
images			
java			
javax			
jdk			
junit			
META-INF			
netscape			
org			
sun			
toolbarButtonGra...			

Run: AccountTest

Tests passed: 1

Det samme gør vi med withdraw metoden. Vi laver en start balance på 1500M hvorefter vi fratrækker 750M. Vi forventer at der så endeligt står 750M på kontoen. Efter at have testet det i et loop 1000 gange kan vi bekræfte at det virker.

```

import ...

class AccountTest2 {
    @Test
    void withdraw() {
        for (int i = 0; i < 1000; i++) {
            Account accountTest2 = new Account();
            accountTest2.deposit(amountToDeposit);
            accountTest2.withdraw(amountToWithdraw);
            System.out.println("Starting Balance " + accountTest2.getBalance());
            System.out.println("Balance after Withdraw " + accountTest2.getBalance());
        }
        int expectedBalance = 750;
        assertEquals(expectedBalance, accountTest2.getBalance());
    }
}

```

Coverage: AccountTest2

Element	Class, %	Method, %	Line, %
apple			
CDIO3	25%	(1/4)	21% (4/19) 0% (7/8...)
com			
gui_codebehind			
gui_fields			
gui_main			
gui_resources			
gui_tests			
images			
java			
javax			
jdk			
junit			
META-INF			
netscape			
org			
sun			
toolbarButtonGra...			

Run: AccountTest2

Tests passed: 1

Codecoverage har vi kørt sammen med vores tests, for at se hvor meget af koden der bliver anvendt når vi afprøver metoderne. Som kommentarer til vores codecoverage kan vi se at der ikke en test som bruger mindre end 50% af metoderne i fra den tilhørende klasse. Det ses så at der er en god sammenhæng mellem funktionaliteterne og de klasser de tilhører.

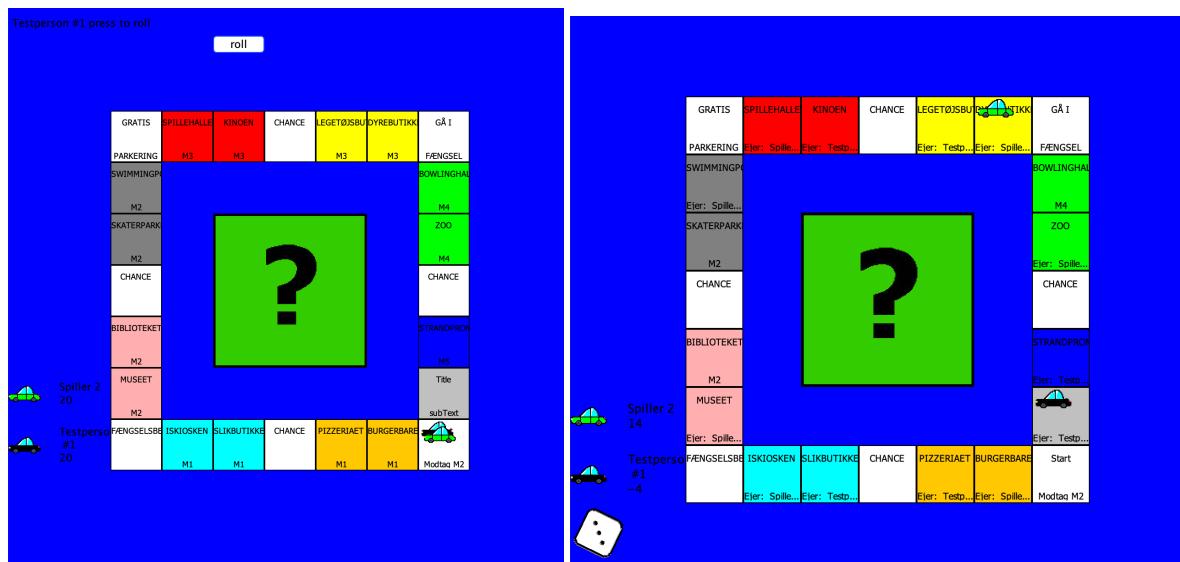
## 6.2 Brugertest

En brugertest er en test, hvor en eller flere personer fra målgruppen afprøver spillet. Det er vigtigt for en god brugertest, at testpersonen ikke selv kender til kodning eller introduceres i hvordan spillet fungerer. Eftersom brugertest omhandler hvor brugervenligt spillet er, dvs. hvordan opfatter en person ude fra projektet spillet.

Til denne brugertest af Monopoly Junior har vi valgt at bruge flere testpersoner for at teste vores spil bredere end ved blot et enkelt individ.

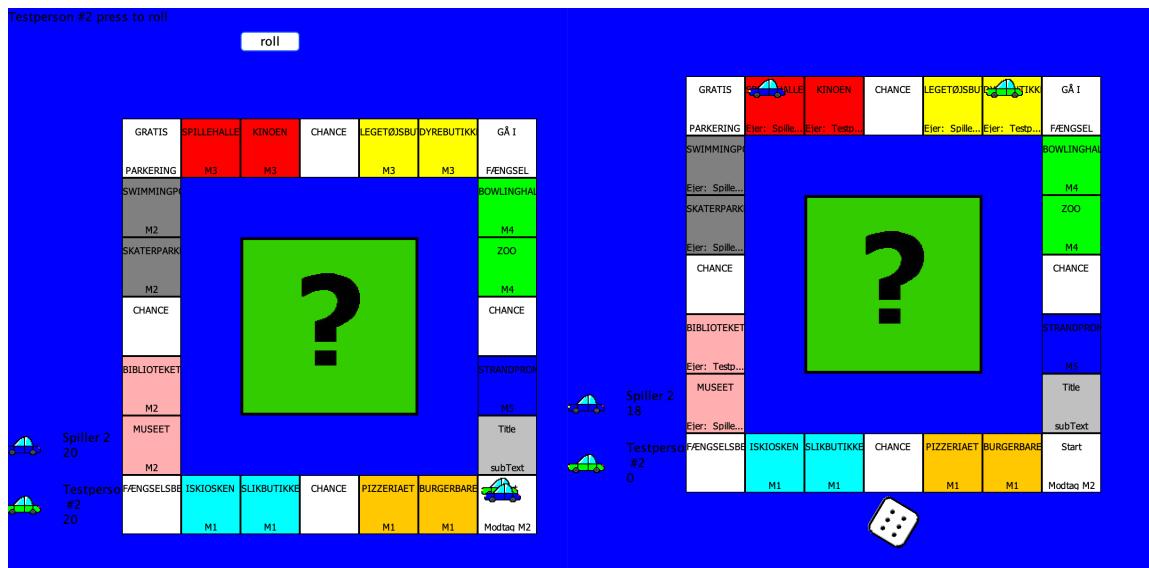
### Brugertest #1

Første test blev foretaget af en mand på 52 år, som ikke kan kode i forvejen. Testpersonen kendte ikke umiddelbart til det almindelig Monopoly Junior. Derfor tog testpersonen sig tid til at efterse pladen for at lokalisere de forskellige felter, samt roll-knappen. Herefter havde testpersonen ingen problemer med at spille spillet.



### Brugertest #2

Anden test blevet foretaget af en pige på 9 år, sammen med en ældre person, der kunne hjælpe med oplæsning af teksten på skærmen. Herfra havde testpersonen heller ikke problemer med at benytte sig at spillet.



## 7 Versionsstyring

Rapporten skal indeholde en vejledning i hvordan man importerer Git-repository i IntelliJ.

Link til dette Monopoly Juniors Git-repository:

<https://github.com/VincentBuchholz/CDIO3.git>

Link til versionering af rapport

[https://drive.google.com/drive/folders/1aUR\\_eGY8bgMzztlS2O1qtyB8tySQUVC?usp=sharing](https://drive.google.com/drive/folders/1aUR_eGY8bgMzztlS2O1qtyB8tySQUVC?usp=sharing)

## 8 Konfigurationsstyring

Vores terningespil kan køre på Mac og Windows systemer med Java 8 & JDK 15 installeret. Så vil det være muligt at kompilere og køre programmet kan køre igennem en kommandoprompt.

Min 500 MB HDD plads på systemet

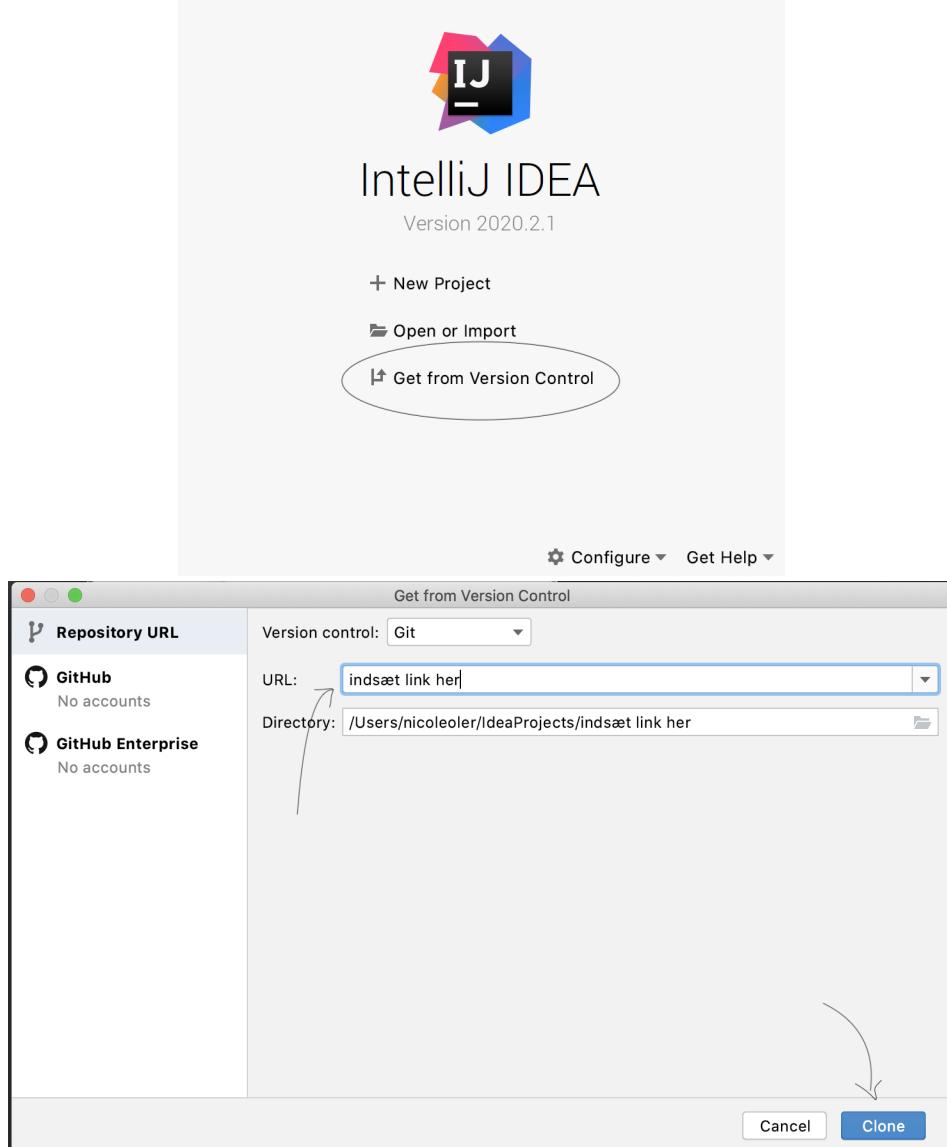
Min 2 GB ram, CPU 1 ghz

I intelliJ

Java 8, Projekt SDK java version 14.0.2

Koden importeres fra et git repository gennem versionsstyring på to måder, for det første ved

invitation til respositoriet, herfra skal invitationen accepteres, hvilket gør det muligt at installere koden. Den anden måde er, hvis linket er opgivet, som det er ovenfor. Herfra kopierer man linket og indsætter det i IntelliJ-programmet, som vist nedenfor.



Efter at have klikket på 'clone'-knappen vil programmeret være downloaded ud fra versionsstyring. Installeringen er dermed færdiggjort og klar til at blive afviklet. Hvis du anvender den rigtige JVM og opfylder vores specificeret krav kan du køre programmet. Først skal du ned i stien: CDIO3\del3\CDIO3\src\main\java\CDIO3 og finde main klassen. Derfra kan du køre main metoden og således spillet.

```
▶ ⏪ ⏴ public static void main(String[] args) {
```

## 9 Projektplanlægning

I udarbejdelse af denne CDIO har vi i et vist omfang gjort brug af unified process, da dette giver et godt overblik over arbejdsprocessen, samt hvad der skulles gøres hvor og hvornår i processen. Derudover har vi gjort brug af GitHub til at kunne arbejde uafhængigt af hinanden, samt se hvad andre gruppemedlemmer har arbejdet på.

## 10 Konklusion

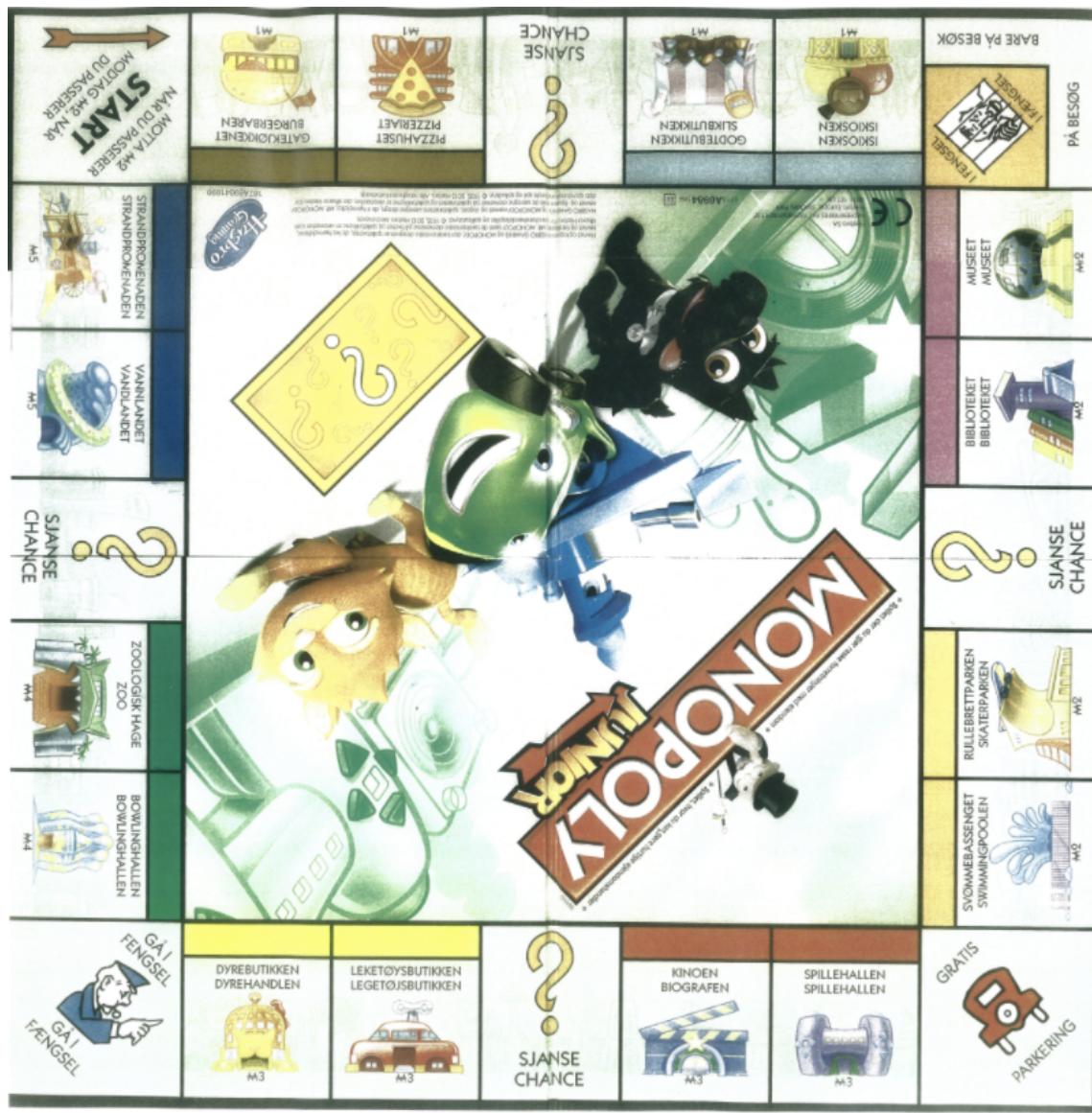
I denne CDIO har vi udviklet et spil til IOOuterActive, og gjort brug af vores erfaringer med unified process til at skabe et godt overblik på arbejdsprocessen, samt gjort brug af GitHub, så vi har kunne arbejde uafhængigt af hinanden, og følge op på det. Vi har skabt et Monopoly junior spil til IOOuterActive, denne gang med en reel spilleplade. Denne arbejdsproces har været præget af vores viden fra kurserne Indledende programmering, Versionsstyring og testmetoder og Udviklingsmetoder til IT-systemer. Vi har derfor skabt og testet et spil som skulle indeholde specifikke krav. Spillet kan spilles af 2-4 personer, vinderen er personen med flest M (penge), når en den første spiller er gået fallit.

## 11 Litteratur- og kildefortegnelse

- Applying UML and Patterns. 3. ed. Craig Larman
- CDIO 1 & 2

## 12 Bilag

## Bilag 1:



## Bilag 2:

<p><b>CHANCE</b></p> <p>Giv dette kort til <b>BILEN</b>, og tag et chancekort mere.</p> <p><b>BIL:</b> På din næste tur skal du cirkone 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!</p> <p></p> <p>© 1936, 2013 Hasbro</p>	<p><b>CHANCE</b></p> <p>Ryk frem til <b>START</b>. Modtag <b>M2</b>.</p> <p></p> <p>© 1936, 2013 Hasbro</p>	<p><b>CHANCE</b></p> <p>Ryk <b>op til</b> 5 felter frem.</p> <p></p> <p>© 1936, 2013 Hasbro</p>
<p><b>CHANCE</b></p> <p><b>GRATIS FELT!</b></p> <p></p> <p>Ryk frem til et <b>orange</b> felt. Hvis det er ledigt, får du det <b>GRATIS!</b> Ellers skal du <b>BETALE</b> leje til ejeren.</p> <p>© 1936, 2013 Hasbro</p>	<p><b>CHANCE</b></p> <p>Ryk 1 felt frem, <b>eller</b> tag et chancekort mere.</p> <p></p> <p>© 1936, 2013 Hasbro</p>	<p><b>CHANCE</b></p> <p>Giv dette kort til <b>SKIBET</b>, og tag et chancekort mere.</p> <p><b>SKIB:</b> 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!</p> <p></p> <p>© 1936, 2013 Hasbro</p>
<p><b>CHANCE</b></p> <p>Du har spist for meget slik.</p> <p><b>BETAL</b> <b>M2</b> til banken.</p> <p></p> <p>© 1936, 2013 Hasbro</p>	<p><b>CHANCE</b></p> <p><b>GRATIS FELT!</b></p> <p></p> <p>Ryk frem til et <b>orange</b> eller <b>grønt</b> felt. Hvis det er ledigt, får du det <b>GRATIS!</b> Ellers skal du <b>BETALE</b> leje til ejeren.</p> <p>© 1936, 2013 Hasbro</p>	<p><b>CHANCE</b></p> <p><b>GRATIS FELT!</b></p> <p></p> <p>Ryk frem til et <b>lyseblåt</b> felt. Hvis det er ledigt, får du det <b>GRATIS!</b> Ellers skal du <b>BETALE</b> leje til ejeren.</p> <p>© 1936, 2013 Hasbro</p>
<p><b>CHANCE</b></p> <p>Du løslades uden omkostninger.</p> <p>Behold dette kort, indtil du får brug for det.</p> <p></p> <p>© 1936, 2013 Hasbro</p>	<p><b>CHANCE</b></p> <p>Ryk frem til Strandpromenaden.</p> <p></p> <p>© 1936, 2013 Hasbro</p>	<p><b>CHANCE</b></p> <p>Giv dette kort til <b>KATTEN</b>, og tag et chancekort mere.</p> <p><b>KAT:</b> 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!</p> <p></p> <p>© 1936, 2013 Hasbro</p>
<p><b>CHANCE</b></p> <p>Giv dette kort til <b>HUNDEN</b>, og tag et chancekort mere.</p> <p><b>HUND:</b> 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!</p> <p></p> <p>© 1936, 2013 Hasbro</p>	<p><b>CHANCE</b></p> <p>Det er din fødselsdag! Alle giver dig <b>M1</b>.</p> <p><b>TILLYKKE MED FØDSELSDAGEN!</b></p> <p></p> <p>© 1936, 2013 Hasbro</p>	<p><b>CHANCE</b></p> <p><b>GRATIS FELT!</b></p> <p></p> <p>Ryk frem til et <b>pink</b> eller <b>mørkeblåt</b> felt. Hvis det er ledigt, får du det <b>GRATIS!</b> Ellers skal du <b>BETALE</b> leje til ejeren.</p> <p>© 1936, 2013 Hasbro</p>
<p><b>CHANCE</b></p> <p>fra berline.</p> <p>alle dine lejligheder</p> <p><b>MØDTAG M2</b></p> <p>Du har løbet</p> <p></p> <p>© 1936, 2013 Hasbro</p>	<p><b>CHANCE</b></p> <p><b>GRATIS FELT!</b></p> <p></p> <p>Ryk frem til et <b>rødt</b> felt. Hvis det er ledigt, får du det <b>GRATIS!</b> Ellers skal du <b>BETALE</b> leje til ejeren.</p> <p>© 1936, 2013 Hasbro</p>	<p><b>CHANCE</b></p> <p><b>GRATIS FELT!</b></p> <p>Ryk frem til <b>Skaterparken</b> for at lave det perfekte grind!</p> <p>Hvis ingen ejer den, får du den <b>GRATIS!</b> Ellers skal du <b>BETALE</b> leje til ejeren.</p> <p></p> <p>© 1936, 2013 Hasbro</p>