Projektopgave efterår 2015 – jan 2016
02312-14 Indledende programmering, 02313 Udviklingsmetoder til IT-Systemer og 02315 Versionsstyring og testmetoder.

Projekt navn: *CDIO_Final*
Gruppe nr.: *38*
Afleveringsfrist: *Mandag den 18/1 2016 Kl. 12:00*

Denne rapport er afleveret via Campusnet

Denne rapport indeholder 40 sider inkl. denne side.

Silas El-Azm Stryhn
s143599

Ramyar Hassani
s143242

Mikkel Hansen
s143603

Frank Thomsen
s124206

Martin Rødgaard
s143043

Timeregnskab

Timeregnskab (hele antal timer)					
	Silas	**Martin**	**Mikkel**	**Frank**	**Ramyar**
Analyse	**2**	**3**	**2**	**6**	**2**
Design	**7**	**1**	**5**	**1**	**6**
Implementering	**15**	**20**	**40**	**23**	**10**
Test	**1**	**3**	**1**	**3**	**2**
Dokumentation	**9**	**7**	**7**	**4**	**5**
I alt	**34**	**34**	**55**	**37**	**25**

Indholdsfortegnelse

Resumé

Vores rapport indeholder et analyseafsnit, hvor vi har analyseret og opdelt alle kundens krav efter MoSCoW metoden. Vi har også udarbejdet nogle diagrammer til analysen samt beskrivelser af disse. I vores design afsnit har vi lavet og beskrevet diverse diagrammer, som gør vores spil mere overskueligt og forståeligt. Det er også i dette afsnit vi kommer ind på blandt andet GRASP. I implementeringsafsnittet har vi blandt andet skrevet om det manglende implantation for at det var et eksakt matador spil og beskrevet vores mest uoverskuelige metode ned i detaljer. Til sidst i rapporten vil vi komme ind på de 4 jUnit tests som vi har lavet af vores spil, som tester nye implementeret funktioner i forhold til CDIO 3 opgaven.

Indledning

Denne rapport præsenterer gennemgangen af den softwaremæssige udvikling af et digitalt brætspil for spilfirmaet IOOuterActive. Firmaets sidste ønske er en eksakt kopi af brætspillet Matador. Dog lægger de vægt på at man hellere implementerer mindre, som fungerer fremfor at man har prøvet at få alt implementeret, og intet virker.

For at kunne udarbejde sådan et spil på den mest effektive måde og for at undgå eventuelle problemer der kunne opstå under udviklingen, har vi været igennem de fire stadier, der er nødvendige for at kunne danne sig en struktureret og overskuelig arbejdsplan: Analyse, design, implementering og test. Rapportens basale formål er at give en grundlæggende forståelse for tankegangen igennem forløbet, hvilke overvejelser der er gjort og hvordan spillet er stykket sammen. Rapporten gennemgår således en analyse af kundens vision for spillet og udarbejdelsen af diverse diagrammer, som er med til at overskueliggøre tilgangen til udviklingen.

Hovedafsnit

Analyse

Vi har analyseret og udarbejdet en kravspecifikation ud fra spillereglerne til matador.
Dertil har vi også udarbejdet nogle diagrammer, der løbende er blevet ændret under udviklingen, efterhånden som vi har skrevet mere og mere på koden. Det vil sige, at diagrammerne der fremgår i denne rapport er et produkt af det færdige spil. Vi har ifølge opgavebeskrivelse udarbejdet hhv. Domæne- og BCE-modeller samt Use-case diagram og beskrivelser og til sidst System-sekvensdiagrammer.

Kravspecifikation

Da det eneste krav fra IOOuterActive var at vi skulle lave et Matador spil, tog vi udgangspunkt i spillereglerne fra det originale brætspil og byggede en kravspecifikation ud fra dette.
Kravene inddelte vi derefter først i funktionelle og ikke funktionelle krav for derefter at bruge MoSCoW metoden på dem og kommentere vores valg.
I implementerings afsnittet kan man læse om hvorfor vi har prioriteret som vi har gjort, da der som sagt er få mangler i vores spil.

1. Basisregler			
1.1 Spillet er mellem 2 og 6 spillere.	Funktionelt	Must have	Dette er et særdeles vigtigt krav da spillet ikke vil kunne spilles uden deltagere.
1.2 Hver spiller starter med en balance på 30000.	Funktionelt	Must have	Det er vigtigt for spillet at spillerne har en balance dog er startbalancens beløb underordnet.
1.3 Spillet består af et spillebræt med 40 felter.(se feltliste)	Funktionelt	Must have	Det er vigtigt for spillet at det består af felter. Derfor har vi valgt at sætte dette krav til must have.
1.4 Spillet består af 28 chancekort.(se kort listen)	Ikke Funktionelt	Could have	Vi synes chance kortene er en mindre del af matadorspillet som er med til at gøre spillet sjovere og vi har derfor valgt at sætte

			dette krav til could have.
2. Spillets Gang			
2.1 Alle spillerne starter på Startfeltet.	Funktionelt	Should have	Det er vigtigt for spillerne at de er placeret på brættet og når der nu er designet et felt til dette synes vi også det er vigtigt at spillerne starter her.
2.2 Spillerne slår på tur med to terninger og rykker sin brik på spillebrættet det antal felter som summen af terningerne viser. Spilleren rykker på den efterfølgende tur videre fra dette felt.	Funktionelt	Must have	Dette er et af de vigtigste krav for spillet og derfor har vi valgt at sætte dette krav til must have.
2.3 Slår en spiller to ens, får spilleren en ekstra tur. Dog bliver spilleren smidt i fængsel hvis han/hun slår to ens 3 gange i træk.	Ikke Funktionelt	Could have	Dette er en mindre del af spillet som er med til at gøre spillet sjovere. Derfor har vi valgt at sætte det til could have.
2.4 En spiller kan i starten af sin tur(før han/hun slår med terningerne) pantsætte sine ubebyggede(uden huse eller hotel) ejendomme og modtage et beløb afhængig af hvilken ejendom som pantsættes.(Se feltliste for de forskellige felters beløb)	Ikke Funktionelt	Could have	Pantsætningen er en lille del af spillet som igen er med til at gøre spillet sjovere. Vi har derfor valgt at sætte dette krav til could have.
2.5 Pantsatte ejendomme indtjener ingen leje.	Ikke Funktionelt	Could have	Dette krav hænger meget sammen med krav 2.4 og er derfor heller ikke særlig højt prioriteret. Dog hvis krav 2.4 er implementeret, er dette krav vigtigt at få med.
2.6 En spiller kan i starten af sin tur(før han/hun slår med terningerne) ophæve en pantsætning. Dette koster 10% mere end det beløb han/hun modtog for pantsættelsen.	Ikke Funktionelt	Could have	Dette krav er hænger meget sammen med krav 2.4 og er derfor ikke særlig højt prioriteret.
2.7 En spiller er bankerot når hans/hendes balance er 0.	Funktionelt	Must have	Det er vigtigt for os at få dette krav med da vi ønsker at vores spil skal kunne slutte og en spiller skal kunne stå tilbage som vinder.
2.8 En spiller forlader spillet når han/hun er bankerot.	Funktionelt	Must have	Det er vigtigt for os at få dette krav med da vi ønsker at vores spil skal kunne slutte og en spiller skal kunne stå tilbage som vinder.
2.9 Spillet slutter når alle på nær en spiller er	Funktionelt	Must	Det er vigtigt for os at få

bankerot.		have	dette krav med da vi ønsker at vores spil skal kunne slutte og en spiller skal kunne stå tilbage som vinder.
3. Felterne			
3.1 Startfeltet			
3.1.1 Når en spiller lander på eller passerer dette felt modtager han/hun 4000.	Ikke Funktionelt	Could have	I det traditionelle matadorspil er dette en vigtig regel, og det er det også efter vores mening så vi har derfor sat dette krav til should have.
3.2 Prøv lykken			
3.2.1 Lander en spiller på dette felt trækker spilleren et 'Chancekort' og følger kortets instrukser.(se liste over chancekort)	Ikke Funktionelt	Could have	Chance kortene er en mindre del af spillet så vi har valgt at sætte dette krav til could have.
3.3 Territory			
3.3.1 Hvis en spiller lander på et felt af denne type kan personen købe feltet hvis dette ikke allerede er ejet af en anden spiller.(Se feltliste for priserne)	Ikke Funktionelt	Must have	At kunne købe grunde i matador er vigtigt, vi har derfor valgt at sætte dette krav til must have.
3.3.2 Alle felter af denne kategori er unikke og tilhøre en farvegruppe sammen med et eller to andre felter.(Se feltliste for farve grupperne)	Ikke Funktionelt	Should have	Vi synes dette krav er vigtigt da det er en stor del af spillet.
3.3.3 Hvis en spiller ejer alle felter i en farvegruppe kan han/hun i starten af sin tur(før han/hun slår med terningerne) købe huse eller hotel på felterne i denne farvegruppe. Husene skal dog købes jævnt, dvs. at der ikke må være mere end et hus forskel på felterne i en farvegruppe. (Se feltliste for prisen på huse og hoteller)	Ikke Funktionelt	Should have	Dette krav er en vigtig del af et matadorspil, da det både gør spillet sjovere men også er med til at reducere spillets længde betydeligt. Vi synes det er vigtig at få implementeret og har derfor valgt at sætte dette krav til should have.
3.3.4 Et Territory kan have op til 4 huse, herefter kan spilleren købe et hotel(dette vil dog fjerne husene)	Ikke Funktionelt	Should have	Hvis man skal kunne bygge huse på sin grund er det også vigtigt at have et loft for dette. Derfor har vi valgt at sætte dette krav til should have.
3.3.5 En spiller kan i starten af sin tur(før han/hun slår med terningerne) sælge sine huse og hoteller til halvdelen af den pris de blev købt for.	Ikke Funktionelt	Could have	Dette krav har vi ikke prioriteret særlig højt, da det ikke er en særlig stor del af spillet.

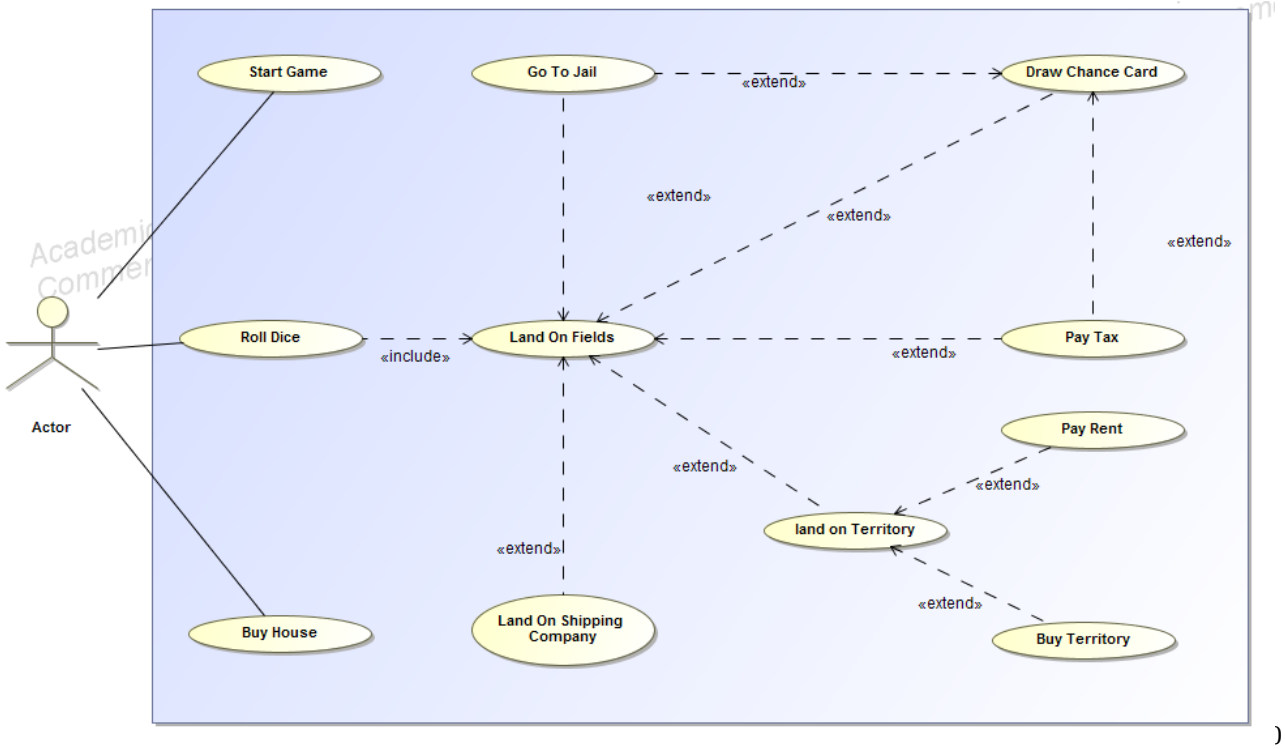
3.3.6 Hvis en spiller lander på et felt af denne type og feltet er ejet af en anden spiller skal spilleren betale leje til ejeren.	Ikke Funktionelt	Must have	Dette krav synes vi er vigtigt, da det er med til at afgøre hvem som vinder spillet og det giver spillerne en grund til at købe grunde.
3.3.6.1 Lejen afhænger af hvilket felt der er tale om(se feltliste for leje afgifterne.)	Ikke Funktionelt	Should have	Dette krav gør spillet sjovere og mere varieret og det synes vi er vigtigt at få med.
3.3.6.2 Ubebyggede grundes leje fordobles hvis alle territories i farvegruppen er ejet af samme spiller.	Ikke Funktionelt	Could have	Dette krav er en lille feature som vi ikke har prioriteret særlig højt.
3.3.6.3 Lejen afhænger af om ejeren har hus/huse eller hotel på det gældende felt.	Ikke Funktionelt	Should have	Dette krav er med til at give spillerne en masse valg om hvor man skal satse sine penge og det synes vi er vigtigt i et matadorspil.
3.4 Brewery			
3.4.1 Hvis en spiller lander på et felt af denne type kan han/hun købe feltet for 3000, hvis dette ikke allerede er ejet af en anden spiller.	Ikke Funktionelt	Must have	Dette krav synes vi er vigtigt og har derfor valgt at sætte kravet til must have
3.4.2 En spiller kan i starten af sin tur(før han/hun slår med terningerne) pantsætte sine Breweries.	Ikke Funktionelt	Could have	Kravet er en meget lille del af spillet og vi har derfor valgt ikke at prioritere det særlig højt.
3.4.3 Hvis en spiller lander på et felt af denne type og feltet er ejet af en anden spiller skal spilleren betale leje til ejeren.	Ikke Funktionelt	Must have	Dette krav synes vi er vigtigt da det ellers ikke vil give mening at købe feltet.
3.4.3.1 Hvis ejeren ejer et Brewery er lejen 100 gange summen af øjnene som spilleren brugte på at lande på dette felt.	Ikke Funktionelt	Should have	Kravet gør at lejen er sjov og anderledes og det synes vi er vigtigt at få med.
3.4.3.2 Hvis ejeren ejer begge Breweries er lejen 200 gange summen af øjnene som spilleren brugte på at lande på dette felt.	Ikke Funktionelt	Should have	Dette krav er en udvidet del af krav 3.4.3.1 og det synes vi også er vigtigt at få med så derfor har vi valgt at sætte dette krav til should have.
3.5 Shipping Company			
3.5.1 Hvis en spiller lander på et felt af denne type kan han/hun købe feltet for 4000, hvis dette ikke allerede er ejet af en anden spiller.	Ikke Funktionelt	Must have	Dette krav synes vi er vigtigt og har derfor valgt at sætte kravet til must have.

3.5.2 Hvis en spiller lander på et felt af denne type og feltet er ejet af en anden spiller skal spilleren betale leje til ejeren. Lejen afhænger af hvor lange Shipping Companies ejeren ejer.(Se feltliste for lejen)	Ikke Funktionelt	Must have	Dette krav gør at det vil give mening at købe Shipping Companies og er derfor vigtigt.
3.5.3 En spiller kan i starten af sin tur (før han/hun slår med terningerne) pantsætte sine Shipping Companies.	Ikke Funktionelt	Could have	Dette krav er ikke prioriteret særlig højt. Da det er en meget lille del af spillet.
3.6 Go to Jail			
3.6.1 Lander en Spiller på dette felt bliver spilleren rykket i fængsel.	Ikke Funktionelt	Should have	Dette krav synes vi er vigtigt at få med i spillet og har derfor valgt at sætte det til should have.
3.7 Jail			
3.7.1 Lander man på dette felt er man på besøg i fængslet.	Ikke Funktionelt	Should have	Dette krav gør at feltet har en funktion for folk som ikke er i fængsel.
3.7.2 Når en spiller er i fængsel har han/hun følgende 3 valgmuligheder.	Ikke Funktionelt	Should have	Dette krav er temmelig vigtigt at få med hvis jail er implementeret, da spilleren ihvertfald skal have en mulighed for at kunne komme ud.
3.7.2.1 Betale 1000 og rykke ud af fængsel og slå med terningerne.	Ikke Funktionelt	Could have	Dette krav er ikke så vigtigt hvis ihvertfald bare en af de andre er med.
3.7.2.2 Bruge et løsladelseskort og rykke ud af fængslet og slå med terningerne.	Ikke Funktionelt	Could have	Dette krav er ikke så vigtigt hvis ihvertfald bare en af de andre er med.
3.7.2.3 Slå med terningerne, hvis man slår 2 ens rykker man ud af fængselet og fortsætter det antal summen af terninge øjnene viser. Dog har man kun mulighed for dette 3 gange og fejler man tredje gang er man tvunget til at bruge muligheden med at betale 1000.	Ikke Funktionelt	Could have	Dette krav er ikke så vigtigt hvis ihvertfald bare en af de andre er med.
3.7.3 Rykker man ud af fængslet fortsætter man fra feltet “Jail”.	Funktionelt	Should have	Dette krav er kun et funktionelt krav såfremt at jail er implementeret, da spilleren skal kunne fortsætte sit spil. Dog er hele jail featuren ikke nødvendigt for at spillet fungerer.

3.8 Parkering			
3.8.1 Lander en spiller på dette felt sker der intet og næste spillers tur begynder.	Ikke Funktionelt	Should have	Dette krav omhandler et enkelt felt på spillebrættet, dog er let at få implementeret så vi synes det er vigtigt at få med.
3.9 Tax			
3.9.1 Lander en spiller på et tax felt mister spilleren et beløb.(Se feltliste for at se beløbene)	Ikke Funktionelt	Must have	Dette krav er en ret lille del af spillet da det kun dækker over 2 felter dog synes vi det er vigtigt at få med.

Use Case Diagram

Vores use case diagram giver et overblik over hvilke use cases vores system indeholder. De er hver især blevet tildelt et use case ID, som der er blevet lavet use case beskrivelser til. Diagrammet viser os overordnet set, hvad aktøren (Spilleren) kan blive udsat for i løbet af spillet. I diagrammet ser vi, at spilleren har 3 hoved muligheder i: starte spillet, købe hus/hotel og at rulle med terningerne. Når spilleren har rullet med terningerne, lander spilleren på en felttype, og den type afgør så hvad der skal ske. Spilleren kan, når alle felter af en farve er ejet, få lov til at købe huse på sine grunde. Spilleren kan også risikere at ryge i fængsel eller at trække et chancekort, hvis han/hun skulle lande på et af de felter, der har disse effekter. De mest centrale use cases vil blive beskrevet og forklaret dybere i use case beskrivelserne på de følgende sider.



Use-case beskrivelser

Use Case Name:	Start Game
Use Case ID:	1
Actors:	Player
Description:	Player starter spillet op.
Preconditions:	Spillet er i gang.
Postconditions:	Sprog, antal spillere og navne er valgt.
Main flow:	1. Vælg sprog (Dansk/Engelsk) 2. Vælg antal spillere mellem 2-6 3. Navngiv spillere
Alternative Flow:	1. Spillet lukkes ned

Use Case Name:	Roll Dice
Use Case ID:	2
Actors:	Player
Description:	Player ruller med terningerne
Preconditions:	Spillet er i gang.
Postconditions:	Player er landet på et felt.
Main flow:	1. Slå med terningerne
Alternative Flow:	1. Spillet lukkes ned 2. Spillet er vundet

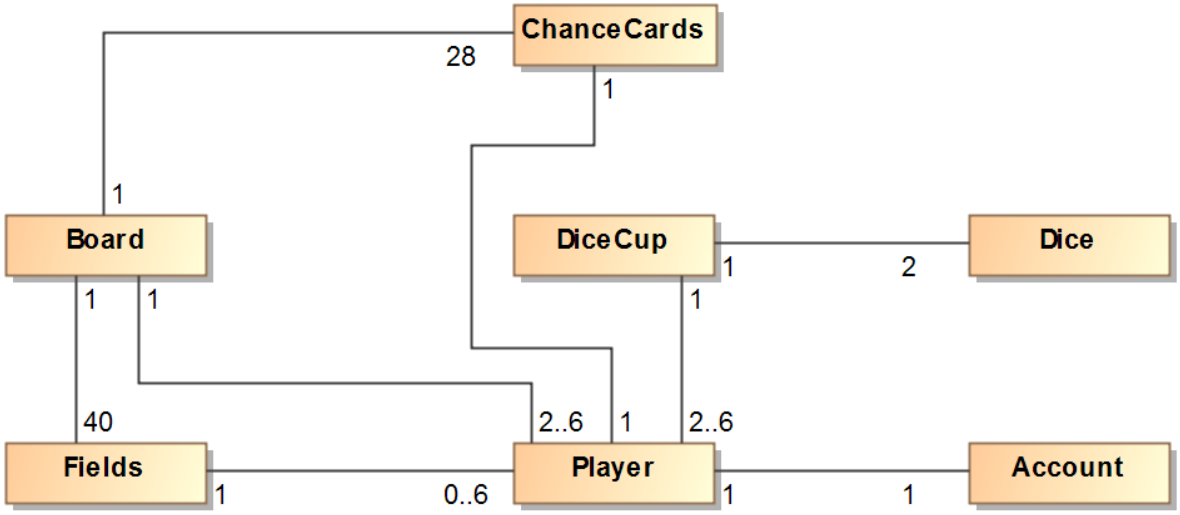
Use Case Name:	Buy House
Use Case ID:	3
Actors:	Player
Description:	Player kan købe huse.
Preconditions:	Spillet ejer alle grunde i en farve.
Postconditions:	Player har købt hus/huse/hotel på en grund.
Main flow:	1. Vælg “Køb hus/hotel” 2. Vælg grund 3. Køb hus 3.1. Hvis der er købt 4 huse, køb hotel.
Alternative Flow:	1. Vælg “Køb hus/hotel” 2. Vælg “Gå tilbage” 3. Slå med terningerne (Roll Dice ID 2)

Use Case Name:	Land On Fields
Use Case ID:	4
Actors:	Player
Description:	Player lander på et felt
Preconditions:	Spillet har slået med terningerne (Roll Dice ID 2).
Postconditions:	Player modtager en besked.
Main flow:	1. Lande på et Territory 1.1. Se Use Case ID 5 2. Lande på et Shipping Company/Brewery 2.1. Hvis feltet ikke er ejet 2.1.1. Mulighed “Køb felt” 2.2. Hvis feltet er ejet 2.2.1. Betale rente til ejeren 3. Lande på GoToJail felt 3.1. Ryk i fængsel 4. Lande på Chance felt 4.1. Træk et kort 4.1.1. Player gør som beskrevet på kortet 5. Lande på Tax felt 5.1. Betal Tax
Alternative Flow:	1. Player er Jailed

Use Case Name:	Land On Territory (Extends fra Land On Fields ID 4)
Use Case ID:	5
Actors:	Player
Description:	Player lander på et Territrory felt
Preconditions:	Spiller har slået med terningerne og har landet på et Territory.
Postconditions:	Player modtager en besked.
Main flow:	1. Hvis feltet ikke er ejet 1.1. Mulighed “Køb felt” 2. Hvis feltet er ejet 2.1. Betal rente til ejeren
Alternative Flow:	1. Spiller lander på andre felttyper 2. Spiller er Jailed

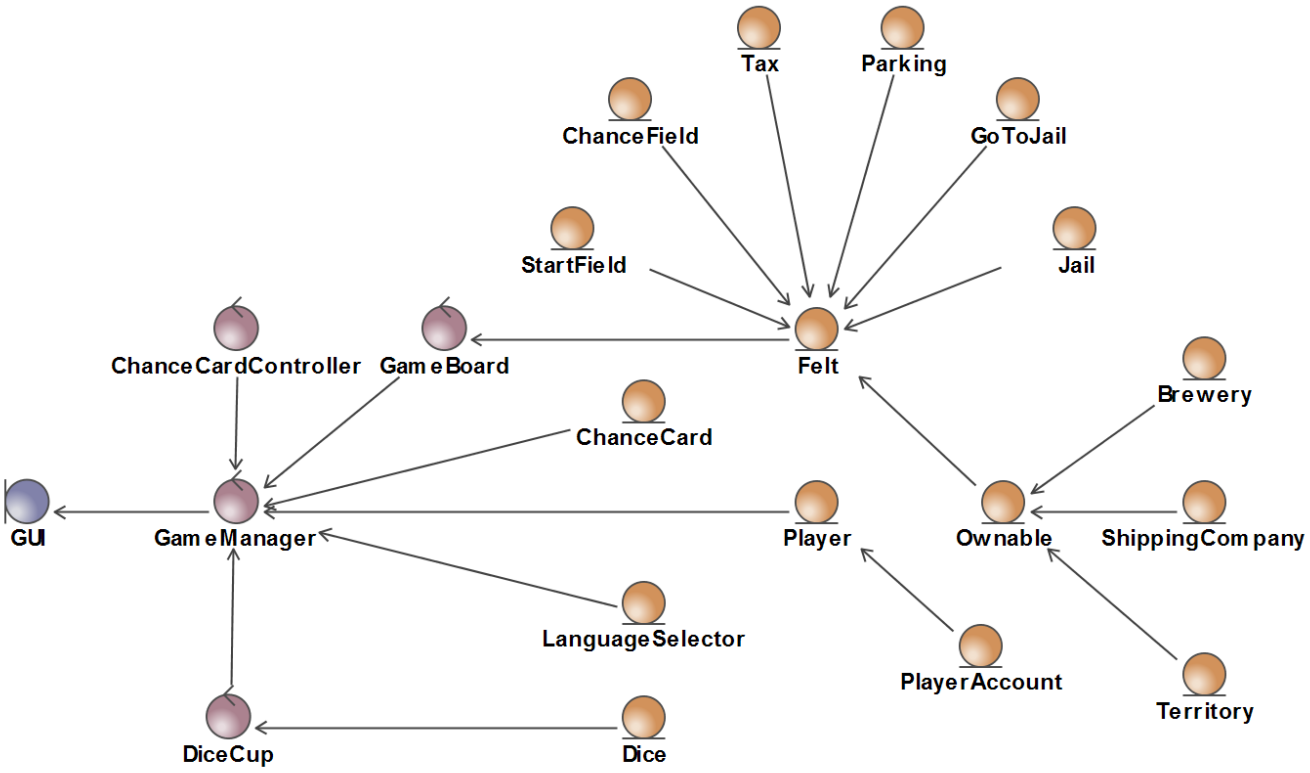
Domænemodel

Vores domæne model er en model af virkeligheden, som vi kan vise vores kunde, så de får en ide om, hvilke elementer, som vi har valgt at have med i vores programmering. Dette er kun byggesten, og klasserne kan have ændret navne, og der kan være tilføjet flere i det endelige program.



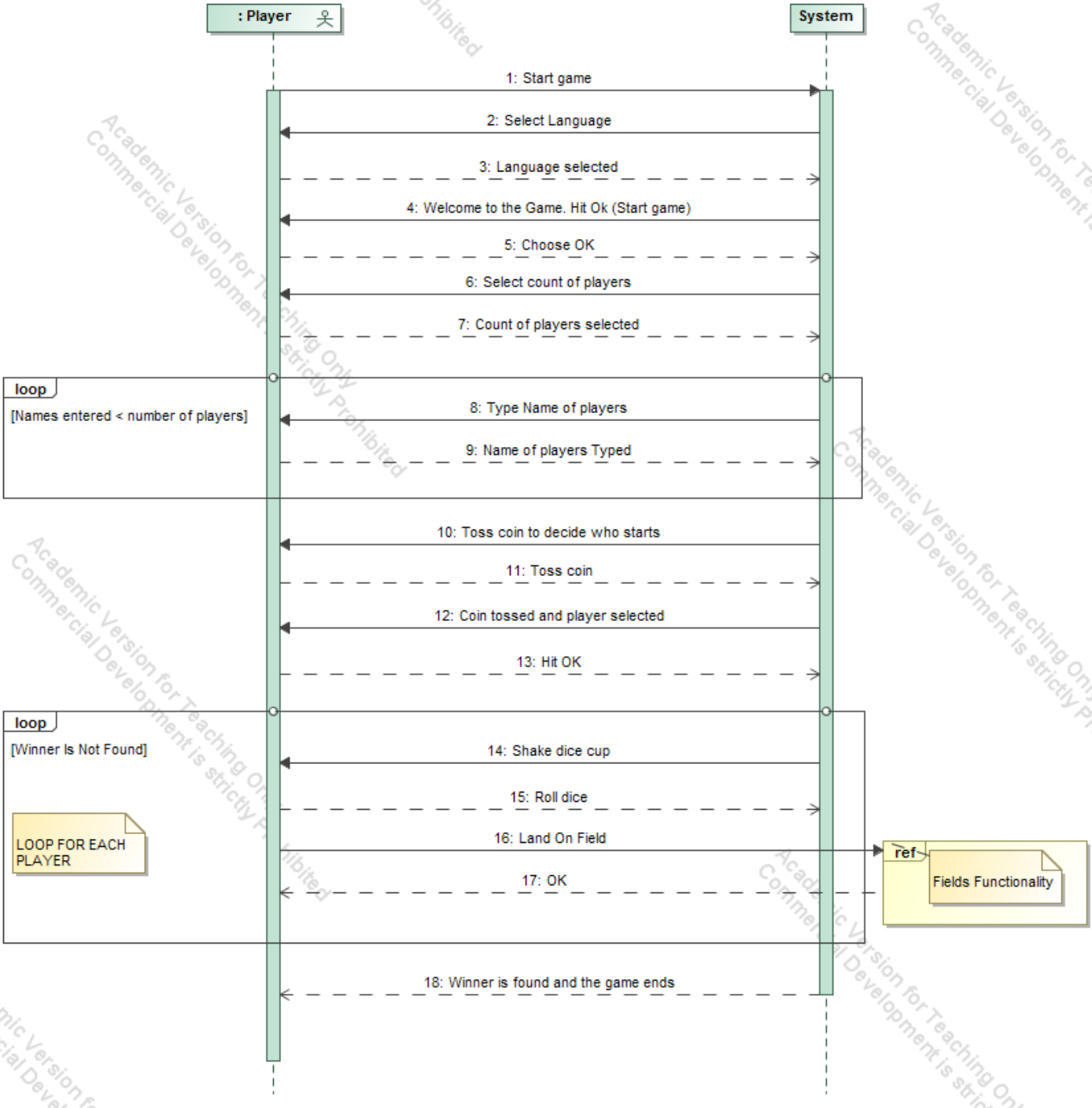
BCE model

Vores BCE model, viser hvilket ansvar vores forskellige klasser har. Vores Boundary klasse "GUI" styrer udseendet, og står for alt interaktion med spilleren. Vores Controller klasser styrer spillet og sørger for at hente de relevante informationer og metoder fra entity klasserne. Controller klassernes funktion er derfor, at sørge for at de forskellige klasser kan kommunikerer med hinanden, selvom der ikke er direkte relation mellem dem. I vores spil er der mere end en controller klasse til at styre spillets flow, da vi har forsøgt at overholde GRASP principperne så meget som muligt.



System-sekvensdiagram

Dette diagram viser hvordan brugeren og systemet kommunikere igennem selve spillet.
Brugeren bliver mødt at af muligheden for at vælge hvilket sprog han/hun ønsker. Dernæst beder systemet brugeren om at angive hvor mange spiller der skal være med (2-6 spiller kan vælges). Brugeren skal efterfølgende skrive navnene på det antal spillere som skal spille og det foregår i et loop, kører indtil at alle spillere har fået tildelt et navn.
Så kaster brugeren en mønt for at vælge hvem der starter. Systemet svarer så at mønten er kastet og at en tilfældig spiller er valgt og spilleren trykker ok for starte. Så ruller brugeren med terningerne og dertil svarer systemet ved at vise terninger på skærmen. Så bliver brugerens brik rykket og lander dernæst på det felt, der svarer til det antal øjne der er rullet. Efterfølgende, afhængig af hvilket felt man er landet på, får brugerne en række muligheder. Der vil være en mulighed for købe det pågældende felt han/hun er landet på, hvis dette ikke er ejet, betale skat hvis man lander på et tax felt, betale husleje, hvis feltet allerede er lejet, ryge i fængsel eller at trække et chancekort, der vil udløse en eller anden effekt. Spillet slutter, når alle brugere undtagen en er gået bankerot.


: Player
System
1: Start game
2: Select Language
3: Language selected
4: Welcome to the Game. Hit Ok (Start game)
5: Choose OK
6: Select count of players
7: Count of players selected
loop
[Names entered < number of players]
8: Type Name of players
9: Name of players Typed
10: Toss coin to decide who starts
11: Toss coin
12: Coin tossed and player selected
13: Hit OK
loop
[Winner Is Not Found]
14: Shake dice cup
15: Roll dice
LOOP FOR EACH PLAYER
16: Land On Field
ref
Fields Functionality
17: OK
18: Winner is found and the game ends

Design

Det kan være en rigtig god idé at udarbejde nogle diagrammer i forbindelse med udviklingen af et system. Diagrammer er med til at overskueliggøre hvordan systemet fungerer i forskellige situationer og det kan være en god måde at visualiserer opbygningen af systemet på overfor kunden. På baggrund af dette har vi udarbejdet nogle diagrammer som giver et overblik over hvordan vores system er sat sammen samt hvilke funktioner der indgår. I designfasen har vi udarbejdet et design-klassediagram og et design-sekvensdiagram, som løbende er blevet ændret i takt med udviklingen af spillet.

Grasp

Vi har forsøgt at overholde nogle GRASP principper, men det har ikke været vores første prioritet. Derfor vil det også fremgå af vores kode af vi desværre har høj kobling for eksempel.

Creator:

Creator pattern er overholdt i form af, at hver enkelt konstruktør selv opretter de objekter som skal bruges. På billedet kan det ses, at vores GameBoard selv sørger for at at køre de tre metoder samt gemme dem i nogle variabler og til sidst sætter den felterne op på GUI'en.

```
public GameBoard(DiceCup cup) {
    guiFields = createGUIFields();;
    diceCup = cup;
    initFields();
    logicFields = createLogicFields();
    GUI.create(guiFields);

}
```

Høj binding:

Vi har opnået forholdsvis høj binding i form af, at vores program er delt i mange klasser, der hver især håndterer hver deres del af vores spil. Altså, PlayerAccount håndterer udelukkende en spillers penge- og værdibeholdning, mens Player håndterer resten der har noget at gøre med en spiller.

Lav kobling:

Nogle steder er har vi opnået lav kobling, dog er der også mange steder hvor koblingen ikke er helt så lav som man kunne ønske sig. Dog er vi også udmærket klar over, at hvis vi fjerner alle vores koblinger vil vi få et rigtig dårligt produkt og, at det gælder om at finde en god mellemvej mellem at have ligegyldige koblinger og slet ikke have nogen. Vi mener, at vi har et okay antal koblinger, dog er det lidt til den høje side.

Information Expert:

Vi har sørget for at give ansvaret for forskellige handlinger til de klasser der har mest data/information til rådighed. Et eksempel kunne være, at Player klassen håndterer alt der har med vores spillere at gøre udover, at pengebeholdningen bliver gemt i PlayerAccount.

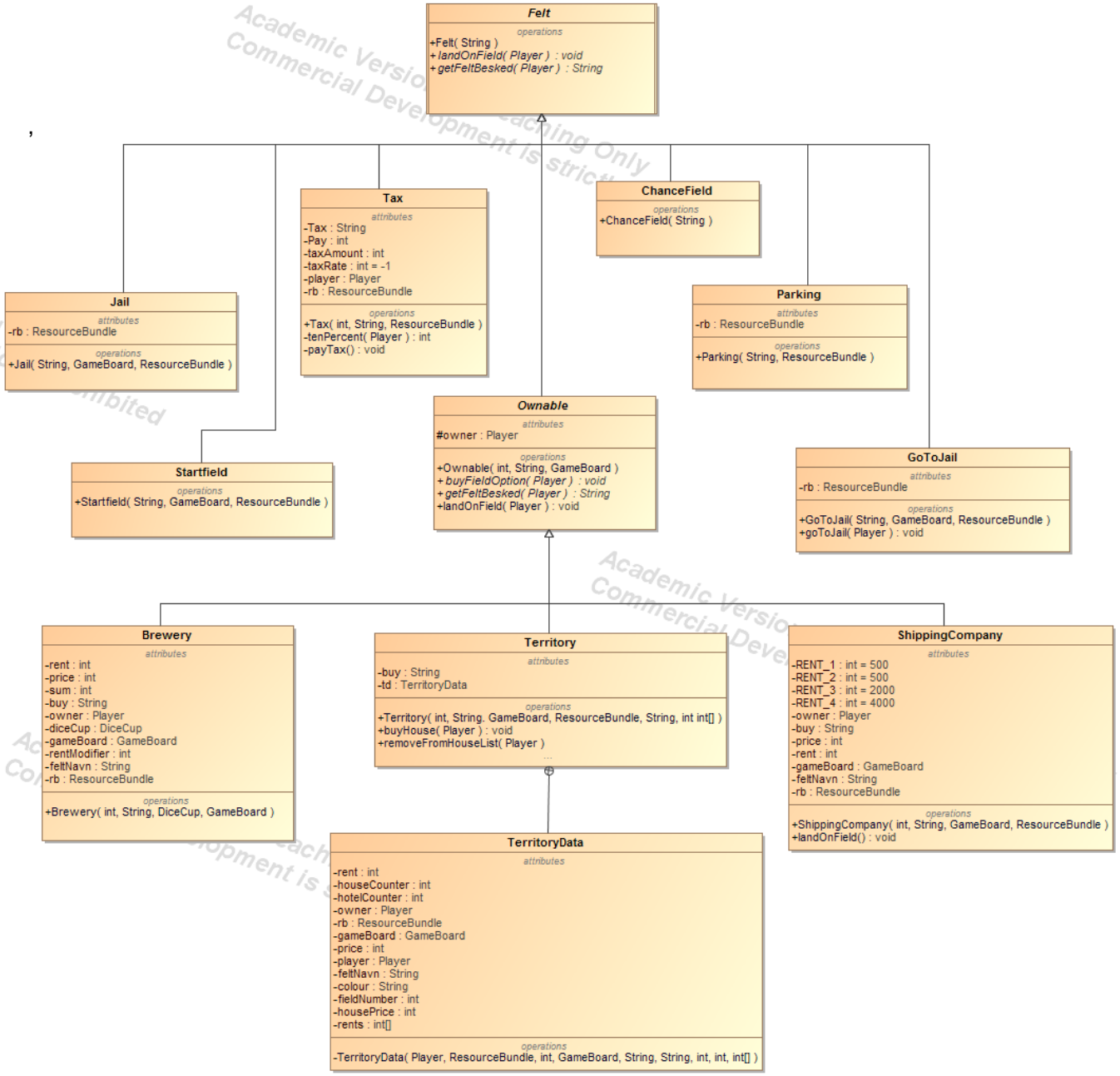
Controller:

Da vi ikke nåede at dele vores kode ordentligt op i controller- og dataklasser, har vi ikke helt levet op til det her pattern. Vi kunne også godt have delt vores GameManager controller en anelse op, da den endte med at blive en anelse stor.

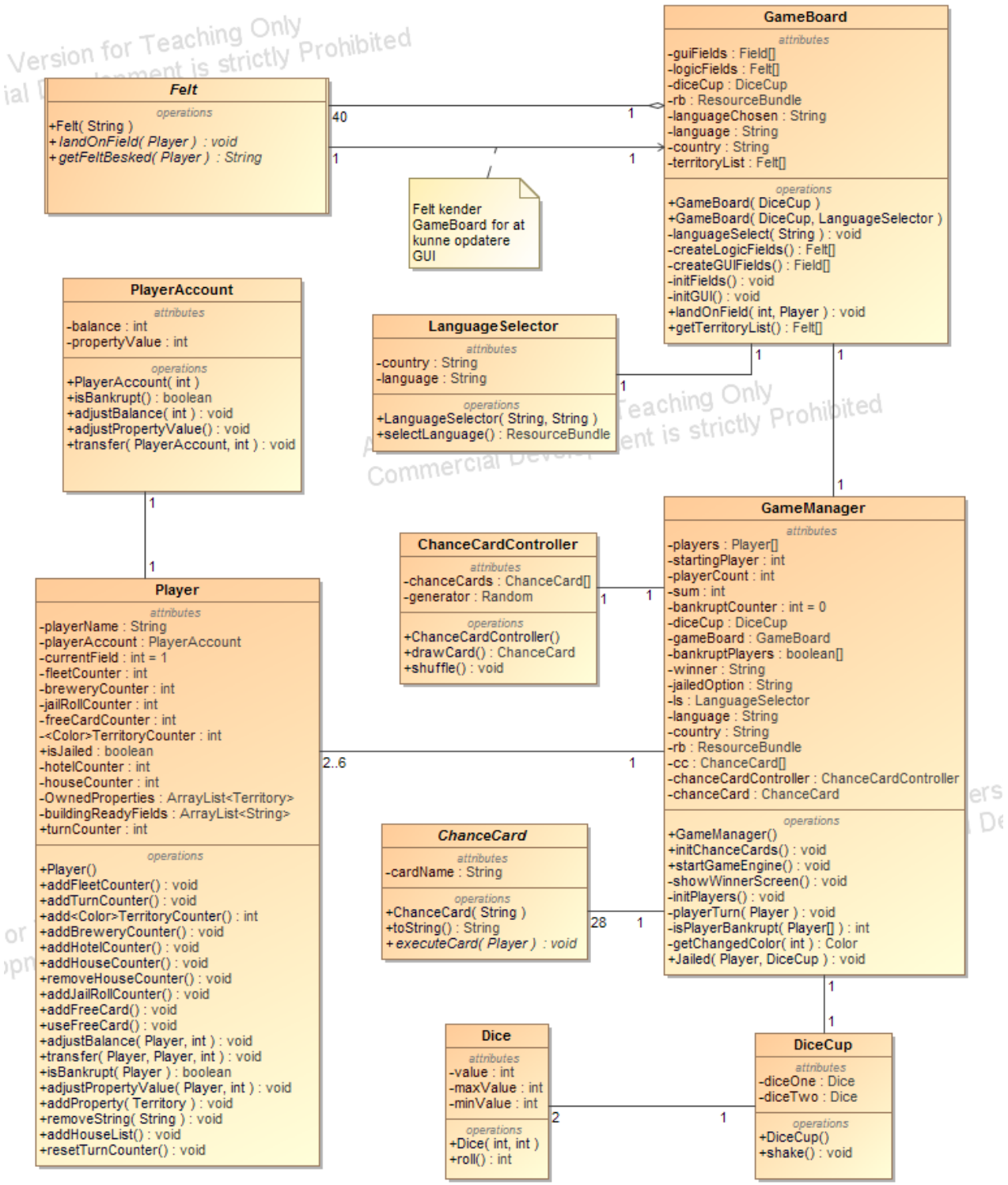
Design-klassediagram

Et design-klassediagram består af de forskellige klasser i vores system, hvori at alle attributter og metoder er vist. Det viser ligeledes forholdet klasserne imellem. Vores diagram er opdelt i 3 dele, for overskuelighedens skyld, da det ellers ville være et meget stort diagram. Det første viser vores Felt klasse, og de tilhørende felt klasser der nedarver fra denne. I disse er deres mest vigtige metoder og attributter vist, men vi har dog valgt at undlade getter og setter metoderne, da disse er mindre vigtige. Som det ses i vores Territory klasse, indeholder denne faktisk en klasse, som man også kan referere til som en indre klasse. Dette er lavet, fordi Territory klassen er en controller i sig selv, og der er derfor lavet en indre klasse til at håndtere alt informationen. Dette er gjort med henblik på at overholde GRASP principperne.

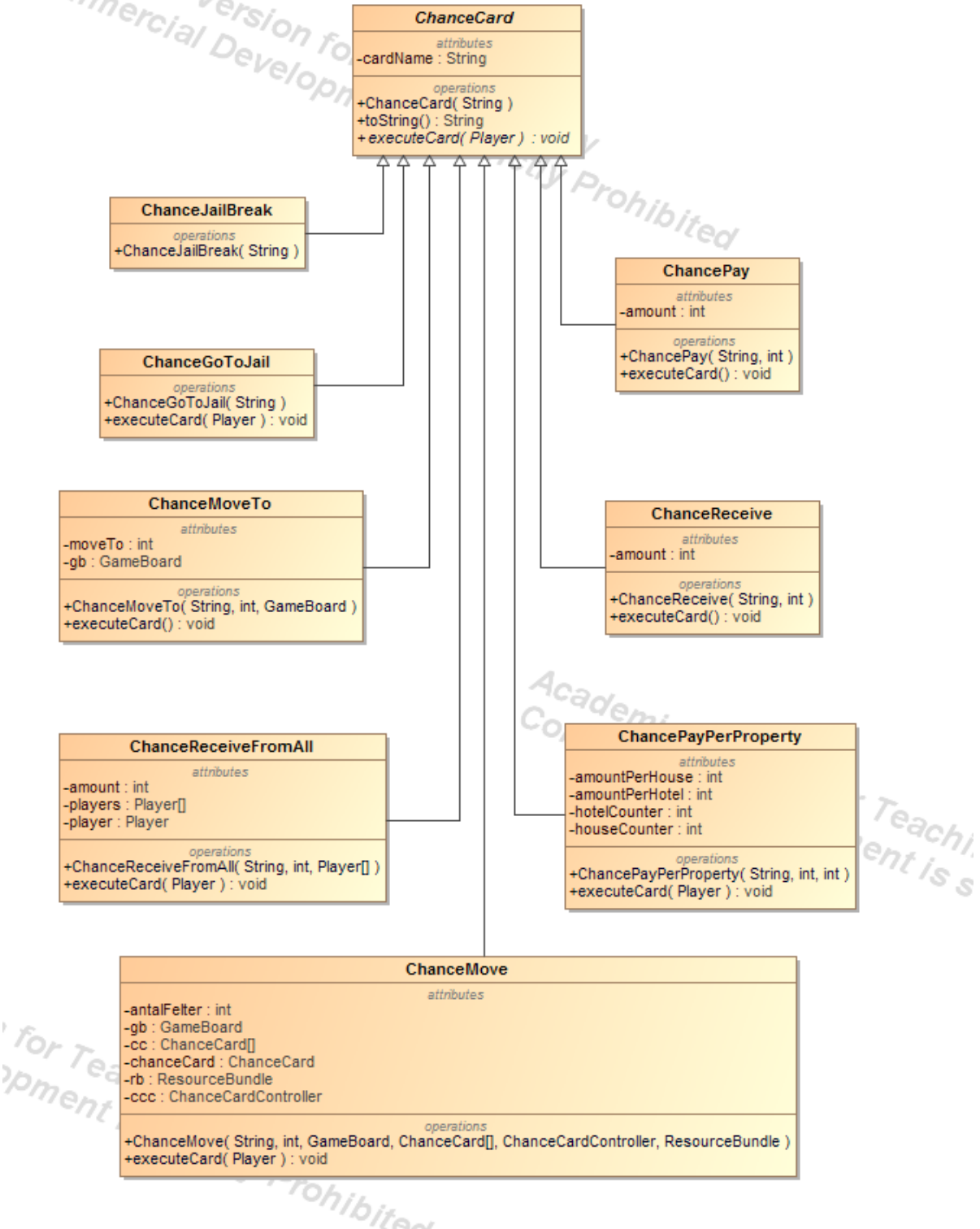
,



I denne del af diagrammet ser vi videre fra Felt klassen, og hvor den rækker hen. I dette diagram er vores 4 hoved controllere også vist, nemlig GameBoard, GameManager, DiceCup og ChanceCardController. Disse klasser sørger for at hente og bruge relevante informationer fra andre klasser. Som eksempel vil ChanceCardController klassen hente den relevante information vedr. chance kortene i klassen ChanceCard, som vil blive beskrevet nærmere i det sidste diagram.



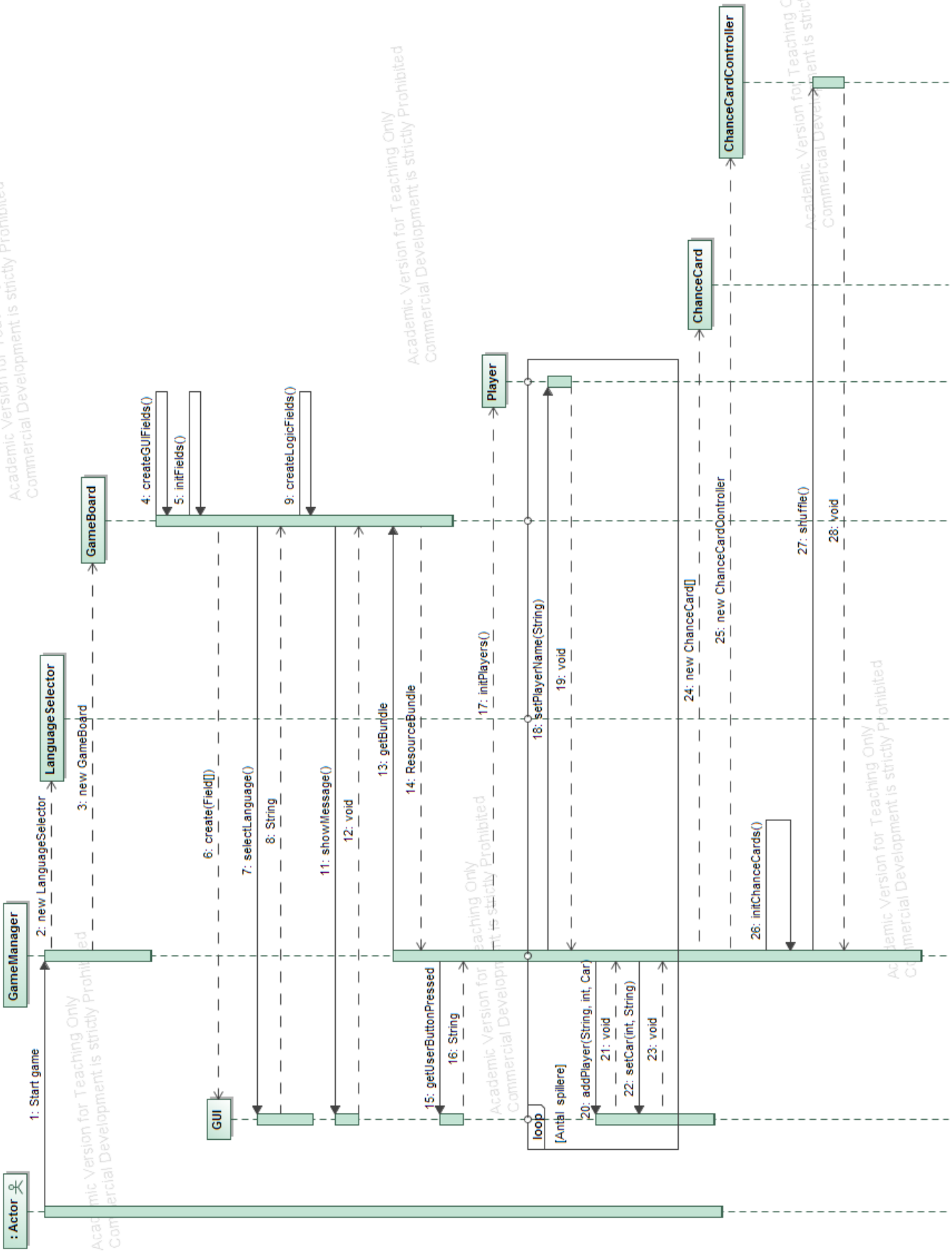
Den sidste del af vores samlede diagram viser den abstrakte klasse ChanceCard, og de nedarvede klasser som indeholder information om effekten af de forskellige chancekort. Der er 7 typer af kort, som er blevet delt op i hver sin klasse. Hver klasse indeholder derfor oplysninger om hvad der sker, hvis du trækker et chancekort af den type. Disse indeholder ligeledes deres interne attributter og metoder.



Design-sekvensdiagram

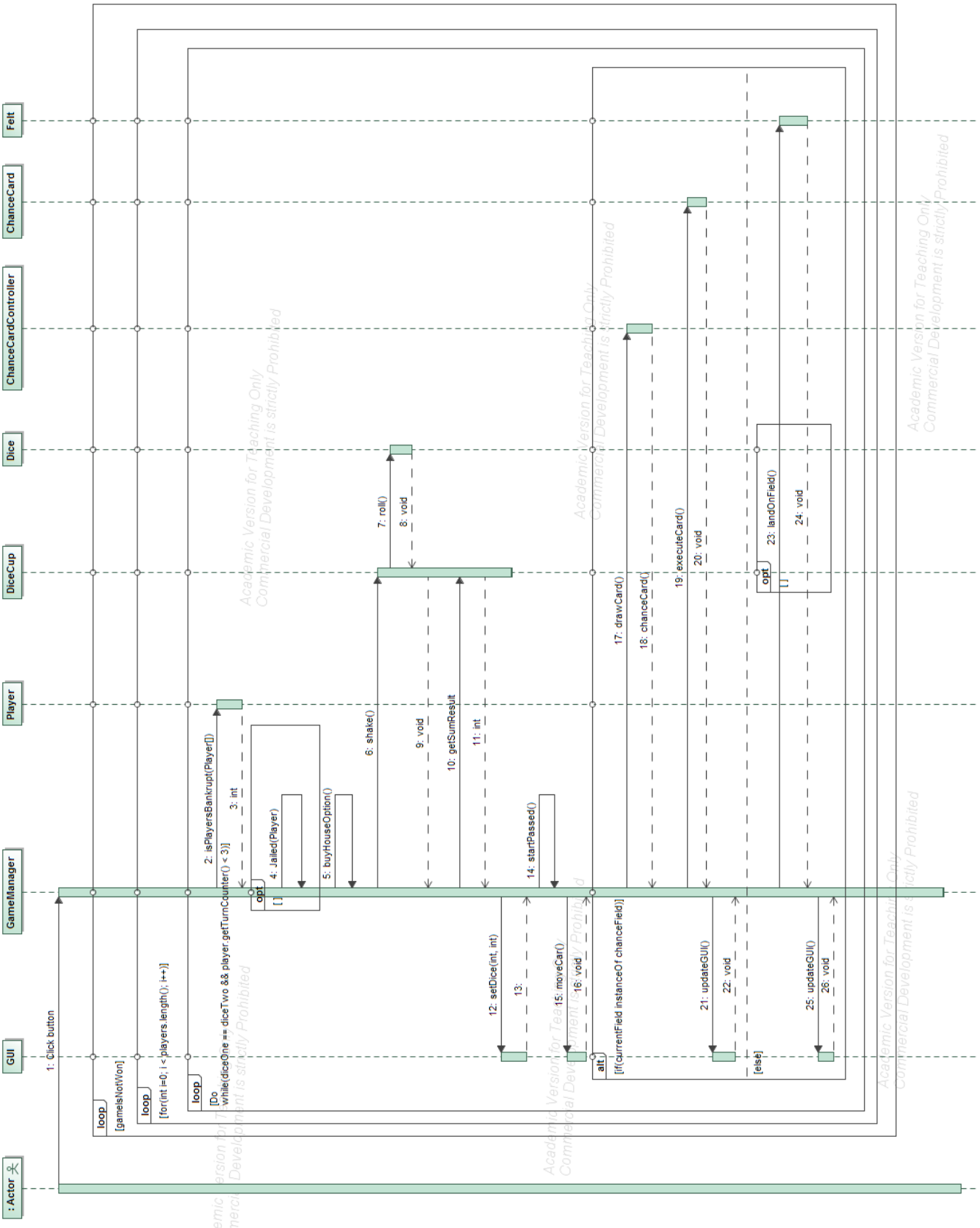
Del 1

Dette er vores system-sekvensdiagram, og ligesom med design-klassediagrammet, er dette også delt op for overskuelighedens skyld. Vi er endt ud med 4 diagrammer, 2 som beskriver hvordan og hvornår metoderne i vores spil bliver udført, og 2 fragmenter som vil beskrive hhv. Jailed metoden og landOnField metoden.
Det første diagram viser, hvordan alle elementerne i vores spil bliver oprettet. Først bliver der oprettet en LanguageSelector, og derefter et GameBoard. Så bliver felterne oprettet og initialiseret, og derefter oprettes der et Field[] som bliver vist på GUI'en. Dernæst bliver der vist en besked i GUI'en der gør det muligt at vælge sprog. Så bliver spillerne inititialiseret og derefter får de i et loop oprettet Navne, og der bliver tilføjet et spillernavn og en brik til GUI'en, indtil at alle spillere er klar. Dernæst bliver der oprettet et ChanceCard[] der indeholder alle vores chancekort, som så bliver blandet og er klar til brug.


: Actor
GameManager
LanguageSelector
GameBoard
GUI
Player
ChanceCard
ChanceCardController
1: Start game
2: new LanguageSelector
3: new GameBoard
4: createGUIFields()
5: initFields()
6: create(Field[])
7: selectLanguage()
8: String
9: createLogicFields()
11: showMessage()
12: void
13: getBundle
14: ResourceBundle
15: getUserButtonPressed
16: String
17: initPlayers()
18: setPlayerName(String)
19: void
loop
[Antal spillere]
20: addPlayer(String, int, Car)
21: void
22: setCar(int, String)
23: void
24: new ChanceCard[]
25: new ChanceCardController
26: initChanceCards()
27: shuffle()
28: void

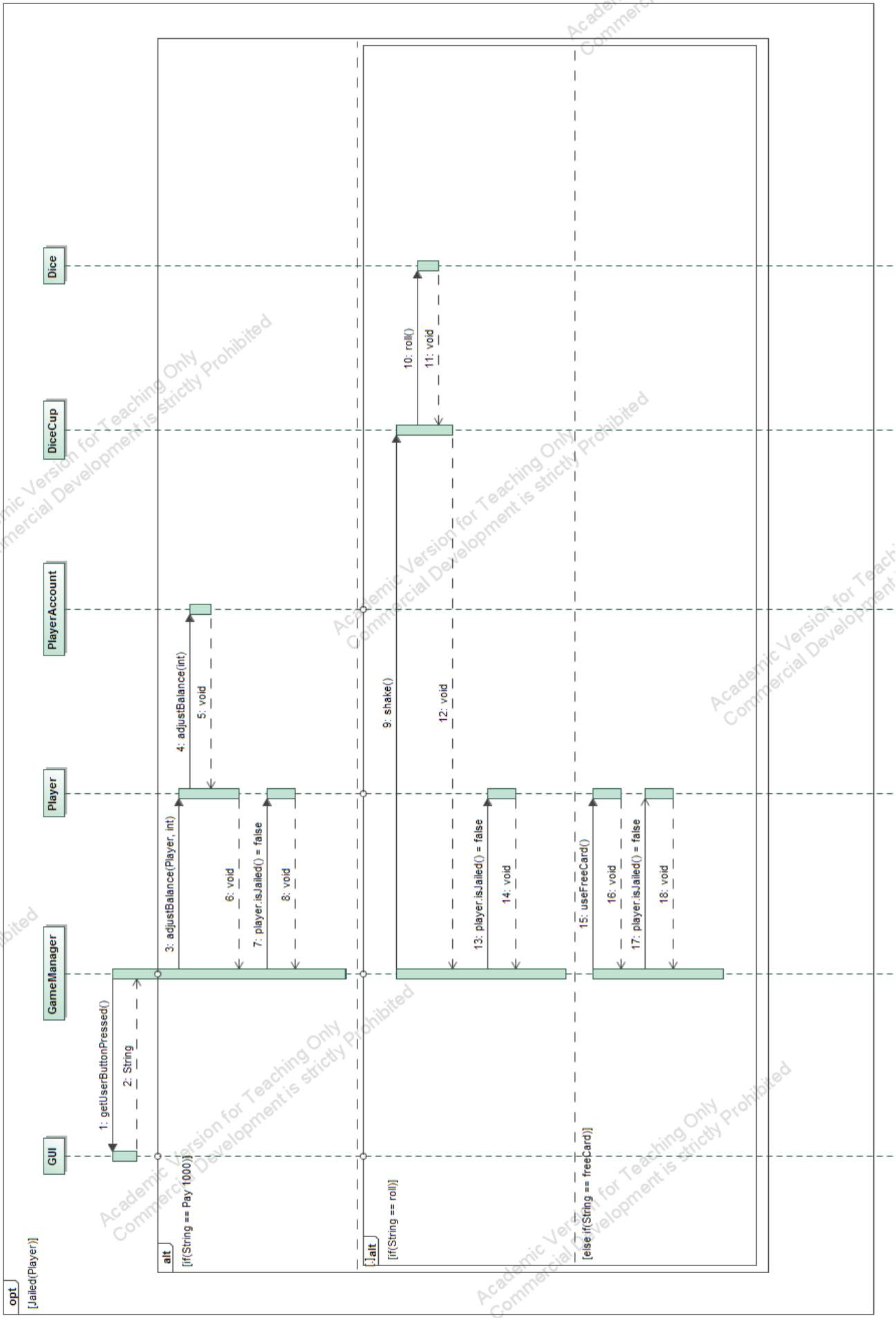
Del 2

Det andet diagram viser, hvordan selve spillet køre, når alt er oprettet. Spillet bliver kørt i et loop, som fortsætter indtil der er fundet en vinder. Derudover er der et forloop er kører hver enkelt spillers tur igennem og vores sidste do-while loop sikrer os, at en spiller kan få op til tre ture såfremt han bliver ved med, at slå to ens. Inde i det sidste loop starter selve spil-logikken for hver spiller. Det første kald, isPlayersBankrupt(Player[]) bruges til at se på hvor mange spillere der er bankrupt. Hvis det er == playerCount - 1 betyder det, at vi har fundet en vinder. Derefter har vi vores jail-logik, som er beskrevet i Del 3. Resten af koden er ret lige til. Hvis man ejer alle grunde med samme farve vil man få muligheden for at købe huse derpå, ellers må man slå med terningerne og rykke det antal felter frem som man slår. Når man lander på et felt bliver der lige tjekket om man er landet på ChanceField eller på et vilkårligt andet felt. Hvis det er ChanceField skal man trække et kort som derefter vil blive eksekveret og GUI'en bliver herefter opdateret. Hvis man derimod lander på et vilkårligt andet felt vil vores landOnField logik træde ind, dette kan ses i Del4, og GUI'en vil så blive opdateret bagefter.



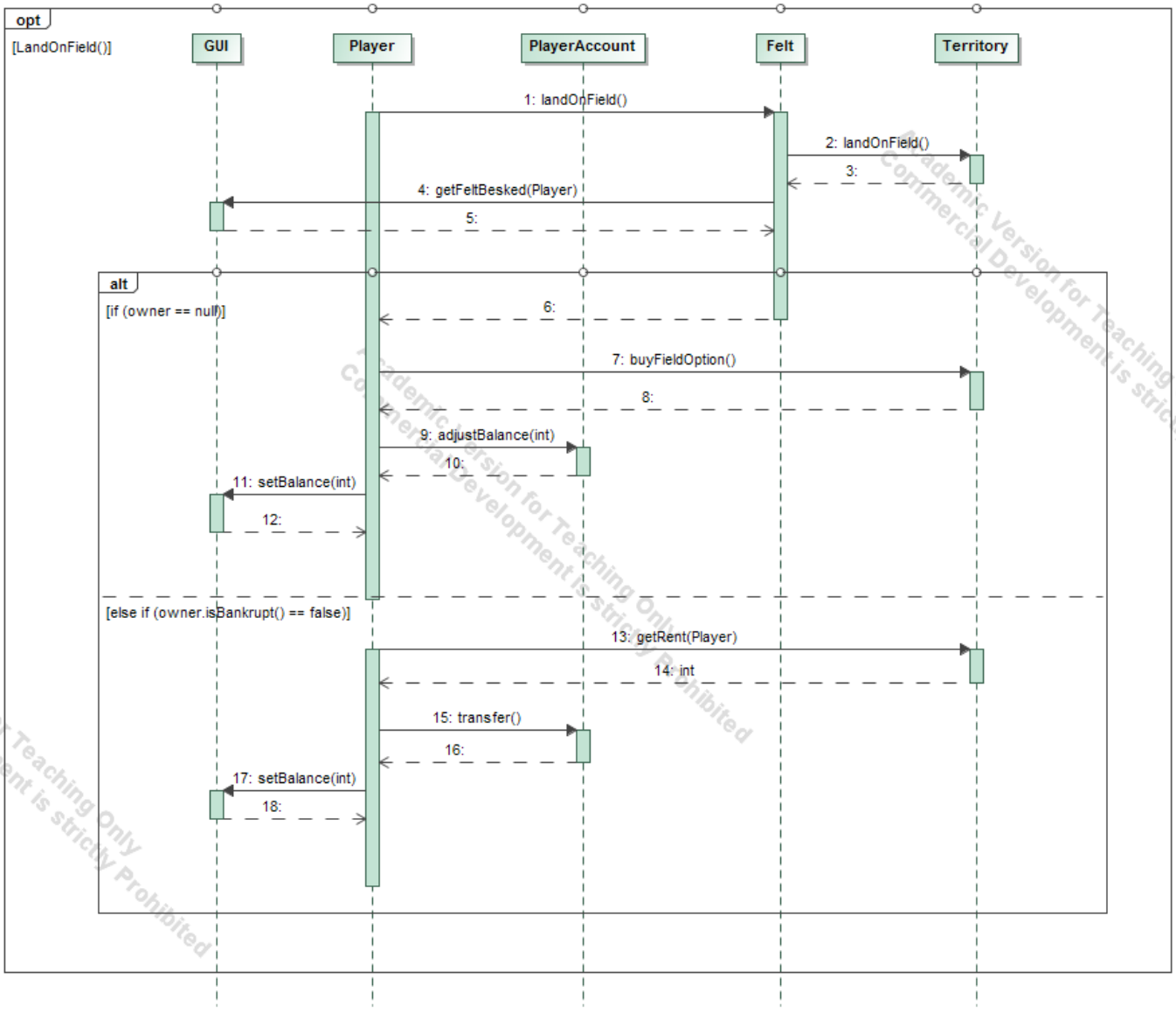
Del3

Dette diagram er et fragment af metoden Jailed, der står for alt hvad der sker mht. til hvis en spiller er blevet fængslet. Spilleren har i alt 3 muligheder for at slippe ud af fængslet, men bliver kun mødt af 3 muligheder på skærmen, hvis han/hun er i besiddelse af et frikendelsekort, som man har mulighed for at få igennem chancekortene. I den første if sætning, kan spilleren vælge at betale en kaution på 1000, hvorefter at hans "status" i fængslet vil blive ændret til false, og kommer derfor ud. I den næste har spilleren mulighed for, at prøve at slå sig ud, ved at slå 2 ens. Lykkes dette ikke første gang forbliver spilleren i fængsel, og får 2 chancer til, ved hans/hendes efterfølgende ture. Det sidste else statement giver mulighed for at benytte sit frikendelsekortet, og gøres der brug af dette, vil man blive løsladt uden at ens pengebeholdning bliver berørt.


opt
[Jailed(Player)]
GUI
GameManager
Player
PlayerAccount
DiceCup
Dice
1: getUserButtonPressed()
2: String
alt
[if(String == Pay 1000)]
3: adjustBalance(Player, int)
4: adjustBalance(int)
5: void
6: void
7: player.isJailed() = false
8: void
alt
[if(String == roll)]
9: shake()
10: roll()
11: void
12: void
13: player.isJailed() = false
14: void
[else if(String == freeCard)]
15: useFreeCard()
16: void
17: player.isJailed() = false
18: void

Del4

Dette er diagrammet over vores landOnField metode. Det beskriver de handlinger der sker, når man lander på et felt, der kan købes, dvs alle vores territory felter. Diagrammet viser, at man lander på et felt, og derefter modtager en besked fra det pågældende felt. Derefter, hvis der ikke er nogen ejer af feltet, får man mulighed for at købe dette felt, hvorefter din pengebeholdning bliver justeret i forhold til hvad feltet koster. I det sidste else statement, kigger den på, om ejeren af feltet er bankerot, og hvis dette ikke er tilfældet, skal den spiller der lander på hans/hendes felt betaler en husleje til ejeren, som så bliver overført fra spiller til spiller.



Implementering

Se bilag for vejledning til installation og systemkrav samt feltliste.

Generelt

Til vores implementering har vi brugt Github, hvor vi har lavet et repository, så alle gruppemedlemmer har kunne arbejde på det samme projekt. Dette gør det lettere at kunne gå tilbage i kodningen, hvis der opstår en fejl, og det gør det lettere at holde styr på, hvad vi hver især har lavet på projektet. Vi har i vores kode prøvet at sætte det op, på den mest overskuelige måde, og har herefter kommenteret stort set alt i de forskellige klasser, så det er nemmere at forstå vores kode.

GUI

Der er blevet gjort nogle enkelte rettelser i GUI'en, som vi mente ville gøre brugeroplevelsen af vores brætspil endnu bedre! Det har også givet spillet et mere flydende flow, i form af at det nu er muligt at benytte Enter-knappen til at trykke på knapperne i spillet, således at man ikke længere er nødsaget til at bruge musen til at trykke hver gang man skal videre i spillet. Dertil er der også også lavet en rettelse der gør, at spillet nu åbner op i midten af skærmen i stedet for i venstre øverste hjørne samt at titel-teksten i felterne er blevet ændret til en størrelse, som felterne kunne indeholde.

Manglende implementation

Vi har i vores implementering valgt at prioritere, hvilke dele af matador spillet, som vi ville have med i vores spil. Vi har valgt at prioritere, da vi hellere vil have et ikke fuldendt spil, som fungere fremfor et spil, som har alle elementer, men ikke alle fungere optimalt.
I vores spil har vi valgt at nedprioritere, pantsætning, at kunne sælge huse/hoteller og så at man skal have et hus på hver grund med samme farve, før man kan bygge nummer to på en af dem. Vi har fået implementeret 28 chancekort, så der mangler vi også nogle stykker, men igen var det nedprioriteret. Derudover nåede vi desværre hellere ikke at dele vores felt-klasser op i controller og data klasser. Det er kun Territory klassen vi har fået delt op, hvor TerritoryData er en private klasse i Territory. Meningen var, at alle klasserne skulle deles sådan op, så vi ville have alt data gemt i disse klasser og al funktionalitet i controller delen.

Vi har også delvist løst det med at man maks kan købe hotel, da muligheden for at købe mere på dem forsvinder, når man har købt et hotel, dog kommer muligheden tilbage hvis man køber et nyt felt, efter muligheden oprindeligt var slettet fra vores drop down menu. Vi valgte at lade det forblive sådan her, da vi prioriteret dette lavere end andre dele af programmet, da et "normalt" spil matador sjældent kommer ud for at købe et nyt felt, efter de har fået et hotel på alle felter af en farve.

ChanceCard:

En af de prøv-lykken kort vi mangler at få implementeret, har vi skrevet koden til som også bør virke. Dog stødte vi på en udfordring i form af NullPointerException, som vi desværre ikke havde tid til at løse. Derfor valgte vi, at udelade det kort, men at beholde koden så man stadig kan se hvordan vi havde lavet det.
Et af de andre kort vi ikke fik lavet (Matador legat), ville ikke have været synderligt svær, at få implementeret da vi allerede har lavet alt forarbejdet til det. Her tænker vi på, at vi allerede har både en spillers pengebeholdning samt en spillers ejendomsværdier. For at få kortet til at virke skal man blot se på om de to værdier lagt sammen er under 15.000 og hvis de er det modtager spilleren 40.000 til sin pengebeholdning.
Vi valgte dog at nedprioritere dette. på trods af at sværhedsgraden ikke var ret høj, for at få helt styr på andre aspekter af vores spil der stadig manglede lidt finpudsning.

Dele af kode

playerTurn(): (Linje 177-321 i GameManager)

Denne metode er en lang og uoverskuelig del af vores spil logik, som foregår i vores GameManager, men pga. tidsmangel nåede vi ikke at dele den op i metoder og derved gøre den mere overskuelig, så vi har valgt at kommentere hele metoden nedenunder, for at kunne give et bedre indblik i hvad vi har foretaget os.

Do/while: (Linje 181-307 og 312-320 i GameManager)

Vi starter med en do while, som køre en spillers tur en gang, og hvis der bliver slået 2 ens, så tæller den én op på getTurnCounter() og hvis det sker 3 gange, så bliver en spiller fængslet.

isPlayersBankrupt(): (Linje 183 i GameManager)

Så tjekker vi i hver spillers tur, om hvor mange spillere der er bankerot og lægger det på vores bankruptCounter.

if(player.getPlayerAccount().isBankrupt() == false): (Linje 185-298 i GameManager)

Herefter køre vi et langt if statement, som starter spillerens tur hvis denne ikke er gået bankerot.

Jailed(player): (Linje 188 i GameManager)

Hvis spilleren så er Jailed, kommer der nogle muligheder som er beskrevet nærmere i denne metode.

if(!player.isJailed): (Linje 191-297 i GameManager)

I dette if statement kører alt vores spillogik omkring en spillers tur, og hvordan spilleren kan købe huse/hotel på sine grunde. Dog tjekker selve if statementet for at se hvor vidt en spiller sidder i fængsel, hvis han gør det skal den efterfølgende kode springes over da den så er irrelevant.

while(loop): (Linje 196-261 i GameManager)

Dette while loop kører så længe at vores boolean loop er true. I dette loop har vi en if/else, som giver en forskellige muligheder i starten af sin tur, alt efter om man ejer alle felter af en farvegruppe eller ikke gør.

If(player.getTerritotyColourCounter == maxAntalColour) / else: (Linje 201-253 i GameManager)

Hvis man ejer alle felter af en farvekode for man muligheden for at slå med terningerne, og købe et hus/hotel.
Hvis man i stedet vælger at slå med terningerne, så sætter den vores boolean loop til false, og bryder ud af loopet.
Men hvis man vælger at købe et hus eller hotel, så fortsætter den i loopet og giver en muligheden for at købe hus.
Hvis man ikke ejer alle felter af en farvekode, så giver den kun en muligheden for at slå med terningerne og sætter vores boolean loop til false, og bryder ud af loopet.

(Linje 263-265 i GameManager)

I disse linjer, slår vi med terningerne, og får det vist på GUI'en.

If(player.getCurrentField() > (player.getCurrentField()+sum)%40): (Linje 267-272 i GameManager)

I denne if sætning tjekker den om man er gået over start, og hvis man er det, bliver der added 4000 til den pågældende spillers konto.

(Linje 274-279 i GameManager) :

De første 3 linjer kode flytter brikken rundt på brættet. Den 5 linje kalder landOnField() metoden for det felt, som man lander på.

If (currentField instanceof ChanceField)/else : (Linje 283-293 i GameManager)

Denne if/else sætning ser på om det er et chance felt man lander på. Hvis det er det, så trækker man et kort, som bliver vist på midten af skærmen, og derefter bliver beskeden udført af spilleren.
Men hvis det ikke er et chance felt, for man landOnField metoden for den pågældende felttype.

if(player.getPlayerAccount().isBankrupt()) : (Linje 295-296 i GameManager)

Denne if sætning fjerner alle biler, for spillere, som er gået bankerot.

Test

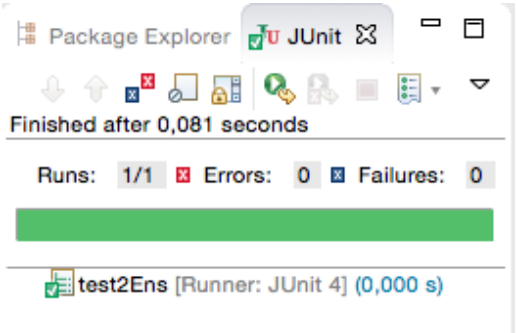
Generelt

Vi har valgt at lave de test som vi gør, for at teste de nye funktioner, som vi har implementeret i dette spil, frem for i det gamle CDIO3. I det gamle spil, testede vi de forskellige felt type klasser, om de fungere korrekt, og det gjorde de. Vi har næsten bare kopieret disse klasser, og deres funktioner, så vi valgte at lade være med at lave de tests igen, og lægge mere fokus på nogle af de nye funktioner, som vi syntes var mere interessante at teste.

JUnit tests

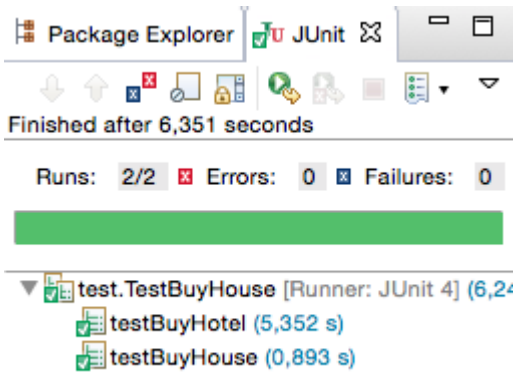
Fælles for vores sidste 3 testcases er, at hastigheden på selve testen er utroligt langsom da den skal starte GUI'en op og herefter vente på user inputs. Hvis det ikke var en del af processen ville de ikke tage så mange sekunder, men være overstået på ca. 20-30ms.
Den første test er der ikke GUI kald i, så GUI'en bliver ikke åbnet og derfor er hastigheden på denne test normal.

TestJail



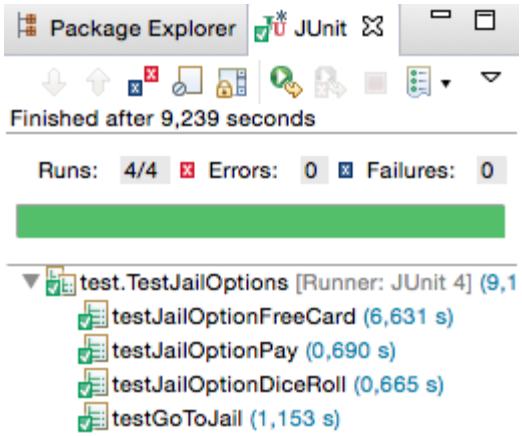
I denne test har vi testet, at hvis man slår 2 ens tre gange i træk, om spilleren så derefter er i fængsel. Denne test fungere optimalt, som det kan ses ud fra resultatet. Testen er sat sådan op, at man først sætter de vigtigste ting til testen op, og derefter tjekker om han starter med at være i fængsel. Det er han ifølge testen ikke, som også var vores forventede resultat. Derefter har vi fået playeren til at slå med terningerne tre gange, og har sat vores diceCup til kun at slå to 1'ere hver gang. Derefter tester vi så, om han er kommet i fængsel, hvilket også passer.

TestBuyHouse



I denne test har vi igen sat testen op ved at lave en player, sætte hans balance og lavet et territory, hvor vi har udfyldt alle parametrene i konstruktøren. Derefter har vi testet, om vi har sat balancen på playeren rigtigt, hvilket vi ifølge testen har. Så satte vi playeren som Owner af territoriet, og bad ham om at købe et hus, hvis pris vi havde sat til 1000. Til sidst testede vi så om playerens balance var faldet med 1000 og om der var kommet et hus på. Vores anden test, viser bare om der er blevet købt hotel, hvilket er 5 huse i vores spil. Alt gik også godt i denne test.

TestJailOptions



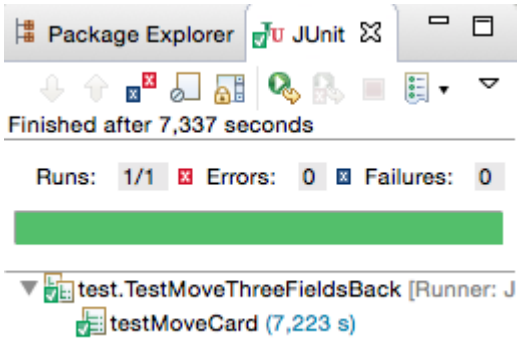
I denne test, har vi testet vores GoToJail felt, og alle de forskellige muligheder man har, for at komme ud af fængslet. De muligheder er, enten at slå to ens, betale 1000 eller bruge sit chancekort, som gør, at man kan komme ud af fængslet.

Vi har igen sat testen op igen, ved at lave en diceCup med to 1'ere, lavet en player, med en balance på 30000 og lavet GoToJail feltet.

Derefter startede vi med at sætte spilleren på feltet GoToJail og så om han kom i fængsel. Derefter testede vi om man kunne betale sig ud, og bagefter se om playerens balance var 29000, hvilket den var og om han stadig var i fængsel.

Næste test var hvis man slog to ens, så ville man komme ud af fængslet. Så vi startede med at sætte ham i fængsel, og derefter få ham til at slå 2 ens, og testede bagefter på om han stadig var i fængsel. Til sidst testede vi om man kan komme ud, ved at bruge sit chancekort. Dette gjorde vi ved at give ham chancekortet og derefter bruge det, og så teste om han stadig var i fængsel og om han stadig havde kortet. Hele testen gik som forventet.

TestMoveThreeFieldsBack



I denne test har vi igen sat den op, ved at lave en player og et chancekort, hvilket gør at man skal rykke 3 felter tilbage. Derefter fik vi playeren til at lande på felt 3, hvilket er et chance felt, så trak han kortet og rykker 3 tilbage, hvilket er til felt 40, som er rådhuspladsen. Vi testede så om playeren stod på rådhuspladsen til sidst, hvilket han gjorde.
Vi valgte denne test, da den går tilbage over start, hvilket i vores Array er ned i minus, derfor synes vi denne test var god at have med.

Konklusion

Vi har fået lavet vores Matador spil, dog ikke en eksakt kopi, da vi ikke havde tid nok til at implementerer alle dele. Man kan læse om de manglende implementationer i rapporten.
I rapporten har vi lavet diagrammer for at få en forståelse af, hvordan vi ville lave vores program og hvordan vi ville formidle vores idéer til programmet videre til andre. I opgaven har vi brugt forskellige metoder, fordelt i vores klasser, som i sidste ende alle bliver kaldt i vores main klasse (Main). Opgavens indhold er blevet lavet ud fra spillereglerne fra Matador, og kan nu bruges som det skal. Næsten alle regler er som sagt blevet implementeret samt nogle fede features som at man kan se hvem som ejer de forskellige felter, et coin toss i starten af spillet der bestemmer hvem som starter, i stedet for det altid er spiller 1 og en sprogvælger, så man kan vælge imellem dansk og engelsk når man spiller spillet.
Spillet er til sidst blevet testet med diverse tests, som blandt andet undersøger om en spiller kan købe et huse eller hoteller, om man kan ryge i fængsel, om de forskellige muligheder i fængsel virker og så om et af chance kortene virker.

Bilag

Git

For at få fat i vores projekt, kan man importere det direkte til Eclipse vha. linket https://github.com/Chaaaaap/CDIOFinal . Dette gøres ved at trykke File → Import → Git → Projects som Git → Clone URI → Indtaster linket i URI-feltet → Next → Finish. Når dette er gjort henter Eclipse selv hele projektet ned. Når først man har hentet projektet ned til Eclipse, skal man blot åbne klassen "main" og køre denne, for at starte spillet.
Man har dog også mulighed for at hente projektet ned som en .ZIP fil, der kan importeres i Eclipse som et eksisterende projekt. Dette gøres ved at trykke File → Import → General → Existing projects into workspace.

Systemkrav

Vores spil kan køres på alle maskiner der som minimum kører med Windows XP. Det eneste systemkrav der er, er at man skal have installeret JAVA 1.8, for at kunne bruge det library som vi har anvendt (jre 1.8.0_60). Man kan dog godt bruge en nyere udgave, så skal man blot ind og ændre i, hvilke libraries der er tilknyttet projektet. Det er ikke noget større problem, men hvis man ikke føler sig sikker på hvordan man tilføjer / fjerner libraries vil vi ikke anbefale det.

Feltliste

Der er blevet vedhæftet en feltliste i form af et pdf dokument i zip filen der indeholder vores projekt.

Beskrivelse af chancekortene

Beskrivelse af alle Chancekortene.
Tag med den nærmeste færge - Flyt brikken frem, og hvis du passerer "START" modtag 4.000 kr.
Gå i fængsel. Ryk direkte til fængsel. Selv om du passerer "START", modtag IKKE 4.000 kr.
Du har kørt frem for "Fuldt stop". betal 1.000 kr. i bøde.
Betal din bilforsikring på 1.000 kr.
Tag med DFDS Seaways. Flyt brikken frem, og hvis du passerer "START" modtager du 4.000 kr.
Modtag udbytte af dine aktier, 1.000 kr.
Da det er din fødselsdag modtager du 200 fra hver spiller.
Du har måtte vedtage en parkeringsbøde. Betal kr. 200 kr. i bøde.
Tag ind på Rådshuspladsen.
Deres præmieobligation er kommet ud. De modtager 1.000 kr. af banken.
I anledning af kongens fødselsdag benådes de herved for fængsel. Dette kort kan opbevares, indtil de får brug for det.
Oliepriserne er steget, og De skal betale: 500 kr. pr. hus og 2.000 kr. pr. hotel.
Ryk tre felter tilbage.
Du har vundet i Klasselotteriet. Modtag 500 kr.
Betal 3.000 kr. for reparation af deres bil.
Kommunen har eftergivet et kvartals skat. Hæv i banken 3.000 kr.
Ryk brikken frem til det nærmeste rederi og betal ejeren to gange den leje, han ellers er berettiget til. Hvis selskabet ikke ejes af nogen kan De købe det af Banken.
Ryk frem til Grønningen. Hvis De passerer "START" modtag 4.000 kr.
Ejendomsskatterne er steget, ekstraudgifterne er: 800 kr. pr. hus og 2.300 kr. pr. hotel.
De havde en række med elleve rigtige i tipning. Modtag 1.000 kr.
Ryk frem til 'START'
De modtager 'Matator-legatet for værdig trængende' på 40.000 kr. Ved værdig trængende forstås at deres formue, dvs deres kontante penge + skøder ´bygninger ikke overstiger 15.000 kr.
Værdien af egen avl fra nyttehaven udgør kr. 200, som de modtager af banken.

Du har modtaget deres tandlægeregning. Betal 2.000 kr
Du har været en tur i udlandet og har haft for mange cigaretter med hjem. Betal 200 kr.
Ryk frem til Frederiksberggade. Hvis De passerer 'START' modtag da 4.000 kr.
Grundet dyrtiden har De fået gageforhøjelse. Modtag 1.000 kr.