
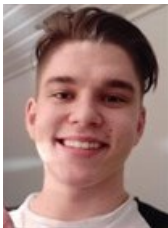



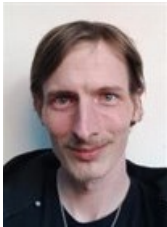

CDIO3

GRUPPE 25

 <p>Adam Harald Jørgensen s200718</p>	 <p>Andreas Engberg Carlsen s205437</p>	 <p>Bastian Emil Falk Larsen s195112</p>
 <p>Jens Skriver Kloster s190265</p>	 <p>Martin Lüthje Hermann s195127</p>	 <p>Michael Rene Lund Jensen s205460</p>

27. november 2020

Indhold

1	Resumé	1
2	Indledning	1
3	Krav	2
4	Analyse	3
4.1	Use case beskrivelser	3
4.2	Fully dressed Use case	4
4.3	Systemsekvensdiagram	5
4.4	Designklassediagram	6
5	Design	7
5.1	Klassediagram	7
5.2	GUIController	7
5.3	ChanceCardController	7
5.4	GameBoard	7
5.5	Sekvensdiagram	9
6	Implementering	10
6.1	Actors	10
6.2	ActorController	10
6.3	GUIController	10
6.4	ChanceCardController	10
6.5	GameBoard	10
6.6	Utility	11
6.7	StringHandler	11
7	Brug	12
8	Test	13
8.1	Test cases	13
8.2	Dokumentation for code coverage	16
9	Konklusion	17
	Litteratur	I

1 Resumé

2 Indledning

IOOuteractive har stillet os en opgave, som går ud på at skulle lave et Monopoly Junior spil. Vi skal selv vurdere hvad der er vigtigst for at spille skal kunne køre, hvilket også betyder at regler må udelades.

3 Krav

Spillet/programmeringsdel

- Monopoly Junior
- Klassesdesign
- Felter (Tile omdøbes til Field) (identificer for felt type)
 - Besøg i fængsel
 - Gå i fængsel
 - Chancefelt
 - Parkeringsplads
 - Start
 - Grunde
 - Farvekode / gruppering
 - Købsværdi
 - Leje
 - Ejer (0 = banken)
- Spillebrikker / biler
- Antal spillere 2-4
- Spiller
 - Alder/fødselsdato
 - Liste af skøder
 - Spiller kan kun have 12 grunde
 - Har en konto - den kan ikke blive negativ
 - Bil med farve - vælges i starten
 - ChanceCard[]
- Deedlist - skøde liste
 - Int array[Total tiles]
- Bank
 - Laves som Spiller
 - 16 skøder fra start
 - 90 Monopoly penge fra start
- ChanceKort
 - ChanceCard[] til draw pile
 - Hvis tom, bland discard pile til at lave en ny draw pile
 - ChanceCard[] til discard pile
 - Kan interagere med andre spillere end den nuværende
 - Kan gemmes
 - Har valgmuligheder
- Man går i ring på brættet.
- Brættet skal se ud som Monopoly Junior spillepladen

Analyse- og designdokumentation

- Krav til artifacts
 - Kravliste
 - Use case diagram
 - Use case beskrivelser - mindst én beskrives fully dressed
 - Domænemodel
 - Systemsekvensdiagram
 - Sekvensdiagram
 - Designklassediagram
 - Klassediagram
- GRASP patterns
- Tydelig sammenhæng mellem kravliste, analyse- og designdokumentation, og implementering.

Implementering

- Lav passende konstruktører.
- Lav passende get og set metoder.
- Lav passende toString metoder
- Lav GameBoard klasse
- Tilføj en toString metode der udskriver alle felterne i arrayet.
- Lav spillet ud fra klasserne beskrevet her og de der er beskrevet i CDIO2.

Dokumentation

- Forklar hvad arv er.
- Forklar hvad abstract betyder.
- Forklar hvad det hedder hvis alle feldklasserne har en landOnField metode der gør noget forskelligt.
- Dokumentation for test med screenshots.
- Dokumentation for overholdt GRASP.

Test

- Lav mindst tre testcases.
- Lav mindst én JUnit test.
- Lav mindst én brugertest.

Versionsstyring

- Versionsstyret rapport (.tex filer i egen branch, merges ind sidst).
- Git repo / afleveres som link.
- Git repo i IntelliJ guide.

4 Analyse

4.1 Use case beskrivelser

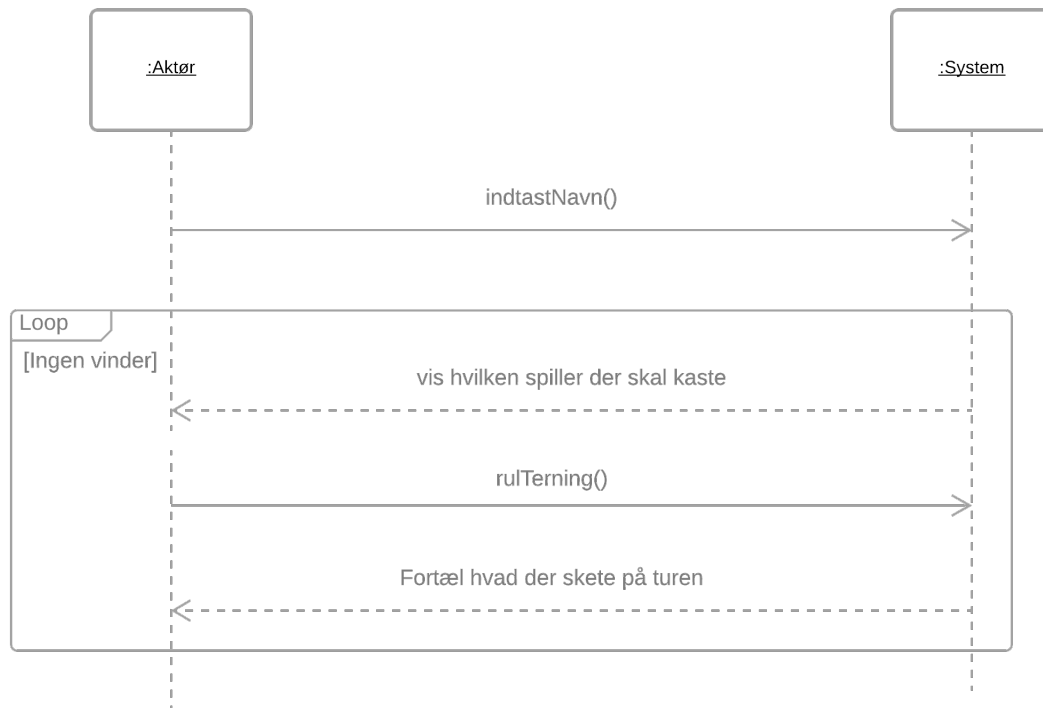
- Spil spillet.
Spilleren spiller spillet.

Sub use case beskrivelser:

- Indtast navn.
Spilleren indtaster sit navn.
- Vælg farve.
Spilleren vælger imellem et udvalg af farve.
- Rul terning.
Spilleren ruller med terningerne.

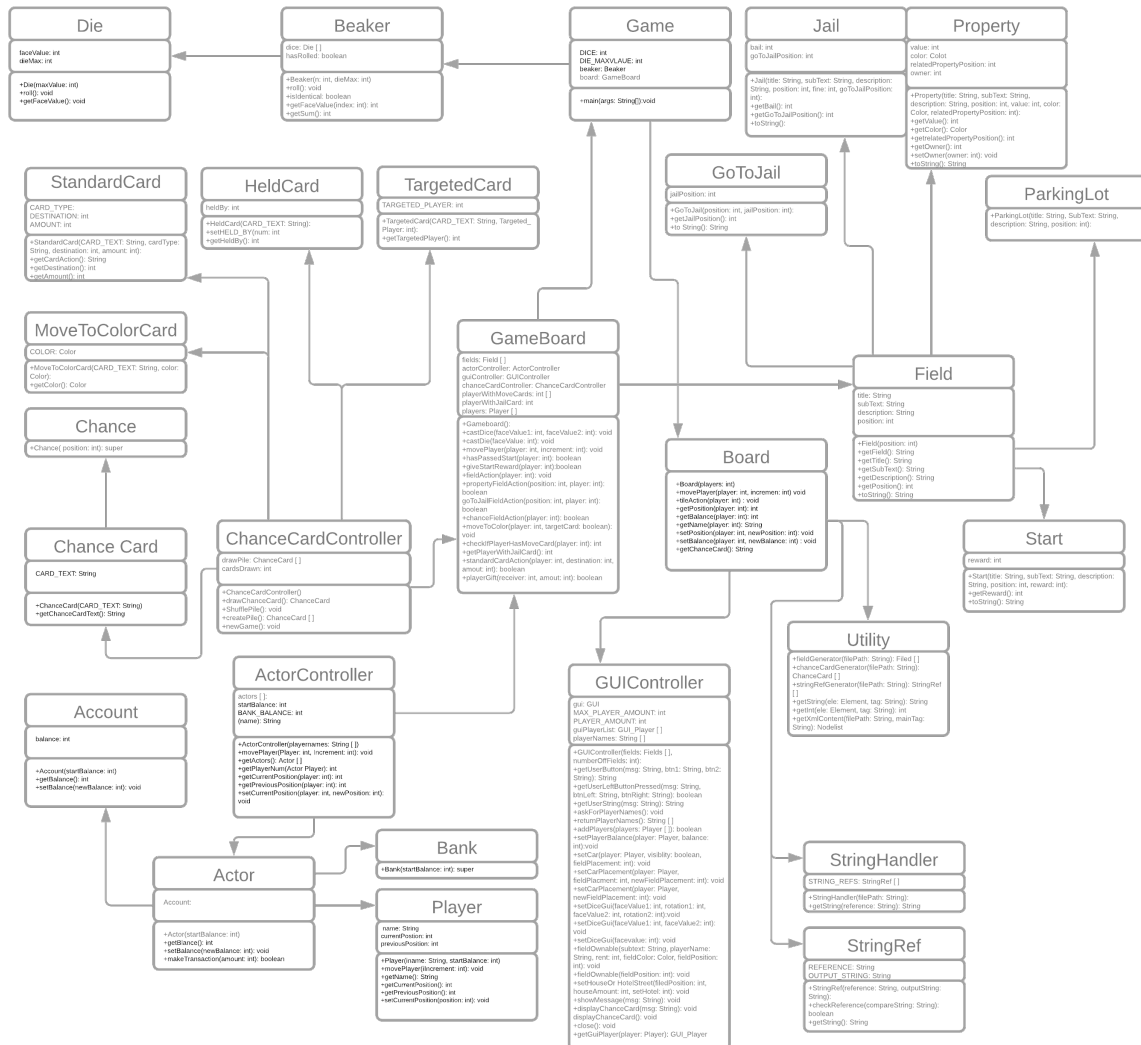
4.2 Fully dressed Use case

Use Case sektion	Beskrivelse
Navn	Start spillet
Scope	Matador Junior spil
Level/niveau	At starte spillet. GUI'en åbner med de rigtige felter, og man kan dernæst oprette spillere.
Primær aktør	Brugere af Matador Junior spillet
Andre interessanter	Sælger af spillet
Forudsætninger/Preconditions	At man er mindst 2 spillere, og man har en pc det kan spilles på
Succeskriterier/postconditions	Start spillet, spil spillet, og spil indtil der er en der vinder
Vigtigste <u>successscenarie</u>	Spilleren starter spillet og oprette det antal af spiller de gerne vil være (min. 2, max 4), og spiller spillet indtil <u>en</u> er gået fallit.
Uvidelser	Lave det til et almindelig spillet Matador i stedet for Junior versionen
Special requirements	En pc med java 14.0+



4.3 Systemsekvensdiagram

4.4 Designklassediagramm



5 Design

5.1 Klassediagram

Klassediagrammet giver overblik over vores klasser i projektet. Vi har indset, at vi skal bruge mange objekter, der er variationer af samme type - f.eks. felter på spillepladen. Derfor gør vi brug af nedarvning. I tilfældet med felter er nedarvning smart, da nogle felter kan ejes af en spiller, mens andre ikke kan. Nogle felter har ingen konsekvenser (Parkeringsplads, fængsel), mens andre har (chance, start, ryk i fængsel, ejendomme). Vi har valgt at bruge ordet **Actor** som generalisering. Dette skal ikke misforstås som aktør i kontekst af UML. I Monopoly Junior har banken egenskaber der til forveksling ligner en spillers, derfor har vi valgt at bruge nedarvning til dette, da vi bruger de samme methods til banken såvel som spilleren. Utility er blevet flyttet udenfor diagrammet, da den ikke passer godt ind ellers. Utility bør muligvis omdøbes til Reader eller lignende, da det er mere beskrivende for hvad den gør - nemlig læser tekst- og XML-filer for at hente data til brug i andre dele af programmet. GameBoard og ChanceCardController skal begge bruge Utility for at generere adskillige objekter fra XML filer. GUIController har behov for at læse fra forskellige tekstfiler, og det bruger vi også Utility til.

5.2 GUIController

Klassen **GUIController** er den klasse som sørger for alt kommunikation til GUI'en, så vi kan nemmere holde styr på det hele ift. GRASP. Dvs. der er kun 1 fil, man skal kigge i, når man vil kommunikere med GUI'en. Klasse skal derud over være henholdsvis minimalistisk i den forstand af, at der skal helst ikke være mere end en 5-10 linjer kode i hver metode (nogle metoder undtaget), da det mest bare er kald, som går videre i systemet.

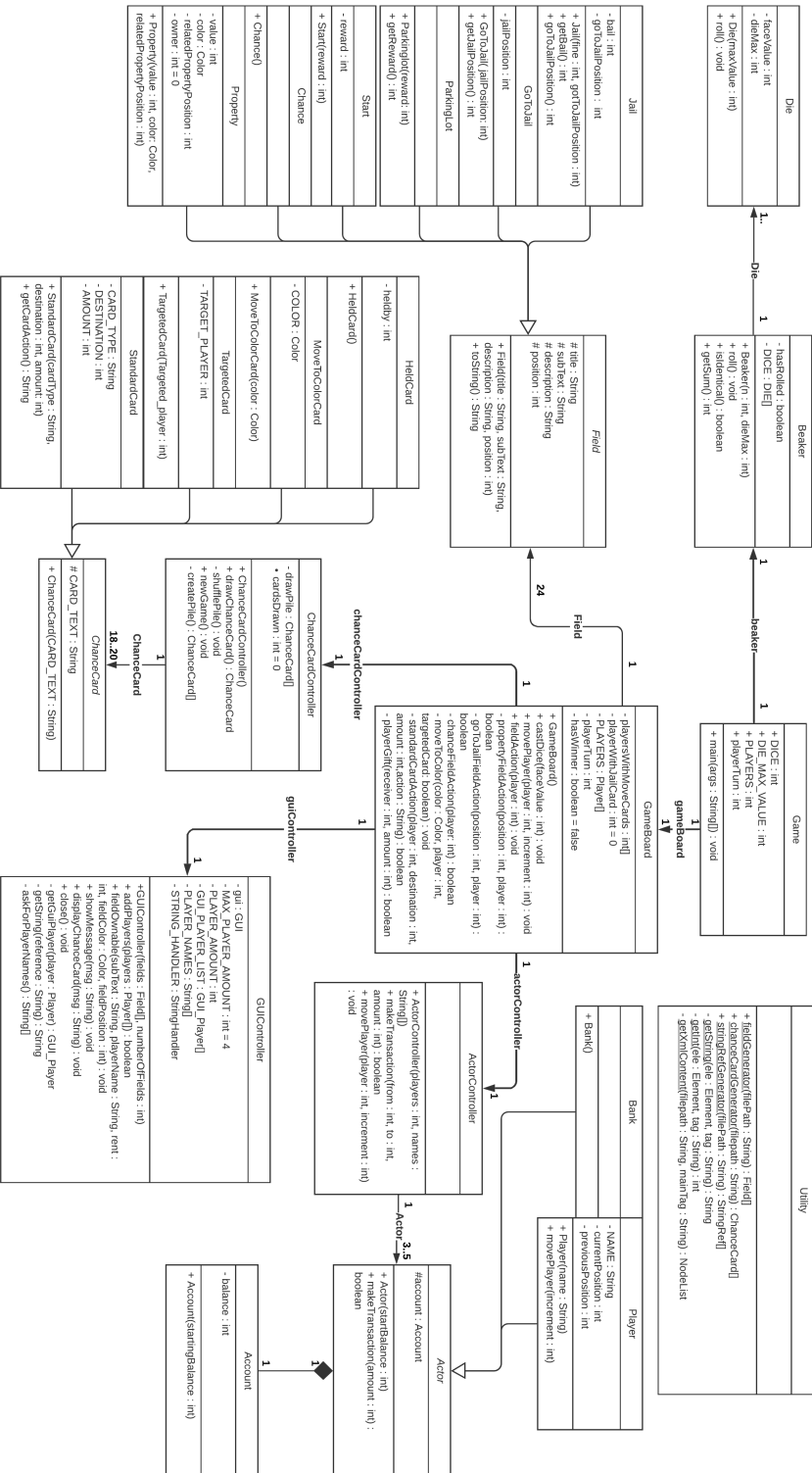
5.3 ChanceCardController

ChanceCardController bliver brugt til at holde styr på alle chancekort(dvs. holde en bunke kort, give mulighed for at trække et kort og blande bunken). Vi har valgt at lave en controller til dette, da det giver bedre mulighed for at overholde GRASP principperne.

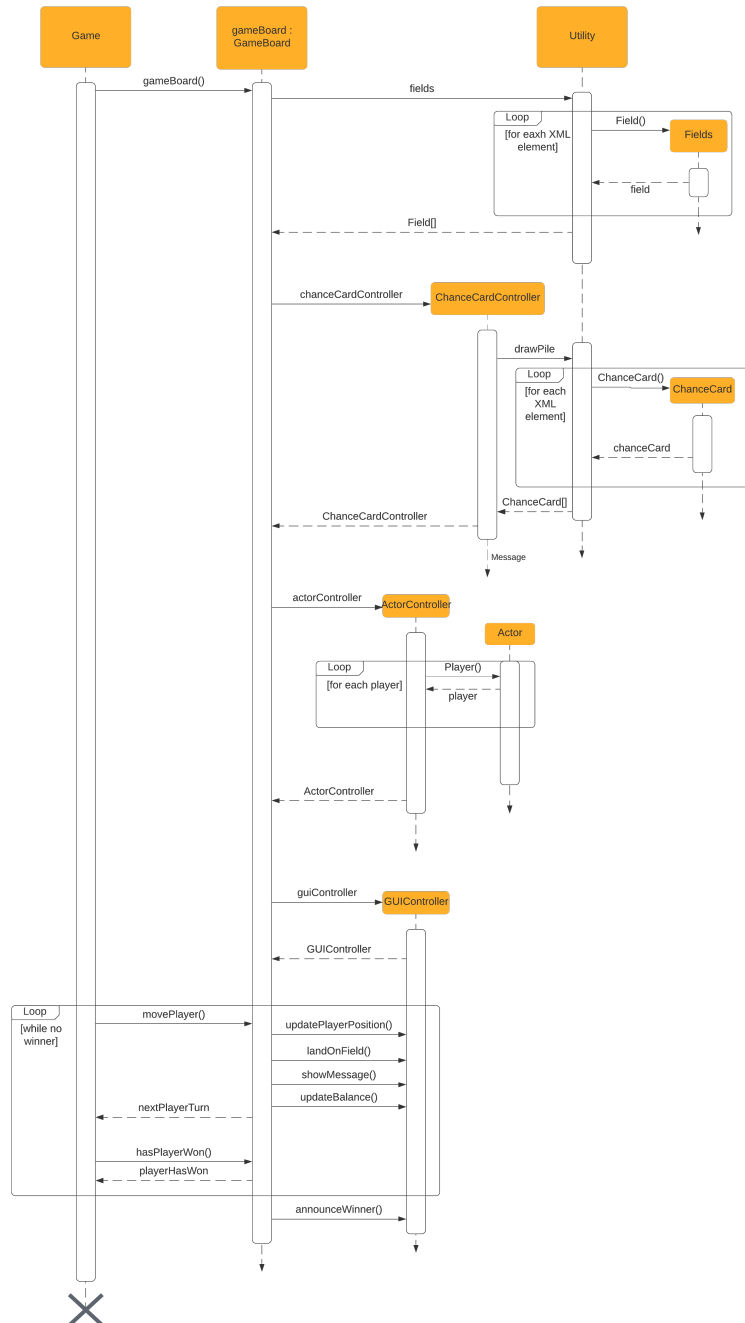
5.4 GameBoard

Denne klasse har vi bestemt til at være den centrale klasse der udfører det meste af handlingerne nødvendige for en spillerunde. **GameBoard** indeholder derfor både en **ActorController** til spillere og banken, en **GUIController** til at styre bruger interfacet samt et **Field** array. **GameBoard** fungerer som controller til felter, og indeholder derfor koden eksekveres for hver af felttyperne.

Main bruger derfor kun en **Beaker** og en **GameBoard** klasse.



5.5 Sekvensdiagram



6 Implementering

6.1 Actors

Da spillet nu skal udvides til et matadorspil har vi brug for en bank. Til dette er det oplagt at bruge nedarvning, da **Bank** og **Player** har mange metoder til fælles. Vi har derfor lavet en **Actor** som superklasse til de fælles attributter og metoder. Den er selvfølgelig **abstract** da vi ikke vil have lavet nogle instanser af den. **Bank** og **Player** klasserne nedarver således fra **Actor** klassen, så de begge kan få en start balance samt lave transactions.

6.2 ActorController

I metoden **makeTransaction** sikrer vi low coupling og high cohesion ved kun at tilgå de forskellige actors, transaktionen skal ske mellem gennem **ActorController**, mens det eneste **Actor** klassen gør når den bliver kaldt af **makeTransaction** er at udregne og sætte den nye balance for den instans af **Actor** som kalder metoden.

6.3 GUIController

Når man opretter et object af **GUIController**, skal der sendes en list af **Fields** med og antal af **Fields**, da det er **GUIController**, som skal starte GUI'en, og oprette de forskellige felter til spillet. Inde i selve constructoren af klassen, vil der så være en switch case, som sørger for at tildele de forskellige felter den rigtige type af **Field** i GUI'en. Derefter kan man kommunikere med GUI'en via det **GUIController** object man har lavet. **GUIController** opretter også spillere på den måde, at den spørger spillere om deres navn, og om man vil oprette flere spiller, derudover tjekker den også på, om man er noget max. Antal spiller, eller om man er under min. Antal spillere. Alle spillernavne bliver derefter smidt ind i et **Array**, som man kan hente med metode `"returnPlayerNames()"`.

6.4 ChanceCardController

Når man opretter et object a **ChanceCardController**, kommer dette til at indeholde et dæk med 20 chancekort. Man har mulighed for at trække et kort(som bliver returneret, da **GameBoard** skal håndtere hvad kortet gør), ved hjælp af **drawChanceCard()** metoden. Dækket bliver kørt igennem fra start til slut, og hvis der bliver kaldt **newGame()** bliver dækket blandet, og variable **cardsDrawn** bliver sat til 0, for at sørge for at man starter forfra med bunken.

6.5 GameBoard

Når **GameBoard** instantieres skal den bruge en liste af fields, hvilket den får fra **Utility** pakken, som læser felterne fra en xml fil. I konstruktøren bliver **GUIController** også instantieret, så et nyt vindue bliver lavet.

Den centrale metode i `GameBoard` kan siges at være `fieldAction`, da det er den metode der eksekveres når spilleren lander på et spil og skal udføre en bestemt handling ud fra feltets type. Der er derfor forskellige metoder for de forskellige felter, som bliver kørt af `fieldAction`.

Hver gang `fieldAction` bliver kaldt tjekkes der om nogen af spillerne er gået fallit ved at se om nogen af transaktionerne ikke er gået igennem.

Hvis spilleren lander på et `Chancefelt` skal der trækkes et chancekort som siger hvad spilleren skal gøre. I nogle tilfælde skal spilleren flytte sig frem på næste træk. Derfor bliver det gemt, og skal tjekkes for hver tur spillerne har.

I løbet af dette bliver `GUIController` også brugt til at ændre brættet i overensstemmelse med hvad der sker i spillets logik.

6.6 Utility

`Utility` indeholder methods til at hente data fra XML filer. Vi bruger `Utility` til at importere data til `ChanceCard` og `Field` objekterne, som skal bruges i programmet, samt til at lave `StringRef` objekter der skal benyttes af `StringHandler`.

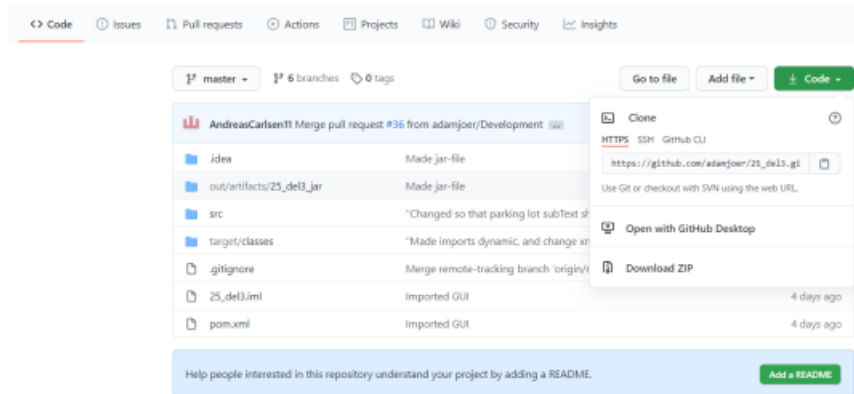
6.7 StringHandler

Denne object controller er lavet for at kunne søge på en `String` reference for at kunne hente den søgte `String` til brug hvor det er relevant (typisk i `GUIController`, der sender beskeder til udskrift i GUIen). Formålet med dette er at kunne skifte sproget fra engelsk til eksempelvis dansk ved blot at oversætte en række strenge i overskuelige filer (her XML).

Download af GIT Repository

Inden på repositoryets startside er der en knap, hvor man kan download koden, gøre følgende:

1. klik på download pilen (det felt hvor der står "Code").
2. Kopier URL'en til selve repositoryet.



Derefter går du ind i den mappe, hvor du ønsker repositoryet skal ligge via cmd, hvor du derefter bruge følgende commando for at kopier repositoryet til mappen:

"git clone URL"

URL -> den url der kan ses i ovenstående billede.

```
C:\Users\Jens\Desktop\git repo>git clone https://github.com/adamjoer/25_del3.git
```

Derefter skal mappen (som i det her tilfælde vil hedde 25_del3) åbnes i et IDE, og så er projektet klar til brug.

Spillet kan blive spillet på følgende måde, da der eksistere en .jar fil:

1. Åben cmd
2. cd indtil følgende dir (når du står i roden af projektet):
 - a. `java -jar out/artifacts/25_del3_jar/25_del3.jar`
3. Spillet går i gang, og er klar til at blive spillet.

Billede af kommando i brug:

```
C:\Users\Jens\Desktop\CDIO0_del3\25_del3>java -jar out/artifacts/25_del3_jar/25_del3.jar
```

7 Brug

8 Test

8.1 Test cases

Test case ID	TC1
Resumé	Test for mange spillere
Krav	Ikke specificeret
Forudsætninger/Preconditions	En spiller har startet spillet
Succeskriterier/postconditions	<u>Spillerne</u> er klar til at spille <u>spillet</u>
Test procedure	<ol style="list-style-type: none"> 1. Start spillet 2. Opret 4 spillere 3. Opret den femte spiller
Test data	<div>▼</div> Player1 navn = "Dres" Player2 navn = "Mik" Player3 navn = "Gurli" Player4 navn = "And" Player5 navn = "Ekstra spiller"
Forventede resultat	Gul'en skriver en streng til spillere, hvori der står at de har opnået max antal spillere, og at de kan vælge at fortsætte spillet eller stoppe det.
Faktiske resultat	Gul'en skrev strengen til spillere, og der er teste både for, at man kan stoppe spillet, men også at man kan fortsætte.
Status	Passed
Testet af	Jens
Dato	25-11-2020
Test miljø	IntelliJ IDEA 2020 2.1 Revision 9563f4f3, Bygget den 26 November, 2020 Windows 10 home

Test case ID	TC2
Resumé	Test for lidt spillere
Krav	Ikke specificeret
Forudsætninger/Preconditions	En spiller starter spillet
Succeskriterier/postconditions	Spilleren er klar til at spille spillet
Test procedure	1. Start spiller 2. Tryk "nej" til at oprette flere spillere
Test data	Ingen
Forventede resultat	GUI'en skriver til spilleren, at han ikke kan spille uden der er mindst 2 spillere
Faktiske resultat	GUI'en skriver at spillet ikke kan gå i gang uden mindst 2 spillere
Status	Passed
Testet af	Jens
Dato	26-11-2020
Test miljø	IntelliJ IDEA 2020 2.1 Revision 9563f4f3, Bygget den 26 November, 2020 Windows 10 home

Test case ID	TC2
Resumé	Test for lidt spillere
Krav	Ikke specificeret
Forudsætninger/Preconditions	En spiller starter spillet
Succeskriterier/postconditions	Spilleren er klar til at spille spillet
Test procedure	1. Start spiller 2. Tryk "nej" til at oprette flere spillere
Test data	Ingen
Forventede resultat	GUI'en skriver til spilleren, at han ikke kan spille uden der er mindst 2 spillere
Faktiske resultat	GUI'en skriver at spillet ikke kan gå i gang uden mindst 2 spillere
Status	Passed
Testet af	Jens
Dato	26-11-2020
Test miljø	IntelliJ IDEA 2020 2.1 Revision 9563f4f3, Bygget den 26 November, 2020 Windows 10 home

8.2 Dokumentation for code coverage



9 Konklusion

Vi har leveret et produkt, der forsøger at leve op til principper. Vi erfarer, at vores tilgang til udspecificering af krav har spændt ben for vores udviklingsarbejde, og denne erfaring tager vi med, således at vi fremover kan arbejde bedre efter et iterativt princip og med fokus på essentielle funktionaliteter først. Must have, should have, would have, could have - i den rækkefølge.

Vi har til trods for dette leveret et produkt, der stadig lever op til mange af de krav, der er blevet stillet - selvom vi har mangler.

Litteratur

CDIO del 1
CDIO del 2