



INDLEDENDE PROGRAMMERING 02312

3 ugers forløb - Final

Gruppe 22

Andreas Vilholm Vilstrup
s205450



Chenxi Cai
s205420



Alexander Solomon
s201172



Oliver Fiedler
s205423

Isabel Grimmig Jacobsen
s205473



Ahmad Shereef
s173750



https://github.com/Fiedler12/22_final

Afleveret d. 18/01-2021

Resumé

Vi har fået til opgave at lave et Matador spil for spilfirmaet IOOuterActive.

Vi har i samarbejde med kunden, kommet frem til en række krav og visioner, som spillet skal indeholde. Ud fra de fundne krav, har vi analyseret og udviklet et fuld funktionsdygtigt Matador spil.

Alle de kravene vi har fundet frem til ligger i afsnittet Krav, og opdelt alt efter om det er et funktionelt, ikke-funktionelt, unødvendige eller krav der kan blive lavet hvis i videreudvikling af spillet. De ikke-funktionelle, er de krav som er noget man kan se fysik, fx spillepladen. De funktionelle krav er det som man skal kunne gøre i spillet, dvs. købe grunde, betale leje osv. Spillet er opbygget så det fungerer på samme måde som det fysiske Matador spil, hvor der kan være 2-6 spillere. Spillerne skiftes til at slå med to terninger og flytte rundt på spillepladen, som har 40 felter, inklusiv start feltet. Rundt på spillepladen finder man både chancekort med en positiv eller negativ belønning, og felter som repræsenterer ejendomme/grunde, rederier og bryggerier. Felterne med ejendomme/grunde, rederier og bryggerier koster penge, hvis et felt er købt og en ny spiller lander på feltet, vil der blive opkrævet leje (hvis man klikker på et felt kommer lejen frem). Felterne er delt op i forskellige farver, så de passer sammen i en serie, hvis en spiller har en hel serien felter bliver der opkrævet dobbelt leje, når der landes på ét af disse felter. Der er også mulighed for at bygge huse eller hoteller hvis man ejer en serie. Der skal altid bygges ligeligt på grundene i serien.

Spillerne kan også ryge i fængsel, som enten koster penge, et løsladelseskort eller også skal man slå to ens, for at komme ud. Spillet går i sin enkelthed ud på at købe ejendomme, tjene penge og være den sidste spiller tilbage. Man ryger ud af spillet hvis man går fallit og ikke kan betale sin gæld.

I rapport finder man metoder og diagrammer, som oplyser om hvordan selve spillet er opbygget og fungere. Vi har bl.a. lavet et use case diagram, som viser de muligheder som spillerne har igennem spillet. Derudover er der lavet et design klassediagram, som giver et overblik over hvilke attributter og metoder vores klasser har, og hvilke relationer imellem. Vi har også fokuseret på implementeringen af de forskellige dele kode, så det er nemmere at forstå. Vi har blandt andet brugt JUnit til at teste den kode vi har udviklet, og dermed sikre at programmet kan køre uden problemer. Vi har bl.a. teste, om der bliver trukket penge fra de enkelte spiller, på de rigtige tidspunkter. Vi har også testet at der bliver fundet en vinder, når han/hun er den eneste spiller tilbage. For at sikre os en spil som er brugervenligt, har vi valgt at teste spillet, på nogen udefra, som ikke har forstand på at kode. Ud fra diagrammer, metoder, implementering og test af programmeringen/koden, kan vi konkludere at spillet fungere som et fysisk Matador spil.

Derudover kan vi konstatere at spillet kunne blive en mere lig det fysiske spil, hvis der havde været mere tid. Der er lagt op til at kunne finjusteringer og videre arbejde, for at gøre spillet endnu mere perfekt.

Indholdsfortegnelse

Resumé	2
Indledning.....	5
Krav.....	5
<i>Funktionelle krav:</i>	5
<i>Ikke-funktionelle krav:</i>	6
<i>Vurderingen af kunden visioner/krav</i>	6
<i>Karv der kunne implementerets hvis var mere tid.....</i>	6
Analyse.....	6
<i>Use case diagrammer.....</i>	6
Use case Brief beskrivelse:.....	8
Use case Fully Dressed beskrivelse:.....	10
<i>Flowchart:.....</i>	13
<i>Domæne model</i>	14
Design	14
<i>Oversigt over de forskellige klasser i programmet.</i>	14
<i>Uml diagram.....</i>	15
Klassediagram ud fra udgangspunktet:	15
Design klassediagram:	17
Sekvensdiagram:.....	19
System Sekvensdiagram:	21
Implementering	24
<i>ChanceCard</i>	24
Afvigelser	24
<i>Matadorlegatet</i>	24
CardDeck.draw().....	25
Set top card	26
<i>Player.....</i>	27
<i>Setpos:</i>	27
<i>Board controller.....</i>	28
<i>Consol.....</i>	28
Start game	28
Consol.....	29
Play game	29
Turn	30
Check subclass i consol	32
CheckPlayerAccount	33
Player pawn/playersbuyback.....	34
Pull Card /Carddeckswitch.....	34
Trade.....	35

Build.....	36
Sell house.....	36
Test	37
<i>Doublerent/byg</i>	37
<i>Terningtest 1</i>	38
<i>Terningtest 2:</i>	38
<i>Test case scenarier:</i>	40
Test case 1	40
Test case 2	43
Test case 3	47
<i>Code Coverage</i>	54
<i>Brugertest:</i>	56
Konfigurations styring.....	56
<i>En guide til når man modtager filen og vil køre den:</i>	56
Projektplanlægning	57
<i>IntelliJ</i>	57
<i>Maven</i>	57
<i>Github</i>	57
<i>Use case</i>	57
<i>Domain model</i>	57
<i>Flowchart</i>	58
<i>System sekvensdiagram</i>	58
<i>Design klassediagram</i>	58
<i>Sekvensdiagram</i>	58
<i>JUnit</i>	58
<i>Gui</i>	58
Konklusion	59
Forklaringsliste.....	60
Timeregnskab	60
Litteraturliste	61

Indledning

Vi er blevet stillet en opgave af spilfirmaet IOOuterActive, som vi i denne rapport udvikler og dokumentere. Opgave går ud på at lave et Matador spil (til voksne), som kan spilles af 2-6 personer. I den klassiske Matador spil, dyster spillerne om at købe og sælge ejendomme, uden at gå konkurs og dermed tabe spillet, vores spil fungere på samme måde. Vi har valgt, i samarbejde med kunden, at være så tro til det oprindelige Matador brætspil som muligt. Der er dog blevet lavet små justeringer, da det hele foregår elektronisk og der derfor er nogle regler der bliver overflødige/unødvendige. Dette bliver vist i form af selve spillet som er udarbejdet i IntelliJ IDEA og i form af denne rapport, som både indeholder, en liste over de krav der er til spillet, diagrammer og modeller som giver et overblik over hvordan spillet fungere og hvordan det er opbygget.

Krav

Funktionelle krav:

- **Ekstra kast** - hvis antallet af øjne viser det samme på begge terninger, får man en ekstra tur.
- **Man ryger i fængsel efter 3 gange tog ens terningekast i træk** - hvis antallet af øjne (på de to terninger) viser det samme antal, på samme tur, og dette sker 3 gange i træk, ryger man i fængsel.
- **Spillerne har 3 muligheder for at komme ud af fængslet** - 1) betal 1000kr, 2) benyt "løsladelses" chance kortet, eller 3) kast de 2 terninger, hvis de er ens, flytter du ud af fængslet, med det antal øjne der vises. Dette kan gentages i tre omgange, men derefter betales der 1000kr.
- **"Prøv lykken" felt** - lander man på en af disse felter, trækkes det øverste kort fra bunken af Lykke-kort. Derefter udføres handlingen og kortet lægges nederst i bunken.
- **Skibsrederier** - ejes 1 så betales 500 kr. Ejes 2 så betales 1000 kr. 3 så betales 2000 kr. 4 betales 4000 kr.
- **Skøder** - Alle grunder har en værdi, både med og uden huse/hoteller på. Ejér spilleren en grund, vil kanten rundt om grunden blive den farve, som spilleren har bil.
- **Når man er i fængsel, kan man ikke være der i mere end tre runder** - Hvis man ikke slår to af samme slags når man har kastet tredje gang, skal man betale 1.000 kr. Og derefter flytte det antal terningerne viser. Når en spiller er i fængsel, må man gerne købe grunde gennem handel eller auktioner gennem de andre spillere.
- **Fallit** - skylder man mere end man har råd til, skal man overdrage alt til sin kreditor, efter at have solgt eventuelle bygninger til banken. Derefter udgår man af spillet.
- **Handel mellem 2 spiller** - Det er muligt at kunne handle mellem spillere, dvs. hvis spiller 1 vil købe en grund som spiller 2 ejer, vil det være muligt at lave et penge byd. Buddet kan enten blive afvist, accepteret eller modbyde.
- **Pantsæt** - Mangler en spiller penge er det muligt at pantsætte en grund. Den kan sagtens købes tilbage igen men det koster 10% oveni og runder op til næste 100.
- **Handle mellem spiller** - Det er muligt at handle spillerne imellem, dog kun med pengebeløb.

Ikke-funktionelle krav:

- En spilleplade - indeholder 40 felter, hvor iblandt der er 6 chance felter, ét “på besøg i fængsel” felt, ét “de fængsles” felt, 4 “raderinger”, 2 “bryggerier”, et parkerings felt, 2 felter hvor der skal betales penge til banken og 1 Startfelt
- 6 biler - der er mulighed for at spille mellem 2-6 spillere
- To terninger - der slås med begge terninger på samme tid
- Huse og hoteler -
- Skøder -
- Start beløb på 30.000kr - alle spillere starter med det samme beløb

Vurderingen af kunden visioner/krav

- **Pengene man starter med, er fordelt ved: 2*5000kr, 5*2000kr, 7*1000kr, 5*500kr, 4*100kr og 2*50kr** - Det er ikke vigtigt, hvilke pengesedler hver spiller har, computeren holder styr på det samlede antal kroner og uafhængigt af hvilke sedler det er.
- **Bankør** - Der skal ikke vælges en bankør, det er computeren.
- **Glemmer at opkræve leje**: Giver ikke mening når der er flere siddende omkring en computer. Man bør prøve at begrænse hvor ofte computeren skifter hænder for at give en bedre brugeroplevelse. Computeren sørger for at huske at opkræve penge.

Karv der kunne implementerets hvis var mere tid.

- **Bytte grunde** - To spillere ønsker at bytte en grund for en anden grund.
- **Salg ”løsladelses” kort** - kunne sælge chance kortet, som gør at man kan komme ud af fængslet. Dvs spiller 1 har fået et Løsladelses kort, og sælger det til spiller 2, som er i fængsel
- **Hurtigt spil** - En hurtigere version af spillet, hvor der er visse ting/grunde tildelt på forhånd.

Analyse

Use case diagrammer.

I starten af det her projekt, havde vi en forestilling om hvordan spillet ville komme til at blive og hvad der var mulighed for at vi kunne nå at lave færdig. Jo længere vi er kommet med at programmere, jo bedre overblik fik vi. Dette startede med hvad vi forventede at den mest basale version af spillet skulle kunne, og de ting vi syntes var vigtige, som skulle implementeres og fungere. Vi har efterfølgende haft mulighed for at kunne implementere flere ting, da tiden var på vores side. Nedenfor er der et use case diagram over de første mål med programmet og hvad vi var sikre på at vi kunne nå at blive færdig med.

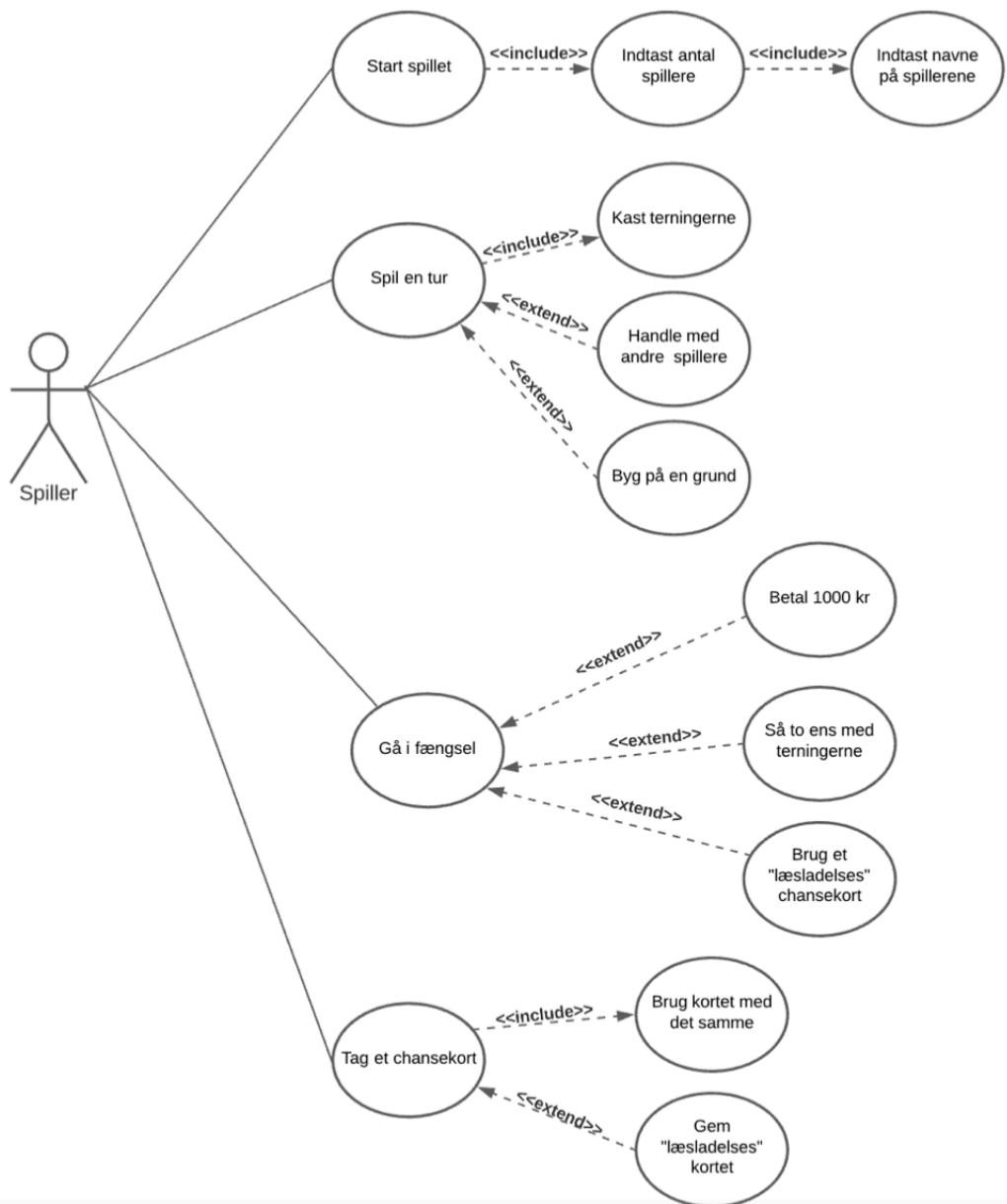


Diagram 1 Det ældre use case diagram

Vi har i samarbejde med kunden, fundet ud af hvilke ekstra til spillerne skulle kunne gøre i spille og fået det implementeret. For eksempel er funktionen "Pantsæt" blevet en mulighed for spillerne, så de har en ekstra livline. Derudover er, hvad de forskellige handlinger gør, også blevet forlænget så det er tydeligere, hvordan spille kan blive spillet.

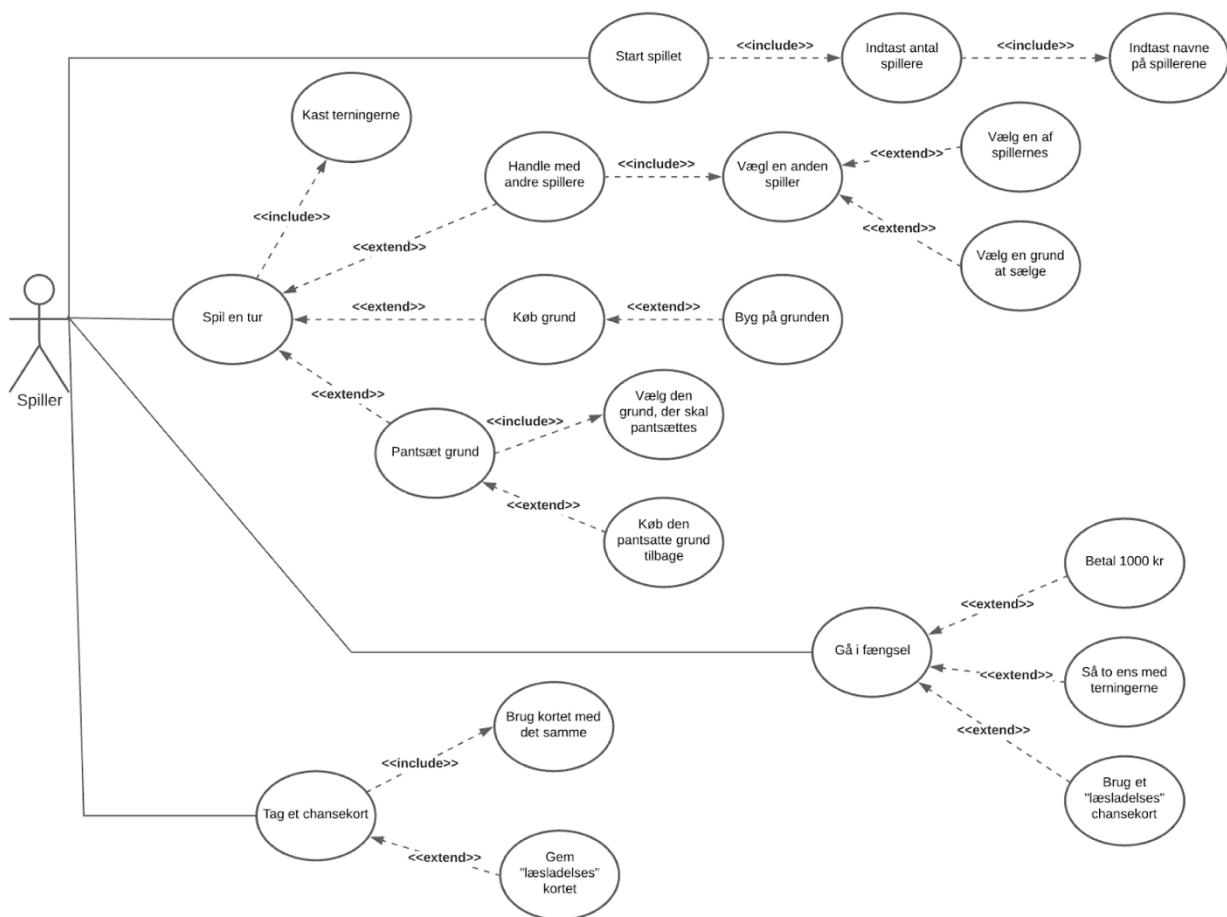


Diagram 2 Det nye use case diagram

På dette diagram ovenfor, kan man se hvordan en spiller skal/kan spille spillet. Derudover viser det også hvordan de forskellige handlinger påvirker spillerens tur, det gør at der en mulighed for selv at vælge/påvirke hvilken retning der spilles. For eksempel kan man tage udgangspunkt i use casen "Spil en tur", der vil spilleren have 4 muligheder som alle fører videre til noget nyt.

På diagrammet kan man se at alle linjerne (ud fra hoved use case'ne) er stiplet, det betyder at use casen enten include'er eller extend'er den efterfølgende mindre use case. For eksempel hoved use casen "Tage et chancekort" medføre det at man skal bruge det med det samme, med derudover kan man vælge/extend at gemme "løsladelses" kortet (til man kommer i fængsel).

Use case diagrammet viser de muligheder spilleren har gennem hele spillet, nedenfor er der forskellige hoved use cases beskrevet.

Use case Brief beskrivelse:

Use case	Beskrivelse
Start spillet	Når spillet startes op, bliver der spurgt om hvor mange der skal være med i spillet. Der kan være mellem 2 og 6 spillere. Computerne/spillet tillader ikke mere eller mindre og går ikke videre før antallet af spillere opfylder kravene.

	Efter der er blevet indtastet antallet af spillere, skal alle spillerne indtaste et navn.
Spil en tur	<p>Når en spillers tur starter, vil der være valgmuligheder, for hvordan turen skal forløbe. Der vil være 4 valgmuligheder, 1: slå med terningen, 2: sælg/køb eller byt ejendomme med en anden spiller, 3: Køb huse eller hoteller, 4: Pantsæt grund</p> <p>Hvis spilleren vælger at slå med terningerne, vil spillerens bil blive flytte det antal øjne som terningen viser. Hvorefter der er 3 nye muligheder, enten kan man købe det felt man er landet på (hvis det ikke er ejet) eller også kan man betale leje (hvis det er ejet) eller også kan man give turen videre til den næste spiller.</p> <p>Hvis spillerne vælger (blandt de første 4 valgmuligheder) at sælge/købe eller bytte ejendomme, skal der vælges en anden spiller man vil handle med og derefter blandt deres ejendomme. Der kommes med et bud, som den anden spiller enten kan acceptere, afvise eller komme med et modbud. Derefter slår man med terningen og handlingerne (beskrevet ovenfor) vil blive gennemført.</p> <p>Hvis spilleren vælger (blandt de første 4 valgmuligheder) at købe huse eller hoteller, vil kunne vælge mellem de grunde hvor man, ejer alle grunde i samme serie. Man skal bygge ligeligt blandt felterne i serien, dvs. først et hus på alle grunde, derefter 2 osv.</p> <p>Hvis spilleren vælger (blandt de første 4 valgmuligheder) pantsætte en grund, vil der være mulighed for at vælge mellem de grunde som man ejer og på den måde få penge til at spille videre. Der er mulighed for at købe grunden tilbage, hvis spilleren skulle komme til penge.</p>
Tag et chansekort	Lander en spiller på et "Prøv lykken" flet, skal spilleren trække et kort fra chancekortbunken. Dette kort kan både være en positiv og negativ handling.
Gå i fængsel	Hvis en spiller lander på feltet "Du fængsles" eller trækker fængsels chance kortet, skal spilleren rykke hen til fængselsfeltet. Spilleren har (på sin næste tur) 3 muligheder for at komme ud af fængslet, 1: betal 1.000kr, 2: buge et løsladelses chance kort, eller 3: så to en med terningerne.
Byg på grunde	For at bygge en grund kræver det at, den samme spiller har alle grundede i en serie fx serien rød, som består "Trianglen" + "Østerbrogade" + "Grønningen". Når man begynder at bygge hus på grundene, skal man gøre det ligeligt fordelt på alle tre. Dvs. først 1 hus på alle tre grunde, derefter 2 huse på alle tre grunde, osv. til man har 4 huse. Når der er 4 huse på en grund, kan man købe et hotel. Når man bygger på en grund, øger man lejen og på den måde får man flere penge, når en anden spiller lander på feltet.

Use case Fully Dressed beskrivelse:

Use case sektion	Beskrivelse	Use case sektion	Beskrivelse
Use case name	Start spillet	Use case sektion	Spil en tur
Scope	Indtast det ønskede antal spillere, derefter indtast navne på de spillere som deltager.	Scope	Der er 3 muligheder, når en tur starter. Det er ikke sikkert at alle 3 mulighed er aktive, for eksempel er det ikke sikker at spilleren har en grund at bygge huse på.
Level	User goal	Level	User goal
Primary actor	De indtaster spiller (mellem 2-6 spillere)	Primary actor	De indtaster spiller (mellem 2-6 spillere)
Success guarantee	Spillerne bliver indtastet	Success guarantee	Spilleren har 3 valgmuligheder, dette kan variere ud fra hvilke kriterier, der er opfyldt. Der vil altid være mindst en mulighed for at gøre noget. Turen slutter når spilleren giver den videre.
Main success scenario	Når der er blevet indtastet et antal spillere, mellem 2 og 6, og alle derefter har angivet et navn. Vil spille fortsætte	Main success scenario	En spiller vil vælge en/flere ud af 3 valgmuligheder (hvis spilleren opfylder kriterierne) og sende turen videre.
Extensions	Der kan være tekniske problemer i forhold til hvad man indtaster først. Hvis der bliver indtastet navne først, vil spillet afvise det indtastede. Dette kan løses ved at teste antallet af spillere (mellem 2-6) ind først. Hvis der derudover er tekniske problemer, stat spiller op igen og start forfra.	Extensions	Hvis en spiller ikke kan vælge en af de tre valgmuligheder, kan man trykke på entre knappen (på tastaturet). Dette skulle gerne få terningerne til at rulle, hvis ikke kan spillet enten give turen videre eller også er man nødt til at lukke det ned og starte op igen.
Special requirements	Der skal som minimum indtastes et antal spiller på minimum 2 og maks. 6, og	Special requirements	Man skal først vælge en ud af tre muligheder, og alt efter hvad spilleren vælger, vil der

	derefter det skal der også indtastes navne på minimum 2 og maks. 6 spillere.		enten komme nye valg, blive rullet med terningen eller komme tilbage til de første tre valgmuligheder.
Technology and data variations list		Technology and data variations list	
Frequency of Occurrence	Der er ikke et behov for at denne funktion bliver testet tit, det kommer dog an på hvor meget man spiller. Hvis man spiller hyppigt, er det ikke nødvendigt at teste tit, hvor imod det kan være en god idé at teste den oftere hvis ikke det spilles tit.	Frequency of Occurrence	Der er ikke et behov for at denne funktion bliver testet tit, det kommer dog an på hvor meget man spiller. Hvis man spiller hyppigt, er det ikke nødvendigt at teste tit, hvor imod det kan være en god idé at teste den oftere hvis ikke det spilles tit.
Miscellaneous	Hvad sker der hvis går lang tid inden der bliver tastet data ind? Fx: hvis man har indtastet et antal spillere på 3 og der går længere før den sidste spiller indtaster navn?	Miscellaneous	Kan man komme tilbage til de første tre valgmuligheder, hvis man kommer til at klikke forkert? Kan man fortryde et tilbud, der er givet til en anden spiller? Kan man blive ved med at komme med et modbud, når man handler spillere imellem?
Use case sektion	Beskrivelse	Use case sektion	Beskrivelse
Use case name	Tag et chancekort	Use case name	Gå i fængsel
Scope	Spilleren er landet på et "Prøv lykken" felt, der bliver trykket det øverste/første kort i bunken af chancekort. På kortet vil der stå en positiv eller negativ ting, som påvirker spilleren.	Scope	Spilleren skal i fængsel, og har tre muligheder for at komme ud af fængslet. Enten ved at betale 1000, eller ved at slå to ens på 3 forsøg, eller ved at bruge "løsladelses" chancekortet (det er trukket når spilleren er landet på "Prøv Lykken")
Level	User goal	Level	User goal

Primary actor	De indtaster spiller (mellem 2-6 spillere)	Primary actor	De indtaster spiller (mellem 2-6 spillere)
Success guarantee	Der bliver tildelt et chancekort, som har en positiv eller negativ indflydelse på spilleren.	Success guarantee	Spilleren kommer ind og ud af fængslet.
Main success scenario	Et kort påvirker spilleren/spillerne negativt eller positivt.	Main success scenario	Spilleren er kommet i fængsel og benytter en af de tre muligheder, til at komme ud.
Extensions	Hvis der bliver trykket et kort, der flytter brikken videre, men ikke giver dig mulighed for at købe eller betale leje, på det felt du lander på, vil turen gå videre til den næste spiller.	Extensions	Når man har slået to ens for at komme ud af fængslet, kan det være man ikke får mulighed for at købe det felt man lander på, i dette tilfælde vil turen bare gå videre til næste spiller.
Special requirements	Du skal være landet på de specifikke Prøv lykken felter og det der står på kortet skal udføres.	Special requirements	Du skal være i fængsel, og du kan kun være det i tre spille omgange.
Technology and data variations list		Technology and data variations list	
Frequency of Occurrence	Der er ikke et behov for at denne funktion bliver testet tit, det kommer dog an på hvor meget man spiller. Hvis man spiller hyppigt, er det ikke nødvendigt at teste tit, hvor imod det kan være en god idé at teste den oftere hvis ikke det spilles tit.	Frequency of Occurrence	Der er ikke et behov for at denne funktion bliver testet tit, det kommer dog an på hvor meget man spiller. Hvis man spiller hyppigt, er det ikke nødvendigt at teste tit, hvor imod det kan være en god idé at teste den oftere hvis ikke det spilles tit.
Miscellaneous	Hvad hvis "løsladelse" kortet ikke bliver gemt eller kommer frem når man er i fængsel? Hvad hvis man ikke modtager det beløbs som står på kortet?	Miscellaneous	Hvad hvis man ikke har 1000 kr og ikke sår to ens? Hvad hvis du ikke kommer ud når du betaler?

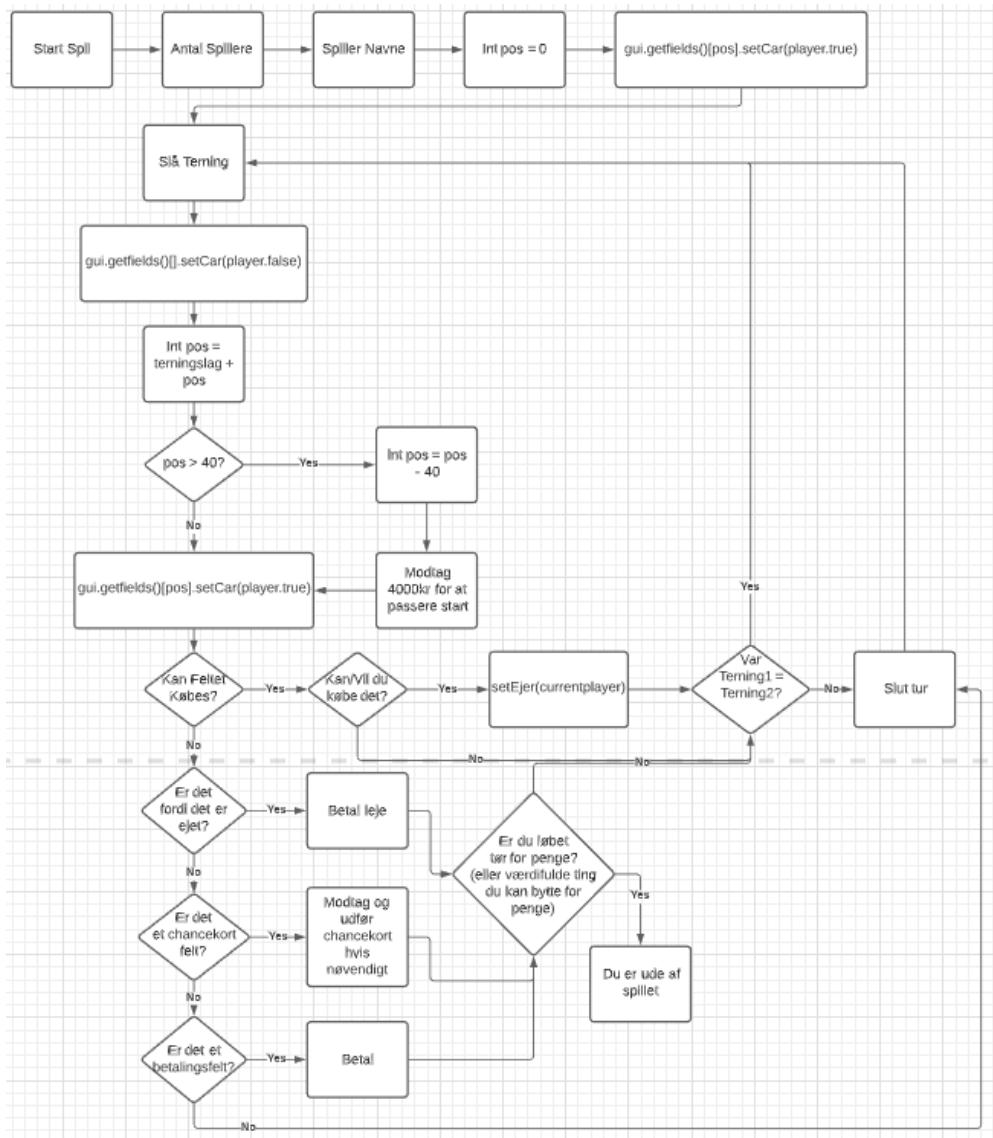
Flowchart:

Her ses et flowchart, som giver et overordnet blik over hvordan vi i starten mente at vores system ville komme til at virke.

Det viser hvordan spillet starter ud med at spørge om antallet af spillere og deres navne. Derefter sætter den så en bil til hver spiller. Herfra starter hver spillers tur, hvori de slår med terningen og så sker der noget unikt eftersom hvilket felt de er landet på.

Konstant vil hver spiller også blive tjekket om de er løbet tør for penge efter deres tur, og hvis de er vil de blive meldt fallit og udgå spillet.

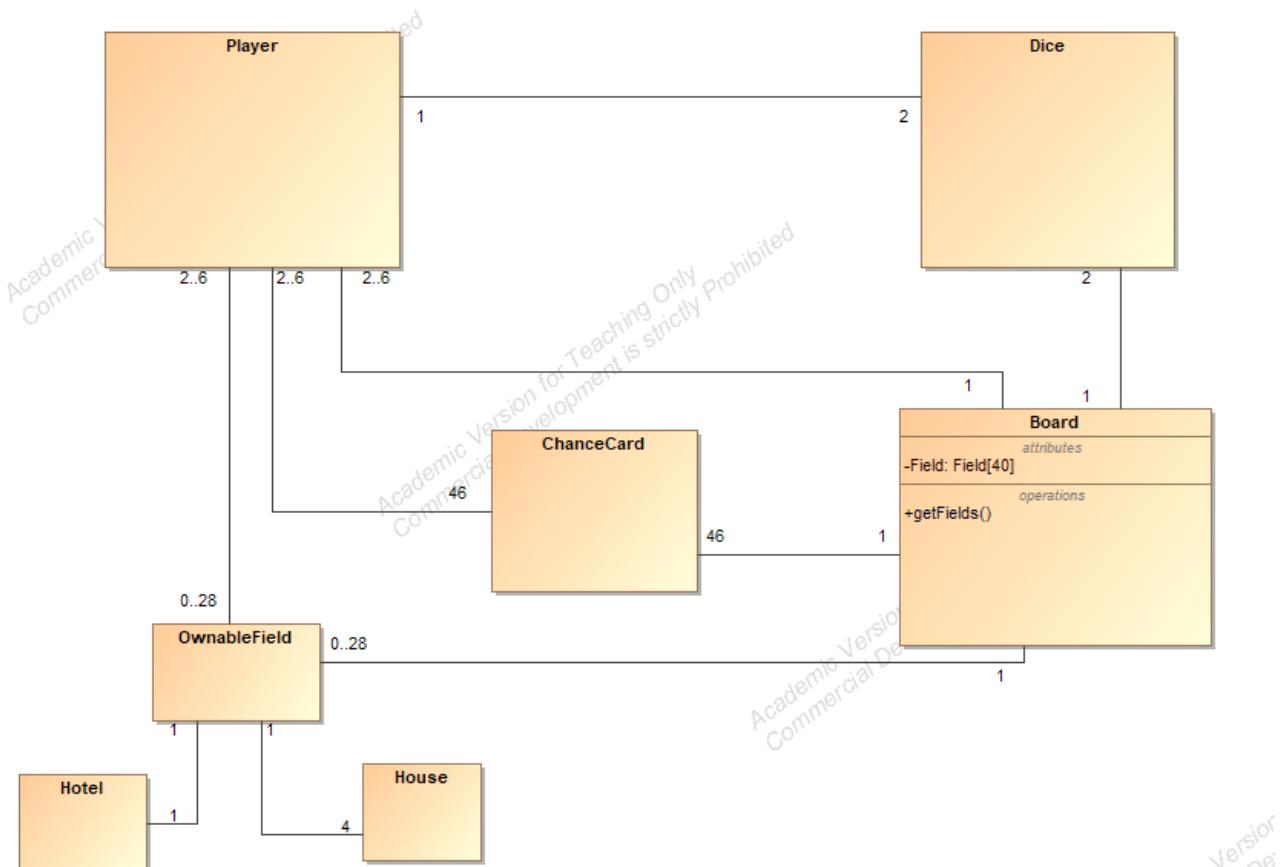
Det færdige produkt tog alle elementerne herfra, men tilføjede mere til. Fx ved starten på hver tur fik spilleren mulighed for at pantsætte, sælge, bygge og slå. Inden hver tur startede, tjekkede den også om spilleren var i fængsel og gav nogle helt andre muligheder hvis det var sagen.



Domæne model

Domænemodellen er et værktøj vi bruger til at identificere alle de fysiske komponenter vi skal have med i vores spil, vi identificerer også hvilket komponenter der interagerer med hinanden og hvilken multipliceret de forskellige komponenter har.

Nedenfor ses vores domæne model, som viser hvordan vores klasser interagere med hinanden, man kan fx. se "Player", "Board" og "Dice", og hvad deres multipliceret er til hinanden, fx kan et "Board" have to til seks "Player", og en "Player" kan bruge to "Dice". Udover det har "Board" og "Player" også relation til "ChanceCard" og "OwnableField", hvor "OwnableField" kan enten have op mod 4 "House" eller et "Hotel" tilknyttet sig.



Design

Oversigt over de forskellige klasser i programmet.

Package - ChanceCard	Package - com.company	Package - Fields
<i>CardDeck:</i> <i>CardTest:</i> <i>ChanceCard:</i>	<i>Account:</i> <i>Consol:</i> <i>Dice:</i>	<i>Board:</i> <i>BoardController:</i> <i>Brewery:</i>

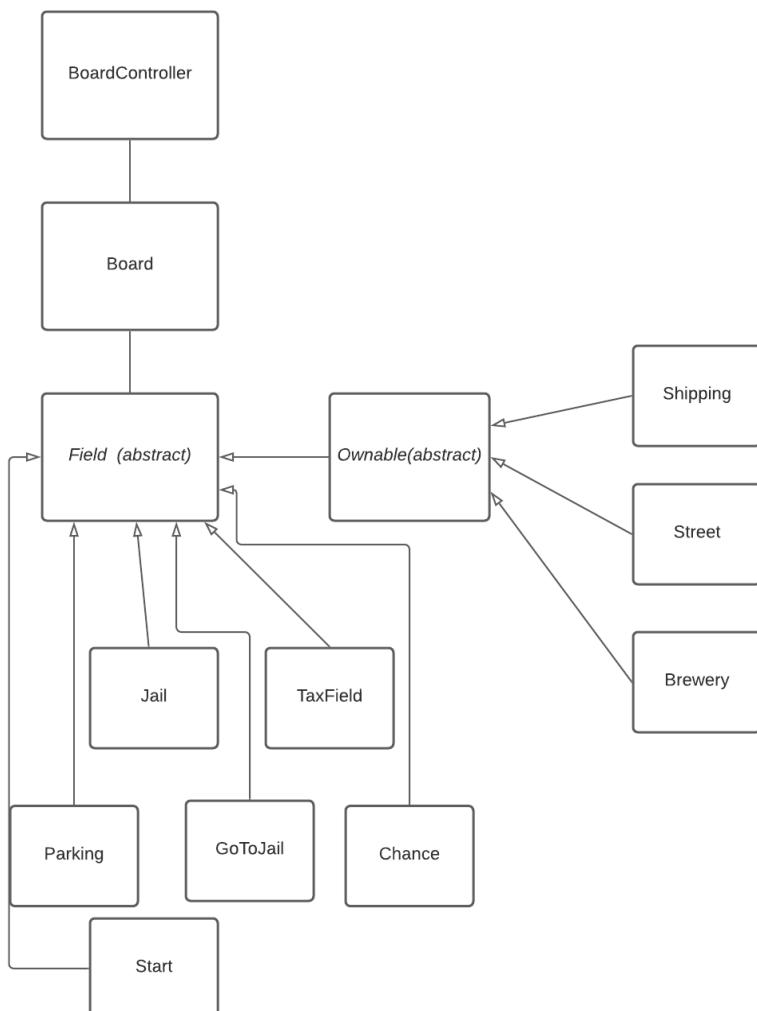
<i>GetOutOfJailCard:</i>	<i>Main:</i>	<i>Chancefield:</i>
<i>GoToJailCard:</i>	<i>Player:</i>	<i>Field(abstract):</i>
<i>IncreasePrice:</i>	<i>PlayerController:</i>	<i>GoToJail:</i>
<i>MoneyFromPlayer:</i>		<i>Jail:</i>
<i>Move:</i>		<i>Ownable(abstract):</i>
<i>MoveToShipping:</i>		<i>Parking:</i>
<i>MoveToSpecific:</i>		<i>Shipping:</i>
<i>PayMoney:</i>		<i>Start:</i>
<i>ReceiveMoney:</i>		<i>Street:</i>
		<i>TaxField:</i>

Uml diagram

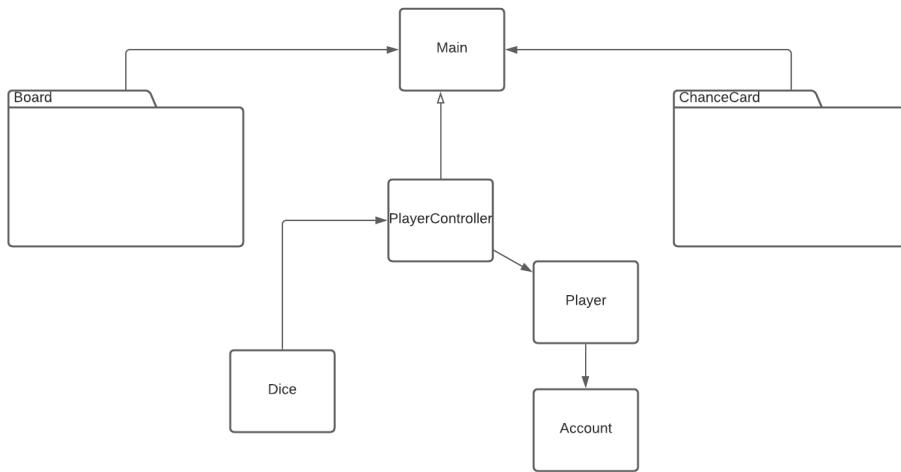
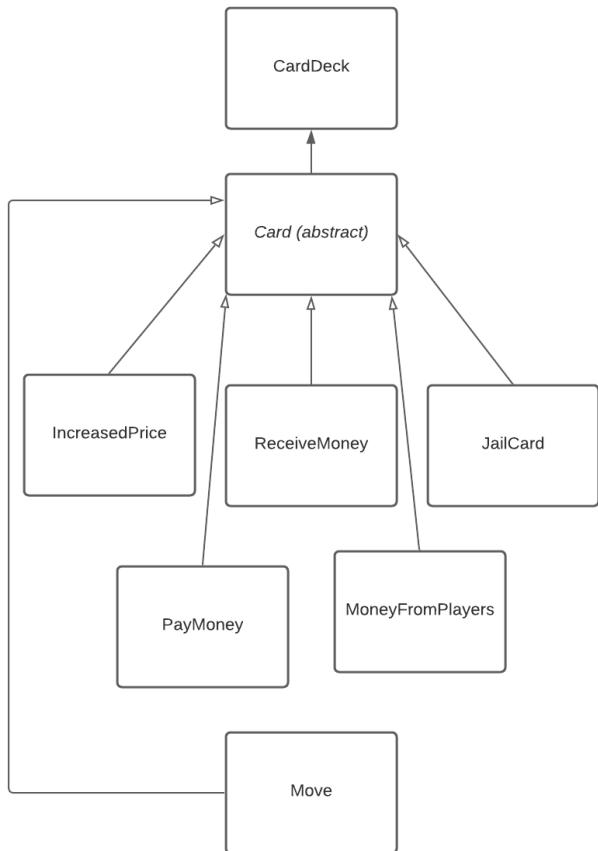
Klassediagram ud fra udgangspunktet:

Nedenfor ser man de klassediagrammer, som var med i M1 afleveringen. Disse diagrammer er dem vores udgangspunkt bygger på, det er vores første plan for, hvad vi forventede var muligt at nå at lave.

Nedenstående refers til som følger i overordnet klassediagram: Board



Disse klasser samles overordnet som: ChanceCards



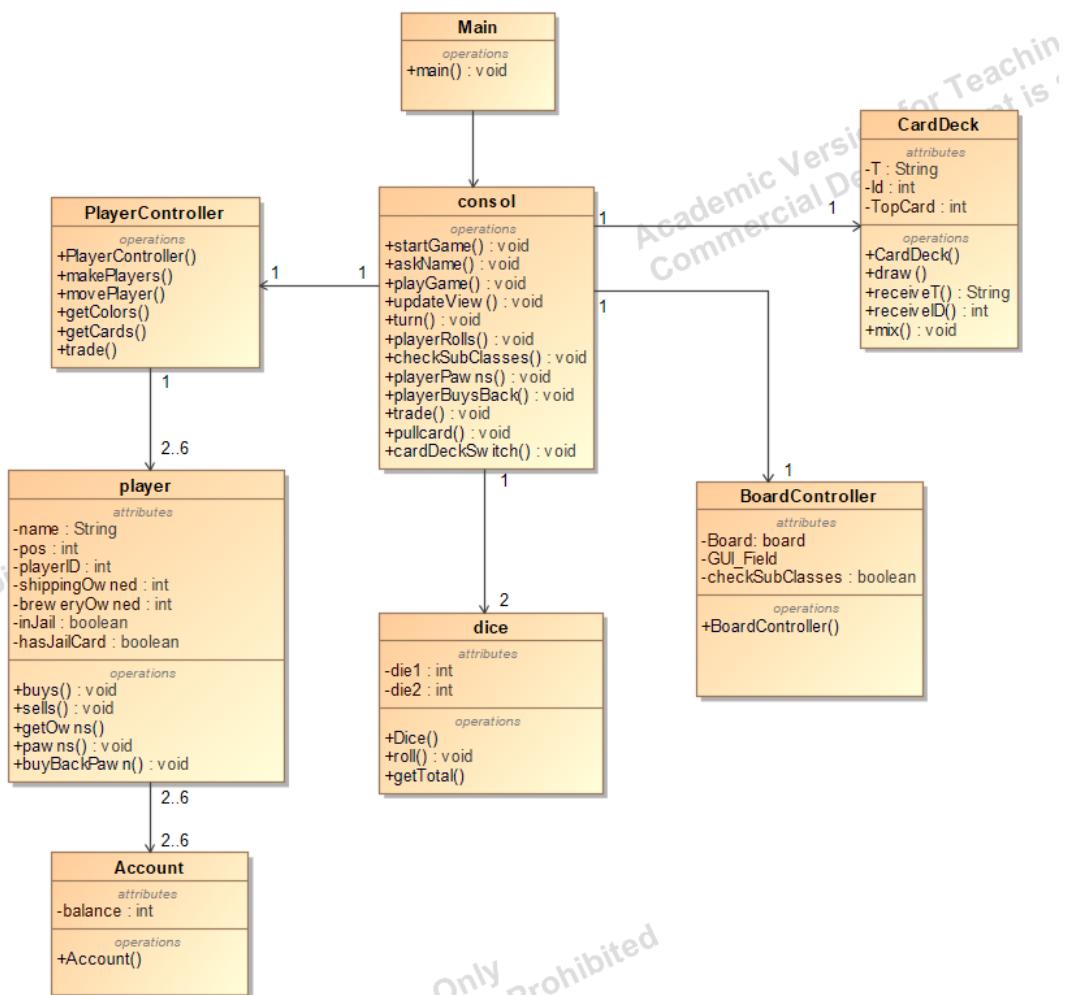
Eftersom at vi har valgt at implementere flere ting i spillet end vi først regnede med, og efterfølgende også har kendskab til de forskellige klasses attributter, videreført på klasse diagrammerne ovenfor og lave et design klassediagram. Dette er beskrevet og vist nedenfor.

Design klassediagram:

Nedenfor vises et design klassediagram, der indeholder de forskellige klasser, der hver især har nogle metoder og attributter. Klasserne har derudover også relationer og forbindelser med andre klasser og dette er også vist i diagrammet nedenfor. Vi har lavet et design klassediagram, for at give et overblik over den kode vi har lavet.

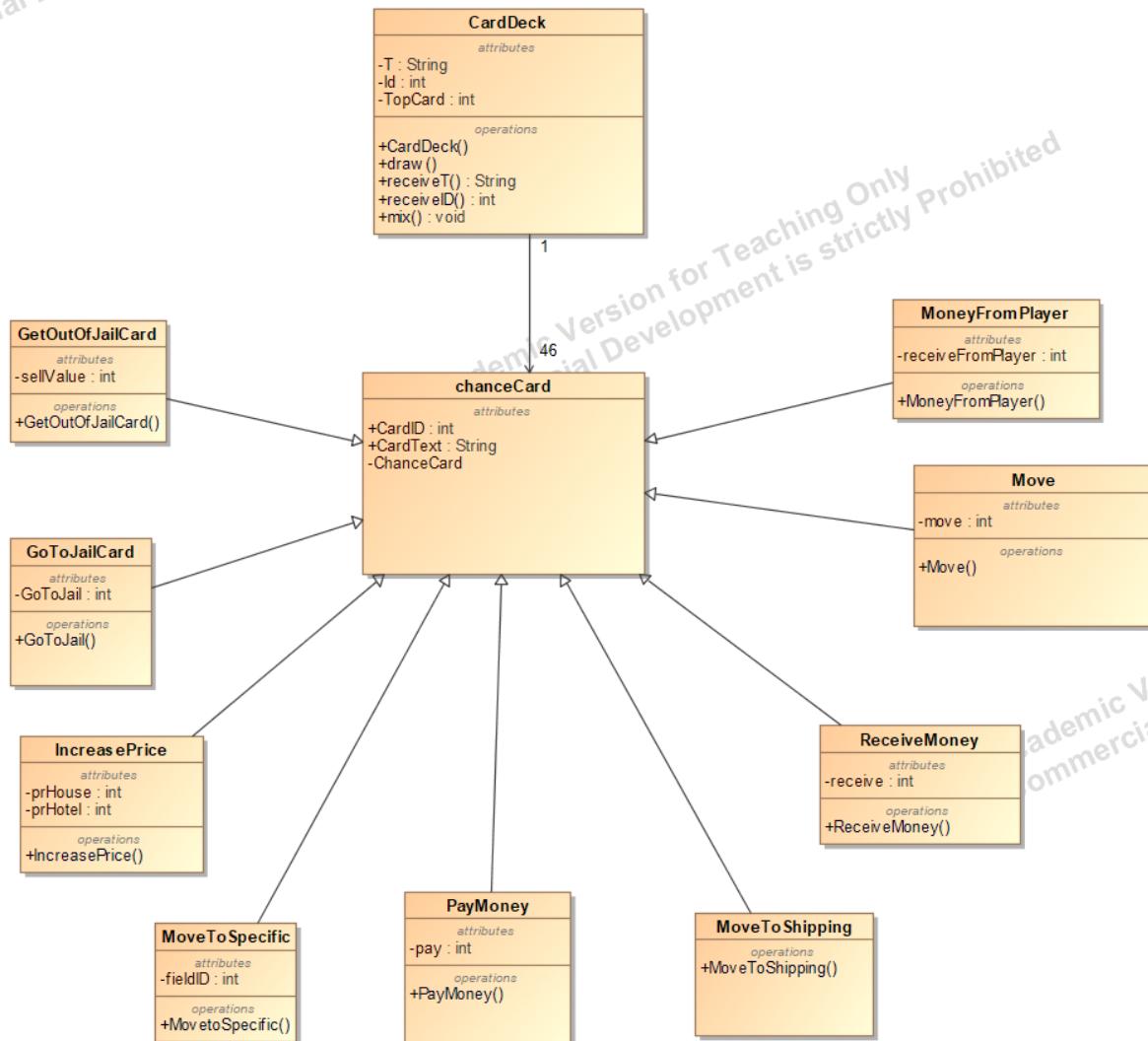
Det er relationerne i et design klassediagram, som fortæller /viser, hurtigt og nemt, hvordan de forskellige klasser er afhængige og bruger hinanden til at få det samlede spil/kode til at fungere som en helhed (dette kan blandt andet ses ud fra pilene i diagrammet).

Da dette er et ret stort og komplekst program, har vi valgt at dele design klassediagrammet op i tre diagrammer, også for at sikre at man kan læse hvad der står. På den første figur (Figur 1) vises det hvordan helle systemet virker generelt. I vores system er det Consolen, den mest centrale klasse. Det er "consolen" der modtager anmodningen udefra, hvorefter den udløser de forskellige muligheder baseret på de forskellige klasser der har "association" til "consolen", og derefter sende det tilbage i en viewable kode til brugeren.

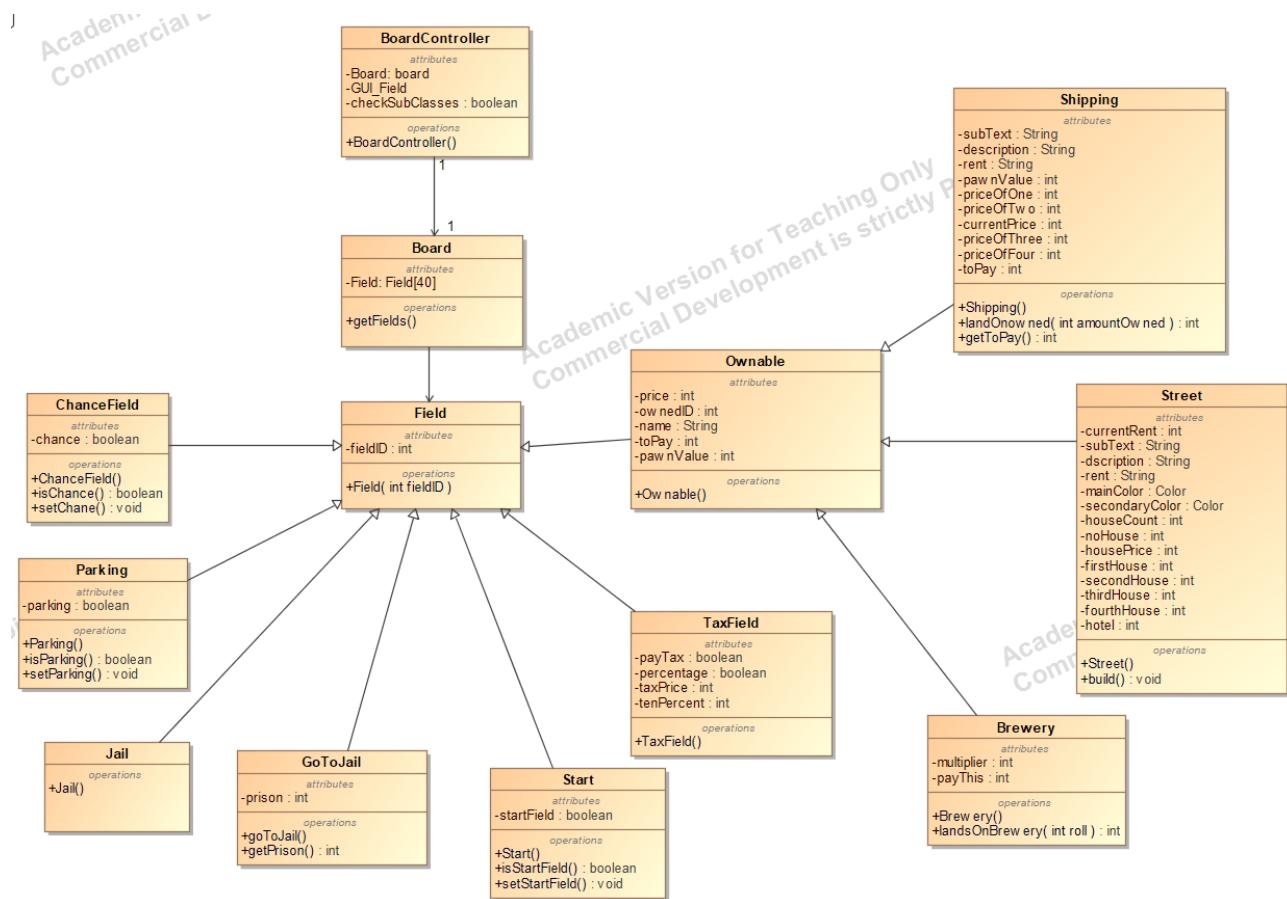


Figur 1

Figur 2 og 3 nedenfor viser en mere detaljeret oversigt over "Boardcontroller" og "CardDeck". Figur 2 viser oversigten over "CardDeck", og hvordan den er associeret til "ChanceCard", derudover er de yderst klasser, for eksempel "GetOutOfJail" og "Move", er nedarvninger af klassen "ChanceCard". Dette koncept ses også på samme måde i figur 3, her omhandler det bare nedarvning til klassen "Field". For eksempel er "Parking" + "GoToJail" + "Jail" + "Start" ovs, nedarvning til klassen "Field"



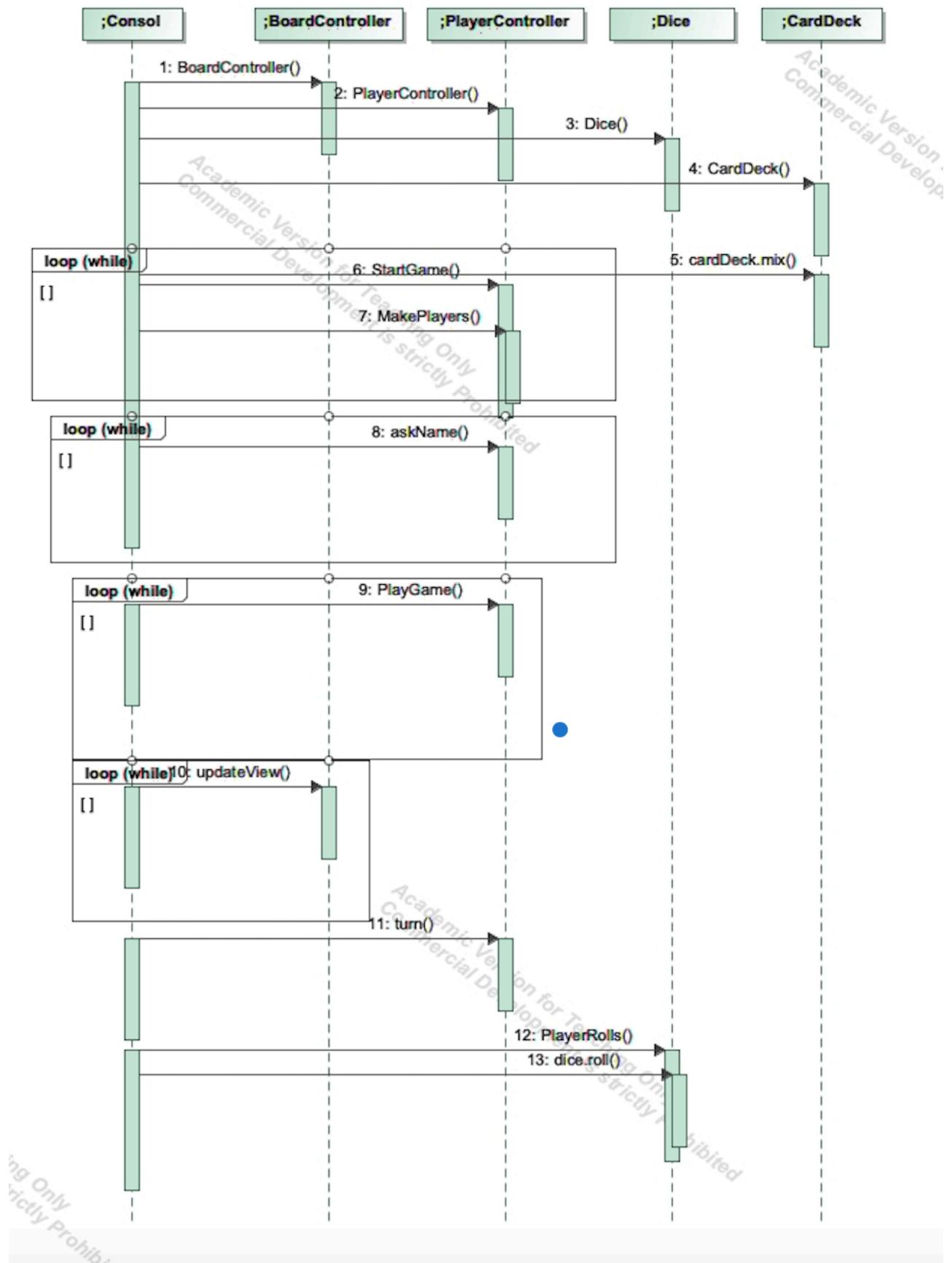
Figur 2



Figur 3

Sekvensdiagram:

Diagrammet foruden viser et udsnit af spillet mellem de vigtigste klasser i spillet, klassen "consol" kommunikerer med klasserne "boardController", "PlayerController", "Dice" og "Carddeck", igennem de kaldte metoder fra klasserne i consolen, hvor der dannes løkker i de centrale metoder, og hvilket giver spillerne funktionaliteten til kører rundt på spillets bræt. Vi har dog valgt at abstrahere nogle detaljer væk i diagrammet, for at gøre det overskueligt.

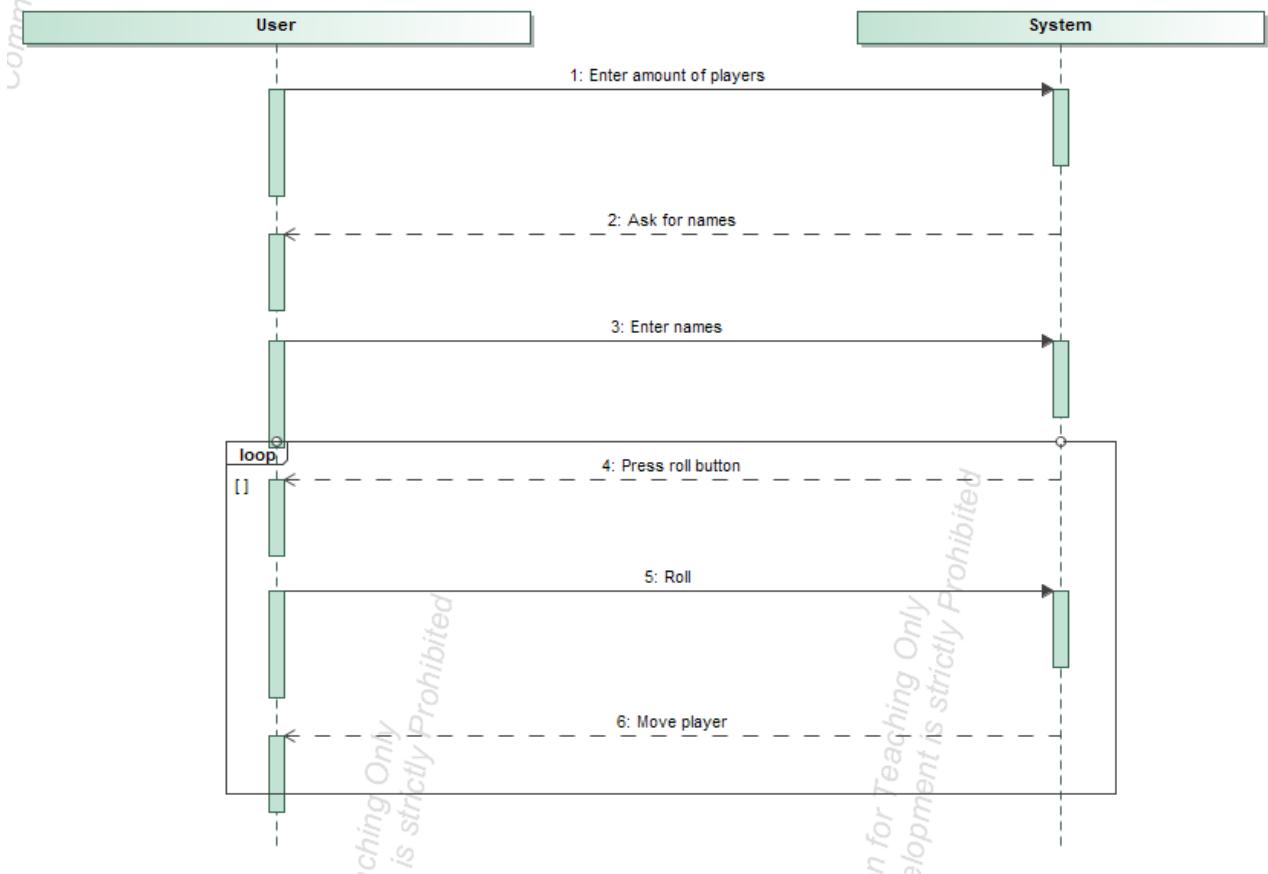


System Sekvensdiagram:

Dette diagram nedenfor, viser spillers interaktion med systemet. I diagrammerne bliver spillerne betegnet som bruger eller user, og er placeret til venstre.

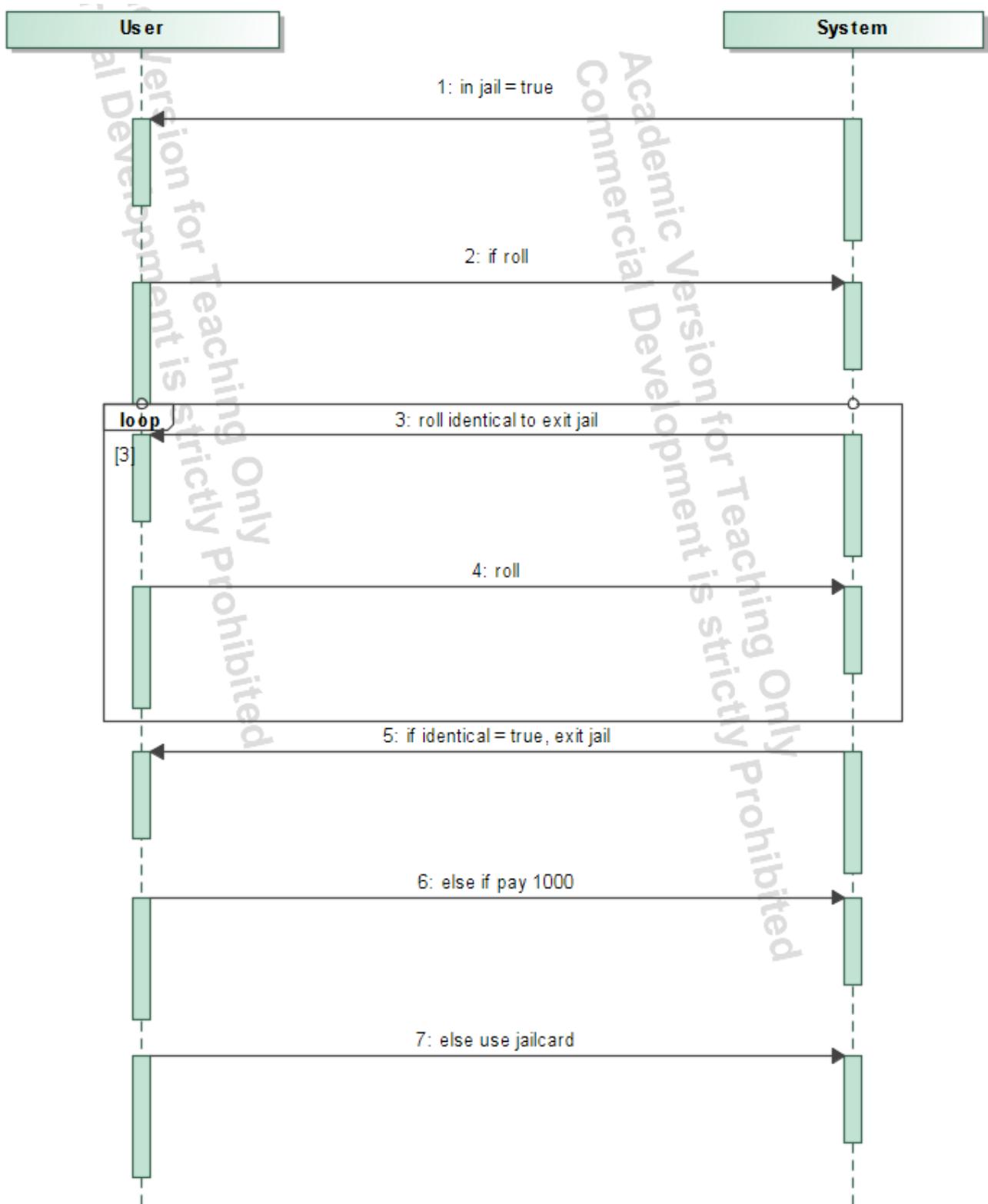
Start spil

I dette diagram ses det hvordan spiller interagere med systemet fra spillets start til første kast med terningen. Det ses at systemet spørger brugeren/useren om antal spiller, efterfulgt af deres navne. Alle spillere skiftes derefter til at kaste med terningen og bliver derefter rykket tilsvarende antal felter som terningens øjne viser.



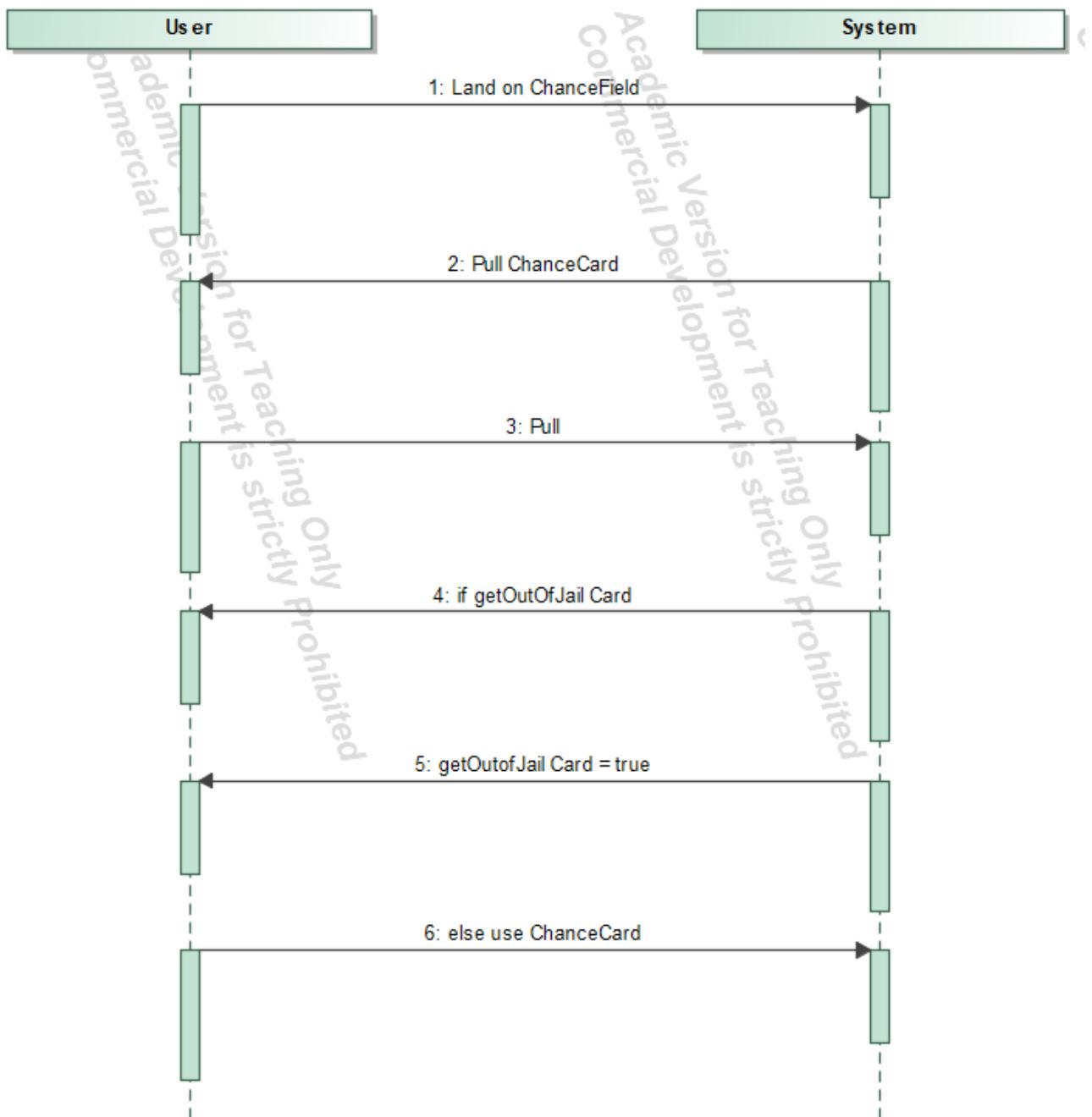
I fængsel

Er en af spillerne uheldige og ryge i fængsel inde i spillet, får man 3 muligheder for at komme ud af fængsel. Brugeren kan slå to ens med terningerne og det har de 3 forsøg til per tur. Brugeren kan også vælge at betale sig ud af fængsel ved at betale systemet 1000 kr. Og den sidste måde hvorpå de kan komme ud af fængsel er, hvis brugeren har været heldig tidligere i spillet og trække et chancekort der benåder dem muligheden for at komme ud af fængsel gratis 1 gang.



Chance kort

I dette systemsekvensdiagram ses hvordan bruger interagerer med systemet, når bruger lander på et chancekort felt. Når brugeren lander på feltet, vil systemet fortælle brugeren at de skal trække et kort, hvis det er et kort, hvorpå man får at vide at det er kongens fødselsdag og man kan bruge kortet til at komme ud af fængsel gratis, kan man derfor gemme det kort til man måske ryger i fængsel senere. Hvis det ikke er et specifikke kort skal brugeren gøre som der står på kortet med det samme.



Implementering

ChanceCard

ChanceCard er en abstrakt klasse der bliver benyttet til at give individuelle kort-klasser et id og et tekstafsnit.

Alle klasser af specifikke chancekort nedarver fra klassen ChanceCard, som er sorteret ud fra deres generelle funktion ved træk af et kort. Dette benyttes fordi der overordnet set er 45 kort, men der er egentlig kun 9 unikke effekter disse har. Det resterende er egentlig blot små variationer af samme kort. Eksempel på dette er klassen Move-klassen der omhandler at flytte spillebrikken eller PayMoney-klassen hvor spilleren skal betale et bestemt beløb til banken. Dermed får de en int funktion til at definere beløbet eller antal felter brugeren skal flytte. I mange tilfælde kan chancekortenes funktion altså blive reduceret til et enkelt tal, hvis blot man ved hvilket slags kort der er tale om.

Klassen "CardDeck" opretter et array af ChanceCard's. Dette array kalder vi "CardDeck". Arrayet er endimensionelt og 46 langt, og består altså udelukkende af objekter der alle er nedarvet fra ChanceCard. Det er ikke i array'et at de individuelle kort får en plads i dækket, og benytter sig af deres konstruktør fra deres klasser, for eksempel at kortet med ID nr. 5 har teksten betal 3000 kr. For at få deres vogn repareret. Det er et kort fra klassen "paymoney" og dermed vil det int fra klassen blive defineret til 3000kr.

Afvigelser

I følgende afsnit redegøres for de afvigelser der er at finde i forhold til det udleverede brætspil. Der er i koden tre chancekort som afviger i varierende grad i forhold til virkeligheden.

Matadorlegatet

Matadorlegatet er det første kort som til en vis grad afviger fra det originale kort. Det skyldes at der ved udarbejdelsen af koden ikke fandtes passende tid til at skabe en metode der udregner en spillers samlede værdi.

Derfor besluttedes det, i kortets samme ånd, at modifcere denne, så i tilfælde af at en spiller trækker dette kort, må en spiller maksimalt have en balance i sin konto på 5000 kr i stedet for en samlede værdi af aktiver på 12000 kr. Det vedkendes her at dette i en suboptimal situation hvor en spiller netop har brugt en stor mængde penge for at øge aktiver, værende enten handel, køb af nye grunde eller at bygge på disse, vil kunne modtage et ekstra kapitalindskud. Det blev dog vurderet at dette kort er en central del af spillet, og i de tilfælde hvor kortet bliver trukket i højere grad giver mindeværdige øjeblikke end følelser af uretfærdighed.

I de 2 kort hvor man normalt bliver bedt om at rykke til nærmeste redderi og betale dobbelt leje hvis den er ejet, har vi det så man betaler normal leje da tiden ikke lige var til det. Det samme var tilfældet med 'kom ud af fængsel' -kortet, da vi ikke har muligheden for at sælge det, men ud over det virker det som normalt.

```

public class CardDeck {
    ChanceCard[] cardDeck;
    String T;
    int ID;
    int TopCard;

    public CardDeck() {
        setTopCard(0);
        cardDeck = new ChanceCard[46];
        cardDeck[0] = new IncreasePrice( text: "Olieprisen er øget med 10 kr." );
        cardDeck[1] = new IncreasePrice( text: "Ejendomsmægleren har fået en 10% prisøgning." );
        cardDeck[2] = new PayMoney( text: "De har kørt ud med en bil og må betale 200 kr." );
        cardDeck[3] = new PayMoney( text: "Betal for vognen med 100 kr." );
        cardDeck[4] = new PayMoney( text: "Betal 200 kr. til en børnegruppe." );
    }
}

```

[CardDeck.draw\(\)](#)

Draw-metoden er en metode der benytter til at finde det nuværende øverste kort, samt at returnere den tekst og type som vi benytter til at finde ud af hvad det givne kort er for en underklasse.

I CardDecks konstruktør defineres hvad der senere hen er det øverste kort i bunken. Essentielt skal topCard blot forstås som en pegepind til at en adresse i arrayet af chancekort.

Metoden returnerer altså et kort hvorefter pegepinden dernæst henviser til det næste kort der skal trækkes.

```

public ChanceCard draw() { //returnerer kort
    ChanceCard card = cardDeck[getTopCard()];
    this.T = cardDeck[getTopCard()].CardText;
    this.ID = cardDeck[getTopCard()].CardID;
    setTopCard(getTopCard() + 1);
    return card;
}

```

- Mix

ChanceCard.Mix er i sin kerne en while-løkke der bliver gentaget 1000 gange.

Metoden finder to tilfældige tal mellem 0 og 45 definerer dem som i og j. Efterfulgt siger vi e er lig med hvad f var og på den måde bytter de 2 kort plads.

Denne metode vil kører 1000 gange så kortene ligger i tilfældig rækkefølge og på den måde laver en blandings metode med 2 nye tilfældige kort der skifter plads 1000 gange hver gang metoden kaldes.

En god ting ved denne metode er at efter de så er blevet blandet så vil bunken holde sig i én rækkefølge. Dette er altså det tætteste bunken kommer på et virkeligt spil.

```
public void mix() {
    int t = 0;
    while (t < 1000) { //blander alle korter
        int i = (int) (Math.random() * 45);
        int j = (int) (Math.random() * 45);

        ChanceCard e = cardDeck[i];
        ChanceCard f = cardDeck[j];

        cardDeck[i] = f;
        cardDeck[j] = e;
        t++;
    }
}
```

[Set top card](#)

SetTopCard metoden er en metode, vi bruger til at starte bunken forfra, det vil sige, når der er blevet trukket mere end 45 kort, vil metoden gøre, så bunken starter forfra, ved at sige

SetTopCard(getTopCard)-45 = hvilket vil få bunken tilbage til det første trukket kort i bunken.

```
public void setTopCard(int topKort) {
    this.TopCard = topKort;
    if(topKort > 45) {
        setTopCard(getTopCard() - 45);
    }
}
```

Player

Player er en klasse for sig selv, hvor vi definerer hvad en player er. En player har fx name, playerID, account og position.

ArrayList

```
List<Integer> owns = new ArrayList<Integer>();
List<Integer> pawned = new ArrayList<Integer>();
```

I forbindelse med at finde den nemmest mulige løsning på hvordan en spiller nemt skal kunne købe og/eller sælge, fandtes at man i alle situationer ville stå med en datamængde der hurtigt ville kunne blive større eller mindre og i visse tilfælde kan datamængden, blandt andet i starten, blive 0. Derfor fandtes det at der vil kunne opstå mange problemer med brug af normale arrays, da disse størrelse skal initialiseres ved oprettelse. Dette er problematisk da spillere som udgangspunkt ikke ejer nogen grunde.

Gruppen besluttede derfor at gå ud over hvad der er pensum, ikke fordi dette var en motivation men derimod fordi det er en ubestridt bedste løsning, og benytte arraylist til at samle referencerne til de ID'er som en spiller ejer.

ArrayLister er arrays som ikke har en specifik størrelse. Det som ArrayLister tilbyder, som ofte er et problem med klassiske arrays er at disse i højere grad er dynamiske og er gode, hvis man er usikker over hvor lang en sådan bør være.

Enhver spiller har to ArrayLister, en til de grunde han ejer og en til de grunde han har pantsat. Altså benyttes der blot et ArrayLister af Integers. Disse ArrayLister refererer til de felter, hvis ID'er bliver defineret ved oprettelse og ikke ændrer sig igennem spillet. Man kan derfor nemt referer til dem.

De situationer hvor ArrayLister giver de største fordele er når en spiller skal have fjernet et ID ved f. eks. salg af en grund. ArrayLister har nemlig en metode der hedder: .remove(object) som ikke fjerner en bestemt plads i listen men derimod fjernes det først kommende tilfælde af det specifikke objekt. Det betyder altså at man kan benytte denne metode til at fjerne et specifikt ID fra listen uden at skulle tage stilling til dens adresse i listen som man ellers ville være nødt til en et klassisk array.

Setpos:

Der er blevet oprettet en Set position metode ved hver player, hvor den tracker players position. I eksemplet nedenfor ses, når en players position er større end 40, vil hans position starte forfra, det sker hver gang når en player har gået en tur på spillepladen. Der vil også blive tilføjet 4000 kr på players konto, fordi når position er større end 40, betyder det også at player enten lander på start eller har passeret start.

If pos under 0 skal pos pluses med 40, et eksempel kunne være at man stod på felt nr 2 får at vide man skal rykke 3 felter tilbage. Så i stedet for at positionen går i minus 1 vil positionen ende på 39.

```

public void setPos(int pos) {
    this.pos = pos;
    if (pos > 39) {
        this.pos = getPos() - 40;
        playerAccount.setBalance(playerAccount.getBalance() + 4000);
    }
    if (pos < 0) {
        this.pos = getPos() + 40;
    }
}

```

Board controller

I board controlleren har vi en association til board klassen, board klassen indeholder et array af fields. Selve "boardcontroller" klassen opretter en boolean der hedder check subclass. I "BoardController" klassen har vi også importeret gui field fra maven. Gui field vil give en visuel spilleplade som vi har definere i board.

Der er lavet en metode der hedder BoardController(), som benytter sig af gui fields, som opretter et array af 40 GUI_Field's. Vi har derefter lavet et while loop der checker og identificere forskellige booleans af check subclass. I check subclass vil koden se om vi bruger en instans fra en af vores field klasser, hvis instansen er sand vil gui field vise den instans den har identificeret som værende sand. Eksemplet nedenfor vil vise at finder loopet en instans af "start" vil gui_field[i] være lig med den title og subtext som den er tildelt.

```

public BoardController() {
    board = new Board();
    gui_fields = new GUI_Field[40];
    int i = 0;
    while (i < 40) {
        checkSubClass = (board.getFields()[i] instanceof Start);
        if (checkSubClass) {
            gui_fields[i] = new GUI_Start( title: "Start", subText: "Få 4.000 kr",
        }
}

```

Mest nævneværdigt i denne klasse er metoden checkFields() der har til funktion at sørge for at brættet konstant er opdateret i forhold til hvorvidt en spiller har mulighed for at bygge på alle grundene og omvendt.

Consol

Start game

Start game er vores metode som primært indeholder de ting som er i spillet når spillet starter, såsom cardDeck.mix(), hvor chancekortene allerede bliver mixede helt i starten. Derefter har vi

benyttet os af `gui_getUserInteger`, til at spørge brugeren hvor mange spillere der skal være, hvor brugeren har input mulighed mellem to og seks, hvis input er mindre end 2 eller større end 6, så vil der blive vist fejl, og input vil være tilgængeligt indtil brugeren har indtastet et tal mellem to til seks, da det er en loop. Efter indtastning af antallet spillere, vil der så blive spurgt efter navnene på spillerne efter rækkefølgen. Der vil blive oprettet en række instanser efter indtastning af navnene, såsom spillers brikker og spillernes account med en værdi på 30000 kr.

```
public void startGame() {
    cardDeck.mix();
    while (true) {
        int numberInput = gui.getUserInteger(msg: "Hvor mange skal spille med? I");
        playerController.makePlayers(numberInput);
        if (numberInput < 2 || numberInput > 6) {
            gui.showMessage(msg: "Fejl! Der må kun være 2-6 spillere");
        } else {
            askName(numberInput);
            break;
        }
    }
}
```

Consol

Consolen indeholder vores længste koder, og største delen af alt vores logik til de forskellige klasser. Consolen er vores kontroller ud fra et model-view-controller-synspunkt.

Det er her man har støbt og samlet de forskellige metoder og muligheder for brugeren.

Havde vi haft mere tid eller sorteret vores consol i flere consoller/kontrollere ville man kunne indedle de forskellige metoder i emner som jail chance card osv. For at få en mere overskuelig consol/kontroller.

Play game

Play game metoden, er en metode der forholder sig til den individuelles spillers status, i starten af hans tur. Metoden har et while loop, som er true hvilket vil sige at den altid vil køre det, som kun har 1 mulighed for at break, hvilket er ved den første if sætning der bliver som tjekker om der kun er 1 spiller tilbage der ikke er fallit og derved vil den sidste spiller tilbage være vinderen.

```

public void playGame() {
    int playerIndex = 0;
    int iDWon = 0;
    while (true) {
        if (playersGivenUp == playerController.getPlayers().length - 1) {
            for (int i = 0; i < playerController.getPlayers().length; i++) {
                if (!playerController.getPlayers()[i].isBankrupt()) {
                    iDWon = i;
                    break;
                }
            }
        }
    }
}

```

Så er der et for loop der kører alle spillerne igennem og den spiller der ikke har playergivesup status er udråbt til vinder. Den næste status der bliver undersøgt er om spilleren har isinjail = true. Så vil spilleren undersøges for om de har en boolean der er true eller false, for om de har et kom-ud-af-fængsel-kort, og derefter få mulighed for enten at bruge det eller lade være. Hvis de ikke har et chancekort eller ikke bruger det til at komme ud af fængslet har vi lavet et while loop der giver mulighed for at slå to ens tre gange til at komme ud af fængslet uden at skulle betale. Hvis de ikke vælger den mulighed, er den sidste mulighed at man skal betale 1000 kr. For at komme ud af fængslet. Her har vi afviget fra de originale regler da vi ikke har det med som siger at efter man har prøvet at slå sig ud 3 gange så bliver man tvunget til at betale sig ud.

Det ærgerlige ved at vi ikke har den feature med er at folk kan “gemme sig” i fængslet hvis det ville være godt for dem.

Turn

I denne metode startes der med at tjekke om spilleren er fallit, hvis dette er tilfældet, bliver spilleren sprunget over. Hvis ikke spilleren er fallit, bliver der lavet en String, med 4 valgmuligheder, som kan blive anvendt. De forskellige valgmuligheder kører i en switch, som gennemgår de forskellige cases.

Den første cases hedder “Køb eller sælg” og her starter den med at hente et array af spillernes id, som bliver brugt i et “for” loop til at finde navnene på de forskellige spillere.

```

case "Køb af spiller.":

    String[] playerNames = new String[playerController.getPlayers().length];
    for (int i = 0; i < PlayerController.players.length; i++) {
        playerNames[i] = playerController.getPlayers()[i].getName();
    }
}

```

Efterfølgende tager koden en boolean fra et spillerinput. Denne bliver benyttet til at bekræfte at spilleren vil handle.

Hvis spilleren gerne vil handle, vil arrayet der blev lavet, blive hentet og benytte det array til at lave en drop-down med navne man kan vælge i mellem. Man kan ikke handle med sig selv, og for at undgå det, har vi lavet en while-løkke, som tjekker for dette.

```

String selectPlayer = gui.getUserButtonPressed( msg: "Hvilken spiller vil du handle med?", playerNames);
int idChosen = 0;
while (idChosen < playerNames.length) {
    boolean chosen = playerController.getPlayers()[idChosen].getName().equals(selectPlayer);
    if (chosen) {
        idChosen = playerController.getPlayers()[idChosen].playerID;
        break;
    }
    idChosen++;
}
if (idChosen == playerIndex) {
    gui.displayChanceCard( txt: "Det er dig selv. Du kan desværre ikke handle med dig selv.");
    break;
} else if (playerController.getPlayers()[idChosen].owns.size() != 0 && idChosen != playerIndex) {
    trade(playerIndex, idChosen);
} else if (playerController.getPlayers()[idChosen].owns.size() == 0) {
    gui.getUserButtonPressed( msg: "Denne spiller har ingen grunde.", ...buttons: "OK");
}
break;

```

På billedet ovenfor kan man se, at der først bliver lavet en int der hedder idChosen = 0 og derefter starter while løkken. Derefter bliver den spiller man ønsker at handle med fundet, hvis id'et ikke passer, "ledes" videre til det passer, med metoden idChosen++

Da vi ikke ønsker at man kan handle med sig selv, må idChosen og playerIndex, ikke være det samme. Derudover kan man heller ikke handle med en spiller som ingen grunde har, dette har vi sikret os ved at "owns.size()" ikke må være lig med 0 (!=0). Hvis idChosen og playerIndex, heller ikke er ens (idChosen != playerIndex), og hvis owns.size() =0, bliver der givet besked om at denne spiller ikke har nogen grunde.

Hvis en spiller gerne vil pantsætte en grund, vil der først blive tjekket om spilleren har en grund.

```

case "Pantsæt":
    if (playerController.getPlayers()[playerIndex].owns.size() == 0 && playerController.getPlayers()[playerIndex].pawned.size() == 0) {
        gui.getUserButtonPressed( msg: "Du har ikke nogen grunde at pantsatte.", ...buttons: "Ok");
        break;
    }

```

Dvs. at det spilleren ejer ikke skal være lig med 0 og det der er pantsat, ikke skal være lig med 0, for at kunne pantsætte. Hvis de ikke er lig med 0, vil der kommen en String som spørg om spilleren vil pantsætte en ny grund eller købe en tilbage. Derefter tjekkes der om owns.size er 0, hvis ikke bruges metoden playerPawns(playerIndex). Hvis spilleren vil købe en grund tilbage, bliver metoden playerBuysBack(playerIndex).

```

case "Pantsat en ny.":
    if (playerController.getPlayers()[playerIndex].owns.size() == 0) {
        gui.getUserButtonPressed( msg: "Du har ikke flere grunde du kan pantsatte.", ...buttons: "Ok");
        break;
    }
    if (playerController.getPlayers()[playerIndex].owns.size() != 0) {
        playerPawns(playerIndex);
        break;
    }

case "Køb grund tilbage.":
    playerBuysBack(playerIndex);
    break;

```

Til sidst bliver spillerens konto tjekket, hvis ikke spilleren slår med terningerne, vil hele den yderste switch blive kørt igen. Derefter bliver der tjekket om spilleren er berettiget til en ekstra tur. Dette ses nedenfor.

```

checkPlayerAccount(playerIndex);
if (!choice.equals("Slå terningen")) {
    updateView(playerController.getPlayers().length);
    turn(playerIndex);
}
extraTurn(playerIndex);
updateView(playerController.getPlayers().length);

```

Check subclass i consol

CheckSubClasses metoden starter med at tjekke om spilleren er landet på et ownable felt, efterfulgt tjekker den om feltet er pantsat eller kan købes, og det er defineres ved et owner ID på henholdsvis -2 og -1.

Hvis feltets ejerID er -1 hvilket betyder at ingen spiller har købt denne, får spilleren mulighed for at købe den. Hvis den bliver købt, vil spillerens navn blive vist på feltet. Derudover ændres feltets ejerID det at tilsvare spillerens ID. I den forbindelse tjekkes det også hvilken nedarvning af Ownable det givne felt er eksempelvis street, shipping eller brewery. Siden der sker unikke ting ved rederierne og bryggerierne, er det nødvendigt at tjekke hvilken type af ownable man køber da disse grundes leje er bestemt på baggrund af hvor mange grunde spilleren ejer af denne type.

```

public void checkSubClasses(int playerIndex) {
    boolean checkOwnable = (boardController.getField())[PlayerController.players[playerIndex].getPos()] instanceof Ownable;
    if (checkOwnable) {
        Ownable ownable = (Ownable) boardController.getField()[PlayerController.players[playerIndex].getPos()];
        GUI_Ownable gui_ownable = (GUI_Ownable) boardController.getGui_fields()[playerController.getPlayers()[playerIndex].getPos()];
        if (ownable.getOwnedID() == -2) {
            gui.getUserButtonPressed( msg: ownable.getName() + " er pantsat.", ...buttons: "Ok");
        }
        if (ownable.getOwnedID() == -1) {
            if (playerController.getPlayers()[playerIndex].playerAccount.getBalance() < ownable.getPrice()) {
                gui.showMessage( msg: "Du har ikke råd til denne grund");
            }
        }
    }
}

```

Der bliver også tjekket om spilleren selv ejer feltet, eller om man er landet på et felt, som er ejet af en anden spiller, hvis feltet er ejet af en anden spiller, skal man betale husleje til ejeren. Huslejen hentes i det respektive Field.

Lejens beløb bliver trukket fra spillerens konto, som er landet på feltet og der bliver efterfulgt sat samme leje ind på spilleren der ejer feltets konto.

I tilfælde af at en spiller lander på en andens spillerens felt og feltet er af brewery klassen, så bruges øjnene fra terningkastet til at gange med enten 100 eller 200. 100 hvis man kun ejer et brewery, 200 hvis man ejer 2 brewery felter. Dette tjekkes ved at enhver spiller har en tæller, der tracker spillerens antal ejede bryggerier.

Nok den største afvigelse vi har i vores spil er at hvis man vælger ikke at købe et felt som er ownable så sker der ikke noget. Vi valgte auktionsfunktionen fra, fra starten af da vi tvivlede på at vi ville kunne nå at have den med, hvilket viste sig at være det rigtige valg.

```
boolean checkBrewery = (ownable instanceof Brewery);
if (checkBrewery) {
    if (PlayerController.players[ownable.getOwnedID()].getBreweryOwned() == 1) {
        int toPay = dice.getTotal() * 100;
        PlayerController.players[playerIndex].playerAccount.setBalance(PlayerController.players[playerIndex].playerAccount.getBalance() - toPay);
        PlayerController.players[ownable.getOwnedID()].playerAccount.setBalance(PlayerController.players[ownable.getOwnedID()].playerAccount.getBalance() + toPay);
        gui.getUserButtonPressed( msg: playerController.getPlayers()[ownable.getOwnedID()].getName() + " ejer denne grund. Du har betalt: " + toPay, ...buttons: "Betal");
        updateView(PlayerController.players.length);

    } else if (PlayerController.players[ownable.getOwnedID()].getBreweryOwned() == 2) {
        int toPay = dice.getTotal() * 200;
    }
}
```

Efter den har checket om det var et ownable felt, og det viser sig man ikke er landet på et ownable felt. Vil man være landet på et start, parkering, tax, chance eller jail felt. Hvor der så er unikke metoder til de 4 slags felter.

CheckPlayerAccount

```
public void checkPlayerAccount(int playerIndex) {
    Player player = playerController.getPlayers()[playerIndex];

    while (player.playerAccount.getBalance() < 0) {
        String fallit = gui.getUserButtonPressed( msg: player.getName() + ", du har ikke penge til at betale husleje. Vælg nu enten at sælge huse/hoteller eller give op, i den forbindelse kalder vi på metoden fra gui'en "gui.getUserButtonPressed()" til at oprette de forskellige knapper, og dertil tilknyttes spillerens navn, med metoden "player.getName()".
```

Metoden starter med at kalde på spilleren fra playerControlleren, hvor vi derefter lavet en while-løkke med en betingelse hvis spillerens balance er under 0, får spilleren muligheden for enten, at pantsætte en af sine grunde, sælge huse/hoteller eller give op, i den forbindelse kalder vi på metoden fra gui'en "gui.getUserButtonPressed()" til at oprette de forskellige knapper, og dertil tilknyttes spillerens navn, med metoden "player.getName()".

Vi benytter en "switch" funktion til oprette de forskellige knapper, case 1 "pantsæt grund" vil spillerens balance først blive opdateret med "updateView()" metoden hvor i den metode vi kalder på spilleren fra playerControlleren, derefter opretter vi en if-sætning, en oversigt over spillerens konto hentes med "getBalance()" metoden, hvis den balance er større eller lig med 0, vil guien vise en tekst der fortæller "du har nu nok penge til at betale det du skylder" hvis spilleren har nok, vil

Iøkken brydes med "break" og spilleren kan fortsætte spillet efter have pantsættet sin grund, derefter bruger vi en "else-sætning" hvis spillernes balance er under 0, og en tekst "du har stadig ikke nok penge til at betale det du skylder" vil blive vist med "gui.showMessage()" metoden.

Med casen "sælg hus/hotel" vil spilleren kunne være i stand til at sælge sit hus, og løkken brydes med "break"

Case "Giv op"

Metoden "playerAccount.setBalance(0)" sætter spillerens balance til at være 0, og vi sætter metoden "setBankrupt()" til værende sand at spilleren er gået fallit, derefter bruger vi metoden "gui.showMessage()" til at vise "du er erklæret fallit og er nu ude af spillet"

Her ser man vores måde at sørge for at folk til sidst er tvunget til at give op. Vi har jo ikke nogen måde at se alle vores værdier i tal, så man bliver i princippet softlocked hvis man ikke har råd efter man har solgt/pantsat alt. Man har derfor også muligheden for at give op lige nå ens konto kommer i minus, hvis man kan se at der alligevel ikke er en chance for at man kan få nok penge.

Player pawn/playersbuyback

```
public void buyBackPawn (int fieldID, int value) {
    owns.add(fieldID);
    int rounded = (((value / 10) + 99) / 100) * 100;
    playerAccount.setBalance(playerAccount.getBalance() - value - rounded);
    pawned.remove((Integer)fieldID);
}
```

I denne metode har vi benyttet os af arraylisten som vi har lavet i klassen "Player".

I player pawns bliver listen dannet så den er lige så lang, som det antal ejendomme spilleren ejer. Vælger man at man gerne vil pantsætte en ejendom oprettes et loop bliver hvor hele listen af ejet grunde kan vælges. Vælger man en grund man gerne vil pantsætte vil et for loop køre hvori navnet på den valgte spiller er lig med den første plads i arrayet, hvis ikke kører den videre indtil de er lig med hinanden, så laves en boolean om hvorvidt han kan pantsætte i tilfælde af han har huse på.

Hvis han kan, kalder vi playerpawns og modtager pantsætnings værdien på konto.

Ejer id sættes til -2 så den er den er nu pantsat og man kan ikke modtage leje.

player buyback er næsten samme metode der er blevet opbygget, den største forskel er at vi har sat købsprisen til at koste 10% mere at købe den tilbage, også bliver tallet også afrundet til det næste 100 beløb.

Pull Card /Carddeckswitch

Pull card metoden fortæller brugeren de er landet på et chancefelt og at brugeren skal trække et chancekort. PullCard metoden benytter sig af DrawCard metoden, som blev forklaret i et tidligere afsnit. Den næste funktion der bliver henvendt til, er den metode vi har kaldt carddeckswitch. Det vigtige at forstå her, er at det ikke er en switch der bliver brugt, men at selve koden er bygget op på samme måde, som var det en switch. Metoden gennemgår via en boolean ligesom checksubclass metoden, hvor koden undersøger hvilket slags chancekort vi har med at gøre. Et eksempel kan være at man har trukket et kort der siger du skal rykke frem til nærmeste rederi. Inde i funktionen er der en while true funktion, der fjerner spillerens gui brik for boardet og

derefter rykker den et felt frem end til, at det felt den står på er et rederi. Grundet tid har vi valgt at simplificere kortet så den del, hvor man skal betale double, hvis ejet af en anden spiller er blevet skåret fra.

Vores move funktioner checker hvert felt om det passer med den adresse man er givet, så man bliver brugeren passere start og modtager 4000 kr. Undtagelsen er når spilleren skal rykkes direkte i fængsel, der bliver brikkens flytte direkte uden at passere start.

Trade

Først i metoden bliver der hentet relevante oplysninger, dvs en køber/byder og en sælger.

Dernæst laves et array, som indeholder antallet af grunde + antallet af navne. Dette bliver brugt i et “for” loop for at finde en spiller som der kan handles med, og for at finde de grunde den spiller ejer.

```
public void trade(int playerIDBuys, int playerIDSells) {
    int[] owns = new int[playerController.getPlayers()[playerIDSells].owns.size()];
    String[] names = new String[playerController.getPlayers()[playerIDSells].owns.size()];
    for (int i = 0; i < owns.length; i++) {
        Ownable ownable = (Ownable) boardController.getField()[playerController.getPlayers()[playerIDSells].owns.get(i)];
        owns[i] = playerController.getPlayers()[playerIDSells].owns.get(i);
        names[i] = ownable.getName();
    }
}
```

Disse oplysninger/metoder bliver efterfølgende brugt til at lave et nyt “for” loop, hvor der bliver lavet en boolean der spørger hvilken grund spilleren ønsker at købe og laver en liste over dem spilleren kan vælge imellem. Vi har lavet en boolean som hedder canTrade, som tjekker om der er bygget noget på en grund eller ej, hvis der er bygget noget på grunden bliver boolean'en false og beskeden nedenfor vil blive vist.

```
if (street.getHouseCount() < 0) {
    gui.showMessage( msg: "Du kan ikke købe denne grund før spilleren der ejer den har solgt sine bygninger.");
    canTrade = false;
}
```

Hvor imod hvis canTrade er true, bliver der kørt en int som gør at man kan give et bud på grunden. Efter at spilleren har angivet det ønskede bud til en anden spiller, tjekkes saldoen for at sikre spilleren har penge nok. Hvis saldoen er mindre, bliver den spiller som skal sælge spurgt om han/hun accepterer, afviser eller vil komme med et modbud.

Hvis spilleren ønsker at sælge:

```
playerController.trade(playerIDSells, playerIDBuys, owns[ownableChosen], offer);
tradeOwnable.setOwnedID(playerIDBuys);
updateView(PlayerController.players.length);
tradeGui_ownable.setOwnerName(playerController.getPlayers()[playerIDBuys].getName());
tradeGui_ownable.setBorder(playerController.colors[playerIDBuys]);
```

På billedet ovenfor ses det at informationen på de forskellige ting bliver indlæst, dvs id'et på den spiller der vil sælge og på den der vil købe, id'et og navnet på den grund der er blevet valgt og tilbuddet. Derefter vil OwnerID blive ændret fra den ene spiller til den anden, efterfulgt af en update på selve brættet, så det der vises stemmer overens med det der er blevet handlet. Da der gælder andre regler for Shipping og Brewery, bliver det også tjekket for og hvis det er en af delene skal der trækkes en af dem fra hos sælgeren og lægges en til hos køberen, for at kunne udnytte fordelen ved at have flere.

```

boolean checkBrewery = (tradeOwnable instanceof Brewery);
if (checkBrewery) {
    playerController.getPlayers()[playerIDSells].setBreweryOwned(playerController.getPlayers()[playerIDSells].getBreweryOwned() - 1);
    playerController.getPlayers()[playerIDBuys].setBreweryOwned(playerController.getPlayers()[playerIDBuys].getBreweryOwned() + 1);
}

```

Hvis spilleren ønsker at komme med et modbud:

Der vil først blive lavet en int som hedder counterOffer, som gør at sælger kan lave et bud og derefter bliver der kørt det, samme som hvad sælger havde accepteret budet, bare omvendt.

Hvis man ikke har penge nok til at købe med det bud man selv/ sælger har givet, vil man få besked om dette.

```

        } else {
            gui.showMessage( msg: PlayerController.players[playerIDBuys].getName() + " har ikke nok penge til at gennemføre det bud");
            break;
        }
    }
} else {
    gui.showMessage( msg: "Du har ikke nok penge til at lave dette bud");
}

```

Man kan kun komme med et modbud engang.

Build

```

| public void build(int playerIndex) {
    int counter = 0;
    int i = 0;
    while (i < playerController.getPlayers()[playerIndex].owns.size()) {
        Ownable ownable = (Ownable) boardController.getField()[playerController.getPlayers()[playerIndex].owns.get(i)];
        boolean checkStreet = ownable instanceof Street;
        if (checkStreet) {
            counter++;
        }
        i++;
    }
    int[] ownsStreets = new int[counter];

```

Metoden begynder at tælle hvor mange felter spilleren ejer: tællerne sættes til at være 0, dernæst bygger vi et ”while-loop” der kører på felterne, og tæller op alle felter ejet af spilleren.

-ID’erne overføres til en array af integers: de antal ejet grunde af spilleren tildeles til spillerens arrayliste i en rækkefølge ved brug af en while-løkke,

”ownsStreets[place]=playerController.getPlayer()[playerIndex].owns.get(i2)” koden her benytter et array til at tildele ”owns” til spillerens ejet bygninger.

Disse overføres til en ny array af Integers hvorefter man også laver en array af Strings der samler gadernes navne, derefter tages et input fra en drop-down-liste der lader spilleren vælge hvor han vil bygge.

Sell house

Sell house metoden fungerer nærmest på den samme måde som build metoden, hvor den tæller først alle street som spilleren ejer, dernæst streets ID, hvor man så har en anden metode for at

sælge i stedet for at bygge til sidst.

```
Street sellHouseOnStreet = (Street) boardController.getField()[canBuild[houseChosen]];
boolean equalBuild = true;
for (int j = 0; j < canBuild.length && equalBuild; j++) {
    Street compareStreet = (Street) boardController.getField()[canBuild[j]];
    if (sellHouseOnStreet.getMainColor().equals(compareStreet.getMainColor())) {
        if (sellHouseOnStreet.getHouseCount() > compareStreet.getHouseCount()) {
            equalBuild = false;
            break;
        } else {
            equalBuild = true;
        }
    }
}
```

Test

Doublerent/byg

Den første test vi har foretaget os er en junit test der vil påpege, om vores kode vil tage dobbelt husleje, når alle af samme farve/kategori ejes. Et eksempel vil være hvis en spiller ejer både hvidovrevej og rødovrevej, så vil spilleren eje alle af denne kategori, og derfor skal den normale husleje som er 50 for hver af grundenedobles til 100, så vi asserter vores expected rent til at være 100.

Den nederste del af assertTrue testen tester om vores grunde kan få lov til at bygge huse på sig. Huse er kun tilladt ved at man ejer alle grunde af samme kategori, og fordi spillerID(0) er ejer begge de 2 grunde ligesom ved dobbelt leje har han mulighed for at bygge huse på grunden og dermed er den true. Nederste linje vises grønningen alene og dermed vil vi forvente at man ikke kan bygge på den.

Testen viser defor at være passed.

```
10
11
12 @Test
13 void doubleRentTest() {
14     BoardController boardController = new BoardController();
15     Street rodovrevej = (Street) boardController.getField()[1];
16     Street hvidovrevej = (Street) boardController.getField()[3];
17     Street groningen = (Street) boardController.getField()[24];
18     rodovrevej.setOwnedID(0);
19     hvidovrevej.setOwnedID(0);
20     boardController.checkFields();
21     assertEquals( expected: 100, rodovrevej.currentRent);
22     assertEquals( expected: 100, hvidovrevej.currentRent);
23     assertEquals( expected: 400, groningen.currentRent);
24     assertTrue(rodovrevej.isCanBuild());
25     assertTrue(hvidovrevej.isCanBuild());
26     assertFalse(groningen.isCanBuild());
27 }
```

BuyTest.doubleRentTest ×

Tests passed: 1 of 1 test – 2s 517 ms

Test Results

- BuyTest
- doubleRentTest()

C:\Users\solom\.jdks\liberica-14.0.2\bin\java.exe ...

Terningtest 1

I terning test har vi valgt at kaste vores terning 100 gange, hvor vi tester både summen af begge terninger og hver enkelt af dem. Det vi tester er deres øjne, fx skal summen af begge terninger være mellem to til tolv, hvis det er så vil der blive printet "Good Dice", hvis det ikke er, vil der blive printet "Bad Dice". For det eneste terning, skal øjne være mellem et til seks, og principippet er det samme som summen af to terning. Der kan ses på billedet nedenfor at testen er lykkes, da der blev printet "Good Dice" ved dem allesammen, og der er blevet sæt flueben ved testen.

The screenshot shows an IDE interface with the following details:

- Project Structure:**
 - Root package: Chancefield
 - Sub-package: test.java.com.company
 - Files: BuyTest, DiceTest (highlighted in blue)
 - Target folder: target
 - External Libraries: Maven dependencies for various JUnit and API Guardian versions.
- Code Editor:** Shows the `DiceTest` class with the following code:

```

    @Test
    void dice() {
        Dice dice = new Dice();
        int t = 0;
        while (t < 100){
            dice.roll();
            dice.getTotal();
            if (dice.getTotal() >= 2 && dice.getTotal() <= 12){
                System.out.println("Good Dice");
            }
            else {
                System.out.println("BAD DICE");
            }
            if (dice.die1 >= 1 && dice.die1 <= 6) {
                System.out.println("Good Die1");
            }
            else {
                System.out.println("BAD DIE1");
            }
            if (dice.die2 >= 1 && dice.die2 <= 6){
                System.out.println("Good Die2");
            }
            else {
                System.out.println("BAD DIE2");
            }
            t++;
        }
    }

```
- Run Tab:** Shows the run configuration for `DiceTest.dice`. It indicates 1 test passed in 39 ms, with the command being `"C:\Program Files\Java\jdk-14.0.2\bin\java.exe" ...`. The results show three lines of output: `Good Dice`, `Good Die1`, and `Good Die2`.

Terningtest 2:

Testen på billedet nedenfor viser hvor mange gange de forskellige slag vil blive slået over 1000 kast. Testen blev gennemført med en normalfordeling der kan ses i graf nedenfor.

The screenshot shows an IDE interface with two main panes. The left pane displays a file tree and the right pane shows the corresponding code.

File Tree:

- src
 - Brewery
 - Chancefield
 - Field
 - GoToJail
 - Jail
 - Ownable
 - Parking
 - Shipping
 - Start
 - Street
 - TaxField
- test
 - java
 - com.company
 - BuyTest
 - DiceTest
- target
- 22_final_2.0.iml
- pom.xml

External Libraries:

- < 14 > C:\Program Files\Java\jdk-14.0.2
- Maven: diplomittdu:matadorgui:3.1.7
- Maven: org.apiguardian:apiguardian-api:1.1.0
- Maven: org.junit.jupiter:junit-jupiter:5.7.0
- Maven: org.junit.jupiter:junit-jupiter-api:5.7.0
- Maven: org.junit.jupiter:junit-jupiter-engine:5.7.0
- Maven: org.junit.jupiter:junit-jupiter-parameterized:5.7.0
- Maven: org.junit.platform:junit-platform-launcher:1.7.0

Code (DiceTest.java):

```

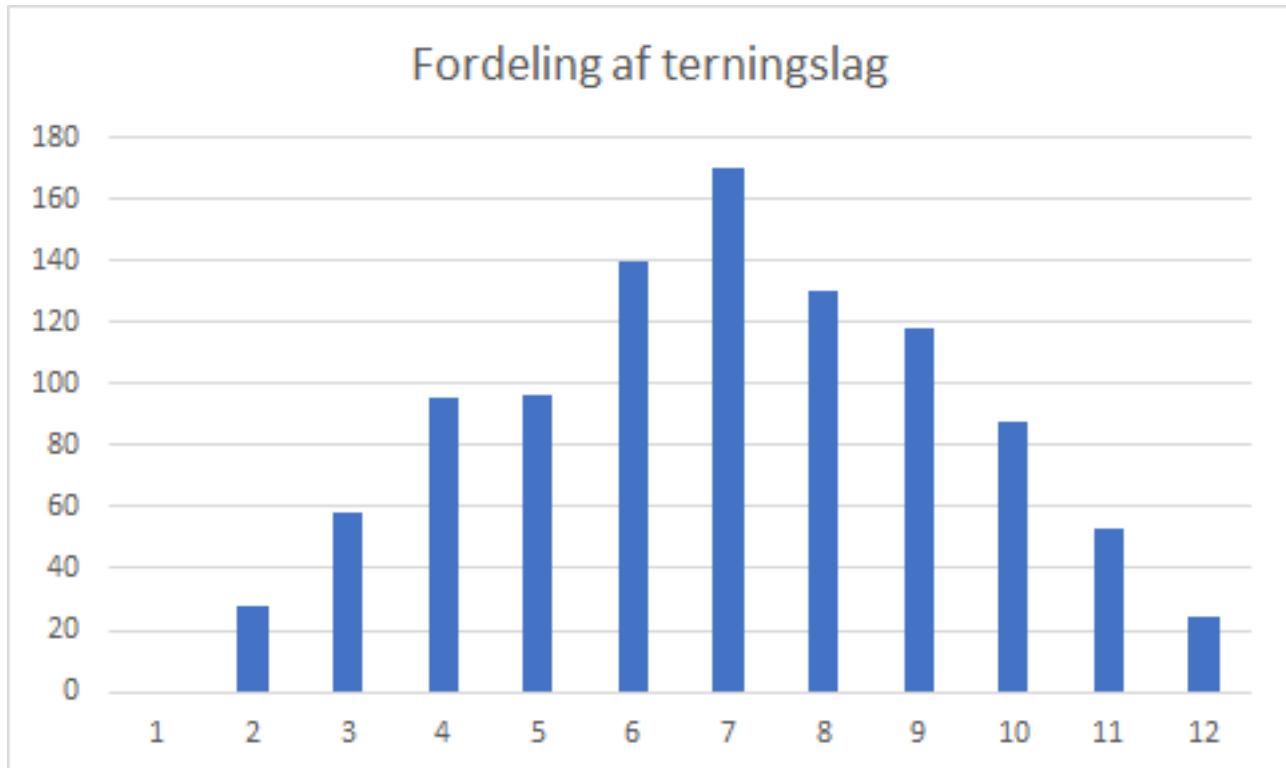
    ...
    }

    @Test
    void diceCount() {
        Dice dice = new Dice();
        int one = 0;
        int two = 0;
        int three = 0;
        int four = 0;
        int five = 0;
        int six = 0;
        int seven = 0;
        int eight = 0;
        int nine = 0;
        int ten = 0;
        int eleven = 0;
        int twelve = 0;
        for (int i = 0; i < 1000; i++) {
            dice.roll();
            switch (dice.getTotal()) {
                case 1:
                    one++;
                    break;
                case 2:
                    two++;
                    break;
                case 3:
    
```

Run Results:

Tests passed: 1 of 1 test – 22 ms

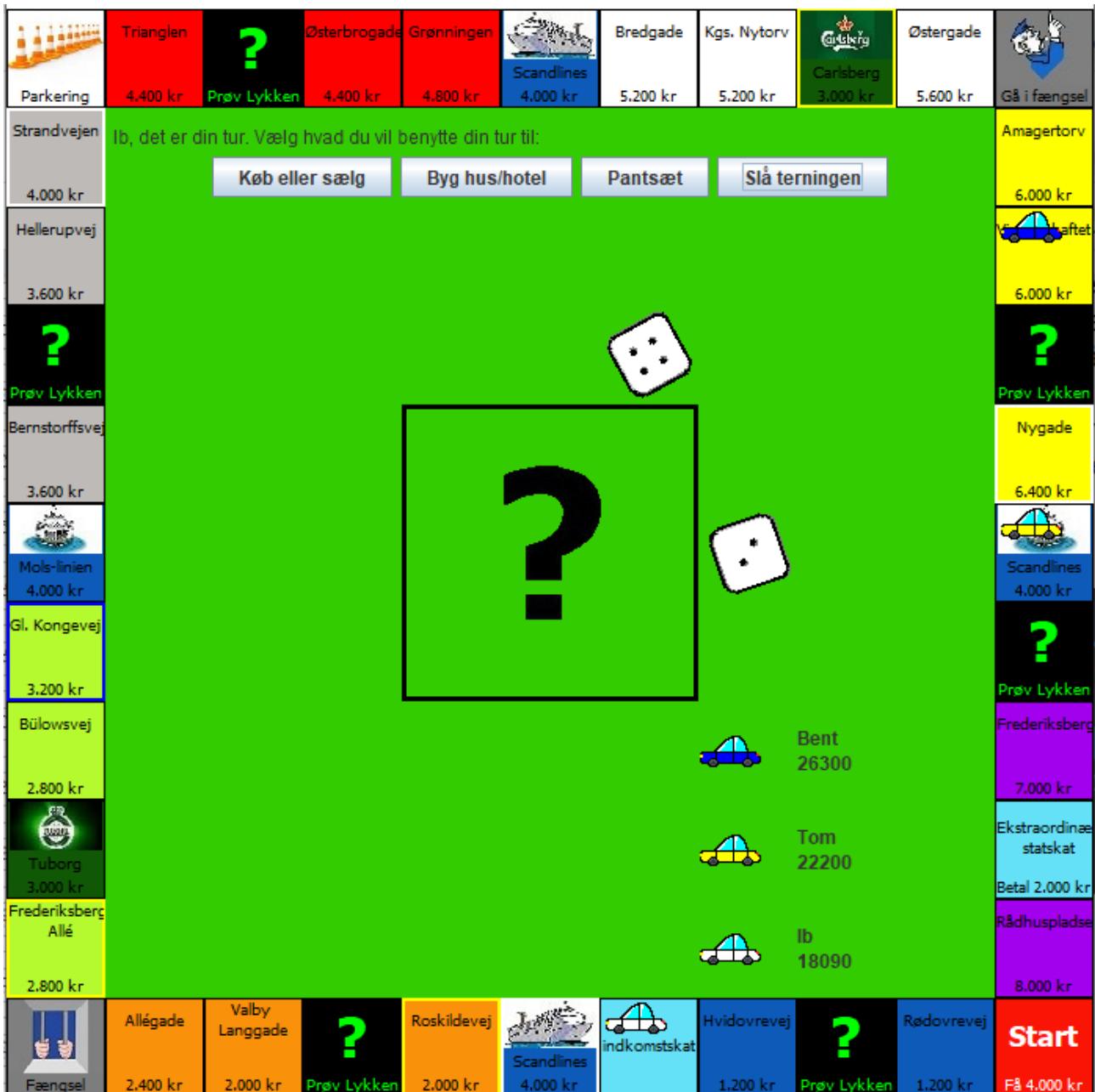
Test	Time	Output
Test Results	22 ms	2 slået: 28
DiceTest	22 ms	3 slået: 58
diceCount()	22 ms	4 slået: 95 5 slået: 96 6 slået: 140 7 slået: 170 8 slået: 130 9 slået: 118 10 slået: 88 11 slået: 53 12 slået: 24



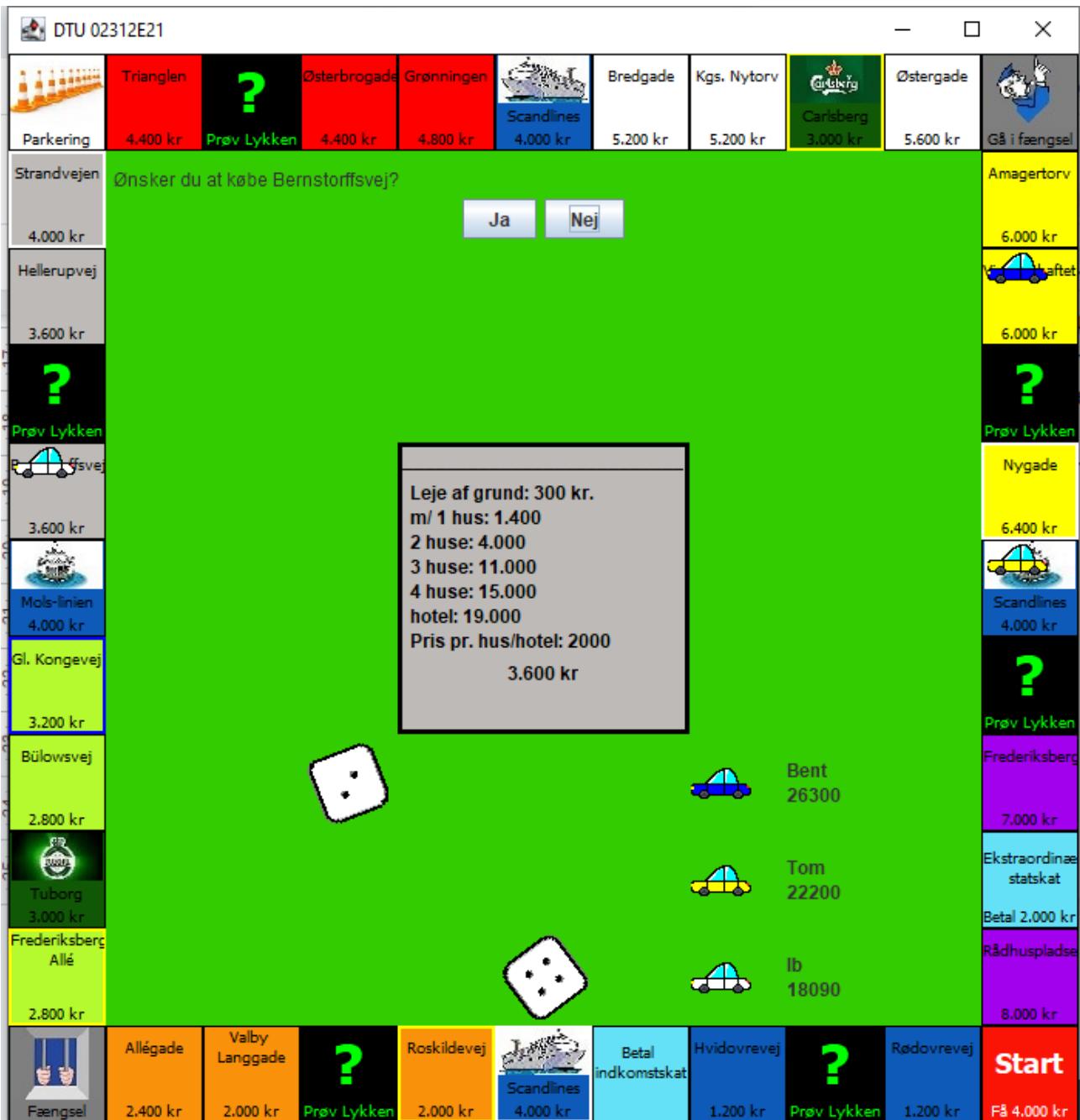
Test case scenarier:

Test case 1

Her vil vi gå igennem scenariet hvor en spiller rykker hen på et felt som kan købes og som ikke er ejet endnu. Dette køber han og vi vil se ejerskabet blive skrevet på grunden samt at den nuværende leje vil blive vist. Her ligger vi også mærke til om han mister det antal penge som grunden koster. I de her scenarier vil der først være blevet spillet 3 runder så vi er midt inde i spillet:



Her har vi spillebrættet hvor alle bilerne har fået lov til at have deres tur 3 gange. Det er ibr tur kan man se. Alle de grunde med farvede kanter er blevet købt og nu rykker vi Ib



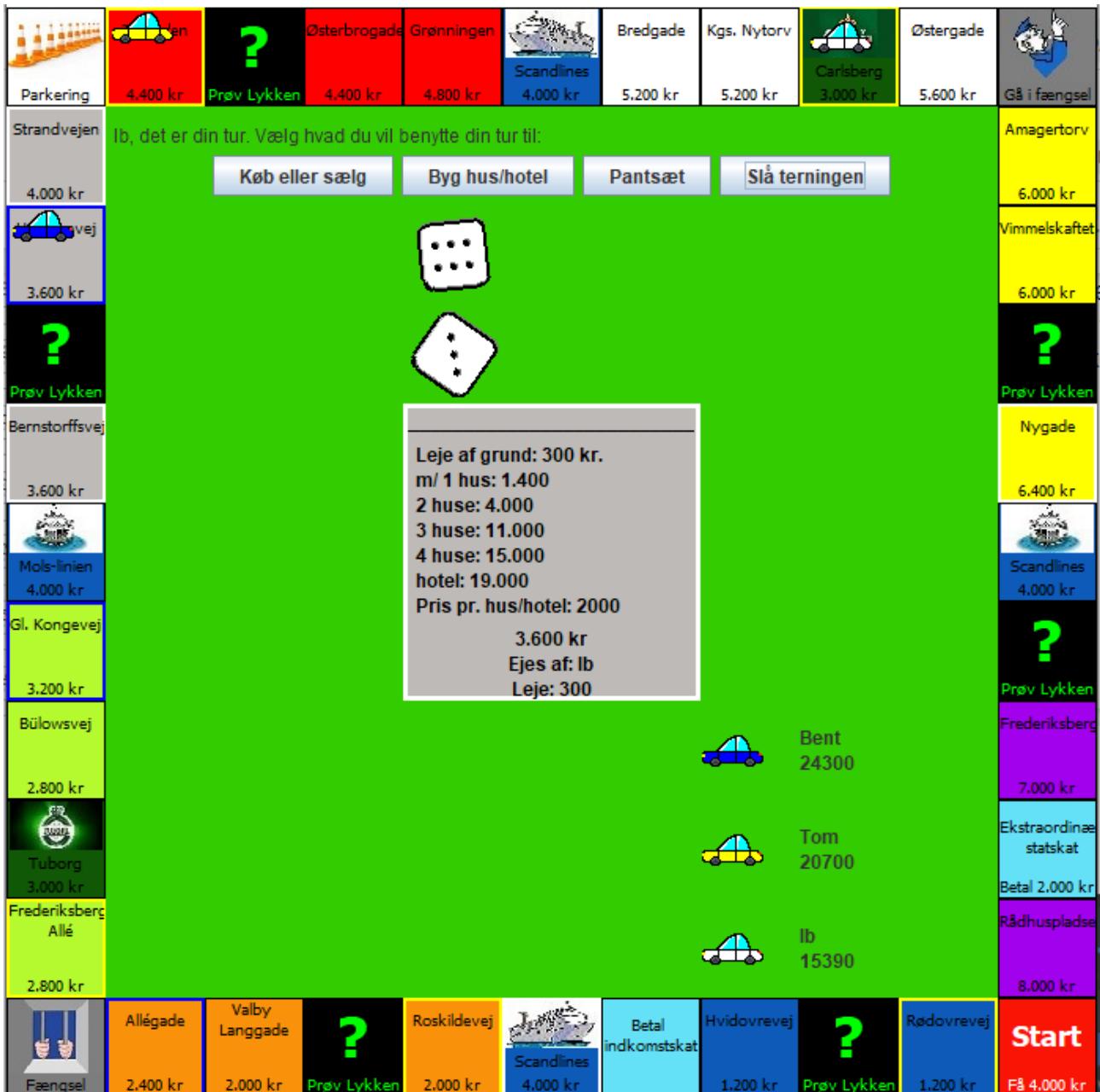
Ib i den hvide bil er nu landet på Bernstorffsvej og vælger at købe den. Hans konto inden han køber den er 18090 kr og prisen på vejen er 3600 kr



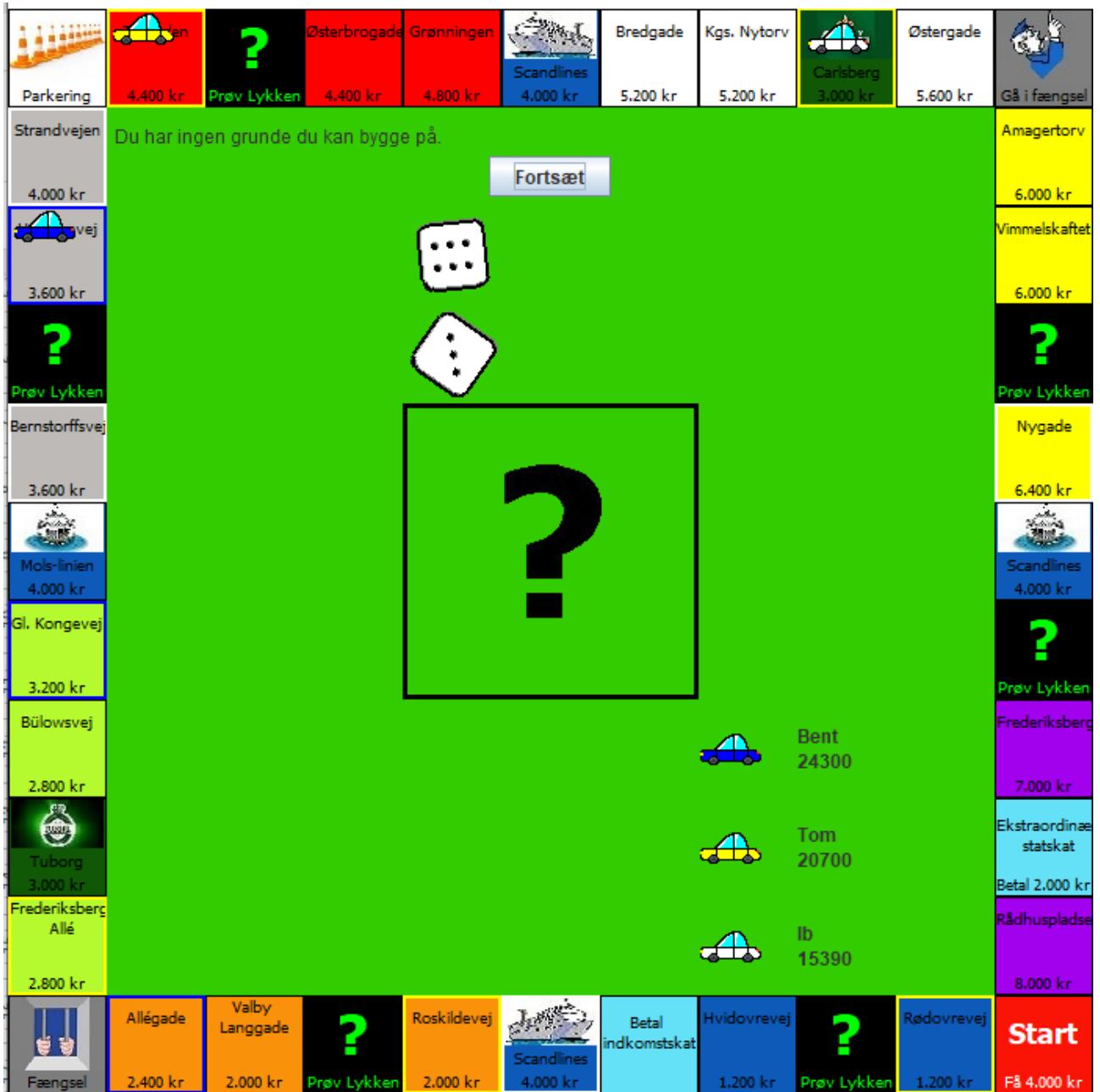
Ib har nu købt grunden og rammen er blevet hvid som hans bil. Der står han ejer den, der står hvad lejen er på den lige nu og vi kan se på hans konto at han har mistet præcis 3600kr

Test case 2

Her vil vi vise at lejen bliver fordoblet når man ejer alle af en farve, man kun kan købe huse hvis man ejer alle af en farve og at man kun kan bygge ligeligt. Så vi kører spillet rundt indtil alle af en farve, er blevet købt.



Her ser vi at alle af de grå felter er blevet købt men Ib ejer kun 2 af dem, så nu køber Ib den grund af Bent og så ser vi om lejen bliver fordoblet. Men først tjekker vi om Ib kan bygge nogle huse endnu:



Det kunne han ikke det han ikke ejer alle af en farve endnu:



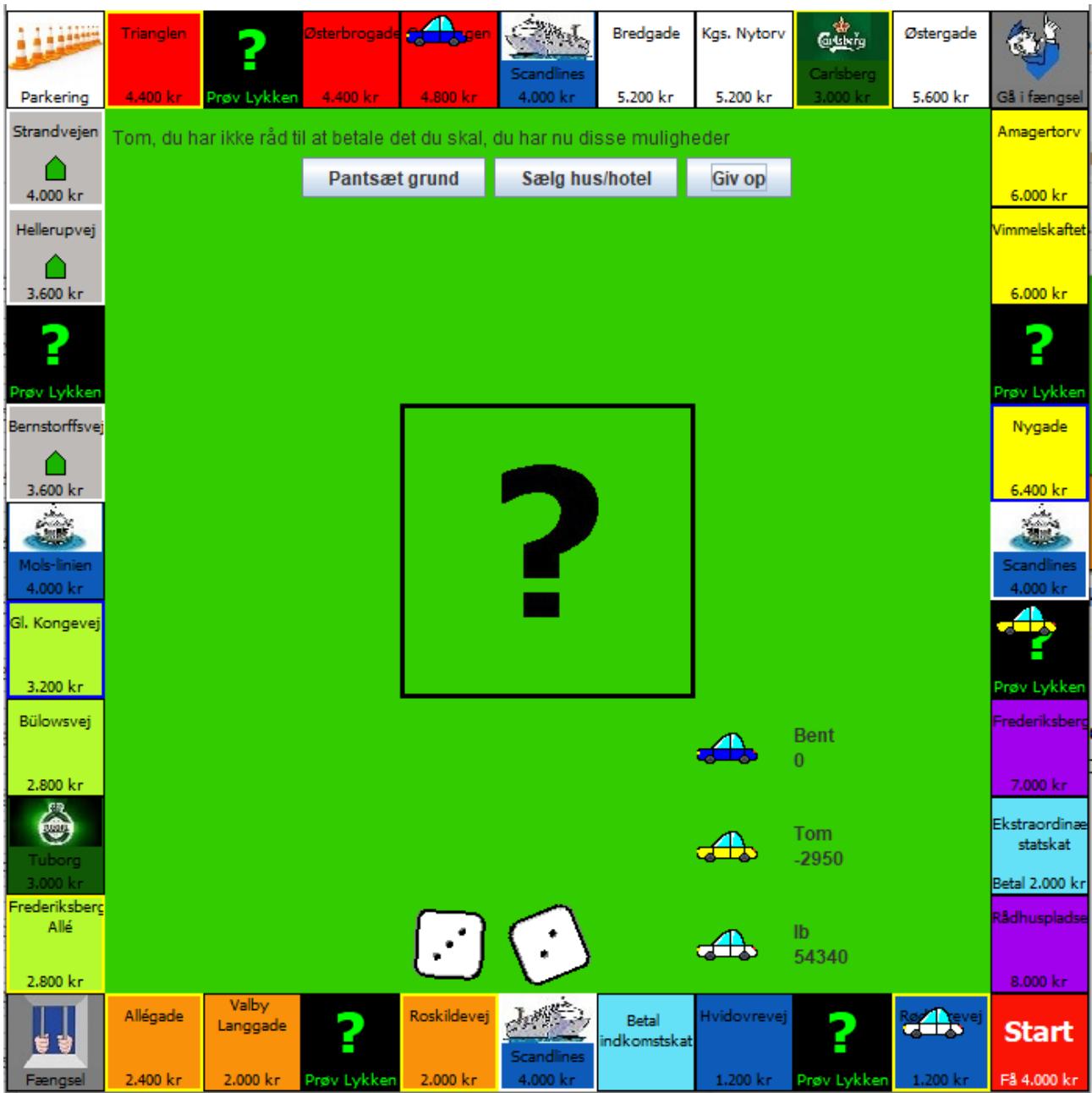
Nu ejer Ib alle de grå og vi kan se at lejen er blevet fordoblet. Nu prøver vi så at købe et hus på Bernstorffsvej. Vi ser også hvad der sker når man prøver at bygge yderligere på den samme grund.



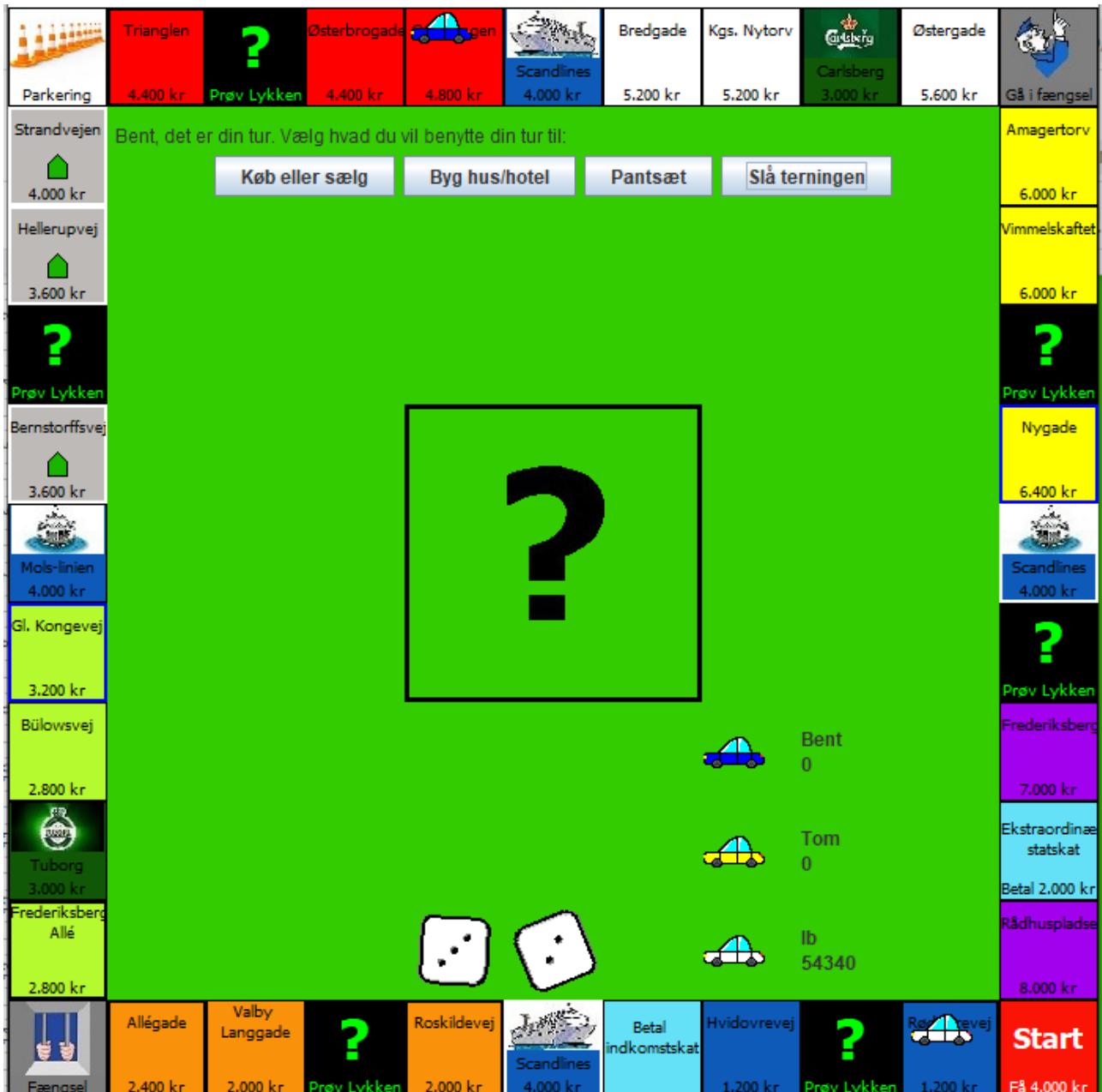
Her lægger vi mærke til at der er blevet bygget et hus på Bernstorffsvej og lejen er blevet højere. Vi prøvede så også at bygge et til hus på Bernstorffsvej men fik beskedden ovenfor.

Test case 3

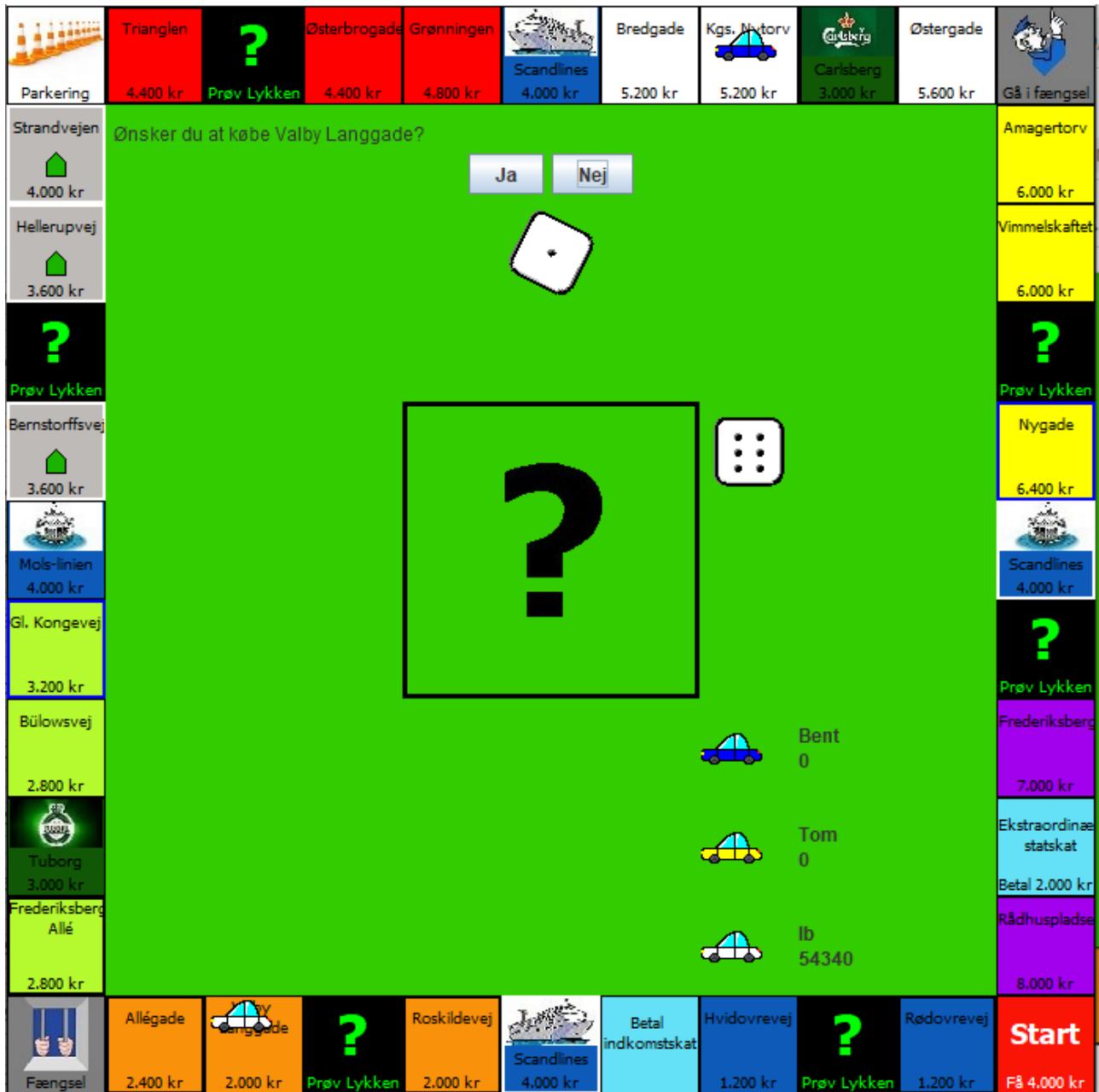
I denne sidste test case vil vi teste om hvorvidt der bliver fundet den rigtige vinder når alle andre spillere har giver op, og hvorvidt når en spillet bliver sprunget over når man har givet op.



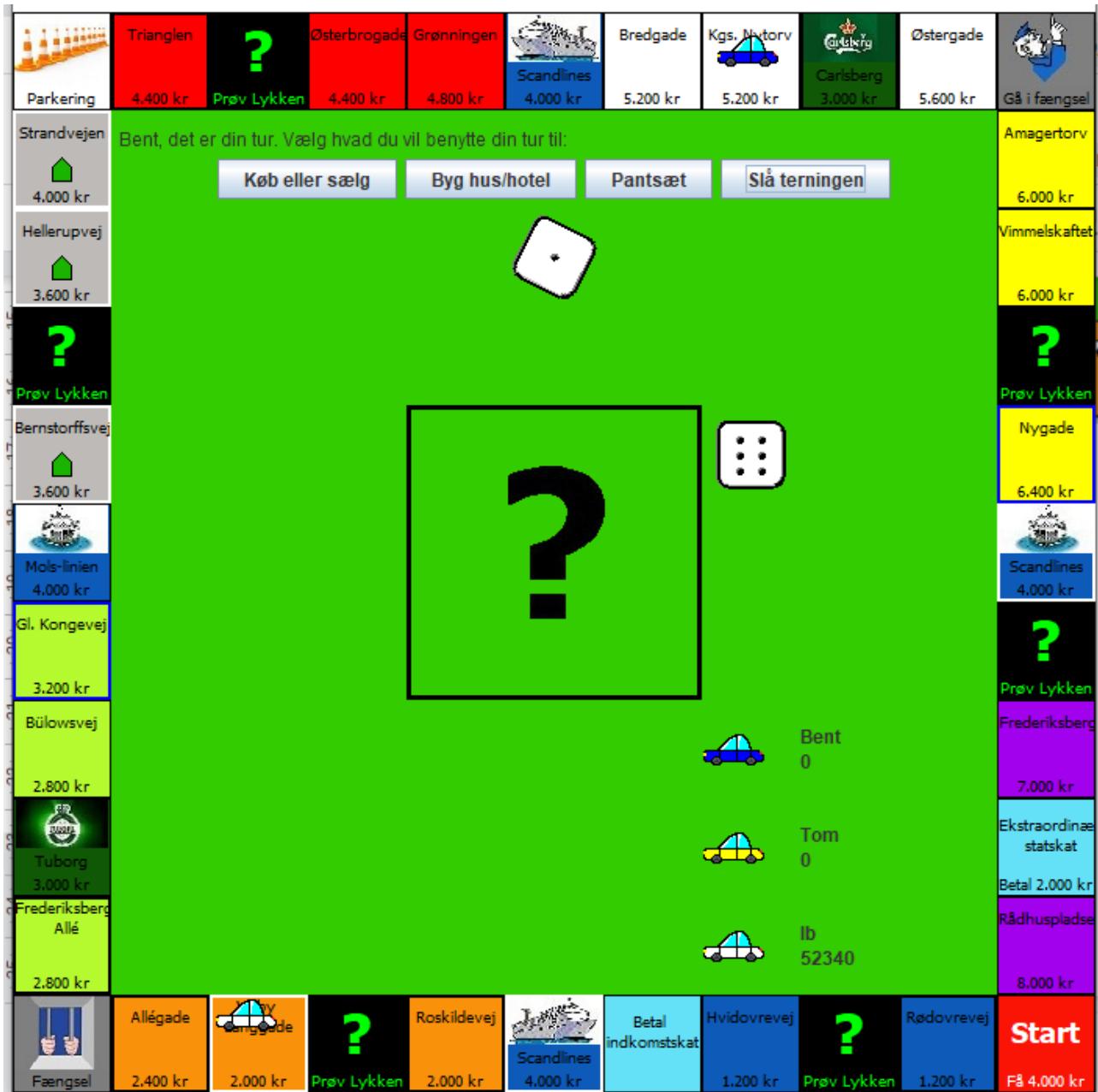
Tom har i det her scenarie lige trukket et chancekort der bad ham betale 3000 kr hvilket fik ham i -2950 kr. Vi prøver at give op og ser at alle hans grunde bliver neutrale igen og kan købes. Han burde også blive sprunget over fra nu af og hans bil burde blive fjernet.



Vi kan se at alle de grunde der før var gule nu, er default igen og hans bil er blevet fjernet fra brættet



Vi ser her at Ib lige har slået og nu står på Valby Langeade, den køber vi og så ser hvis tur det er lige efter.



Det blev Bents tur. Nu prøver vi et spil med bare 2 spillere så den første der taber burde erklære en vinder.

	Trianglen	?	Østerbrogade	Grønningen		Bredgade	Kgs. Nytorv		Østergade	
Parkering	4.400 kr	Prøv Lykken	4.400 kr	4.800 kr	Scandlines 4.000 kr	5.200 kr	5.200 kr	Carlsberg 3.000 kr	5.600 kr	Gå i fængsel
Strandvejen	4.000 kr	2, du har ikke råd til at betale det du skal, du har nu disse muligheder								
		Pantsæt grund	Sælg hus/hotel	Giv op						
	3.600 kr									
?	Prøv Lykken									?
Bernstorffsvej	3.600 kr									Prøv Lykken
Mols-linien	4.000 kr									Nygade
Gl. Kongevej	3.200 kr									6.400 kr
Bülowsvej	2.800 kr									Scandlines 4.000 kr
Tuborg	3.000 kr									?
Frederiksberg Allé	2.800 kr									Prøv Lykken
	Allégade	Valby Langgade	?	Roskildevej		Betal indkomstskat	Hvidovrevej	?	Rødovrevej	Start
Fængsel	2.400 kr	2.000 kr	Prøv Lykken	2.000 kr	Scandlines 4.000 kr	1.200 kr	Prøv Lykken	1.200 kr	Få 4.000 kr	

Her har vi gul 2 som nu giver op.

De har solgt nogle gamle møbler på auktion, modtag 1000kr

2 -300
1 48900

Parkering	Trianglen	Prøv Lykken	Østerborgade	Grønningen	Scandlines 4.000 kr	Bredgade	Kgs. Nytorv	Carlsberg 3.000 kr	Østergade	Gå i fængsel
Strandvejen	4.000 kr	Vinderen er nu fundet! Det blev: 1 Tillykke!							Amagertorv	
Hellerupvej	3.600 kr					OK			Vimmelskaftet	
Prøv Lykken									Prøv Lykken	
Bernstorffsvej	3.600 kr								Nygade	
Mols-linien	4.000 kr								6.400 kr	
Gl. Kongevej	3.200 kr								Scandlines 4.000 kr	
Bülowsvej	2.800 kr								Prøv Lykken	
Tuborg	3.000 kr								Frederiksberg	
Frederiksberg Allé	2.800 kr								7.000 kr	
Fængsel	Allégade	Valby Langgade	Prøv Lykken	Roskildevej	Scandlines 4.000 kr	Betal indkomstskat	Hvidovrevej	Prøv Lykken	Rødovrevej	Start
	2.400 kr	2.000 kr		2.000 kr					1.200 kr	Få 4.000 kr

Så kom denne meddelelse frem og det var den rigtige vinder der blev fundet

Code Coverage

Coverage: Main ×

100% classes, 81% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %
ChanceCard	100% (11/11)	80% (20/25)	94% (107/113)
com.company	100% (6/6)	90% (50/55)	77% (654/849)
Fields	100% (13/13)	100% (40/40)	86% (321/373)

Coverage: Main ×

100% classes, 94% lines covered in package 'ChanceCard'

Element	Class, %	Method, %	Line, %
CardDeck	100% (1/1)	85% (6/7)	97% (72/74)
ChanceCard	100% (1/1)	100% (1/1)	100% (4/4)
GetOutOfJailC...	100% (1/1)	100% (1/1)	100% (3/3)
GoToJailCard	100% (1/1)	50% (1/2)	75% (3/4)
IncreasePrice	100% (1/1)	33% (1/3)	66% (4/6)
MoneyFromPl...	100% (1/1)	100% (2/2)	100% (4/4)
Move	100% (1/1)	50% (1/2)	75% (3/4)
MoveToShippi...	100% (1/1)	100% (1/1)	100% (2/2)
MovetoSpecific	100% (1/1)	100% (2/2)	100% (4/4)
PayMoney	100% (1/1)	100% (2/2)	100% (4/4)
ReceiveMoney	100% (1/1)	100% (2/2)	100% (4/4)

Coverage: Main

Element	Class, %	Method, %	Line, %
Account	100% (1/1)	100% (3/3)	100% (6/6)
Consol	100% (1/1)	94% (16/17)	75% (571/755)
Dice	100% (1/1)	100% (3/3)	100% (8/8)
Main	100% (1/1)	100% (1/1)	100% (4/4)
Player	100% (1/1)	85% (17/20)	81% (39/48)
PlayerController	100% (1/1)	90% (10/11)	92% (26/28)

Element	Class, %	Method, %	Line, %
Board	100% (1/1)	100% (2/2)	100% (44/44)
BoardController	100% (1/1)	100% (4/4)	76% (157/204)
Brewery	100% (1/1)	100% (1/1)	100% (3/3)
Chancefield	100% (1/1)	100% (2/2)	100% (5/5)
Field	100% (1/1)	100% (1/1)	100% (3/3)
GoToJail	100% (1/1)	100% (2/2)	100% (4/4)
Jail	100% (1/1)	100% (1/1)	100% (2/2)
Ownable	100% (1/1)	100% (6/6)	100% (12/12)
Parking	100% (1/1)	100% (2/2)	100% (5/5)
Shipping	100% (1/1)	100% (3/3)	85% (18/21)
Start	100% (1/1)	100% (2/2)	100% (5/5)
Street	100% (1/1)	100% (12/12)	96% (56/58)
TaxField	100% (1/1)	100% (2/2)	100% (7/7)

I billederne ovenfor kan man se code coverage for et spil der blev gennemført med 2 spillere som blev spillet til ende. Her kan vi se at alle vores klasser bliver brugt og at de fleste af vores linjer og metoder også bliver brugt. Vi har nogle klasser hvor metode dækningen er lidt bagefter som fx i 'IncreasePrice' men det ville højest sandsynligt være fordi chancekortet der relaterer til den klasse aldrig blev trukket i spillet, eller ikke var relevant da den blev trukket.

Vi kunne nok have haft en større % coverage med et spil med 6 spillere som blev spillet indtil der blev fundet en vinder.

Brugertest:

Vi lavede en brugertest hvor en af vores gruppemedlemmer spillede med hans ven og en hvor en af vores gruppemedlemmer spillede med sin kæreste. Det er desværre svært i coronatiderne at få mulighed for brugertest med lige hvem man ville men vi har fået fat i nogle af de nærmeste. Det kunne jo fx være fedt at få fat i en ældre person som ikke havde så meget styr på computere: I den første test med et gruppemedlems ven, er personen der tester 21 år med okay forståelse for computere. Hans kommentarer var følgende: "Spillet fungerer umiddelbart fint og var rimelig nemt at gå til. Dog var der ting som ikke lige var klart til at starte med som at man kunne trykke på hvert felt for at få udvidede information om huspriser og leje. De ting havde nok taget mig lidt tid at opdage hvis ikke at du (gruppemedlemmet) var ved siden af mig. Jeg ville nok heller ikke have opdaget at lejen blev dobbelt efter at man ejer alle af en farve lige med det samme".

Sofus ung fyr på 22 år, studerende på CBS og har en god forståelse for computer. Sofus synes at spillet i det store hele fungerer som matador skal. Det fungerer godt: 1) turen rundt om pladen, 2) køb af grunde, 3) godt at der er beskeder som guider en igennem spillet, det gør det let at forstå, 4) godt at der er en pantsæt funktion, 5) virker godt at turen går videre automatisk, når man har slået og køb/betalt/afvist den grund man lander på.

Det fungerer mindre godt: 1) størrelsen på plade, det kunne være godt hvis den kunne komme i fuld skærm, så man bedre kan se, 2) manglende oversigt over hvilke grunde man ejer, 3) irriterende at beskeden fra chancekortet bliver stående i midten af pladen, selv når turen er forbi, 4) ikke umiddelbart oplagt hvad man skulle gøre for at se lejen.

Konfigurations styling

I skal dokumentere platformens dele med versionsnummer så den kan genskabes til senere brug.

Windows

Windows 10, 64-bit operativsystem/mac OS catalina

IntelliJ

2020.2.4

Pom xml

3.1.7

[En guide til når man modtager filen og vil køre den:](#)

Når man modtager zipfilen downloader man den til et sted hvor man kan finde den igen. Så går man ind på IntelliJ's startside (hvor man kan create new project eller import project osv). Så trykker du på "Import Project" og finder din downloadede zipfil og åbner den. Derefter popper der nogle beskeder. På dem skal du bare trykke næste og acceptere alt hvad den beder om indtil den til sidst lader dig trykke "Finish".

Nu burde du have projektet åbnet på IntelliJ. For at åbne klasserne skal du trykke på pilen ved siden af mappen med navnet 22_final, så klikke på pilen ved siden af mappen src. Herfra kan du se pdffilen hvor vores rapport ligger. Du burde herfra også kunne se alle vores klasser ligge ved siden af. Dem dobbeltklikker du så for at åbne dem.

For at køre programmet skal du så åbne den klasse der hedder main og trykke på den grønne pil der står ud for public class Main på linje 1.

Her kan du støde ind i et problem hvor den siger at du bruger en for gammel version af SDK/JDK. Hvis det er problemet, skal du trykke på File->Project Structure og så tjekker du om du har en version som er 12 eller højere, oppe i højre hjørne lige under et + og et -. Hvis ikke skal du gå på google og søge "java jdk" og gå ind på det første link og downloade en nyere JDK og følge guiden på hjemmesiden. Så går du ind på Project structure igen og trykker på + og finder din downloadede JDK og apply den så fjerner du din gamle version så kun den nye står tilbage. Så trykker du apply.

Næste gang du så kører programmet kan du risikere den stadig siger fejl og der kommer en meddeelse hvor den beder dig opdatere. Tryk på den, lad den opdatere og så burde det hele virke.

Projektplanlægning

IntelliJ

IntelliJ bliver benyttet til at kode i Java. Derudover tilbyder IntelliJ en god integration af Github. Dette må anses som værende svært nødvendigt med et multiplum af mennesker arbejdende på dette projekt.

Maven

Maven er et værktøj som primært bliver brugt til java projekter. Den kan bruges til at hente biblioteker udefra, som har færdig kodning i sig. Vi benytter maven til at hente prædefineret kodning til GUI.

Github

Github bliver benyttet i projektet til at give flere personer mulighed for at arbejde på det samme projekt. Derudover giver det mulighed for versionsstyring i forbindelse med gendannelse af ældre versioner samt at kunne holde styr på hvem der laver hvad. Derudover giver det også mulighed for branching.

Use case

Vi har benyttet use case til at give en oversigt af vores aktører, og hvad systemet skal kunne, dermed er den også med at illustrere forholdet mellem aktører og casene.

Domain model

Vi har lavet en domain model, da den giver et overordnet blik over vores klasser og hvilken attributter de har. Derudover kan man se relationer mellem klasserne. Den er forholdsvis simpel at se på, derfor har vi valgt at lave den, da den vil give kunden et overblik over vores system.

Flowchart

Et flowchart er en visuel præsentation af hvordan koden kommer til at fungere. Dette viser step-by-step beslutningsprocessen for programmet og de parametre den arbejder indenfor. Dette laves i høj grad for at simplificere programmeringen i forbindelse med implementeringen.

System sekvensdiagram

Et systemsekvensdiagram bliver brugt til at vise et bestemt scenerie i en use case, de situationer som eksterne aktører skaber samt deres rækkefølge. Dette diagram bliver ofte brugt til at visualisere den succesfulde use case eller eventuelle komplikerede alternativer.

I dette diagram er der betydelig fokus på hvordan forskellige klasser og metoder interagerer med hinanden og hvordan grænserne krydses. Det er derfor også en effektiv metode at vise forskellige klassers metoder benyttes og i hvilke sammenhænge.

Design klassediagram

Vi har lavet en design klassediagram til kunden, da den giver et overblik over vores system og hvilken/hvor mange klasser vi har, dermed kan man også se deres attributter og metoder.

Sekvensdiagram

Et sekvensdiagram bruges til at vise samarbejdet mellem objekter. I et sekvensdiagram kan man se når en aktion bliver lavet fra et objekt til et andet og om den aktion kræver at det ene objekt kan arbejde videre eller er nødt til at vente på et respons efter dens aktion.

Kunden vil kunne bruge dette til nemt at få et overblik over hvilke objekter vi arbejder med, hvordan de arbejder sammen, rækkefølgen de arbejder i og afhængigheden mellem objekterne.

JUnit

Bruges til at forstå og teste logikken vi har lavet inde i intellij.

JUnit er et testprogram man specifikt bruger til programmeringssproget java. JUnit er en del af gruppen JUnit, som er en samling af andre test frameworks. Alle frameworks der ligger i gruppen xUnit, bliver navngivet efter, hvad forbogstavet for programmeringssproget er (et stort bogstav), efterfulgt af Unit (stort U)

Gui

Gui står for graphical user interface, hvilket vi også benytter i vores projekt/kodning. GUI bliver brugt til at give noget visuelt til vores matador junior spil.

Konklusion

I denne opgave har vi udviklet et Matadorspil, som spilles på computeren. Opgaven blev stillet af firmaet IOOuterActive, som satte en række krav, der skulle opfyldes. Nedenstående tabel viser hvordan disse er opfyldt:

Krav	Status	Beskrivelse af hvordan
Ekstra kast	✓	Koden er bygget op til at identificere om terningernes øjne viser det samme og hvis de gør, modtager samme spiller en ny tur.
Fængsel, hvis en spiller slår samme antal øjne med begge terninger 3 gange i træk.	✓	En if statement der identifierer hvor mange gange en spiller har slået to ens.
Spilleren har 3 muligheder for at komme ud af fængslet.	✓	1 En boolean der identifierer om man har et "kom ud af fængsel"-kort. 2 Prøv at slå to ens. 3 Betal 1000 kr.
Reduceret handel mellem spillere	✓	Spiller kan købe grunde af hinanden for et pengebeløb.
"Prøv Lykken"-felt	✓	Chancekort der bliver udvalgt fra et array af kort, når man lander på "Prøv Lykken"-feltet.
Skibsrederier	✓	Skibsrederier er også et felt der kan ejes, men der er blevet implementeret den del, hvor en funktion identificerer, hvor mange skibsrederier der ejes og sætter lejen efter hvor mange rederier spilleren ejer.
Skøder	✓	Skøder kan købes, ejes fra banken og de kan købes fra andre spillere for et penge beløb
Fallit	✓	Ved hjælp af en switch finder kontrolleren ud af om man kan sælge eller pantsætte sine ejendomme, så man har nok penge til at betale det man skylder.

Programmet er blevet programmeret i intellij og der er blevet lavet fire forskellige slags tests, for at sikre spillets kvalitet. Der er blevet benyttet to brugertests, tre forskellige junit tests, én code coverage test og to enkeltstående test cases. Alle disse viste at spillet fungerer som det skal og dermed at de funktionelle krav er opfyldt. Reliabiliteten er høj, da der er brugt et bredt udvalg af forskellige tests til kvalitetssikringen.

Forklaringsliste

- Forklar hvad arv er med evt. Et eksempel
En klasse (underklassen) kan arve metoder og variabler fra en anden klasse (superklassen) ved hjælp af "extends". Ekstra metoder og variabler kan føjes til underklassen.
Underklassen er derfor generelt større (fylder mere i hukommelsen) end superklassen.
- Forklar hvad abstract betyder.
En abstrakt klasse er en klasse med abstrakte metoder, og en metode erklæret abstract har et metodehoved, men ingen krop. Den kan kun erklæres i en abstrakt klasse. Nedarvede klasser skal definere de abstrakte metoder (eller også selv være abstrakte)
- Fortæl hvad det hedder hvis alle fieldklasserne har en landOnField metode der gør noget forskelligt.
Polymorphy, fordi en klasse nedarver fra en orginal.
- Dokumentation for overholdt GRASP.
I vores kodning har vi overvejede GRASP, og koden er bygget op på den måde, så de har lav afhængighed til hinanden, hvilket er Low Coupling, et eksempel på dette er vores konto der kun er associeret med klassen spiller. Vi har også brugt model view control, hvor klasserne har hver især deres ansvarsområde, dette har til formål at gøre koden mere overskueligt.

Timeregnskab

Tidsforbrug i antal minutter		1						
UGE		4/1/2021	5/1/2021	6/11/2020	7/1/2021	8/1/2021	9/1/2021	10/1/2021
Navn/Dato								
Alexander Solomc		250	240	360	320	380	300	370
Andreas Vilholm V		250	240	360	320	380	300	370
Ahmad Shereef		250	240	360	320	380	300	370
Chenxi Cai		250	240	360	320	380	300	370
Isabel Grimmig Ja		250	240	360	320	380	300	370
Oliver Fielder		250	240	360	320	380	300	370
UGE		2						
Navn/Dato		11/01/2021	12/01/2021	13/01/2021	14/01/2021	15/01/2021	16/01/2021	17/01/2021
Alexander Solomc		420	400	450	430	440	420	520
andreas Vilholm V		420	400	450	430	440	420	520
Ahmad Shereef		420	400	450	430	440	420	520
Chenxi Cai		420	400	450	430	440	420	520
Isabel Grimmig Ja		420	400	450	430	440	420	520
Oliver Fielder		420	400	450	430	400	420	520

Litteraturliste

- CDID del3
- https://www.w3schools.com/java/java_arraylist.asp
- Applying UML and patterns by Graig Larman