

CDIO-3

Brætspil



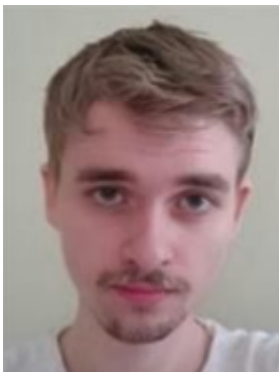
Lukas Bek (s153475)



Lars Sørensen (s144859)



Sebastian Hoppe (s154306)



William Wiberg (s153368)



Magnus Haakonsson (s153947)

Gruppe: 13

Afleveres: 28-11-2015

Opgavetype: Rapport,
Softwareteknologi

Fag: 02312-02313-02314-02315

Rapporten består af 28 antal sider
Rapporten er afleveret via Campusnet

Timeregnskab

CDIO-del 3							
Time-regnskab							
Dato	Deltager	Design	Impl.	Test	Dok.	Andet	I alt
20-10-2015	Lars	5	27	5	5	1	43
20-10-2015	Magnus	8	27	2	5	1	43
20-10-2015	Lukas	8	24	5	7	1	45
20-10-2015	William	4	24	9	5	1	43
20-10-2015	Sebastian	4	28	7	2	1	42

Indholdsfortegnelse

Timeregnskab	2
Indledning	5
Kundens Vision	6
Kravspecifisering	6
Konfigurationsstyring	7
Use cases	8
Use case diagram	8
Use cases	8
Analyse	14
Domænemodel	14
BCE-model	15
Spillet's logik	16
Spillet's regler	16
Ture	16
Hvordan spillet vindes	16
Design	17
Design Sekvens Diagram	17
Design klasse diagram	18
Test	19
Negativ test af gyldigt antal spillere	19
Negativ test af gyldige variable for antal spillere	20
Arv	21
Abstract	21
Klasser:	21
metoder:	21
LandOnField metoden	21
GRASP	22
Creator	22
Information Expert	22
Low Coupling	22
High Cohesion	22
Controller	22
Konklusion	23

Bilag 24

 Bilag 1 24

Indledning

Objektivet i denne opgave er at omdanne en given vision til krav, og derfra analysere de krav, der er fundet. Kravene skal analyseres for at se hvilke, der er funktionelle og hvilke der er non-funktionelle - de funktionelle er de vigtigste da disse har en effekt på hvordan spillet skal kodes og hvordan spillet skal fungere, de non-funktionelle har stadig en effekt på hvordan spillet kommer til at fungere, men den er meget mindre, da det er sådan noget som f.eks. spillet er brugervenligt. Kravene skal derfra laves om til skemaer, modeller og use-cases. Ud fra analysen kan koden herefter dannes. Mens at koden bliver lavet, skal den versioneres til et centralt repo, da kunden gerne vil kunne følge med i udviklingen af deres program. Når koden er færdig skal den testes, så brugerens og spillets krav efterses for fejl og mangler.

Kundens Vision

Kravspecifisering

Krav:

Krav fra tidligere projekt:

1. Spilles på maskinerne i DTU uden forsinkelser
2. To terninger, som nemt kan skiftes
3. Spilleren lander på et forudbestemt felt, som har en forudbestemt værdi/effekt
4. Spillet skal let kunne oversættes til andre sprog
5. Spiller og hans pengebeholdning skal kunne bruges i andre spil

Nye og redigeret krav:

1. Spil mellem 2-6 personer
2. Starter med 30.000
3. Vinder er sidste mand som ikke er bankerot (Spillet slutter når der findes en vinder)
4. Spilleren er ude af spillet når de når under 0
5. Når spiller lander på et felt, skal de kunne rykke videre fra det felt
6. Der skal være en spilleplade
7. Man går i ring på brættet

Non-funktionelle krav

1. Projektets udførelse skal kunne ses i et GIT repository

Krav til dokumentation

1. Tests, der viser at koden er afprøvet og virker JUnit (heriblandt)

Kommentar til krav

Dette projekt er en videre-byggelse af det tidligere CDIO-2 projekt. De oprindelige krav gælder derfor stadig i det nye projekt og samtidigt er der lavet flere tilføjelser og redigeringer til de tidligere krav. Kravene er blevet delt op for nemmere at kunne overskue de nye krav, da de tidligere krav allerede eksisterer fra programmets opbygning.

Konfigurationsstyring

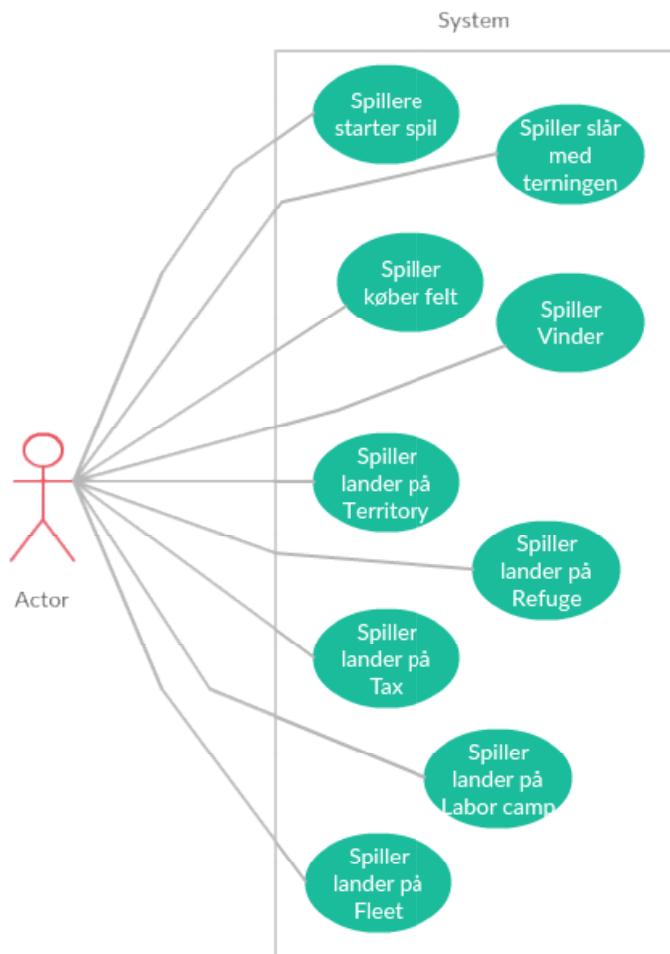
I dette projekt er udviklings- og produktionsplatformen ens og kan beskrives som en enkelt platform. Platformen inkluderer følgende:

- OS: Windows 8.1
- Java Version 8 update 66
- Eclipse Version: Mars Release (4.5.0)
- Gui.jar Version 2.0.1

For at kunden skal kunne følge med i projektets udvikling, bliver projektet versioneret med GitHub. Link: https://github.com/Quaade94/13_CDIO_3

Use cases

Use case diagram



Figur 1 Use case diagram

Use cases

1. Spillere starter spil
2. Spiller slår med terningen
3. Spiller køber et felt
4. Spiller vinder
5. Spiller lander på Territory
6. Spiller lander på Refuge
7. Spiller lander på Tax
8. Spiller lander på Labor camp
9. Spiller lander på Fleet

Spiller starter spil
ID: 01
Kort beskrivelse: Spillere vil starte spillet
Primær Aktør: Spiller
Sekundære Aktører: Programmet selv
Forudsættelser: Der er nok spillere til at starte spillet
Primært Flow: 1. Spilleren med programmet starter spillet
Efterfølge: Spillet er nu i gang
Alternative Flows:

Spiller slår med terninger
ID: 2
Kort beskrivelse: Spiller slår med terninger
Primær Aktør: Spiller
Sekundære Aktører: Programmet selv
Forudsættelser: Spillet er startet
Primært Flow: 1. Spiller trykker på knap for at rulle terningerne. 2. Resultat af slaget vises. 3. Resultat bliver lagt sammen. 4. Spiller rykker antal felter som resultatet gav. 5. Gemmer placering på pladen til næste tur.
Efterfølge: Turen skifter til den anden spiller
Alternative Flows:

Spiller køber felt
ID: 3
Kort beskrivelse: Spilleren har slået og lander derefter på et felt, som er til salg
Primær Aktør: Spiller
Sekundære Aktører: Programmet selv
Forudsættelser: Spillet er startet, spiller står på et felt der er til salg
Primært Flow: <ol style="list-style-type: none"> 1. Spiller lander på felt der kan købes 2. Spiller betaler passende beløb for felt 3. Spiller bliver ejer af felt
Efterfølge: Næste spillers tur
Alternative Flow: <ol style="list-style-type: none"> 1) <ol style="list-style-type: none"> 1. Spiller lander på felt der kan købes 2. Spiller vil betale, men har ikke nok penge til rådighed 3. Feltet er fortsat uden ejer 2) <ol style="list-style-type: none"> 1. Spiller lander på felt der kan købes 2. Spiller har penge, men vælger ikke at købe feltet 3. feltet er fortsat uden ejer

Spiller vinder
ID: 4
Kort beskrivelse: En anden spiller går bankerot og der er kun én spiller tilbage
Primær Aktør: Spiller
Sekundære Aktører: Programmet selv
Forudsættelser: Spillet er startet, spiller går bankerot
Primært Flow: <ol style="list-style-type: none"> 1. Der er kun én spiller tilbage 2. Tilbageværende spiller vinder
Efterfølge: Spillet slutter
Alternative Flow:

Spiller lander på Territory felt
ID: 5
Kort beskrivelse: Spilleren lander på et Territory felt
Primær Aktør: Spiller
Sekundære Aktører: Programmet selv
Forudsættelser: Spillet er startet, det er spillerens tur
Primært Flow: <ol style="list-style-type: none"> 1. <Include spiller slår med terninger> 2. Spiller lander på Territory felt 3. Spiller får tilbud om at købe feltet
Efterfølge: Næste spillers tur
Alternative Flows: <ol style="list-style-type: none"> 1. <Include spiller slår med terninger> 2. Spiller lander på Territory felt 3. Spiller betaler ejer af felt passende beløb for feltet

Lander på Refuge
ID: 6
Kort beskrivelse: Spilleren har slået og lander derefter på et Refuge felt
Primær Aktør: Spiller
Sekundære Aktører: Programmet selv
Forudsættelser: Spillet er startet, spiller står på et Refuge felt
Primært Flow: <ol style="list-style-type: none"> 1. <Include spiller slår med terninger> 2. Spiller lander på Refuge felt 3. Spiller modtager passende beløb
Efterfølge: Næste spillers tur
Alternative Flow:

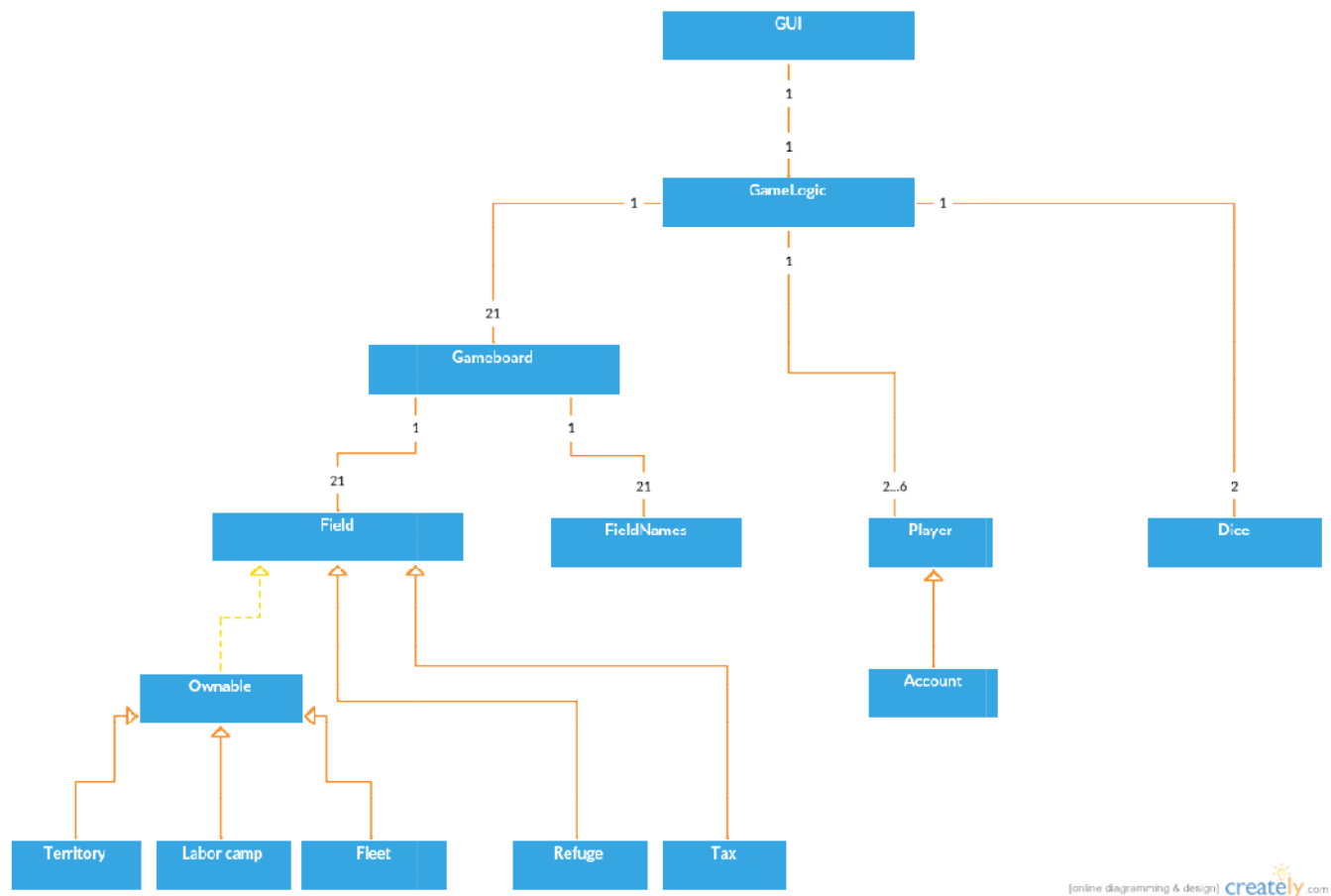
Lander på Tax
ID: 7
Kort beskrivelse: Spilleren har slået og lander derefter på et Tax felt
Primær Aktør: Spiller
Sekundære Aktører: Programmet selv
Forudsættelser: Spillet er startet, spiller står på et Tax felt
Primært Flow: <ol style="list-style-type: none"> 1. <Include spiller slår med terninger> 2. Spiller lander på Tax felt 3. Spiller betaler et fast beløb
Efterfølge: Næste spillers tur
Alternative Flow: <Include spiller slår med terninger> Spiller lander på Tax felt Spiller betaler 10% af egen pengebeholdning

Lander på Labour camp
ID: 8
Kort beskrivelse: Spilleren har slået og lander derefter på et Labour camp felt
Primær Aktør: Spiller
Sekundære Aktører: Programmet selv
Forudsættelser: Spillet er startet, spiller står på et Labour camp felt
Primært Flow: <ol style="list-style-type: none"> 1. <Include spiller slår med terninger> 2. Spiller lander på Labour camp felt 3. Spiller bliver tilbudt at købe felt
Efterfølge: Næste spillers tur
Alternative Flow: <Include spiller slår med terninger> Spiller lander på Labour camp felt Spiller betaler ejer af felt passende beløb

Lander på Fleet
ID: 8
Kort beskrivelse: Spilleren har slået og lander derefter på et Fleet felt
Primær Aktør: Spiller
Sekundære Aktører: Programmet selv
Forudsættelser: Spillet er startet, spiller står på et Fleet felt
Primært Flow: <ol style="list-style-type: none"> 1. <Include spiller slår med terninger> 2. Spiller lander på Fleet felt 3. Spiller bliver tilbudt at købe felt
Efterfølge: Næste spillers tur
Alternative Flow: <Include spiller slår med terninger> Spiller lander på Fleet felt Spiller betaler ejer af felt passende beløb

Analyse

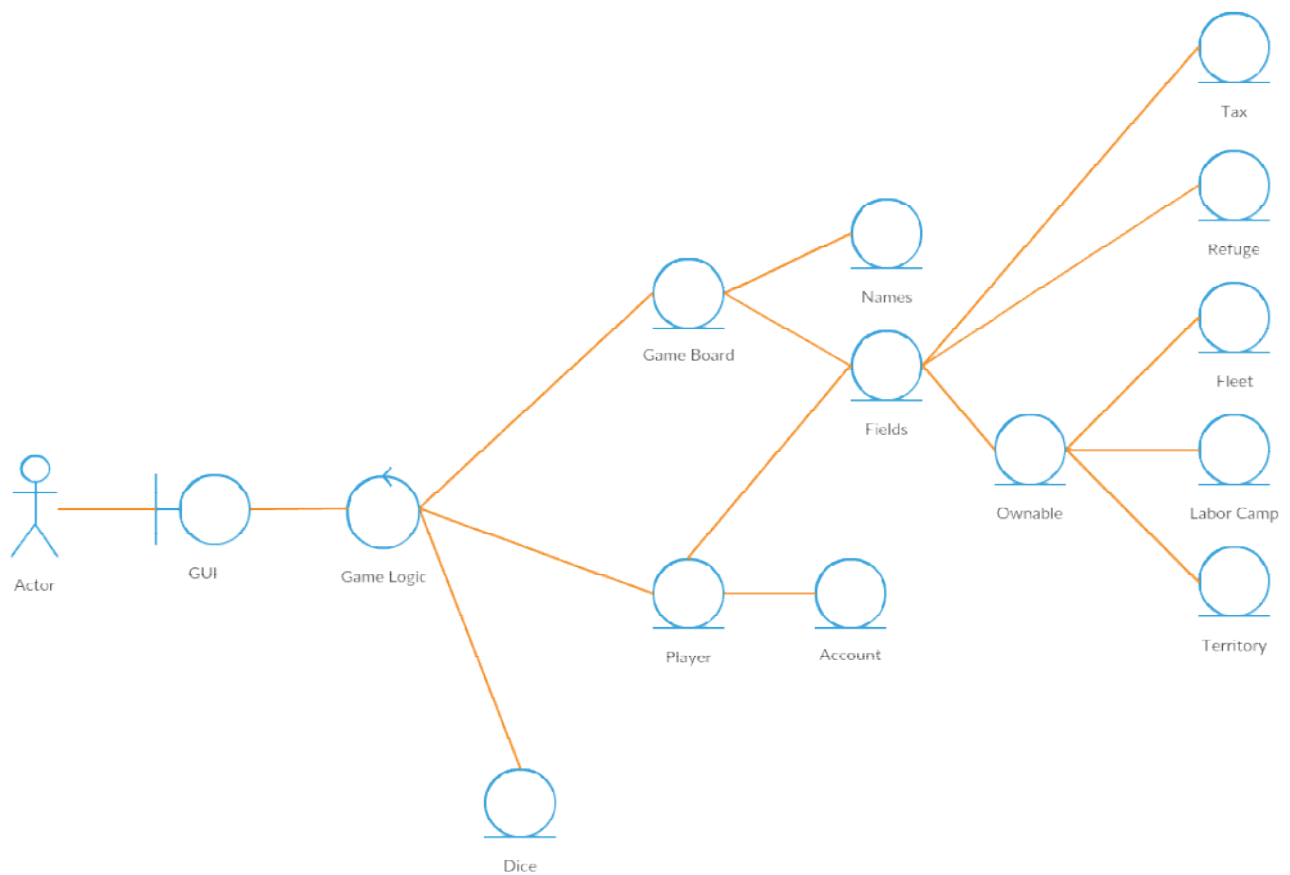
Domænemodel



Figur 2 Domænemodel

Her er domænemodellen, som er gruppens tidlige vision på hvordan projektet skal se ud. Ud fra dette vil der oprettes de respektive klasser i eclipse, som så vil fungere som byggestenene for projektet. Alt her er dog ting der kan ændres, alt efter hvad gruppen skulle støde ind i af problemer.

BCE-model



Figur 3 BCE model

Her vises hvordan de forskellige klasser sender information til hinanden, og hvordan denne information herefter sendes videre til en GUI ved hjælp af en controller. Dette kan give et overblik over hvordan informationen skal håndteres.

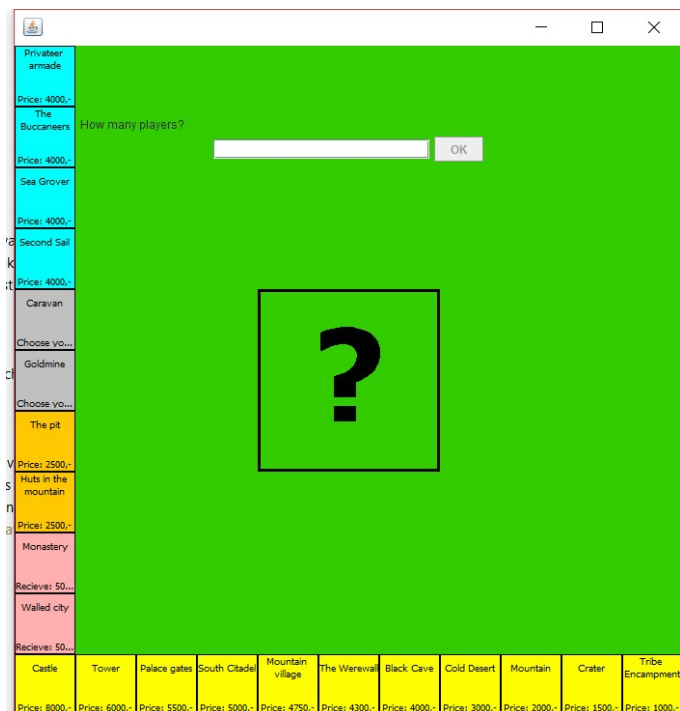
Spillets logik

I dette afsnit vil den logik, der ligger bag programmet forsøges forklaret.

Spillets regler

Ture

Til at begynde med, starter alle spillerne fra spillerplads 0, det vil sige den position ved siden af



Figur 4 Spillerbrættet

første plads på spillerpladen. Spillerpladen kører fra bunden af højre hjørne til bunden af venstre, herefter op til toppen af venstre hjørne og kører nu i ring (se evt. figur 4). Når den første spiller ruller med terningerne vil personen derfor lande på et felt, der er ens med de terningernes øjne lagt sammen. Herefter bliver det spiller nummer 2's tur, som ligesom spiller 1 ruller med terningen og rykker. Herefter fortsættes dette, hvis der er tilføjet flere spillere og når alle spillere har haft sin tur bliver det spiller 1 igen. Alt efter hvilke felter spilleren lander på, vil der ske nogle ting med hans pengebeholdning. Hvad felternes effekt gør, er beskrevet i opgavebeskrivelsen.

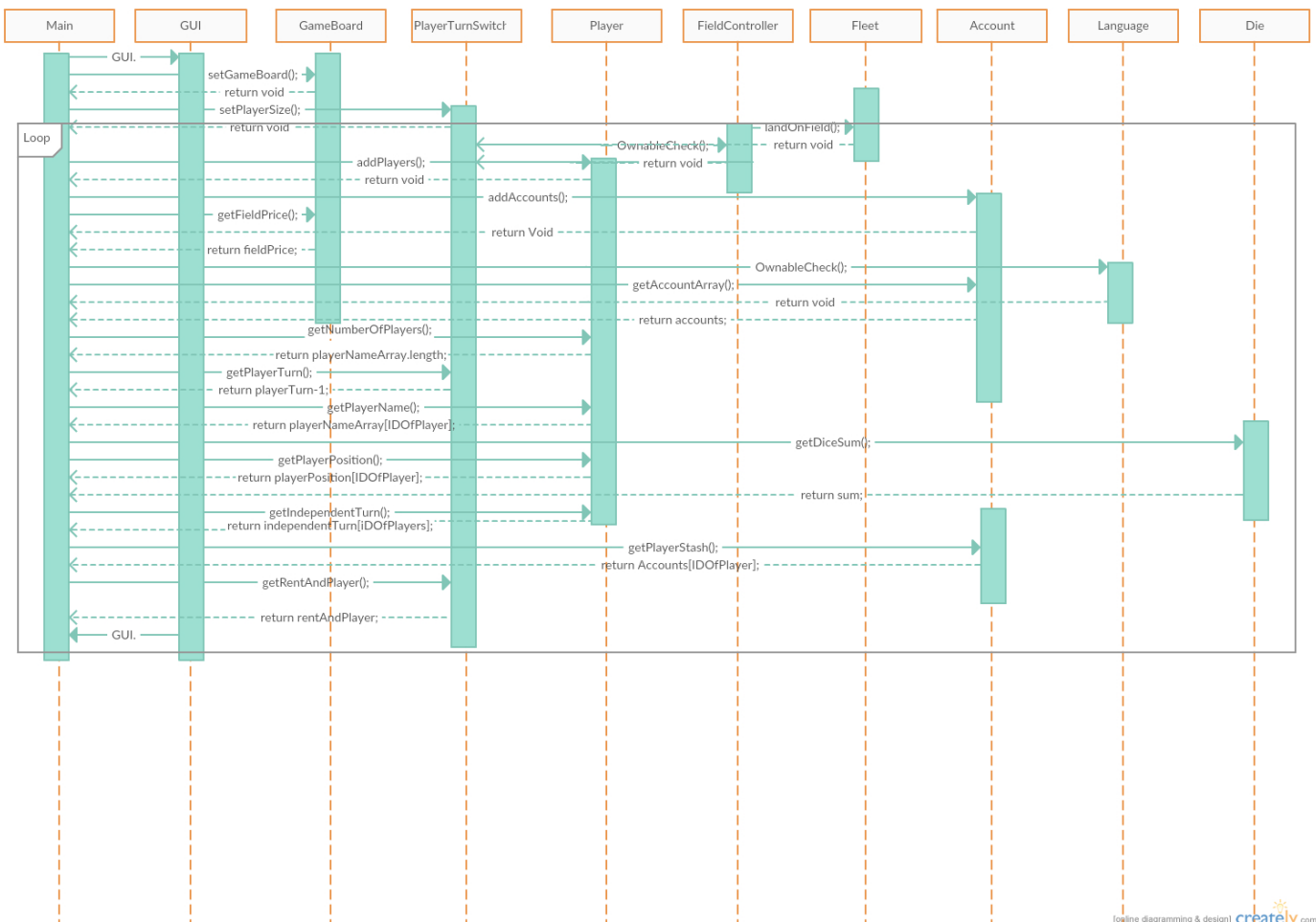
Hvordan spillet vinder

Alle spillerne har en pengebeholdning. Spillerne vil igennem spillet få ændret i deres pengebeholdning og når en spillers pengebeholdning ryger under nul, vil spilleren være "ude" af spillet. Spilleren vil dog stadigvæk eje alle de grunde han ejer, så spillere skal stadigvæk betale for at lande på spillerens grunde.

Når der kun er en spiller tilbage, vil denne spiller have vundet spillet.

Design

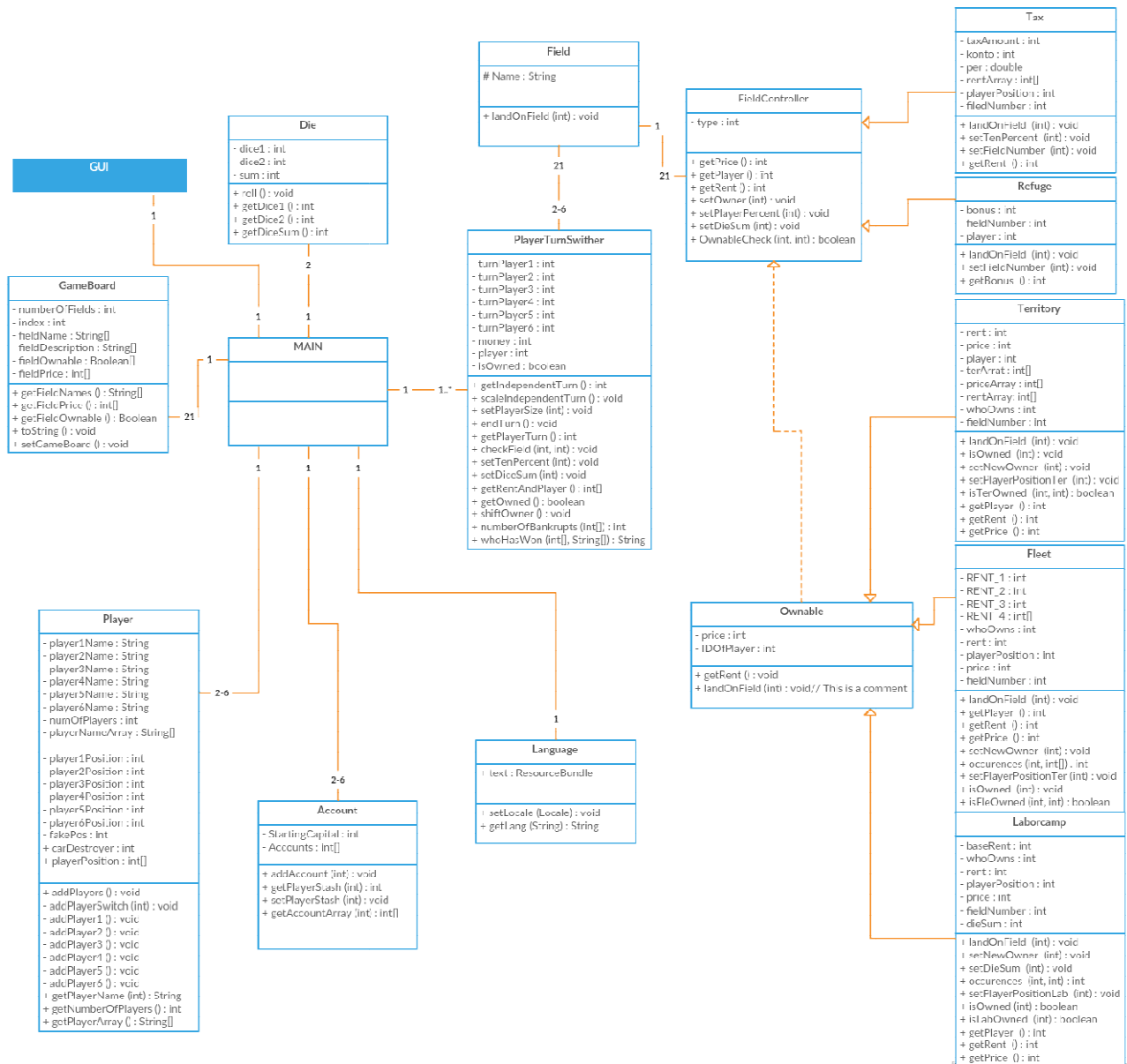
Design Sekvens Diagram



Figur 5 System Sekvens Diagram

Her vises hvad der sker i systemet i hvad der svarer til en "normal" tur. I denne instans lander spilleren på et "fleet" felt og i slutning går turen videre til næste spiller.

Design klasse diagram



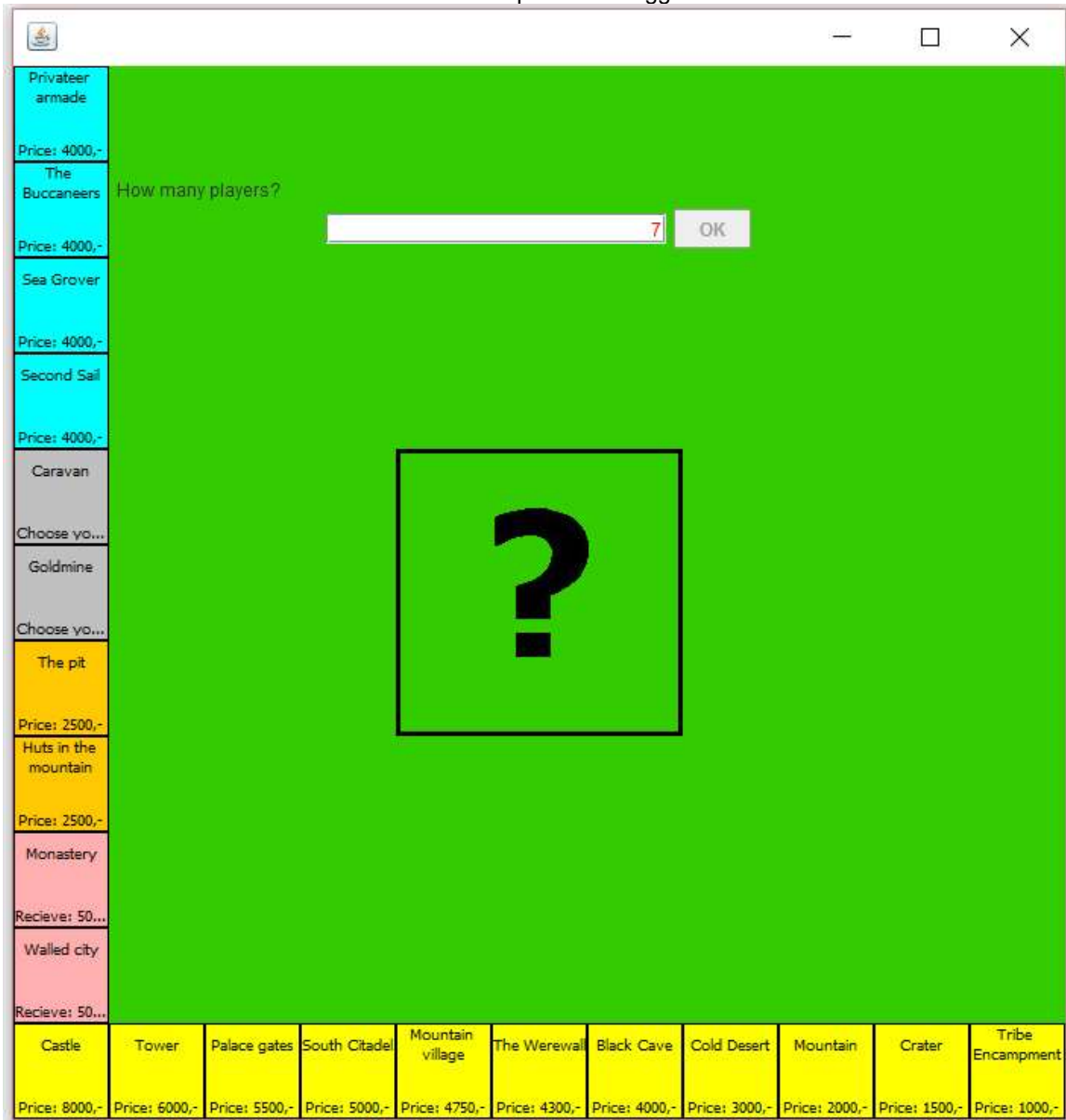
Figur 6 Design Klasse Diagram

Her er et diagram over alle de klasser, samt deres indhold, som findes i programmet. Der er kommet en del flere end der var i domænemodellen, hvilket er blevet fundet nødvendigt under udviklingsprocessen.

Test

Negativ test af gyldigt antal spillere

Her blev der testet hvad der sker hvis antallet af spillere ikke ligger inden for intervallet 2-6.

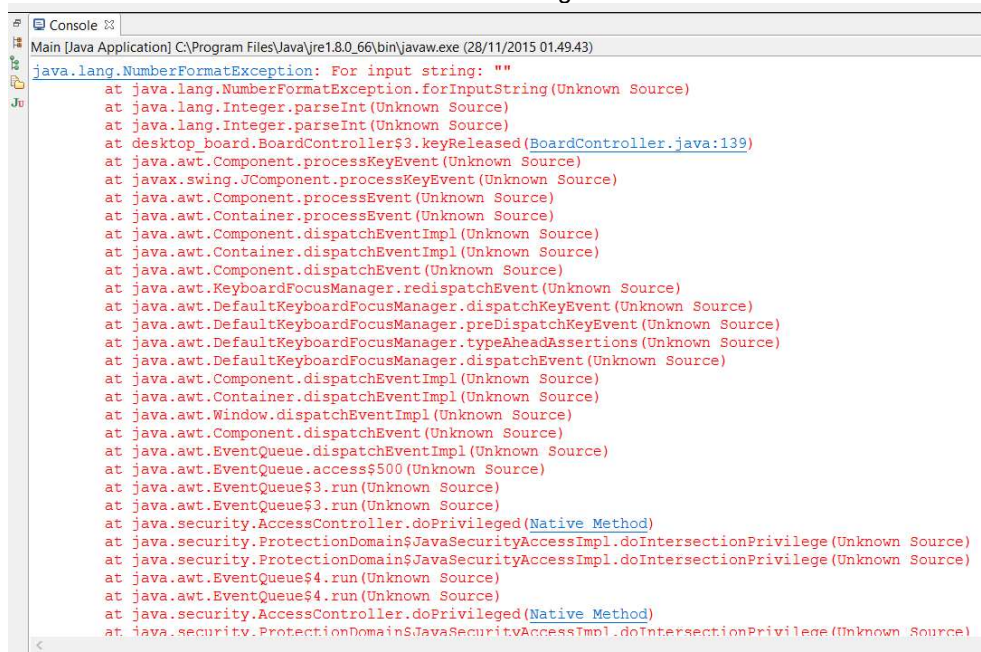


Figur 7 Viser at der ikke kan indtastes et tal, der ikke er mellem 2-6

Hvis antallet af spillere ikke sættes mellem 2-6, som var et af kravene, skifter skriften af tallet til rød og "ok" knappen fader ud og kan ikke anvendes.

Negativ test af gyldige variable for antal spillere

Her blev der testet hvad der skete hvis man forsøgte at indtaste en variabel som ikke var en int.



```
Console
Main [Java Application] C:\Program Files\Java\jre1.8.0_66\bin\javaw.exe (28/11/2015 01:49:43)
java.lang.NumberFormatException: For input string: ""
    at java.lang.NumberFormatException.forInputString(Unknown Source)
    at java.lang.Integer.parseInt(Unknown Source)
    at java.lang.Integer.parseInt(Unknown Source)
    at desktop_board.BoardController$.keyReleased(BoardController.java:139)
    at java.awt.Component.processKeyEvent(Unknown Source)
    at javax.swing.JComponent.processKeyEvent(Unknown Source)
    at java.awt.Component.processEvent(Unknown Source)
    at java.awt.Container.processEvent(Unknown Source)
    at java.awt.Component.dispatchEventImpl(Unknown Source)
    at java.awt.Container.dispatchEventImpl(Unknown Source)
    at java.awt.Component.dispatchEvent(Unknown Source)
    at java.awt.KeyboardFocusManager.redispatchEvent(Unknown Source)
    at java.awt.DefaultKeyboardFocusManager.dispatchEvent(Unknown Source)
    at java.awt.DefaultKeyboardFocusManager.preDispatchKeyEvent(Unknown Source)
    at java.awt.DefaultKeyboardFocusManager.typeAheadAssertions(Unknown Source)
    at java.awt.DefaultKeyboardFocusManager.dispatchEvent(Unknown Source)
    at java.awt.Component.dispatchEventImpl(Unknown Source)
    at java.awt.Container.dispatchEventImpl(Unknown Source)
    at java.awt.Window.dispatchEventImpl(Unknown Source)
    at java.awt.Component.dispatchEvent(Unknown Source)
    at java.awt.EventQueue.dispatchEventImpl(Unknown Source)
    at java.awt.EventQueue.access$500(Unknown Source)
    at java.awt.EventQueue$3.run(Unknown Source)
    at java.awt.EventQueue$3.run(Unknown Source)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.security.ProtectionDomain$JavaSecurityAccessImpl.doIntersectionPrivilege(Unknown Source)
    at java.security.ProtectionDomain$JavaSecurityAccessImpl.doIntersectionPrivilege(Unknown Source)
    at java.awt.EventQueue$4.run(Unknown Source)
    at java.awt.EventQueue$4.run(Unknown Source)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.security.ProtectionDomain$JavaSecurityAccessImpl.doIntersectionPrivilege(Unknown Source)
```

Figur 8 Viser at der ikke kan indtastes noget, som ikke er en int

Hvis inputtet til antal spillere ikke er en integer bliver inputtet med det samme slettet og der dukker en fejlmeddelelse op som vist på screenshottet.

Interessant kan det være at notere at den samme meddelelse kommer op hvis man prøver at slette en integer med backspace (både gyldigt mellem 2-6, eller ugyldigt). Dette er ikke ligefrem intuitivt, da det er meget normalt at slette antal spillere hvis man fx trykker forkert eller hvis antallet af spillere der vil være med ændrer sig.

Hvis man først indtaster en integer og derefter indtaster andre non-integer variable, bliver dem der ikke er integers slettet på samme måde som før, men der bliver ikke vist en fejlmeddelelse. Yderligere, hvis man indtaster et gyldigt antal spillere og derefter indtaster en non-integer variabel, bliver denne slettet, men man kan derefter stadig trykke ok og fortsætte spillet uden nogle fejlrapporter i konsollen. Intet af dette påvirker brugeren, da fejlmeddelelserne ikke kan ses andre steder end i konsollen.

Herudover laves en masse JUnit tests inde i selve programmet, som kan køres hvis det haves på computeren.

Arv

Arv er en af de funktioner som befinder sig inden for objektorienteret programmering. Arv gør det muligt for en klasse at hente objekter og metoder fra en anden klasse. Det er den afledte klasse som arver sin opførsel fra en hovedklasse. Den afledte klasse hedder en subklasse og hovedklassen kalder man en superklasse. Subklassen kan tilføje sine egne metoder og variabler, som differentiere den fra superklassen. Superklassen kan have flere subklasser, men subklassen har kun én superklasse. Subklassen har altså en 'er en/et' forhold til superklassen (uddyber i eksempel).

For et eksempel på arv kan vi forestille os en superklasse som hedder 'Transportmiddel'. Herunder findes der en lang række subklasser som bl.a. Cykel, Bil, Båd, Fly, osv. Her kan man sige at subklasse er en superklasse, altså 'Cykel er et Transportmiddel'. Hver subklasse har altså en række metoder og objekter som er ens for dem alle, de findes alle i superklassen 'Transportmiddel'. Hver subklasse fx. 'fly' har sin egen unikke variabler og metoder som fx. 'jetmotor'. Dette kaldes for Arv.

Abstract

Klasser:

En abstrakt klasse i java kan indeholde abstrakte metoder. Klasser der er abstrakte kan initieres, hvis man vil bruge metoder indeholdt i en abstrakt klasse, skal man nedarve den til en anden klasse hvor man kan give metoderne krop.

metoder:

Abstrakte metoder kan kun dannes i en abstrakt klasse, disse må ikke dannes med krop men kun med et semikolon til sidst, men de kan indeholde parameterlister ligesom andre metoder.

```
public abstract String getString(int StringNumber);
```

Ved nedarvning af den abstrakte klasse kan der metoderne "overskrives" og dannes med krop.

LandOnField metoden

Når man kalder landOnField metoden, bliver der testet hvilken type af felt det er. Alt efter hvilken type man er landet på bliver price og rent opdateret i den enkelte klasse, så når denne senere bliver kaldt er det ikke en forkert price eller rent der bliver kaldt, og det er heller ikke fra en anden type af felt.

GRASP

I det følgende bliver det dokumenteret hvordan grasp principperne bruges i forhold til det udførte projekt.

Creator

Vi gør brug af Creator princippet, da vi kun opretter vores objekter én gang med undtagelse af GameBoard objektet.

Player() og Accounts() bliver oprettet i Main klassen, som det eneste sted, hvor Main sender info videre til andre klasser. GameBoard objektet bliver simpelthen oprettet flere steder, da vi mente det gav mere mening i vores tilfælde.

Information Expert

Vi tager i vores kode, udgangspunkt i Information Expert princippet, da vi opsplitter ansvaret ud mellem flere klasser og holder det objektorienteret. Eksempelvis holder Klassen "Die" styr på alle metoder der beskæftiger sig med at bruge terningekast.

Low Coupling

Vi fik problemer med at overholde Low Coupling principperne, idet vi sender nogle info fra Main -> PlayerTurnSwitch -> FieldController -> (Field klasser), og tilbage igen.

I Low Coupling skulle man helst undgå, at information bliver delt for mange gange mellem klasser. Men i forholdet mellem Account-, Player- og Mainklasserne overholder vi disse principper noget bedre.

High Cohesion

Alle vores klasser har et specifikt ansvar. Account klassen holder styr på spilleres balancer. Player klassen holder styr på spillere og Territory klassen har ansvaret for, hvordan Territory felterne fungerer. Ét sted vi har en klasse som har flere ansvar. PlayerTurnSwitcher har ansvar for både at holder styr på hvilken spillers tur det er, samt at dele information mellem Main og FieldController.

Controller

Igen overholder vi principperne med undtagelse af nogle få tilfælde. Vi har Main, som fungerer som controller. Den styrer terninge-kastene og henter information fra passende klasser og integrere dem i GUI'en.

I vores undtagelse har vi Tax-klassen, som kalder en metode til GUI, hvor den beder om et userinput, hvilket ikke er en controller, men som stadig kommunikerer med GUI.

Konklusion

Efter længere tids arbejde er programmet nu udviklet. Dets funktionalitet er blevet gennemtestet igennem automatiske JUnit tests i programmet og manuelle tests, som er blevet dokumenteret på tekst- og billedform. Der er lavet et design-klasse-diagram for hele programmet og et system-sekvens-diagram for LandOnFleet.

Samtidigt er hele projekt igennem forløbet blevet testet og versioneret igennem GitHub.

Forløbet har dog ikke foregået uden problemer. Da der i CDIO-2 projektet var blevet lavet en egen GUI, var der en del ekstra arbejde, da det tidligere projekt nu skulle implementeres, så den passede med den nye GUI. Her opstod en del problemer, som blev ved med at sætte projektet tilbage. Dette førte i sidste ende til at gruppen kom i tidsproblemer og gjorde at der blev taget diverse kompromiser for at få projektet færdig til deadline.

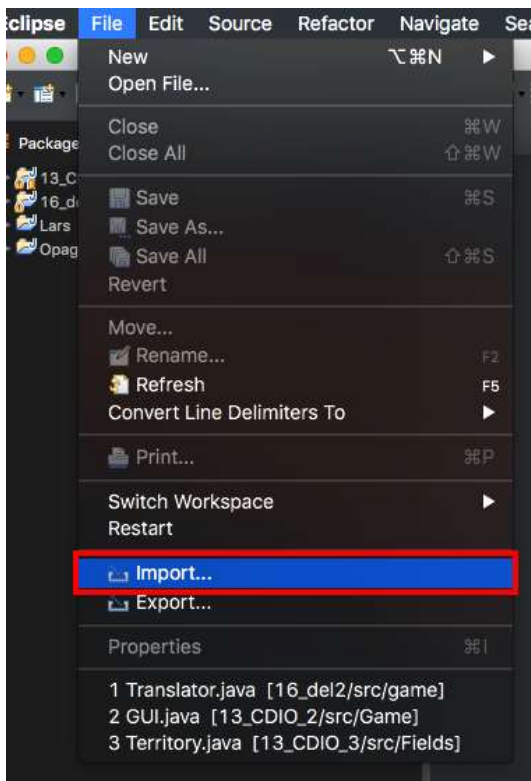
Bilag

Bilag 1

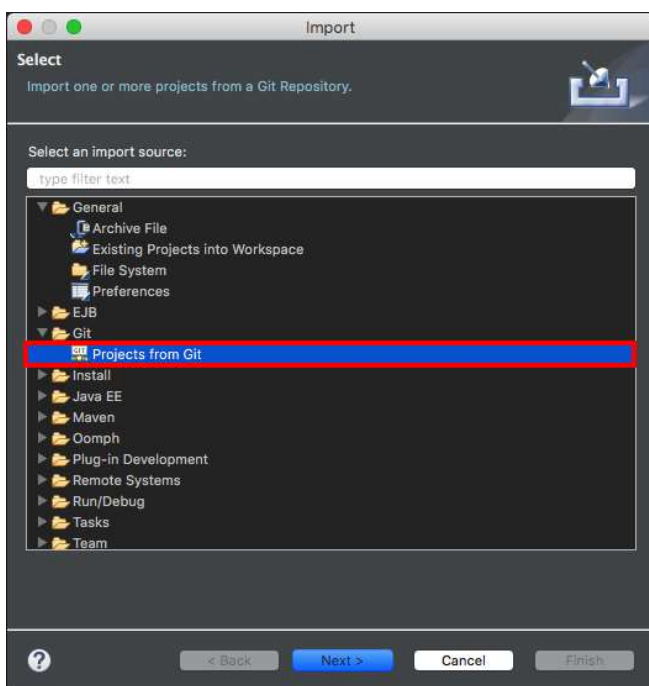
Vejledning i at importere gitrepo'et i eclipse.

Vejledning: ImportereGit-repository til Eclipse

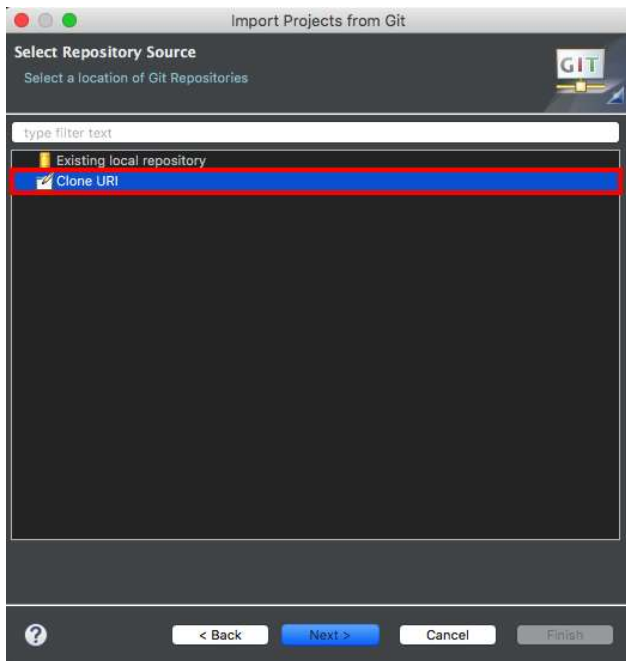
1) Gå til File > Import:



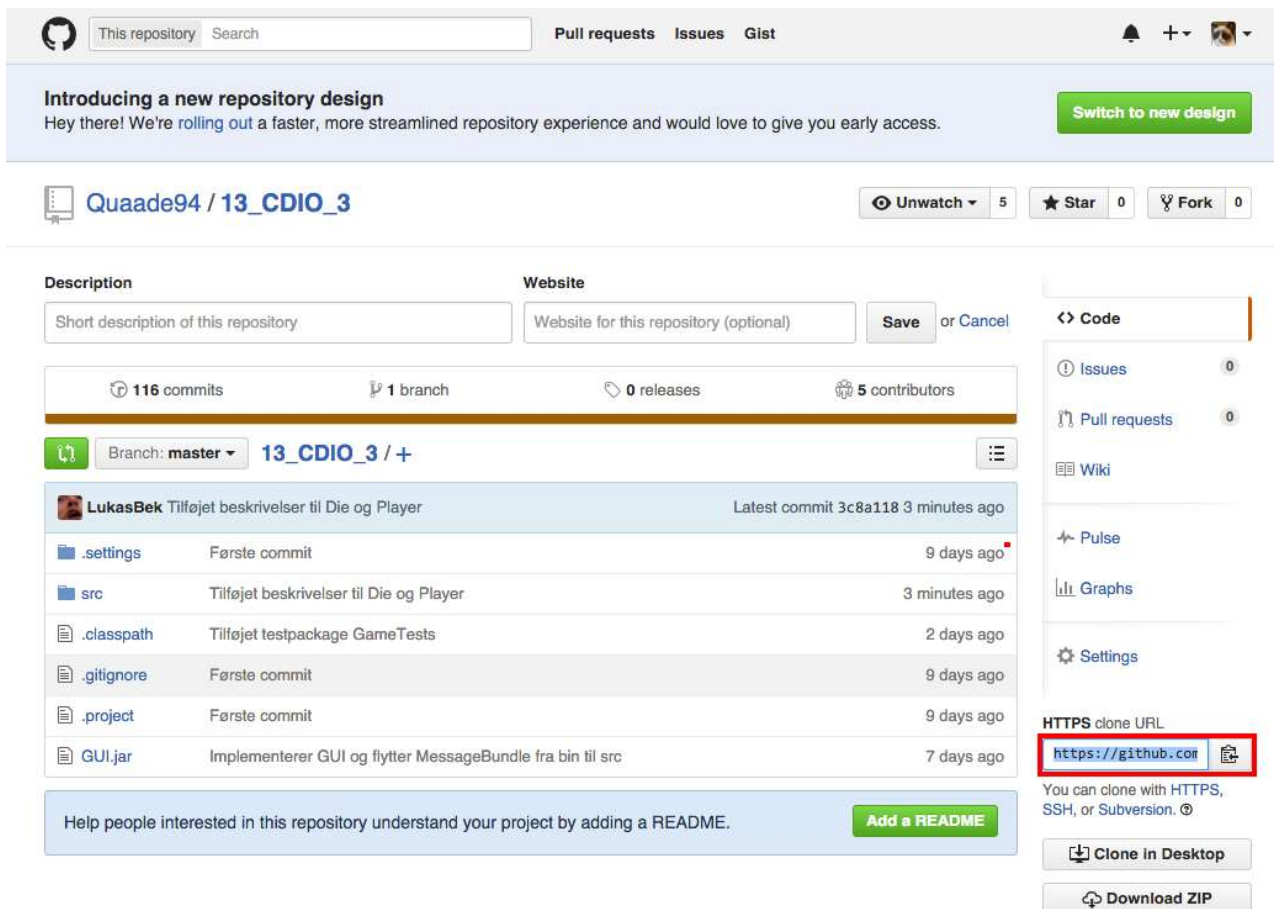
2) Tryk på 'Projects from Git', og derefter 'Next':



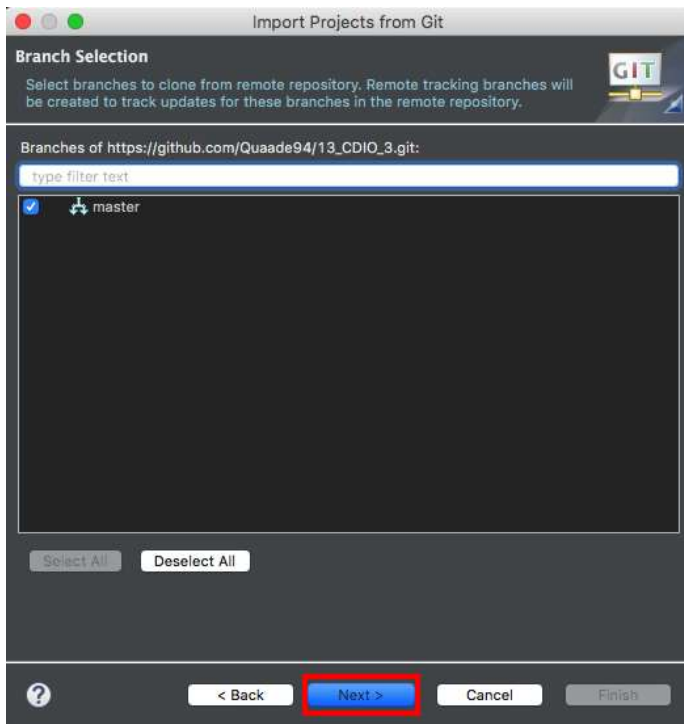
3) Trykpå 'Clone URI' og derefter 'Next':



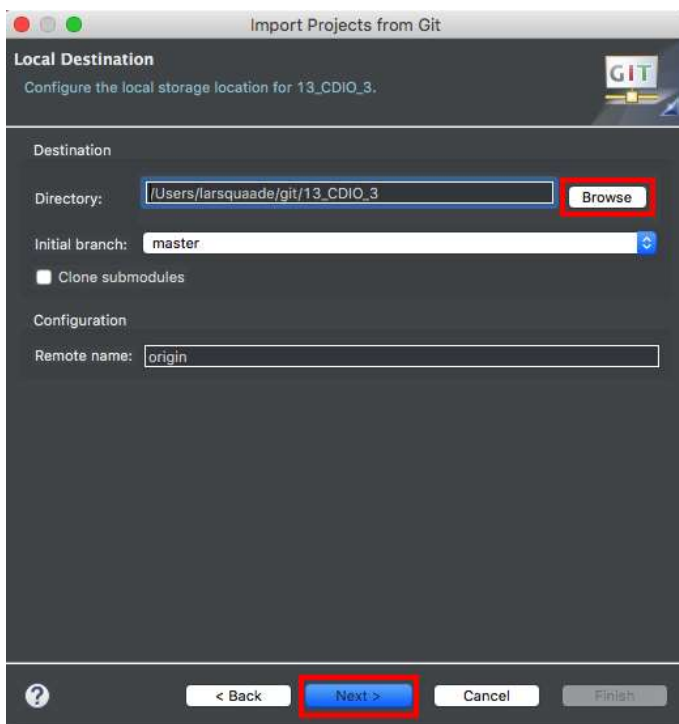
4) Under det ønskede repository på GitHub, find HTTPS clone URL og kopiere den til clipboard.



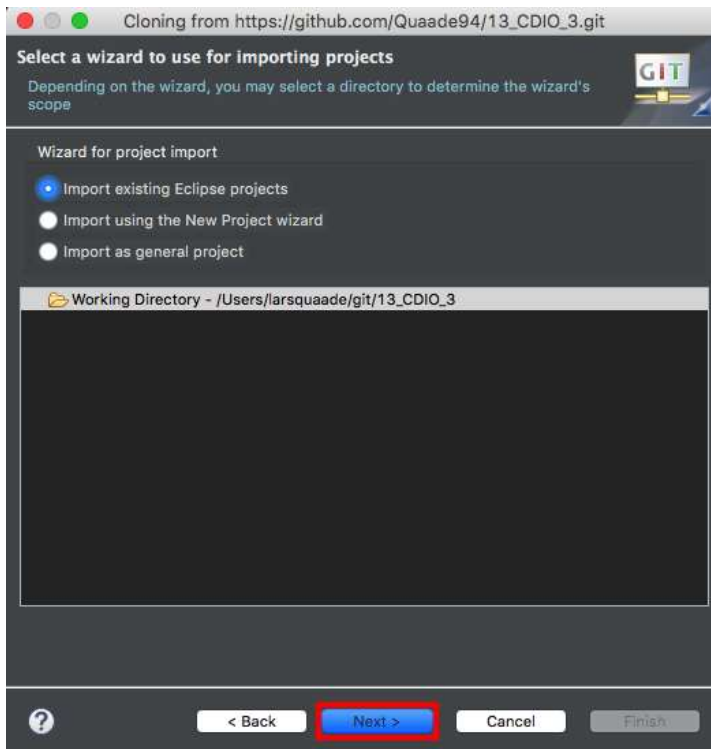
5) Tryk 'Næste':



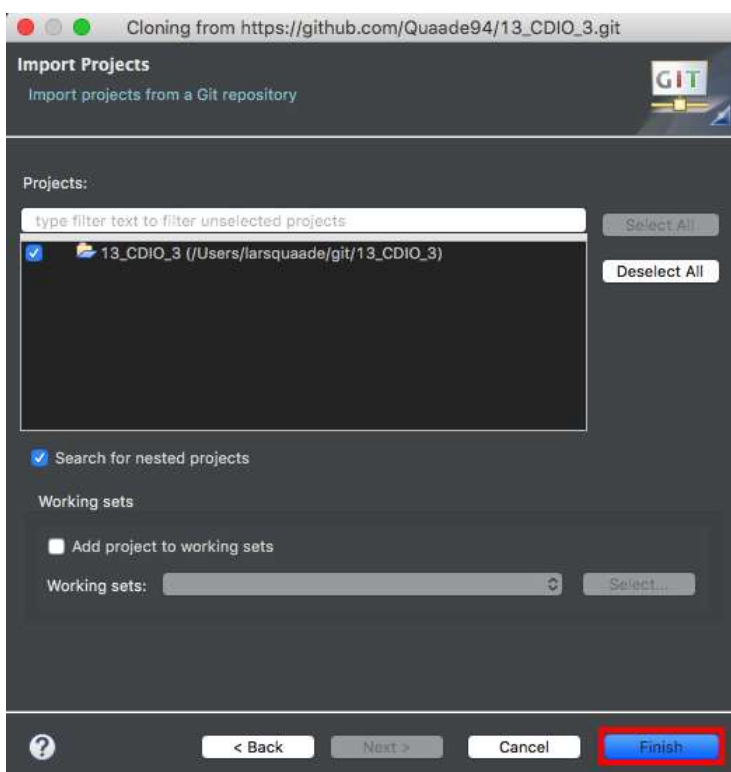
6) Vælgetsikkertsted på din computer, hvor du vil gemme dit lokale GIT-Repository og derefter tast 'Næste'.



7) Tryk 'Næste':



8) Tryk 'Finish':



9) DitGit-Repository er nu Importeret til din Package Explorer i Eclipse:

