

---

# CDIO 2

Gruppe 6

---

Lykke Flor Andersen  
s221694



Anthon Hertz Bie  
s224290



Sara Barslund  
s221703



Emil S. Simonsen  
S193965



Lærke Jull Kragskov  
s215714



28. oktober 2022

## Resumé

### Timeregnskab

#### Oversigt over arbejdsmål i løbet af ugen:

Mandag d. 10/10	Tirsdag d. 11/10	Onsdag d. 12/10	Torsdag d. 13/10	Fredag d. 14/10
Spørgsmål og svar til projektlederne	Opstart og overblik over analyser	Analyse	Analyse	Start på kode

Mandag d. 24/10	Tirsdag d. 25/10	Onsdag d. 26/10	Torsdag d. 27/10	Fredag d. 28/10
Sidste del af koden laves færdigt og små fejl rettes	Code freeze Fokus på rapport	Færdiggør rapport	Korrekturlæsning	Projektet afleveres

#### Oversigt over timefordeling:

Krav: 20 min.

Spg. til kunde og projektleder: 15 min.

Projektplanlægning: 30 min.

Kodning i alt: 7 timer.

Diagrammer og analyse: 5 time.

Test: 3 time.

Andre indhold til rapport: 3 timer.

# Indholdsfortegnelse

<b>Timeregnskab</b>	<b>1</b>
<b>Indholdsfortegnelse</b>	<b>2</b>
<b>1. Indledning</b>	<b>3</b>
<b>2. Projektplanlægning</b>	<b>3</b>
<b>3. Krav</b>	<b>3</b>
<b>4. Analyse</b>	<b>4</b>
<b>4. 1 Use case diagram</b>	<b>4</b>
<b>4. 2 Use case beskrivelser</b>	<b>5</b>
<b>Brief</b>	<b>5</b>
<b>Fully dressed</b>	<b>5</b>
<b>4.3 Domænemodel</b>	<b>7</b>
<b>4. 4 Systemsekvensdiagram</b>	<b>8</b>
<b>5. Design</b>	<b>8</b>
<b>5.1 Design klassediagram</b>	<b>9</b>
<b>5.2 Design sekvensdiagram</b>	<b>10</b>
<b>6. Test</b>	<b>10</b>
<b>7. Konfiguration</b>	<b>11</b>
<b>8. Konklusion</b>	<b>11</b>
<b>9. Bilag</b>	<b>11</b>

## 1. Indledning

Nedenstående rapport har til formål at give et indblik i de processer der har ført til det endelige program. Her er der blandt andet blevet anvendt relevante analyseværktøjer til at skabe et overblik over grundlaget for programmet. Der er blevet gjort brug af relevante diagrammer, metoder og modeller som fungerer understøttende for hvordan programmet skal fungerer og hvordan det mest optimalt skal bruges.

## 2. Projektplanlægning

CDIO 2-projektet er et projekt der bliver udarbejdet i fællesskab med uddelegering af ansvarsområder. Hertil vil hvert gruppemedlem få ansvar for en lille del af alle opgaverne, som tilsammen vil udgøre de forventninger der er til programmet som kunden og projektlederne har. Projektet er opbygget efter først at lave intenst forarbejder i spørgsmål til kunden og projektlederne, og herefter en tidsplanlægning som gør det muligt for gruppen at kunne opretholde tidsfrister.

## 3. Krav

Krav til spillet:

- Spil mellem to personer.
- Skal kunne spilles på DTU's databarer uden bemærkelsesværdige forsinkelser (under 0,3 sekunder.)
- Spillerne slår på skift med 2 terninger og lander på et felt med numrene fra 2 - 12.
- At lande på hvert af disse felter har en positiv eller negativ effekt på spillernes pengebeholdning.
- Der skal udskrives en tekst omhandlende det aktuelle felt.
- Spilleren får tildelt eller fratrukket point alt efter hvilket felt spilleren lander på.
- Hvis spilleren lander på felt 10 får spilleres en ekstra tur.
- Spillerne starter med en pengebeholdning på 1000.
- Spillet er slut når en spiller når 3000.

- Spillet skal let kunne oversættes til andre sprog.
- Det skal være let at skifte til andre terninger.

Krav til programmet:

- Overhold objektorienteret design principper.
- Benyt de terninger I allerede har lavet.

Krav til kode:

- Lav en klasse Spiller der indeholder en spillers attributter og funktioner.
- Lav tilsvarende en klasse Konto der beskriver Spillerens pengebeholdning.
- Overvej hvilke typer attributterne i Spiller, samt i Konto skal have. Lav passende konstruktører.
- Lav passende get og set metoder.
- Lav passende toString metoder.
- Tilføj metoder til at indsætte og hæve penge på en Konto.
- Ændr nu Konto-klassen således at der ikke kan komme en negativ balance, ligeledes skal metoderne fortælle om transaktionen er blevet gennemført (**Hint:** brug statementet **return** til at returnere denne information).
- Lav det spil kunden har bedt om med de klasser I nu har.
- Hvis I vælger at bruge GUI'en kan I evt. benytte metoderne i Bilag 1.
- Husk at skrive en oversigt over pakkerne og deres klasser - klassernes ansvarsområder og evt. spændende funktioner.

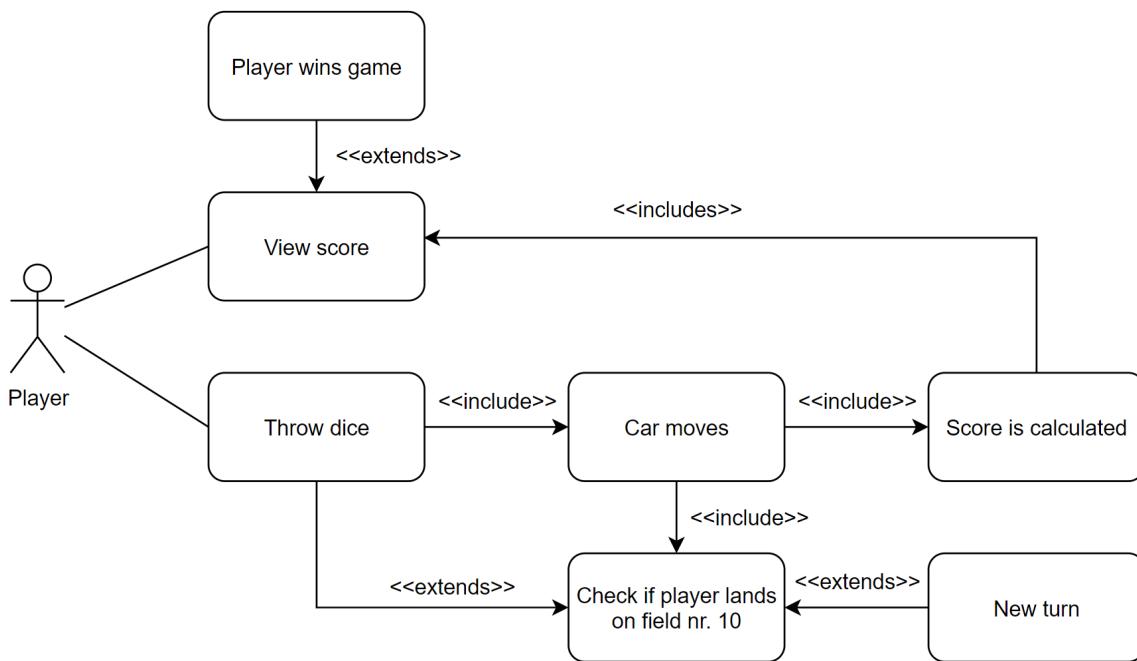
## 4. Analyse

### 4. 1 Use case diagram

Der er kun en aktør, da spillerne i et use-case diagram må anses som en og samme funktion. Aktøren har den mulighed at kunne spille spillet. Use case diagrammet er derfor visualiseret meget simpelt som kan ses på modellen nedenfor.

Aktøren vil have mulighed for at slå med terningerne, og se scoren, som også anfører om han har vundet. Ift. scoren vil "Player wins the game" blive aktiveret såfremt aktøren får 3000 point. Hvis spilleren slår med terningerne, medfører det at bilen rykkes. Dette vil medfører at

en scoren for aktøren bliver opdateret. Det vil yderligere medfører en kontrol af, om bilen er landet på felt nr. 10. Der er her to muligheder, hvis han ikke er landet på punkt 10 vil aktørens tur enten ende, men hvis aktøren er landet på punkt 10, vil aktøren få lov at slå med terningerne en gang til og derved få en ekstra tur.



## 4. 2 Use case beskrivelser

### Brief

En spiller kaster med terningerne og rykker det antal felter frem som der vises. Der opnås point alt efter hvad feltet viser. Turen går derefter videre til den anden spiller. Dette fortsættes indtil en af spillerne har opnået 3000 point.

### Fully dressed

Use Case section	Kommentar
<b>Use Case Name</b>	Spil CDIO2-spillet
<b>Scope</b>	Det nye CDIO2 spil
<b>Level</b>	Bruger niveau

<b>Primary Actors</b>	Spiller 1 og spiller 2
<b>Stakeholders</b>	Spillerne: gidsler  Projektleder: ressourceperson  Kunden: ressourceperson
<b>Preconditions</b>	Spilleren er logget ind på en databar på DTU.
<b>Success guarantee</b>	En spiller kan vinde spillet.
<b>Main Success Scenario</b>	1. Spiller kaster terninger 2. Spiller rykker frem 3. Spiller får tildelt et antal point 4. Turen går videre til næste spiller 5. Spillet slutter når en spiller når 3000 point.
<b>Extentions</b>	2. Spiller rykker frem <ul style="list-style-type: none"> <li>a. Spiller lander på felt 2               <ul style="list-style-type: none"> <li>i. Spiller får tildelt 250 point</li> </ul> </li> <li>b. Spiller lander på felt 3               <ul style="list-style-type: none"> <li>i. Spiller mister 100 point</li> </ul> </li> <li>c. Spiller lander på felt 4               <ul style="list-style-type: none"> <li>i. Spiller får tildelt 100 point</li> </ul> </li> <li>d. Spiller lander på felt 5               <ul style="list-style-type: none"> <li>i. Spiller mister 20 point</li> </ul> </li> <li>e. Spiller lander på felt 6               <ul style="list-style-type: none"> <li>i. Spiller får tildelt 180 point</li> </ul> </li> <li>f. Spiller lander på felt 7               <ul style="list-style-type: none"> <li>i. Spiller får tildelt 0 point</li> </ul> </li> <li>g. Spiller lander på felt 8               <ul style="list-style-type: none"> <li>i. Spiller mister 70 point</li> </ul> </li> <li>h. Spiller lander på felt 9               <ul style="list-style-type: none"> <li>i. Spiller får tildelt 60 point</li> </ul> </li> <li>i. Spiller lander på felt 10</li> </ul>

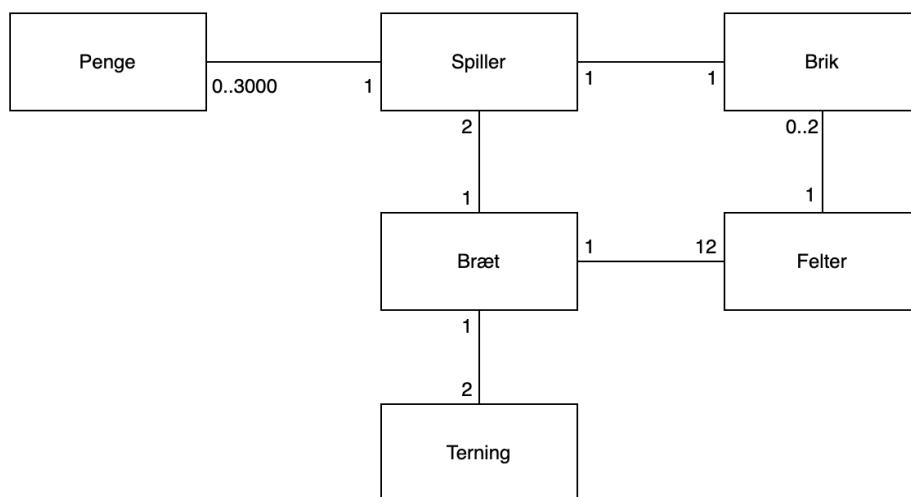
	<ul style="list-style-type: none"> <li>i. Spilleren får et kast mere</li> <li>j. Spiller lander på felt 11                     <ul style="list-style-type: none"> <li>i. Spiller mister 50 point</li> </ul> </li> <li>k. Spiller lander på felt 12                     <ul style="list-style-type: none"> <li>i. Spiller får tildelt 650 point</li> </ul> </li> </ul>
<b>Frequency of Occurrence</b>	Skal virke 100% af tiden.

## 4.3 Domænemodel

I forbindelse med udarbejdelsen af programmet er der blevet lavet en domænemodel.

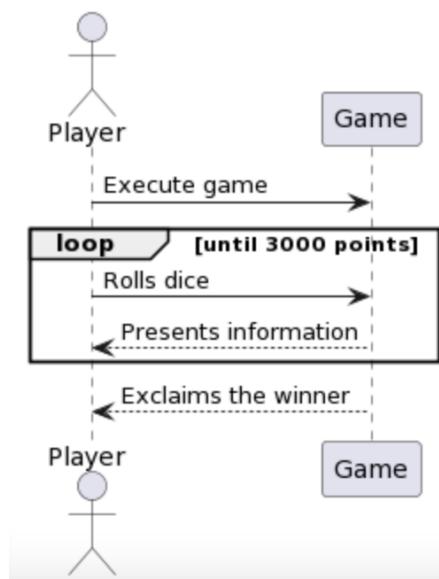
Domænemodellen viser hvad der indgår i spillet og hvordan de forskellige elementer spiller sammen. Det kan blandt andet ses hvordan at spillet består af 2 spillere som hver har én brik. Hver brik kan stå på ét felt, og ét felt kan minimum have 0 brikker og maks have 2 brikker stående på sig. I spillet er der i alt 12 felter som fordeler sig på 1 bræt. Til brættet er der to terninger. Herudover kan det også ses at hver spiller kan have en vis mængde penge som minimum kan være 0 og maks være 3000.

Model 1



## 4. 4 Systemsekvensdiagram

Der vil være en aktør bestående af en spiller, som igangsætter et system og som kan sende input til systemet. Inputtet består i at kunne rulle terningerne. Dette vil medfører at bilen rykker sig fysisk på spillepladen, og en ny score bliver præsenteret for spilleren, indtil en win-betingelse bliver opfyldt. Når win-betingelsen opfyldes vil systemet sende en besked om, hvilken spiller som vandt.



1

## 5. Design

### Oversigt over klasser og deres funktion:

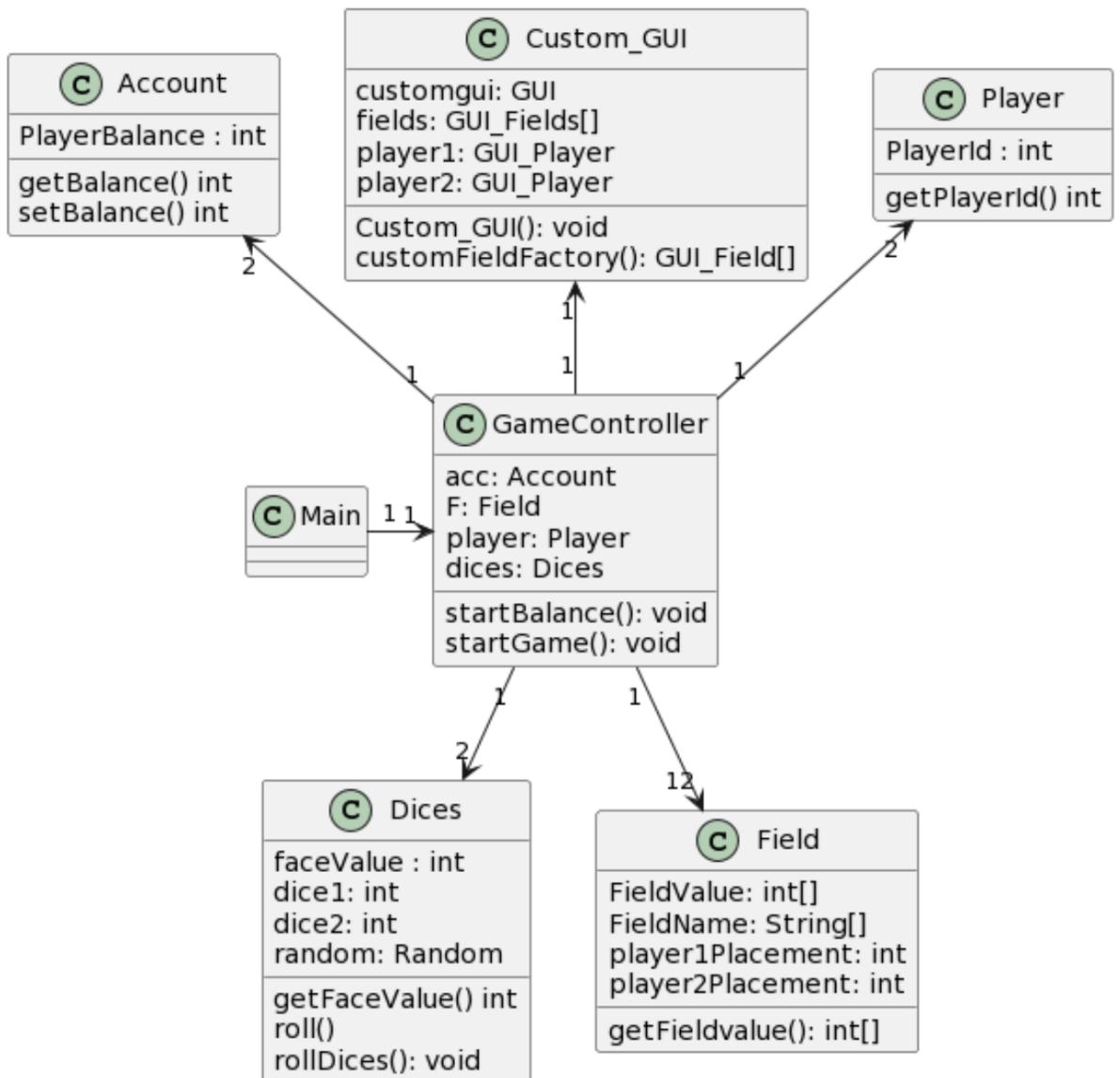
1. Package: Game
  - a. Class: Account - Spillerens point-konto
  - b. Class: Custom\_GUI - Modificeret GUI med tilpasset felter
  - c. Class: Dices - Slår terninger
  - d. Class: Field - Spillerens position på brættet
  - e. Class: GameController - Styrer spillets gang
  - f. Class: Main - Starter controlleren og derved spillet
  - g. Class: Player - Giver et playerID
2. Package: game\_test
  - a. Class: DiceTest - Tester om terningerne kan slå mere eller mindre end 6
  - b. Class NegativeAccountTest - Tester om spillerens konti kan blive negativ

<sup>1</sup> Bilag 1

- c. Class: probabilityOfNegativeTest - Tester hvor ofte en spillers konti bliver negativ.

## 5.1 Design klassediagram

Game Controlleren er forbundet med alle kasserne, da den sørger for spillets logik og for at integrere GUI herpå. Main klassen instantiere Game Controlleren og køre hele spillet derigennem. På diagrammet er der undladt at de fleste gettere og setttere, af hensyn til at gøre diagrammet mere overskueligt.

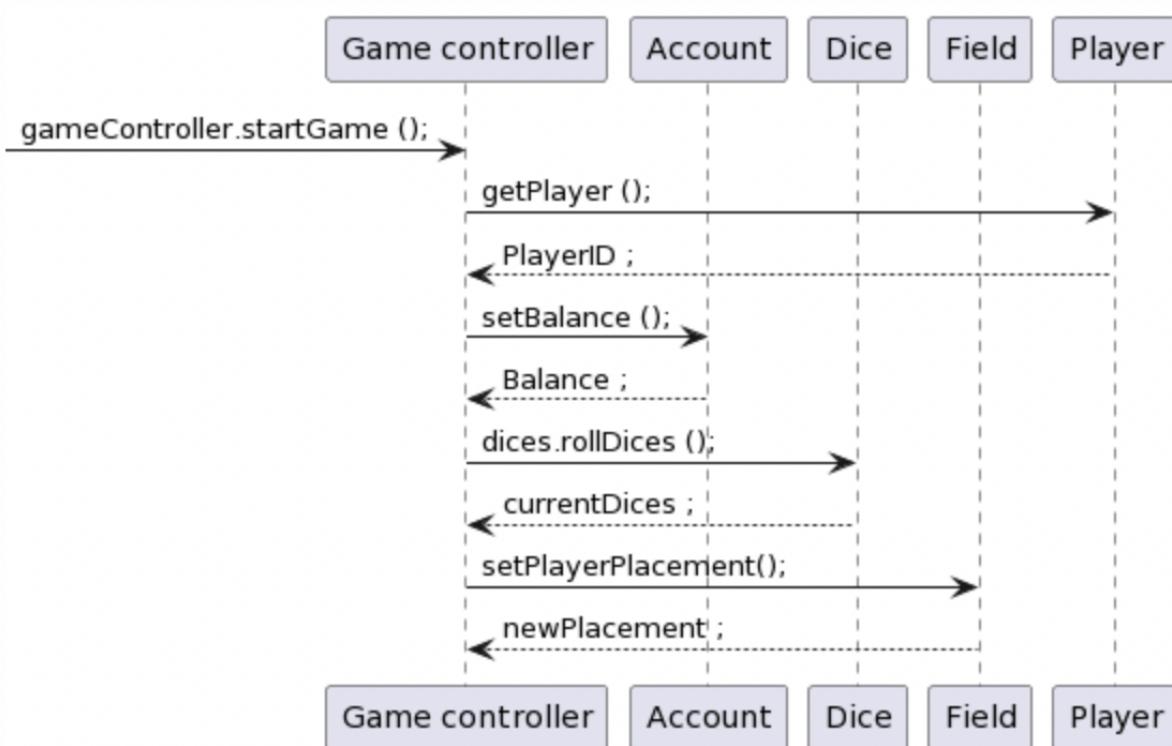


<sup>2</sup>

<sup>2</sup> Bilag 2

## 5.2 Design sekvensdiagram

I et design sekvensdiagram ser man de forskellige klasser i programmet og den logiske struktur systemet har. Det starter med at en metode sætter spillet i gang, som derefter igangsætter Game Controlleren. Det er Game Controlleren, som sender de forskellige metoder ud til de forskellige klasser. Det bliver så sendt retur det resultat fra klasserne, til Game Controller.



3

## 6. Test

Vi har lavet følgende tests med success:

**DiceTest:** Ved 100 kast er der 0 slag hvor en af de to terninger har øjne højere end 6 eller lavere end 1. Dette er en whitebox-test da vi i spillet bruger det samlede antal terninger, og i testen bruger de enkelte terninger.

---

<sup>3</sup> Bilag 3

**NegativeAccountTest:** Tester om en spillers points kan være negativ ved at indsætte en negativ værdi i account. Testen er success, og spillerne kan således ikke have en negativ konto.  
Er en whitebox-test da kun boolean statementet som skal stoppe spillet testes.

**probabilityOfNegativeTest:** Testen tester hvad chancen er for, at en spiller får en negativ konto. Chancen er bestemt til at være 145/100.000, altså en 0.00145% chance for at en spiller får en negativ konto og programmet stopper.  
Er en whitebox-test da det er en simulering af spillet for en enkelt spiller som køres, i stedet for spillet selv.

## 7. Konfiguration

Krav til at køre programmet:

- Styresystem med java 17.0 eller nyere

Start af programmet:

- Programmet startes ved at dobbeltklikke CDIO2.jar

For at finde kildekoden kan denne blive downloadet fra Github ved at trykke på linket<sup>4</sup>. Derefter trykker du på “Code” knappen og derefter “Download ZIP”.

## 8. Konklusion

Rapporten belyser arbejdsprocessen i gruppen, samt viser passende modeller og diagrammer fra et analyse og design perspektiv. På den måde formår vi at tilfredsstille kundens krav til projektet. Projektet er lavet objektorienteret og segmenteret passende sådan at kunden nemt kan tilføje eller modificere koden. Dog er der plads til optimering ved næste CDIO projekt ift. strukturering af kode. Relevante tests er lavet på den statistiske sandsynlighed for at balancen aldrig kan blive negativ.

## 9. Bilag

- Bilag 1: Systemsekvensdiagram

---

<sup>4</sup> Bilag 4

```
@startuml
actor Player
participant Game
Player -> Game : Execute game
loop until 3000 points
Player -> Game : Rolls dice
Game --> Player : Presents information
end
Game --> Player : Exclaims the winner
@enduml
```

- Bilag 2: Designklassediagram

```
@startuml
class Account
class Custom_GUI
class Dices
class Field
class GameController
class Main
class Player
Main "1" -> "1" GameController
Player "2" <-- "1" GameController
Account "2" <-- "1" GameController
GameController "1" --> "2" Dices
GameController "1" --> "12" Field
Custom_GUI "1" <-- "1" GameController

class Dices {
  faceValue : int
  getFaceValue() int
  roll()
}

class GameController {
  acc: Account
```

F: Field

```
player: Player
dices: Dices
startBalance(): void
startGame(): void
}
```

```
class Custom_GUI{
    customgui: GUI
    fields: GUI_Fields[]
    player1: GUI_Player
    player2: GUI_Player
    Custom_GUI(): void
    customFieldFactory(): GUI_Field[]
}
```

```
class Dices{
    dice1: int
    dice2: int
    random: Random
    rollDices(): void
}
```

```
class Field {
    FieldValue: int[]
    FieldName: String[]
    player1Placement: int
    player2Placement: int
    getFieldvalue(): int[]
}
```

```
class Player {
    PlayerId : int
    getPlayerId() int
}
class Account {
```

```
PlayerBalance : int  
getBalance() int  
setBalance() int  
}
```

```
@enduml
```

- Bilag 3: Design sekvensdiagram

```
@startuml
```

```
participant "Game controller" as B  
participant "Account" as C
```

```
participant "Dice" as E  
participant "Field" as F  
participant "Player" as G
```

```
-> B : gameController.startGame ();  
B -> G : getPlayer ();  
G --> B : PlayerID ;  
B -> C : setBalance ();  
C --> B : Balance ;  
B -> E : dices.rollDices ();  
E --> B : currentDices ;  
B -> F : setPlayerPlacement();  
F --> B : newPlacement ;
```

```
@enduml
```

- Bilag 3: Github repository:  
<https://github.com/sarabarslund/CDIO2.git>