



Technical University of Denmark

CDIO

32_del3

GRUPPE 32



LUCAS J. GRAMME
s164863



ALEXANDER
NØRGAARD
s182934



MADS JENSEN
s185124



FREDRIK MEJLSBYE
s185130



MARIA KAJGAARD
s185108

30. november 2018

Indhold

| | |
|--|-----------|
| 1 Concieve | 1 |
| 1.1 Resume | 1 |
| 1.2 Vision | 1 |
| 1.3 Kravliste samt prioritering (MoSCoW) | 2 |
| 1.4 Navneordsanalyse og domæne | 3 |
| 1.5 Domænemodel | 5 |
| 2 Design | 6 |
| 2.1 Design klassediagram | 7 |
| 2.2 GRASP | 7 |
| 2.3 Sekvensdiagram | 8 |
| 2.4 Systemsekvensdiagram | 9 |
| 3 Implement | 10 |
| 3.1 Versionsstyring | 10 |
| 3.2 Programmering | 12 |
| 3.2.1 Designklassediagram revideret | 12 |
| 3.2.2 Udeladelser | 13 |
| 3.2.3 Vores kode | 14 |
| 3.3 Udførelse af test | 16 |
| 3.3.1 Test af ID: 0001 | 16 |
| 3.3.2 Test af ID: 0002 | 17 |
| 3.3.3 Test af ID: 0003 | 19 |
| 3.3.4 Test af ID: 0004 | 20 |
| 3.4 Brugertests | 22 |
| 3.4.1 Resultat af brugertest: | 22 |
| 3.5 Import af JUnit fra Maven | 23 |
| 3.6 Platforme og version | 25 |
| 4 Documentation | 26 |
| 4.1 Arv | 26 |
| 4.2 Abstrakte klasser | 27 |
| 4.3 Polymorfi | 28 |
| 4.4 Generering af Bat og Jar fil | 28 |
| 5 Bilag | 30 |
| 5.1 Klassediagram | 30 |
| 5.2 Monopoly junior spilleregler | 31 |
| 5.3 Brugertest | 35 |
| 5.4 Test Krav | 35 |
| 5.5 Test cases | 36 |

1 Concieve

1.1 Resumé

Vi har efter kundens ønske udviklet et Monopoly Junior-inspireret brætspil for 2-4 personer, som vindes ved at når en spiller går konkurs og spilleren med højest pengebeholdning vinder spillet. Specifikke krav til spillet er blevet givet af kunden, samt en liste over yderligere opgaver fra vores projektleder. Kravene er beskrevet yderligere i vores afsnittet *Kravliste*.

Ud fra prioriteringen i kravlisten, da er alle krav implementeret undtagen kravet angående fængselstraf, som også var det lavest prioriteret. Det var dog givet i opgavebeskrivelsen, at enkelte regler måtte udelades i implementeringen, hvilket også er derfor dette krav var nedprioriteret.

Efter kravene var blevet defineret og prioriteret, da blev vores modeller konstrueret, så der var en klar sammenhæng hele projektet igennem, samt en klar tråd mellem faserne *concieve, design, implement* (foruden *operate*, som vi så ikke har benyttet os af i dette projekt)

Under implementeringen blev den tildelte GUI modificeret så den havde de ønskede funktioner. Der blev kun oplevet få komplikationer med implementeringen, samt der i afsnittet *3.2.1 Designklasse diagram revideret* er udkommenteret dele fra vores designfase. Der kan i dette afsnit også læses om dé komplikationer der under projektet er opstået, og hvordan disse er blevet håndteret og derefter løst.

I testfasen blev vores program testet med succes og testene havde det forventelige udfald. Det konkluderes at programmet er konstrueret succesfuldt, og er klart til at blive overleveret til kunden.

Der har projektet igennem været fokus på benyttelse af GRASP, samt intelligent brug af arv, polymorfi samt abstrakte klasser.

1.2 Vision

Kundens ønske er et Monopoly Junior spil, som kan spilles af 2-4 spillere. Spillet skal være så virkelighedstro som muligt, dog er der plads til få ændringer såfremt det fremgår i kravprioriteringen.

Monopoly er traditionelt brætspil der er publiceret for næsten 100 år gammelt imens den forgængere har eksisteret i over et århundrede. Spillet er et must-have og har prydet familiehjem af alle nationaliteter i mange år.

Kort fortalt så er spillets hovedformål at opbygge så stor pengebeholdning som muligt. Det starter med at der bliver slæbt med en terning som afhængig af sin værdi fra 1-6 afgør hvor langt ens figur rykker sig på spillebrættet.

Der findes forskellige typer af felter, som alle har forskellige konsekvenser. Felterne er klassificeres som hhv.: grunde, chance-felt samt 4 hjørnefelter (start, parkering, fængselbesøg og fængsel).

Når en spiller lander på en grund skal spilleren, hvis grunden ikke allerede er ejet af en spiller, købe grunden. Hvis denne grund er ejet af en anden spiller skal man til gengæld betale husleje (svarende til grundens værdi).

Hvis en spiller lander på et chancefelt - da skal spilleren trække et chancekort som både kan give et positivt, såvel som negativt udfald for spilleren.

Ud af de 4 hjørnefelter da har felterne parkering samt fængelsesbesøg intet udfald, og fungere som blanke felter, hvorimod man ved start modtager 2\$ og på fængselsfeltet skal man sidde over en runde.

Spilleren med den højeste pengebeholdning, når den første spiller går konkurs vinder.

Det er nu tid til at dette spil skal digitaliseres, og dette ønskes gjort så at spillet så vidt muligt forbliver det samme som vi alle kender og elsker.

Det er nu vores arbejdsopgave at lave systemet så det har så mange af de oprindelige spillefunktioner med som muligt. Dog er det først og fremmest en prioritet at spillet kan spilles, hvorefter nogle funktioner har højere prioriteter end andre, hvilket kan læses i afsnittet *Kravliste*.

For samtlige monopoly junior regler, se bilag 4.2.

1.3 Kravliste samt prioritering (MoSCoW)

Funktionelle krav vises med sort skrift

Ikke-funktionelle krav vises med blå skrift

Must have:

- Spillet skal være mellem 2-4 personer.
- Der slås med én terning.
- Terningværdien i et slag svarer til det antal felter en spiller må rykke frem på spillepladen.
- Ved hver tur fortsætter spilleren fra fletet spilleren sluttede på i forgangne runde.
- Man går i ring på spillebrættet
- Hver spiller starter med et givet beløb, der afhænger af hvor mange der spiller spillet
 - v. 2 spillere: \$20 hver
 - v. 3 spillere: \$18 hver
 - v. 4 spillere: \$16 hver
- Spillet er slut når en spiller løber tør for penge (Altså går konkurs).
- Spillet vindes ved at have flest penge når en anden spiller er gået konkurs.
- Det er kun muligt at rykke fremad.
- Man skal købe et felt hvis det er ledigt (ingen ejer det).
- Lander man på et felt som allerede er ejet, da skal man betale husleje (beløbet som står på fletet).
- Man modtager \$2 hver gang man lander på eller passerer START.
- Lander man på én af de 4 "CHANCE-felter på brættet, da får man et chancekort. Instrukserne på chancekortet udføres.
- Spillet skal kunne spilles på maskinerne i DTU's databarer uden bemærkelsesværdige forsinkelser.

Should have:

- I tilfælde af uafgjort (ved slut), da medregnes værdi i ejendomme

Could have:

- Ejendomme er delt op i par med farkekodert
 - Hvis en spiller ejer begge ejendomme i samme farve, da er huslejen det dobbelte af det beløb der står på fletet.

Want to have:

- Når der landes på "GÅ I FÆNGSEL" da skal man ved næste tur betale \$1, eller bruge chancekortet "Du løslades uden omkostninger- hvis man har der. Derefter kan man tage sin tur.

1.4 Navneordsanalyse og domæne

I navneordsanalysen vil vi forsøge at finde på så mange navneord som muligt, der kan have en relation til spillet. Derefter vælger vi de mest relevante og benytter dem som navn på klasser og attributter i vores design.
Vi tager udgangspunkt i use casen "Spil spillet"

Liste med vilkårlige navneord

- *Spillebræt*
- *Terning*
- *Figur*
- *Bank*
- *Bolig*
- *Chancefelt*
- *Hotel*
- *Bil*
- *Start*
- *Spiller*
- *Grunde*
- *Husleje*
- *Runde*
- *Chancekort*
- *Konto*
- *Felt*
- *Fængsel*
- *Parkering*

Navneord som er udvalgt

- *Chance-kort*
 - Et sæt á 24 kort som alle har en konsekvens printet derpå. Udfaldet kan både være godt eller dårligt.
- *Spillebræt/Bræt*
 - Pladen hvorpå spillet spilles
- *Terning*
 - Almindelig 6-sidet terning nummereret fra 1-6.
- *Figur*
 - En spillers 'brik' som flyttes rundt på spillepladen og indikere den pågældende spillers position
- *Spiller*
 - En medvirkende i spillet. Der er mellem 2-4 spillere.
- *Husleje*
 - Prisen som betales når en anden spiller (end ejeren af grunden) lander på feltet

- *Grund*

- Felter som kan købes for symbolske beløb, hvorefter den pågældende spiller ejer det, og kan indkræve husleje

- *Felt*

- Brikernes mulige placeringer, hvormed spillet er opbygget

- *Konto*

- En spillers bankkonto

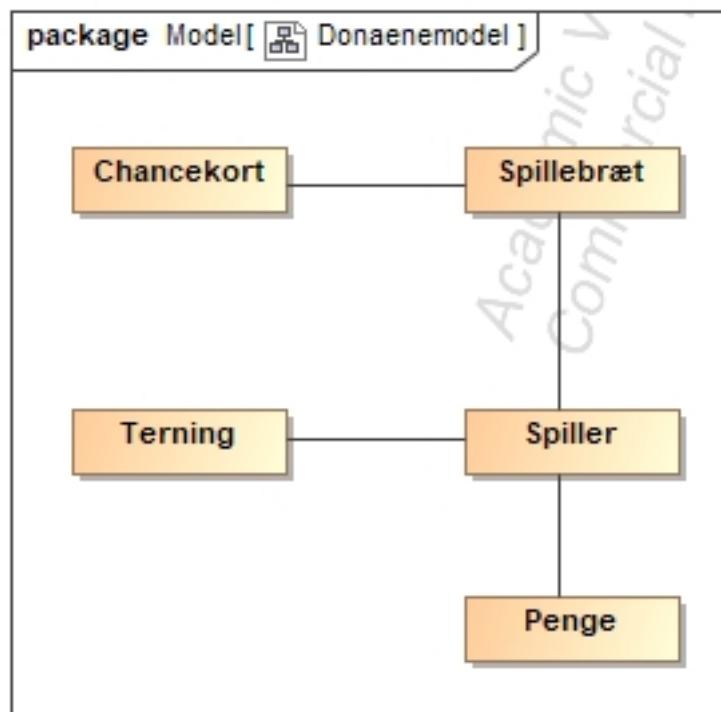
Liste med verber

- Slå terning
- Ryk figur
- Betal husleje

Hvilke navneord som er attributter

- *Spillebræt* [Board]
- *Terning* [Die]
- *Spiller* [Player]
- *Grunde* [Property]
- *Konto* [Account]

1.5 Domænemodel



Figur 1: Domænemodel

Dette er vores domænemodel. Den viser det simple forhold mellem de forskellige elementer som dette spil indeholder: chancekort, spillebræt, Spiller, terning og penge. Dette er grundelementerne i spillet.

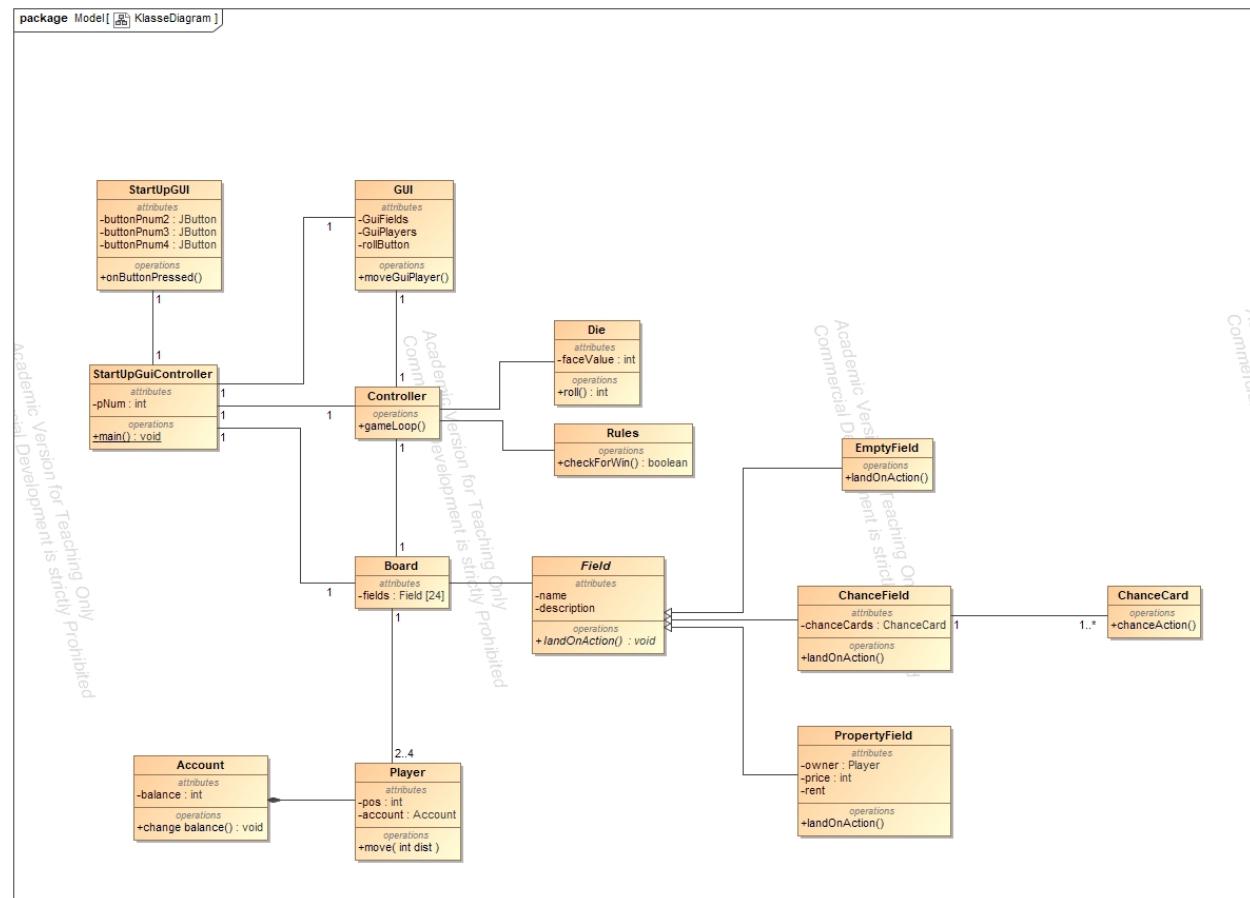
2 Design

Use-case beskrivelse

| Use case: Spil spillet | |
|--|--|
| ID: 1 | |
| Scope: | Matador-junior spil. |
| Brief description: | Spillerne spiller et spil matador. |
| Primary actors: | Spillere. |
| Secondary actors: | None. |
| Preconditions: | Der er 2-4 spillere tilstede. |
| Main flow: | <ol style="list-style-type: none"> 1. En spiller ruller med terningen 2. Spilleren rykker frem et antal felter der svarer til terningens øjne. 3. Hvis feltet der landes på kan købes, købes dette. 4. Passeres start tildeles der et kapital indskud. 5. Næste spiller går således til punkt 1 |
| Postconditions: | En spiller har vundet. |
| Alternative flows: | <ol style="list-style-type: none"> a. Hvis feltet eges af en anden spiller betales der husleje til den gældende spiller. b. Er der ikke kapital nok til at købe feltet eller betale leje, sluttes spillet og spilleren har således tabt. c. Når man landet på et chancefelt, trækkes et chancekort og beskeden på chancekortet udføres. |
| Technology and data restrictions: | <ul style="list-style-type: none"> -Udviklet på Windows 10 - 64 bit -Udviklet til at understøtte 1920x1080 skærmopløsning, andre oplosninger kan give problemer med at vise programmet korrekt |

2.1 Design klassediagram

Nedenfor ses måden hvorpå vi har valgt at designe vores program.



Figur 2: Klassediagram

2.2 GRASP

Vi har i dette projekt arbejdet med og tænkt over vores GRASP-mønstre. Vi har brugt tid på at designe et designklassediagram, som har lav kobling og høj cohesion. For eksempel har vi en klasse som står for tre hovedelementer. Hovedelementet er StartUpController som er ansvarlig for GUI, Controller og Board. Dette gør, at de andre klasser ikke behøver at lave andre objekt udover deres eget ansvarsområde. Vi har i CDIO3 fremfor i CDIO2 fjernet flere af de funktionelle dele af game-loopet og lagt dem ud i de forskellige klasser som controlleren kender. Controlleren kender til Die Rules klasserne, som indeholder de funktionelle dele af game-loopet.

Vi har også vores creator, controller og informations expert.

Creator

- Player er en Creator til account, da den har det største og eneste relation til account.
- StartUpController er en creator til StartUpGUI, da har en stærk realtion og opretter objektet.

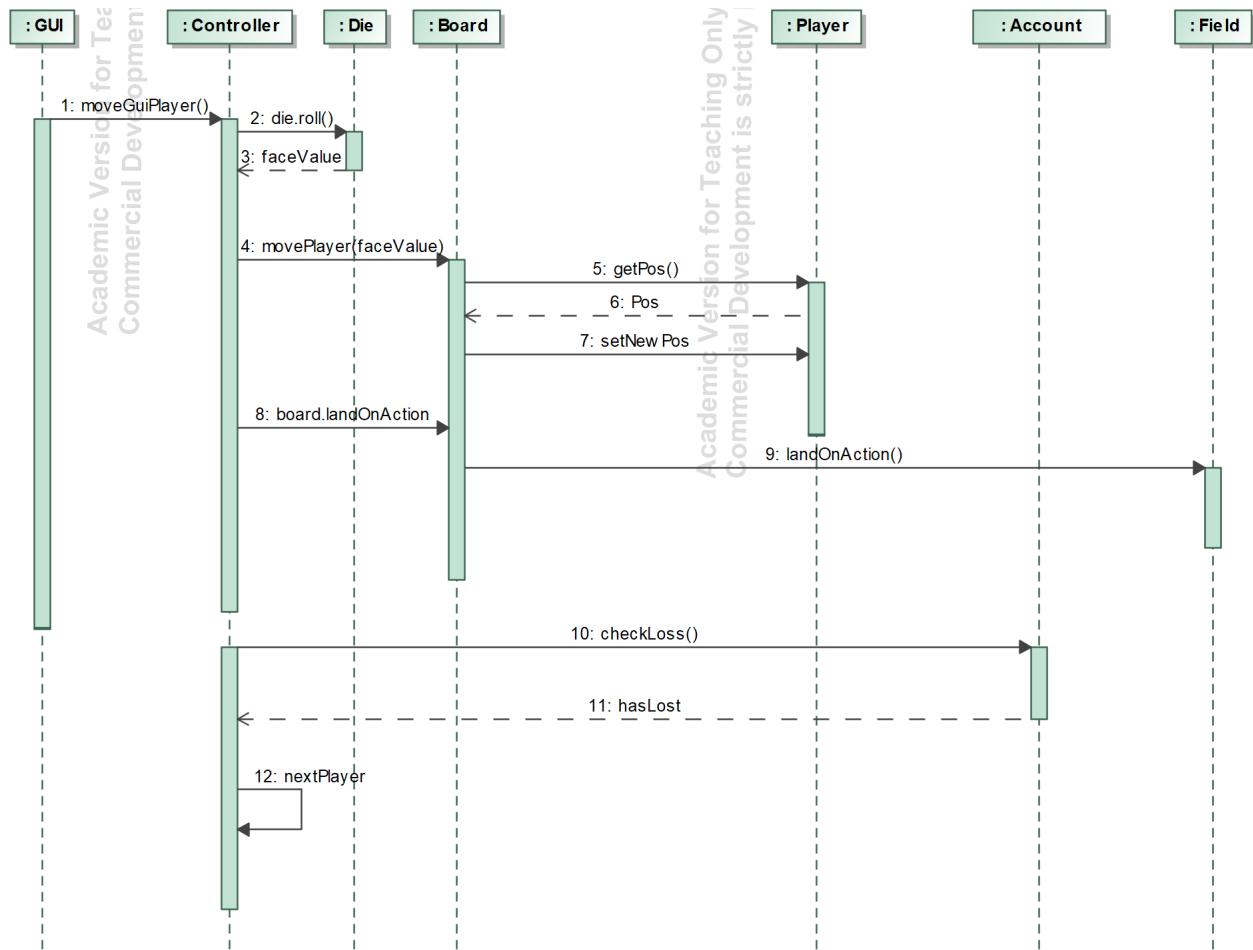
Information Expert

- Board er vores Information Expert, da den placerer ansvaret til den klasse, som har oplysningerne til at løse opgaven. Fordelen ved at gøre denne klasse Information Expert er, at det giver høj binding og lav kobling.

Controller

- Controller klassen er vores Controller, da den har ansvaret for systemets andre objekter og fastlægger rækkefølgen på udførsel af handlingerne. Fordelen ved dette er, at det understøtter høj binding og lav kobling.

2.3 Sekvensdiagram



Figur 3: Sekvensdiagram

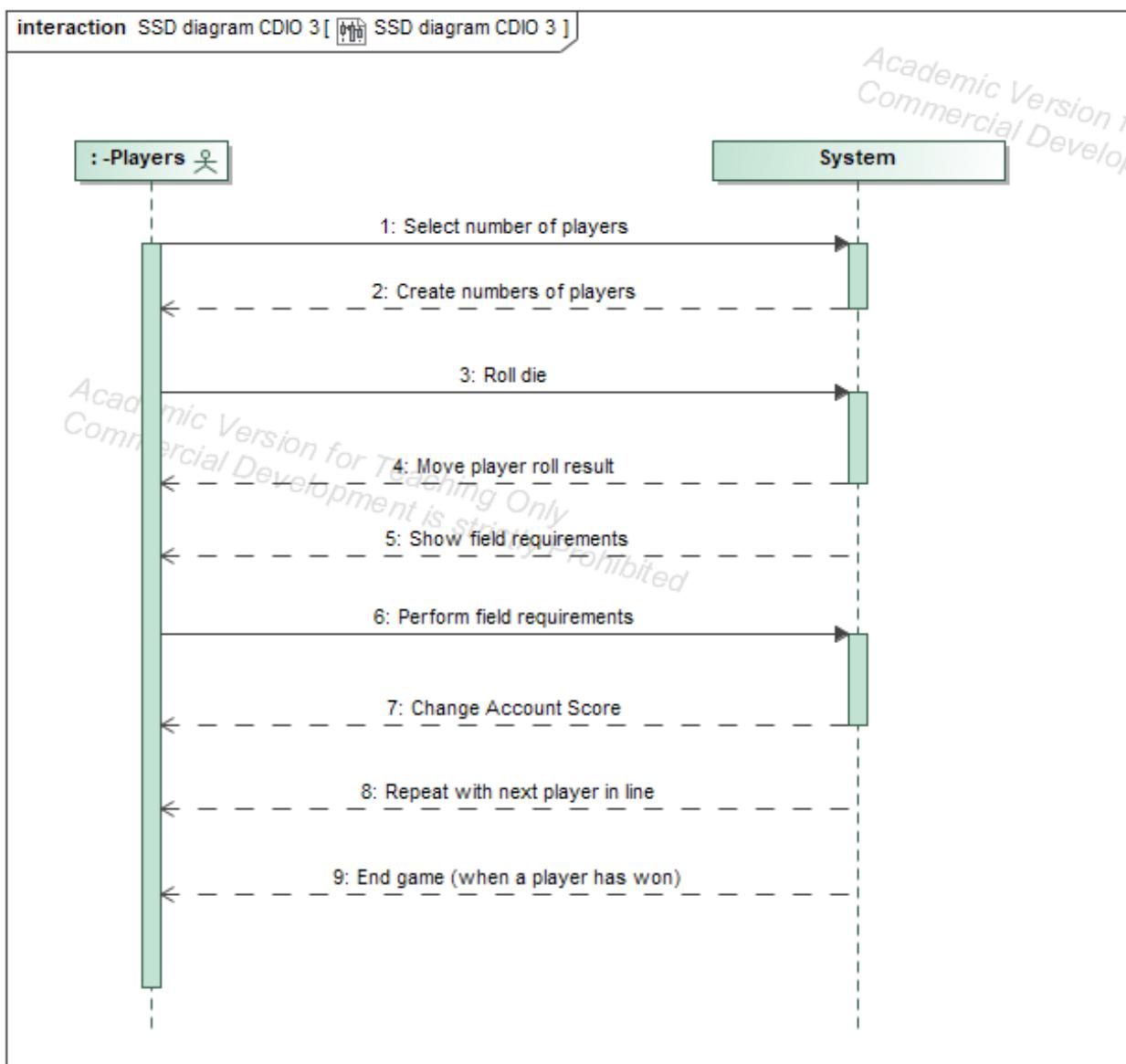
I dette sekvensdiagram har vi beskrevet spillets gang, når alle klasser er blevet oprettet som objekter af vores controller. Når controlleren har skiftet til næste spiller igennem vores `nextPlayer()` function, bliver vores bruger bedt om at række deres brik gennem GUI'en. Når spilleren trykker på en knap, bliver signalet "movePlayer" sendt til vores controller, som derefter kører igennem game loop'et.

I loopet kalder controlleren en `face value` fra vores `die`-objekt, som den derefter kan sende videre til `board`, når den kalder `movePlayer`-metoden. Herfra kører `board` `getPosition` og derefter `set newPosition`, for den player der har slået slaget.

Når den nye position er sat, får controlleren vores `board`-objekt til at kalde en `landOnAction` fra en af vores 3 subklasser af `field`-klassen: `ChanceField`, `EmptyField`, og `PropertyField`. Alt efter hvilken man lander på, vil der ske en forskellig ting, som vi ikke vil komme ind på her i diagrammet.

Når denne `landOnAction` er gennemført vil vores controller kontrollere om en spiller har tabt spillet. En spiller har tabt spillet når han/hun ikke kan betale husleje eller opkøbe en grund. Hvis der ikke er en taber, vil controlleren køre loopet forfra.

2.4 Systemsekvensdiagram



Figur 4: Systemsekvensdiagram

Beskrivelse

Vores SSD viser at man som det første skal vælge hvor mange spillere der skal spille spillet (2 til 4 spillere), hvorefter systemet vil tilpasse spillet til antallet af spillere.

Derefter slår første spiller med terningen, hvorefter den pågældende spillers brik bliver rykket det antal felter, svarende til terningens øjne.

Herved viser GUI'en hvad der skal ske på det pågældende felt, og derefter udfører spilleren handlingen. Hvis handlingen har en påvirkning på spillerens pengekonto, da ændres denne.

Dette forsætter med næste spiller osv. indtil én af spillerne er gået konkurs, og spillet slutter.

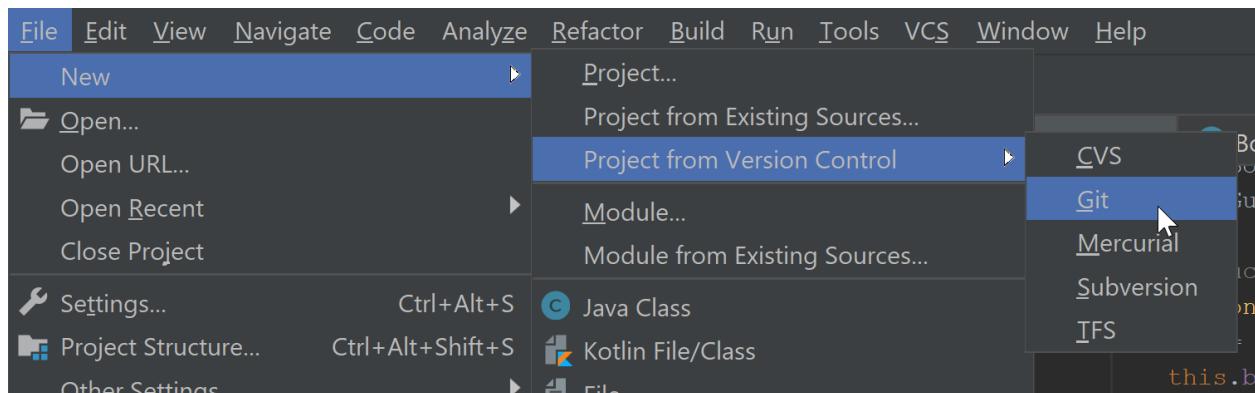
3 Implement

3.1 Versionsstyring

Til denne rapport har vi benyttet git til at håndtere vores versionsstyring. Vores repository er blevet oprettet og opbevares i GitHub. Vi vil nu vise hvordan man importerer sådan et git-repository i IntelliJ:

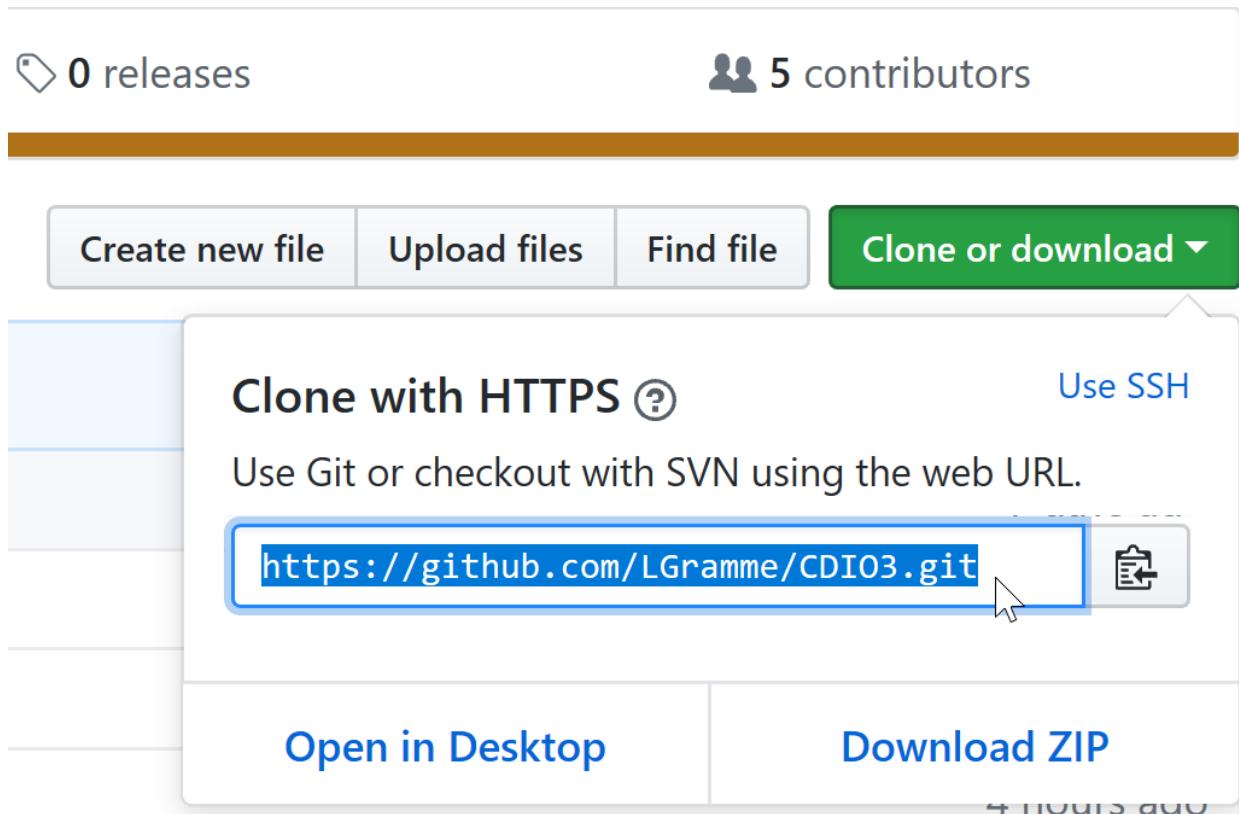
For at importere et git-repository i IntelliJ, kan man vælge at starte et helt nyt projekt, hvor man trækker et repository ind, direkte fra en URL. Man starter med at trykke på "File" oppe i venstre hjørne og går derefter af vejen: *File → New → Project from version control → Git.*

Nedenfor ses et screenshot af dette.



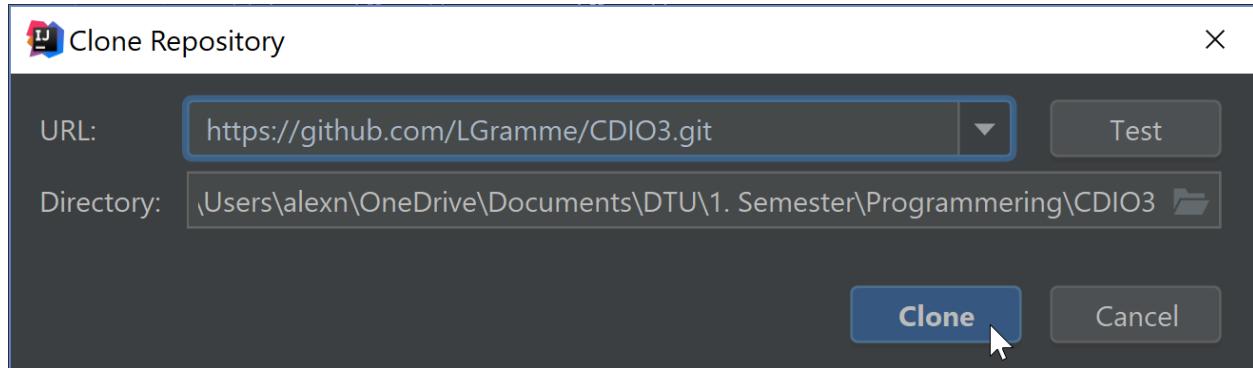
Figur 5: Screen shot, step1

Hertil skal du bruge en URL på det git repository man gerne vil clone. I vores tilfælde finder vi denne URL på GitHub. Når man har fundet det ønskede repository, skal man trykke på knappen "Clone or download". Herfra kan vi kopiere den nødvendige URL:



Figur 6: Screen shot, step2

Denne URL kan du nu indsætte i IntelliJ:

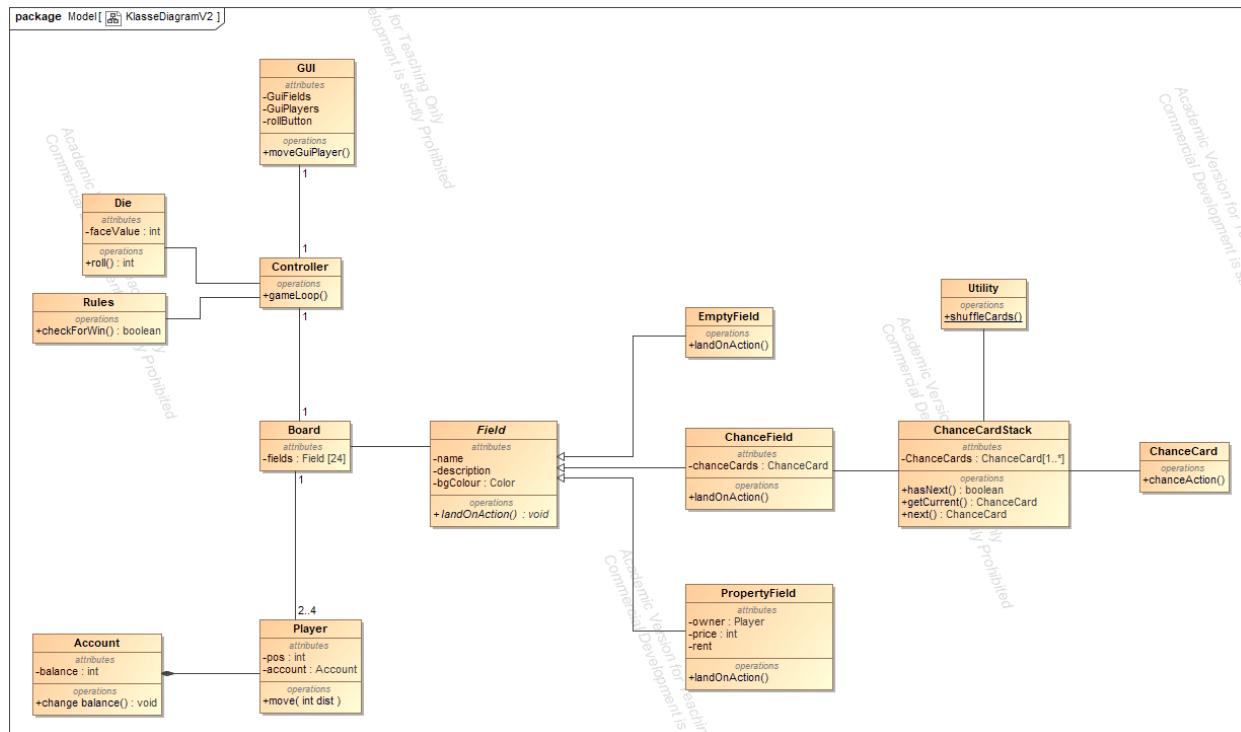


Figur 7: Screen shot, step3

Nu vil IntelliJ importere hele indholdet fra vores git-repository.

3.2 Programmering

3.2.1 Designklassediagram revideret



Figur 8: Designklassediagram over vores endelige produkt

Designklassediagrammet ovenfor på figur 8 viser hvordan det endelige design af vores spil kom til at se ud. Vi har foretaget følgende ændringer:

- StartUpController & StartUpGui er blevet slettet.
 - Disse to klasser skabte en unødvendig kompleksitet til vores program. Særligt da vores startUpGui kun havde en funktion som kunne laves i vores matador GUI.
 - Grunden til at disse klasser blev lavet fra starten, var en grundlæggende misforståelse af GUI'en.
- Klassen der hed GUI er blevet omdøbt til GuiHandler.
 - Den oprindelige klasse var ment til at være klassen som styrede GUI'en tidligere. Men vi har senere hen lavet den til en wrapper-klasse, kaldet GuiHandler.
 - En anden grund til det var, at vi ikke skulle forveksle vores GUI-klasse med den udleverede GUI-klasse.
- Tilføjet ChanceCardStack-klassen.
 - Denne klasse er oprettet for at håndtere det array af chanceCards som skal være i spil. Dette tager noget funktionalitet, som ellers ville have været i ChanceField klassen.
 - Dette er en singleton, så vi sørger for at vi kun har en stack af ChanceCards som bliver givet til ChanceField-klassen.
- Utility-klassen.
 - Utility-klassen har en statisk metode, som blander et array af ChanceCards.

3.2.2 Udeladelser

I denne sektion vil vi fremlægge de udeladelser vi har valgt at have i vores program. Årsagerne vil også blive diskuteret.

- Fængsel
 - Under afsnittet kravliste, i sektion 1.3 valgte vi at nedprioritere fængselsfeltet, da det ikke er så essentielt for spillerens oplevelse. Da vi kom under tidspres, blev dette fravalgt som det første.
- Get out of jail free card
 - I kraft af at vi ikke har inkluderet fængselsfeltet har vi også udeladt dette kort.
- Brugerhandlinger
 - Vi har valgt ikke at inkludere brugerhandlinger, som for eksempel, "ryk op til 5 felter frem".
 - Da dette skulle interagere med GUI'en, men da vi havde store problemer med at få denne til at fungere ind til sent i forløbet, blev dette også udeladt.
- Monopoly's møntfod
 - Monopoly har et bestemt tegn for spillets møntfod, som er et M med en streg igennem. Da dette ikke er en del af ASCII har vi valgt at bruge tegnet \$.

3.2.3 Vores kode

I vores programmering af spillet, har vi haft nogle overvejelser i forhold til specifikke klasser og metoder. Disse vil vi diskutere herunder:

takeTurn()-metoden

Denne metode ligger i Controller-klassen og står for at kalde alle metoder der skal til for at en spiller gennemfører sin tur.

I metoden bedes spilleren om at slå og slaget resultat vises. vi bruger så en sleep() funktion på linie 8 til at lade spilleren se terningen inden deres grafiske placering så bliver opdateret på linie 10. Pausen er på 400 milisekunder eller 0,4 sekunder. Vi har gjort dette så der ikke sker for meget på skærmen på en gang. det ville blive en smule kaotisk ellers.

På linie 13 begynder der så et do-while loop som indeholder al koden til at feltets specielle regler udføres. Vi har lagt det i en do-wihle løkke for at denne handling altid udføres mindst en gang. Den første linie i løkken sætter så dets egen betingelse til falsk således at det kun bliver udført én gang medmindre at et felt har en handling der ændrer på dette.

Det er kun chancekort der kan det. Hvis man får et chanceKort der flytter dig vil den også sætte variablen isActive til true så man igen kommer en gang igenem løkken og udfører den handling som feltet man landede på kræver.

I slutningen af hver løkke opdateres GUI'en så spilleren kan se alle trin i hvor de lander og hvad der sker med dem.

```

1 private void takeTurn() throws InterruptedException {
2     Field currentField;
3     Player selectedPlayer = board.getPlayers()[ currentPlayer ];
4
5     guiHandler.waitForRoll("Player" + (currentPlayer + 1) + " please roll a die");
6     int rollValue = die.Roll();
7     guiHandler.showDie(rollValue);
8     Thread.sleep(400); // setting delay between roll and moving of the car.
9     board.movePlayer(selectedPlayer, rollValue);
10    guiHandler.updateGui(board.getPlayers(), board.getFields());
11
12    do {
13        selectedPlayer.setIsActive(false);
14        currentField = board.getFields()[ selectedPlayer.getPos() ];
15        guiHandler.msgInMidle(currentField.getMessage(selectedPlayer));
16        currentField.landOnAction(selectedPlayer, board.getPlayers(), board.getFields());
17        board.UpdateRent();
18        guiHandler.updateGui(board.getPlayers(), board.getFields());
19    } while (selectedPlayer.getIsActive());
20 }
```

Listing 1: takeTurn()-method from Controller.java

ChanceCardStack som singleton

Vores ChanceCardStack er tidligere blevet kaldt for en singleton, det vil vi her uddybe. Vi har i 02313 hørt om singletons, klasser der kun tillader at man skaber et objekt af dem og så returnerer dette når man fremover beder om et objekt af klassen. Dette designmønster har vi brugt til at lave vores ChanceCardStack-klasse. Vi gjorde dette fordi at vi gerne ville give alle ChanceField-instanser den samme stak af kort, men ikke ville lave statiske variable. For at lave en singleton fulgte vi mørnstret ret slavisk. I ChanceCardStack-klassen lavede vi en statisk variabel af klassens egen type. Vi lavede en privat konstruktør og en public metode getStackInstance() som returnerer en instans af klassen selv. Hvis den stiske instans gemt i klassen ikke referer til et objekt endnu kaldes konstruktøren. I alle tilfælde ender det med at metoden returnerer den statiske instans af klassen.

```

1  private static ChanceCardStack stackInstance = null;
2
3  public static ChanceCardStack getStackInstance() {
4      if (stackInstance == null) {
5          stackInstance = new ChanceCardStack();
6      }
7
8      return stackInstance;
9  }
10
11 private ChanceCardStack(){
12     cardNum = 0;
13
14     chanceCards = new ChanceCard[] {
15         //TODO make the rest of the chance cards
16         new ChanceCard(ChanceCardAction.MoveToStart, 2),
17         new ChanceCard(ChanceCardAction.MoveByAmount, 5),
18         new ChanceCard(ChanceCardAction.MoveToColour, Color.red),
19         new ChanceCard(ChanceCardAction.MoveByAmount, 1),
20         new ChanceCard(ChanceCardAction.ChangeBalance, -2),
21         new ChanceCard(ChanceCardAction.MoveToColour, Color.blue),
22         new ChanceCard(ChanceCardAction.MoveToColour, Color.cyan),
23         new ChanceCard(ChanceCardAction.Birthday, 1),
24         new ChanceCard(ChanceCardAction.MoveToColour, Color.pink),
25         new ChanceCard(ChanceCardAction.ChangeBalance, 2)
26     };
27
28     Utility.shuffleCards(chanceCards);
29 }
```

Listing 2: ChanceCardStack dannes som singleton

shuffle()-metoden

I pakken TechnicalServices har vi en Klasse kaldt Utilities. In that class there is a method called shuffle(). Denne metode er statisk da den skal kunne bruges på et hver array i vores program. Det bliver lige nu kun brugt i ChanceCardDeck men vi valgte at lave det til en statisk metode i Utility-klassen da vi kunne forestile os at samme kode kunne bruges på andre arrays såsom fields arrayet hvis man ville have et tilfældigt bræt i en senere version eller lignende.

I metoden bruger vi SecureRandom til at genererer tilfældige tal. Vi går igennem arrayet og bytter hvert element i arrayet ud med et element på en tilfældig plads fra i til slutningen af arrayet.

```

1 /**
2 * Method to shuffle values in array changed to fit shuffling cards.
3 * @param array This is the array to be shuffled.
4 */
5 public static void shuffleCards(Object[] array) {
6     int n = array.length;
7     SecureRandom random = new SecureRandom();
8
9     for (int i = 0; i < array.length; i++) {
10
11         int randomValue = i + random.nextInt(n - i);
12         Object randomElement = array[randomValue];
13         array[randomValue] = array[i];
14         array[i] = randomElement;
15     }
16 }
```

Listing 3: shuffle()-metode fra Utility

3.3 Udførelse af test

Vi har defineret en række tests vi kunne foretage os af vores program, hvoraf vi har valgt dem der er mest relevant for os. Alle vores test cases kan ses i bilag 5.5. Ud af alle testsne har vi udvalgt test case nr: 0001, 0002, 0003, 0004.

3.3.1 Test af ID: 0001

Der skal undersøges om antallet af spillere der vælges er korrekt, samt at disse har den rigtige start kapital. Således er der udført en JUnit test, hvor koden kan ses forneden:

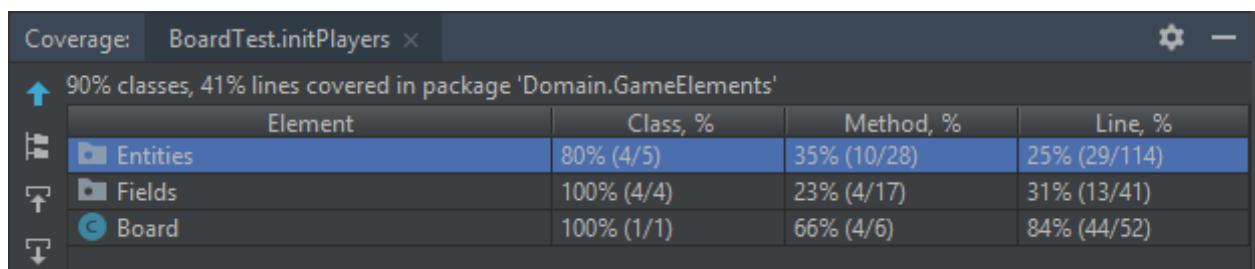
```

1  @Test
2    void initPlayers() {
3      Board board = new Board();
4      // Test for 2 players
5      board.initPlayers(2);
6      assertEquals(2, board.getPlayers().length);
7      assertEquals(20, board.getPlayers()[0].getAccount().getScore());
8      // Test for 3 players
9      board.initPlayers(3);
10     assertEquals(3, board.getPlayers().length);
11     assertEquals(18, board.getPlayers()[0].getAccount().getScore());
12     // Test for 4 players
13     board.initPlayers(4);
14     assertEquals(4, board.getPlayers().length);
15     assertEquals(16, board.getPlayers()[0].getAccount().getScore());
16   }

```

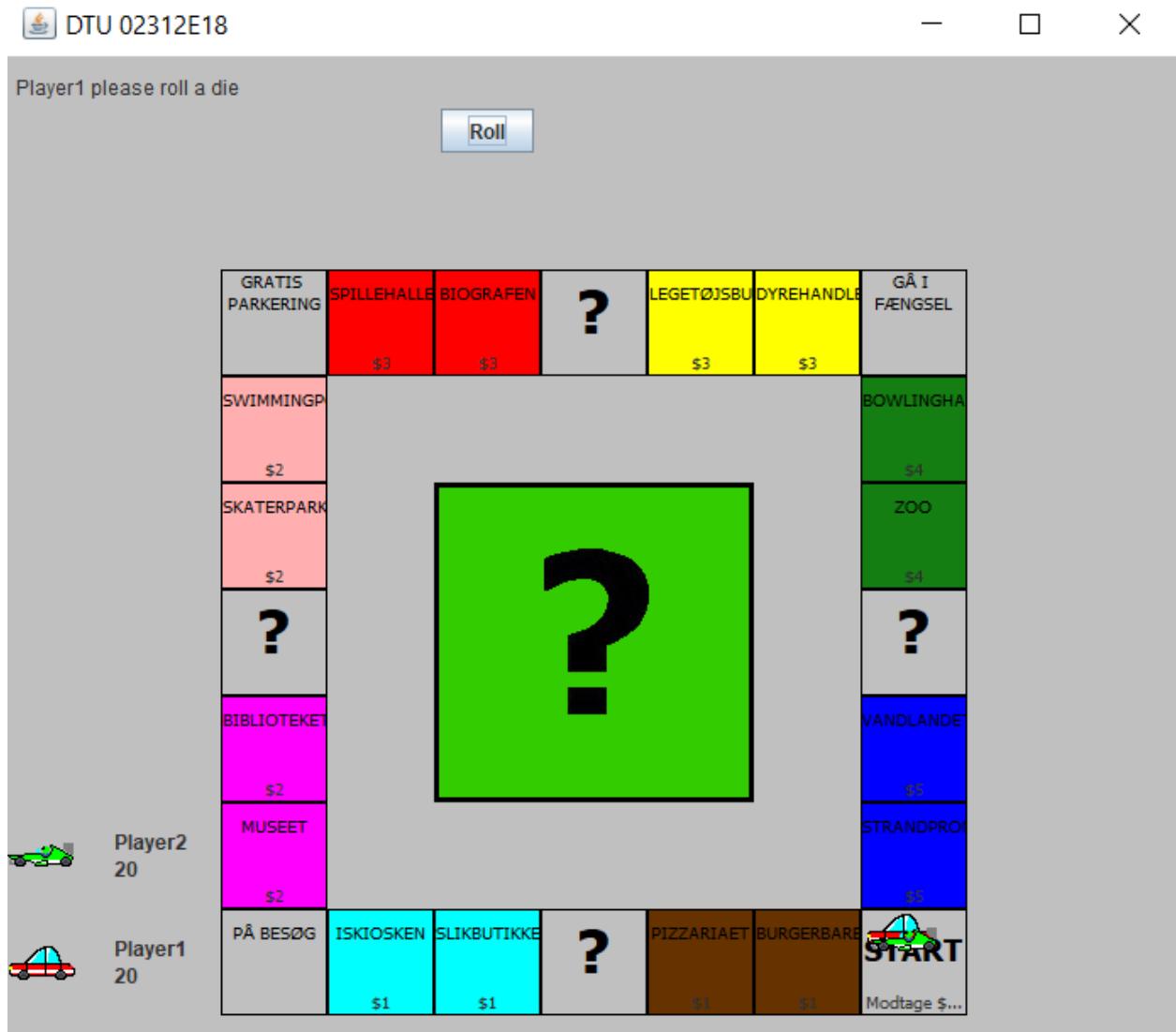
Listing 4: Test kode for test case 0001

Der kan ligeledes se at testen kommer rundt i flere forskellige områder af koden med en code coverage der kan ses på figur 26.



Figur 9: Code coverage for initPlayers()-method

Resultatet af testen kan således ses på den GUI der er blevet udviklet her forneden på figur 10.



Figur 10: Resultat af test 0001

3.3.2 Test af ID: 0002

I denne test skal der undersøges om hvorvidt der er funktionel bevægelighed på spillepladen.

```

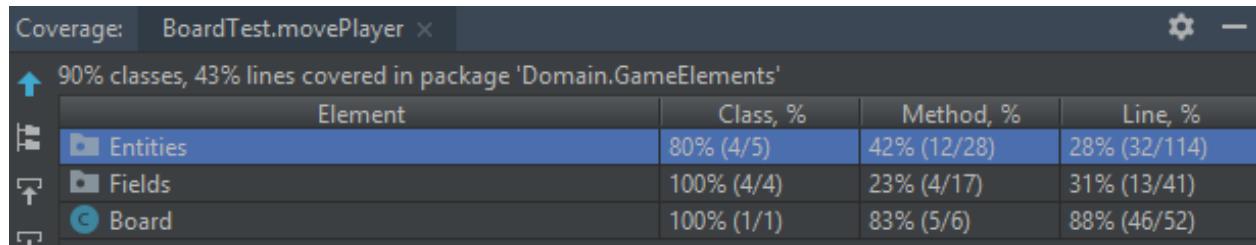
1 @Test
2     void movePlayer() {
3         Board board = new Board();
4         board.initPlayers(2);
5         Player player1 = board.getPlayers()[0];
6         Player player2 = board.getPlayers()[1];
7         // Save starting positions
8         int startPos1 = player1.getPos();
9         int startPos2 = player2.getPos();
10        // Test for accurate starting location
11        assertEquals(0, startPos1);
12        assertEquals(0, startPos2);
13        // Test movement
14        board.movePlayer(player1, 2);
15        board.movePlayer(player2, 3);
16        assertEquals(2, player1.getPos());
17        assertEquals(3, player2.getPos());
18        // Moving players to tile before START
19        player1.setPos(23);
20        player2.setPos(23);
21        // Setting balance to pre-defined value
22        int player1Bal = player1.getAccount().getScore();

```

```
23     int player2Bal = player2.getAccount().getScore();  
24     // Test for balance increase  
25     board.movePlayer(player1, 1);  
26     board.movePlayer(player2, 1);  
27     assertEquals(player1Bal + 2, player1.getAccount().getScore());  
28     assertEquals(player2Bal + 2, player2.getAccount().getScore());  
29 }
```

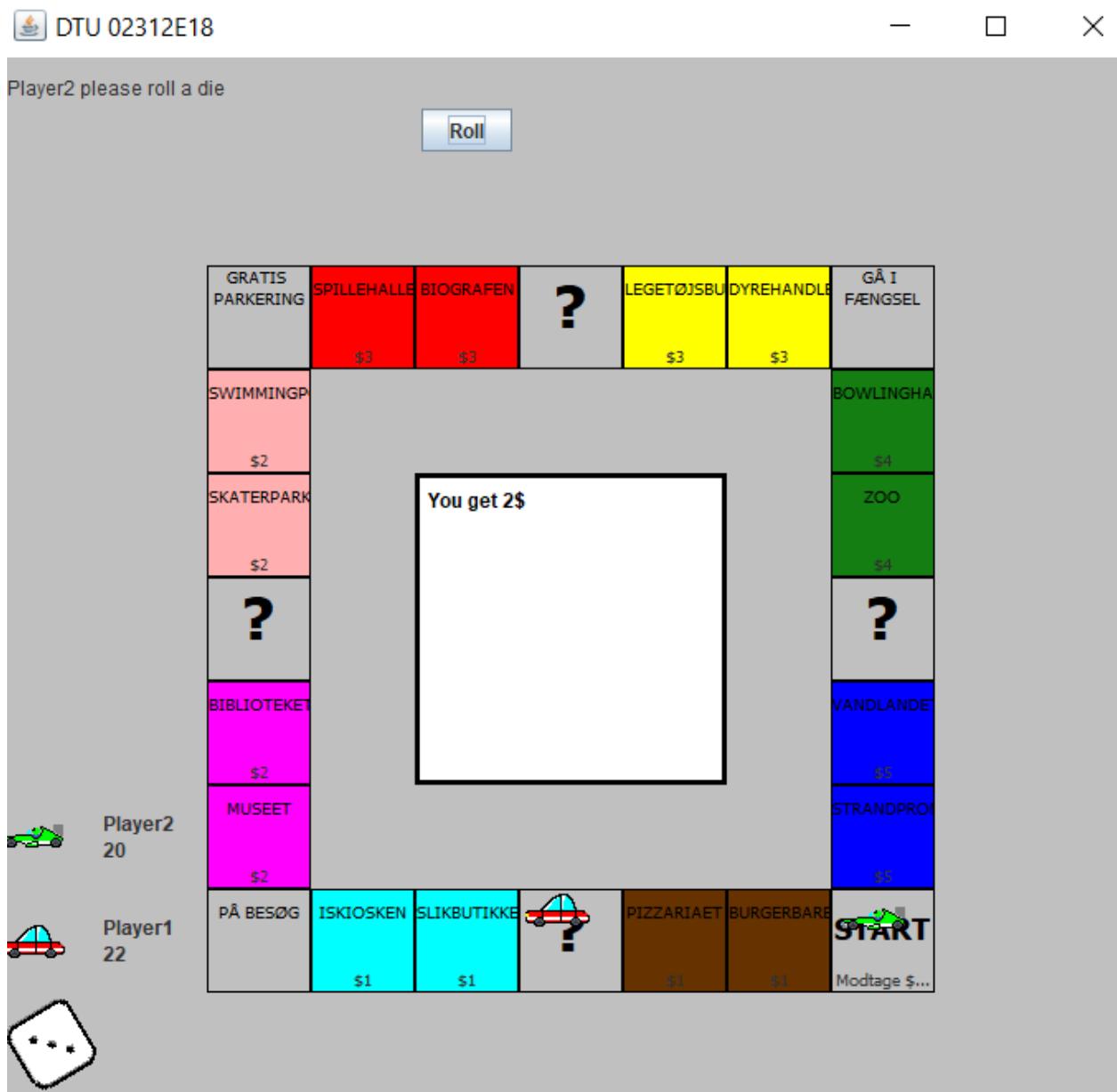
Listing 5: Test kode for test case 0002

Der kan således ses nedenstående på figur 11 størrelsen af code coverage.



Figur 11: Code coverage for movePlayer()-metoden

Resultatet af testen kan således ses på figur 12.



Figur 12: Resultat af test 0002

3.3.3 Test af ID: 0003

Der skal testes at der kun slås med en enkelt terning og at der ikke er mulighed for at række baglæns. Dette er gjort ved at undersøge om hvorvidt terningens kast interval ligger i [1, 6].

```

1 package Domain.GameElements.Entities;
2 import org.junit.jupiter.api.Test;
3 import static org.junit.jupiter.api.Assertions.*;
4 class DieTest {
5
6     @Test
7     void roll() {
8         Die die = new Die();
9         int lessThanOne = 0, correctValue = 0, moreThanSix = 0;
10        // Rolls die 100000 times and tracks the roll value
11        for(int i = 0; i < 100000; i++){
12            int value = die.Roll();
13            if(value < 1){
14                lessThanOne++;
15            } else if(value > 0 && value <= 6){
16                correctValue++;
}

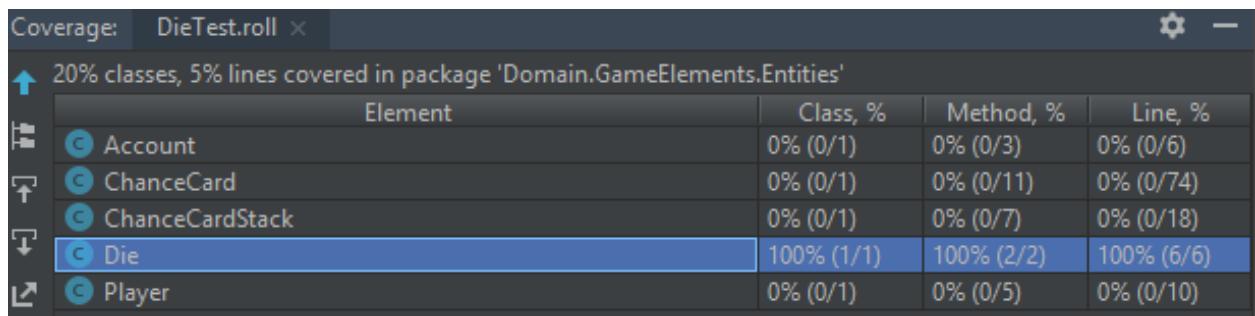
```

```

17         } else if(value > 6){
18             moreThanSix++;
19         }
20     }
21     //Examines if any die values are below one or above six
22     assertEquals(0, lessThanOne);
23     assertEquals(0, moreThanSix);
24     assertEquals(100000, correctValue);
25 }
26 }
```

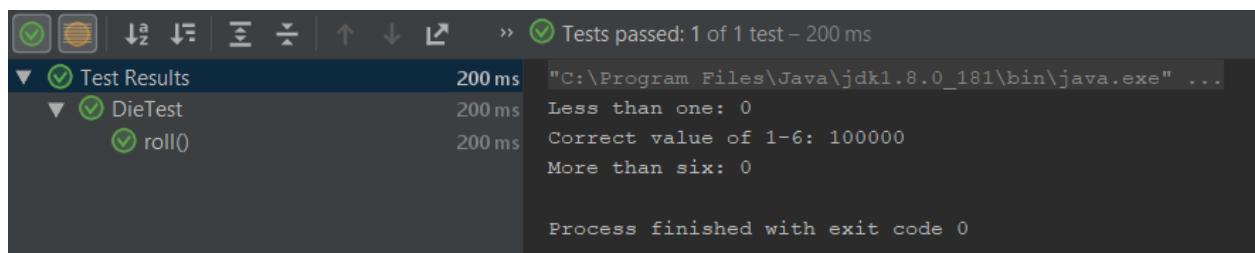
Listing 6: Test kode for test case 0003

Denne test rammer kun muligheden for at række baglæns indirekte, da det kun er terningens værdier der undersøges. Graden af code coverage kan ses på figur 13



Figur 13: Code coverage for roll()-metoden

Resultatet for testen kan ses her forneden på figur 14.



Figur 14: Resultatet af test 0003

3.3.4 Test af ID: 0004

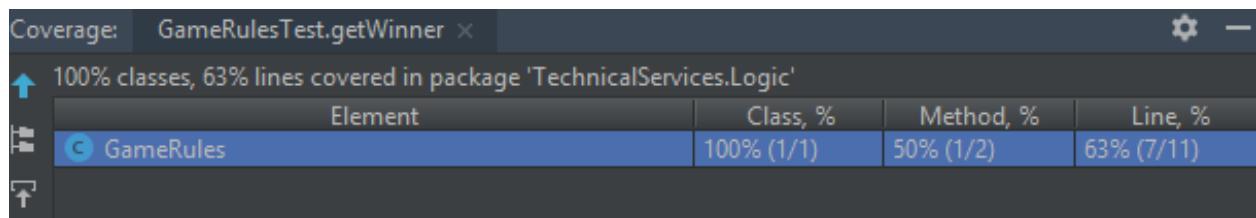
Der skal testes om hvorvidt der kan findes en vinder af spillet når dette slutter af den ene eller den anden årsag.

```

1 package TechnicalServices.Logic;
2 import Domain.GameElements.Board;
3 import Domain.GameElements.Entities.Player;
4 import org.junit.jupiter.api.Test;
5 import static org.junit.jupiter.api.Assertions.*;
6
7 class GameRulesTest {
8     @Test
9     void getWinner() throws Exception {
10         Board b = new Board();
11         b.initPlayers(2);
12         GameRules GR = new GameRules();
13         Player p1 = b.getPlayers()[0];
14         Player p2 = b.getPlayers()[1];
15         //Adds a lot of money to player1
16         p1.getAccount().changeScore(100);
17         //Test for winner = player1
18         assertEquals(p1, GR.getWinner(b));
19     }
20 }
```

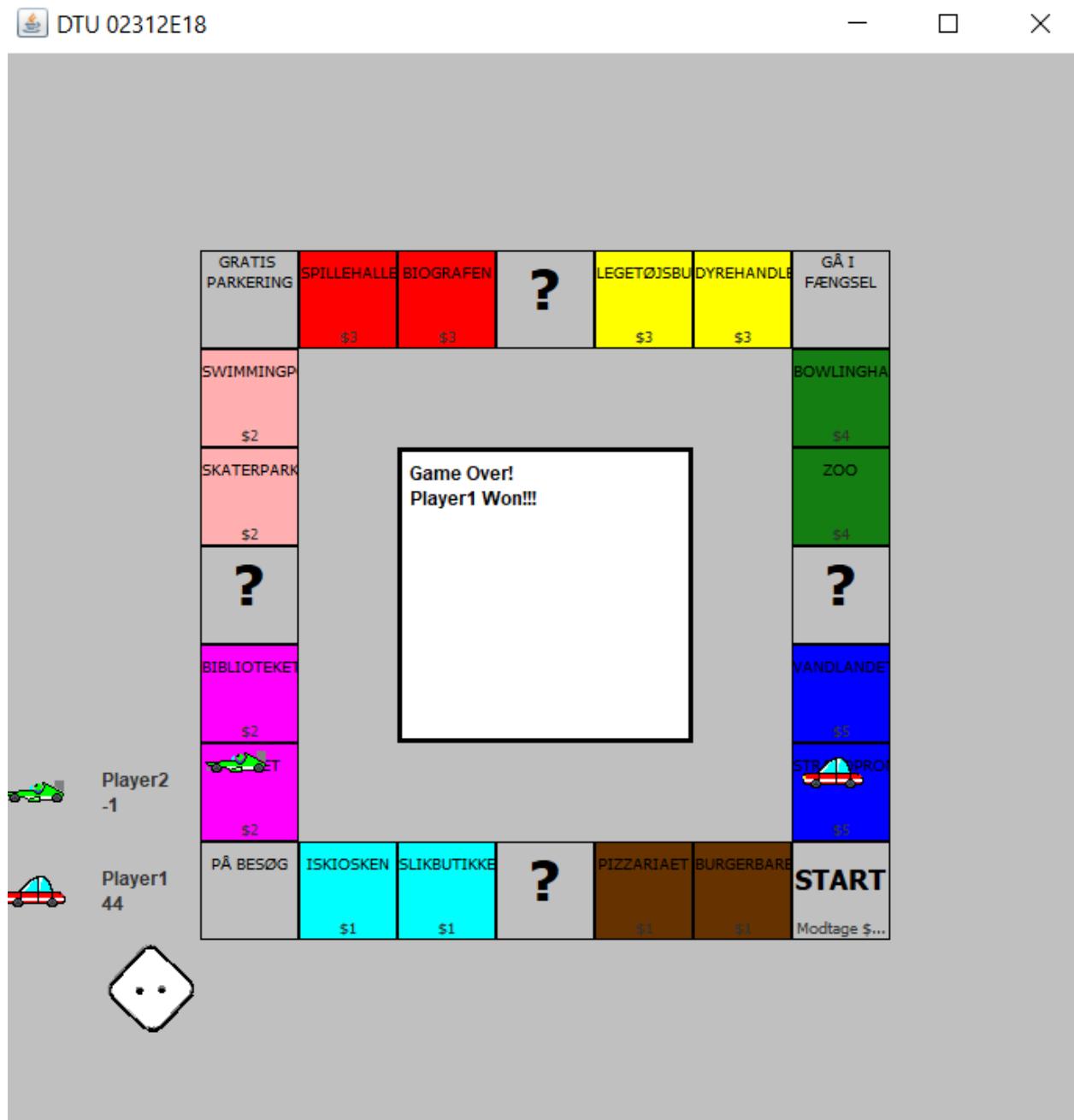
Listing 7: Test kode for test case 0004

Testen undersøger hvilken player der har den største kapital. Hvor meget af koden der går igennem kan ses forneden på figur 15 der viser code coverage for testen.



Figur 15: Code coverage for getWinner()-metoden

Resultatet af testen kan ligeledes ses forneden på figur 16.



Figur 16: Resultatet af test 0004

3.4 Brugertests

Vi har valgt at føre brugertests af vores program, for at få input fra udefrakommende der ikke har erfaring med programmering, for at få en forståelse for hvordan en normal person opfatter vores spil. Brugertestens fremgangsmåde er blevet defineret på forhånd for at sikre at forholdne for alle vores testpersoner har været så tæt på hinanden som muligt. Denne beskrivelse af brugertesten findes i bilag 5.3.

3.4.1 Resultat af brugertest:

Brugertest er lavet på Cecilia lavet af Fredrik:

Hun blev introduceret kort til spillet, hvor jeg forklarede hende hvad spillet går ud på og bad hende tænke højt mens hun prøvede spillet.

Vores bat og jar fil var ikke klar, så den er lavet, hvor spillet blev åbnet gennem intellij.

Cecilia læser teksten der står øverst i billedet, hvor hun skal skrive hvor mange spillere der skal være med. Hun vælger to spillere, som er det mindste antal spillere. Da hun tester spillet selv, spiller hun for begge spillere. Efter som hun får startet spillet for alvor, begynder at trykke rundt på de forskellige felter og figurerne. Da intet sker, trykker hun på roll knappen i toppen. Efter bilen har rykket sig, lægger hun ikke mærke til teksten der står i midten af skærmen, da den forsvinder når hun rykker mussen hen på andre felter. Det gør hun ikke lægger mærke til om nogen ejer feltet eller om det er frit.

Hun begynder at lægge mærke til teksten nogle ture senere. Hun er dog lidt i tvivl omkring teksten, da der står "Cough up 'rent' \$ please", hvor rent er prisen for grunden. Hun forstår ikke helt hvorfor scoreren ændrer sig, når hun lander på et felt. Da hun synes det er uklart hvis felt det er. Hun bliver også i tvivl om hvornår spillet slutter, da hun havde et tilfælde, hvor spiller1 kom ned på 0\$. Få ture senere lykkes det hende at afslutte spillet.

Hvad synes Cecilia om spillet?

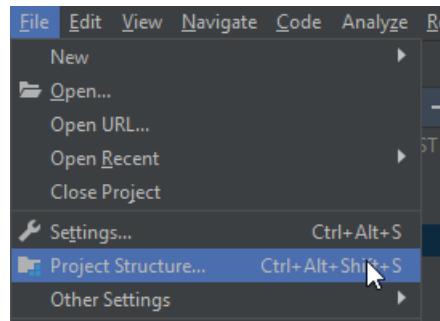
Hun synes spillet var godt lavet og det fungerer. Men spillernes score er lidt uklar muligvis pga. teksten der forekommer, når man lander på de enkelte felter. Teksten har også en tildens til at forsvinde let, hvis man rykker mussen væk fra roll knappen.

3.5 Import af JUnit fra Maven

For at kunne udføre Junit tests bliver vi nødt til at importere JUnit, dette kan gøres nemt ved brug af Maven. Udførelsen af dette kan gøres i løbet af et par trin:

Step 1

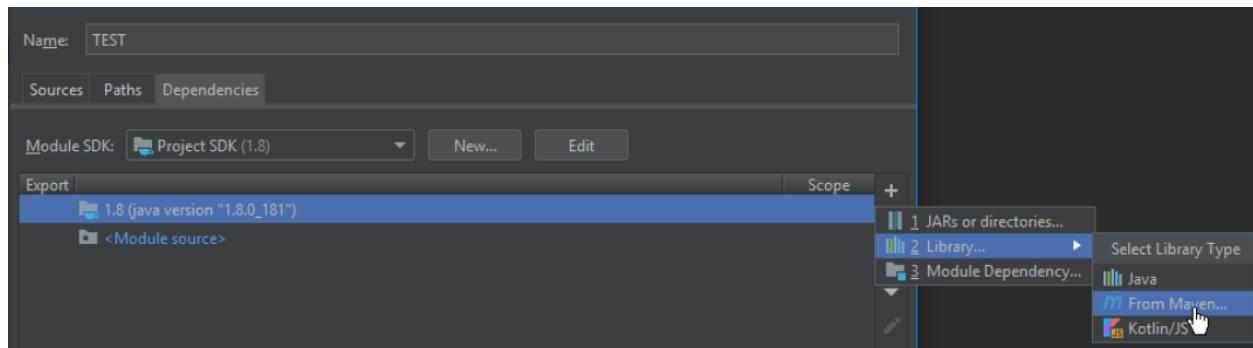
For at kunne benytte Maven til at importere et JUnit bibliotek starter man med at følge stien *File* → *Project Structure*



Figur 17: Tilgåelse af projekt struktur

Step 2

Hertil åbnes et nyt vindue og der følges stien + → *Library* → *From Maven*.

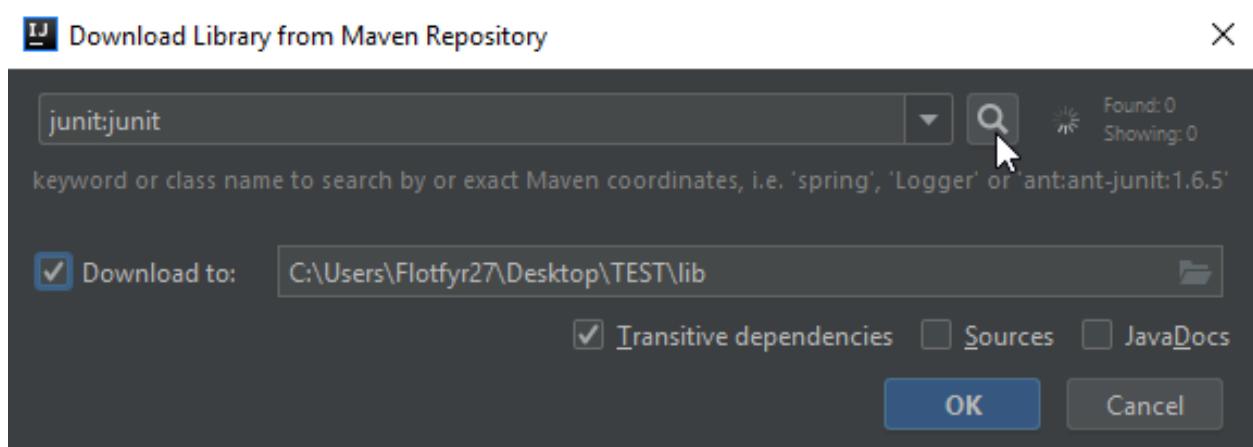


Figur 18: Tilføjelse af dependency fra Maven

Step 3

Således vil der forekomme et søgefelt, hvori dette vi søger på junit:junit.

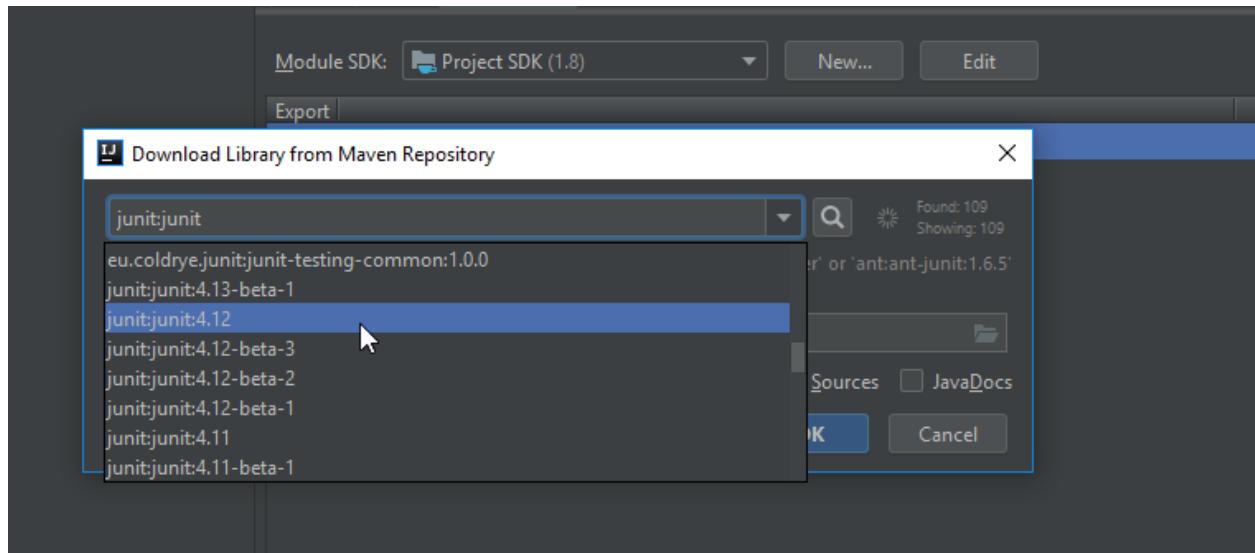
Der skal også sikres at boksen *Download to:* er udfyldt.



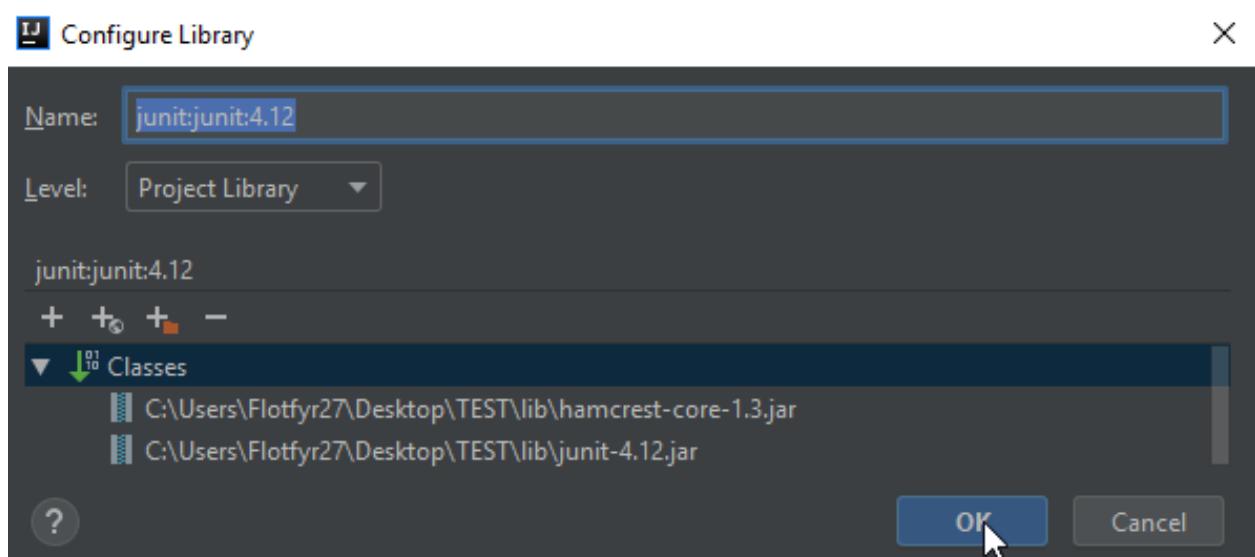
Figur 19: Fremskaffelse af søgeresultater

Step 4

Når søgningen afsluttes vælges den nyeste non-beta version af JUnit, der trykkes OK.



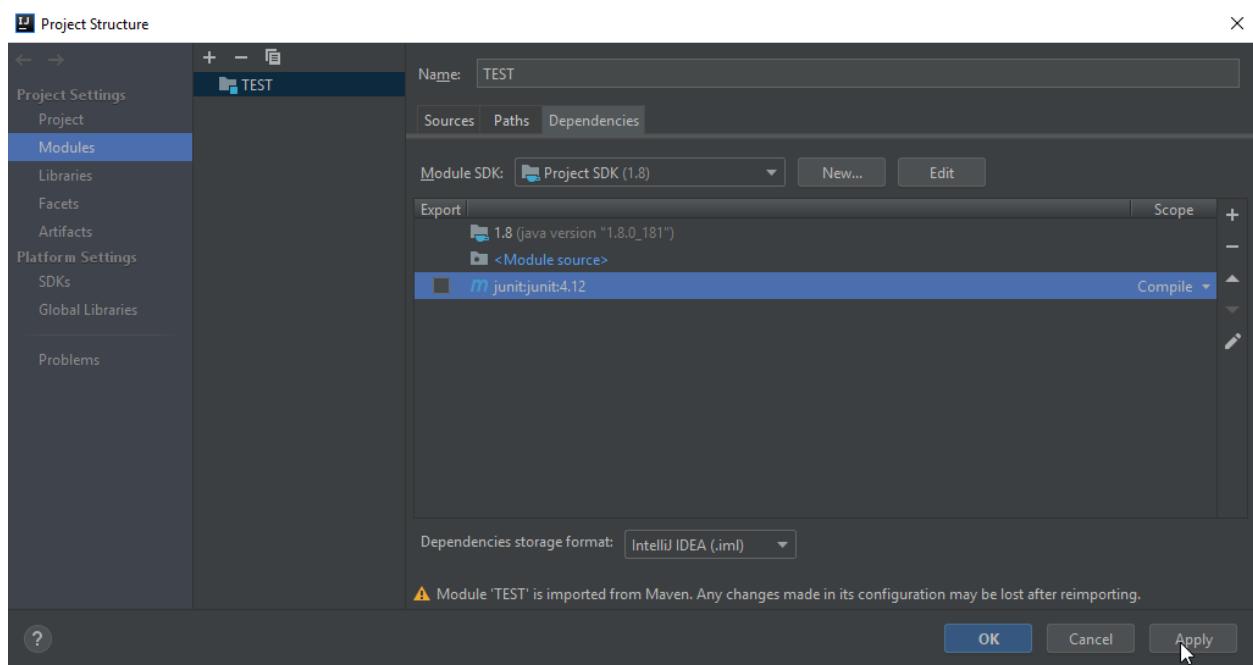
Figur 20: Den nyeste version af junit vælges



Figur 21: Nyeste version er valgt og der trykkes OK

Step 5

Nu skal tryk proceduren *Apply* → *OK* følges og JUnit er tilføjet til dit bibliotek.



Figur 22: JUnit er tilføjet til bibliotektet

3.6 Platforme og version

Til udviklingen af vores spil har vi brugt følgende platforme:

- IntelliJ - IntelliJ IDEA 2018.2.5 (Ultimate Edition), Build #IU-182.4892.20
- JRE: 1.8.0_152-release-1248-b19 amd64
- Maven - version 3.3.9
- Windows 10 x64

4 Documentation

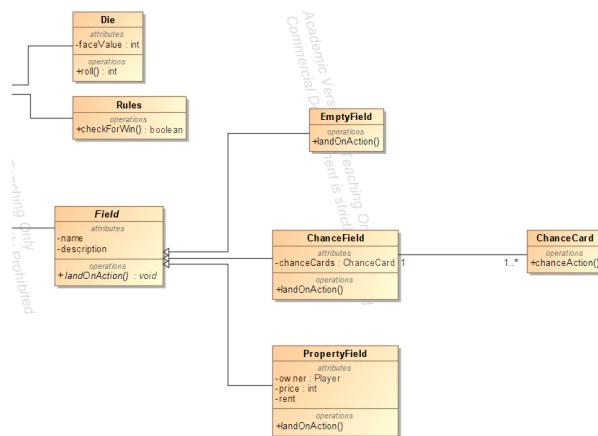
4.1 Arv

Når man omtaler arv i forhold til programmering, så er der tale om en relation. Arv kan også omtales som generalisering (når en klasse nedarver fra en anden), eller specialisering (når en klasse bliver nedarvet fra). Det viser og handler om en hierarkisk struktur, som er opdelt i superklasser og subklasser. Da det er en subklasse der arver fra superklasse, så er superklassen mere generel end dens subklasser.

Det er muligt med nedarvning at lave en ny klasse (subklasse) ud fra en allerede eksisterende klasse. Subklassen vil derfor arve superklassens egenskaber, herunder attributter og associeringer. Det er dog muligt at erstatte nogle af egenskaberne eller supplere med nye.

Ved at benytte sig af generalisering og nedarvning, da skaber det både bedre overblik, samt det fremmer forståelsen. Derudover så skaber det også en meget mere overskuelig kode, og reducere mængden af kode der skal skrives. Det er i vores klassediagram illustreret med pile, og ved at kigge på pilenes retning kan det også ses hvilke klasser der er hhv. superklasser og subklasser.

Herunder kan det blandt andet ses hvordan klasserne EmptyField, ChanceField og PropertyField nedarver fra klassen Field. Pilens retning symboliserer hvilken klasse der arver fra hvilken. Pilen går fra den klasse der arver, og til den som der arves fra. Klassen som der arves fra (i dette tilfælde Field) har ikke kendskab til klasserne EmptyField, ChanceField og PropertyField, men disse tre klasser er godt klar over deres relation til klassen Field.



Figur 23: Udsnit af Klassediagram

4.2 Abstrakte klasser

En abstrakt klasse er en klasse der defineres som værende abstrakt og indeholder typisk mindst én abstrakt metode. Hvis man har en super-klasse og en gruppe sub-klasser hvoraf sub-klasserne alle arver en metode der skal gøre det samme, men denne metode benytter en anderledes tilgang alt efter hvilken sub-klasse den kaldes fra. Hertil ville det være ideelt at definere klassen og den omtalte metode som værende abstrakt.

Der kan ikke oprettes en instans af en abstrakt klasse. I dette projekt har vi benyttet os af en abstrakt klasse der benyttes som eksempel her forneden:

```
1 package Domain.GameElements.Fields;
2 import Domain.GameElements.Entities.Player;
3 import java.awt.*;
4
5 public abstract class Field {
6     private String name, subtext;
7     private Color bgColour;
8
9     public Field(String name, String subtext, Color bgColour){
10         this.name = name;
11         this.subtext = subtext;
12         this.bgColour = bgColour;
13     }
14     public String getName(){
15         return name;
16     }
17     public String getSubtext(){
18         return subtext;
19     }
20     public Color getBgColour(){
21         return bgColour;
22     }
23     public abstract String landOnAction(Player current, Player[] players, Field[] fields);
24 }
```

Listing 8: Eksempel på abstrakt klasse

4.3 Polymorfi

I dette projekt har vi oprettet en field klasse, som i vores tilfælde fungerer som en superklasse til de mere specifikke subklasser, herunder: chanceField, emptyField, og propertyField. For at disse subklasser kan lave en handling når felterne på vores spil bliver landet på, skal de have en landOnField metode som de får fra superklassen field via nedarvning.

Det er dog vigtigt at denne metode er forskellig alt efter hvilken subklasse der tilhører det felt vores spiller er landet på. Derfor bliver vi nødt til at ændre vores landOnField metode i hver af vores subklasser, så de afspejler vores ønskede action, som derved bliver forskellig fra vores field superklasse. Dette kaldes polymorfi.

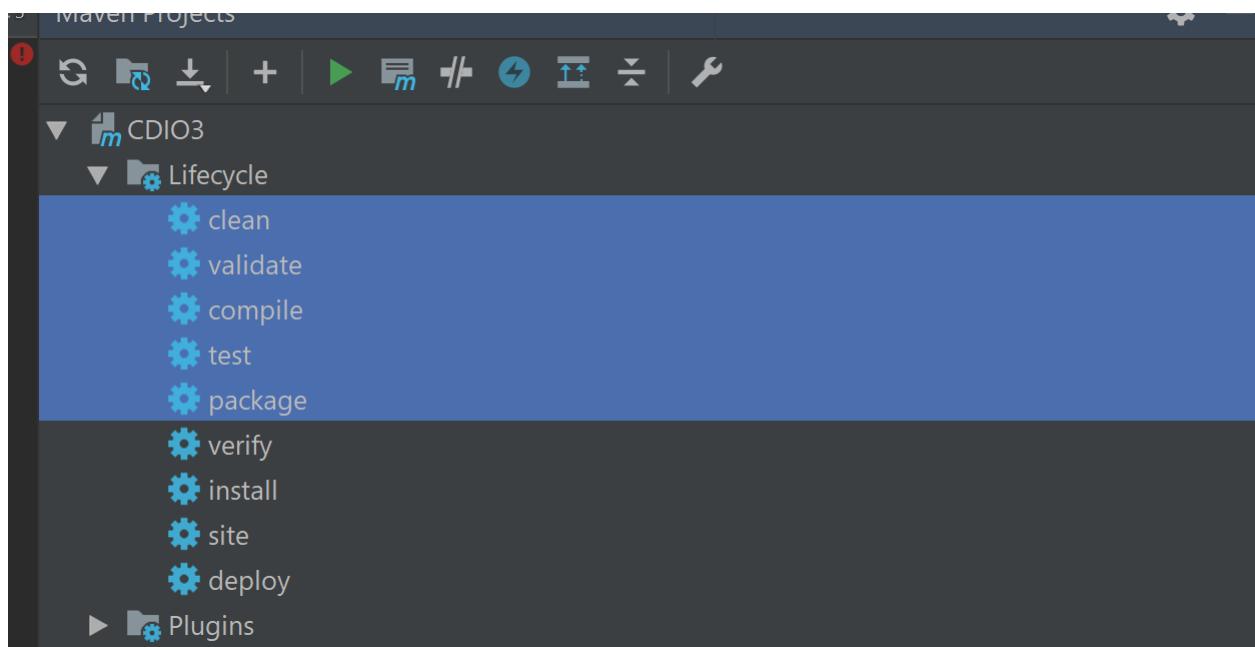
4.4 Generering af Bat og Jar fil

Vores bat og jar fil er genereret gennem maven. Så vi har tilføjet:

```
<build>
  <plugins>
    <plugin>
      <artifactId>maven-assembly-plugin</artifactId>
      <configuration>
        <archive>
          <manifest>
            <mainClass>Domain.Controller.Controller</mainClass>
          </manifest>
        </archive>
        <descriptorRefs>
          <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
        <finalName>MonopolyJunior</finalName>
        <appendAssemblyId>false</appendAssemblyId>
      </configuration>
      <executions>
        <execution>
          <id>make-assembly</id>
          <phase>package</phase>
          <goals>
            <goal>single</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

Figur 24: Code to build Jar file

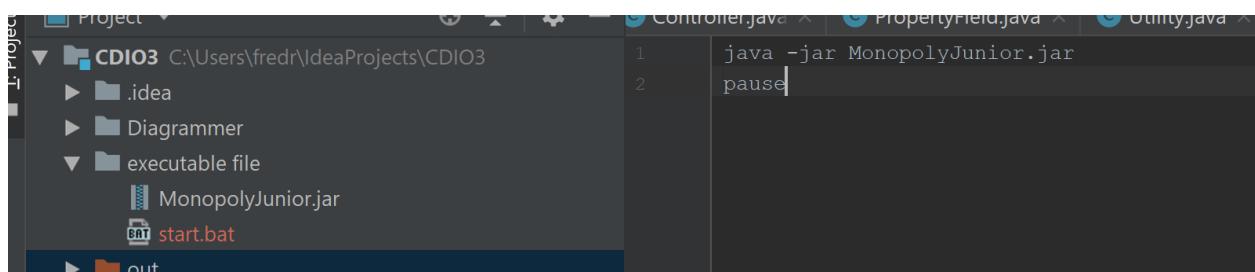
På den måde er det nu muligt at lave en jar fil gennem maven. Jeg åbner maven ude i venstre side og markerer følgende:



Figur 25: Creating the Jar file

Under lifecycle markeres alle filerne til og med package, hvorefter der trykkes run. Herefter bliver der lavet en Jar fil.

Efter etter laver man en fil og siden den ligger i samme mappe, behøver man ikke at lave en reference til hvor den er gemt, som man normalt skal. Der skal kun stå følgende i filen:

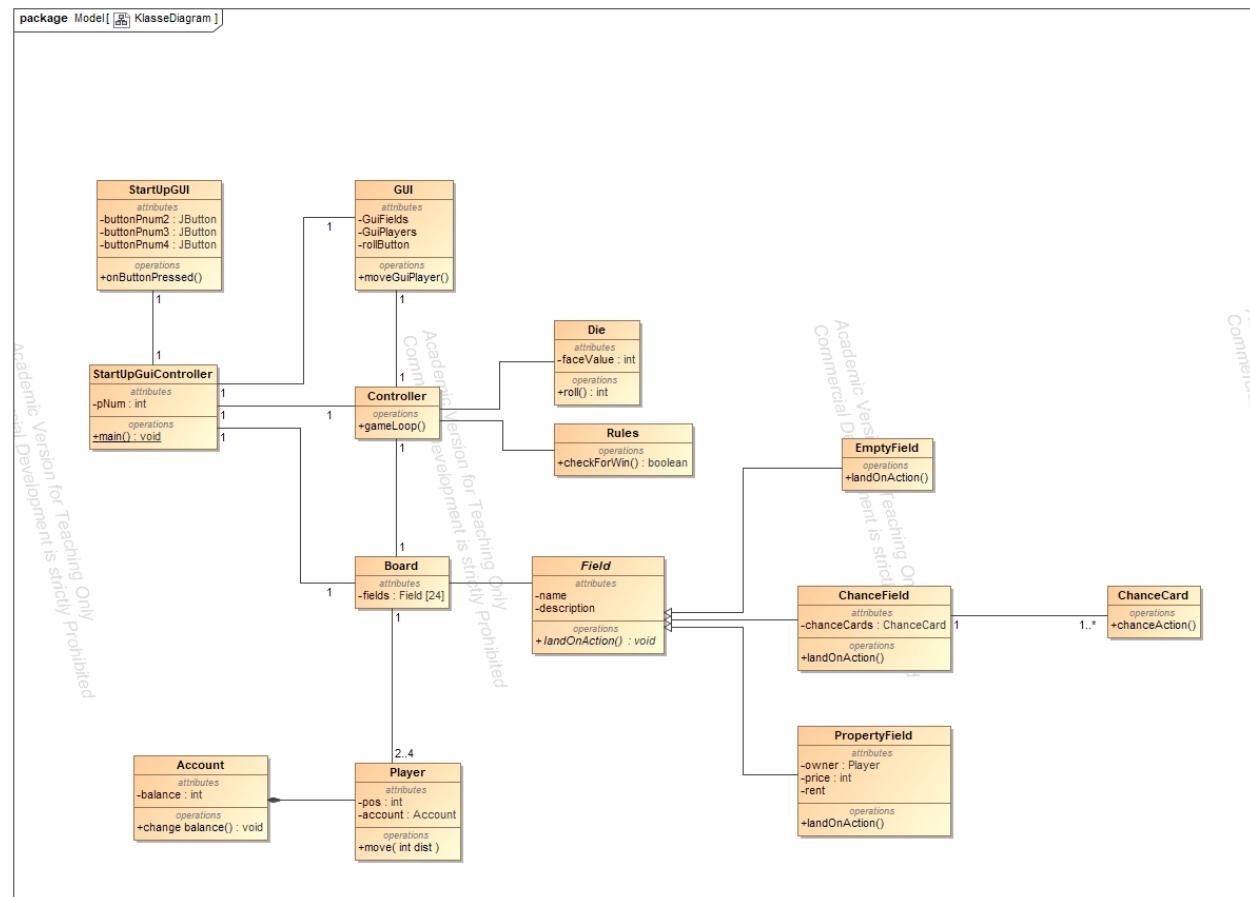


Figur 26: Bat file

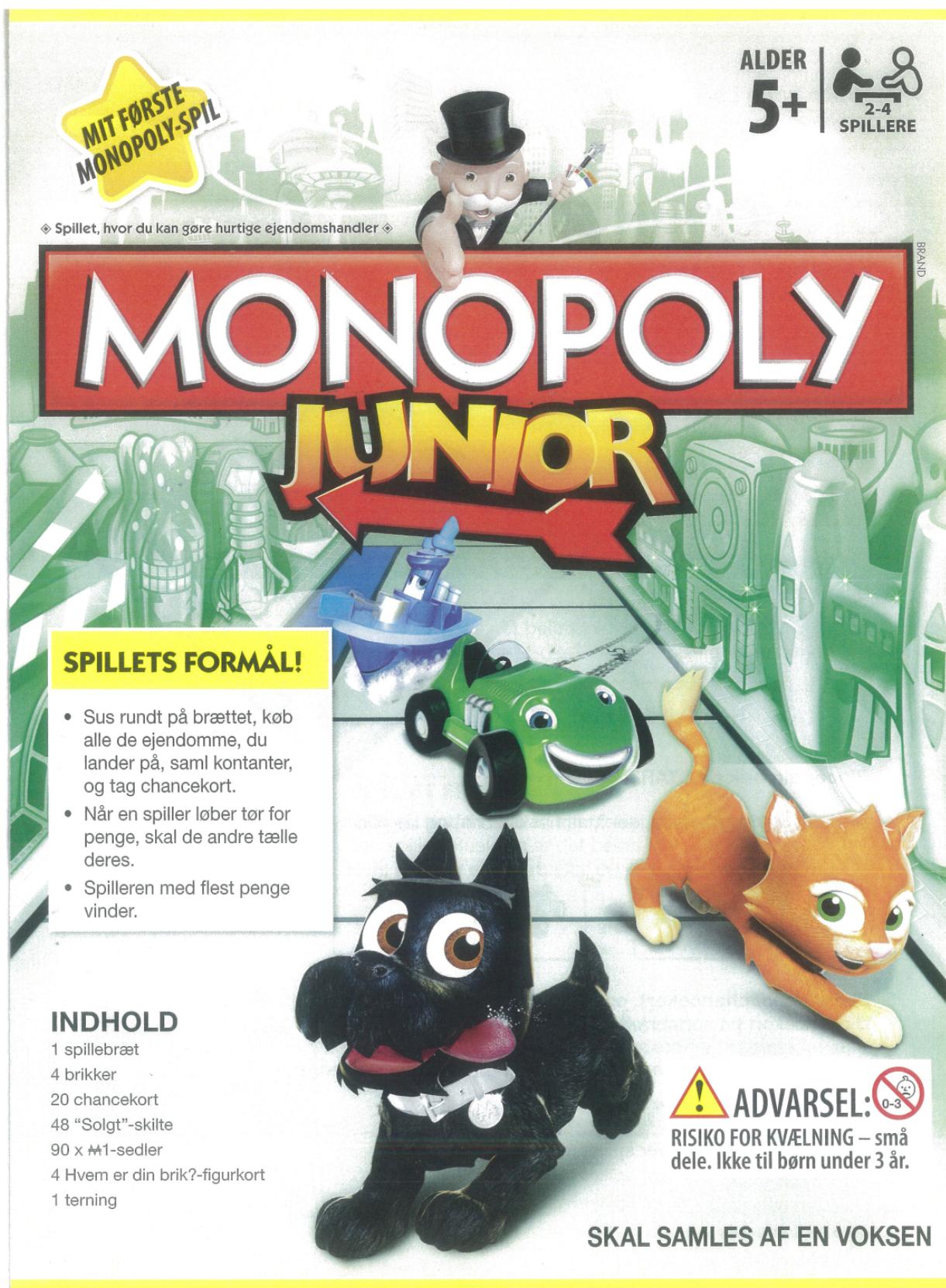
Herefter kan man dobbeltklikke eller højreklikke og tryk run på filen og spillet vil starte.

5 Bilag

5.1 Klassediagramm



5.2 Monopoly junior spilleregler



Figur 27: Monopoly junior regler side 1

Første gang I spiller

Tryk alle 48 "Solgt"-skilte ud af paparket. Smid overskydende stumper ud.

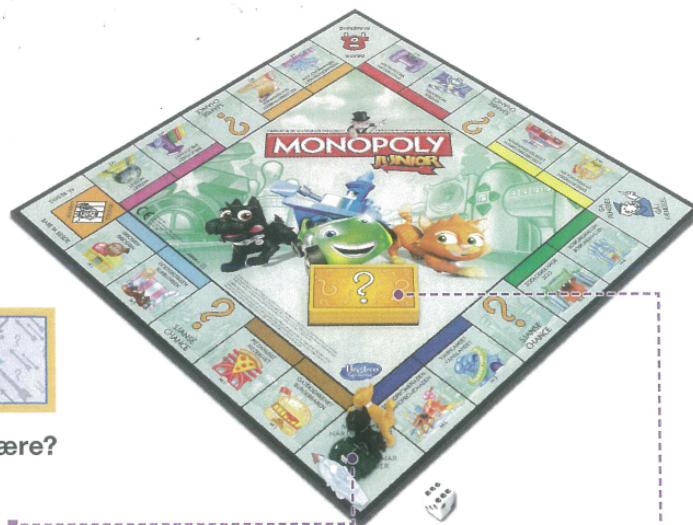
OPSÆTNING!

1. Åbn spillebrættet, og anbring det mellem spillerne.

2. Tag chancekortene, og fjern de 4 figurkort.

De skal ikke bruges i spillet. Bare læs dem, og vælg!

De 4 figurkort:



Hvem vil du være?

3. Placér din brik på START!

4. Find de 12 "Solgt"-skilte, der matcher din brik, og læg dem foran dig.



5. Bland de 20 chancekort, og læg dem med billedsiden nedad på feltet til chancekort på spillebrættet.

6. Vælg en spiller til at være bankør. Bankøren passer på pengene.

Så skal der deles penge ud:

For 2 spillere: Giv hver spiller ₪20

For 3 spillere: Giv hver spiller ₪18

For 4 spillere: Giv hver spiller ₪16



Figur 28: Monopoly junior regler side 2

SPIL!

Sådan vinder du

Vær den spiller, der har flest penge, når en anden spiller går fallit (ikke har råd til at betale husleje, købe en ejendom eller betale en afgift fra et chancekort).

Sådan spiller du

- 1. Den yngste spiller starter!** Spillet fortsætter mod venstre.
- 2. Kast terningen, og flyt din brik det antal felter, som øjnene viser, med uret rundt på brættet fra START.**

 - Ryk **altid** frem, aldrig tilbage.
 - **Hver gang** du passerer eller lander på START, modtager du **M2!**

- 3. Hvor landede du?** Læs alt om de forskellige felter, før du starter.

Hvis du lander på:



ET LEDIGT FELT

Hvis ingen ejer det, skal du købe det!

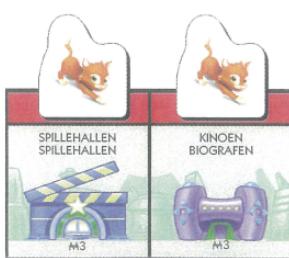
- Betal banken det beløb, der står på feltet.
- Placér et af dine "Solgt"-skilte på det farvede bånd øverst på feltet, så alle kan se, at du ejer det.



ET EJET FELT

Hvis en anden spiller ejer feltet, skal du betale husleje til den spiller. Huslejen er det beløb, der står på feltet.

Hvis du selv ejer det, skal du ikke gøre noget.



ET PAR = DOBBELT HUSLEJE!

Hvis en spiller ejer begge ejendomme i samme farve, er huslejen det dobbelte af det beløb, der står på feltet!

4. Vend for at se oplysninger om de øvrige felter på brættet og om, hvordan du vinder!



Figur 29: Monopoly junior regler side 3

BRAÆTTETS FELTER

START



Modtag $\text{M}2$ fra banken, hver gang du passerer eller lander på START.

CHANCE



- Tag det øverste chancekort, læs det højt, og følg instruktionerne.
- Læg brugte kort tilbage nederst i bunken.



Gå lige i fængsel! Du passerer ikke START. Du modtager ikke $\text{M}2$. I starten af din næste tur skal du betale $\text{M}1$ eller bruge "Du løslades uden omkostninger"-kortet, hvis du har det. Kast derefter terningen, og ryk som normalt. Du kan godt modtage husleje, mens du er i fængsel.



Hvis du lander her, er du bare på besøg.



Bu behøver ikke gøre noget, bare snup dig en pause.

5. Det var det! Så er det næste spillers tur.

VIND!

- Hvis du ikke har penge nok til at betale husleje, købe en ejendom, som du lander på, eller betale afgiften fra et chancekort, er du gået fallit! Og så er spillet slut!
- De andre spillere tæller deres penge, og den, der har flest, har VUNDET!
- Uafgjort? Tæl, hvor meget dine ejendomme er værd, og læg det til dine penge!

AVANCERET!

Når du har lært de grundlæggende regler, kan du prøve at spille på følgende måde og se, hvem der vinder.

- Hvis du ikke har penge nok til at betale husleje eller en afgift fra et chancekort, skal du betale gælden med dine ejendomme.
- Hvis du skylder en anden spiller penge, får den spiller dine ejendomme. Hvis du skylder banken penge, bliver dine ejendomme sat til salg igen.
- Hvis du stadig ikke kan betale, er du gået fallit, og spillet slutter. Alle tæller deres penge for at se, hvem der har vundet!

HASBRO GAMING- og MONOPOLY-navnet og -logoet, spillebrættets særegne design, de 4 hjørnefelter, MR. MONOPOLY-navnet og -figuren samt de særegne elementer på spillebrættet og spillebrikkerne er varemærker, der tilhører Hasbro for dets ejendomshandlende spil og spiludstyr.

© 1935, 2013 Hasbro. Alle rettigheder forbeholdt.

Produceret af: Hasbro SA, Rue Emile-Boechat 31, 2800 Delémont CH.

Repræsenteret af: Hasbro Europe, 4 The Square, Stockley Park, Uxbridge, Middlesex, UB11 1ET, UK.

Hasbro Nordic Consumer Services:

Hasbro Denmark, Ejby Industrivej 40, 2600 Glostrup, Denmark. 43 27 01 00. hasbrodk@hasbro.dk

Opbevar disse oplysninger som reference.

Farver og indhold kan variere fra det illustrerede.

www.hasbro.dk

128A69841080



1113A6984108-108

Figur 30: Monopoly junior regler side 4

5.3 Brugertest

Dette er beskrivelsen af fremgangsmetoden hvorved vi har udført vores brugertest:

- Introducer brugerne kort til spillet, hovedsageligt ved at forklare at spillet er et matador junior spil. Forklar at det er vigtigt at de husker at tænke højt når de spiller spillet. evt. snakke med hinanden om deres oplevelse.
- Lad brugeren selv starte og begynde spillet.
- Sørg for ikke at have interaktion med brugerne underspilperioden.
- Observatøren sørger for at optage/ føre diktakt af spilforløbet.
- Når spillet er overstået, få feedback fra spillerne. Det er vigtigt at man ikke forklarer hvorfor spillet fungerer som det gør, men tag imod brugerens ufarvede kritik.
- Feedback beskrives i sektionen om tests under dokumentation.

Når testen er gennemført må observatøren gerne markere kritik efter emnerne: Brugervenlighed, Forståelse for spillet, grafisk udførelsel, og andet.

5.4 Test Krav

Vi vil teste følgende krav:

- 1. Man skal kunne vælge mellem 2, 3 og 4 spillere
- 2. Der bliver kun slæt med en terning.
- 3. Ens figur forsætter fra fletet, som man sluttede sidste tur på
- 4. Man går i ring omkring brætspillet
- 5. Alle spiller starter med det samme beløb
- 6. Spillet vindes, når en anden spiller løber tør for penge.
- 7. Vinderen er den spiller, som har flest penge/egendomme, når spillet slutter
- 8. Man kan kun rykke fremad
- 9. Man rykker sig det antal felter der passer til værdien af terningen
- 10. Man skal købe et felt, hvis det er ledigt.
- 11. Der betales husleje, hvis en anden ejer fletet.
- 12. Man modtager penge, hver gang man passerer start
- 13. Test chancekortene
 - 13a. Ryk frem til START og modtag 2\$
 - 13b. Ryk op til 5 felter frem.
 - 13c. Ryk frem til et orange felt, hvis det er ledigt får man det gratis ellers skal der betales husleje.
 - 13d. Ryk 1 felt frem eller tag et chancekort mere
 - 13e. Betal 2\$ til banken
 - 13f. Ryk frem til HOT.PINK eller grønt felt og modtag det gratis, ellers skal der betale husleje
 - 13g. Ryk frem til lyseblåt felt, hvis det er ledigt får du det gratis ellers skal der betales husleje
 - 13h. Tillykke du har fødselsdag, alle giver dig 1M
 - 13i. Ryk frem til pink eller mørkeblåt felt og modtag det gratis, ellers skal der betale husleje
 - 13j. Modtag 2\$
 - 13k. Ryk frem til rødt felt, hvis det er ledigt får du det gratis ellers skal der betales husleje
- 14. Man skal kunne se beskeden på ens chancekort
- 15. Man skal kunne se hvad man slår med terningerne.
- 16. Der kommer en besked om hvem og når en spiller har vundet.
- 17. Der trækkes et chancekort, når man lander på et chancefelt

5.5 Test cases

| |
|---|
| Test ID: 0001 |
| TestNavn: |
| Antallet af spillere samt startkapitalet. |
| Test af Krav: |
| 1 og 5 |
| Beskrivelse: |
| Vi tester om det er muligt, man kan vælge mellem 2, 3 og 4 spillere før spillet starter, og om alle spillere starter med det samme antal penge. |
| Forventning: |
| Vi forventer at man kan vælge mellem 2-4 spillere og at alle starter med det samme startkapital. |
| Resultat: |
| XX. |

Screenshot:

| |
|--|
| Test ID: 0002 |
| TestNavn: |
| Test af bevægelse på spillebrættet |
| Test af Krav: |
| 3, 4, og 12 |
| Beskrivelse: |
| Vi tester om man rykker det antal man slår. Samt om man starter på det felt den sidste tur sluttede på. Når man kommer til det sidste felt og passerer start forsætter man fra start af og man modtager 2\$. |
| Forventning: |
| Vi forventer, at man forsætter hvor sidste tur sluttede og når man passere start, så forsætter man fra start og modtager 2\$ |
| Resultat: |
| XX. |

Screenshot:

| |
|---|
| Test ID: 0003 |
| TestNavn: |
| Test af terning |
| Test af Krav: |
| 2, 8, 9 og 15 |
| Beskrivelse: |
| Vi tester der bliver slået med en terning samt om man rykker det antal felter som bliver slået. Det skal kun kan være muligt at rykke fremad. |
| Forventning: |
| Der vises kun en terning værdi og den værdi rykker spilleren sig med fremad. |
| Resultat: |
| XX. |

Screenshot:

| |
|--|
| Test ID: 0004 |
| TestNavn: |
| Test om spillet kan vindes |
| Test af Krav: |
| 6, 7 og 16 |
| Beskrivelse: |
| Vi tester om spillet vindes, når den en spiller løber tør for penge og om der kommer en besked om hvem som har vundet. |
| Forventning: |
| Spillet vindes når den ene spiller løber tør for penge og der kommer en besked hvem der har vundet. |
| Resultat: |
| XX. |

Screenshot:

| |
|---|
| Test ID: 0005 |
| TestNavn: |
| Når man lander på et felt |
| Test af Krav: |
| 10, 11 og 17 |
| Beskrivelse: |
| Vi tester hvad der sker når man lander på et felt, om man køber grunden, betaler husleje eller om der bliver trukket et chancekort, når man lander på et chancefelt |
| Forventning: |
| Grunden bliver købt, når man lander på grunden, der bliver betalt husleje, hvis man lander på en grund der ikke er ens og man får et chancekort, når man lander på et chancefelt. |
| Resultat: |
| XX. |

Screenshot:

| |
|--|
| Test ID: 0006 |
| TestNavn: |
| Når man trækker et chancekort |
| Test af Krav: |
| 12, 13a og 14 |
| Beskrivelse: |
| Når man trækker kortet "ryk frem til start og modtag 2\$" og om man kan se teksten på kortet |
| Forventning: |
| Det forventes at man bliver rykket frem til start og modtager 2M for at passere start og beskeden på kortet vises. |
| Resultat: |
| XX. |

Screenshot:

| |
|---|
| Test ID: 0007 |
| TestNavn: |
| Når man trækker et chancekort 2 |
| Test af Krav: |
| 13b |
| Beskrivelse: |
| Når man trækker kortet "ryk op til 5 felter frem" |
| Forventning: |
| Det forventes at man kan vælge mellem 0-5 felter, som man vil rykke frem og derefter rykkes ens figur det antal frem. |
| Resultat: |
| XX. |

Screenshot:

| |
|---|
| Test ID: 0008 |
| TestNavn: |
| Når man trækker et chancekort 3 |
| Test af Krav: |
| 13c |
| Beskrivelse: |
| Når man trækker kortet "Ryk frem til et HOT.PINK felt, hvis det er ledigt får man det gratis ellers skal der betales husleje." |
| Forventning: |
| Vi forventer at man bliver rykket frem til et HOT.PINK felt, hvis det er ledigt får man feltet gratis ellers betales der husleje. |
| Resultat: |
| XX. |

Screenshot:

| |
|--|
| Test ID: 0009 |
| TestNavn: |
| Når man trækker et chancekort 4 |
| Test af Krav: |
| 13d |
| Beskrivelse: |
| Når man trækker kortet ”Ryk 1 felt frem eller tag et chancekort mere” |
| Forventning: |
| Vi forventer at man har valgmuligheden mellem at vælge et nyt chancekort eller rykke et felt frem. |
| Resultat: |
| XX. |

Screenshot:

| |
|--|
| Test ID: 0010 |
| TestNavn: |
| Når man trækker et chancekort 5 |
| Test af Krav: |
| 13e |
| Beskrivelse: |
| Når man trækker kortet ”Betal 2\$ til banken” skal spilleren, som trak kortet betale 2\$ til banken. |
| Forventning: |
| Vi forventer, at når kortet trækkes betales 2\$ tilbage til banken. |
| Resultat: |
| XX. |

Screenshot:

| |
|---|
| Test ID: 0011 |
| TestNavn: |
| Når man trækker et chancekort 6 |
| Test af Krav: |
| 13f |
| Beskrivelse: |
| Når man trækker kortet ”Ryk frem til HOT.PINK eller grønt felt og modtag det gratis, ellers skal der betale husleje” |
| Forventning: |
| Vi forventer at man bliver rykket frem til et HoT.PINK eller grønt felt, hvis det er ledigt før man fletet gratis ellers betales der husleje. |
| Resultat: |
| XX. |

Screenshot:

| |
|---|
| Test ID: 0012 |
| TestNavn: |
| Når man trækker et chancekort 7 |
| Test af Krav: |
| 13g |
| Beskrivelse: |
| Når man trækker kortet "Ryk frem til turkisblåt felt, hvis det er ledigt får du det gratis ellers skal der betales husleje" |
| Forventning: |
| Vi forventer at man bliver rykket frem til et turkisblåt felt, hvis det er ledigt får man feltet gratis ellers betales der husleje. |
| Resultat: |
| XX. |

Screenshot:

| |
|---|
| Test ID: 0013 |
| TestNavn: |
| Når man trækker et chancekort 8 |
| Test af Krav: |
| 13h |
| Beskrivelse: |
| Når man trækker kortet "Tillykke med fødselsdag, alle giver dig 2\$" vil alle modspillerne give spilleren 2\$ hver. |
| Forventning: |
| Vi forventer at spilleren, som modtager chancekortet, modtager 1M fra alle modspillerne |
| Resultat: |
| XX. |

Screenshot:

| |
|--|
| Test ID: 0014 |
| TestNavn: |
| Når man trækker et chancekort 9 |
| Test af Krav: |
| 13i |
| Beskrivelse: |
| Når man trækker kortet "Ryk frem til lilla eller mørkeblåt felt og modtag det gratis, ellers skal der betale husleje" |
| Forventning: |
| Vi forventer at man bliver rykket frem til et pink og mørkeblåt felt, hvis det er ledigt får man feltet gratis ellers betales der husleje. |
| Resultat: |
| XX. |

Screenshot:

| |
|---|
| Test ID: 0015 |
| TestNavn: Når man trækker et chancekort 10 |
| Test af Krav: 13j |
| Beskrivelse: Når man trækker chancekortet ”Modtag 2\$”, vil spilleren, som trak kortet modtage 2\$. |
| Forventning: Vi forventer at man modtager 2\$, når man trækker kortet. |
| Resultat: XX. |

Screenshot:

| |
|--|
| Test ID: 0016 |
| TestNavn: Når man trækker et chancekort 11 |
| Test af Krav: 13k |
| Beskrivelse: Når man trækker kortet ”Ryk frem til rødt felt, hvis det er ledigt får du det gratis ellers skal der betales husleje” |
| Forventning: Vi forventer at man bliver rykket frem til et rødt felt, hvis det er ledigt får man feltet gratis ellers betales der husleje. |
| Resultat: XX. |

Screenshot: