

Videregående programmering 02324 - Gruppe 18

## Aflevering 3

---

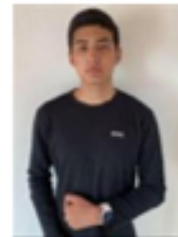
af gruppe 18



Jawahir Farah Yusuf  
s205414



Emin Ölcet  
s205410



Alec Ranjitkar  
s205427



Saif Al-Rubei  
s205470



Hassan Kassem  
s205409



Aya Fahim  
s205466

11. Juni 2021

## Indholdsfortegnelse

<b>Indledning:</b> .....	<b>3</b>
<b>Problembeskrivelse:</b> .....	<b>4</b>
<i>Kontekst &amp; baggrund:</i> .....	4
<i>MoSCoW-model:</i> .....	4
<i>Persistent data:</i> .....	5
<i>Hvad man kan bygge videre på, fx kode i starten, DB-tilknytning:</i> .....	6
<b>Kravspecifikation (analyse)</b> .....	<b>7</b>
<i>Begreber &amp; funktioner:</i> .....	7
<i>Usecases</i> .....	10
<i>Domænenmodel:</i> .....	13
<i>Aktivitetsdiagrammer:</i> .....	14
<i>Beskriv ikke funktionelle krav</i> .....	15
<i>UC-diagram</i> .....	16
<b>Database og softwaredesign:</b> .....	<b>17</b>
<i>Overblik over de vigtigste komponenter inkl., tilkobling af databasen:</i> .....	17
<i>Klassediagram:</i> .....	18
<i>Sekvensdiagram:</i> .....	19
<b>Implementering:</b> .....	<b>19</b>
<b>Håndbog:</b> .....	<b>20</b>
<b>Konklusion:</b> .....	<b>21</b>
<b>Referencer:</b> .....	<b>22</b>
<b>Appendix:</b> .....	<b>22</b>

## Indledning:

Der skal laves en digital version af roborally spillet. Spillet kan minust spilles af 2 og maximum 6 spillere. Alle deltagere kan programmere deres robot ved brug af adskillige kort, som kan udføre en instruks. Spillet er et kapløb, hvor det gælder om at hver spiller skal få sin robot til at nå forskellige checkpoints i forskellige rækkefølge. Spille banen består af 64 felter, og har forskellige forhindringer, der gør at spillernes robotter ikke kan komme frem alle vegne.

I forbindelse med tidligere afleveret projekter om kode udvikling og implementering af roborally spillet vil der i denne rapport tages fat af en optimeret version og sidste udgave af spillet. For beskrive spillets krav, er der udarbejdet en prioritet kravliste ved brug af MoScow. Derudover er der i kravspecifikationen præsenteret begreber og funktioner. De ikke funktionelle krav er blevet specificeret. Dernæst bliver der præsenteret i form af forskellige use cases og use case diagram samt et domain model, hvilket har til formål at visualiserer den grundlæggende struktur for spillet.

Ydermere bliver der i rapporten sat fokus på spillets funktionalitet, og design, ved at der er udarbejde en domain model, og design klasse diagram, over hele det udviklet spil og sekvensdiagrammer til centrale dele af spillet.

## Problembeskrivelse:

### Kontekst & baggrund:

Kunden har tildelt os en opgave, hvor vi skal lave et spil for vores kunde. Kunden har valgt at vi skal udvikle spillet Roborally og til det har vi fået tildelt en skabelon og visse krav som skal udføres. skabelonen er udelukkende et bræt og vi skal derfra indføre spille-logikken, det vil vi gøre ved at kigge spillets regler igennem og implementerer dem på spillebrættet. Desuden prioriterer vi implementeringen af spillet ud fra de krav som vi er blevet stillet, hvilket synliggøres i MoSCoW-modellen.

### MoSCoW-model:

<b>Must have</b>	<p>Spillets er en virtuel version af brætspillet Roborally, og skal derfor have de samme primære funktionaliteter såsom:</p> <ul style="list-style-type: none"><li>• 2-6 spiller</li><li>• Spilleplade bestående af 64 felter</li><li>• Ethvert spiller har en robot.</li><li>• At kunne lave et nyt spil</li><li>• At kunne gemme et spil</li><li>• At kunne loade et gammelt spil som er gemt</li><li>• At robotterne skal kunne programmeres af spilleren via kort, der navigere robotterne rundt på spillepladen</li><li>• at spillerne programmere deres robotter samtidig</li><li>• At der er checkpoint på spillepladen</li><li>• At der er minimum 2 ekstra features på spillebrættet kunne f.eks. være forhindringer på spillebrættet osv.</li><li>• Den spiller der først når alle checkpoints har vundet</li><li>• At kunne gemme eller hente spil</li><li>• At hver runde i spillet består af 3 faser:<ul style="list-style-type: none"><li>○ Upgrade Phase</li><li>○ Programmering Phase</li><li>○ Activation phase</li></ul></li></ul>
------------------	--

<b>Should have</b>	<ul style="list-style-type: none"> <li>• Board elementer (diverse kommandoer spiller robot skal gøre hvis de ender på det felt)</li> <li>• Hver spiller for 30 sekunder til at programmere en robot</li> </ul>
<b>Could have</b>	<ul style="list-style-type: none"> <li>• Damage Cards ( Er et kort der som udløser en konsekvens for spilleren)</li> <li>• Spillernes robot kan falder ud over spillepladen</li> <li>• Hvis spillers robot falder ud over spillepladen for spilleren et damage card</li> </ul>
<b>Will not have</b>	<ul style="list-style-type: none"> <li>• Når robotterne er færdig med at udføre deres kommander, bliver der skudt med laser, fra element boardene.</li> <li>• Hvis man bliver ramt af laser, skal man programmere et damage card</li> </ul>

## Persistent data:

Kunden har tildelt nogle krav, som de mener skal være permanente i databasen, herunder tildelte de os en skabelon med 2 tabeller for at gemme spillets tilstand. Disse tabeller er Game og Player.

Game indeholder:

- gameId, som er det enkelte spils unikke identifikator
- name, som indeholder det enkeltes spils navn og dato fra da den blev gemt
- phase, som fortæller hvilken phase spillet er i
- currentPlayer, som fortæller hvilken spillers tur det er.

player indeholder:

- gameId, som er det enkelte spils unikke identifikator
- playerId, som er spillerens unikke identifikator
- name, som indeholder det enkeltes spils navn og dato fra da den blev gemt
- color, som er den farve som spilleren har.
- positionX, er den position som spilleren har på boardet ud fra x-aksen
- positionY, er den position som spilleren har på boardet ud fra y-aksen
- heading, er den retning som den enkelte spillere peger mod.

Vi har desuden selv implementeret en tabel der kunden ønskede at spillernes kort også blev gemt. Denne tabel hed CardFields. CardFields indeholder:

- gameId, som er det enkelte spils unikke identifikator
- playerId, som er spillerens unikke identifikator
- Type,
- position, bestemmer hvilken rækkefølge kortene står
- visible, er om de er synlige eller ej
- command, hvilken command kortet har

### **Hvad man kan bygge videre på, fx kode i starten, DB-tilknytning:**

Til at starte med kan vi bygge videre på kortene således at der kommer flere kort, og at de får egenskaber, til at udføre noget, altså ved at flytte spillerne. Man kunne også gøre spillepladen mere avanceret, ved at tilføje forhindringer og felt aktioner, såsom vægge og conveyor belts.

Desuden kan vi tilknytte spillet til en database, så man får muligheden for at gemme spillets tilstand, for at loade det senere og fortsætte hvor man slap.

# Kravspecifikation (analyse)

## Begreber & funktioner:

### Taksonomi:

Taksonomi er en liste af begreber. Der er blevet lavet en liste af begreber for hovedord og udsagnsord ud fra roborally. Hovedordene og udsagnsordene er blevet adskilt fra hinanden da de spiller to forskellige roller. Hovedord spiller typisk en rolle i klasse diagrammerne mens udsagnsordene er hvor aktiviteterne og adfærden af softwaren kommer ind.

### Hovedord (nouns):

- Robots
- Programming Cards
- Board
- Fields
- Checkpoints
- Walls
- Convenables
- Heading

### Udsagnsord (verbs):

- New Game
- Save Game
- Load Game
- Forward, Fast Forward, Turn Right, Turn Left, Turn Left or Turn Right
- Finishing programming
- Execute program
- Execute current program

## Glossar:

Glossar er en beskrivelse af de forskellige begreber som er blevet anvendt, deres egenskaber og hvilke forhold de har til hinanden.

### Hovedord (nouns):

Robots	Robotterne er princippet spilleren brik, som står på nogle af felterne på spillepladen. Robotten er afhængig af spillekortene da robotten ikke kan gøre noget hvis den ikke får en commando.
Programming Cards	Programmerings Card er spillekortene som anvendes til at styre robotten så spillet kan gennemføres og man kan finde en vinder. Forholdet mellem Robotten og spillekortene er derfor stor da robotten er princippet er afhængig af spillekortene.
Board	Board er spillebrættet som spillet bliver spillet på. Brættet består af felter hvori nogle af dem har forhindringer. I princippet er alt afhængig af spillepladen da spillet ikke kan spilles uden brættet.
Fields	Fields er felterne på spillebrættet
Checkpoints	Checkpoints anvendes så spillet i princippet så spillerne kan nå deres mål og man kan finde en vinder i spillet. Uden checkpoints vil det derfor ikke være muligt at gennemfører spillet. Derfor er selve spillet af afhængig af checkpoints.
Walls	Nogle af felterne på spillebrættet har walls. Walls anvendes stort set bare som forhindringer til at distrahere de forskellige spiller som er med i spillet, da walls ligger forskellige steder på brættet er der et forhold mellem brættet og felterne på spillepladen da walls er afhængig af felterne. Da der ikke kan være forhindringer uden felter.



Conveyor belts	Conveyor belts er en feltaktion, som skubber robotten i den retning pilen peger mod. Conveyor belts har et forhold til brættet, da det er der de befinder sig, og at den skubber spilleren hen til et andet felt på brættet.
Heading	Heading er det der definere hvilken vej robotten plejer mod og ved at ændre på heading kan man ændre på retningen robotten peger mod. Uden heading vil det derfor ikke være muligt at dreje robotterne, og derfor er der et forhold et lille forhold mellem dem da, robotten i princippet stadig godt kan gå, frem, tilbage og tilside men robotens hoved vil bare ikke pege mod retningen som robotten går.

#### Udsagnsord (verbs):

New Game	New Game anvendes til at lave et nyt spil
Save Game	Save Game anvendes til at gemme spillet i en database, så man kan komme ind på det det gemte spil igen.
Load Game	Load Game anvendes til at komme ind i de spil som er blevet gemt. Hvis der ikke er et spil som er gemt ville denne kommando ikke været muligt og derfor er den afhængig af "save game".
Forward, Fast Forward, Turn Right, Turn Left, Turn Right or Turn Left	Disse funktioner er alle programmerings kortene som hver især har forskellige egenskaber. Forward - Robotten tager et skridt frem Fast Forward - Robotten tager to skridt frem Turn Right - Robotten drejer til højre Turn Left - Robotten drejer til venstre Turn Right or Turn Left giver spilleren mulighed for at dreje robotten til højre eller venstre
Finishing Programming	Denne funktion låser de kort fast spilleren har valgt, så de kan executes.

Execute	Execute eksekvere de kort spilleren låste fast med finish programming i samme rækkefølge som spilleren har valgt.
Execute current register	Execute current register eksekvere et kort af gangen af de kort spilleren har valgt med finish programming.

## Usecases

### New game:

**Use case description:** Brugeren trykker på den for at starte et nyt spil

**Precondition:** For at kunne lavet et nyt spil er man nødt til at trykke på file

**Postcondition:** Et nyt spil er startet.

### Save game:

**Use case description:** Funktion som bruges til at gemme et ufærdigt spil

**Precondition:** Brugeren skal have startet eller være i gang med et spil

**Postcondition:** Spillet bliver gemt i en databaser

### Load game:

**Use case description:**Når spilleren trykker på den her knap, kan de vælge fortsætte på et gemt ufærdigt spil de allerede var igang med.

**Precondition:** Brugeren skal have gemt et tidligere spil.

**Postcondition:** Brugeren kan fortsætte med at spille et tidligere gemt spil.

### Finish programming:

**Use case description:**Brugeren kan færdiggøre programmeringen af sin robot og få robotten til at bevæge sig.

**Precondition:** Der skal være valgt nogle kort med kommandoer for at aktiveringsfasen kan starte.

**Postcondition:** At aktiveringsfasen går i gang.

**Select numbers of Players:**

**Use case description:** Ved denne knap for man lov til at vælge antallet af spiller mellem 2 - 6.

**Precondition:** Virker kun når hvis der laves et nyt spil

**Post condition:** Der er samme antal af robotter på spillebrættet som det antal af spiller der er blevet valgt.

**Choose board:**

**Use case description:** Ved denne knap for brugeren lov til at vælge hvilket bræt der skal spilles på mellem de 2 forskellige bræt der er

**Precondition:** Man skal have valgt spiller,

**Postcondition:** Det valgt spillebræt er blevet uploadet og man kan nu spille spillet.

**Execute program:**

**Use case description:** Denne funktion får robotten til at bevæge sig

**Precondition:** At robotten 'er færdig programmeret

**Post condition:** Robotten har bevæget sig på pladen

**Execute current register:**

**Use case description:** Execute current register eksekvere et kort af gangen af de kort spilleren har valgt med finish programming

**Precondition:** At der er minimum et kort på the commando card field.

**Postcondition:** Spillers robot, laver en bevægelse på boardet.

**New file:**

**Use case description:** Flie er en knap som giver dig valgmuligheder, til at load eller starte et spil.

**Precondition:** Man skal have en IDE der kan køre java kode, og køre koden

**Post condition:** Man kan vælge imellem exit , new game, load game.

**Exit game:**

**Use case description:** Exit game er en knap i spillet som giver brugeren lov til at forlade pågældende spil som er i gang

**Precondition:** Der skal være et spil som er i gang før det er muligt

**Postcondition:** Spillet er blevet afsluttet og lukket ned.

**Program robot:**

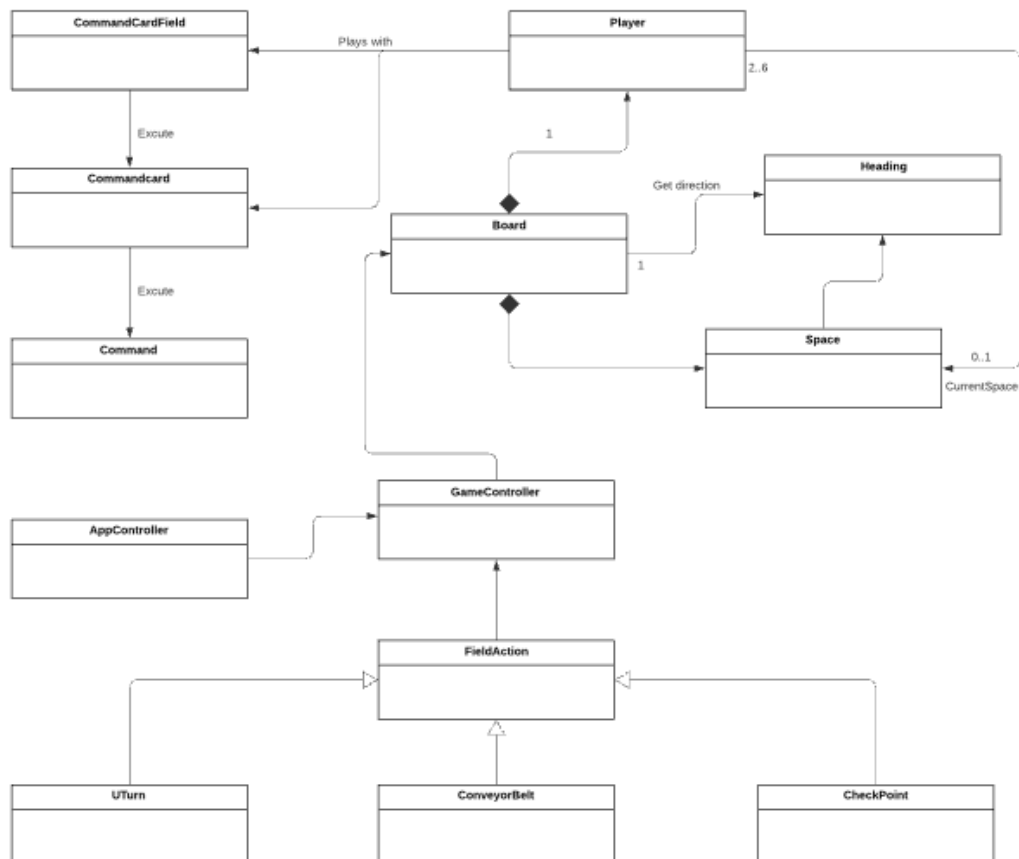
**Use case description:** Denne use case rykker man på commando kortene, og bestemme hvordan robotten vil bevæge sig hen af boardet.

**Precondition:** Man skal være i gang med et spil

**Postcondition:** Spillers robotten er programmeret med commando kortene

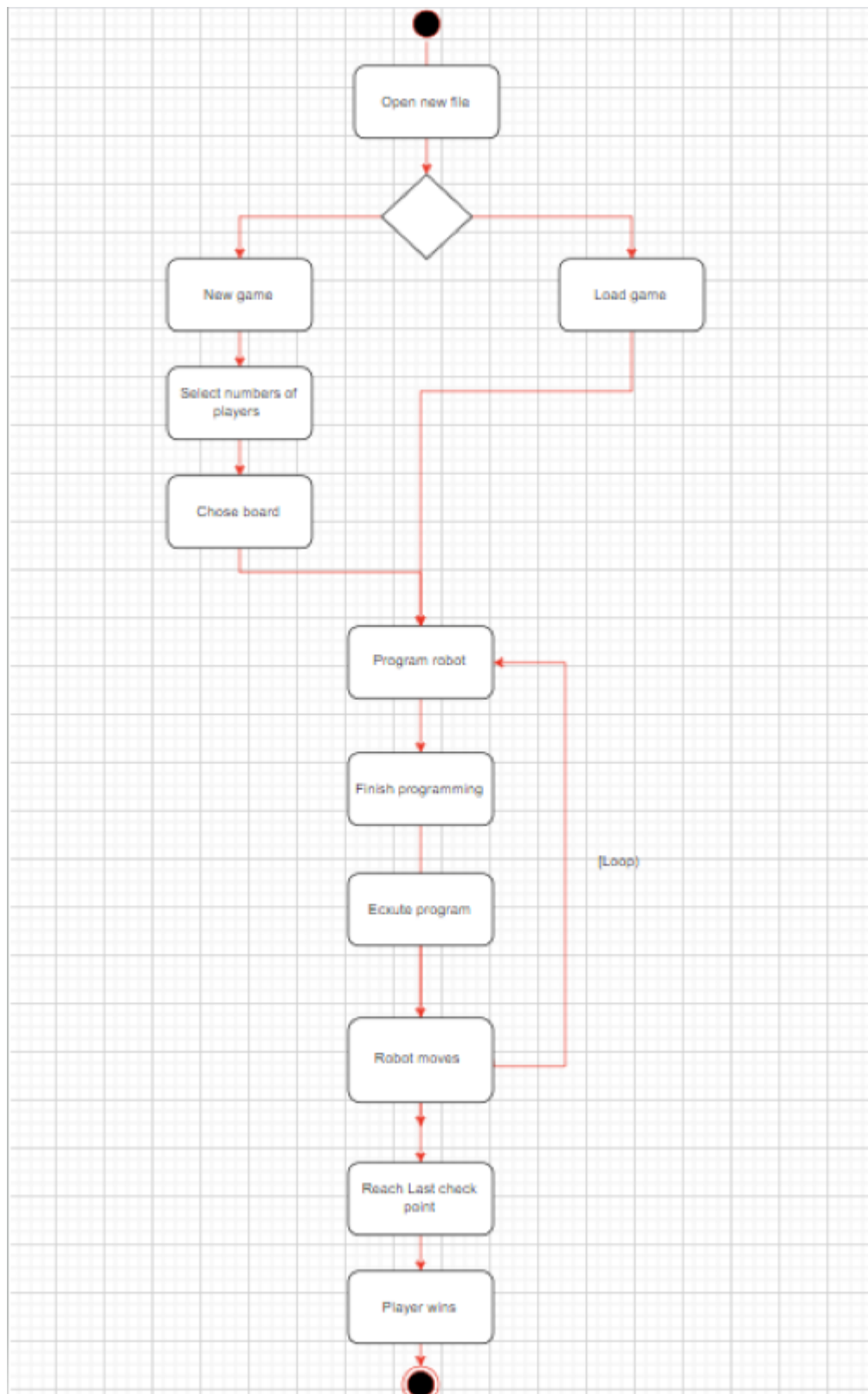
## Domænemodel:

Et Domain model anvendes til at repræsentere de virkelige objekter og de konceptuelle klasser, samt forholdet visuelt mellem dem. Den visuelle repræsentation giver en forståelse af systemet. Derudover har den også til formål at hjælpe udvikleren med at designe systemet til vedligeholdelse og trinvis udvikling. Ydermere vil modellen også hjælpe udviklerne og kunden at identificere eventuelle misforståelser mellem dem, da modellen forestiller funktionen af systemet



## Aktivitetsdiagrammer:

Aktivitetsdiagram er et behavioral diagram, der illustrerer flowet i et system fra start til slut. Det viser hvilke trin, som er involveret i udførelsen af use-casene, og modellere software elementerne såsom funktionerne, og operationerne i softwaren.



## Beskriv ikke funktionelle krav

### Usability:

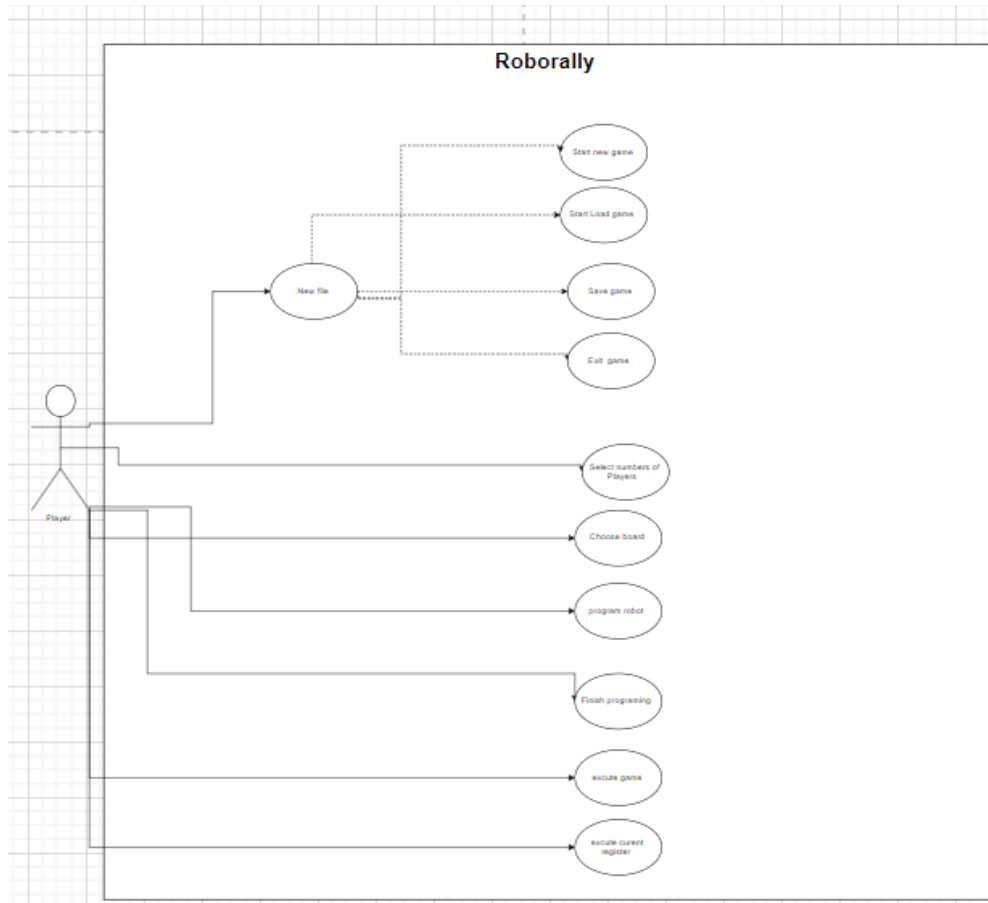
- Spillets sprog er på engelsk
- Det er et Multiplayer spil

### Supportability

- Spillet er programmeret i java men det er i princippet ikke nødvendigt at have hverken java eller en java compiler for at kunne spille spillet. Det kræver blot at være på det samme netværk som den enhed der host webappen, og at have adgang til en browser. Den enhed der hoster skal til gengæld have java, npm og en java ide/compiler installeret. Da vi kun tester spillet lokalt på den enhed der hoster, antager vi så at det kræver at have ovenstående software installeret.

## UC-diagram

Use case diagrammet illustrerer hvordan brugeren interagerer med systemet. Diagrammet der er konstrueret nedenfor kan man se at hvordan brugeren der skal spille roborally vil interagerer med det.





## Database og softwaredesign:

### Overblik over de vigtigste komponenter inkl., tilkobling af databasen:

Gennem udviklingen af dette spil har vi skulle benytte os af databaser, for at nogle ønskede funktionaliteter skulle fungere. Vi blev nemlig bedt om at gøre det muligt at gemme og lade spillets tilstand. Til at starte med laver vi en database. For at forbinde den med vores spil, som kører på java 15 installerer vi jdbc, som sørger for at java kan snakke med en mysql database.

```
String url = "jdbc:mysql://" + HOST + ":" + PORT + "/" + DATABASE + "?serverTimezone=UTC";

System.out.println("Connecting database...");

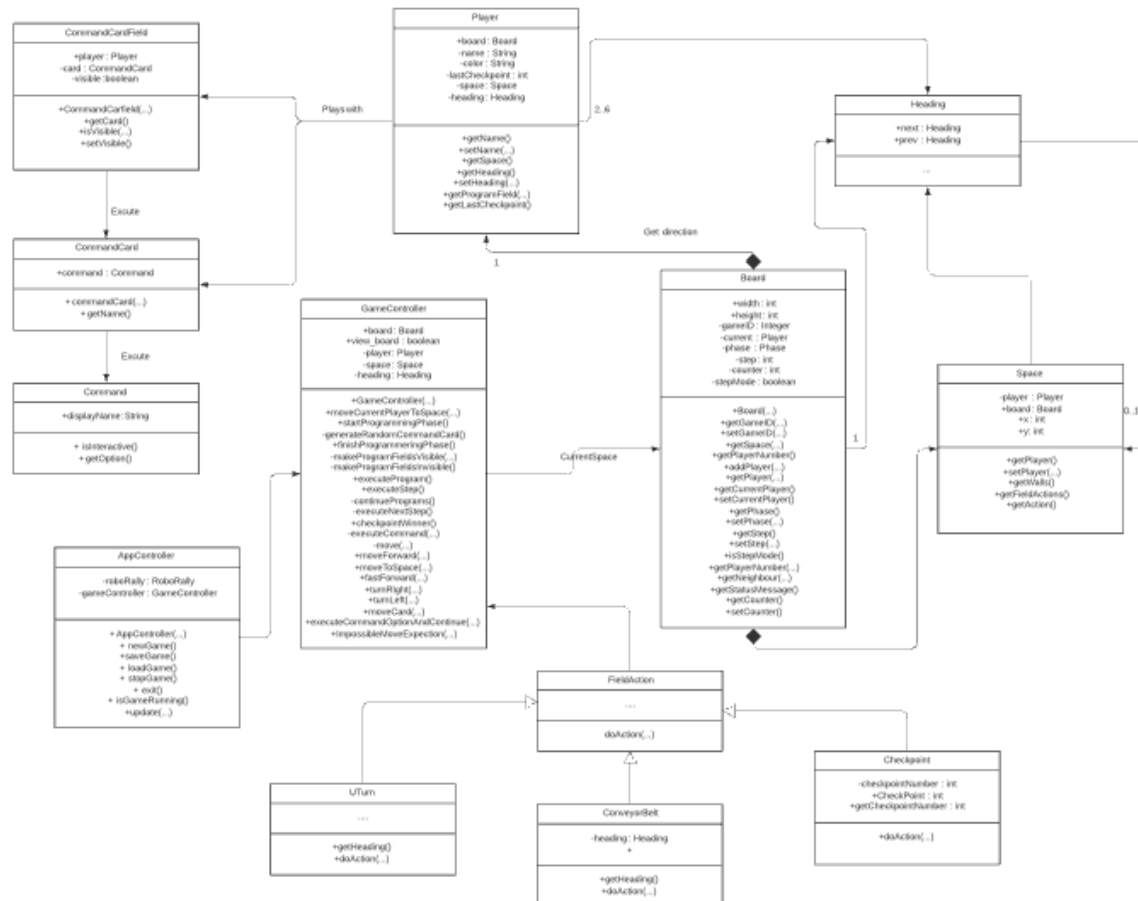
try {
    connection = DriverManager.getConnection(url, USERNAME, PASSWORD);
}
```

Når vi laver databasen noterer vi så det navn vi har givet den, og den port vi har benyttet. Vi definerer også et brugernavn og en adgangskode, som vi også skal indtaste i koden, for at de forbinder.

# Klassediagram:

I klassediagrammet har vi valgt at illustrere hver af klassernes metoder og hvilke variable som indgår i hver enkelte klasse. Da billedet er utydeligt har vi valgt, at vedhæfte et link nedenfor:

- [https://lucid.app/lucidchart/invitations/accept/inv\\_975a589a-f129-418d-aeed-ab135b40f2ac?viewport\\_loc=-126%2C-70%2C3911%2C1909%2CHWEp-vi-RSFO](https://lucid.app/lucidchart/invitations/accept/inv_975a589a-f129-418d-aeed-ab135b40f2ac?viewport_loc=-126%2C-70%2C3911%2C1909%2CHWEp-vi-RSFO)

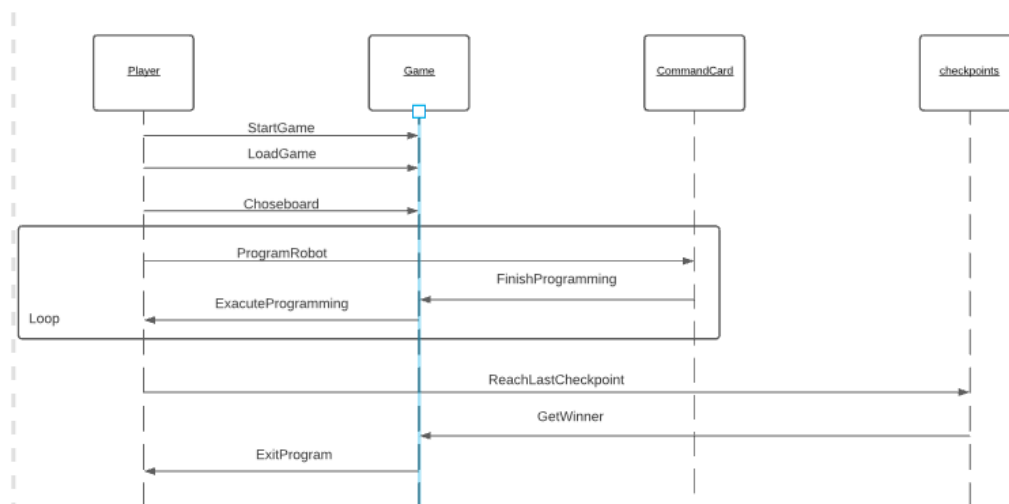


## Sekvensdiagram:

Sekvensdiagrammet har til formål at vise de enkelte objektinteraktioner sorteret efter den specifikke tidssekvens. De objektinteraktioner vi har taget udgangspunkt i er:

- Player
- Game
- CommandCard
- checkpoints

Disse objekter er opstillet i øverst i firkanterne og derefter anvender vi use cases til at forklare interaktionerne efter tidssekvensen.



## Implementering:

Vi har anvendt JDK 15 til at programmere vores spil samt Maven til at implementere såkaldte 'third-party' libraries, såsom MySQL og Google Guava. Spillet er sat op på den måde, at vi kan enten oprette en ny instans af spillebrættet eller indlæse gemte værdier og objekter fra vores MySQL database. Spillebrættet har forhindringer, såsom vægge, uturns og conveyorbelts som har sine fordele og ulemper. Disse forhindringer indlæses fra en JSON fil og håndteres ved brug af abstrakte klasser hvor 'FieldAction' er vores superklasse.

Vi kunne udvide vores spil ved f.eks. at tilføje flere forhindringer såsom pits(der vil slette ens checkpoints og nulstille spillerens startposition) og/eller checkpoints for at gøre spiloplevelsen mere udfordrende

## Håndbog:

- Hvordan installerer man selve software


Standalone:

Til at starte med skal man have installeret mysql, java 15, jdbc og en ide. Først starter man databasen op med følgende mySQL kommandoer:

```
1 • CREATE DATABASE pisu DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;  
2 • CREATE USER IF NOT EXISTS 'user'@'localhost' IDENTIFIED BY 'password';  
3 • GRANT ALL ON pisu.* TO 'user'@'localhost';
```

Den laver så en database ved navn pisu, og laver en bruger ved navn "user" og adgangskoden "password", derefter giver den alle de nødvendige (og også unødvendige) tilladelser til brugeren.

Når databasen er sat op skal man åbne sin ide med java 15 og hente den kode vi har

lavet. Så åbner man pom.xml filen op i projektet  pom.xml og reloader projektet ved brug af maven, for at sørge for at de rigtige dependencies er på plads.

 Maven

▶  Reload project

Web app: Java, npm og en ide skal installeres på den enhed hvor spillet skal hostes fra, spiller man fra en enhed der ikke er host behøver man kun en browser.

- Hvordan starter man softwaren

Standalone: Nu kan man åbne ide'en og køre spillet gennem StartRoborally klassen.

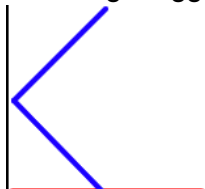
Web app: For at køre softwaren på en host enhed starter man med at kører backend, det gør man gennem DemoApplication klassen, som man kører. Dernest kører man frontend, ved at køre følgende kommando i roborally\_frontend

hovedmappen:  npm start Kommandoen åbner desuden automatisk browseren på localhost:3000, hvor man bliver mødt med spillet. For at starte softwaren på en anden enhed en host, skal man følge de samme steps, på en host enhed, og så kan man derfra koble til den ved at indtase host ip adresse + ":3000"

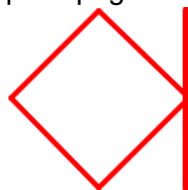
- Hvordan bruger man software (den eksisterende GUI er I ikke nødt til at diskutere med alle detaljer; men giv et overblik og diskuter jeres tilføjelser); evt. med nogle screenshots

Stand alone:

Udover den tildelte gui har vi implementeret nogle feltaktioner, såsom conveyor belts, u turn og vægge:



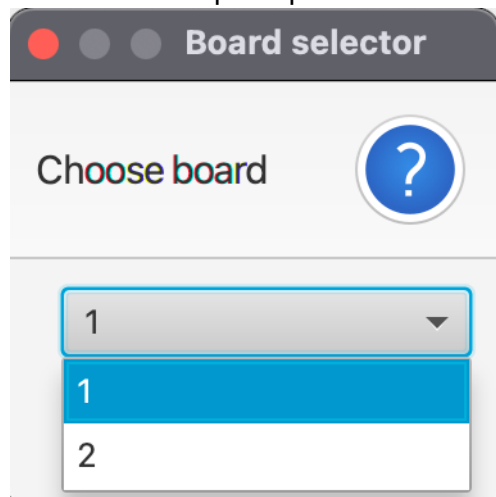
Når man lander på dette felt vil ens robot blive skubbet i den retning pilen peger mod. Heading på robotten vil ikke ændres.



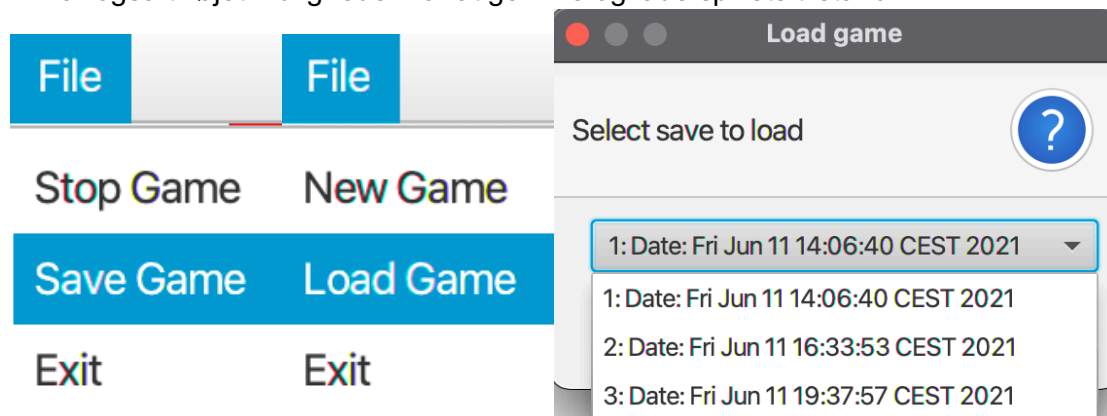
Når man lander på dette felt vil ens robot ændre sin heading med 180 grader uafhængigt af hvilken vej robotten allerede peger. Robottens placering på brættet forbliver det samme.

[REDACTED] Dette er ikke en feltaktion, men en forhindring mellem felterne, den sørger for at robotten ikke kan gå igennem den. Det er dog muligt at springe over en væg hvis man benytter sig af fast forward kortet hvis man er ved siden af væggen.

Vi har tilføjet mulighed for at vælge mellem 2 forskellige bræt for at det ikke bliver alt for ensartet at spille spillet.



Vi har også tilføjet muligheden for at gemme og lade spillets tilstand.



Web app: Udover den eksisterende GUI er der ingen yderligere tilføjelser.

## Konklusion:

Vi kan hermed konkludere, at vores projekt gik forholdsvis efter planen. Vi fik implementeret hele standalone programmet og tilkoblet denne til vores database, dog havde vi ikke mulighed for at få databasen til at gemme spillerens kort i samme rækkefølge som den var, og desuden gemmer databasen ikke spillerens register. Trods alt virker selve spillet og de features vi ønskede at implementere fungere uden problemer.

I henhold til Web appen har vi fået denne til at komme op at kører, men grundet tidspres fik vi desværre ikke implementeret yderligere på Web appen.

## Referencer:

Vi har gennem dette projekt anvendt bogen:

- Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development udgave 3.

## Appendix:

Hovedord (nouns):

- Robots
- Programming Cards
- Board
- Fields
- Checkpoints
- Walls
- Convenables
- Heading

Udsagnsord (verbs):

- New Game
- Save Game
- Load Game
- Forward, Fast Forward, Turn Right, Turn Left, Turn Left or Turn Right
- Finishing programming
- Execute program
- Execute current program

## Glossar:

Glossar er en beskrivelse af de forskellige begreber som er blevet anvendt, deres egenskaber og hvilke forhold de har til hinanden.

### Hovedord (nouns):

Robots	Robotterne er princippet spilleren brik, som står på nogle af felterne på spillepladen. Robotten er afhængig af spillekortene da robotten ikke kan gøre noget hvis den ikke får en commando.
Programming Cards	Programmerings Card er spillekortene som anvendes til at styre robotten så spillet kan gennemføres og man kan finde en vinder. Forholdet mellem Robotten og spillekortene er derfor stor da robotten er princippet er afhængig af spillekortene.
Board	Board er spillebrættet som spillet bliver spillet på. Brættet består af felter hvori nogle af dem har forhindringer. I princippet er alt afhængig af spillepladen da spillet ikke kan spilles uden brættet.
Fields	Fields er felterne på spillebrættet
Checkpoints	Checkpoints anvendes så spillet i princippet så spillerne kan nå deres mål og man kan finde en vinder i spillet. Uden checkpoints vil det derfor ikke være muligt at gennemfører spillet. Derfor er selve spillet af afhængig af checkpoints.
Walls	Nogle af felterne på spillebrættet har walls. Walls anvendes stort set bare som forhindringer til at distrahere de forskellige spiller som er med i spillet, da walls ligger forskellige steder på brættet er der et forhold mellem brættet og felterne på spillepladen da walls er afhængig af felterne. Da der ikke kan være forhindringer uden felter.

Conveyor belts	Conveyor belts er en feltaktion, som skubber robotten i den retning pilen peger mod. Conveyor belts har et forhold til brættet, da det er der de befinder sig, og at den skubber spilleren hen til et andet felt på brættet.
Heading	Heading er det der definere hvilken vej robotten plejer mod og ved at ændre på heading kan man ændre på retningen robotten peger mod. Uden heading vil det derfor ikke være muligt at dreje robotterne, og derfor er der et forhold et lille forhold mellem dem da, robotten i princippet stadig godt kan gå, frem, tilbage og tilside men robotens hoved vil bare ikke pege mod retningen som robotten går.

#### Udsagnsord (verbs):

New Game	New Game anvendes til at lave et nyt spil
Save Game	Save Game anvendes til at gemme spillet i en database, så man kan komme ind på det det gemte spil igen.
Load Game	Load Game anvendes til at komme ind i de spil som er blevet gemt. Hvis der ikke er et spil som er gemt ville denne kommando ikke været muligt og derfor er den afhængig af "save game".
Forward, Fast Forward, Turn Right, Turn Left, Turn Right or Turn Left	Disse funktioner er alle programmerings kortene som hver især har forskellige egenskaber. Forward - Robotten tager et skridt frem Fast Forward - Robotten tager to skridt frem Turn Right - Robotten drejer til højre Turn Left - Robotten drejer til venstre Turn Right or Turn Left giver spilleren mulighed for at dreje robotten til højre eller venstre
Finishing Programming	Denne funktion låser de kort fast spilleren har



	valgt, så de kan executes.
Execute	Execute eksekvere de kort spilleren låste fast med finish programming i samme rækkefølge som spilleren har valgt.
Execute current register	Execute current register eksekvere et kort af gangen af de kort spilleren har valgt med finish programming.

## Usecases

### New game:

**Use case description:** Brugeren trykker på den for at starte et nyt spil

**Precondition:** For at kunne lavet et nyt spil er man nødt til at trykke på file

**Postcondition:** Et nyt spil er startet.

### Save game:

**Use case description:** Funktion som bruges til at gemme et ufærdigt spil

**Precondition:** Brugeren skal have startet eller være i gang med et spil

**Postcondition:** Spillet bliver gemt i en databaser

### Load game:

**Use case description:**Når spilleren trykker på den her knap, kan de vælge fortsætte på et gemt ufærdigt spil de allerede var igang med.

**Precondition:** Brugeren skal have gemt et tidligere spil.

**Postcondition:** Brugeren kan fortsætte med at spille et tidligere gemt spil.

### Finish programming:

**Use case description:**Brugeren kan færdiggøre programmeringen af sin robot og få robotten til at bevæge sig.

**Precondition:** Der skal være valgt nogle kort med kommandoer for at aktiveringsfasen kan starte.

**Postcondition:** At aktiveringsfasen går i gang.

**Select numbers of Players:**

**Use case description:** Ved denne knap for man lov til at vælge antallet af spiller mellem 2 - 6.

**Precondition:** Virker kun når hvis der laves et nyt spil

**Post condition:** Der er samme antal af robotter på spillebrættet som det antal af spiller der er blevet valgt.

**Choose board:**

**Use case description:** Ved denne knap for brugeren lov til at vælge hvilket bræt der skal spilles på mellem de 2 forskellige bræt der er

**Precondition:** Man skal have valgt spiller,

**Postcondition:** Det valgt spillebræt er blevet uploadet og man kan nu spille spillet.

**Execute program:**

**Use case description:** Denne funktion får robotten til at bevæge sig

**Precondition:** At robotten 'er færdig programmeret

**Post condition:** Robotten har bevæget sig på pladen

**Execute current register:**

**Use case description:** Execute current register eksekvere et kort af gangen af de kort spilleren har valgt med finish programming

**Precondition:** At der er minimum et kort på the command card field.

**Postcondition:** Spillers robot, laver en bevægelse på boardet.

**New file:**

**Use case description:** Flie er en knap som giver dig valgmuligheder, til at load eller starte et spil.

**Precondition:** Man skal have en IDE der kan køre java kode, og køre koden

**Post condition:** Man kan vælge imellem exit , new game, load game.

**Exit game:**

**Use case description:** Exit game er en knap i spillet som giver brugeren lov til at forlade pågældende spil som er i gang

**Precondition:** Der skal være et spil som er i gang før det er muligt

**Postcondition:** Spillet er blevet afsluttet og lukket ned.

**Program robot:**

**Use case description:** Denne use case rykker man på commando kortene, og bestemme hvordan robotten vil bevæge sig hen af boardet.

**Precondition:** Man skal være i gang med et spil

**Postcondition:** Spillers robotten er programmeret med commando kortene