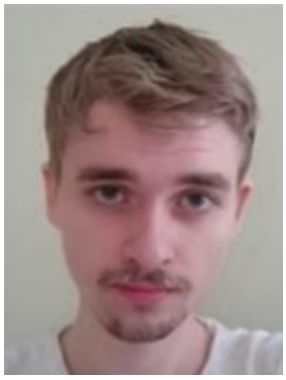
CDIO-Final

Matador

Lukas Bek (s153475)

Lars Sørensen (s144859)

Sebastian Hoppe (s154306)



William Wiberg (s153368)

Magnus Haakonsson (s153947)

Gruppe: 13

Afleveres: 18-01-2016

Opgavetype: Rapport,
Softwareteknologi

Fag: 02312
Rapporten består af 47 antal sider, incl bilag
Rapporten er afleveret via Campusnet

Timeregnskab

Navn	Implementereing	Design	Test	Dokumentation	Andet	Ialt
Lars Sørensen	43	8	2	4	9	66
Lukas Bek	43	3	10	8	6	70
Magnus Haakonson	43	10	2	9	6	70
Sebastian Hoppe	40	5	10	5	4	64
William Wiberg	25	12	3	4	6	50

Indhold

Indledning

Vi er en gruppe af studerende, der har fået til opgave at lave et matador spil. Opgaven kommer fra et fra et projektforløb, som tager udgangspunkt i nogle CDIO opgaver, som der gang på gang bliver bygget videre på. Dette er det sidste i opgaveforløbet og opgavebeskrivelsen lyder sig på intet andet end “lav et matadorspil”.

Der skal udarbejdes diverse artefakter for at få en ide om hvordan og hvorfor programmet skal udformes, som det nu skal. Her bruges bl.a. use cases og tilhørende usecase diagrammer, en BCE model for at få overskuelighed over controllerne og til slut et design-klasse-diagram for at få et overblik over alle klasserne. Hele programmet udføres med GRASP princippet i baghovedet, for at have en bedre sammenhængende og mere overskuelig kode.

Projektet versioneres ved hjælp af GitHub og rapporten ved hjælp af googledocs.

Analyse

CDIO projekterne er lavet tidligere og der kan derfor bruges en del erfaring fra de projekter. Det er dog ikke andet end terningen, som direkte kan trækkes over, da de sidste projekter desværre har været bygget dårligt. Derfor forsøges der at få bedre styr på arkitekturen af programmet i dette projekt. Der er således blevet lavet passende use cases, samt et udkast på hvordan vi forventer et klassediagram ville se ud.

Kravspecifisering

Der er ikke blevet stillet nogle krav til dette projekt, andet end "Lav et matadorspil". Derfor er der nogenlunde taget udgangspunkt i nogle af de tidligere projekters
krav. Resten af kravene, som egentlig er hvad der bestemmer hvordan spillet skal udformes er op til gruppen selv. I kapitlet "Spillets logik" vil der blive beskrevet hvad spillet kan.

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
6. Spil mellem 2-6 personer
7. Starter med 30.000
8. Vinder er sidste mand som ikke er bankerot (Spillet slutter når der findes en vinder)
9. Spilleren er ude af spillet når deres balance når under 0
10. Der skal være en spilleplade
11. Man går i ring på brættet

Non-funktionelle krav:

12. Projektets udførelse skal kunne ses i et GIT repository

Krav til dokumentation:

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

Use cases

Use cases

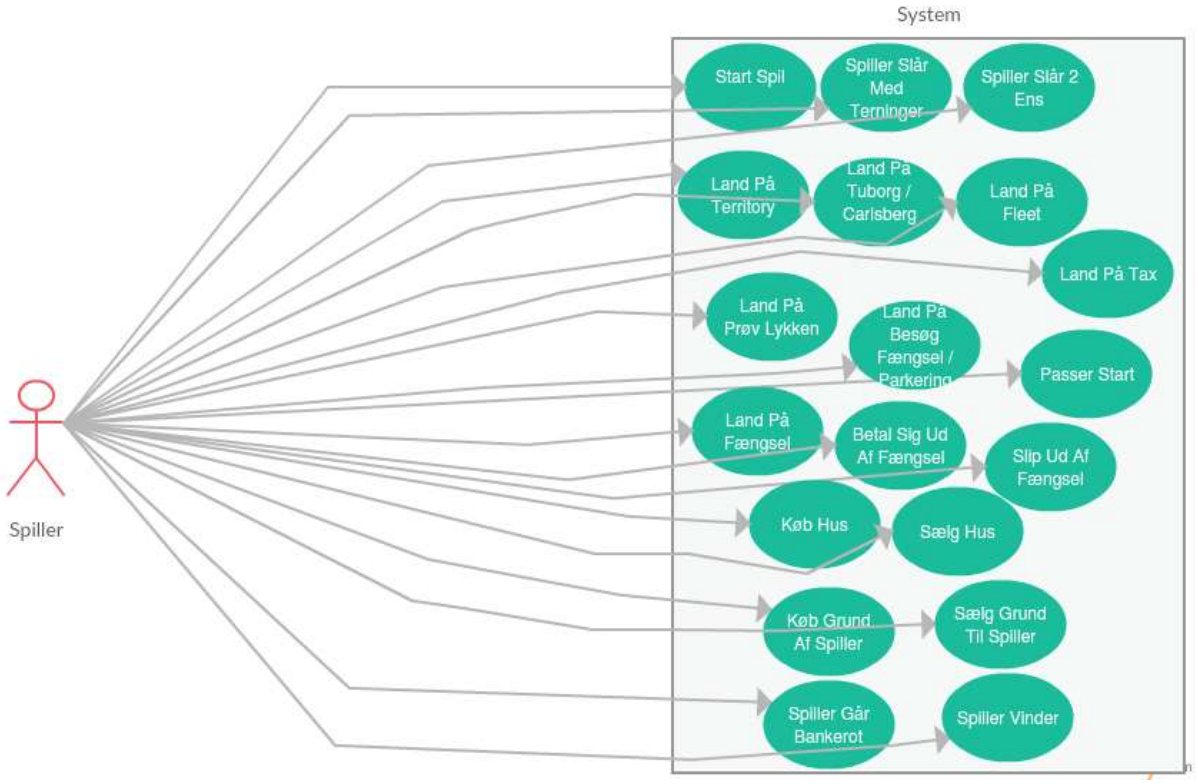
Alle uses cases er lagt under bilag 4:

1. Start Spil
2. Spiller Slår Med Terninger
3. Spiller Slår 2 Ens
4. Land På Territory
5. Land På Tuborg / Carlsberg
6. Land På Fleet
7. Land På Tax
8. Land På Prøv Lykken
9. Land På Besøg Fængsel / Parkering
10. Passer Start
11. Land På Fængsel
12. Betal Sig Ud Af Fængsel
13. Slip Ud Af Fængsel
14. Køb Hus
15. Sælg Hus
16. Køb Grund Af Spiller
17. Sælg Grund Til Spiller
18. Spiller Går Bankerot
19. Spiller Vinder

Her demonstreres en, der viser hvad der sker når en spiller har fået sin tur og vælger at betale for at komme væk fra fængsel feltet:

Betal Sig Ud Af Fængsel
ID: 12
Kort beskrivelse: Spiller betaler penge for at komme ud af fængsel
Primær Aktør: Spiller
Sekundære Aktører:
Forudsættelser: Det er spillerens tur, spiller sidder i fængsel og har status: fængslet
Primært Flow: 1. Spiller trykker på knappen “Pay” 2. Spiller får trukket 1000kr fra sin konto. 3. Spiller er ikke længere fængslet
Efterfølge: Turen skifter til den anden spiller
Alternative Flows:

Use case diagram

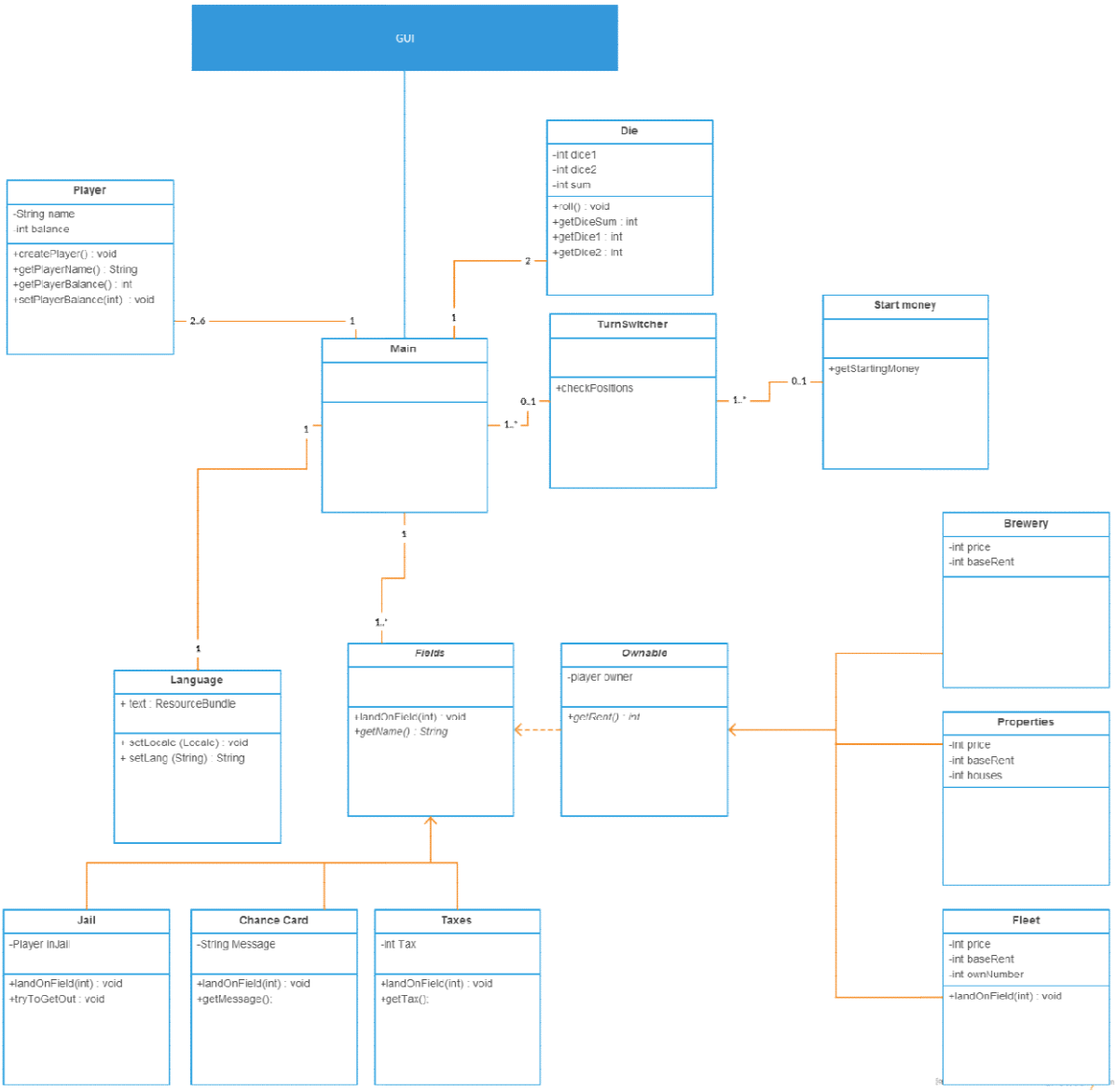


Figur 1 Use case diagram

På diagrammet ses spilleren som aktør og kassen repræsenterer systemet som indkapsler de forskellige use cases.

Klassediagram

Dette er som nævnt kun et udkast, så det vil ændre sig under programmets skabelse, men herunder ses det første klassediagram:



Figur 2 Klassediagram

Som det kan ses er det mangelfuldt, men de mangler vil blive fyldt, som programmet bliver skabt og herefter lægges i et design klasse diagram, som viser alle klasserne og deres forhold.

Spillets logik

Til dette projekt var den eneste beskrivelse og det eneste krav: “Lav et matadorspil”. Derfor ligger de krav der stilles til spillet, samt hele den logik spillet vil have op til os. Derfor vil der i det følgende afsnit forsøges beskrevet hvordan programmet opfører sig, samt de features det indeholder.

1. Spillets start

I starten af spillet skal brugeren indtaste hvor mange spillere der sammenlagt vil spille spillet, som holder sig inden for spillets grænser: 2-6 spillere. Herefter bliver alle spillere bedt om at indtaste deres navn, og som de gør det, vil de se deres navn og bil på GUI’en. Efter at alle har oprettet sig, bliver det spiller 1’s tur.

2. Spillerens tur

2a) normal tur

Der er to forskellige ture en spiller kan have - en fængsels tur og en “normal”-tur. Når det er den normale tur, vil spilleren have tre muligheder: 1. er at at rulle med terningerne, hvilket vil rykke hans bil den mængde felter, som han slår. Slår han to ens, får han lov til at slå igen, efter den effekt det felt han landede på er taget i betragtning. Slår han dog to ens tre gange i træk vil spilleren blive fængslet og hans tur slutte.

2)b fængsels tur

Er spilleren i fængsel har han to muligheder. Personen kan enten betale 1000 for at komme ud, eller vælge at prøve at slå to ens i tre slag. Vælger han den første mulighed vil han betale de 1000, hvorefter turen vil slutte. Næste gang det så bliver hans tur, vil han være ude af fængslet og derfor have en normal tur. Vælges der i stedet at rulle med terninger, vil personen prøve at se om han kan slå to ens øjne hen over tre slag, og hvis personen fejler vil han hans tur slutte og han vil stadigvæk være i fængsel.

3. Felter

3a) De felter der kan ejes

Der findes en del forskellige felter, der alle gør forskellige ting. De mest normale felter er dem, som på spillepladen er farvet. De kan købes af spillerne, hvis de ikke allerede er ejet. Er de ejet, skal alle spillere, som ikke er ejeren betale penge til ejeren, når de lander på feltet. Hvis en spiller ejer alle felter inden for den samme farve kan de bygge op til fire huse, og herefter ét hotel, hvilket hver især vil forhøje den mængde penge, som de andre spillere skal betale for at lande på dem.
Udover disse kan spillerne også købe færge-felterne og bryggerierne. Disse kan der ikke bygges huse på, men hvis en spiller ejer flere af færge-felterne stiger prisen andre spiller skal betale på alle færge-felter, som den spiller ejer. Ligeledes med bryggerierne, der skal spilleren betale 100 gange det tal de slår med terninger, hvis en anden spiller ejer det. Dette stiger til 200, hvis spilleren der ejer dem, har begge bryggerierne.

3b) De resterne felter

Der findes også andre felter end de tidligere nævnte. Prøv lykken giver spilleren et prøv lykken kort, som har en tilfældig effekt, som f.eks. modtag 1000, ryk i fængsel eller betal 3000.
Tax felter betyder at spilleren der lander på det skal betale en forudbestemt sum.
Fængslet har et fængsel felt, som ikke gør noget når spilleren lander på det, men hvis en spiller er i fængsel, vil de sidde fast på det felt. Det andet fængsel felt "go to prison" betyder, som det lyder af sit navn, at spiller ryger i fængsel.
Herefter er der til slut start feltet, som er her spilleren starter spillet, og hver gang en spiller passerer det, modtager de 4000. Parkerings feltet gør ikke noget når spilleren lander på det.

4. Hvordan man vinder

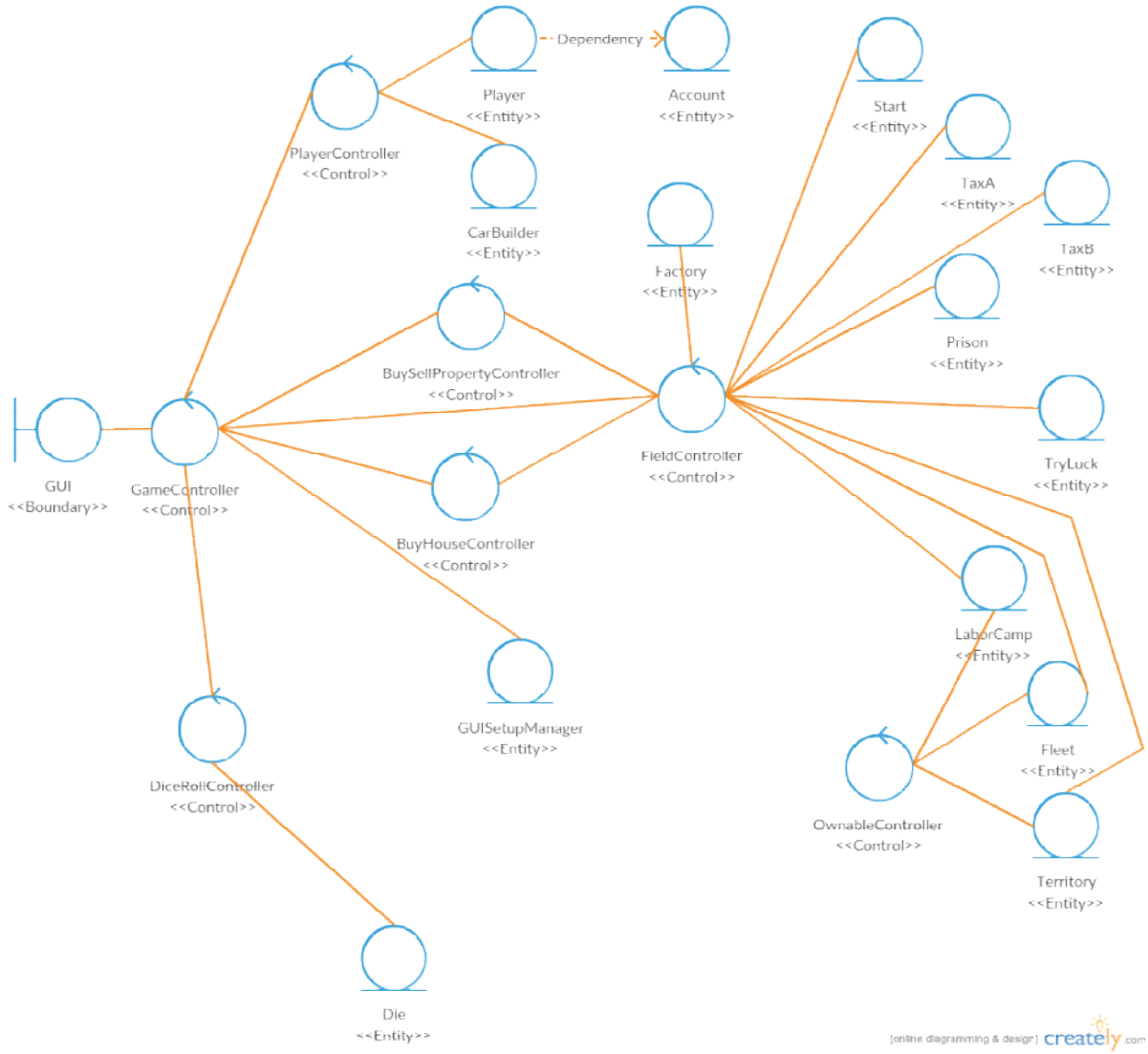
Alle spillerne starter ud med 30000 i deres pengebeholdning. Hvis spillerens pengebeholdning falder under 0, vil spilleren gå bankerot og de vil herefter være ude af spillet og mister alt de ejer. Den sidste spiller, som ikke er gået bankerot, vil være vinderen af spillet.

5. Køb og salg

Både grunde og huse kan købes og ligeledes sælges. En grund kan sælges og købes af alle spillere, hvis begge spillere går med til handlen. Grunden kan dog ikke købes og sælges hvis der er huse på. Huse kan sælges af ejeren af grunden for halvdelen af den pris de købte det for.

Design

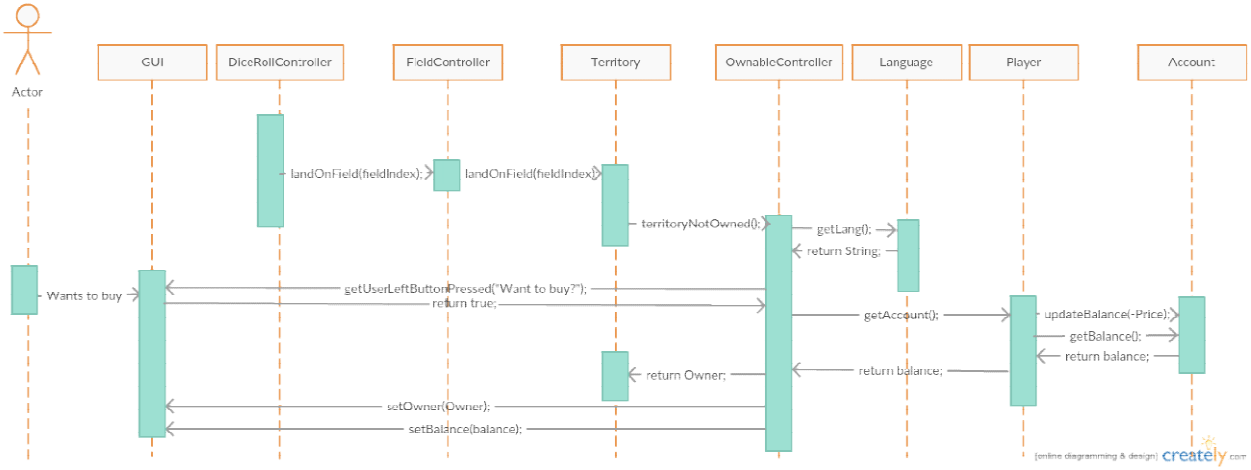
BCE-Model



Figur 3 BCE model

Denne BCE-Model viser forholdene mellem vores Boundary(GUI) og vores Controllers og deres forhold til vores Entities. Den viser blandt andet hvordan GameControlleren er en hoved kontroller, som styrer alle de andre kontrollere.

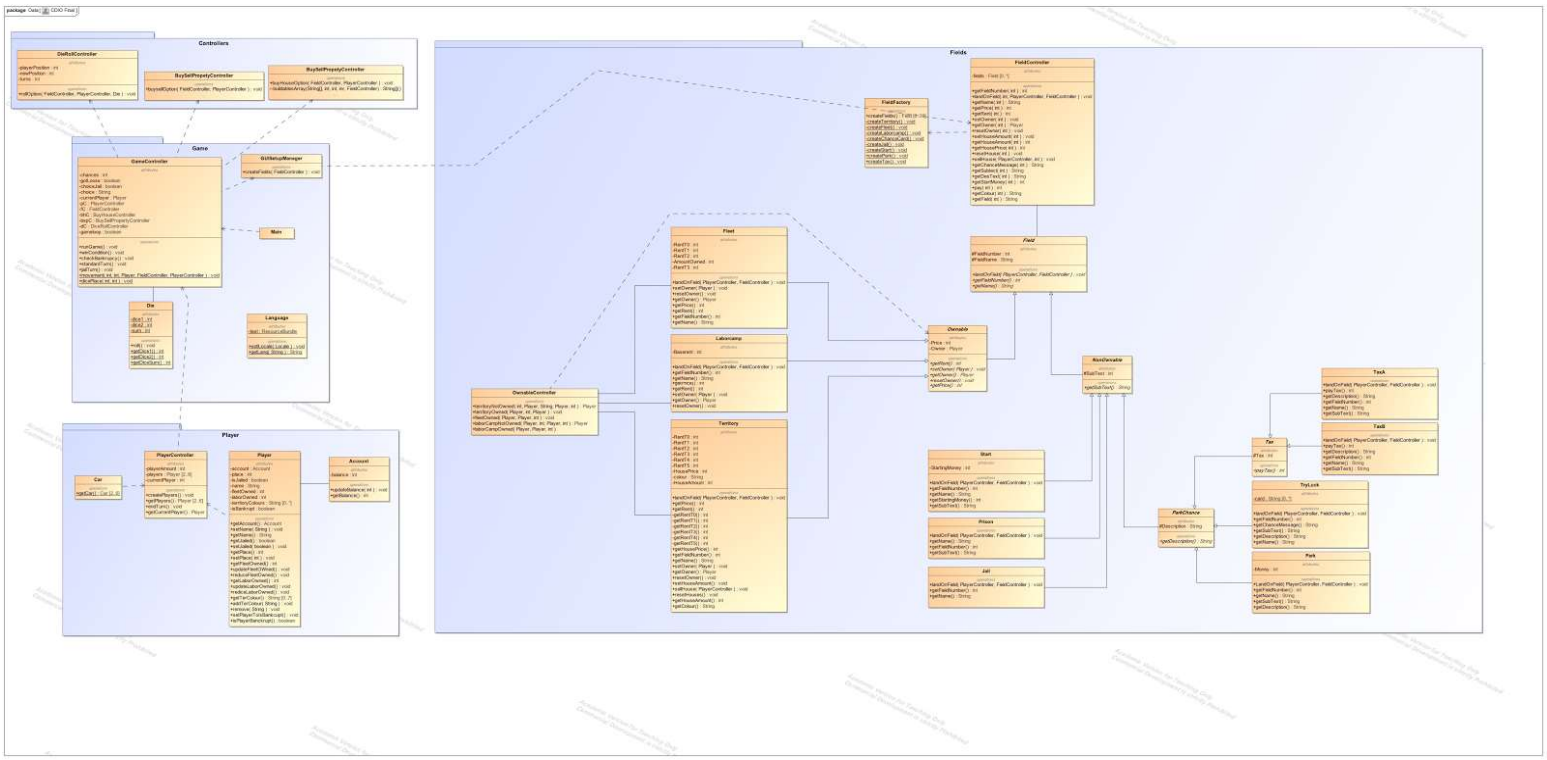
Design Sekvens Diagram



Figur 4 Design Sekvens Diagram

Dette diagram viser en instans af et flow, hvor en spiller er landet på et felt som han gerne vil købe. Aktøren er spilleren selv, som bliver spurgt om, hvorvidt han vil købe et felt eller ej, hvilket han svarer ja til. I dette flow, går vi ud fra, at spilleren har penge til at købe feltet og at det ikke allerede er ejet af en spiller.

Design klasse diagram



Figur 5 Design Klasse diagram

Design Klassediagrammet ovenfor viser hvordan at samtlige klasser i Matador spillet i vores system snakker sammen. (Øverst til venstre: Controllers : DieRollController, BuySellPropetyController, BuySellHouseController ; Midt til venstre : Game : GameController, GUISetupManeger, Main, Die, Language ; Nederst til venstre : Player: Car, PlayerController, Player, Account ; Til højre : Fields : Beskrives lidt mere i næste afsnit .

Tættere på Design Klassediagrammet (felterne)

I Matador er der forskellige felter, og alle disse skal opføre sig forskelligt: nogle giver penge andre tager, nogle kan man købe, og andre kan man modificere til at koste mere når nogen står på dem. Derfor har vi været nødt til at lave forskellige klasser der hver især nedarver forskellige karakteristika så de kan opføre sig på forskellige måder.

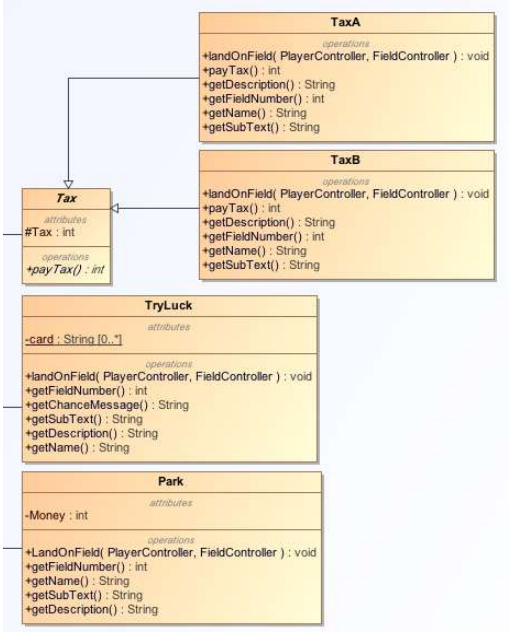
Til højre ses de felter der skal indeholde en Subtext, altså en ekstra information om feltet i form af en String som skal bruges for at lave feltet færdigt på vores GUI, denne Subtext findes i alle de felter der ikke kan ejes, dvs. ikke på gader, rederier og bryggerier. Men start, fængsel, parkering og andre skal bruge denne information.

Ydermere er der også felter der skal bruge flere tekstuelle informationer, denne gang en såkaldt Description. Disse klasser ses

Figur 6 Udsnit af DSD

til højre. På samme måde som Subtexten har disse så denne ekstra information som ikke andre klasser har.

Dog har alle felterne noget til fælles. Den information som alle felterne har til fælles ligger oppe i den øverste klasse, som i dette tilfælde er "Field" klassen. Denne abstrakte klasse skal have den information som alle de andre klasser som nedarver fra den også skal have, dvs. deres nummer på spillepladen og deres navn, og de metoder der følger med, sammen med en "landOnField" metode der sørger for at det rette sker, lige meget hvilket felt man er landet på, om det så er Rådhuspladsen eller parkeringen.

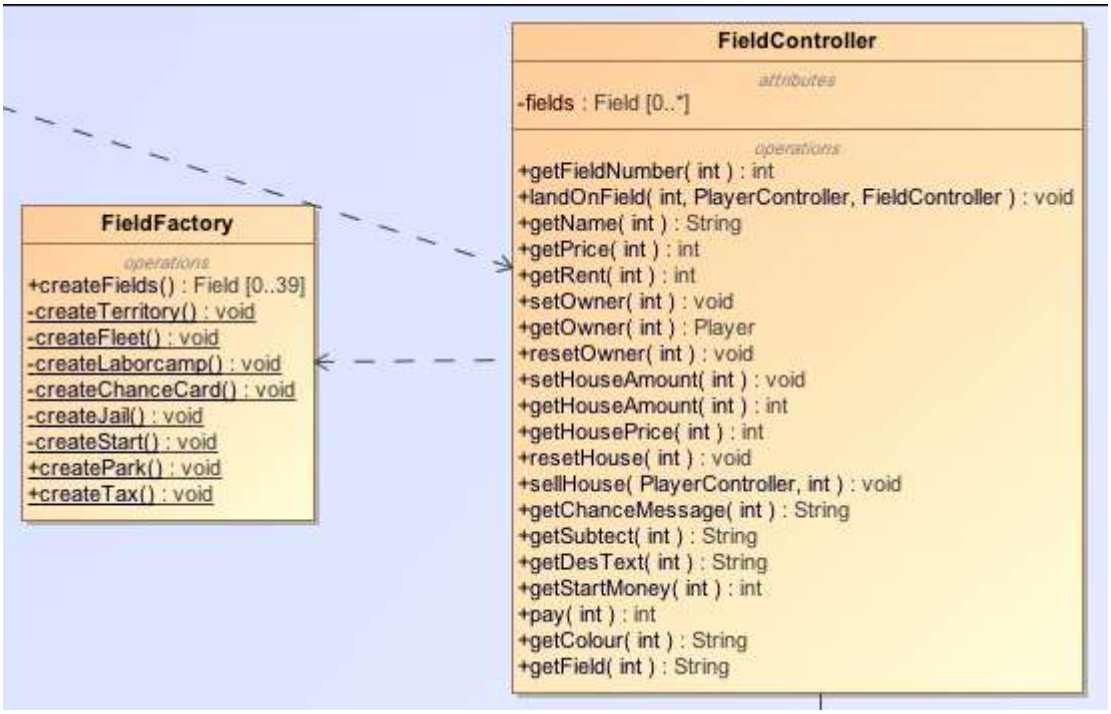


Figur 7 Udsnit af DSD

De felter der kan ejes er også specielle og skal have nogle af deres egne informationer, men der er hver især forskellige, men igen har de ting der er ens, disse fås fra den abstrakte klasse "Ownable". Denne klasse giver variabler og metoder, der er fælles for alle de felter der kan ejes, såsom ejeren og prisen for at købe feltet, dernæst kommer metoder til at hente denne information og lejen for at lande på felter der allerede er ejet. (Disse klasser kan ses på venstre side). Men de felter der kan ejes er også hver især forskellige. Eksempelvis kan gaderne have huse, hvor de andre felter ikke kan. Så denne klasse skal specifikt have variabler og metoder til at kunne håndtere dette, hvor bryggerierne kun har en grundleje, der ændres alt efter hvad der blev slået med terningerne. samt en fordobling hvis ejeren af feltet også ejer det andet bryggeri. På samme måde har rederierne noget unikt, navnligt at man skal betale mere leje jo flere man ejer. På denne måde kan de arve nogle ting fra en overklasse, men da de alle er lidt forskellige og skal noget lidt andet, kan de slås sammen i en klasse uden at nogle af metoderne og variablerne bliver overflødige.

Figur 8 Udsnit af DSD

Det hele bliver bundet sammen af klassen "FieldController", der fungere som mellemmand mellem felt objekterne og resten af systemet. Denne klasse indeholder metoder til at kunne hente eller ændre i alle de relevante variabler i alle de forskellige FieldKlasser. Dette kan lade sig gøre da den modtager alle Field Objekterne fra klassen FieldFactory, hvis eneste formål er at skabe de 40 objekter til alle de forskellige felter, og derefter returnere dem til "FieldController" klassen, der derefter har dem i et Array, og det er derfra at alle "Field" klassernes metoder bliver kaldt fra.
"FieldController" og "FieldFactory" kan ses på næste side.



Figur 9 Udsnit af DSD

Alle ovenstående udklip af DKD findes i en forstørret version i bilag 1.

Implementering

Oprettelse af spiller

Når aktøren har valgt hvor mange spillere der skal spille, begynder vores PlayerController at oprette et spiller array, alt efter hvor mange spillere aktøren har valgt. Derefter begynder den at lave hvert enkelt spillerobjekt, hvor den også opretter en GUI-bil til alle spillere. Den kører et og loop spørger om et navn fra de forskellige spillere, som den så tjekker. Hvis navnet eksisterer bliver booleanen *nameExists* sat til true, og spillet beder spilleren om at indtaste et nyt navn. Det samme sker, hvis spilleren har indtastet et navn der er mindre end 3 karakterer eller større end 17. Hvis spilleren gør alt rigtigt, bliver String *name* defineret som det valgte navn. Derefter bliver spillerobjektet skabt og der bliver kaldt en .setName() metode, som giver objektet det valgte navn. Samtidigt med at spillerobjektet bliver oprettet for spilleren, ligger det i konstruktøren at der også oprettes et tilhørende account objekt, til dette spillerobjekt. Denne account kaldes igennem spilleren med getAccount() metoden, som henter account objektet. Et kald til at se balancen for en spiller vil derfor se sådan ud: PlayerController.getPlayers()[x].getAccount().getBalace(), hvor x er den spillers balance man ønsker at få.
Loopet kører indtil alle spillere har fået oprettet et navn og en bil og er blevet placeret på GUI'en.

landOnField() metoden

landOnField() metoden bliver kaldt, når en spiller har slået med terningerne og lander på et felt. landOnField() bestemmer hvad der skal ske herefter. DiceRollController kalder landOnField() metoden umiddelbart efter at den har placeret spilleren på feltet, til FieldControlleren. landOnField() har en parameter der er et fieldindex, som FieldControlleren bruger til, at finde ud af hvilken slags field-type, som skal køre landOnField() metoden, hvorefter den kalder landOnField() metoden til passende type. Lad os sige at spilleren er landet på et Territory felt, som de hedder i systemet. Territorys landOnField() metode tjekker hvorvidt feltet spilleren er landet på er ejet eller ej, hvorefter den laver ét af to metodekald til OwnableControlleren. Hvis feltet ikke er ejet, bliver territoryNotOwned() kaldt, og omvendt hvis det er ejet bliver territoryOwned() kaldt.

Not owned:

OwnableControlleren spørger om spilleren har lyst til at købe feltet, gennem et GUIkald. Hvis spilleren siger ja, tjekker den om spilleren har råd til at købe feltet.
Hvis spilleren har råd, trækker den pengene og opdaterer ejeren af spilleren samtidig med, at den opdaterer hvor mange af den farve felter spilleren ejer via addTerColour() metoden.
Hvis spilleren ikke har råd, vil den komme op med en besked om, at spilleren ikke har penge, hvorefter processen slutter.

Owned:

OwnableControlleren tjekker om spilleren har penge til at betale den rente der kræves.
Hvis spilleren har råd til at betale, bliver pengene trukket fra spilleren og derefter tilføjet til ejeren af feltet, hvorefter GUI'en opdateres.
Hvis spilleren ikke har råd til at betale, bliver alle de penge spilleren har tilbage på kontoen

overført til ejeren af feltet og spilleren får sin konto sat til -1, som er en nødvendighed for, at systemet kan registrere at spilleren er gået bankerot.

Metoden fungerer stort set ens for Fleet og LaborCamp felterne også.

DiceRollController

Selve die klassen er simpel. Den generere to tilfældige tal mellem 1 og 6 ved brug af Math.random() metoden og så returnerer den begge tal og en sum. DiceRollControlleren står for at sætte spillerens nye position i playerController så systemet forstår at spilleren har en ny position og at noget skal køres i landOnField() metoden. Det er også i DiceRollControlleren at spilleren får lov til at trykke på 'Roll' på GUI'en som derefter viser to terninger inden for et givent felt på GUI'en med de øjne som bliver slået og skriver summen som en besked til brugeren. DiceRollController har den vigtige opgave at give en spiller mulighed for en ekstra tur hvis spilleren kaster to ens. Hvis spilleren kaster to ens for tredje gang, er det også her at spilleren får status som 'jailed' og bliver flyttet over på fængsels feltet. En if() sætning sørger for at hvis summen af terningerne overstiger 40, som er mængden af felter på brættet, så trækkes der 40 fra den nye position, så spilleren kører rundt om brættet til de første felter igen.

Language klassen

I vores program har vi skulle lave beskeder, som GUI'en skal vise til spilleren i diverse scenarier. For let at kunne ændre sproget på programmet, også for folk uden viden om java, eller programmering i det hele taget, har vi valgt at benytte en MessageBundle.

Én MessageBundle er en .properties fil, som har en "kode" , og en besked. Koden skal benyttes i programmet for at få beskeden vist. Skulle man have lyst til at ændre sproget, eller bare ændre i beskederne generelt, er alt man skal gøre at gå ind i MessageBundle_en_GB.properties og omskrive beskederne. Man skal ikke rode i koderne, da det er dem programmet bruger, for at vide hvilken besked den skal vise.
Vores Language klasse har en metode getLang(); som har en String parameter. Den kalder på getLocale, som henter MessageBundlen og bagefter finder den passende besked alt efter hvilken String den har fået. getLang()'s String skal altså være koden der passer til den besked man vil have i MessageBundle.

Vil man hente en bestemt besked i programmet, skal man først importerer klassen via:

import Game.Language;

Derefter kan man kalde metoden getLang() statisk. Her er ét eksempel:

```
Language.getLang("SRT");
```

Dette vil returnere beskeden:

Start Game

Denne besked står i MessageBundle således:

SRT = Start Game

Hvis man vil ændre det til dansk, ville man ændrer det til dette:

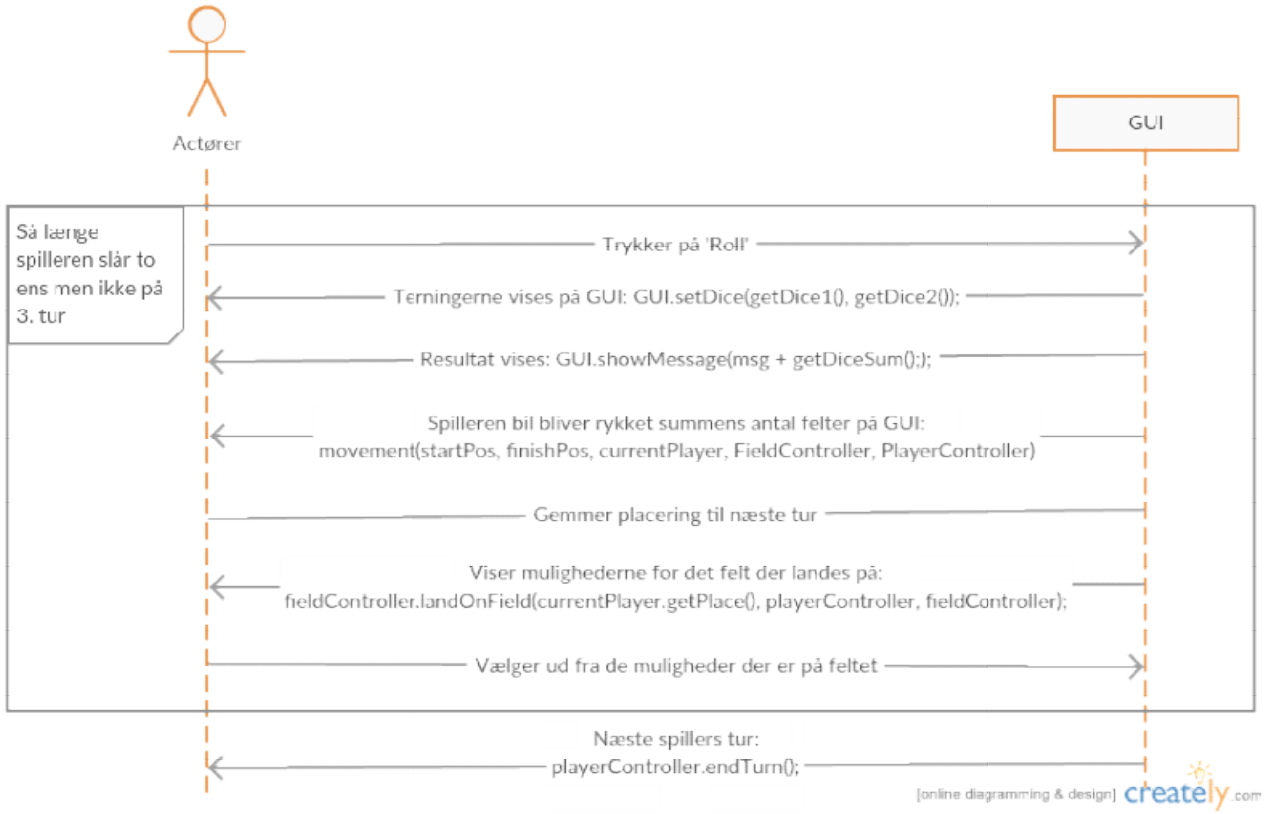
SRT = Start Spil'

Controllere

GameControlleren er vores hovedcontroller. Det er den der holder spillet kørende og den, der styrer de andre controllere. Controllere som ligger i Controller packagen, som DiceRollController, var oprindeligt en del af GameControlleren, men de blev deres egne Controllere, da GameControlleren blev for lang.
OwnableControlleren er den eneste, der ikke har et direkte forhold med GameControlleren.

GUI (Graphical User Interface)

System Sekvens Diagram



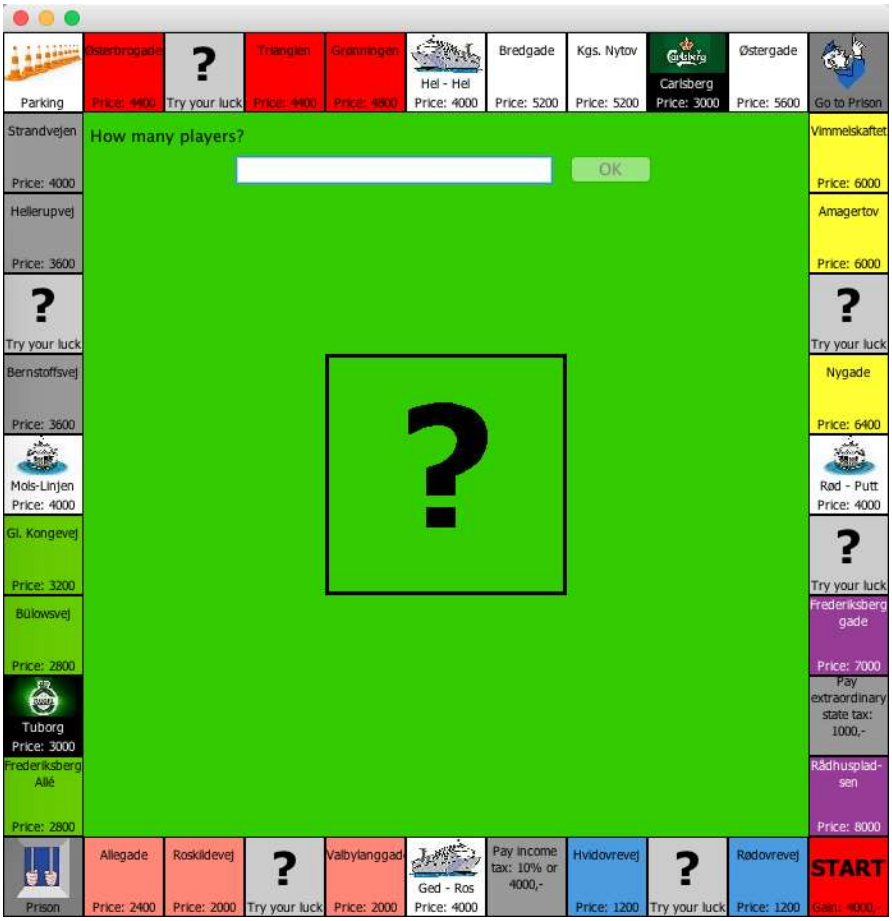
Figur 10 System Sekvens Diagram

Dette diagram er et visuel oversigt over et terningekast, baseret på Use Case diagrammet 'Spiller slår med terningerne'. Den viser metoderne brugt under et terningekast til at ændre i brugergrænsefladen. I dette tilfælde bliver der både vist summen af terningerne samt to billedfiler med øjnene på de to terninger. Da det er muligt at få en ekstra tur hvis man kaster to ens er der lagt et 'loop' over terningekastet som gentager forløbet inden for området indtil udsagnet bliver falsk.

GUI'en

GUI'en (Graphical User Interface) oprettes som det første når programmet startes, og har hensigt til at give aktøren mulighed for at interagere med systemet (se evt. SSD diagram).
Vi fik udleveret denne GUI i starten af vores forløb, som vi har haft mulighed for at bruge til vores matador spil som det løbende er blevet formet. Vi har valgt at bruge pladen, som den er oprindeligt, med alle 40 felter tegnet op, uden at bruge creator metoden, der giver mulighed for selv at ændre på hvert felt. Vi har dog lavet om på priserne og underteksten teksten, så hvert felt er bundet til dens tilhørende objekt i systemet. Det har den fordel at det er nemt at oversætte spillet til et andet sprog og ændre på priserne alle steder samtidigt.

Når en bil fik en ny position på spillepladen brugte vi GUI'ens metoder til at fjerne bilen fra dens nuværende felt og sætte den på dens nye (GUI.removeCar(); og GUI.setCar();). Vi opdagede at ved kun at bruge get og set metoden blev 'teleporteret' direkte til dens nye position, hvilket virkede forvirrende og uoverskueligt, da bilen havde det med at forsvinde for os. Vi fandt derfor på en løsning, hvor vi lavede en metode 'movement' som bruger spillerens startposition, slutposition og navn til at flytte brikken et felt ad gangen med et delay(Thread.sleep();) i et loop indtil han nåede frem til slutpositionen. Det fik spillerens bil til at køre over hvert felt for at nå frem, og derfor så det meget mere overskueligt ud hvem der havde tur, og hvem der lige var flyttet hvorhen. Denne metode løste også et problem som vi havde, nemlig startpenge. Metoden sørger nu for at man altid får startpenge når man passere start, men ikke hvis man er 'jailed'. Så hvis man bliver fængslet får man status som 'jailed' og så sørger en simpel if sætning for at man ikke får penge over start på vej i fængsel.



Figur 11 Skærmbillede af vores færdige GUI når programmet startes

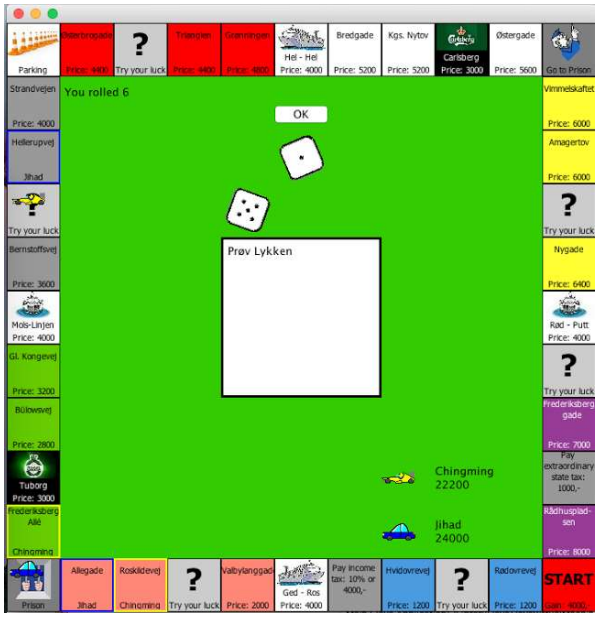
Udfordringer med GUI

GUI'en gav os dog nogle få udfordringer. Der er 3 metoder som vi kan bruge til at ændre det som stod på felterne og når man holder musen over et felt. Der fandtes ikke en setRent metode, som betyder at den rent som står på brættet når det det er ejet kan ikke ændres. Det ville ikke have været noget problem hvis ikke det var fordi der stod 'Ejet af:' på dansk da vores spil er på engelsk. Det samme gælder for 'prøv lykken' kortet, hvor det kun er muligt at kalde metoden, men ikke ændre i den String som den viser på dansk.

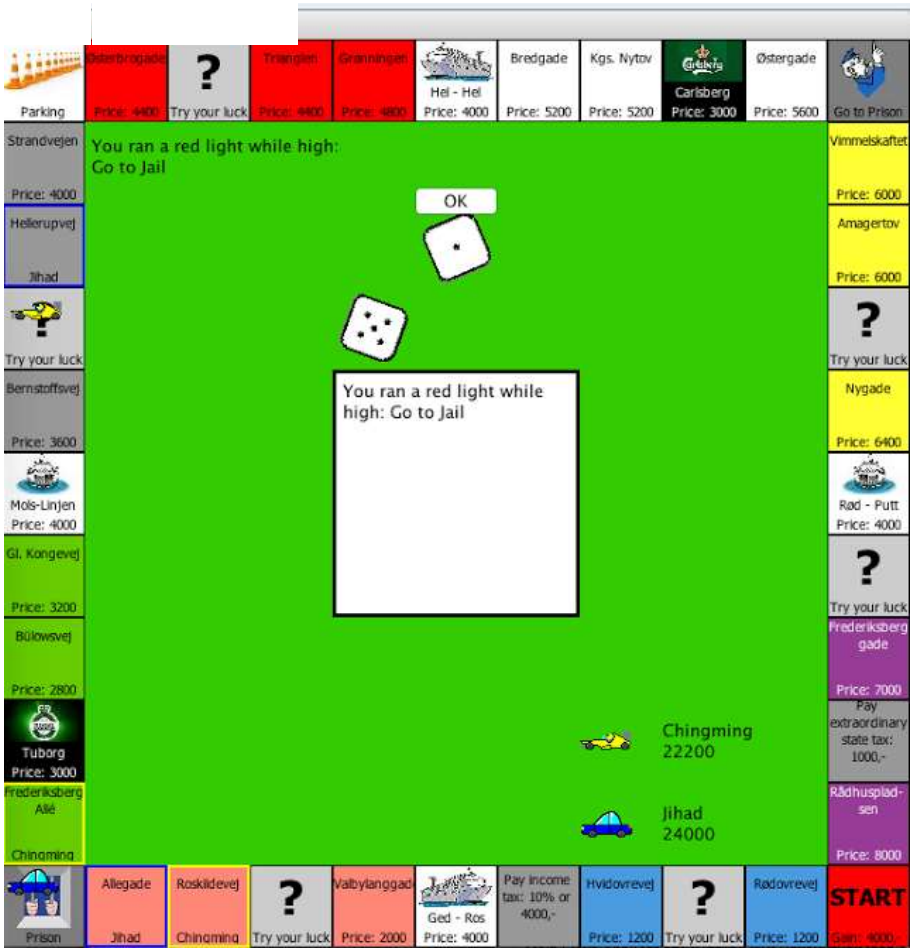
Figur 12 Eksempel på Ejes Af: og Leje: 75, som ikke kan ændres

Prøv lykken gav også andre udfordringer. Når den skal vise et kort, så skal man trykke på 'prøv lykken' i midten af GUI'en. Derefter sker det som der står på kortet og det bliver den næste persons tur med det samme men det er ikke altid at spilleren kan nå at se hvad der stod på kortet,

og det bliver meget forvirrende. Det kræves også at musen holdes over midten af pladen for at man kan se kortet. For at gøre dette mere brugervenligt, har vi valgt at bruge 'message' metoden så prøv lykken kortet kommer frem som en besked, og brugeren skal trykke 'ok' før det bliver den næste spillers tur.



Figur 11 Eksempel på hvordan det ser ud når en spiller lander på prøv lykken



Figur 13 Eksempel på hvordan det ser ud når spilleren får et prøv lykken kort

Tests

Der er til programmet lavet en del tests. Under hele udførelsen af programmet er der blevet testet manuelt om tingene virker, og til slut er der blevet oprettet JUnit tests. Disse JUnit tests er oprettet for at forsøge at teste om de forskellige dele af programmet virker som de skal.

Der er lavet JUnit tests af ChanceCards, Fields, Payers og House. som alle tester deres respektive ansvarsområde. Alle tests går igennem, men de har hjulpet os med at kvalitetssikre projektet, i den forstand at metoder og generelle funktioner i projektet virker efter hensigten.

I alle JUnit-testene køres der en @BeforeClass, som sætter hele miljøet op. Det ligger i @BeforeClass og ikke @Before, da disse er ens for alle klasser, hvilket betyder at de kun skal køres den ene gang, nemlig i starten, hvorimod de værdier i @Before skal køres før hver af testene. Den kode som ligger under @Before er lavet for at nulstille alle de data, som egentlig testes på under testene. Et eksempel er under playertests:

```
//Resets the players position to 0 (start) and resets their jail status and balance
for (int i = 0 ; i < pC.getPlayers().length ; i++){
    if (pC.getPlayers()[i].getPlace()!=0){
        pC.getPlayers()[i].setPlace(0);
    }
    if (pC.getPlayers()[i].getJailed()==true){
        pC.getPlayers()[i].setJail(false);
    }
    pC.getPlayers()[i].getAccount().resetBalance();
}
```

Figur 124 Eksempel på en @Before der køres før hver test

Her sker tre ting, som nulstiller spillerens funktioner, som bliver styret af PlayerControlleren; For det første sætter den alle spillernes plads til 0 (start), i tilfælde af at de skulle være blevet rykket i en tidligere test. Den sætter deres fængselsstatus til false, i tilfælde de skulle være blevet fængslet og til slut nulstiller den deres balance til udgangspunktet - i tilfælde af at den skulle være blevet ændret.

Konfigurations- og versionsstyring

Udviklings- og produktionsplatformen brugt til at udvikle Matadorspillet er i dette projekt ens og kan beskrives som en samlet platform. Den inkluderer følgende:

- OS: Windows 8.1
- Java Version 8 update 66
- Eclipse Version: Mars Release (4.5.1)
- Gui.jar Version 2.0.1

Produktets udvikling er dokumenteret og versioneret med GitHub, og kan ses på nedenstående link:

https://github.com/Quaade94/13_CDIO_FINAL

Videreudvikling

Gennem projektet blev der prioriteret og det var vigtigt for os at vi havde et spil der kunne fungere, sådan at all felterne havde en funktion. Derfor blev pantsætning, som er beskrevet i use case ID: 20, nedprioriteret. Dette kunne implementeres i en evt. videreudvikling af spillet.

Yderligere er der i det klassiske Matador spil mange flere prøv lykken kort, som man kunne arbejde videre med. Dette betyder også at der i vores spil er en anden balance, mellem hvilke prøv lykken kort man kan trække, da vi har implementeret et lavere antal end i det originale. For eksempel har vi implementeret et prøv lykken kort, hvor man rykker til Rådhuspladsen. Dette gør Rådhuspladsen lettere at lande på, da kortet udgør en langt større procentdel af kortene end normalt og derfor har en anden værdi end i det originale spil. Denne balance kunne blive genoprettet i en evt. videreudvikling, som følge af flere prøv lykken kort.

Konklusion

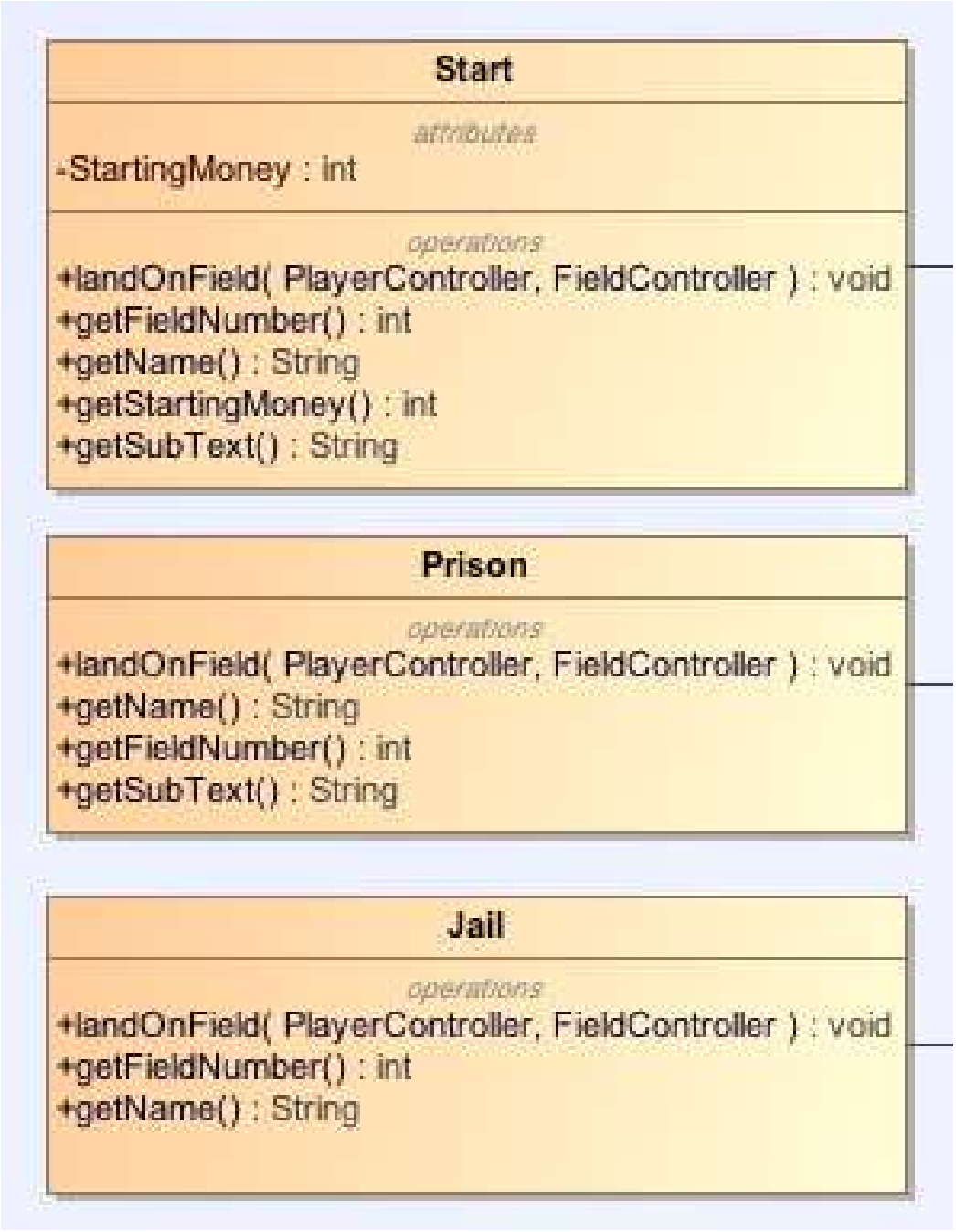
Projektet er blevet færdig og indeholder langt de fleste planlagte features. Igen havde vi kun genbrugt meget begrænset fra CDIO 3, da det projekt ikke var objektorienteret.
Features der ikke nåede at blive implementeret er for eksempel det at kunne pantsætte grunde. Vi valgte ikke at prioritere denne feature, da vi mener spillet kan spilles med stor tilfredsstillelse uden den.
Derudover også benådingskortet mangler, da vi mente det var et meget stort arbejde i forhold til hvor stor betydning den ville have i spillet.

Til gengæld fik vi inkluderet features så som salg af huse og grunde, som vi ellers ikke havde sat vores håb op efter, men som har givet meget til spillets taktiske aspekt, der derved gør spillet det sjovere. Altså selvom vi ikke nåede alt, kan vi konkluderer at vi har fået de vigtigste features med.

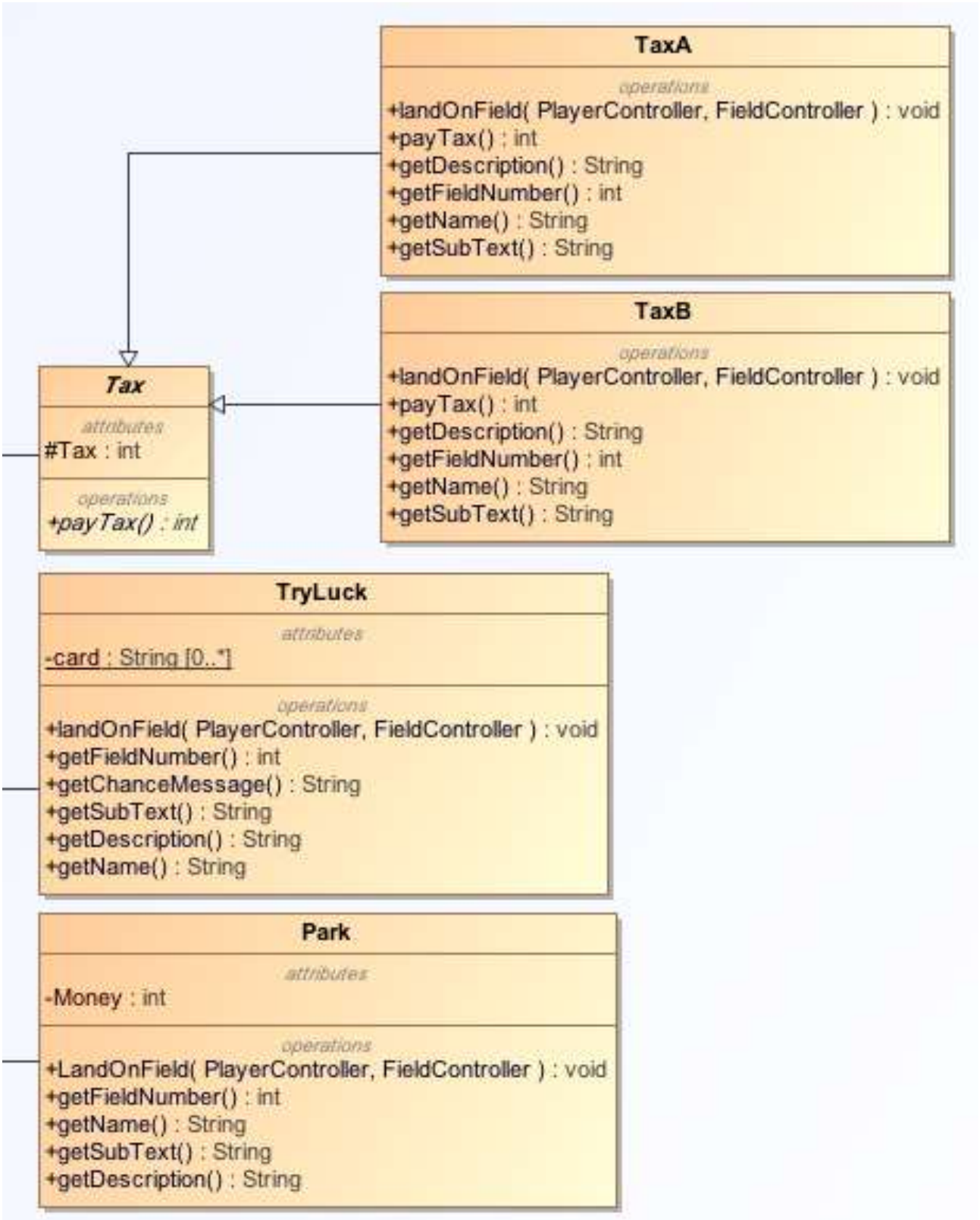
Bilag

Bilag 1, DKD

Figur 6:



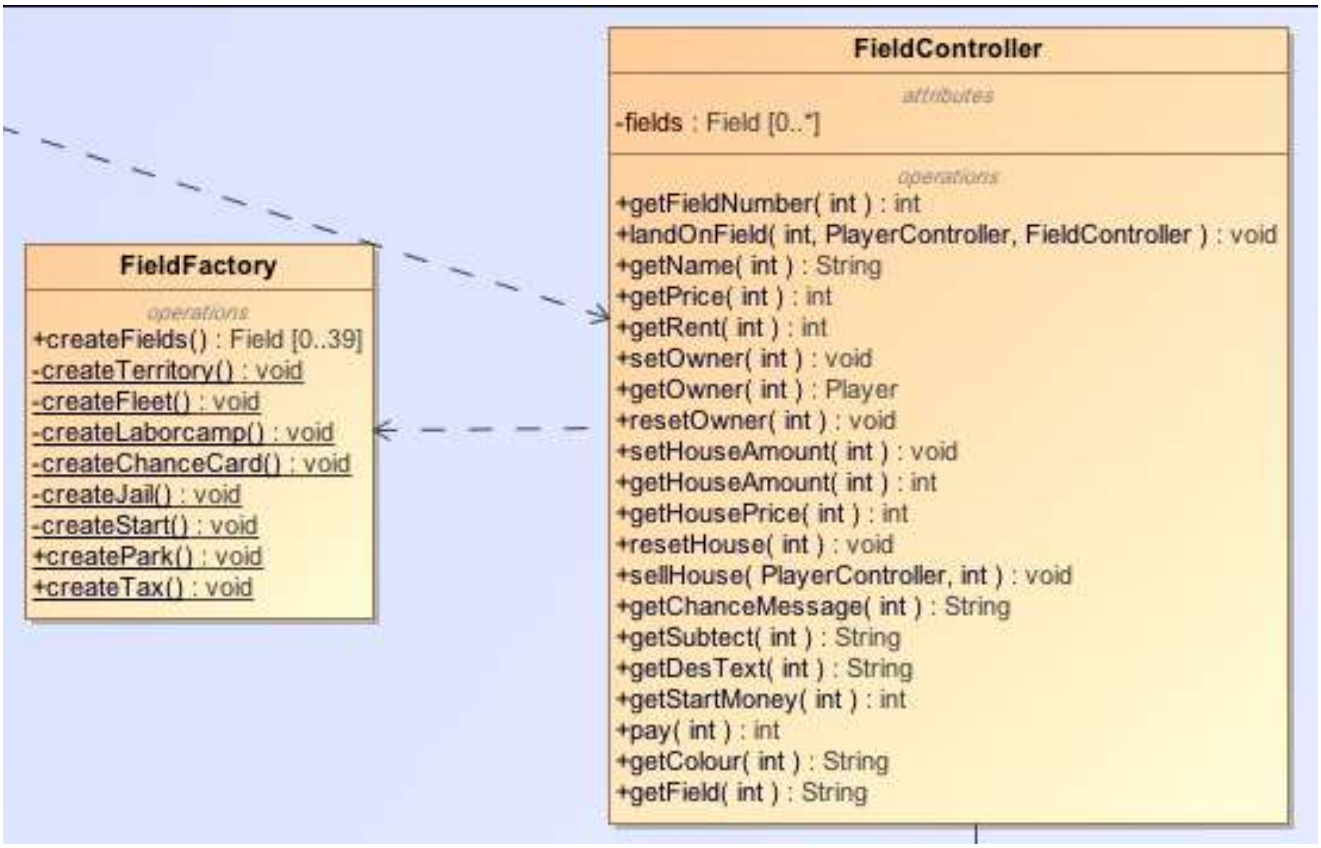
Figur 7:


TaxA
operations
+landOnField(PlayerController, FieldController) : void
+payTax() : int
+getDescription() : String
+getFieldNumber() : int
+getName() : String
+getSubText() : String
TaxB
operations
+landOnField(PlayerController, FieldController) : void
+payTax() : int
+getDescription() : String
+getFieldNumber() : int
+getName() : String
+getSubText() : String
Tax
attributes
#Tax : int
operations
+payTax() : int
TryLuck
attributes
-card : String [0..*]
operations
+landOnField(PlayerController, FieldController) : void
+getFieldNumber() : int
+getChanceMessage() : String
+getSubText() : String
+getDescription() : String
+getName() : String
Park
attributes
-Money : int
operations
+LandOnField(PlayerController, FieldController) : void
+getFieldNumber() : int
+getName() : String
+getSubText() : String
+getDescription() : String

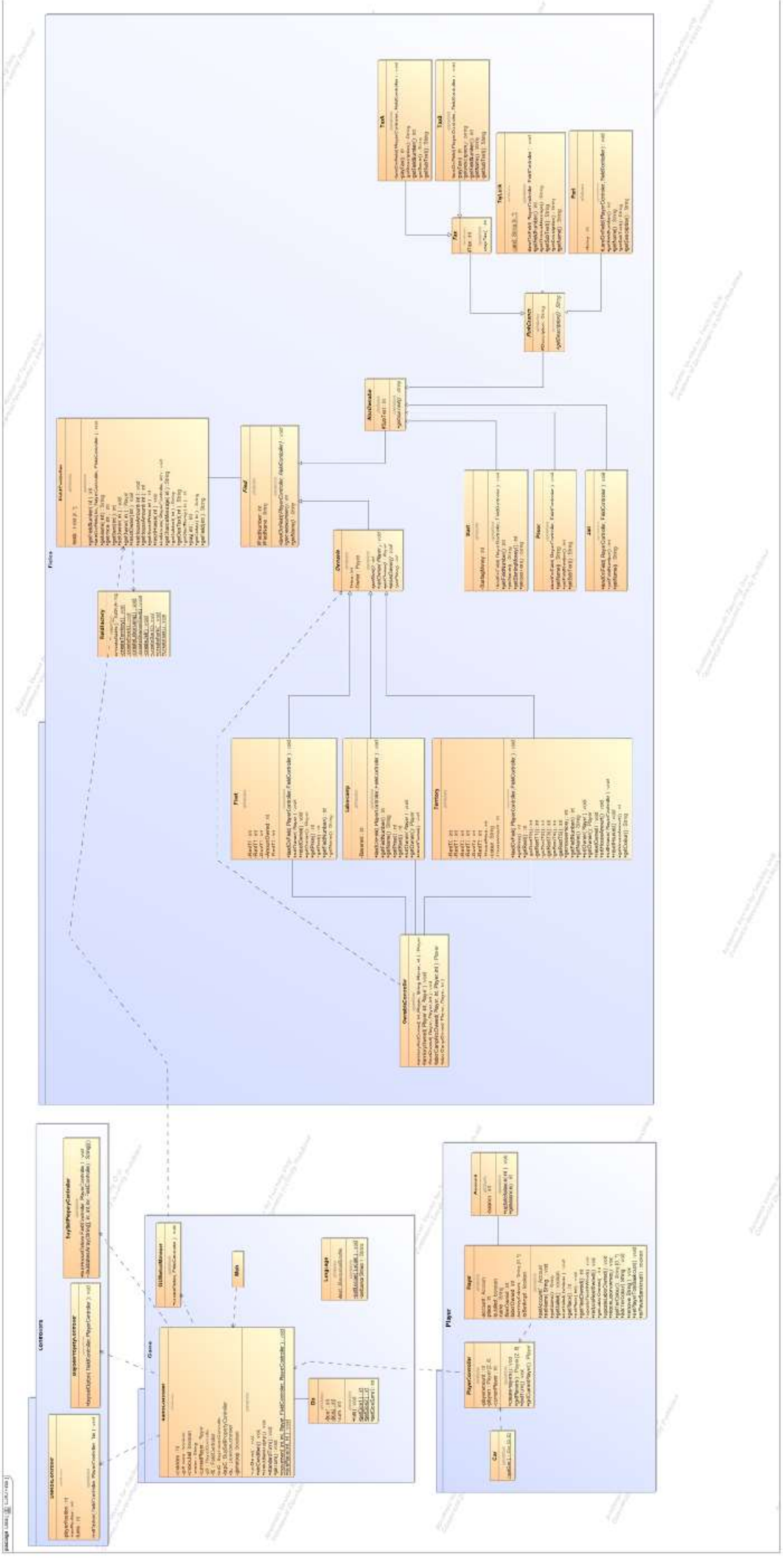
Figur 8:


Fleet
attributes
-RentT0 : int
-RentT1 : int
-RentT2 : int
-AmountOwned : int
-RentT3 : int
operations
+landOnField(PlayerController, FieldController) : void
+setOwner(Player) : void
+resetOwner() : void
+getOwner() : Player
+getPrice() : int
+getRent() : int
+getFieldNumber() : int
+getName() : String
Laborcamp
attributes
-Baserent : int
operations
+landOnField(PlayerController, FieldController) : void
+getFieldNumber() : int
+getName() : String
+getPrice() : int
+getRent() : int
+setOwner(Player) : void
+getOwner() : Player
+resetOwner() : void
Territory
attributes
-RentT0 : int
-RentT1 : int
-RentT2 : int
-RentT3 : int
-RentT4 : int
-RentT5 : int
-HousePrice : int
-colour : String
-HouseAmount : int
operations
+landOnField(PlayerController, FieldController) : void
+getPrice() : int
+getRent() : int
-getRentT0() : int
-getRentT1() : int
-getRentT2() : int
-getRentT3() : int
-getRentT4() : int
-getRentT5() : int
+getHousePrice() : int
+getFieldNumber() : int
+getName() : String
+setOwner(Player) : void
+getOwner() : Player
+resetOwner() : void
+setHouseAmount() : void
+sellHouse(PlayerController) : void
+resetHouses() : void
+getHouseAmount() : int
+getColour() : String

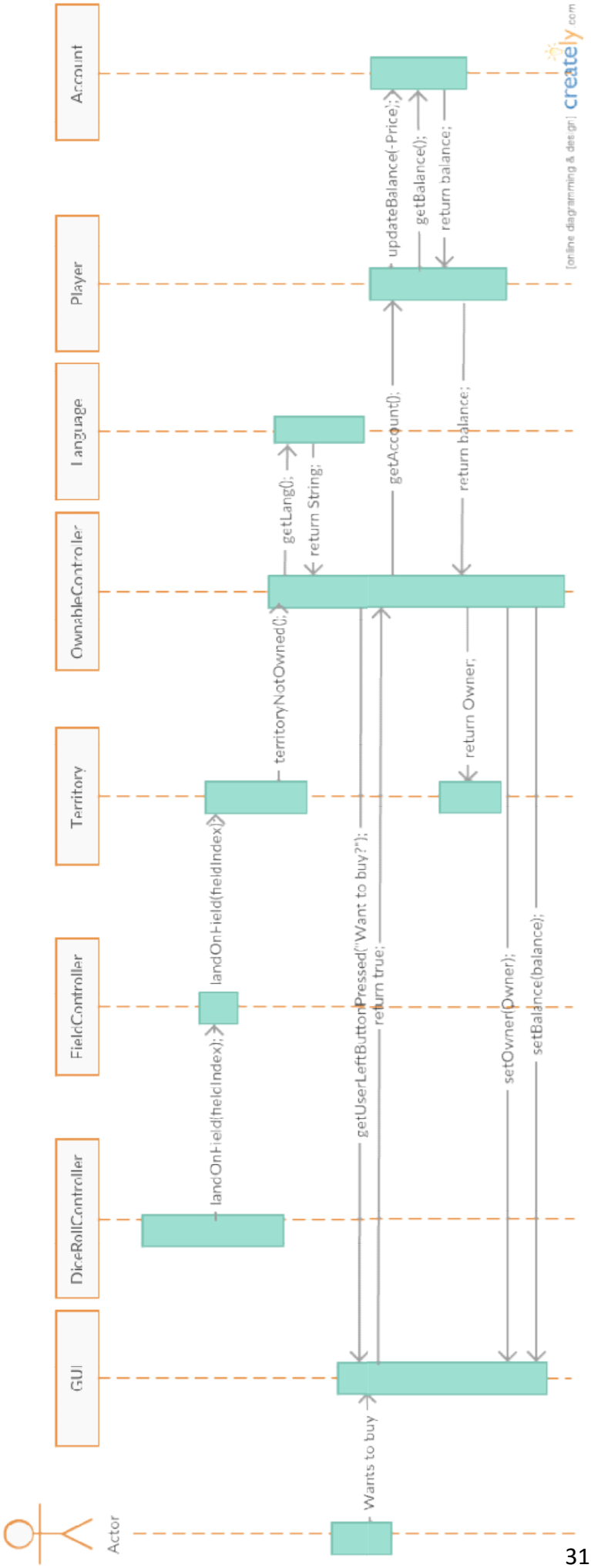
Figur 9:


FieldController
attributes
-fields : Field [0..*]
operations
+getFieldNumber(int) : int
+landOnField(int, PlayerController, FieldController) : void
+getName(int) : String
+getPrice(int) : int
+getRent(int) : int
+setOwner(int) : void
+getOwner(int) : Player
+resetOwner(int) : void
+setHouseAmount(int) : void
+getHouseAmount(int) : int
+getHousePrice(int) : int
+resetHouse(int) : void
+sellHouse(PlayerController, int) : void
+getChanceMessage(int) : String
+getSubtect(int) : String
+getDesText(int) : String
+getStartMoney(int) : int
+pay(int) : int
+getColour(int) : String
+getField(int) : String
FieldFactory
operations
+createFields() : Field [0..39]
-createTerritory() : void
-createFleet() : void
-createLaborcamp() : void
-createChanceCard() : void
-createJail() : void
-createStart() : void
+createPark() : void
+createTax() : void

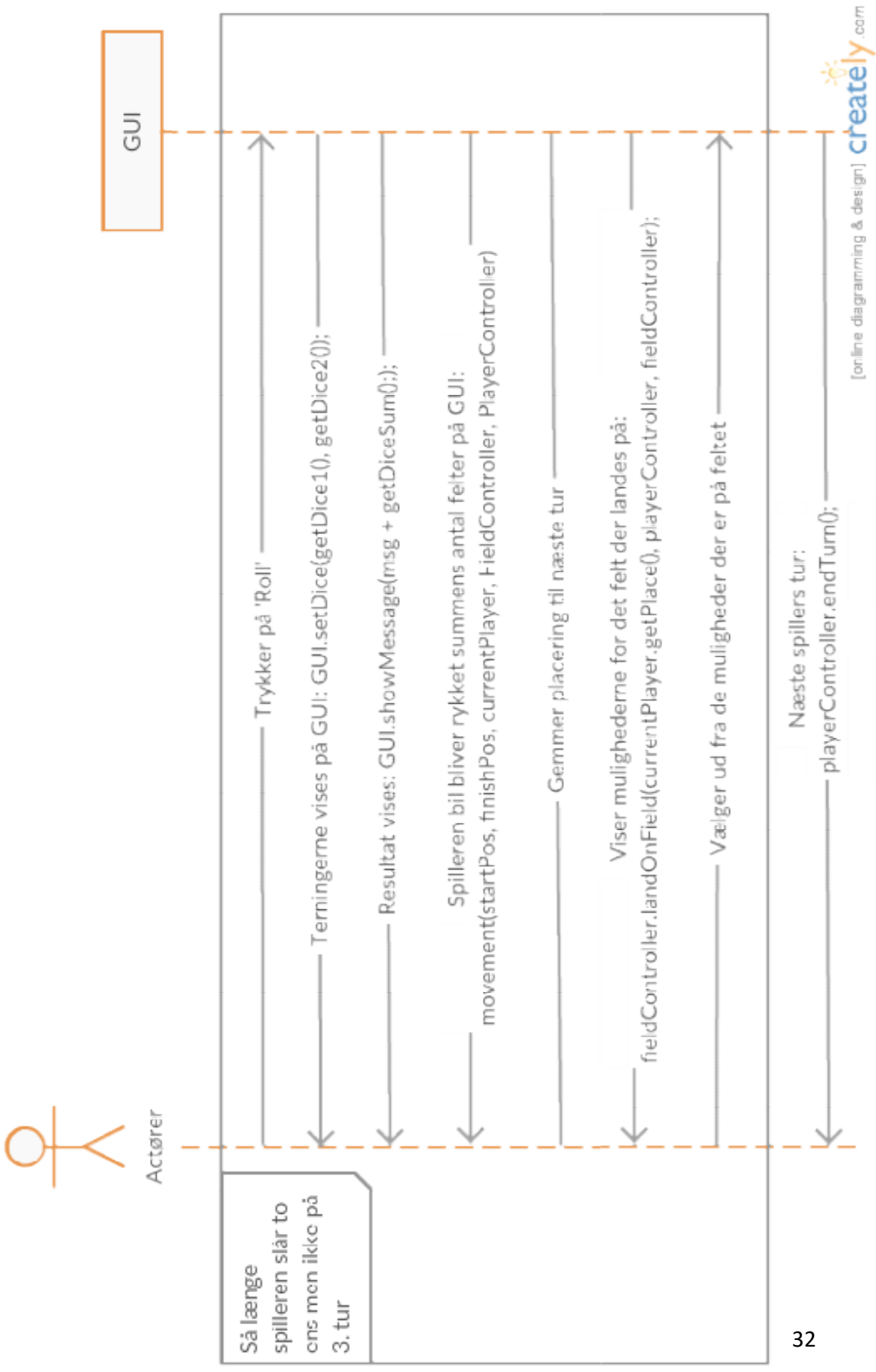
selve DKD'en:



Bilag 2, DSD


Account
Player
Language
OwnableController
Territory
FieldController
DiceRollController
GUI
Actor
updateBalance(-Price);
getBalance();
return balance;
getLang();
return String;
getAccount();
return balance;
territoryNotOwned();
return Owner;
landOnField(fieldIndex);
landOnField(fieldIndex);
getUserLeftButtonPressed("Want to buy?");
return true;
setOwner(Owner);
setBalance(balance);
Wants to buy
[online diagramming & design] creately.com

Bilag 3, SSD


GUI
Actører
Så længe spilleren slår to ens men ikke på 3. tur
Trykker på 'Roll'
Terningerne vises på GUI: GUI.setDice(getDice1(), getDice2());
Resultat vises: GUI.showMessage(msg + getDiceSum());
Spilleren bil bliver rykket summens antal felter på GUI: movement(startPos, finishPos, currentPlayer, FieldController, PlayerController)
Gemmer placering til næste tur
Viser mulighederne for det felt der landes på: fieldController.landOnField(currentPlayer.getPlace(), playerController, fieldController);
Vælger ud fra de muligheder der er på feltet
Næste spillers tur: playerController.endTurn();
[online diagramming & design] creately.com

Bilag 4, use cases

Start Spil
ID: 01
Kort beskrivelse: Spillere vil starte spillet
Primær Aktør: Spiller
Sekundære Aktører:
Forudsættelser: Der er nok spillere til at starte spillet
Primært Flow: 1. Spilleren med programmet starter spillet 2. Spilleren indtaster antal spillere 3. Spilleren indtaster spillernes navne
Efterfølge: Første spiller starter sin tur
Alternative Flows: 1. Spilleren med programmet starter spillet 2. Spilleren indtaster et ugyldigt antal spillere og programmet fortsætter først når et gyldigt antal antal spillere er valgt 3. Spilleren indtaster, efter at have indtastet et gyldigt antal spillere, et ugyldigt spillernavn og programmet fortsætter først når der er indtastet et gyldigt navn

Spiller Slår Med Terninger
ID: 02
Kort beskrivelse: Spiller slår med terninger
Primær Aktør: Spiller
Sekundære Aktører:
Forudsættelser: Spillerens tur er startet
Primært Flow: 1. Spiller trykker på “roll” for at rulle terningerne. 2. Resultat af slaget vises. 3. Resultat bliver lagt sammen. 4. Spillerens bil på GUI’en rykker antal felter som resultatet gav. 5. Gemmer placering på pladen til næste tur.
Efterfølge: Turen skifter til næste spiller
Alternative Flows: 1. <Include Spiller Slår 2 Ens>

Spiller Slår 2 Ens
ID: 03
Kort beskrivelse: Spiller slår med terninger
Primær Aktør: Spiller
Sekundære Aktører:
Forudsættelser: Spillers tur er startet
Primært Flow: 1. <Include Spiller Slår med Terninger> 2. De 2 terninger har samme værdi 3. Spiller færdiggører sin tur og tager en ekstra tur
Efterfølge: Turen skifter til næste spiller
Alternative Flows: A 1. <Include Primært Flow> 2. Spiller har slået 3 gange 2 ens i træk og rykker til feltet “Besøg I Fængsel” og får status: Fængslet

Land På Territory
ID: 04
Kort beskrivelse: Spilleren lander på et Territory felt
Primær Aktør: Spiller
Sekundære Aktører:
Forudsættelser: Spillet er startet, det er spillerens tur
Primært Flow: 1. <Include spiller slår med terninger> 2. Spiller lander på et Territory 3. Spiller får tilbud om at købe feltet 4. Spiller trykker “Ja” og har tilstrækkeligt med penge på konto 5. Spiller bliver sat som ejer af grunden
Efterfølge: Næste spillers tur
Alternative Flows: A 1. <Include spiller slår med terninger> 2. Spiller lander på et Territory 3. Spiller betaler ejer af felt passende beløb B 1. <Include spiller slår med terninger> 2. Spiller lander på et Territory 3. Spiller får tilbud om at købe feltet 4. Spiller trykker “Ja” og har ikke tilstrækkeligt med penge på konto 5. Spiller får besked om at han ikke har råd C 1. <Include spiller slår med terninger> 2. Spiller lander på et Territory 3. Spiller får tilbud om at købe feltet 4. Spiller vælger ikke at købe feltet

Land På Tuborg / Carlsberg
ID: 05
Kort beskrivelse: Spilleren har slået og lander derefter på et Labour camp felt. Labor camps: Tuborg felt og Carlsberg felt.
Primær Aktør: Spiller
Sekundære Aktører:
Forudsættelser: Spillet er startet og det er spillerens tur
Primært Flow: 1. <Include spiller slår med terninger> 2. Spiller lander på Labour camp felt 3. Spiller bliver tilbudt at købe feltet 4. Spiller trykker "Ja" og har tilstrækkeligt med penge på konto 5. Spiller bliver sat som ejer af grunden
Efterfølge: Næste spillers tur
Alternative Flow: A 1. <Include spiller slår med terninger> 2. Spiller lander på Labour camp felt 3. Spiller betaler ejer af felt passende beløb B 1. <Include Spiller Slår Med Terninger> 2. Spiller lander på et Labour camp felt 3. Spiller bliver tilbudt at købe felt 4. Spiller trykker "Ja" men har ikke penge nok 5. Spiller får en besked om ikke at have nok penge på kontoen C 1. <Include Spiller Slår Med Terninger> 2. Spiller lander på Labour camp felt 3. Spiller bliver tilbudt at købe felt 4. Spiller trykker "Nej" og der sker ikke mere

Land På Fleet
ID: 06
Kort beskrivelse: Spilleren har slået og lander derefter på et Fleet felt
Primær Aktør: Spiller
Sekundære Aktører:
Forudsættelser: Spillet er startet og det er spillerens tur
Primært Flow: 1. <Include Spiller Slår Med Terninger> 2. Spiller lander på Fleet felt 3. Spiller bliver tilbudt at købe felt 4. Spiller trykker “Ja” og betaler prisen
Efterfølge: Næste spillers tur
Alternative Flow: A 1. <Include Spiller Slår Med Terninger> 2. Spiller lander på Fleet felt 3. Spiller betaler ejer af felt et passende beløb B 1. <Include Spiller Slår Med Terninger> 2. Spiller lander på Fleet felt 3. Spiller bliver tilbudt at købe felt 4. Spiller trykker “Ja” men har ikke penge nok 5. Spiller får en besked om ikke at have nok penge på kontoen C 1. <Include Spiller Slår Med Terninger> 2. Spiller lander på Fleet felt 3. Spiller bliver tilbudt at købe felt 4. Spiller trykker “Nej” og der sker ikke mere

Land På Tax
ID: 07
Kort beskrivelse: Spilleren har slået og lander derefter på et Tax felt
Primær Aktør: Spiller
Sekundære Aktører:
Forudsættelser: Spillet er startet og det er spillerens tur
Primært Flow: 1. <Include spiller slår med terninger> 2. Spiller lander på Tax felt 3. Spiller betaler et fast beløb
Efterfølge: Næste spillers tur
Alternative Flow: 1. <Include spiller slår med terninger> 2. Spiller lander på Tax felt 3. Spiller betaler 10% af egen pengebeholdning

Land På Prøv Lykken
ID: 08
Kort beskrivelse: Spilleren har slået og lander derefter på et Prøv Lykken felt
Primær Aktør: Spiller
Sekundære Aktører:
Forudsættelser: Spillet er startet og det er spillerens tur
Primært Flow: 1. <Include spiller slår med terninger> 2. Spiller lander på Prøv Lykken felt 3. Spiller bliver tildelt et tilfældigt kort i “bunken” og der sker en effekt, baseret på hvilket kort der blev trukket
Efterfølge: Næste spillers tur
Alternative Flow:

Land På Besøg Fængsel / Parkering
ID: 09
Kort beskrivelse: Spiller lander på feltet, Besøg Fængsel eller Parkering
Primær Aktør: Spiller
Sekundære Aktører:
Forudsættelser: Spillet er startet og det er spillerens tur
Primært Flow: 1. <Include Spiller Slår Med Terninger> 2. Spiller lander på Besøg Fængsel feltet, eller Parkering feltet
Efterfølge: Næste spillers tur
Alternative Flows:

Passer Start
ID: 10
Kort beskrivelse: Spilleren passere start
Primær Aktør: Spiller
Sekundære Aktører:
Forudsættelser: Spillet er startet, spiller har været en omgang rundt på spillerpladen
Primært Flow: 1. <Include spiller slår med terninger> 2. Spiller rykker frem og passerer eller lander på start 3. Spiller bliver tildelt 4000 kr
Efterfølge: Næste spillers tur
Alternative Flow: 1. Spiller trækker et Prøv Lykken kort, der rykker spilleren i fængsel. 2. Spiller passerer start. 3. Spiller modtager ikke nogle penge.

Land På Fængsel
ID: 11
Kort beskrivelse: Spilleren har slået og lander derefter på Fængsel feltet
Primær Aktør: Spiller
Sekundære Aktører:
Forudsættelser: Spillet er startet og det er spillerens tur
Primært Flow: 1. <Include spiller slår med terninger> 2. Spiller lander på Fængsel felt 3. Spiller bliver rykket til Besøg i Fængsel feltet 4. Spiller får status: “fængslet”
Efterfølge: Næste spillers tur
Alternative Flow:

Betal Sig Ud Af Fængsel
ID: 12
Kort beskrivelse: Spiller betaler penge for at komme ud af fængsel
Primær Aktør: Spiller
Sekundære Aktører:
Forudsættelser: Det er spillerens tur, spiller sidder i fængsel og har status: fængslet
Primært Flow: 1. Spiller trykker på knappen “Pay” 2. Spiller får trukket 1000kr fra sin konto. 3. Spiller er ikke længere fængslet
Efterfølge: Turen skifter til den anden spiller
Alternative Flows:

Slip Ud Af Fængsel
ID: 13
Kort beskrivelse: Spiller slipper ud af fængsel ved at slå med terninger
Primær Aktør: Spiller
Sekundære Aktører:
Forudsættelser: Spillet er startet, spiller sidder i fængsel
Primært Flow: 1. Spiller trykker Roll 2. Spiller slår to ens og slipper ud af fængslet og rykker summen frem
Efterfølge: Næste spillers tur
Alternative Flows: A 1. Spiller trykker roll 2. Spiller slår ikke to ens 3. Der bliver roll'et igen automatisk 4. Spiller slår to ens og slipper ud af fængslet B 1. Spiller trykker roll 2. Spiller slår ikke to ens 3. Der bliver roll'et igen automatisk 4. Spiller slår ikke to ens 5. Der bliver roll'et igen automatisk 6. Spiller slår to ens og slipper ud af fængslet C 1. Spiller trykker roll 2. Spiller slår ikke to ens 3. Der bliver roll'et igen automatisk 4. Spiller slår ikke to ens 5. Der bliver roll'et igen automatisk 6. Spiller slår ikke to ens og turen skifter til næste spiller.

Køb Hus
ID: 14
Kort beskrivelse: Det er spillerens tur og spilleren vil gerne købe huse på sine grunde
Primær Aktør: Spiller
Sekundære Aktører:
Forudsættelser: Spillet er startet og det er dens spillers tur, som ønsker at købe huse
Primært Flow: 1. Spiller trykker “Buy/Sell Houses” 2. Spiller trykker på “Buying” 3. Spiller vælger hvilke grunde han vil købe huse på 4. Spiller bliver trukket passende penge 5. Huse bliver placeret på ønskede grunde
Efterfølge: Det er stadig spillerens tur
Alternative Flow: A) 1. Spiller trykker “Buy/Sell Houses” 2. Spiller trykker på “Buying” 3. Spiller vælger hvilke grunde han vil købe huse på 4. Spiller har ikke råd til at købe huse og kan derfor ikke købe huse på valgte grund B 1. Spiller trykker “Buy/Sell Houses” 2. Spiller trykker på “Buying” 3. Spiller ejer ikke 3 grunde af samme serie og får en besked om at han ikke har nogle grunde der kan bygges huse på 4. Spiller kan trykke ok for at blive sat tilbage i menuen og kan igen vælge mellem de forskellige muligheder

Sælg Hus
ID: 15
Kort beskrivelse: Det er spillerens tur og spilleren vil gerne sælge nogle huse
Primær Aktør: Spiller
Sekundære Aktører:
Forudsættelser: Spillet er startet og det er dens spillers tur, som ønsker at sælge huse
Primært Flow: 1. Spiller trykker “Buy/Sell Houses” 2. Spiller trykker “Selling” 3. Spiller vælge hvilke grunde han vil sælge hus på 4. Spiller får en besked om hvor vidt han er sikker på at han vil sælge sit hus for den halve pris 5. Spiller trykker “Ja” og mister sit hus og får de penge tilbage
Efterfølge: Det er stadig spillerens tur
Alternative Flow: A 1. Spiller trykker “Buy/Sell Houses” 2. Spiller trykker “Selling” 3. Spiller vælger hvilke grunde han vil købe huse på 4. Spiller får en besked om hvor vidt han er sikker på at han vil sælge sit hus for den halve pris 5. Spiller trykker “Nej” og GUI’en går tilbage til menuen B 1. Spiller trykker “Buy/Sell Houses” 2. Spiller trykker “Selling” 3. Spiller får at vide at han ikke har nogen huse at sælge

Køb Grund Af Spiller
ID: 16
Kort beskrivelse: Spilleren vil købe en grund af en anden spiller
Primær Aktør: Spiller
Sekundære Aktører:
Forudsættelser: Det er spillerens tur
Primært Flow: 1. Spiller trykker “Buy/Sell properties” 2. Spiller trykker “Buying” 3. Spiller vælger hvem der skal købes af 4. Spiller vælger hvilken grund han vil købe 5. Spiller vælger prisen han vil betale 6. Sælgeren bliver spurgt om han vil acceptere handlen 7. Sælgeren accepterer og penge bliver overført mellem konti 8. Spilleren (køberen) bliver sat som den nye ejer
Efterfølge: Spilleren har stadig sin tur
Alternative Flows: A 1. Spiller trykker “Buy/Sell properties” 2. Spiller trykker “Buying” 3. Spiller vælger hvem der skal købes af 4. Spiller vælger hvilken grund han vil købe 5. Spiller vælger prisen han vil betale 6. Sælgeren bliver spurgt om han vil acceptere handlen 7. Sælgeren trykker “nej” til handlen 8. GUI’en går tilbage til menuen og spilleren har stadig sin tur B 1. Spiller trykker “Buy/Sell properties” 2. Spiller trykker “Buying” 3. Spiller vælger hvem der skal købes af 4. Der er ikke nogen grunde at købe og spilleren kan trykke OK for at komme tilbage til menuen

Sælg Grund Til Spiller
ID: 17
Kort beskrivelse: Spilleren vil sælge en af sine grunde til en anden spiller
Primær Aktør: Spiller
Sekundære Aktører:
Forudsættelser: Det er spillerens tur
Primært Flow: 1. Spiller trykker “Buy/Sell properties” 2. Spiller trykker “Selling” 3. Spiller vælger hvilken grund der skal sælges 4. Spiller vælger hvem den skal sælges til 5. Spiller vælger prisen den skal sælges for 6. Køberen bliver spurgt om han vil acceptere handlen 7. Køberen accepterer og penge bliver overført mellem konti 8. Køberen bliver sat som den nye ejer
Efterfølge: Spilleren har stadig sin tur
Alternative Flows: A 1. Spiller trykker “Buy/Sell properties” 2. Spiller trykker “Selling” 3. Spiller vælger hvilken grund der skal sælges 4. Spiller vælger hvem den skal sælges til 5. Spiller vælger prisen den skal sælges for 6. Køberen bliver spurgt om han vil acceptere handlen 7. Køberen trykker “nej” til handlen 8. GUI’en går tilbage til menuen og spilleren har stadig sin tur B 1. Spiller trykker “Buy/Sell properties” 2. Spiller trykker “Selling” 3. Spiller har ikke nogen grunde at sælge og kan trykke “OK” for at komme tilbage til menuen

Spiller Går Bankerot
ID: 18
Kort beskrivelse: En spillers konto går under 0 og han bliver fjernet fra spillet
Primær Aktør: Spiller
Sekundære Aktører:
Forudsættelser: Spillet er startet og det er spillerens tur
Primært Flow: 1. <Include Spiller Slår Med Terninger> 2. Spiller lander på en anden spillers grund 3. Spiller kan ikke betale det han skal 4. Ejeren af grunden får de resterende penge på spillerens konto 5. En besked om at denne spiller er ude af spillet kommer op 6. Spillerens bil bliver fjernet fra GUI'en 7. Spillerens konto viser: -1 8. Spillerens grunde bliver frit tilgængelige og kan købes som et ikke-ejet felt
Efterfølge: Næste spillers tur
Alternative Flow: 1. <Include Spiller Slår Med Terninger> 2. Spiller lander på prøv lykken eller tax og skal betale, eller skal betale fødselsdagspenge eller prøver at betale for at slippe ud af fængslet 3. Spiller kan ikke betale det han skal 4. En besked om at denne spiller er ude af spillet kommer op 5. Spillerens bil bliver fjernet fra GUI'en 6. Spillerens konto viser: -1 7. Spillerens grunde bliver frit tilgængelige og kan købes som et ikke-ejet felt

Spiller Vinder
ID: 19
Kort beskrivelse: En anden spiller går bankerot og der er kun én spiller tilbage
Primær Aktør: Spiller
Sekundære Aktører:
Forudsættelser: Spillet er startet, en spiller går bankerot
Primært Flow: 1. <Include Spiller Går Bankerot> 2. Der er kun én spiller tilbage 3. Tilbageværende spiller vinder
Efterfølge: Spillet slutter
Alternative Flow: