



02312 62531 62532

INDLEDENDE PROGRAMMERING, UDVIKLINGSMETODER TIL IT-SYSTEMER OG
VERSSIONSTYRING OG TESTMETODER

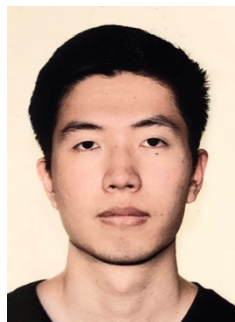
CDIO 3

Gruppe 22

Andreas Vilholm Vilstrup
s205450



Chenxi Cai
s205420



Alexander Solomon
s201172



Oliver Fiedler
s205423



Isabel Grimmig Jacobsen
s205473



Ahmad Shereef
s173750



Afleveret d. 27/11-2020

Resumé

Vi har fået til opgave at lave et Monopoly junior spil, for spilfirmaet IOOuterActive. De har opstillet en række krav og visioner som vi i samarbejde har analyseret specificeret og udviklet. De givne krav er at finde i afsnittet af samme navn. Kravene har vi valgt at dele op i funktionelle- og ikke funktionelle krav, som giver et overblik over hvilke krav, som brugerne skal kunne benytte sig af og hvilke krav der er for at kunne fysisk se hvad der sker i spillet. Derudover har kunden opstillet andre krav, som af forskellige grunde ikke var muligt at implementere i spillet (kig i afsnittet Vurdering af kundens krav/visioner).

Selve spillet er udviklet sådan, at det på samme måder som et fysisk Monopoly junior, og skal spilles af 2-4 spillere. De medvirkende spillere skiftes til at slå med én terning, og flytte hver deres brik rundt på spillepladen, med 24 felter, inklusiv start feltet. Fordelt på spillepladen er der både chancefelter med en positiv eller negativ belønning, derudover er der også felter som repræsenterer ejendomme. Felterne med ejendomme koster penge, men når det er købt og en ny spiller lander på det samme felt bliver der nu opkrævet leje (lejen er den samme som det koster at købe feltet og skal gives til den, spiller der ejer feltet). Felterne er delt op i forskellige farver, som gør at de passer sammen to og to, hvis en spiller har begge felter i samme farve, kan han/hun få dobbelt leje, når en anden spiller lander på en af disse to felter. Spillerne kan også være så uheldige at ryge i fængsel og det skal man betale for at komme ud af. Spillet går i sin enkelthed ud på at købe ejendomme, tjene på dem og være den spiller med flest penge når spillet er slut. Spillet slutter, hvis en spiller ikke kan betale en ejendom eller leje. Vinderen er den med flest penge.

Denne rapport indeholder metoder og diagrammer, der er med til at oplyse hvordan selve spillet er opbygget. Der er bl.a. lavet et use case diagram, som viser hvilke muligheder spillerne har og hvad de skal gøre for at spille spillet. Der er også lavet et Flowchart diagram, som viser og fortæller om programmets processer og hvordan de virker. For at få et godt overblik over hvilke attributter og metoder vores klasser har, og hvilke relationer de har mellem sig, har vi lavet et Design klassediagram. Helheden af diagrammer og modeller er med til at fremme spillet.

Vi har brugt JUnit til at teste den kode vi har udviklet, og dermed sikre at programmet kan køre uden problemer. Vi har bl.a. testet, om der bliver trukket penge fra en spiller der lander på et tomt felt. Vi har også sikret os at der bliver fundet en vinder, hvis en spiller ikke kan betale. Da det er et spil til børn, har vi lavet en brugertest, for at få en viden om det er et let håndgribeligt og brugervenligt spil.

Ud fra diagrammer, metoder og test af programmeringen/koden, kan vi konkludere at spillet er let at spille og forstå. Derudover kan vi også konkludere at der godt kunne komme forbedringer, så det bliver mere tydeligt for børn, hvordan de kommer i gang med spillet. I vores tilfælde, har tiden været en begrænsning, men de mangler der er i forhold til karv, vil kunne blive realiseret med mere tid.

Indholdsfortegnelse

Resumé	2
Indledning	5
Krav	5
<i>Funktionelle krav:</i>	<i>5</i>
<i>Ikke -funktionelle krav:</i>	<i>6</i>
<i>Vurdering af kundens krav/visioner.....</i>	<i>6</i>
Analyse	7
<i>Use Cases.....</i>	<i>7</i>
<i>Flowchart.....</i>	<i>10</i>
<i>Domæne model</i>	<i>12</i>
Design	13
<i>Design klassesdiagram.....</i>	<i>13</i>
<i>Sekvensdiagram</i>	<i>14</i>
<i>System sekvens diagram</i>	<i>15</i>
Implementering	16
<i>Beskrivelse af klasser.....</i>	<i>16</i>
Dokumentation – mangler.....	19
<i>Forklar hvad arv er med evt. Et eksempel</i>	<i>19</i>
<i>Forklar hvad abstract betyder.</i>	<i>19</i>
<i>Fortæl hvad det hedder hvis alle fieldklasserne har en landOnField metode der gør noget forskelligt.</i>	<i>19</i>
<i>Dokumentation for overholdt GRASP.</i>	<i>19</i>
Test.....	20
<i>Testcase 1</i>	<i>20</i>
<i>Testcase 2</i>	<i>23</i>
<i>Testcase 3</i>	<i>27</i>
<i>Code coverage af vores klasser</i>	<i>29</i>
<i>JUnit.....</i>	<i>29</i>
<i>Brugertest.....</i>	<i>30</i>
Konfigurationsstyring	30
<i>En guide til når man modtager filen og vil køre den:</i>	<i>30</i>
Projektplanlægning	31
<i>Intellij:.....</i>	<i>31</i>

<i>Maven:</i>	31
<i>Github:</i>	31
<i>Use case:</i>	31
<i>Domain model:</i>	31
<i>Flowchart:</i>	31
<i>System sekvensdiagram:</i>	31
<i>Design klassediagram:</i>	31
<i>Sekvensdiagram:</i>	32
<i>JUnit:</i>	32
<i>Gui:</i>	32
Konklusion	32
Timeregnskab.....	33
Litteraturliste	33

Indledning

I denne rapport dokumenterer og udvikler vi en opgave, som er blevet stillet af spilfirmaet IOOuterActive. Opgaven indeholder dele fra de foregående CDIO delopgave, og ender ud med at et Monopoly junior spil. Spillet er et klassisk Monopoly junior, der foregår mellem 2-4 spillere, som skiftes til at kaste en terning og rykke rundt på spillepladen.

Kunden har opstillet nogle krav (se afsnittet om krav), som vi har prøvet at implementere og opfylde. Blandt andet vil kunde gerne have udviklet et spillebræt, som har forskellige butikker/aktivitetssteder og pris for både at købe dem og lande på dem efter de er købt. Hver deltagende spiller får hver en brik men en farve, som skal rykke rundt på spillepladen, antallet af felter en spiller skal flytte, bliver bestemt ud fra hvilket antal øjne terningen viser. Spillerne kan enten lande på felter, som de kan købe eller skal betale leje. De kan også lade på felter som giver et chancekort eller sender dig i fængsel. Spillet vindes ved at være den der har flest penge tilbage, når en anden spiller ikke kan betale leje eller købe en ejendom.

Derudover indeholder rapporten også modeller og diagrammer, som er med til at give et overblik og en forståelse for hvordan spillet virker og hvad det indeholder.

Krav

Funktionelle krav:

Lander eller passerer START, modtager man M2! - I det én spiller passere stat, vil spillet automatisk tildele personen M2.

Man skal altid rykke frem og ikke tilbage - Det vil være selve spillet, som rykker brikkerne ud fra hvad terningen slår. Derfor er det ikke nødvendig at specificere at man kun må flytte brikken frem. Der er dog en undtagelse i spillet, hvis en spiller bliver bedt om at rykke i fængsel.

Ryk med uret - Det er intet problem da man bare laver sine ryk med positive værdier så de kun rykker samme vej med uret.

Lander man på et ledigt felt SKAL det købes - Prisen på feltet/ ejendommen står nederst på feltet.

Hvis en anden spiller lander på et felt der er ejet, skal man betale husleje - Prisen for at leje, er den samme som det koster at købe feltet/ejendommen. Beløbet skal betales til den spiller som har købt feltet/ejendommen, dette vil computeren automatisk gøre for spillerne.

Hvis man lander på et felt som er en del af en serie, som én person ejer, betale man dobbelt husleje - I tilfælde af at én spiller ejer to felter i samme farve, og en anden spiller lander på en af disse to felter, skal spilleren betale dobbelt leje.

Hvis man lander på et chancefelt, får man et kort med en opgave/en handling - På spillepladen er der 4 chance felter, som alle udløser et lykkekort/chancekort. Chance kortene giver forskellige muligheder i enten en positiv eller negativ retning.

Gå i fængsel felt, føre dig til fængslet uden at passere start, og på din næste tur skal man enten bruge en m1 eller et løsladelseskort og så kan du rykke videre som normalt - Dette felt er det eneste felt der kan få brikken, på den spiller der lander på feltet, til at flytte forbi start uden at få penge og lande på fængselsfeltet.

Lander man på feltet "På besøg" er du på besøg i fængslet og intet sker - Dette felt koster ikke spilleren, som lander på feltet, noget i bøde. Det er bare en lille pause, til det igen bliver spillerens tur.

Lander du på "Gratis parkering" kan du slappe af, intet sker - Der vil ikke ske spilleren noget, som lander på dette felt. Det er en lille pause til den spiller der lander på feltet.

Vind spillet: er der én der ikke kan betale slutter spillet og den med flest penge vinder - Hvis en spiller ikke kan betale et felt/ en ejendom, er spilleren nødt til at sælge andre ejendomme, ellers er spillet tabt. Den spiller som har flest penge (i det én spiller løber tør for penge) har vundet, hvis det bliver uafgjort mellem to andre spillere, deler de sejren.

Ikke -funktionelle krav:

1 spillebræt - Spillet indeholder en standard spilleplade, hvor alle funktionerne er indbygget og som er med til at kontrollere spillets fremgang.

2-4 spillere - Det er valgfrit om man vil være 2 eller 3 eller 4 spillere.

20 chancekort - Vi har valgt kun at have 16 chancekort med i spillet grundet, tidsmangel.

48 solgt skilt - Vi har i samarbejde med kunden vurderet at, det ikke er nødvendigt at skulle have 48 solgt skilte, med ubegrænset antal. Da det er et computerspil, kan computerne bare lave de solgt skilte der er brug for.

90 m1-sedler - Da det ikke er en fysisk brætspil, er der ingen grund til at have et begrænset antal penge i spillet. Vi har derfor i samarbejde med kunden valgt at spillet indeholder alle de penge man måtte få brug for.

1 terning - spillet indeholder kun en terning, som alle spillerne skal skiftes til at bruge.

Vurdering af kundens krav/visioner

En spiller skal være bankør - Vi har vurderet i samarbejde med kunden, at det ikke er nødvendigt at vælge en spiller til at være bankør. Dette skyldes at computeren kan udgøre det for bankør, og dermed han holde styr på pengene og hvilke spiller der ejer hvilke felter.

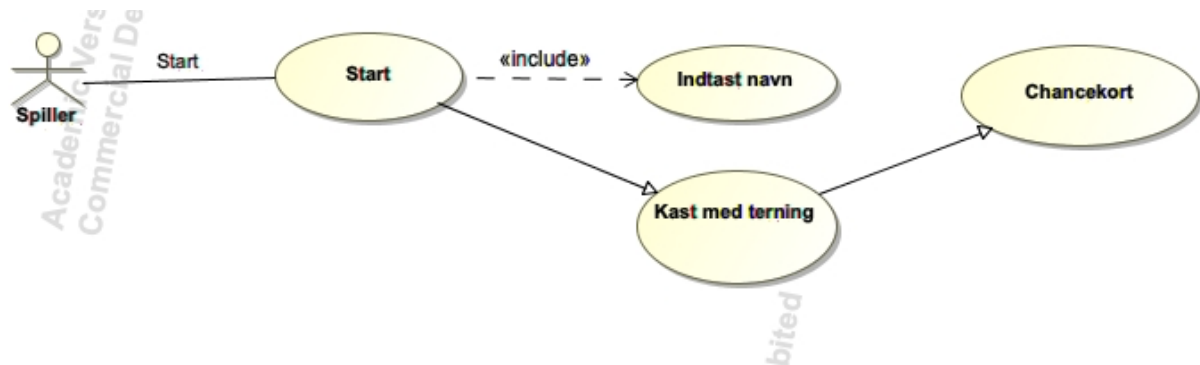
4 figurer kort - Når spillerne indtaster deres navn, vil de automatisk også få tildelt en figur og en farve. Computeren vil holde styr på hvem der er hvem og hvad de har købt og hvor mange penge de har. Vi er derfor blevet enige om, at det ikke er nødvendigt at skulle uddele kort der passer til de individuelle spillebrikker. Computeren vil fortælle navnet på den spiller, hvis tur det er og computeren vil og holde styr på hvilken brik den spiller har. Derfor er det ikke nødvendigt at have 4 spillekort, til at fortælle dette.

Den yngste spiller starter - I samarbejde med kunden, har vi vurderet at der ikke var tid nok til at implementere en metode der skal finde den yngste spiller. Det betyder at det er op til spillerne selv at finde ud af hvilken spiller, der er yngst og dermed skal starte.

Analyse

Use Cases

Use case diagram -mangler en beskrivelse.



På diagrammet ovenfor, ses hvad de enkelte spillere kan/skal gøre, og hvad deres handlinger medføre. De stiplede linjer i diagrammet indikerer hvad handlingen medfører, fx. Når spillerne trykker start, vil det medføre at man skal indtaste antal af spillere og deres navne.

Use case Brief beskrivelse:

Use case	Beskrivelse
Start spillet	Tryk enter for at starte spillet og blive ført til siden.
Indtast navn og fødselsdato	Spillet beder dig om at indtaste dit navn, for at identificere de individuelle spillere.
Kast terningen	Tryk for at kaste med tegningen.
Chancekort	Hvis man har modtaget et chancekort der kan aktiveres, har spilleren muligheden for at benytte sig af kortet efter terningerne er kastet.

Use case Fully Dressed beskrivelse:

Use case sektion	Beskrivelse	Use case sektion	Beskrivelse
Use case name	Start spil	Use case name	Indtast navn
Scope	Tryk på start knappen for at komme i gang med spillet, og derefter vil	Scope	Spillet ville spørge efter alle deltagernes navn. Der kan være op til 4 spillere, som alle vil blive spurgt om at indtaste personoplysninger (afslut med enter/ok knap). Når alle

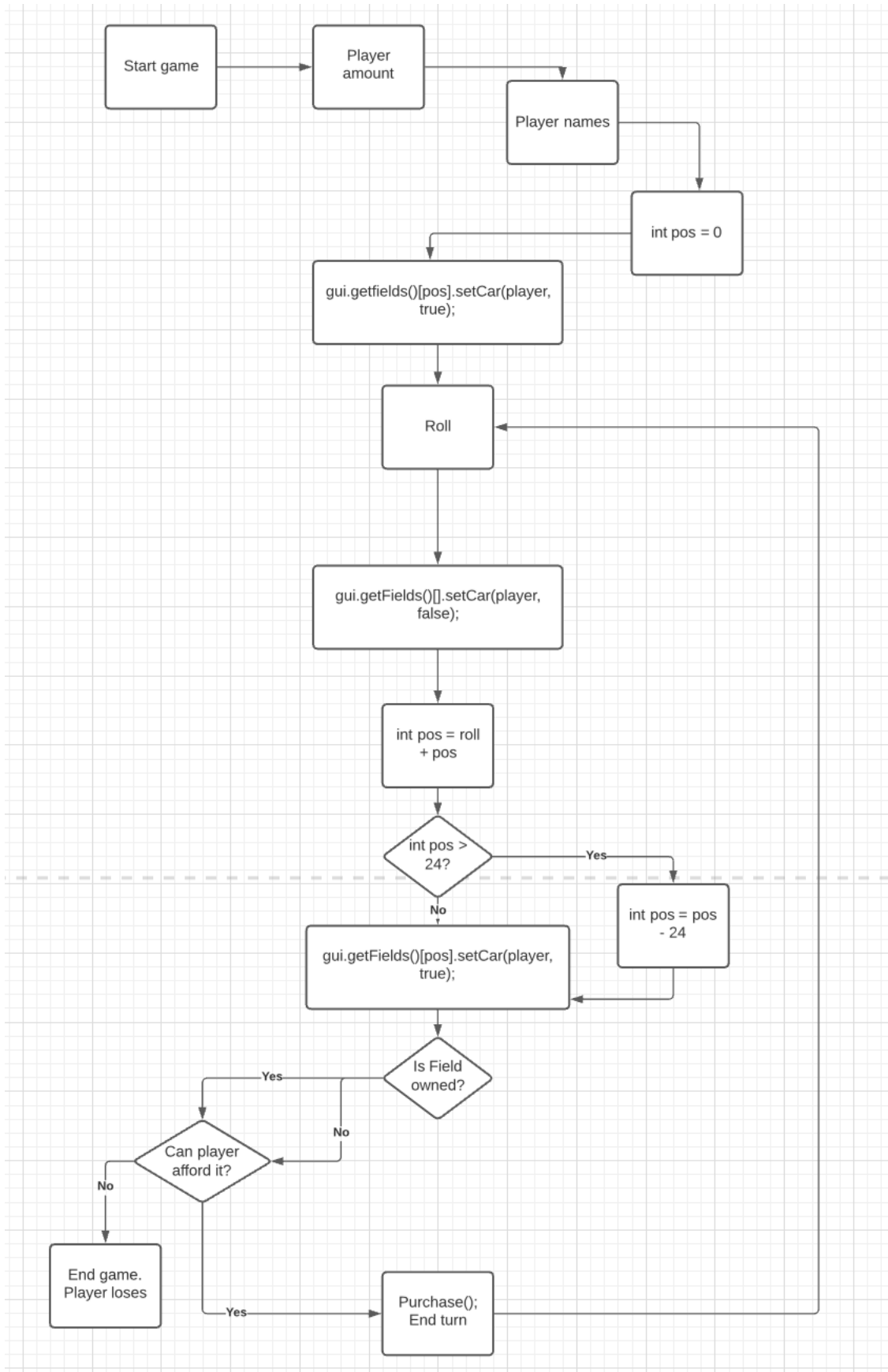
	man blive sendt videre til en ny side.		de deltagende spillere har indtastet oplysninger, vil den første spiller kunne kaste med terningen.
Level	User goal	Level	User goal
Primary actor	2-4 spillere	Primary actor	2-4 spillere
Success guarantee	Når spilleren har trykket på start knappen vil det medfører et succes scenario, da programmet vil kunne fortsætte til næste del af spillet.	Success guarantee	Når alle de deltagende spiller har indtastet deres navn.
Main success scenario	At trykke på start knappen og komme videre i spillet	Main success scenario	Alle deltagere har indtaste deres data og der bliver fundet en spiller som skal starte.
Extensions	Ved tekniske problemer skal programme genstarte.	Extensions	Hvis der er problemer med at indtaste data, luk spillet ned og start forfra.
Special requirements	Tryk enter eller knappen der fører spilleren videre til næste fase i spillet.	Special requirements	Der skal minimum indtaste data på 2 personer og maks. 4 personer.
Technology and data variations list	-	Technology and data variations list	-
Frequency of Occurrence	Test af programmet kan foretages på en ugentlig basis.	Frequency of Occurrence	Test af programmet kan foretages på en ugentlig basis.
Miscellaneous	Hvad vil programmet gøre ved brug af andre taster, mens programmet venter på man klikker start knappen.	Miscellaneous	Hvad sker der hvis går lang tid inden der bliver tastet data ind?

Use case sektion	Beskrivelse	Use case sektion	Beskrivelse
Use case name	Kast terning	Use case name	Chance Kort
Scope	Spillet vil nu bede en spiller om at trykke på kast terningknappen, for at kaste terningerne og derefter vil antallet af øjne vise hvor mange felter du skal flytte frem. Når turen er slut, bliver Spiller 2 bedt om det samme og på den måde fortsætter spillet.	Scope	Undervejs i spillet vil man komme ud for forskellige chancekort, hvis en spiller trækker kom ud af fængsel kort vil man kunne benytte sig af det kort, hvis man bliver smidt i fængsel.
Level	User goal	Level	User goal
Primary actor	2-4 spillere	Primary actor	2-4 spillere
Success guarantee	Terningen vil vise et bestemt antal øjne, og derefter rykker spilleren det antal felter frem, som terningen viser.	Success guarantee	Bruger chance kortet efter spilleren er røget i fængsel.
Main success scenario	Terningen viser et tilfældig antal øjne hver gang der bliver kastet.	Main success scenario	Er en spiller røget i fængsel og har modtaget et chancekort der hedder, kom ud af fængsel. Kan spilleren der er røget i fængsel benytte sig af det chancekort.
Extensions	Hvis roll knappen ikke fungere, kan det forsøges med enter knappen på tastaturet.	Extensions	Chancekort bliver tildelt til en spiller der lander på chance feltet, hvis de ikke modtager et kort, fortsætter spillet.
Special requirements	Der er to funktioner der kan kaste terning, enter og roll knappen.	Special requirements	En spiller skal lande på chance feltet for at modtage et chancekort.
Technology and data variations list		Technology and data variations list	

Frequency of Occurrence	Test af programmet kan fortages på en ugentlig basis	Frequency of Occurrence	Test af programmet kan fortages på en ugentlig basis
Miscellaneous	Vil programmet kunne vise det forkerte antal øjne, eller rykke et forkert antal felter baseret på terningen?	Miscellaneous	Man behøver ikke at bruge fængselskortet igennem hele spillet og vil derfor kunne ende med et chancekort som spiller, når spillet slutter.

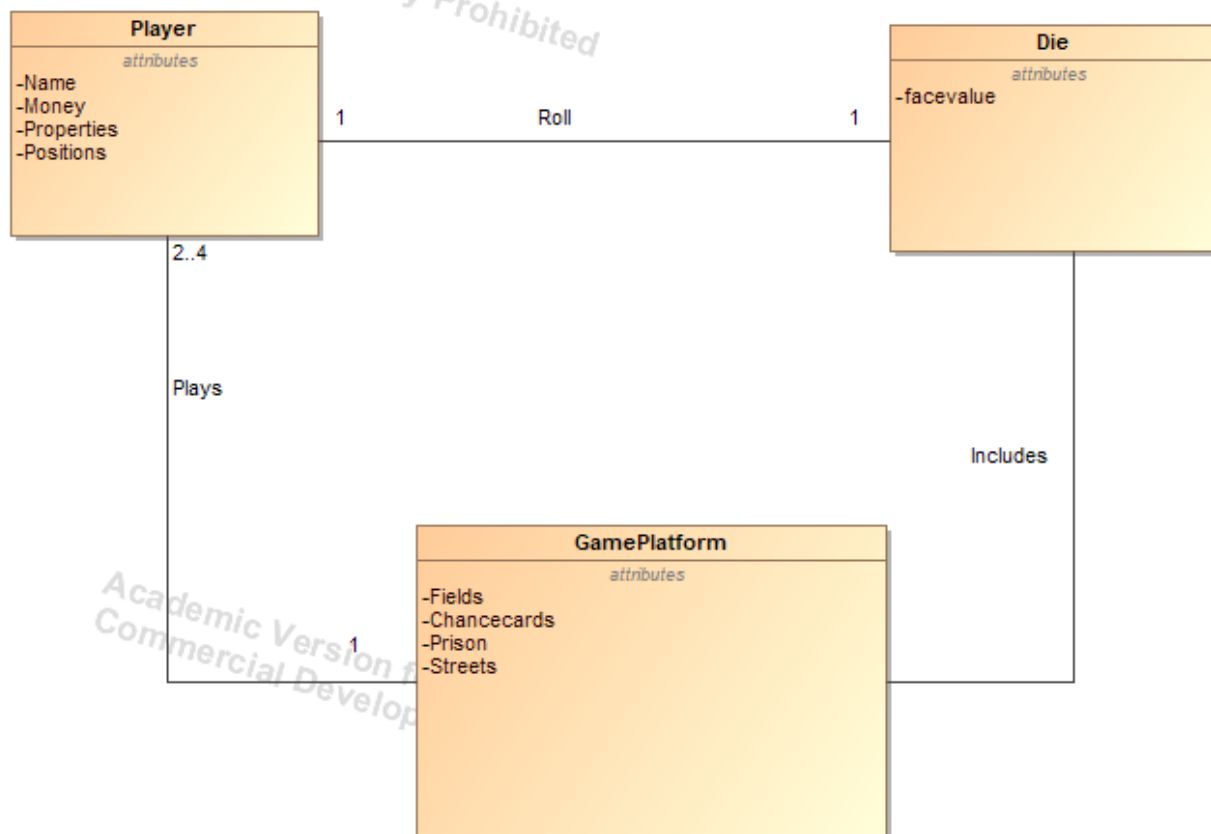
Flowchart

Her ses et flowchart, som giver et overordnet blik over hvordan vores system kommer til at virke. Det kan ses, at spillet starter med at spørge om antal af spiller, dermed spillernes navne. Derefter slå man terningen, og brikker rykkes på felter. Hvis spillers position er over 24, hvilke betyder de har passeret startfeltet, så starter man forfra. Hvis spiller lander på et felt, som kan købes, skal man enten købe feltet eller betale leje til ejeren, hvis den er ejet, hvis spilleren ikke kan betale/købe så slutter spillet.



Domæne model

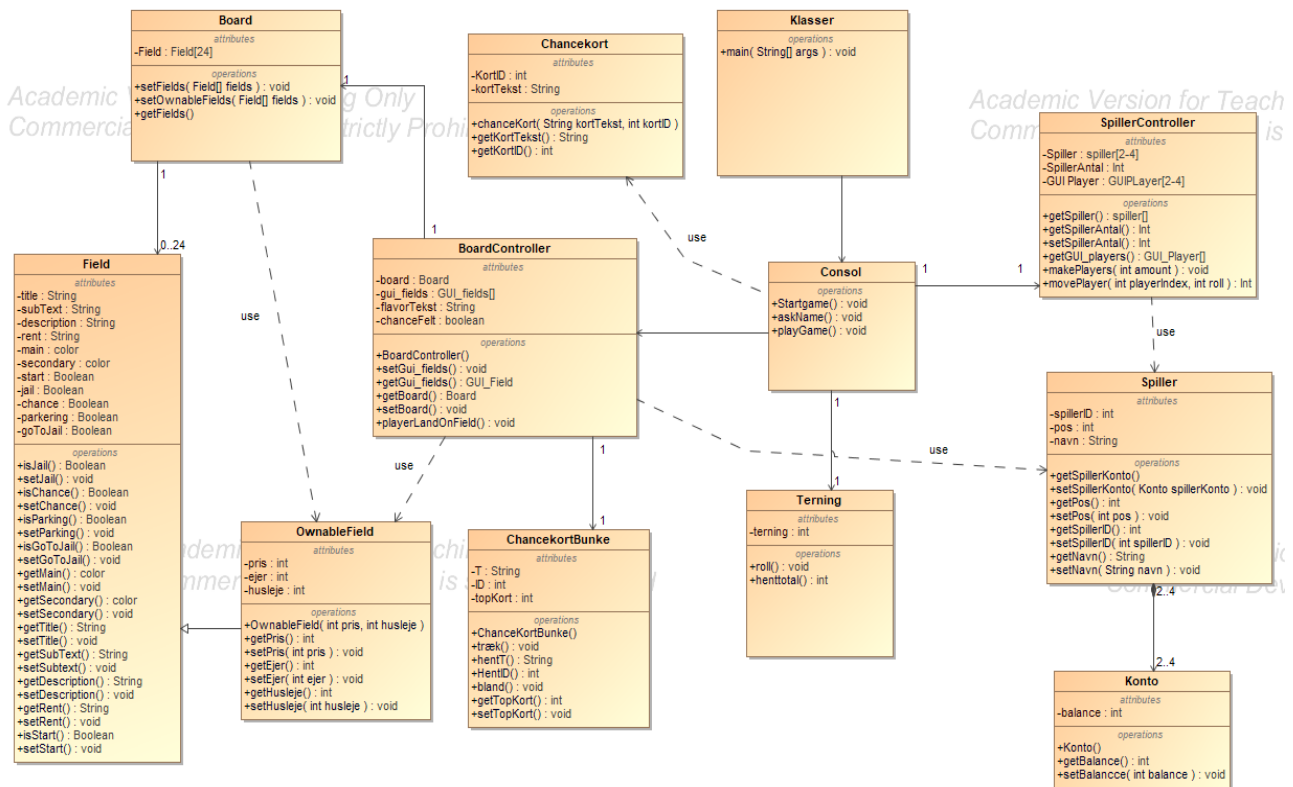
Nedenfor ses der en domain model, som viser de klasser vi har, hvor deres attributter også vises. Vi har 3 klasser, "Player", "die", "GamePlatform", hvori modellen viser deres relationer til hinanden. Der ses multiplicitet indikator mellem klasserne, for eksempel har GamePlatform to til fire Player, og en Player har 1 Die.



Design

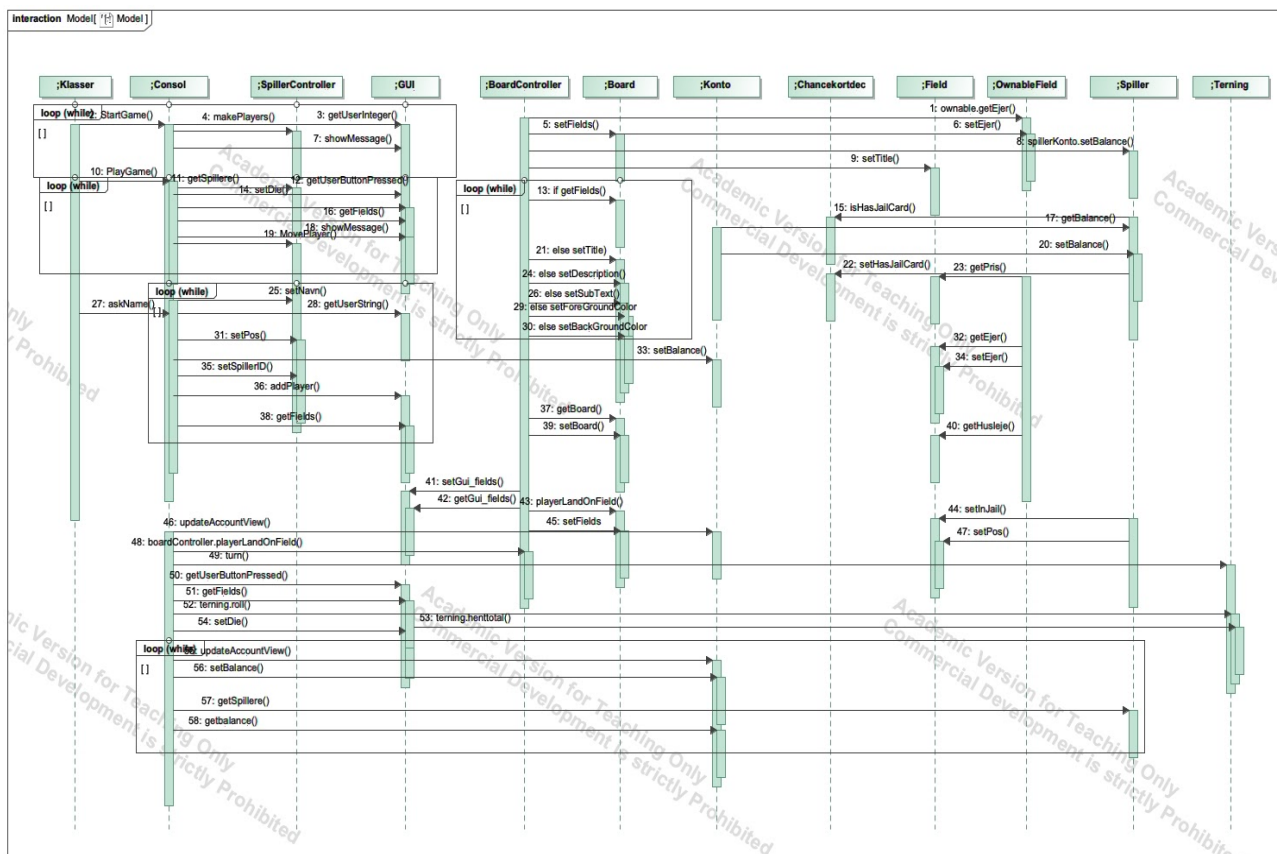
Design klassediagram

Design klassediagrammet nedenfor viser hvilket attributter og metoder vores klasser har, og deres relationer til hinanden. Dette giver et overblik over vores kode, da man kan se deres relationer, så ved man hvordan klasserne er afhængige af hinanden, og hvilke klasser bruger hinanden, det kan fx ses ud fra pilen.



Sekvensdiagram

Nedenfor ses samspillet mellem objekterne, klassen "Klasser" kommunikerer med klassen "Console" igennem metoderne fra koden, hvor igennem kommunikationen også dannes løkker som ses foruden.



Implementering

Beskrivelse af klasser

Board:

I board klassen, har vi lavet vores egne felter ved brug af et array, hvert felt der er lavet, er blevet tildelt en titel og en beskrivelse af feltet. OwnableFields er også i klassen i et andet array og er blevet tildelt en pris og en husleje. Vi har også en getFields metode så vi kan hente felterne hvor end vi nu har brug for dem.

```
public void setOwnableFields(Field[] fields) {
    fields[1] = new OwnableField( pris: 1, husleje: 1);
    fields[2] = new OwnableField( pris: 1, husleje: 1);
    fields[4] = new OwnableField( pris: 1, husleje: 1);
    fields[5] = new OwnableField( pris: 1, husleje: 1);
    fields[7] = new OwnableField( pris: 2, husleje: 2);
    fields[8] = new OwnableField( pris: 2, husleje: 2);
    fields[10] = new OwnableField( pris: 2, husleje: 2);
    fields[11] = new OwnableField( pris: 2, husleje: 2);
    fields[13] = new OwnableField( pris: 3, husleje: 3);
    fields[14] = new OwnableField( pris: 3, husleje: 3);
    fields[16] = new OwnableField( pris: 3, husleje: 3);
    fields[17] = new OwnableField( pris: 3, husleje: 3);
    fields[19] = new OwnableField( pris: 4, husleje: 4);
    fields[20] = new OwnableField( pris: 4, husleje: 4);
    fields[22] = new OwnableField( pris: 5, husleje: 5);
    fields[23] = new OwnableField( pris: 5, husleje: 5);
}
```

BoardController

I BoardControlleren tjekker vi hvilket type felt det er ved hjælp af booleans og if-sætninger. Herinde har vi også noget af teksten der bliver vist i spillet ved alle felter undtagen ownableFields, men det er så der hvor vi kalder på de beskrivelser vi har lavet af ownableFields i board klassen. Det er også i denne klasse vi har lavet funktionaliteterne til vores board ved hjælp af playerLandOnField metode. Den tager argumenterne for hvilken spiller og hvor han står, og så tjekker den om hvilken type felt det er. Hvis det er et OwnableField har vi nogle if-sætninger til at finde ud af om det er et allerede ejt felt, om du eller andre ejer det osv. Og hvad der sker ved de forskellige scenarier.

Der er også `playerLandOnFieldFree` metoden som er den metode vi henter hvis man får et chancekort der kan give et gratis felt.

BoardControllerTest

Det er en Junit test som tester om man kan købe og eje et felt. Vi henter et board, opretter en spiller og rykker ham til et `ownableField` og tjekker om ejer ID er den samme som hans spiller ID efter han har købt det.

```
@Test
void playerLandOnField() {
    BoardController boardController = new BoardController();
    SpillerController spillerController = new SpillerController();
    spillerController.makePlayers( amount: 1);
    SpillerController.spillere[0] = new Spiller();
    SpillerController.spillere[0].setNavn("testNavn");
    SpillerController.spillere[0].setPos(0);
    SpillerController.spillere[0].setSpillerID(0);
    SpillerController.spillere[0].spillerKonto.setBalance(20);
    OwnableField ownable = (OwnableField) boardController.board.getFields()[1];
    boardController.playerLandOnField(spillerController.getSpillere()[0], fieldID: 1);
    assertEquals(spillerController.getSpillere()[0].getSpillerID(), ownable.getEjer());
}
```

Chancekort

Her laver vi nogle attributer til chancekortne og laver en konstruktør vi kalder `ChanceKort` med argumenterne korttekst som er string og `KortID` som er int.

ChanceKortBunke

Her har vi en konstruktør med et array af strings som fortæller flavorteksten til hvert chancekort vi har med. Og en træk metode til at trække et af kortene. Den træk metode bliver så beskrevet længere nede.

Så har vi en bland metode som bliver lavet med et while loop som køre 1000 gange med `math.random`. Så har vi metoden `getTopKort` for at kortene bliver trukket i samme rækkefølge.

Consol

Her har vi al funktionaliteten for at køre spillet. Vi har metoden `startGame` som spørger hvor mange spillere der er med og giver fejl hvis man giver et inkorrekt antal.

Vi har `askName` metoden som spørger om navnet på spillerne, opretter deres spiller på brættet, og sætter spillernes konto til en startværdi.

Vi har `playGame` metoden bestående af en if-sætning som sørger for at antallet af spillere ikke går 'out of bounds', en if-sætning som tjekker om spilleren er i fængsel og om de har et 'kom ud af fængsel's kort. Alt det ligger i et while-loop som kører tur metoden som kun breaker hvis en spillers konto kommer under 0.

Tur metoden bliver så beskrevet nedenunder hvor man ruller med terningen, rykker spillerne, updaterer spillet og viser tilhørende flavortekst. Den sørger også for at hvis man rammer et chancefelt så referer den til nedenstående chancekortsswitch.

Vi har `updateView` metoden som sørger for at holde alles kontoer up to date.

Vi har `endGame` metoden som sorterer spillerne og ser hvem der har flest penge og på baggrund af det kårer en vinder og en taber.

Så har vi til sidst vores `boardControllerSwitch` som arbejder sammen med chancekortbunke om at vise alt om chancekortne. I denne switch har vi alle funktionaliteterne som er krævet ved hvert chancekort og sørger for at de hører sammen med de tilførende strings fra chancekortbunke.

Field

Field klassen indeholder 4 variabler af typen `String` og 2 af typen `Color`. De 6 variabler er der blevet lavet en `set` og `get` funktion til. Denne funktion bliver brugt til at definere en masse attributter i form af strings og booleans. Her opretter vi også nogle getter og settere som vi kan bruge i andre klasser, samt boolean getter og settere.

```
String title;
String subText;
String description;
Color main;
Color secondary;

boolean start;
boolean jail;
boolean chance;
boolean parkering;
boolean goToJail;
boolean ownable;
```

Klasser

Denne klasse bruges til at starte og køre programmet. Det er denne klasse som henter alle metoderne fra console til at køre det fulde spil. Det er vigtigt for os at denne klasse er kort og overskuelig.

Konto

Inde i klassen `konto` bliver kontoen oprettet. Her bliver der lavet en værdi af typen `int` der er defineret ved `balance`. `Balance` funktionen bliver der lavet en `get` og `set` funktion. Balancen kan bruges til at blive kaldt når penge indholdet ændres eller sættes til en værdi.

KortTest

`KortTest` er ikke en del af spillet, men bruger den for at tjekke om vores blandingsfunktion og om hvorvidt den trækker kortene i samme rækkefølge virker for Chancekortbunken ved at trække 100 kort efter den er blevet blandet.

OwnableField

I ownableField indeholder klassen attributterne og getter/setter-metoderne som er nødvendige for når vi bruger ownableField i andre klasser. I denne klasse har vi også extended til Field.

Spiller

I spiller klassen er der nogle datatyper der bliver defineret. Pos og spillerID bliver defineret til at være af typen int. inJail og hasJailCard er defineret som booleans.

Der er blevet lavet en get og set funktion ved værdierne: om de har et jailcard, om de er i fængsel, hvilken position de har, hvilken spiller vi snakker om, og om hvad de forskellige spillere hedder.

Set pos metoden indeholder også den funktion, giver spillerne 2 moneter på deres konto når de kommer forbi felt 23.

SpillerController

Her opretter vi spillerne og gui spillerne, laver en metoden til at rykke spillerne og metoden til at sortere spillerne efter konto for at finde ud af hvem der har flest penge i deres konto til at kåre en vinder.

Terning

Vi har en terning der bliver defineret som en talværdi int. Terningen har en roll metode der kan vise et tilfældigt tal mellem 1 og 6.

Dokumentation – mangler

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 original.

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.

Test

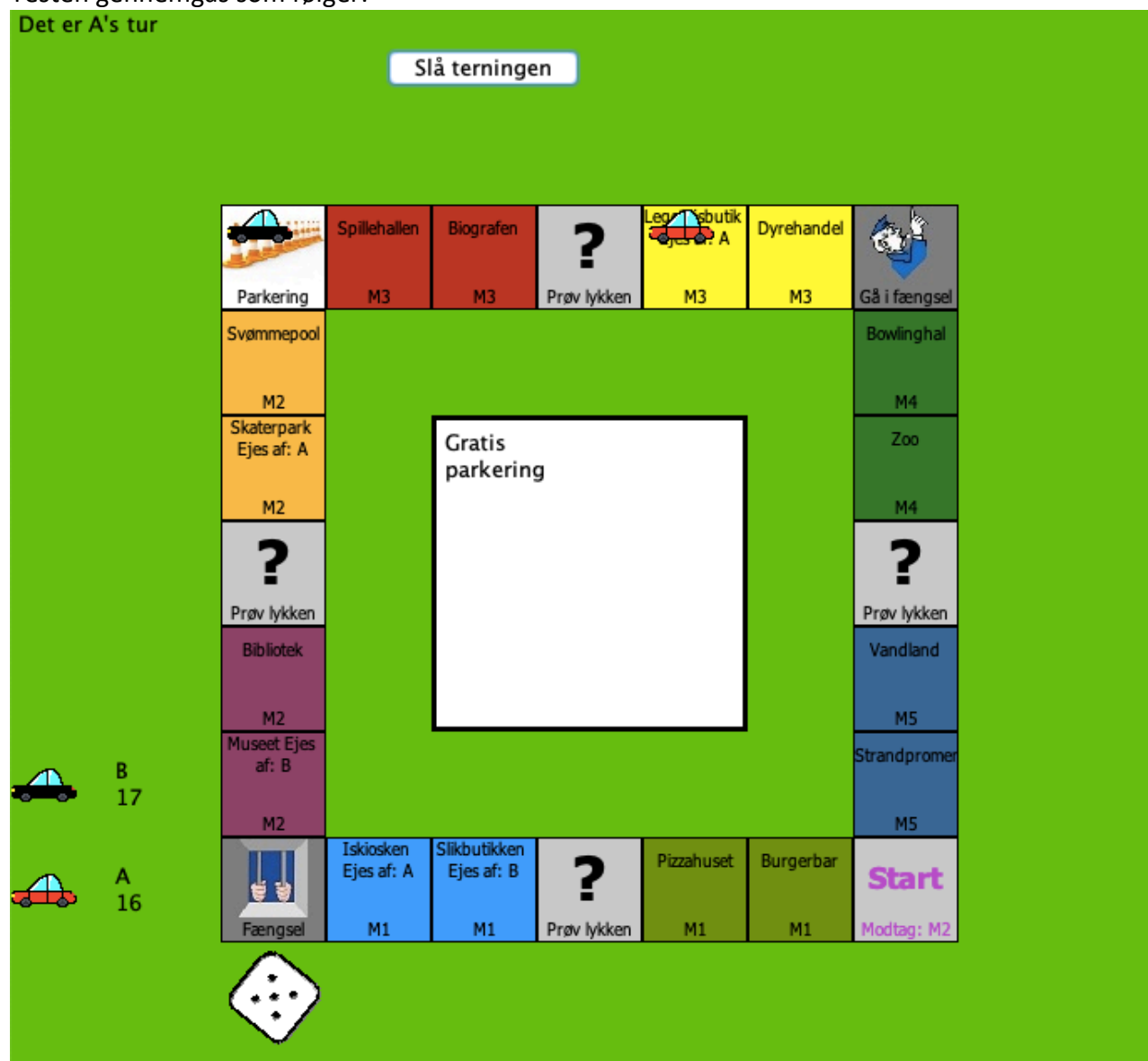
Testcase 1

I det følgende vil der blive gennemgået en test af scenariet: En spiller lander på et felt som ikke er ejet. Dette bliver købt. Der vil blive taget udgangspunkt i at spillet er gået i gang og der er blevet købt visse ejendomme før og spiller er egentligt i gang. Derfor vil der blive kørt minimum tre runder før testen startes.

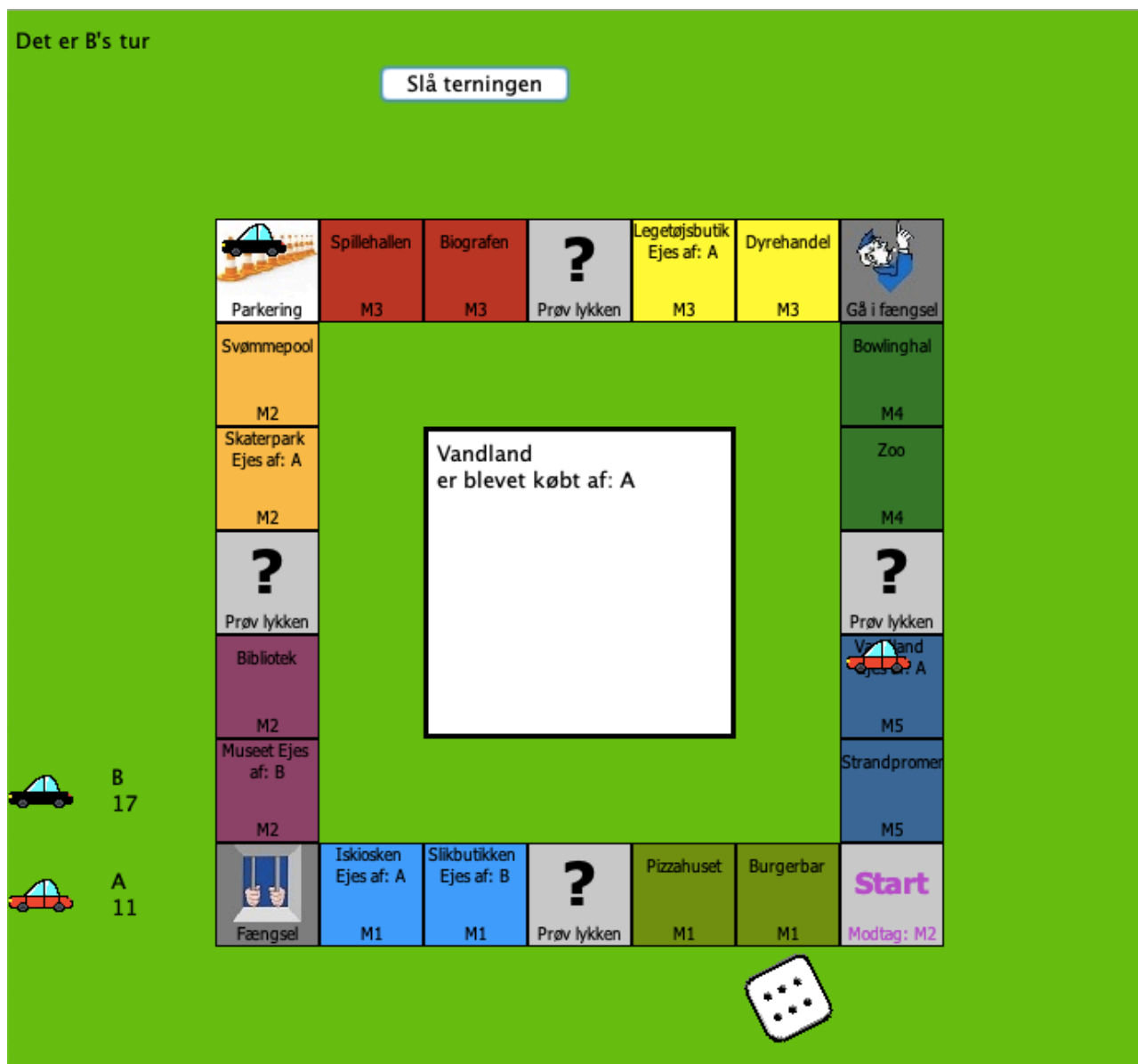
Dernæst skal det næste felt man lander på der kan købes.

Det forventes at ejeren skiftes og pengene trækkes.

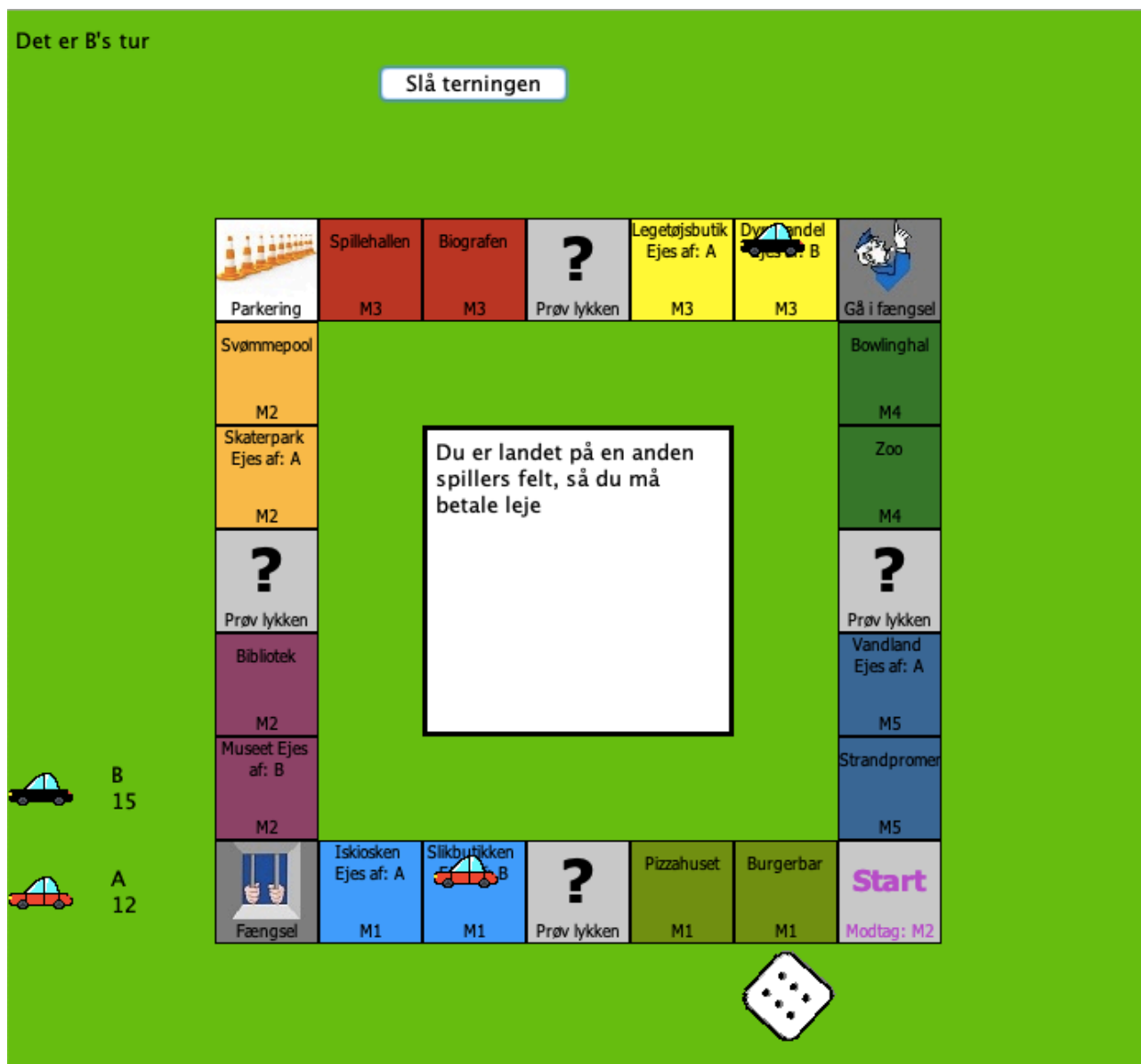
Testen gennemgås som følger:



Tre fulde runder er blevet gennemgået og dernæst skal det næste kort købes hvis muligt:



Det ses at A rykker fra legetøjsbutikken til vandland. Ydermere kan det også aflæses at A bliver trukket 5 moneter.



Det kan derudover aflæses at A nu er ejer efterfølgende.
Testen konkluderes altså som succesfuld.

Testcase 2


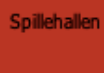







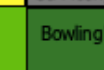


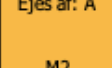



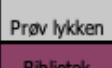








Dernæst testes træk af chancekort, samt konsekvensen heraf.

Nedenfor ses et igangværende spil, hvor det nu er B's tur.

På det følgende vil der blive slået med terningen.


Det er B's tur

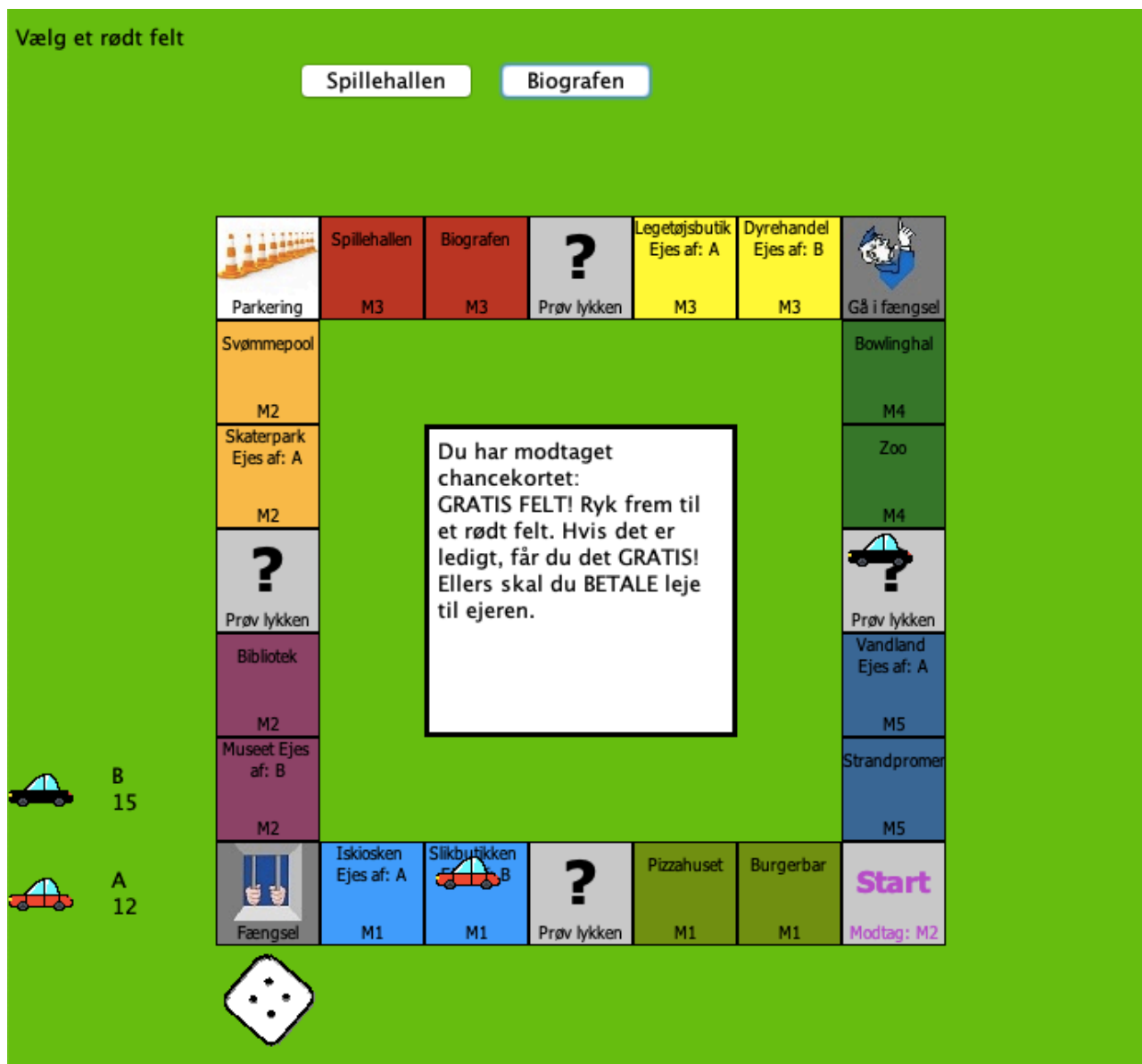
Slå terningen

 Parkering M3	 Spillehallen M3	 Biografen M3	 Prøv lykken	 Legetøjsbutik Ejes af: A M3	 Dyrerandel Ejes af: B M3	 Gå i fængsel
 Svømmepool M2					 Bowlinghal M4	
 Skaterpark Ejes af: A M2					 Zoo M4	
 Prøv lykken					 Prøv lykken	
 Bibliotek M2					 Vandland Ejes af: A M5	
 Museet Ejes af: B M2					 Strandpromenade M5	
 Fængsel	 Iskiosken Ejes af: A M1	 Slikbutikken Ejes af: B M1	 Prøv lykken	 Pizzahuset M1	 Burgerbar M1	 Start Modtag: M2

B 15

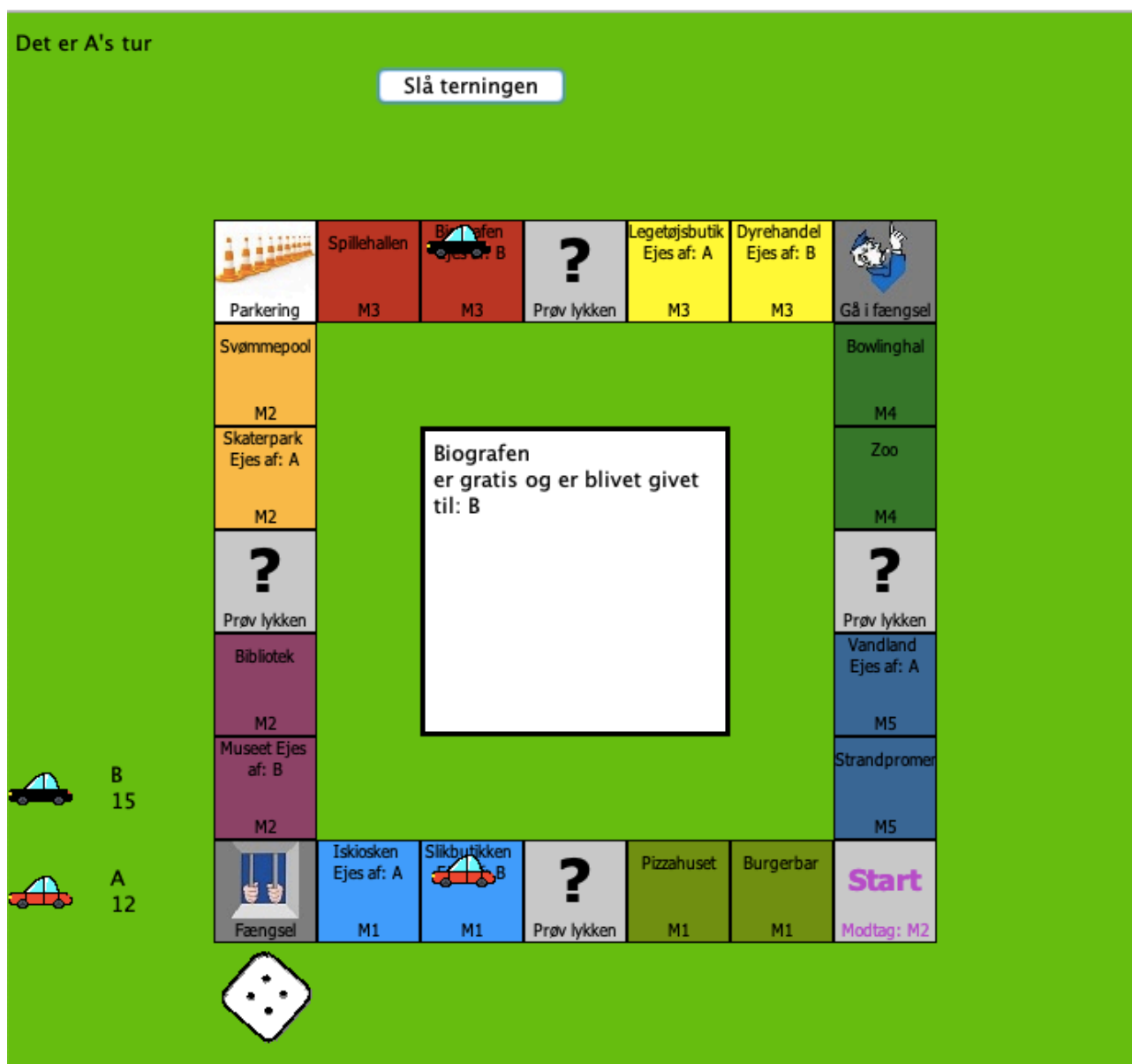
A 12





Det ses ovenfor at spilleren lander på et chancefelt og trækker kortet: "GRATIS FELT" Ryk frem til et rødt felt. Hvis det er ledigt, får du det GRATIS! Ellers skal du BETALE leje til ejeren."



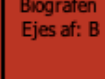

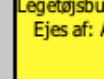


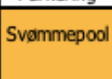
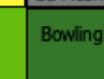
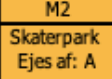

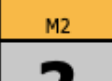

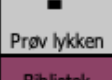

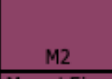

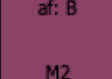






I dette tilfælde vil "Biografen"-feltet blive valgt:



Det ses at B rykkes frem til "Biografen" og det kan ydermere aflæses at B nu ejer feltet og at der ikke er blevet trukket nogen penge.

Det er A's tur

Slå terningen

 Parkering M3	 Spillehallen M3	 Biografen Ejes af: B M3	 Prøv lykken M3	 Legetøjsbutik Ejes af: A M3	 Dykvand Ejes af: B M3	 Gå i fængsel M3
 Svømmepool M2	Du ejer allerede feltet					 Bowlinghal M4
 Skaterpark Ejes af: A M2						 Zoo M4
 Prøv lykken M2						 Prøv lykken M5
 Bibliotek M2						 Vandland Ejes af: A M5
 Museet Ejes af: B M2						 Strandpromenaden M5
 Fængsel M1	 Isbutik Ejes af: A M1	 Snickers Ejes af: B M1	 Prøv lykken M1	 Pizzahuset M1	 Burgerbar M1	 Start Modtag: M2

B 15

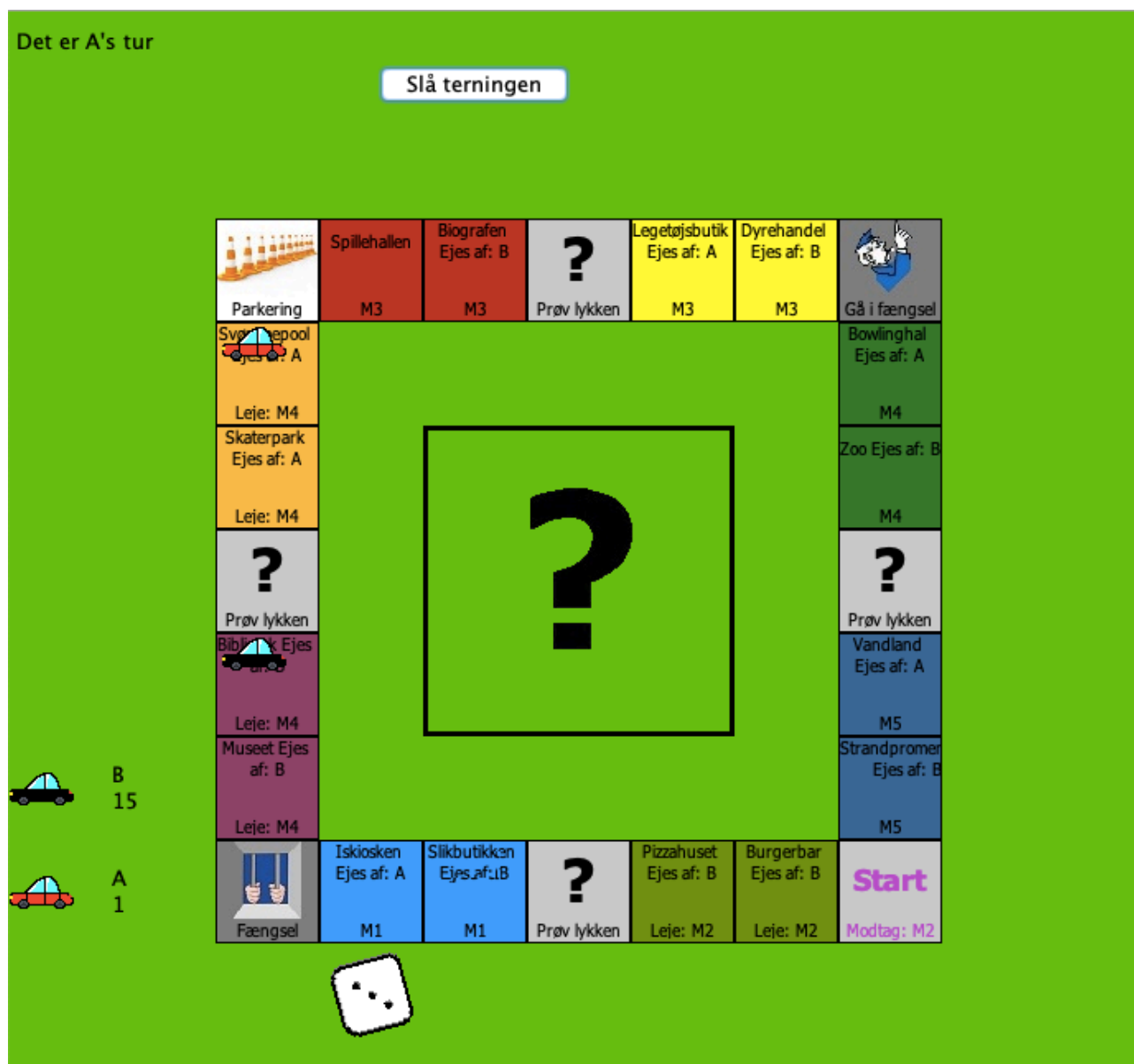
A 12

3

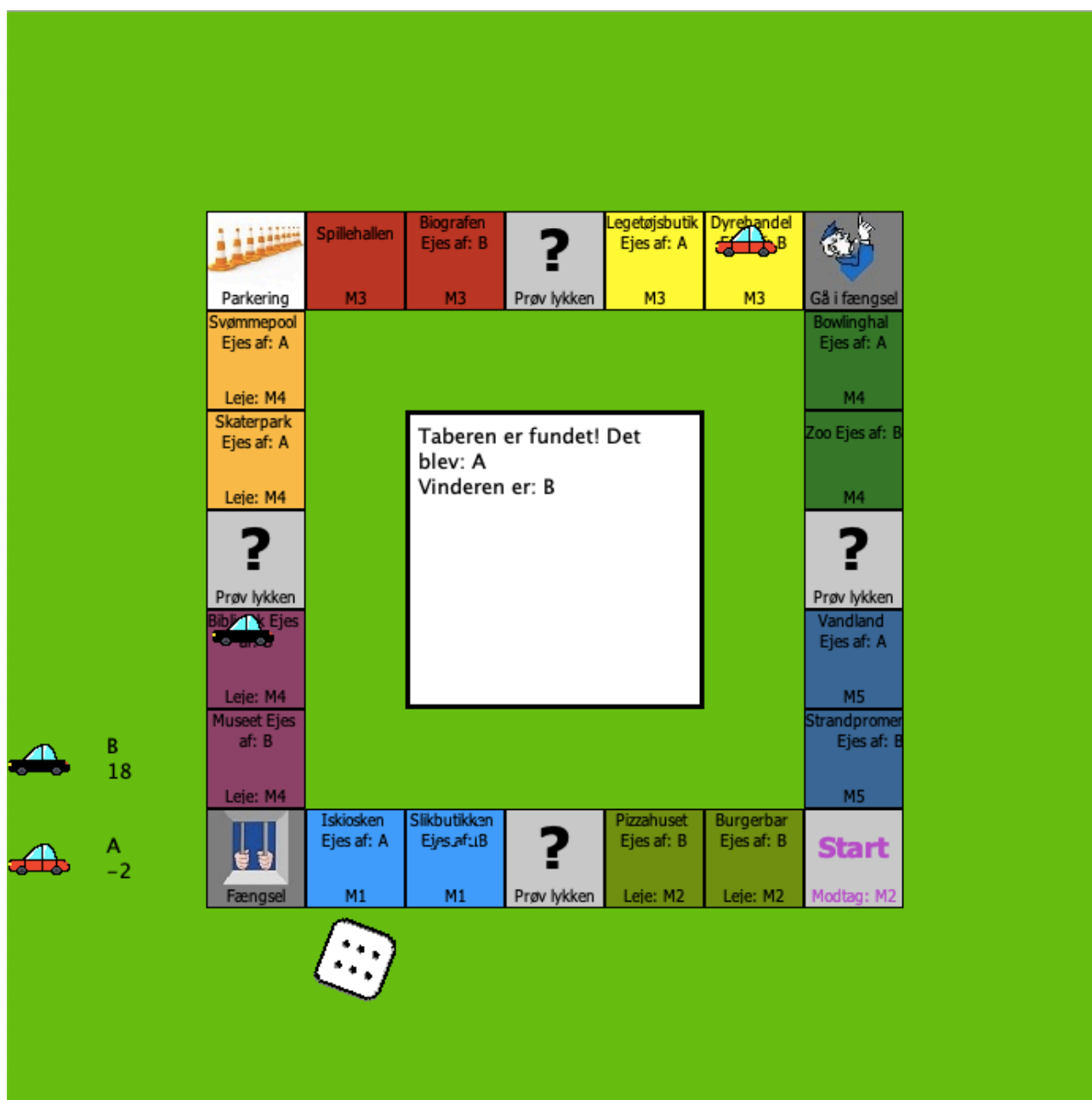
Testen konkluderes som værende succesfuld.

Testcase 3

I sidste testcase vil det blive testet hvorvidt en vinder bliver fundet når en spiller ikke kan betale. A ses her lige inden sin tur. A har 1 point og vil derfor med stor sandsynlighed tabe spillet hvis spilleren ender i et scenarie hvor han skal betale mere end 1 monet.

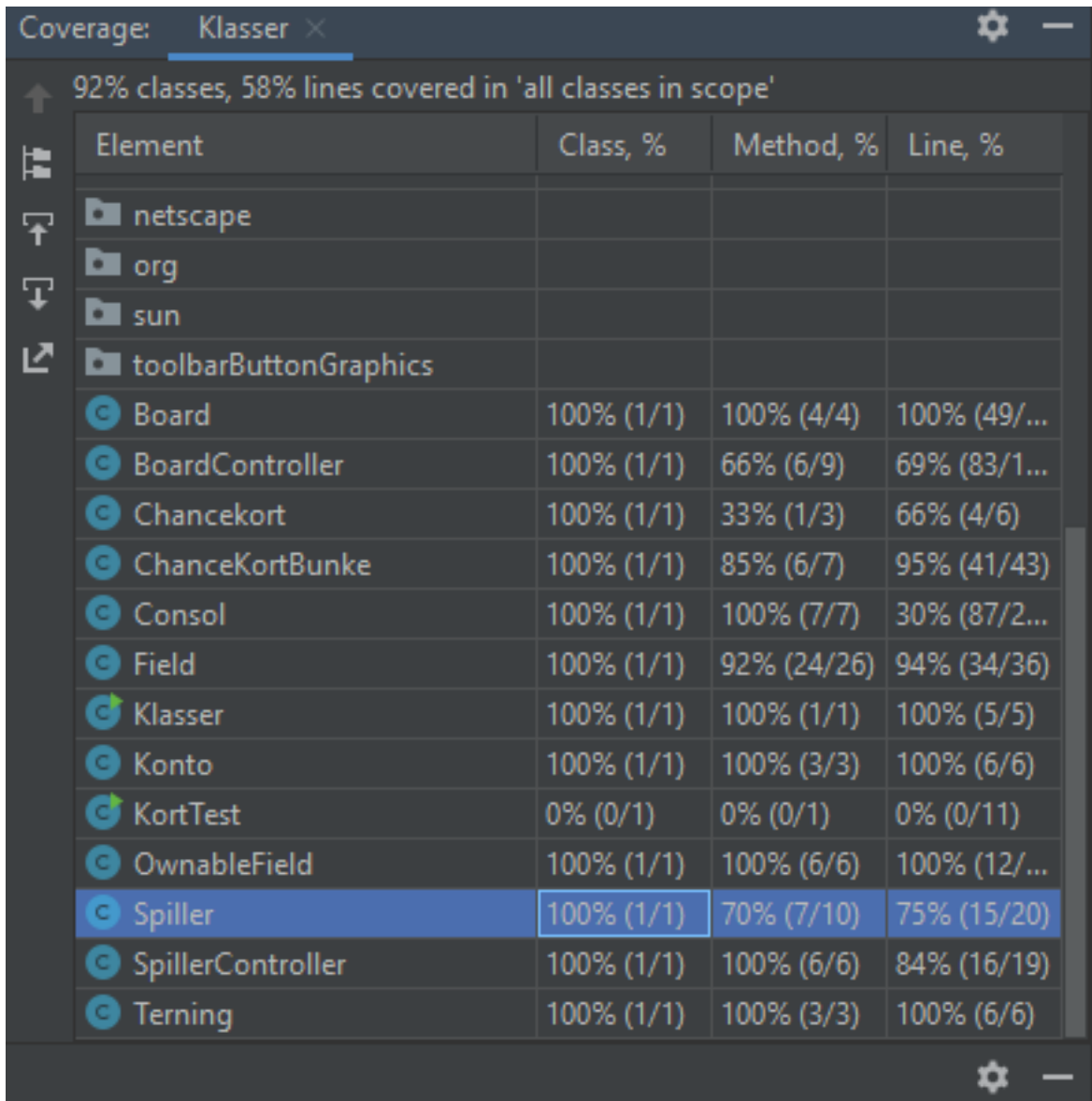


Efter A har slået lander han på "Dyrehandel" som er ejet af B. Dette stiller A i en situation hvor han skal betale flere penge end han har og bør derfor tabe.



Som det fremgår ovenfor, er dette også tilfældet.
Testen konkluderes til at være en succes.

Code coverage af vores klasser



The screenshot shows the 'Coverage: Klasser' window in IntelliJ IDEA. At the top, it states '92% classes, 58% lines covered in 'all classes in scope''. Below this is a table with columns: Element, Class, %, Method, %, and Line, %. The table lists various classes and their coverage percentages. The 'Spiller' class is highlighted in blue.

Element	Class, %	Method, %	Line, %
netscape			
org			
sun			
toolbarButtonGraphics			
Board	100% (1/1)	100% (4/4)	100% (49/...
BoardController	100% (1/1)	66% (6/9)	69% (83/1...
Chancekort	100% (1/1)	33% (1/3)	66% (4/6)
ChanceKortBunke	100% (1/1)	85% (6/7)	95% (41/43)
Consol	100% (1/1)	100% (7/7)	30% (87/2...
Field	100% (1/1)	92% (24/26)	94% (34/36)
Klasser	100% (1/1)	100% (1/1)	100% (5/5)
Konto	100% (1/1)	100% (3/3)	100% (6/6)
KortTest	0% (0/1)	0% (0/1)	0% (0/11)
OwnableField	100% (1/1)	100% (6/6)	100% (12/...
Spiller	100% (1/1)	70% (7/10)	75% (15/20)
SpillerController	100% (1/1)	100% (6/6)	84% (16/19)
Terning	100% (1/1)	100% (3/3)	100% (6/6)

På billedet ovenfor ses en code coverage test af vores main klasse, så testen viser hvilket klasser vi bruger i løbet af et helt spil. Kort test er en klasse vi har oprettet for at teste hvorvidt vores blandefunktion fungerer. Resten af vores klasser som har en relation til spillet bliver brugt, man kan forklare de metoder der ikke har den højeste % benyttelse, ved for eksempel at sige det ikke er alle chancekort der bliver brugt i løbet af et helt spil.

JUnit

Under klassen BoardControllerTest får vi lavet en Junit test, hvor vi tester metoden playerLandOnField(), først bliver der oprettet et board med boardcontroller, (spiller, spillerens navn, spillerens position, spillerens ID, spillerens konto) bliver oprettet med spillercontroller, vi får derefter kørt testen med assertEquals metoden.

Brugertest

Der er blevet lavet en brugertest, hvor brugeren ikke kunne komme videre fra startskærmen. Grunden var at brugeren ikke havde læst teksten der beder spilleren om at skrive hvor mange deltager der er med, hvilket skal være et tal mellem 2 og 4. Brugeren prøvede at skrive deres navn i feltet og da det ikke er et tal, sad brugeren fast. Efter at læse teksten på skærmen fandt brugeren ud af det, hvilket resulteret i at spillet kunne gennemføres problemfrit. Havde vi mere tid til at lave koden ville vi kunne sætte os ind i om gui teksten ville kunne blive gjort større.

Konfigurationsstyring

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_del3, så klikke på pilen ved siden af mappen src. Herfra kan du se pdf-filen 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 klasser 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 meddelelse hvor den beder dig opdatere. Tryk på den, lad den opdatere og så burde det hele virke. Det er også et krav at I bruger Maven til at hente junit.

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 scenarie 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 komplicerede 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 effektive 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:

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 Monopoly junior spil, som var med fokus på en række krav som spillet skulle kunne. Monopoly junior spillet kan spilles af 2 til 4 personer med deres eget navn, som unik indikator. Spillerne slår skiftevis med en terning, terningens øjne viser hvor mange felter de skal gå frem på det spille board der er blevet lavet. Det Board der er blevet lavet, indeholder 24 felter, som hver har et navn og pris. Spillet går i loop indtil en spiller ikke kan betale for at stå på en andens spillers felt eller ikke kan købe en grund, spilleren med flest point på det givne tidspunkt vinder spillet.

De 3 forskellige cases der blev testet, var succesfulde. Brugertesten påpegede at der muligvis kunne laves ændringer til test størrelsen. Kravene og brugervenligheden ville have været fuldført i højere grad, havde der været en længere tidsfrist. Det er endt med et Monopoly junior spil der funktionelt for 2 til 4 spiller.

Timeregnskab

Tidsforbrug i antal minutter					
UGE	46				
Navn/Dato	9/11/2020	10/11/2020	11/11/2020	12/11/2020	13/11/2020
Alexander Solomonsen	0	0	0	260	300
Andreas Vilholm V	0	0	0	260	300
Ahmad Shereef	0	0	0	260	300
Chenxi Cai	0	0	0	260	300
Isabel Grimmig Jensen	0	0	0	260	300
Oliver Fielder		0	0	260	300
UGE	47				
Navn/Dato	16/11/2020	17/11/2020	18/11/2020	19/11/2020	20/11/2020
Alexander Solomonsen	0	0	150	250	320
andreas Vilholm V	0	0	150	250	320
Ahmad Shereef	0	0	150	250	320
Chenxi Cai	0	0	150	250	320
Isabel Grimmig Jensen	0	0	150	250	320
Oliver Fielder	0	0	150	250	320
UGE	48				
Navn/Dato	23/11/2020	24/11/2020	25/11/2020	26/11/2020	27/11/2020
Alexander Solomonsen	0	0	170	300	360
andreas Vilholm V	0	0	170	300	360
Ahmad Shereef	0	0	170	300	360
Chenxi Cai	0	0	170	300	360
Isabel Grimmig Jensen	0	0	170	300	360
Oliver Fielder	0	0	170	300	360

Litteraturliste

- <https://github.com/diplomit-dtu/MatadorGUIDe/tree/master/src/main/java> - (27/11 - 2020)
- CDIO del 2 - (27/11 - 2020)
- https://docs.google.com/document/d/1y_YFZXelKJql_rJER-NWdEx6VPFrUeMrW6AOvhWriq8/edit# (27/11 - 2020)
- Objektorienteret programmering i Java – Jacob Nordfalk (6. ugeve, 1 oplag)
- Applying UML and patterns – Craig Larman (3. Udgave, 2005)