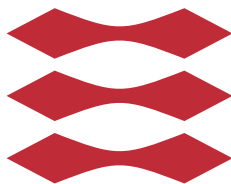


# DTU



## DANMARKS TEKNISKE UNIVERSITET

---

### CDIO 3

---

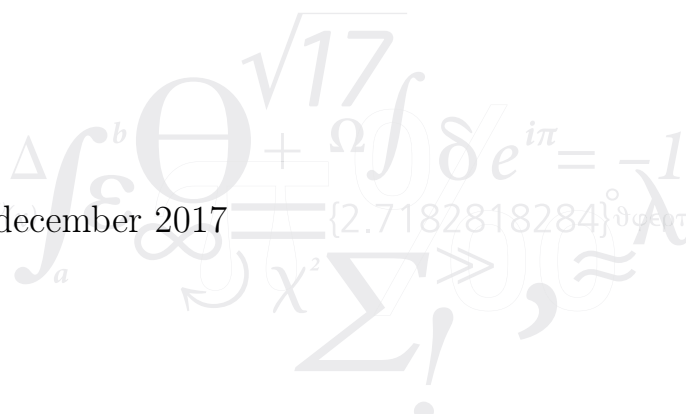
***Gruppe 18:***

Ali Harb El-Haj Moussa (s175119)  
Magnus Hansen (s175198)  
Niklaes Dino Robbin Jacobsen (s160198)  
Oliver Storm Køppen (s175108)  
Sebastian Bilde (s175116)

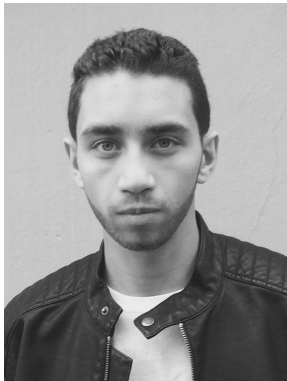
***Underviser:***

Ian Bridgwood, 02313  
Stig Høgh, 02314  
Sune Nielsen, 02315

$f(x+\Delta x) = \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x)$   
1. december 2017



# Gruppens medlemmer



Ali Harb El-Haj  
Moussa (s175119)



Magnus Hansen  
(s175198)



Niklaes Dino Robbin  
Jacobsen (s160198)



Oliver Storm Køppen  
(s175108)

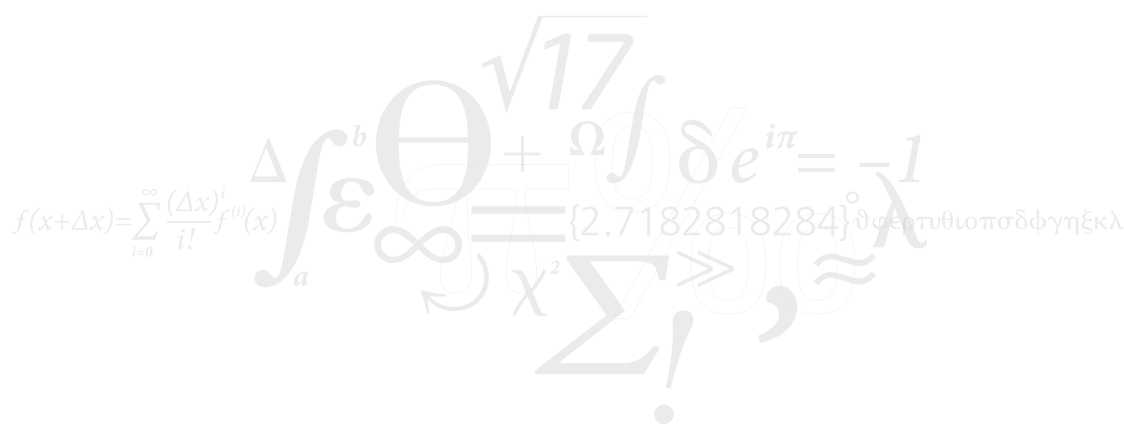


Sebastian Bilde  
(s175116)

## Ekstra informationer

Projekt på Github: <https://github.com/group-18/CDI03>

Dokumentation (JavaDoc): <https://group-18.github.io/CDI03/>



## Abstract

As with both CDIO1 and CDIO2, this project is combining the three IT-related courses *Development methods for IT-systems*, *Introductory programming*, and *Version control and test methods*. Like the former project the purpose is to create the well-known 'Monopoly'-game, and whilst the latter projects were parts of the game, this project combines those parts into 'Monopoly-junior'.

We succeeded creating the 'Monopoly-junior'-game without sacrificing any requirements, thus fulfilling the requirements of the customer and the integrity of a real 'Monopoly'-game which is known for its key components such as choosing the type of player you are and letting the youngest player start.

The fundamental methods such as throwing a die, changing the players' names et cetera have all been unit tested and been documented in the code with a corresponding javadoc.

The game has been analyzed and designed with the help of usecase-, sequence-, and classdiagrams all written in the UML-notation as part of a agile development proces to simulate the industry standard.

To conclude, we are all pleased with the different phases of the development process as well as the final product.

$$f(x+\Delta x)=\sum_{i=0}^{\infty}\frac{(\Delta x)^i}{i!}f^{(i)}(x)$$
$$\int_a^b \varepsilon \Theta_+^\Omega \delta e^{i\pi} = -1$$
$$\infty = \{2.7182818284\}^\circ \lambda$$
$$\chi^2 \sum_i \gg \approx$$

# Timeregnskab

Nedenfor vises vores forbrug i timer for projektet, inddelt i kategori og navn.

CDIO 3						
Deltager	Analyse	Design	Implementering	Test	Dokumentation	I alt
Ali Moussa	6	9	9	1	3	28
Magnus Hansen	1	1	25	5	3	35
Niklaes Jacobsen	10	6	11	2	3	32
Oliver Køppen	5	8	14	1	5	32
Sebastian Bilde	2	5	17	1	4	29

# Indhold

<b>1</b>	<b>Indledning</b>	<b>3</b>
<b>2</b>	<b>Problemformulering</b>	<b>4</b>
<b>3</b>	<b>Analyse</b>	<b>5</b>
3.1	Kravspecifikation . . . . .	5
3.2	Interessentanalyse . . . . .	11
3.3	Use case diagram . . . . .	11
3.4	Use case . . . . .	12
3.5	Risicianalyse . . . . .	14
3.6	Domænemodel . . . . .	15
<b>4</b>	<b>Konfigurationsstyring</b>	<b>16</b>
4.1	Konfigurationsvejledning . . . . .	17
4.2	Vejledning til eksport samt kørsel af program . . . . .	18
<b>5</b>	<b>Design</b>	<b>19</b>
5.1	Flowdiagram . . . . .	19
5.2	Klassediagram . . . . .	20
5.3	Systemsekvensdiagram . . . . .	21
5.4	Sekvensdiagram . . . . .	22
<b>6</b>	<b>Implementering</b>	<b>23</b>
6.1	Brugervejledning for programmet . . . . .	23
6.2	Verificering ift. krav og validering ift. kunde . . . . .	24
6.3	Dokumentation af kode . . . . .	25
<b>7</b>	<b>Test</b>	<b>27</b>
7.1	Unit test . . . . .	27
7.2	Coverage test . . . . .	28
7.3	Monkey test . . . . .	28
7.4	Testcases . . . . .	28
<b>8</b>	<b>Projektplanlægning</b>	<b>33</b>
<b>9</b>	<b>Konklusion</b>	<b>35</b>
<b>A</b>	<b>Bilag</b>	<b>36</b>
A.1	Reglementet til Matador Junior . . . . .	36

# 1 | Indledning

IOOuterActive har igen modtaget en opgave, denne opgave vil i projektet blive referet til CDIO3. I vores projekt CDIO3, skal vi designe et junior matadorspil, der inderholder alle de regler vi kan nå at implementerer. Vi skal altså tage stilling til hvilke regler der er højest prioriteret, og sørge få at få implementeret disse, således at spillet ville kunne køre optimalt. Spillet spilles af 2 til 4 personer, og vil foregå som vores forrige projekt, ved at brugeren interagerer med GUI'en ved hjælp af simple knapper. Dette gør spillet nemt forståeligt at spille, og nemt at holde øje med stillingen. Vi vil desuden benytte nogle af klasserne i koden fra vores forrige projekt CDIO2. Rapporten vil desuden indeholde en tydelig og klar dokumentation af vores kode, derudover vil den også indeholde et antal test cases af koden, samt et antal JUnit tests af vores kode.

## 2 | Problemformulering

Er det muligt at producere en udgave af Matador Junior, hvor de fleste regler er implementeret? Kan dette gøres inden for den givne tidsramme, uden at gå for meget på kompromis med kravene? Dette vil vi undersøge, og prøve at få færdigudviklet et matadorspil, der indeholder alle kravene, der er til spillet. Vi vil liste kravene op efter deres indflydelse på spillet, således at vi når at få implementeret de væsentligste krav. Vi vil igennem opgaven udvikle programmet ud fra vores analyse af krav, og udfra vores opstillede design model med de respektive klasser.

## 3 | Analyse

I dette kapitel vil vi gennemgå kravspecifikation til programmet, samt optegne og forklare forskellige modeller.

### 3.1 Kravspecifikation

Herunder ses en liste over krav til spillet, udformet ud fra opgavebeskrivelsen og det udleverede regelsæt.<sup>1</sup>

#### Alle krav stillet

1. Der skal oprettes forskellige typer felter, samt en spilleplade.
2. Hvert felt skal påvirke spillernes pengebeholdning forskelligt, og have en udskrift om hvilket felt han ramte.
3. Spillerne skal kunne lande på et felt, og fortsætte derfra på næste slag.
4. Spillerne skal gå i ring rundt på felterne, på spillepladen, med uret (man kan ikke gå mod uret)
5. Spillet skal kunne køre på DTU's databar-computere.
6. 2-4 spillere skal kunne spille spillet.
7. Den yngste spiller starter.
8. Hver gang man passerer start, modtager man M2 (penge).
9. Alt efter om der er 2, 3 eller 4 spillere, skal der gives forskellige start-penge.
10. Man skal have mulighed for at vælge hvilket figurkort man vil være.  
Figurkortet vil så have forskellig indflydelse ift. chancekort.
11. Lander en spiller på et ledigt felt, køber spilleren feltet og har nu ejerskab af feltet.
12. Lander en spiller på et ejet felt, betaler denne spiller husleje til den spiller der ejer feltet.

---

<sup>1</sup>Enkelte krav er hentet fra tidligere CDIO2-rapport af samme gruppe.



13. Ejer en spiller to ejendomme af samme farve er huslejen det dobbelte af det beløb, der står på feltet.
14. Der skal være tydeligtgjort på spillebrættet hvilke ejede felter, som hører til hvilken spiller.
15. Lander en spiller på et chancefelt, skal der vises en beskrivelse af en hændelse, samt en værdi i plus eller minus på spillerens konto.
16. Lander en spiller på et "Gå i fængsel-felt", flyttes spilleren til "Fængsel-feltet". Man får ikke  $M(\text{penge})$  ved passage af start.
17. Er en spiller i fængsel, kan man enten betale  $M1$  (penge), eller bruge et chancekort med løsladelses-mulighed. Mens man er i fængsel tjener man stadig leje.
18. Lander en spiller på "Fængsel-feltet", sker der intet.
19. Lander en spiller på "Gratis Parkering", sker der intet.
20. Spillet sluttet, når en spiller ikke har penge nok til at betale husleje, betale en ejendom, eller negativ værdi på et chance-kort.
21. Når spillet er slut, findes den der har flest penge, som så vinder.
22. Har flere spillere samme antal penge, når spillet er slut, tælles alle ejede ejendomsværdi'er, og lægges til pengebeholdningen.
23. (Avanceret) Hvis en spiller ikke har nok penge til at betale husleje eller en afgift fra et chancekort, skal gælden betales med dine ejendomme.
24. (Avanceret) Hvis en spiller skylder penge til en anden spiller, får den spiller dine ejendomme, skylder man banken penge, bliver ejendommene sat til salg igen.
25. (Avanceret) Hvis man skylder penge, og man ikke har nogen penge eller ejendomme, så er man fallit, og spillet slutter.

## MoSCoW

MoSCoW, er et værktøj som kan bruges til at prioritere krav. Herunder ses et eksempel på MoSCoW, og hvad de enkelte kategorier kan indeholde.<sup>2</sup>

<b>Mo</b>	"Must have"	De mest vitale krav, vi ikke kan undgå.
<b>S</b>	"Should have"	Vigtige krav, som ikke er vitale.
<b>Co</b>	"Could have"	The 'nice-to-haves'
<b>W</b>	"Won't have (this time)"	Things that provide little to no value you can give up on

Vi bruger herunder MoSCoW til at prioriterer vores krav, ved hjælp af MoSCoW.

<b>Must have</b>	<b>1.</b>	Der skal oprettes forskellige typer felter, samt en spilleplade.
	<b>2.</b>	Hvert felt skal påvirke spillernes pengebeholdning forskelligt, og have en udskrift om hvilket felt han ramte.
	<b>3.</b>	Spillerne skal kunne lande på et felt, og fortsætte derfra på næste slag.
	<b>4.</b>	Spillerne skal gå i ring rundt på felterne, på spillepladen, med uret (man kan ikke gå mod uret)
	<b>5.</b>	Spillet skal kunne køre på DTU's databar-computere.
	<b>6.</b>	2-4 spillere skal kunne spille spillet.
	<b>11.</b>	Lander en spiller på et ledigt felt, køber spilleren feltet og har nu ejerskab af feltet.
	<b>12.</b>	Lander en spiller på et ejet felt, betaler denne spiller husleje til den spiller der ejer feltet.
	<b>20.</b>	Spillet sluttet, når en spiller ikke har penge nok til at betale husleje, betale en ejendom, eller negativ værdi på et chance-kort.
	<b>21.</b>	Når spillet er slut, findes den der har flest penge, som så vinder.

<b>Should have</b>	<b>14.</b>	Der skal være tydeligtgjort på spillebrættet hvilke ejede felter, som hører til hvilken spiller.
	<b>15.</b>	Lander en spiller på et chancefelt, skal der vises en beskrivelse af en hændelse, samt en værdi i plus eller minus på spillerens konto.
	<b>16.</b>	Lander en spiller på et "Gå i fængsel-felt", flyttes spilleren til "Fængsel". Man får ikke M(penge) ved passage af start.
	<b>18.</b>	Lander en spiller på "Fængsel-feltet", sker der intet.
	<b>19.</b>	Lander en spiller på "Gratis Parkering", sker der intet.

<sup>2</sup>MoSCoW tabellen der vises, er hentet fra tidligere CDIO2-rapport af samme gruppe.

<b>Could have</b>	<b>7.</b>	Den yngste spiller starter.
	<b>8.</b>	Hver gang man passerer start, modtager man M2 (penge).
	<b>9.</b>	Alt efter om der er 2, 3 eller 4 spillere, skal der gives forskellige startpenge.
	<b>13.</b>	Ejer en spiller to ejendomme af samme farve er huslejen det dobbelte af det beløb, der står på feltet.
	<b>17.</b>	Er en spiller i fængsel, kan man enten betale M1 (penge), eller bruge et chance-kort med løsladelses-mulighed. Mens man er i fængsel tjener man stadig leje.

<b>Won't have</b>	<b>10.</b>	Man skal have mulighed for at vælge hvilket figurkort man vil være. Figurkortet vil så have forskellig indflydelse ift. chancekort.
	<b>22.</b>	Har flere spillere samme antal penge, når spillet er slut, tælles alle ejede ejendomsværdier, og lægges til pengebeholdningen.
	<b>23.</b>	(Avanceret) Hvis en spiller ikke har nok penge til at betale husleje eller en afgift fra et chancekort, skal gælden betales med dine ejendomme.
	<b>24.</b>	(Avanceret) Hvis en spiller skylder penge til en anden spiller, får den spiller dine ejendomme, skylder man banken penge, bliver ejendommene sat til salg igen.
	<b>25.</b>	(Avanceret) Hvis man skylder penge, og man ikke har nogen penge eller ejendomme, så er man fallit, og spillet slutter.

## FURPS+

FURPS+ er en metode til at kategorisere / klassificere krav.

Vi har i denne opgave brugt FURPS+ metoden til at kategorisere de krav, som er prioriteret i MoSCoW analysen. Vi har ikke valgt at medtage alle FURPS+ punkterne, men medbragt dem der giver mening for opgaven.

Herunder ses et eksempel på FURPS+, og hvad de enkelte kategorier kan indeholde.<sup>3</sup>

<b>F</b>	Functionality	Egenskaber, ydeevne, sikkerhed
<b>U</b>	Usability	Menneskelige faktorer, hjælp, dokumentation
<b>R</b>	Reliability	Fejlfrekvens, fejlretning, forudsigelighed
<b>P</b>	Performance	Svartider, nøjagtighed, ydeevne, ressourceforbrug
<b>S</b>	Supportability	Anvendelighed, tilpasningsevne, vedligeholdbarhed
<b>+</b>	Implementation: Interface:  Operations: Packaging: Legal:	Ressourcebegrænsninger, sprog og værktøjer, hardware Begrænsninger forårsaget af kommunikation med eksterne systemer Systemstyring i dets operationelle ramme F.eks. en fysisk boks F.eks. licenser

<sup>3</sup>FURPS+ tabellen der vises, er hentet fra tidligere CDIO2-rapport af samme gruppe.

Vi her herunder en oversigt over krav til opgaven, kategoriseret ved hjælp af FURPS+. Der er ikke medtaget de krav der er i "Won't have" kategorien i MoSCoW-analysen.

**Functionality:**

8. Hver gang man passerer start, modtager man M2 (penge).
9. Alt efter om der er 2, 3 eller 4 spillere, skal der gives forskellige start-penge.
11. Lander en spiller på et ledigt felt, køber spilleren feltet og har nu ejerskab af feltet.
12. Lander en spiller på et ejet felt, betaler denne spiller husleje til den spiller der ejer feltet.
13. Ejer en spiller to ejendomme af samme farve er huslejen det dobbelte af det beløb, der står på feltet.
15. Lander en spiller på et chancefelt, skal der vises en beskrivelse af en hændelse, samt en værdi i plus eller minus på spillerens konto.
16. Lander en spiller på et "Gå i fængsel-felt", flyttes spilleren til "Fængsel". Man får ikke M(penge) ved passage af start.
17. Er en spiller i fængsel, kan man enten betale M1 (penge), eller bruge et chancekort med løsladelses-mulighed. Mens man er i fængsel tjener man stadig leje.
18. Lander en spiller på "Fængsel", sker der intet.
19. Lander en spiller på "Gratis Parkering", sker der intet.
20. Spillet sluttes, når en spiller ikke har penge nok til at betale husleje, betale en ejendom, eller negativ værdi på et chance-kort.
21. Når spillet er slut, findes den der har flest penge, som så vinder. Har flere spillere samme antal penge, tælles alle ejede ejendomsværdi'er, og lægges til pengebeholdningen.

**Usability:**

6. 2-4 spillere skal kunne spille spillet.

**(+) - Interface:**

1. Der skal oprettes forskellige typer felter, samt en spilleplade.
14. Der skal være tydeligtgjort på spillebrættet hvilke ejede felter, som hører til hvilken spiller.
2. Hvert felt skal påvirke spillernes pengebeholdning forskelligt, og have en udskrift om hvilket felt han ramte.
3. Spillerne skal kunne lande på et felt, og fortsætte derfra på næste slag.
4. Spillerne skal gå i ring rundt på felterne, på spillepladen, med uret (man kan ikke gå mod uret)
7. Den yngste spiller starter.  
Figurkortet vil så have forskellig indflydelse ift. chancekort.

**(+) - Operations:**

5. Spillet skal kunne køre på DTU's databar-computere.

### Implementerede krav

Herunder ses de krav, der ved hjælp af MoSCoW er blevet udvalgt til at blive implementeret i programmet.

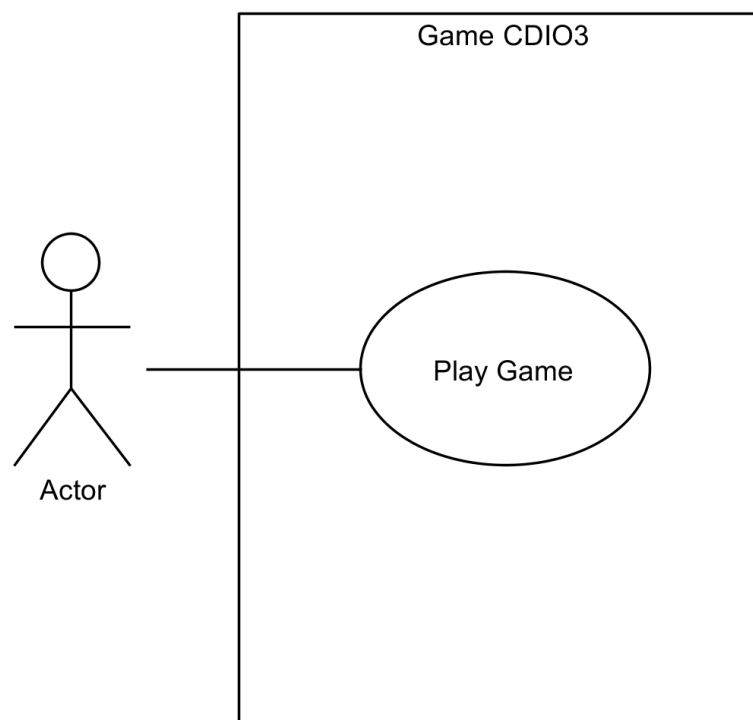
1.	Der skal oprettes forskellige typer felter, samt en spilleplade.
2.	Hvert felt skal påvirke spillernes pengebeholdning forskelligt, og have en udskrift om hvilket felt han ramte.
3.	Spillerne skal kunne lande på et felt, og fortsætte derfra på næste slag.
4.	Spillerne skal gå i ring rundt på felterne, på spillepladen, med uret. (man kan ikke gå mod uret)
5.	Spillet skal kunne køre på DTU's databar-computere.
6.	2-4 spillere skal kunne spille spillet.
7.	Den yngste spiller starter.
8.	Hver gang man passerer start, modtager man M2 (penge).
9.	Alt efter om der er 2, 3 eller 4 spillere, skal der gives forskellige start-penge.
11.	Lander en spiller på et ledigt felt, køber spilleren feltet og har nu ejerskab af feltet.
12.	Lander en spiller på et ejet felt, betaler denne spiller husleje til den spiller der ejer feltet.
13.	Ejer en spiller to ejendomme af samme farve er huslejen det dobbelte af det beløb, der står på feltet.
14.	Der skal være tydeligtgjort på spillebrættet hvilke ejede felter, som hører til hvilken spiller.
15.	Lander en spiller på et chancefelt, skal der vises en beskrivelse af en hændelse, samt en værdi i plus eller minus på spillerens konto.
16.	Lander en spiller på et "Gå i fængsel-felt", flyttes spilleren til "Fængsel". Man får ikke M(penge) ved passage af start.
17.	Er en spiller i fængsel, kan man enten betale M1 (penge), eller bruge et chance-kort med løsladelses-mulighed. Mens man er i fængsel tjener man stadig leje.
18.	Lander en spiller på "Fængsel", sker der intet.
19.	Lander en spiller på "Gratis Parkering", sker der intet.
20.	Spillet sluttet, når en spiller ikke har penge nok til at betale husleje, betale en ejendom, eller negativ værdi på et chance-kort.
21.	Når spillet er slut, findes den der har flest penge, som så vinder. Har flere spillere samme antal penge, tælles alle ejede ejendomsværdi'er, og lægges til pengebeholdningen.

## 3.2 Interessentanalyse

Interresent	Interesse / Mål
Spillere	Kunne styre et system, der styrer et spil mellem 2-4 personer, hvor i der kastes en terning. Man ser resultatet af dette slag med det samme. Herefter skal værdien af terningen af dette slag bestemme, hvilket felt spilleren landede på, og fortælle om feltet er ejet, eller frit til at købe. Man ender med at betale til en spiller, eller betale for at købe den pågældende grund.

## 3.3 Use case diagram

Use cases bliver lavet for at skabe overblik over det kommende program. Ligeledes har vi valgt at inddrage et use case-diagram for at illustrere, hvordan det ser ud for vores program. Use case-diagrammet er meget beskedent, idet der kun er én aktør, nemlig spilleren, som kun kan gøre én ting, nemlig at spille spillet.



Figur 3.1: Use case diagram

## 3.4 Use case

### Brief

Spillet starter med en dropdown menu hvorpå der angives antallet af deltagende spillere. Efterfølgende indtastes navn på spillerene og efterfulgt af endnu en dropdown menu med det formål at vælge den yngste spiller. Før spillets start tildeles de hver en pengebeholdning. Den yngste spiller starter og kaster derved med terningen. Terningen øje eller faceValue tages og bruges som udgangspunkt til bergening af hvor mange felter spilleren skal flytte sig. På GUI'en vises feltets oplysninger, samt hvorvidt det er ejet eller ej. Der slås igen med terningen, spilleren lander på et felt, og et bestemt scenarie tages i brug, afhængig af felt oplysningerne, hvor så samme procedure følger indtil en spiller erklæres bankerot. Der tælles point på spillerne, og spilleren med den største pengebeholdning vinder spillet.

### Fully dressed

Følgende er en beskrivelse af use casen 'playGame' i en fully dressed udgave.

<b>Use case:</b> PlayGame
<b>ID:</b> UC1
<b>Brief description</b> Spillere skal kunne spille spillet, altså playGame og slå med en terning.
<b>Primary actors:</b> Spillere
<b>Secondary actors:</b> Ingen.
<b>Preconditions:</b>
<b>Main flow:</b> <ol style="list-style-type: none"> <li>1. Spillet startes, antal spillere vælges (minimum 2 og maksimum 4 spillere) samt indtaster deres navn. Den yngste spiller specificeres, og er den første der slår med terningen.</li> <li>2. Spillet flytter spilleren afhængig af faceValue fra terningen.</li> <li>3. Feltets oplysninger udskrives til spilleren via GUI.</li> <li>4. Ny runde finder sted, og næste spiller kaster med terningen.</li> <li>5. Spillet giver nye ture indtil at en af spillerens pengebeholdning går i minus eller ikke har råd til købe/betaling af ejendom.</li> <li>6. Går en spillers pengebeholdning i minus, erklæres hermed spilleren bankerot, og spillet slutter</li> </ol>

**Postconditions:** Ingen.

**Alternative flow:**

1. **Spiller lander på ikke ejet felt**
  - (a) Spiller køber felt.
  - (b) Spiller bliver tildelt felt.
2. **Spiller lander på ejet felt**
  - (a) Spiller betaler ejeren af felt. Har ejeren mere end et felt, betales dobbelt beløb.
  - (b) Ejer modtager beløb.
3. **Spiller passerer start felt**
  - (a) Spiller får adderet 2M i sin pengebeholdning.
4. **Spiller lander på fængsel felt**
  - (a) Spiller flyttes til fængsel feltet.
  - (b) Spiller modtager ikke 2M ved passering af start feltet.
  - (c) Spiller betaler 1M efter løsladelse fra fængsel, eller ved brug af 'Du løslades uden omkostninger' kortet.

Tabel 3.1: Use case 1



## 3.5 Risicianalyse

For at skabe et overblik over de risici, som kan forekomme i udviklingsprocessen, oprettes der en risicitabel, som fortæller om risicien, dens påvirkning (skala 1-10, hvor 1 er mindst, 10 er højest) samt påvirkningsgrad (samme skala).

Der er udelukkende blevet oprettet en risicitabel over udviklingsprocessen og altså ikke de forskellige use cases, hvilket man i den agile udviklingsmetode, Unified Process, normalvis ville gøre brug af, da man ønsker at lave de højstprioriterede use cases indledningsvis.

### 1. Problemer med GUI.

Påvirkning = 10

Sandsynlighed = 8.

Det er klart, at spillet skal se godt ud og ikke skal køre i kommandoprompt/terminal. Vi arbejdede med GUI i CDIO2, men da der er kommet ny GUI, kan der også opstå nye problemer, derfor er det vigtigt at tage højde for dette i god tid og læse op på dokumentationen, om hvordan man bruger GUI'en.

### 2. Overtrædelse af deadline.

Påvirkning = 3

Sandsynlighed = 10.

Vi bruger Asana til at give hinanden lektier for og holde styr på, hvad vi mangler i projektet. Vi har strategisk valgt at sætte os til at lave tingene i god tid, således vi kan tage højde for eventuelle problemer, og det er derfor ikke et stort problem, hvis deadline bliver overtrædt (i begyndelsen).

### 3. Problemer med design af kode.

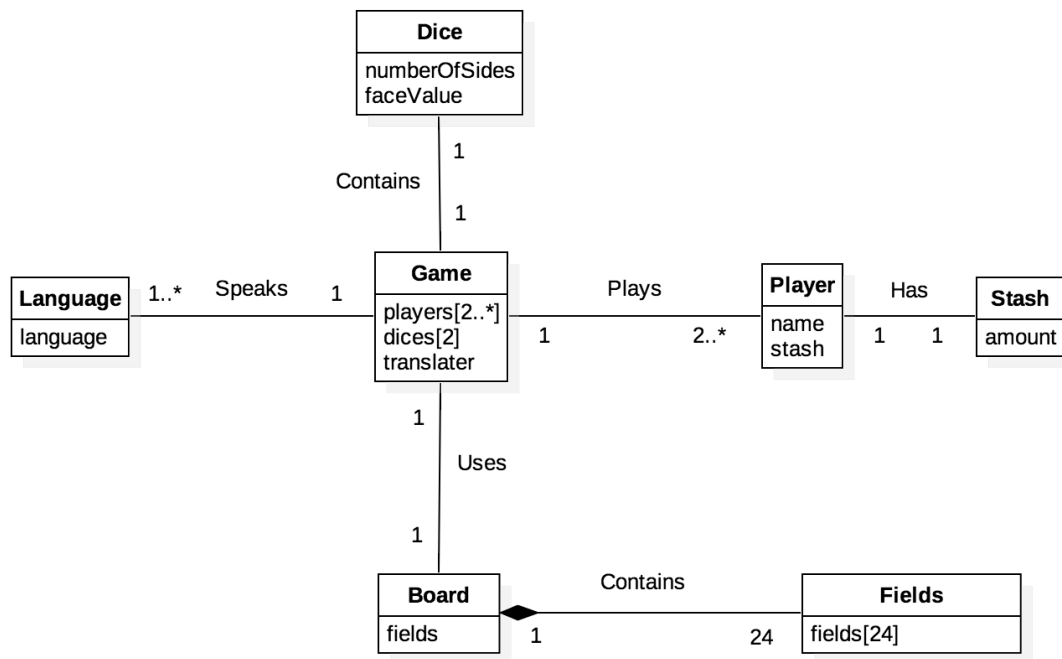
Påvirkning = 7

Sandsynlighed = 8.

Ligesom i de foregående projekter, kan man hurtigt komme til at følge en vandfaldsmodel, idet man tegner en masse diagrammer, men så hænger det ikke sammen med, hvordan spillet rent faktisk kommer til at fungere. Vi prøver til vores bedste evner at designe og kode parallelt.

De ovenstående situationer er blevet skrevet op for at gøre os opmærksomme på de potentielle vanskeligheder i udviklingsprocessen.

## 3.6 Domænemodel



Figur 3.2: Domænemodel

## 4 | Konfigurationsstyring

I dette afsnit berører vi konfigurationen af programmet. En beskrivelse af de forskellige krav, samt nødvendige programmer for at køre systemet. Dette inkluderer både kørsel af spillet, men også vedligeholdelse af programmet, med compiling, installation. Vi kommer samtidigt ind på hvorledes koden importeres fra et GIT repository. Konfigurationsstyring er delt op i to forskellige platforme.

### Udviklingsplatformen

I vores udviklingsplatform har vi brugt følgende software:

- Apple MacOS High Sierra & Microsoft Windows 10
- IntelliJ - Version 2017.2.6
- JAVA (jre-8u151 & jdk-9.0.1)
- matadorgui.jar (Hentet via Maven)<sup>1</sup>

### Produktionsplatformen

Produktionsplatformen er alt det software der skal bruges til at køre vores færdige program:

- Operativsystem der kan køre JAVA
- JAVA (jre-8u151 & jdk-9.0.1)

Skal man så videreudvikle på programmet skal man brug andre konfigurationer:

- Apple MacOS High Sierra & Microsoft Windows 10
- IntelliJ - Version 2017.2.6 (kan være anden IDE, dog er der medfølgende indstillinger i IntelliJ, som der er fordele i.
- JAVA (jre-8u151 & jdk-9.0.1)
- matadorgui.jar (Hentet via Maven)<sup>2</sup>

---

<sup>1</sup>[https://github.com/diplomit-dtu/Matador\\_GUI/raw/repository](https://github.com/diplomit-dtu/Matador_GUI/raw/repository)

<sup>2</sup>[https://github.com/diplomit-dtu/Matador\\_GUI/raw/repository](https://github.com/diplomit-dtu/Matador_GUI/raw/repository)

## 4.1 Konfigurationsvejledning

Følgende procedure er målrettet et Eclipse miljø, taget i betragtning at projektet skal kunne iscenesættes i Eclipse. Samtidigt skal projektet importeres fra et GIT repository, hvor så en vejledning for dette følger i dette afsnit.<sup>3</sup>

### Hente projekt fra repository, GIT

Det er flere måder at hente et projekt fra et GIT repository, hvor det i sidste ende afhænger af om man gør brug af en lokal installation eller web udgave af det omtalte GIT klient. I vores tilfælde havde vi gjort brug af GitHub til håndtering af vores GIT repository. Ved lokalisering af projektet på GitHub skal man blot klikke på 'Clone or download', herefter hentes en lokal kopi af det omtalte projekt. For at kunne åbne projektet i eksempelvis Eclipse, skal projektet blot importeres.

### Compiling, installering og afvikling af kildekoden, Eclipse

Eftersom at projektet er tilgængelig på maskinen lokalt, skal man blot importere projektet i Eclipse. Dette gøres ved at åbne Eclipse og lokalisere menuen 'File' på menubjælken. Under 'File' klikkes derefter på 'Import', og typen 'Projects from Folder or Archive' vælges. Mappen der indeholder vores lokale GIT kopi lokaliseres og sættes som kilde, og processen afsluttes vha. knappen 'Finish'.

### *Processen ser som følger:*

*File -> Import -> Projects from Folder or Archive -> Source: GIT kopi -> Finish*

For at kunne compile og afvikle programmet, skal man blot klikke på det ikon, der repræsenterer en trekant omringet af en grøn cirkel. Eventuelt kan man nøjes med at klikke lokaliserer menuen 'Run' på menubjælken.

---

<sup>3</sup> Da CDIO3 er blot en revideret udgave af CDIO2, er der tale om en magen konfigurationsvejledning til den fundet i CDIO2, og som resultat, er følgende vejledning baseret på vejledningen fra CDIO2 rapporten.

## 4.2 Vejledning til eksport samt kørsel af program

Følgende er en vejledning til hvordan man compiler en såkaldt .jar file (java archive) ved brug af Eclipse. Resultatet vil da være en java fil der kan afvikles via eksempelvis terminal. Selve kildekoden pakkes ind i en .jar fil.

Primært er der to måder at compile en runnable jar fil, enten ved brug af terminal/commando prompt eller via eksport funktionen i Eclipse. Vi vælger at tage udgangspunkt i det sidst nævnte, og vejledningen ser som følger:

### Eksport af program

1. Åbn projekt i Eclipse og lokaliserer menuen 'File' på menubjælken.
2. Vælg derefter eksport funktionen under menuen 'File'.
3. Find 'Java' eksport konfigurationen, vælg derefter 'Runnable JAR file'.
4. I det ny opstået vindue vælges opstarts konfigurationen, hvori der specificeres hvilken 'Java Application' der skal opfattes som start.
5. I samme vindue vælges også sti for eksport destination. Derudover skal der også tages et valg i forhold til hvordan biblioteket skal behandles.
6. Ved tryk på 'Finish' eksporter Eclipse med udgangspunkt i den valgte konfiguration projektet til en runnable jar fil.

### Kørsel af program

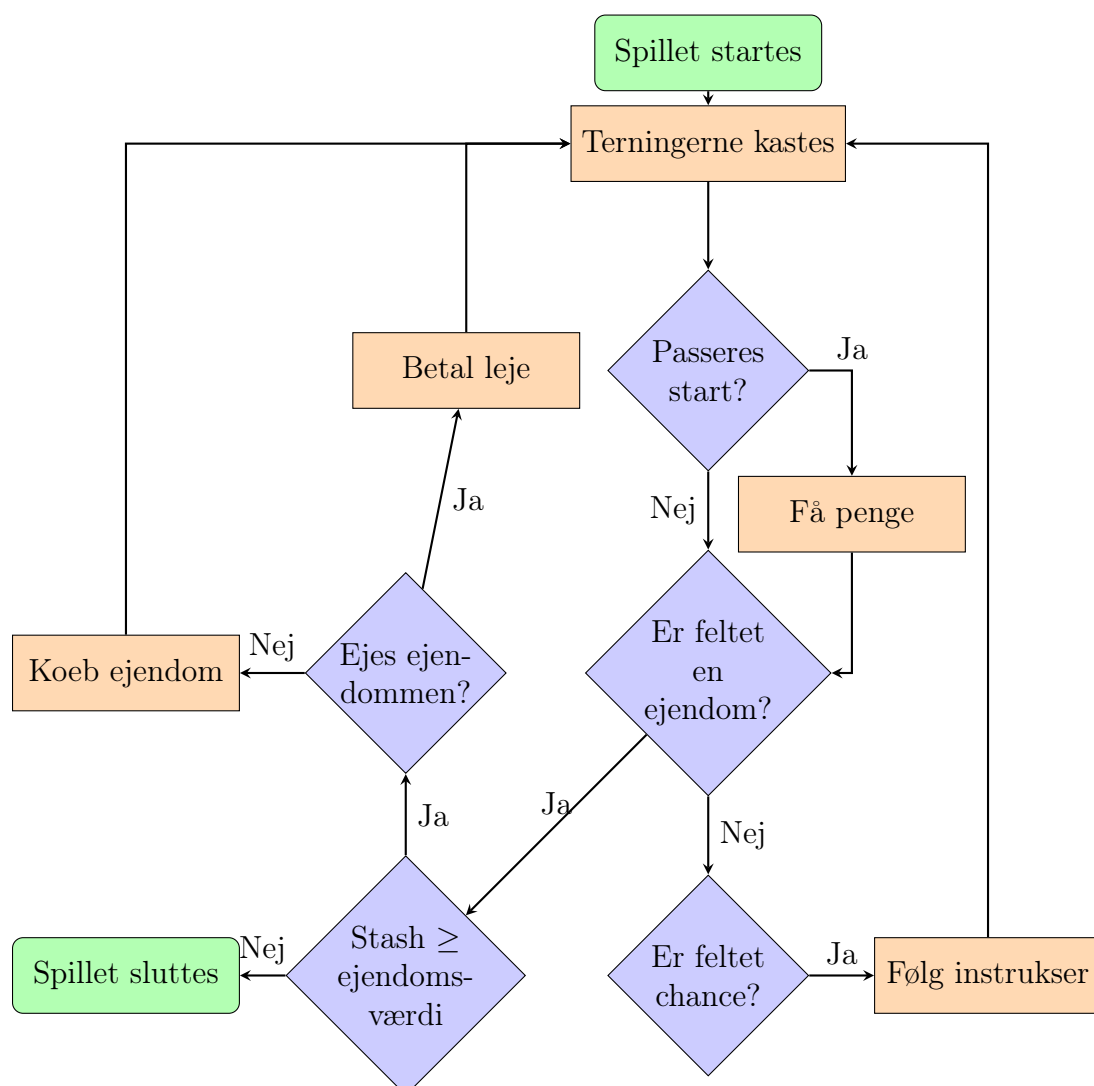
1. For at åbne/afvikle den tidligere eksporteret jar fil skal man blot køre programmet i terminal/commando prompt.
2. I terminalen skrives følgende: *java -jar 'navn på jar-fil'.jar*.
3. Herefter burde programmet hvis alt går vel starte op.

## 5 | Design

I dette kapitel vil vi gennemgå spillets design, og vi vil vise et flow-, klasse- og system sekvensdiagram.

### 5.1 Flowdiagram

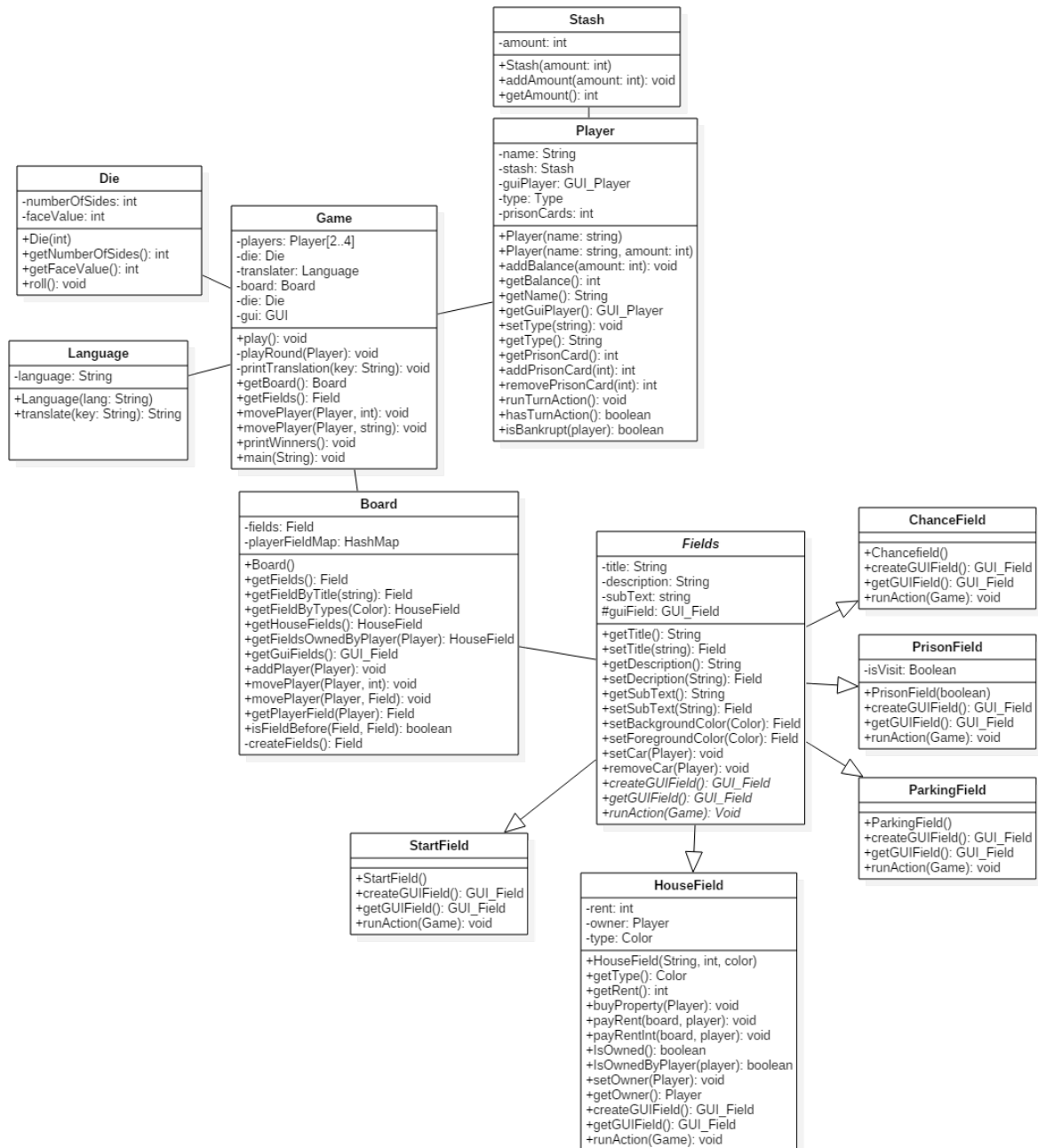
Indledningsvis tegnede vi et flowdiagram for at skabe overblik over projektet, så vi kunne forstå, hvordan spillet skulle udføres. Flowdiagrammet blev brugt som baggrund for de efterfølgende diagrammer, som bliver vist i projektet.



## 5.2 Klassediagram

Herunder ses klassediagrammet for vores spil.

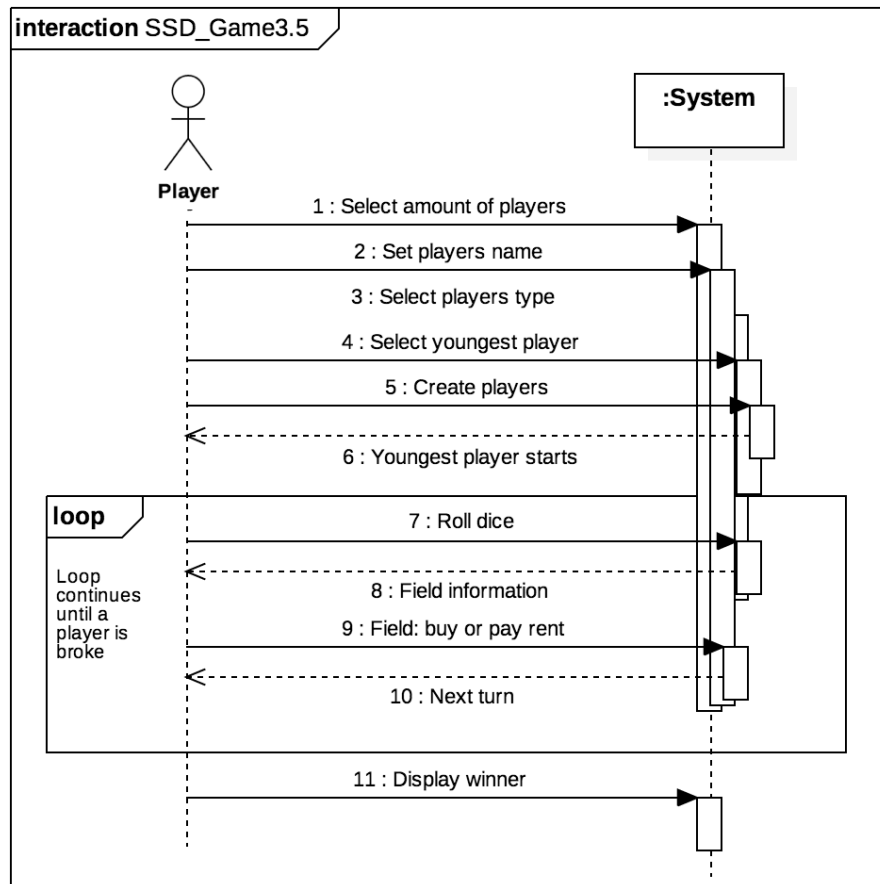
Et computergenereret klassediagram, kan ses i bilag.



Figur 5.1: Klassediagram

## 5.3 Systemsekvensdiagram

Et systemsekvensdiagram viser typisk en form for simplificeret tidslinje over flowet i programmet. Følgende diagram viser interaktionen mellem aktøren spiller og system.



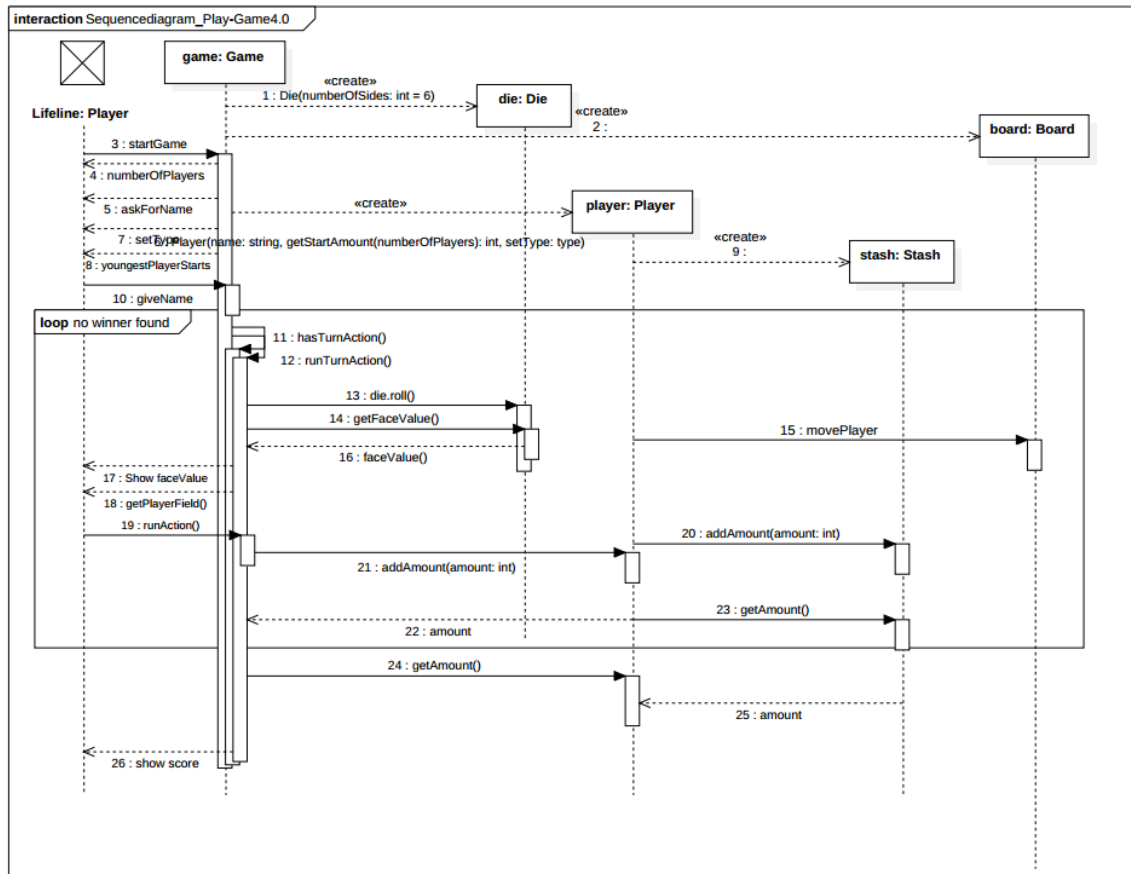
Figur 5.2: Systemsekvensdiagram

Som start bliver der spurgt om antal af spillere, og da dette udføres visuelt via en dropdown/user selection, udtrykkes denne handling som 'select'. Eftersom antal af spiller vælges, skal navne på spillere tastes ind. Spiller type samt specificering af den yngste spiller gøres på præcis samme måde som i første trin, altså user selection. Før spillets start tildeles de hver en pengebeholdning afhængig af antallet af spillere. På baggrund af trin 6 bliver den yngste spiller den første der starter runden og kaster med terningen. Roll dice ruller terningen hvor så faceValue afgøre hvilket felt spilleren lander på. På GUI'en vises feltets oplysninger, samt hvorvidt det er ejet eller ej. Ejes feltet skal der betales leje, ejes det ikke skal det købes. Efterfølgende er der ekstra tur, vha. 'next turn', hvilket slår igen med terningen, spilleren lander på et felt, og et bestemt scenarie tages i brug, afhængig af felt oplysningerne, hvor så samme procedure følger indtil en spiller erklæres bankerot. 'Display winner' tæller point på spillerne, og spilleren med den største pengebeholdning vinder spillet,



hvilket bliver vist på GUI.

## 5.4 Sekvensdiagram



Figur 5.3: Sekvens diagram

## 6 | Implementering

Vi har i dette projekt taget udgangspunkt i CDIO2, som er det første projekt, hvor vi bl.a. virkelig har gjort brug af planlægningsværktøjet Asana og kommunikationsværktøjet Slack. I det her projekt var det nødvendigt for os at programmere, sideløbende med en analyse- og designfase. Det viste sig at være en kæmpe hjælp at have et klassediagram, mens man koder, fordi man på klassediagrammet har en idé om, hvordan koden skal se ud.

Det er dog klart, at diagrammerne bliver ændret i løbet af kodningen, fordi det er irrationelt at tro, at uforudsete ting ikke kan forekomme. I 'Udviklingsmetoder til IT-Systemer' har vi altså lært ikke at følge en vandfaldsmodel, men en agil model, hvor projektet deles op i iterationer, som det vises i kapitlet *Projektplanlægning*.

Vi har prøvet til vores bedste evner at følge GRASP-modellen, således vi får en attraktiv low coupling og high cohesion.

### 6.1 Brugervejledning for programmet

Programmets brugerinterface er meget simpelt sat op, og guider brugeren nemt igennem hvad han/hun skal gøre. Når man starter programmet op, vil det åbne en GUI, hvor den vil starte med at spørge, hvor mange spillere, der skal spille spillet (spillet tager imod 2, 3 og 4 spillere). Dette indtaster brugeren og trykker herefter på ok, hvorefter spillet spørger om spiller 1's navn og ønsker derefter at vide, hvilken *type* spiller, han/hun er. Det samme gør sig gældende for de næste spillere i samme rækkefølge.

Spillet fortæller nu, at det er spiller 1's tur, hvorefter man skal trykke på knappen *kast*, således at der kommer en visualisering af et terningekast og et tekstfelt med terningens værdi. Herefter vil programmet opdatere spillerens score, hvis altså han/hun har købt en grund eller skal betale leje, og give turen videre. Dette fortsætter programmet med, indtil der kun er én spiller med en positiv pengebeholdning. Brugeren kan desuden holde musen over de forskellige felter, for at få beskrivelsen af feltet, og dens leje.

## 6.2 Verificering ift. krav og validering ift. kunde

I dette afsnit vil der forekomme en mindre diskussion ift. verificering af at produktet lever op til de opstillede krav, samt validering af krav fundet i oplæget og hvorvidt de stemmer over ens med kundens ønsker.

### Verificering ift. krav

Verificering i forhold til krav tager udgangspunkt i kravspecifikationen fundet under afsnittet analyse, og tager hånd om på de forskellige opstillede krav, og hvorvidt produktet lever op til disse krav. Udfra det dannes et helheds billede over opfyldte krav, og er der evt. mangler noteres disse.

Generelt så har vi bestræbet os efter at få opfyldt det største antal krav muligt. I alt er der tale om 25 krav, hvori de essentielle krav for at spillet følger reglerne blev prioriteret. På baggrund af mangel på tid samt større besværligheder har det slet ikke været muligt, at implementerer de mere avancerede krav, nemlig krav nr. 22, 23, 24 samt 25. Størstedelen af kravene der rækker fra 1-21 har vi prøvet så vidt som muligt at implementere, dog skal det tages i betragtning at der muligvis er et par krav der ikke blev implementeret. Jævnfør afsnittet 'Implementerede krav' under kravspecifikation.

### Validering ift. kunde

Som udtrykt i spillereglerne fra Matador Junior, er der tale om andre regler end dem man ellers bruger i et normalt matador. Kunden ønsker et Matador Junior spil, hvor det forrige del bliver udbygget med forskellige typer af felter, samt en decideret spilleplade. Derudover skal spillerne kunne lande på et felt og så fortsætte derfra på næste slag. Spillerne går i ring på brættet, hvorpå antal spillere må ikke være mindre end 2 og større end 4.

IOOuterActive mener at spillet lever op til kundens forventninger. Et matadorspil der følger reglerne fra Matador Junior, samt lever op til kundens krav er blevet udviklet.

## 6.3 Dokumentation af kode

### Forklaring af Arv

I programmeringssprog betyder nedarvning, at en klasse udvider en anden klasse, den kan altså bruge super klassens metoder. Dette gør det mere overskueligt at holde styr på sine forskellige klasser, og det gør det også nemmere, da man ikke skal skrive alle de nedarvede metoder igen, for hver subklasse man vil oprette. Et eksempel på en nedarvning, er et fartøj. Her vil det være muligt at lave forskellige nedarvninger på denne superklasse, f.eks. køretøj og luftfartøj. Man kan altså bruge nedarvning når ens subklasser har en relation til superklassen. Et andet eksempel, som vi gennemgik i klassen, er personer på DTU. Dette kan jo opdeles i mange subklasser, da vi både har studerende og ansatte, derudover har vi nogle undergrene til de ansatte, der specificerer præcist hvad for en ansat de er, og hvilke rettigheder de skal have til systemet, alt efter hvilken ansat de er. Her er nedarvning perfekt, da man slipper for at skrive meget af den samme kode igen.

### Forklaring af abstract

En abstract klasse er en klasse hvor man tillader at der er metoder der ikke bliver implementeret, med dette menes der at disse metoder vil blive implementeret i subklasserne, således at den abstrakte klasse sender metoden videre til den klasse, der kan håndtere den information, den bliver kaldt på. Dette vil altså sige, at hvis man opretter en abstract metode, så skal denne metode også indgå i alle subklasserne. Et eksempel er her i vores CDIO3 rapport, benytter vi os bl.a. af en abstract metode, der hedder runAction. Denne abstrakte metode ligger i vores superklasse Field, når man så kalder runAction på et objekt 'field', så vil den sende denne metode videre til den respektive subklasse, hvis felt den blev kaldt på. Det vil altså sige, at hvis den bliver kaldt på et HouseField, så vil metoden runAction blive kørt som runAction metoden i HouseField, hvorimod hvis den bliver kaldt på et PrisonField, så vil metoden runAction blive kørt som runAction metoden i PrisonField. Dette gør koden meget nem overskuelig, og nem at videreudvikle på, da man bare kan gå ind i de respektive metoder man vil ændre.

### Fortæl hvad det hedder hvis alle fieldklasserne har en landOnField metode der gør noget forskelligt

Dette ville være en abstract metode, ligesom den vi har beskrevet ovenfor. En sådan abstract metode vil ikke blive implementeret i superklassen, det vil altså sige, at superklassens metode vil blot lyde på 'public abstract void landOnField(parameter)', hvor parameteret kan være f.eks. en spiller eller et game objekt. Herefter ville man så override metoden til alle subklasserne, og herefter implementerer hvad hvert felt skal gøre. Dermed vil den, når man kalder metoden på et felt, sende metoden videre til den respektive subklasse, hvis felt den blev kaldt på.

## Dokumentation for overholdt GRASP

GRASP er en række guidelines, man kan følge for at udvikle objektorienteret med bevidst klasseansvar. Den inddeler klasser i funktioner såsom controller (tager imod inputs fra spiller) og creator (skaber objekter) samt henviser til, at man skal opnå lav kobling og høj samhørighed.

Vi følger til dels GRASP-modellen, idet vi har fordelt ansvaret i mange klasser, som hver har deres ansvarsområde eksklusivt vores *Controller*, Game-klassen, som også agerer som *Creator*.

En enkelt runde i køres som en metode `playRound()`, som er i en do-while-løkke, som kører, indtil en spiller er bankerot.

## Java-doc

Vi bruger javadoc til at generere en dokumentation. Vores javadoc kan findes på <https://group-18.github.io/CDIO3/>, hvor næsten hver metode er blevet beskrevet. Måden hvorpå en javadoc genereres i Java IDE er blevet illustreret forinden:

```
1  /**
2   * Get the balance of a player.
3   *
4   * @return the balance of the player.
5   */
```

## 7 | Test

Overordnet set kan man inddele tests i henholdsvis black box test og white box test.

Black box test, også kendt som funktionel test, handler om, hvorvidt systemet fra et eksternt perspektiv opfører sig som forventet. Det smarte ved denne test er, at man kender input og output og alt det der sker i systemet er sagen uvedkommen for testeren, og derfor kræver det ikke indblik i koden for at kunne black box teste.

White box test, også kendt som strukturel test, handler derimod om, hvorvidt systemet fra et indre perspektiv opfører sig som forventet. Man vælger i white box test at have fokus på programmets logik og dette kan være med til at skabe en høj kodedækning, da man eksempelvis tester, om alle forgreninger kan udføres.

Til vores program har vi valgt at lave tre typer tests; unit test, kodedækningstest og monkey test. Det skal hertil nævnes, at der i projektet er blevet udført integrationstest, da vi ikke bare har valgt en *big bang-approach* og først testet systemet til sidst.

### 7.1 Unit test

Unit test går i sin helhed ud på at teste de enkelte metoder i spillet, det er altså derfor en white box test.

Vores unit tests er dokumenteret i koden, således man kan se opbygningen på hvert enkelte test. Nedenstående uddrag af vores kode beskriver en unit test af `getAmount()` og `startAmount()`:

```
1      @Test
2      public void getAmount() throws Exception {
3          Stash stash1 = new Stash();
4          Stash stash2 = new Stash(100);
5
6          Assert.assertEquals(0, stash1.getAmount());
7          Assert.assertEquals(100, stash2.getAmount());
8      }
```

Først oprettes `stash1` af klassen `Stash`, og da den ikke initialiseres med en værdi, så forventer vi, at `getAmount()` returnerer en værdi på 0. Ligeledes når `stash2` bliver oprettet, så bliver den initialiseret med 100, og derfor forventes der også en returnværdi på 100, når `getAmount()` kaldes.

## 7.2 Coverage test

Spillet er blevet kodedækket på to forskellige måder. Den første måde gik ud på, at vi kørte det normale spil med fire spillere, og vi fik kodedækningsprocent på: Klasser brugt = 100%, metoder brugt = 78%, linier brugt = 79%. Vi var imidlertid ikke tilfredse med det her resultat, og vi kunne se, at den lave kodedækning til dels hang sammen med, at man simpelthen ikke nåede at bruge alle metoder.

Vi valgte derfor at ændre bankerot-værdien således, at man skulle have en beholdning på -500, før man tabte. Dette gjorde, at spillet kørte i langt længere tid, og resultatet endte med at blive: Klasser brugt = 100%, metoder brugt = 85%, linier brugt = 82%. Dette resultat godtogede vi, fordi det viste sig, at mange af metoderne hang sammen med en ikke-implementeret metode, der hang sammen med typekortene, som i dette projekt er blevet udkommenteret.

## 7.3 Monkey test

Monkey test (dansk: abetesten) er en black box test, hvor man sådan set bare prøver at taste en helt masse og 'stresse' spillet. Det er en smart metode til at finde fejl ved programmet, og vi har hyppigt brugt den, fundet fejl og undersøgt dem via debug-funktionen.

## 7.4 Testcases

Vi har udarbejdet en række test cases ud fra kravende som er specificeret i kapitel 3.1. Disse test cases bekræfter at kravende er overholdt og programmet kan derfor godkendes bl.a. ud fra disse tests.

<b>Unique ID</b>	TC1
<b>Summary</b>	Kan programmet modtage de korrekte navne?
<b>Requirements</b>	
<b>Preconditions</b>	
<b>Postconditions</b>	
<b>Procedure</b>	<ol style="list-style-type: none"><li>1. Start spillet</li><li>2. Indtast navnet "Adam" og tryk ok</li><li>3. Indtast navnet "Eva" og tryk ok</li></ol>
<b>Test data</b>	<ul style="list-style-type: none"><li>• Navn 1: Adam</li><li>• Navn 2: Eva</li></ul>
<b>Expected result</b>	Spillerne er blevet oprettet med navn og vises på brættet
<b>Actual result</b>	Som forventet
<b>Status</b>	Godkendt
<b>Tested by</b>	Oliver Storm Køppen
<b>Date</b>	1/12-2017
<b>Environment</b>	IntelliJ on Win10



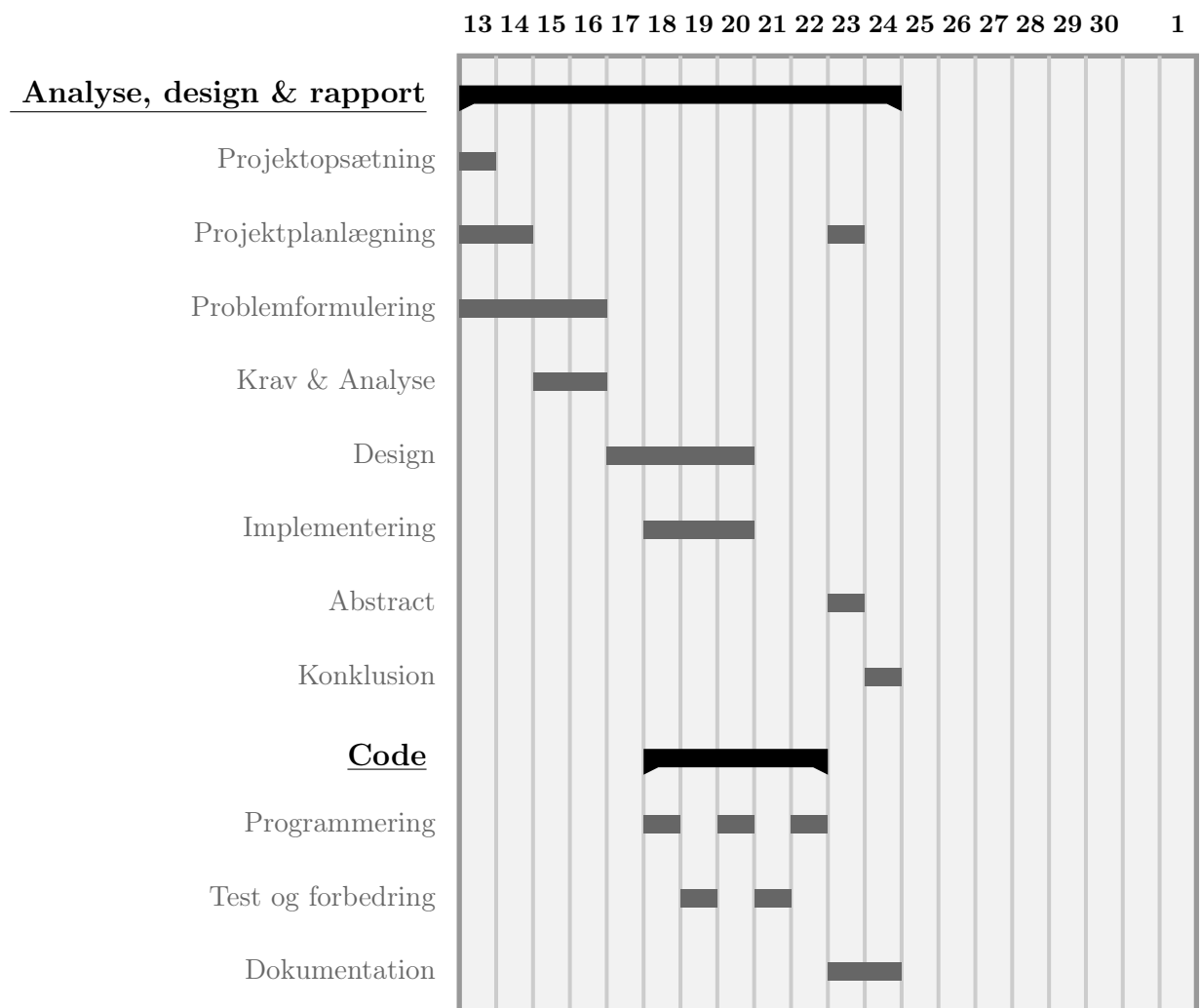
<b>Unique ID</b>	TC2
<b>Summary</b>	Afsluttes spillet, når en spiller har en beholdning på $<0$ ?
<b>Requirements</b>	
<b>Preconditions</b>	To spillere er blevet oprettet og en af spillernes beholdning er på $<0$
<b>Postconditions</b>	Spillet afsluttes
<b>Procedure</b>	<ol style="list-style-type: none"> <li>1. Start spillet</li> <li>2. Spillerne er oprettet</li> <li>3. Spillet køres, indtil den ene spiller har en beholdning på <math>&lt;0</math></li> </ol>
<b>Test data</b>	<ul style="list-style-type: none"> <li>• Navn 1: "Adam"</li> <li>• Navn 2: "Eva"</li> </ul>
<b>Expected result</b>	Adams er gået fallit.
<b>Actual result</b>	Som forventet
<b>Status</b>	Godkendt
<b>Tested by</b>	Oliver Storm Køppen
<b>Date</b>	1/12-2017
<b>Environment</b>	IntelliJ on Win10

<b>Unique ID</b>	TC3
<b>Summary</b>	Ryger man i fængsel, hvis man har et fængselskort?
<b>Requirements</b>	Spiller er oprettet, og spilleren har et fængselskort
<b>Preconditions</b>	Spilleren er landet på fængselsfeltet
<b>Postconditions</b>	Spilleren kan spille videre uden at komme i fængsel
<b>Procedure</b>	<ol style="list-style-type: none"> <li>1. Start spillet</li> <li>2. Indtast navnet "Adam" og tryk ok</li> <li>3. Adam har modtaget et fængselskort</li> <li>4. Adam lander på fængselsfeltet</li> </ol>
<b>Test data</b>	
<b>Expected result</b>	Spilleren fortsætter spillet uden at havne i fængsel.
<b>Actual result</b>	Som forventet
<b>Status</b>	Godkendt
<b>Tested by</b>	Niklaes Jacobsen
<b>Date</b>	30/11-2017
<b>Environment</b>	IntelliJ on MacOS

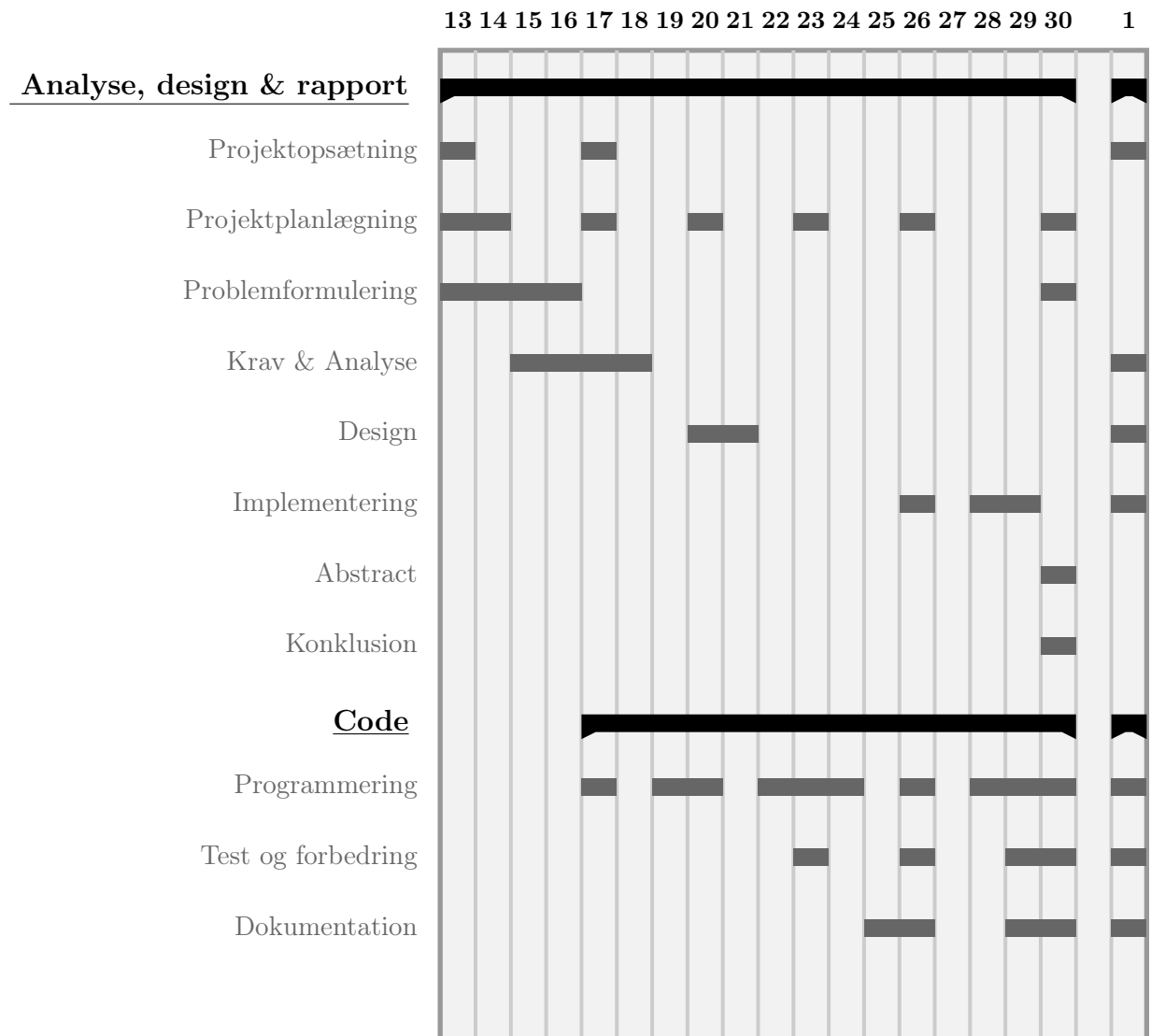
<b>Unique ID</b>	TC4
<b>Summary</b>	Vil spillerens pengeholdning ændrer sig i korrespondance til feltets værdi?
<b>Requirements</b>	
<b>Preconditions</b>	TC1 og spilleren landet på et felt med værdi, som endnu ikke er ejet
<b>Postconditions</b>	Spillerens pengebeholdning er nu blevet mindre i relation til køb af felt
<b>Procedure</b>	<ol style="list-style-type: none"> <li>1. Start spillet</li> <li>2. Spillere oprettes</li> <li>3. Spilleren lander på Isbutikken</li> </ol>
<b>Test data</b>	<ul style="list-style-type: none"> <li>• Spillers startbeholdning er 20</li> <li>• Isbutikken: M2</li> </ul>
<b>Expected result</b>	Spillernes pengebeholdning er nu 18
<b>Actual result</b>	Som forventet
<b>Status</b>	Godkendt
<b>Tested by</b>	Oliver Storm Køppen
<b>Date</b>	1/12-2017
<b>Environment</b>	IntelliJ on Win10

## 8 | Projektplanlægning

Diagrammet herunder er inddelt i dage indtil aflevering indelt i 19 dage.  
Diagrammet viser reelle datoer fra d. 13 november 2017 til og med d. 1 december 2017. Diagrammet viser den planlagte tid, vi lagde fra start af projektet.



Diagrammet herunder er inddelt i dage indtil aflevering indelt i 19 dage.  
Diagrammet viser reelle datoer fra d. 13 november 2017 til og med d. 1 december 2017. Diagrammet viser den reelle tid brugt på projektet.



## 9 | Konklusion

Dette projekt har bragt os i større udfordringer end noget andet projekt. Et grundigt analysearbejde, gav os et godt grundlag for det vi skulle til at udvikle. Vi havde en klar plan om at være færdig ca. en uge før vi skulle aflevere. Desværre startede vi ud med at producere spillet, med den nye version af GUI'en, og en stor del af alt vores gamle kode fra CDIO 2. Dette resulterede i at vi måtte starte på ny med vores kode, for at kunne skabe nogle metoder og variabler, som vi kendte. Vi ville til at starte med gerne have flere funktionaliteter med, men pga. vi startede forfra på kode, har tiden har været en væsentlig faktor i produktionen.

Vi mener at vi har levet op til opgavens mindstekrav, samt øget den funktionalitet der er på spillet, til et fornuftigt niveau. Kvaliteten i produktet skyldes også det grundige arbejde, i analyse- og design-fasen.

Denne opgave har helt klart været en læringsprocess for hele gruppen. Vi har lært hvordan vi, i et fremtidigt projekt skal håndtere ligendene opgaver. Dette inkluderer kombineret af flere projekter / programmer, samt opdateringer i eksterne systemer.

# A | Bilag


## A.1 Reglementet til Matador Junior

Herunder ses de officielle regler til Matador Junior-spillet, som vi har baseret vores spil på:


**OPSÆTNING!**

1. Åbn spillebrættet, og anbring det mellem spillerne.
2. Tag chancekortene, og fjern de 4 figurkort.  
De skal ikke bruges i spillet. Bare læs dem, og vælg!


De 4 figurkort:



Hvem vil du være?




3. Placer din brik på START!
4. Find de 12 "Solgt"-skilte, der matcher din brik, og læg dem foran dig.



5. Bland de 20 chancekort, og læg dem med billedsiden nedad på feltet til chancekort på spillebrættet.
6. Vælg en spiller til at være bankør. Bankøren passer på pengene.  
Så skal der deles penge ud:

For 2 spillere: Giv hver spiller #20  
For 3 spillere: Giv hver spiller #18  
For 4 spillere: Giv hver spiller #16



<sup>1</sup>MONOPOLY JUNIOR, Hasbro, 2013, [www.hasbro.dk](http://www.hasbro.dk).

## SPIL!

### Sådan vinder du

Vær den spiller, der har flest penge, når en anden spiller går fallit (ikke har råd til at betale husleje, købe en ejendom eller betale en afgift fra et chancekort).

### Sådan spiller du

1. **Den yngste spiller starter!** Spillet fortsætter mod venstre.
2. **Kast terningen, og flyt din brik det antal felter, som øjnene viser, med uret rundt på brættet fra START.**
  - Ryk **altid** frem, aldrig tilbage.
  - **Hver gang** du passerer eller lander på START, modtager du **42!**
3. **Hvor landede du?** Læs alt om de forskellige felter, før du starter.

Hvis du lander på:



#### ET LEDIGT FELT

Hvis ingen ejer det, skal du købe det!

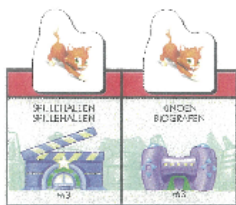
- Betal banken det beløb, der står på feltet.
- Placér et af dine "Solgt"-skilte på det farvede bånd øverst på feltet, så alle kan se, at du ejer det.



#### ET EJET FELT

Hvis en anden spiller ejer feltet, skal du betale husleje til den spiller. Huslejen er det beløb, der står på feltet.

Hvis du selv ejer det, skal du ikke gøre noget.








#### ET PAR = DOBBELT HUSLEJE!

Hvis en spiller ejer begge ejendomme i samme farve, er huslejen det dobbelte af det beløb, der står på feltet!

4. Vend for at se oplysninger om de øvrige felter på brættet og om, hvordan du vinder!



## BRÆTTETS FELTER

**START**  
Modtag #2 fra banken, hver gang du passerer eller lander på START.

**CHANCE**  
• Tag det øverste chancekort, læs det højt, og følg instruktionerne.  
• Læg brugte kort tilbage nederst i bunken.

**GÅ I FÆNGSEL**  
Gå lige i fængsel! Du passerer ikke START. Du modtager ikke #2. I starten af din næste tur skal du betale #1 eller bruge "Du løslades uden omkostninger"-kortet, hvis du har det. Kast derefter terningen, og ryk som normalt. Du kan godt modtage husleje, mens du er i fængsel.

**PÅ BESØG**  
Hvis du lander her, er du bare på besøg.

**GRATIS PARKERING**  
Du behøver ikke gøre noget, bare snup dig en pause.

**5. Det var det!** Så er det næste spillers tur.

## VIND!

1. Hvis du ikke har penge nok til at betale husleje, købe en ejendom, som du lander på, eller betale afgiften fra et chancekort, er du gået falit! Og så er spillet slut.
2. De andre spillere tæller deres penge, og den, der har flest, har VUNDET!
3. Uafgjort? Tæl, hvor meget dine ejendomme er værd, og læg det til dine penge!

## AVANCERET!

Når du har lært de grundlæggende regler, kan du prøve at spille på følgende måde og se, hvem der vinder.

1. Hvis du ikke har penge nok til at betale husleje eller en afgift fra et chancekort, skal du betale gælden med dine ejendomme.
2. Hvis du skylder en anden spiller penge, får den spiller dine ejendomme. Hvis du skylder banken penge, bliver dine ejendomme sat til salg igen.
3. Hvis du stadig ikke kan betale, er du gået falit, og spillet slutter. Alle tæller deres penge for at se, hvem der har vundet!

HASBRO GAMING- og MONOPOLY-navnet og -logoet, spillebrædets særegne design, de 4 hjerne- eller MR. MONOPOLY-navnet og -figurer samt de særegne elementer på spillebrættet og spillekortene er varemærker, der tilhører Hasbro for dette ejendomsforvaltnings spil og spiludstyr.  
© 1935, 2013 Hasbro. Alle rettigheder forbeholdt.



Producent af: Hasbro SA, Rue Emile-Boeldhat 31, 2630 Delémont CH.  
Repræsenteret af: Hasbro Europe, 4 The Square, Stockley Park, Uxbridge, Middlesex, UB11 1ET, UK.

Hasbro Nordic Consumer Services:  
Hasbro Denmark, Eby Industivej 40, 2500 Glostrup, Danmark. 43 27 01 00. hasbrodk@hasbro.dk

Godevardele oplysninger som referens.  
Farver og indhold kan variere fra det illustrerede.

[www.hasbro.dk](http://www.hasbro.dk)

128A69841080

1113A6984108-108

