



02312 62531 62532

Indledende programmering, Udviklingsmetoder til IT-systemer og
Versionsstyring og testmetoder

CDIO 3

Gruppe 16

Mathias Løgholt Westergaard
s224293



Marcus Prehn-Chang
s224294

Mathias A. S. Lindblad
s165247



Magnus Raagaard Kjeldsen
s224282

Chayanee Christensen
s210973



Mads Villum Nielsen
s214485

25. November 2022

Resumé

Vi har fået til opgave at designe og udvikle Matador Junior på baggrund af kundens vision og krav. Som led i analysen af visionen og kravene, har vi udarbejdet relevante artefakter.

Heriblandt en use case beskrivelse, et domæne diagram og et systemsekvensdiagram.

Efterfølgende har vi lavet et design af systemet ved hjælp af et klassediagram. Artefakterne er løbende blevet opdateret i flere iterationer så det kan bruges til dokumentation af systemets arkitektur og virkemåde. Vi delte programmet op i klasser, og gav én person ansvaret for mindst én klasse. Vi programmerede sideløbende og arbejdede sammen om at få klasserne til at fungere sammen. Vi foretog brugertests, skrev J-Unit tests og har udarbejdet tre testrapporter. Dette hjalp os med at finde fejl, som vi efterfølgende har kunne eliminere. Overordnet set, opfylder programmet tilsyneladende de stilne krav.

Timeregnskab

Analyse	5 timer
Design	5 timer
Kode	50 timer
Test	5 timer
Rapport	15 timer

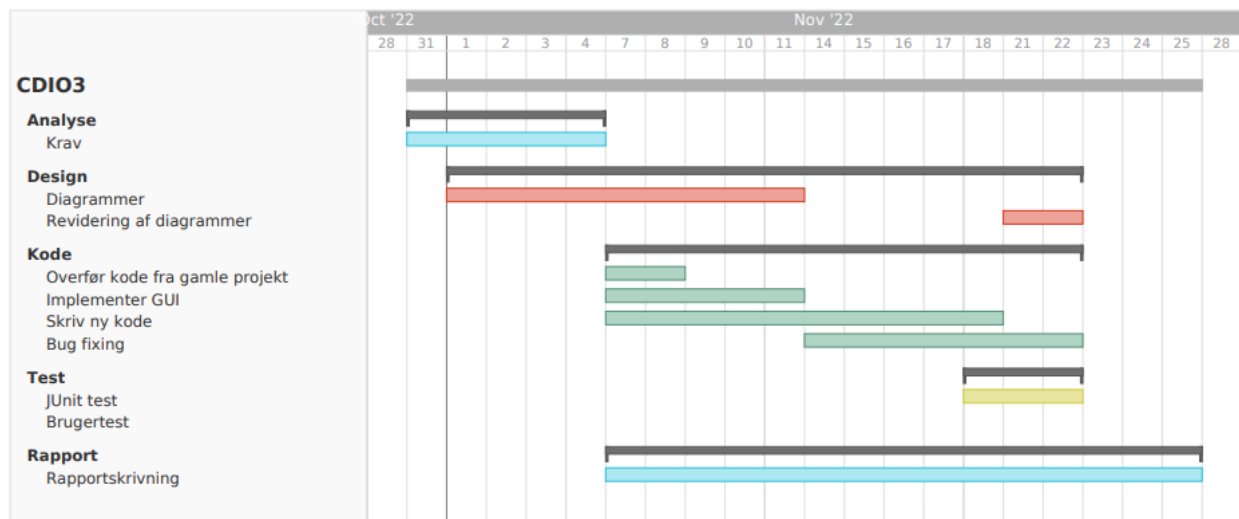
Resumé	1
Timeregnskab	1
Indledning	2
Projekt-planlægning	3
Krav	4
Analyse	6
Use case Beskrivelse	6
Use Case diagram	8
Domæne model	8
Sekvensdiagram	9
Design	10
Design Klassediagram	10
System sekvensdiagram	11
Implementering	12
Dokumentation	16
GRASP-Patterns	17
Test	17
Konklusion	33
Bilag	34

Indledning

I denne opgave er vi blevet bedt om at lave spillet Matador Junior. Spillet spilles med 2-4 spillere. Spillerne skiftes til at slå med terninger og rykker rundt på et spillebræt. Spillerne kan købe ejendomme, og hvis man lander på en ejendom, som er ejet af en anden spiller, skal man betale leje. Deriblant er der “prøvlykken”-kort, som får spilleret til at gøre en speciel handling eller tage en beslutning. Efter at én spiller er gået bankerot, vinder den spiller, som har flest penge. Spillet skal overholde nogle særlige krav. Eksempelvis, skal vi implementere et GUI. For at kunne lave programmet optimalt har vi i første omgang planlagt, hvordan projektet skal forløbe. Derefter har vi anvendt forskellige diagrammer og artefakter, såsom en use case beskrivelse og et klasse diagram, så opgaven er blevet mere overskuelig. Derefter vil vi designe systemet og implementere det. I rapporten kommer vi også ind på hvilke GRASP patterns vi har anvendt under designfasen af vores program. Selve koden til vores spil kan man læse om i implementerings afsnittet, og vi har lavet nogle test, som tester forskellige klasser og metoder i vores kode.

Projekt-planlægning

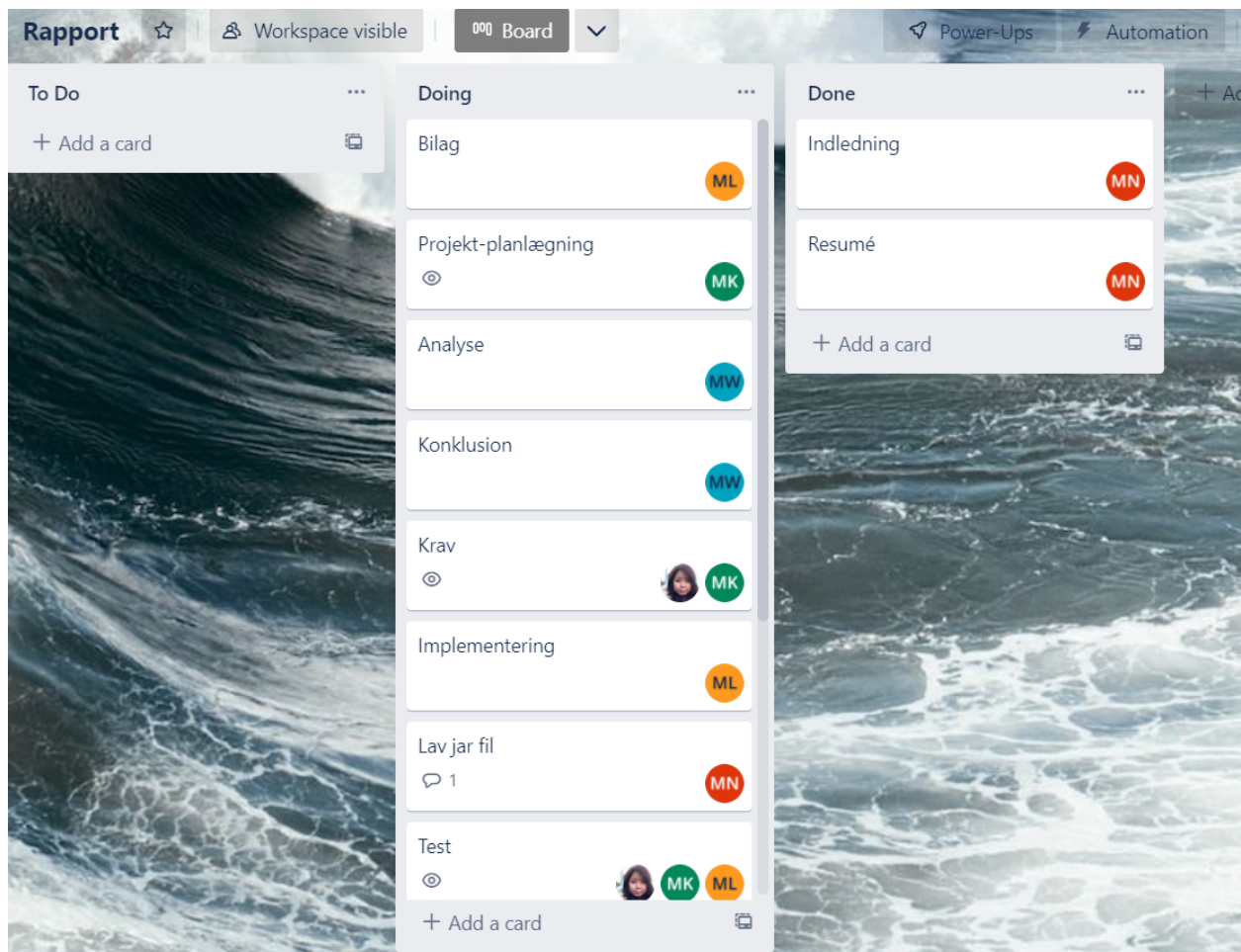
Til dette projekt lavede vi en tidsplan over hvad der skulle laves i de 4 uger vi havde til at udvikle spillet. Vi lavede tidsplanen som et Gantt diagram. Her er planen delt op i 5 dele; analyse, design, kode, test og rapportskrivning. Dette har vi gjort for at få et bedre overblik over hvor lang tid vi havde til projektet og for at dele projektet op i mindre og overskuelige bider.



Vi starter med at analyser kravene som kunden har kommet med og de regler der er i Monopoly junior. Både her og gennem hele projektet har vi været i kontakt med kunder når vi har haft

spørgsmål. Derefter begyndte vi at designe hvordan programmet skulle opsættes. I den anden uge ville vi overføre koden fra CDIO 2 og gemme det der godt kunne bruges. Efter programmet er lavet og kan spilles vil vi begynde at teste og fikse de fejl testene finder. Til sidst vil vi indføre et code freeze på store funktioner så vi ikke risikere at programmet ikke virker når det skal afleveres til kunden.

Til at holde styr på hvad der skulle skrives i rapporten og hvem der skulle skrive det, brugte vi et trello board. Her skrev vi alle afsnit op og gav en person ansvaret for at det. Den person skulle så sikre sig at det afsnit blev lavet og hvem der skulle skrive på det. Dette blev gjort så vi vidste alle afsnit blev lavet og hvis der var et afsnit ikke blev lavet kunne vi se hvem der skulle have gjort det. Vi endte dog ikke med at have problemer.



Krav

Vi har startet med at analysere kundes vision for at finde kravene til det spil vi skal udvikle. Her skriver vi så de væsentlige punkter ned. I det indgår både kundes egne krav og spillets regler.

Kundens vision

*“I skal udvikle et **Monopoly Junior** spil. Vurder hvad der er det vigtigste for at spillet kan spilles! Implementer de væsentligste elementer for at spillet kan spilles. I må **gerne** udelade regler - priorité!*

Nu har vi terninger og spillere på plads, men felterne mangler stadig en del arbejde. I dette tredje spil ønsker vi derfor at forrige del bliver udbygget med forskellige typer af felter, samt en decideret spilleplade.

Spillerne skal altså kunne lande på et felt og så fortsætte derfra på næste slag. Man går i ring på brættet.

Der skal nu være 2-4 spillere.”

De krav vi har fundet i kundes vision er:

Kundens krav

- Spillet kan spilles
- Spilleren kan lande på et felt og fortsætte derfra
- Der skal være et regelsæt for spillet, som skal overholdes
- Spillet skal kunne have 2 til 4 spillere
- Der skal være en brugergrænseflade

Vi har også skrevet de regler ned som vi har fundet i Monopoly junior regelsættet. Vi har valgt ikke at medtage spillets avancerede regler, da vi ikke ville have nok tid til at færdigudvikle spillet. Og da de ikke er nødvendige for kunden.

Spillets standard regler

- Yngste spiller starter
- Passer start og få 2 penge
- Hvis spilleren har nul penge skal de gå bankerot og spillet slutter
- Der skal være specialfelter som f.eks. fængslet
- Der skal være chance kort som spilleren kan trække

Vi har delt kravene op i funktionelle- og non funktionelle krav.

Funktionelle krav

R1	Spillet kan spilles
R2	Spilleren kan lande på et felt og fortsætte derfra
R3	Yngste spiller starter
R4	Passer start og få 2 Monopoly penge
R5	Hvis spilleren har nul penge skal de gå bankerot og spillet slutter
R6	Der skal være specialfelter som f.eks. fængslet
R7	Der skal være chance kort som spilleren kan trække

Non-funktionelle krav

R8	Spillet skal kunne have 2 til 4 spillere
R9	Der skal være en brugergrænseflade

Analyse

I dette afsnit kan man se nogle af de diagrammer og artefakter, som vi har lavet for at gøre det nemmere at designe spillet

Use case Beskrivelse

Vi har lavet en fully dressed use case beskrivelse, som har gjort spillets gang tydelig for os.

Use case 1.0	Spil spillet
Primær aktør	Spillere
Success guarantee	Spillet er blevet spillet
Main success scenario	<ol style="list-style-type: none">1. Spillerne åbner programmet2. Spillerene indtaster deres alder3. Hvis man er i fængsel betal 1 monopoly dollar<ol style="list-style-type: none">a. Hvis man har et “get of jail free card” skal man bruge detb. Hvis man man ikke har nogle penge4. Spiller kaster med terningerne5. Spiller rykker frem efter terningernes anvisning6. Hvis spiller har passeret startfeltet<ol style="list-style-type: none">a. Spiller modtager 2 monopoly-dollars7. Spiller lander på et felt<ol style="list-style-type: none">a. Hvis feltet er en grund<ol style="list-style-type: none">i. Hvis feltet er ejet af en anden spillerii. Hvis feltet ikke er ejet

	<ul style="list-style-type: none"> b. Hvis feltet er et chance-felt c. "Go to jail" felt d. Hvis feltet er et "free parking" eller "just visiting" felt e. Feltet er start <p>8. Næste spillers tur</p>
Alternativt flow	<p>3.a) Kom ud af fængsel og mist kortet.</p> <p>3.b)*</p> <p>7.a.i.)* Betal leje</p> <p>7.a.ii.)* Spilleren skal købe feltet</p> <p>7.b)* Vis og udfør chance-kort</p> <p>7.c) Ryk spillebrik til fængsels feltet. Modtag ikke penge for at komme over start.</p> <p>7.d) Der sker ingenting</p> <p>7.e) Gå til punkt 6.a, og skip 6.a næste tur.</p> <p>*) Hvis spiller ikke har råd til en transaktion, og ikke ejer et felt, sluttet spillet. De resterende spilleres penge tælles op. Den med flest penge vinder. Hvis det er lige, tælles hver spillers ejendomme. Hvis spilleren derimod ejer et felt, skal spilleren give sit felt til den person/bank han skylder penge.</p>
Specielle krav	

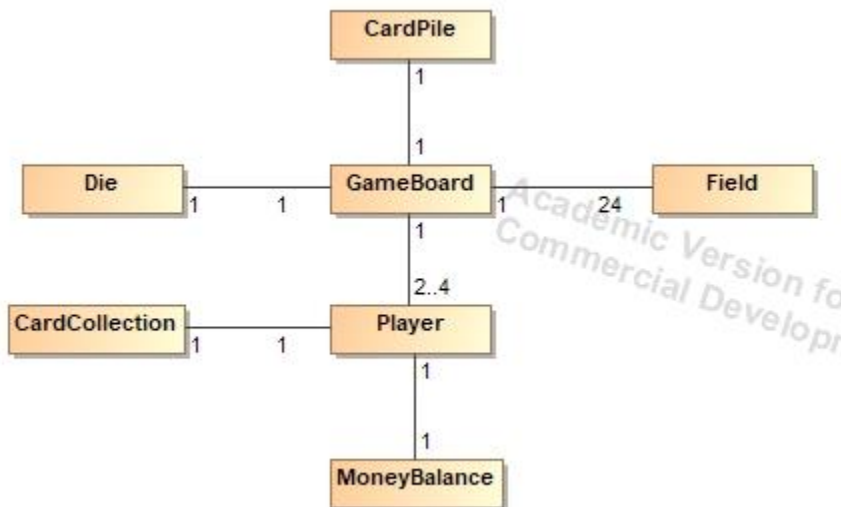
Use Case diagram

Vi har lavet et use case diagram.



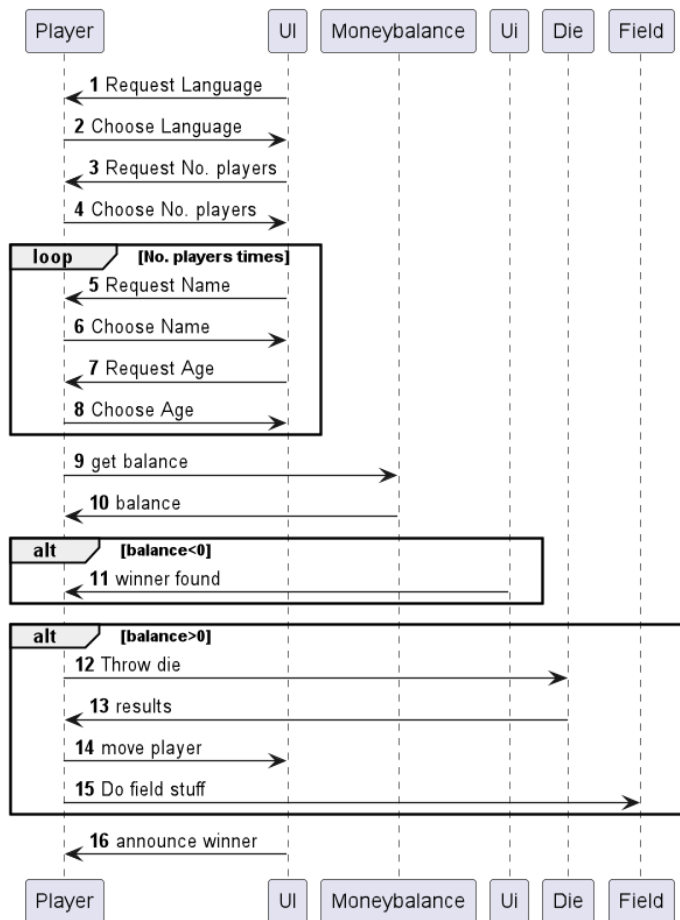
Domæne model

Vi har lavet en domænemodel for at få et overblik over hvilke dele af domænet, som skal snakke sammen og hvad der i det hele taget indgår i domænet.



Sekvensdiagram

Vi har lavet et sekvens diagram som danner et overblik over spillets gang og hvilke klasser der snakker sammen på hvilke tidspunkter

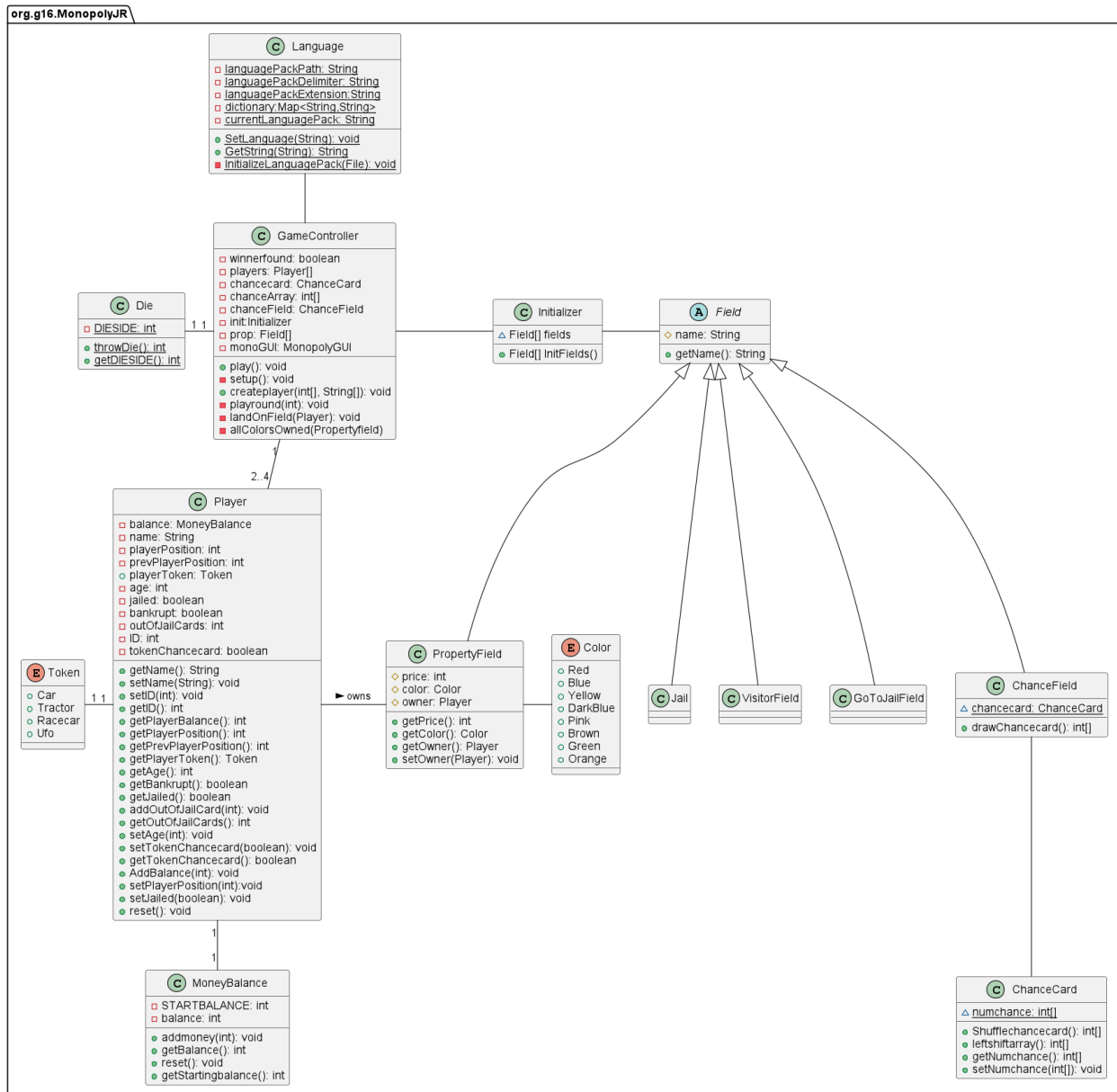


Design

I dette afsnit kan man se hvilke diagrammer og artefakter vi har brugt til at designe vores system.

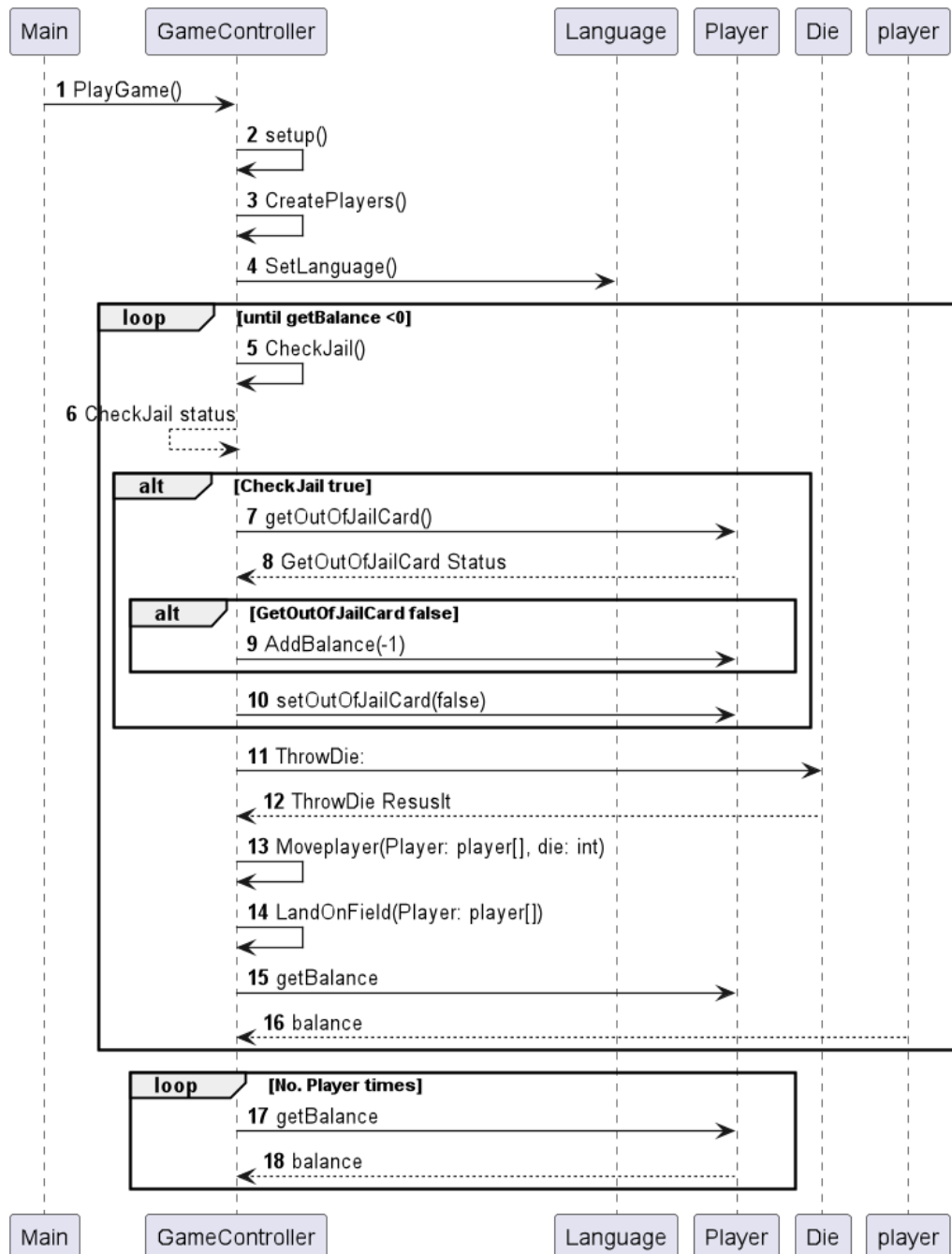
Design Klassediagram

Vi har lavet et design klassediagram, for at danne et bedre overblik over domænet, samt finde ud af hvilke metoder og attributter de forskellige klasse skulle have.



System sekvensdiagram

Vi har lavet et system sekvensdiagram for at se hvordan systems flow er. Der er blevet udeladt nogle metodekald i diagrammet. Der bliver f.eks kaldt mange forskellige metoder i “landOnField” metoden.



Implementering

Spillet består af 2 essentielle dele GUI som spillerne interagerer med og koden der håndterer logikken.

Player

I spillet kan man spille to op til fire spillere. Vi har så valgt at bruge Player klassen fra vores CDIO 2 projekt og opdateret klassen med nye funktioner. En af Monopoly junior regler er at hver spiller har en brik på spillebrættet der er så chance kort der udføre specielle funktioner på spilleren. Vi bruger så en enumerator klasse til at gemme de forskellige brikker man kan spille som. Player klassen har så en atribut kaldet token der indeholder en brik.

```
public Token playerToken;
```

```
public enum Token {  
    4 usages  
    Car,  
    3 usages  
    Tractor,  
    3 usages  
    Racecar,  
    3 usages  
    Ufo  
}
```

Når vi så opretter spillerne, giver vi dem en token.

```
tractor = new GUI_Car(Color.blue, Color.green, TRACTOR, HORIZONTAL_DUAL_COLOR);  
guiPlayer = new GUI_Player(player, startingBalance, tractor);
```

Når en af spillerne så trækker dette specifikke chance kort kan vi se hvilken spiller der er bilen og så den tilsvarende spiller få kortet.



Fields

I Monopoly junior er der 5 forskellige felter man kan lande på; ejendom, på besøg, fængsel, gå til fængsel og prøvlykken felt. I stedet for at have 5 forskellige klasser med de samme attributter, så som navn, har vi brugt arv til at spare på koden. Vi har så en abstrakt Field klasse som indeholder et navn og en getter metoder der giver navnet som en string.

```
3 public abstract class Field {  
4     protected String name;  
    Magnus  
5     public Field(String name) { this.name = name; }  
    8  
    Magnus +1  
9     public String getName() { return name; }  
    2  
3 }  
4
```

Det mest forekommende felt er ejendoms feltet. Vi har så en PropertyField klasse der extender Field. Den indeholder en pris, farve og ejer. Ligesom i spiller klassen har vi lavet en enumerator klasse der indeholder alle de farve felterne kan være.

```
4  
56 usages Magnus  
5 public class PropertyField extends Field {  
    2 usages  
6     protected int price;  
    2 usages  
7     protected Color color;  
    2 usages  
8     protected Player owner;  
    16 usages Magnus
```

Når vi så opretter et ejendoms felt giver vi den et navn, farve og pris.

```

16 usages  Magnus
9      public PropertyField(String name, Color startingColor, int price){
10          super(name);
11          this.color = startingColor;
12          this.price = price;
13      }

```

Vi opretter alle felter inde i vores initializer klasse, hvor vi indsætter værdierne og giver dem en plads på spillebrættet.

```

25 usages
public class Initializer {
    Field[] fields = new Field[24];

    /**
     * Sets values for every field in the game
     * And adds them into an array
     * @return array with fields
     */
    2 usages  Magnus +2
    public Field[] InitFields(){
        //field 0
        VisitorField start = new VisitorField( name: "start");
        fields[0] = start;
        //field 1
        PropertyField burgerBaren = new PropertyField( name: "burger", Color.Brown, price: 1);
        fields[1] = burgerBaren;
        //field 2
        PropertyField pizzaHuset = new PropertyField( name: "pizza", Color.Brown, price: 1);
        fields[2] = pizzaHuset;
    }
}

```

Vi har så en liste over alle felter der er i spillet i et array. Her kan vi så læse værdierne fra arrayet i GameControllern.

Chancekort

Vores chancekort bliver repræsenteret af et array af integers, som går fra 1 til 20. I starten af spillet bliver arrayet blandet af denne metode.


```

public int[] Shufflechancecard(){
    Random rand = new Random();

    for (int i = 0; i < numchance.length; i++) {
        int randomIndexToSwap = rand.nextInt(numchance.length);
        int temp = numchance[randomIndexToSwap];
        numchance[randomIndexToSwap] = numchance[i];
        numchance[i] = temp;
    }
    System.out.println(Arrays.toString(numchance));
    return numchance;
}

```

Hver gang man lander på et prøvlykken felt bliver metoden “doChanceCard” kaldt. “doChanceCard” er en metode som har et switch case med 20 cases. Den case som bliver udført er den der svarer til den første plads i arrayet af integers. Til sidst i “doChanceCard” bliver der kaldt en metode som leftshifterarrayet.

```

public int[] leftshifterarray(){
    int[] proxy = new int[numchance.length];
    for (int i = 0; i < numchance.length-1; i++) {
        proxy[i] = numchance[i + 1];
    }
    proxy[numchance.length-1] = numchance[0];
    return proxy;
}

```

Når man leftshifter arrayet lægger man det kort man har udført nederst i bunken.

Dokumentation

- **Forklar hvad arv er.**

Arv er en software mekanisme brugt i UML. Arv bruges til at overføre attributer og metoder fra en klasse til en anden. Dette sker ved at der sættes en base-klassen også kendt som superklassen og så kobles den til en sub-klasse som så fører arven videre. En sub-klasse kan også arve fra flere forskellige base-klasser.

- **Forklar hvad abstract betyder.**

Abstract er noget som en klasse eller en metode kan være. Hvis en metode er abstract vil det sige at den ikke kan instantieres. Dette bruges når man anvender polymorfi, og man ikke vil have at man skal kunne instantiere superklassen.

En metode kan også være abstract, men kun hvis klassen den er i også er abstract. Man lavede en abstract metode ved ikke at lave implementationen og så først gøre det i en af subklasserne.

- **Fortæl hvad det hedder hvis alle fieldklasserne har en landOnField metode der gør noget forskelligt.**

Hvis alle field klasserne har en “landOnField” metode, som gør noget forskelligt hedder det at man overrider metoden. Override er en af måderne man opnår run time polymorfisme. Det hedder run time polymorfisme, fordi der først bliver besluttet hvilken en af metoder der bliver udført ved runtime.

GRASP-Patterns

Et eksempel på et pattern, som vi har brugt, er *Polymorphism*. Vores abstrakte klasse “*Field*” er en superklasse som arves af de forskellige typer af felter, som vi har på brættet. Dette giver mening, da alle felterne i høj grad er ensartede, men alligevel har lidt forskellig funktionalitet. Dette har i høj grad gavnet strukturen af vores program, og gør det nemmere at ændre i felterne eller tilføje nye typer af felter.

Derudover er vores “*GameController*”-klasse lavet ud fra mønstret *Controller*. Denne håndterer det overordnede flow i programmet. Ved at lave GameController klassen, skulle vi passe på, at den ikke blev for stor. Dog vurderede vi, at det var den bedste løsning, da det skaber højere samhørighed mellem de andre klasser i programmet. Generelt i programmet, har vi indtænkt *Low Coupling* og *High Cohesion* og forsøgt at lave klasser med veldefinerede ansvarsområder og færrest mulige koblinger til andre klasser. Derudover har vi i Player klassen indtænkt *Creator* mønstret, idét den bliver brugt til at instantiere MoneyBalance objekter. Dette gør, at alle MoneyBalance-objekterne ikke skal kobles til GameController. Derfor skaber det højere samhørighed og lavere kobling.

Test

For at teste programmet, og for at se om det opfylder kravene, har vi lavet 10 JUnit test fordelt på to kategorier; spiller test og game controller test. Her tester vi de to klassers funktionalitet.

Derudover har vi lavet en brugertest for at se om brugeren kan lide produktet og få feedback fra hvad vi kan gøre bedre. og en begrænsnings test.

Unit test

Til hver unit test giver vi det et id og opskriver en test case.

Spiller test

PT01. Hvis en spiller passere start skal de få 2\$

Test case ID	PT01
Resumé	Test at en spiller får 2\$, hvis de passerer start
Krav	R4
preconditions	Spillet spilles
postconditions	Spillet bliver stadig spillet
Test procedure	<ol style="list-style-type: none">1. En spiller bliver oprettet2. Deres position bliver sat til felt 203. Deres position bliver opdateret til felt 24. Deres balance bliver læst
Test data	Spillerens balance = 20
Forventet resultat	Spillerens balance bliver nu forhøjet med 2
Faktisk resultat	Spillerens balance = 22
Status	Passeret
Testet af	Magnus Kjeldsen
Dato	22-11-2022
Test	Intellij IDEA 2022.2 (Ultimate Edition)

environment	Build #IU-222.3739.54, built on August 16, 2022 On Windows 11 Home
-------------	---

Test-koden

```

19      @Test
20      public void passStartTest() {
21          Player player = createPlayer( age: 12, ID: 0, Test: "Test");
22          player.setPlayerPosition(20);
23          player.setBalance(20);
24          int prevBalance = player.getPlayerBalance();
25          player.setPlayerPosition(2);
26          if (player.getPlayerPosition() < player.getPrevPlayerPosition()) {
27              player.AddBalance(2);
28          }
29          assertEquals( expected: prevBalance+2, player.getPlayerBalance());
30      }
31

```

Et af reglerne i Matador junior er at når en spiller passere start for de 2 Monopoly penge. Vi tester så om de gør det.

I testen opretter vi så en spiller og deres position bliver så sat på boarded. Vi gemmer så værdien for deres balance. Derefter rykker vi spilleren over start feltet. Vi kører så koden metoden checkPassStart. Vi kan dog ikke køre metoden fra gameController klassen da den indholder GUI og i testen opretter vi ikke GUI. Derfor må vi køre koden sådan som den står på linje 26. Vi kontrollerer så til sidst om spilleren har fået 2 monopoly penge. Med denne test fandt vi ikke nogen fejl i vores kode.

PT02. Hvis en spiller kommer i fængsel og de ingen “kom ud af fængsel” kort har betaler de så for at komme ud?

Test case ID	PT02
Resumé	Hvis en spiller kommer i fængsel og de ingen “kom ud af fængsel” kort har betaler de så for at komme ud?

Krav	R4
preconditions	Spillet spilles
postconditions	Spillet bliver stadig spillet
Test procedure	<ol style="list-style-type: none"> 1. En spiller bliver oprettet 2. De bliver sat i fængsel 3. Der bliver checket om de har et "kom ud af fængsel" kort 4. Deres balance bliver læst
Test data	Spillerens balance = 20
Forventet resultat	Spillerens balance bliver nu lavere med 1
Faktisk resultat	Spillerens balance = 19
Status	Passeret
Testet af	Magnus Kjeldsen
Dato	22-11-2022
Test enviroment	Intellij IDEA 2022.2 (Ultimate Edition) Build #IU-222.3739.54, built on August 16, 2022 On Windows 11 Home

Test-koden

```
32      @Test
33      public void jailTest(){
34          Player player = createPlayer( age: 12, ID: 0, Test: "Test");
35          player.setJailed(true);
36          player.setBalance(20);
37          int prevBalance = player.getPlayerBalance();
38          if (player.getJailed()){
39              if (player.getOutOfJailCards() > 0){
40                  player.addOutOfJailCard(-1);
41                  player.setJailed(false);
42              }
43              if (player.getPlayerBalance() > 1){
44                  player.AddBalance(-1);
45                  player.setJailed(false);
46              }
47          }
48          assertEquals( expected: prevBalance-1, player.getPlayerBalance());
49      }
50
```

Når en spiller ryger i fængsel og de ikke har en et “kom ud af fængsel” kort skal de ifølge reglerne betale 1 monopoly penge for at blive lødsladt.

I testen opretter vi en spiller og sætter dem i fængsel. Vi kører så metoden der håndtere fængslet. Vi forventer at spilleren mister en monopoly penge. Med denne test fandt vi ikke nogen fejl i koden.

PT03. Hvis en spiller kommer i fængsel og de har et “kom ud af fængsel” kort kommer de så gratis ud?

Test case ID	PT03
Resumé	Hvis en spiller kommer i fængsel og de har et “kom ud af fængsel” kort kommer de så gratis ud?
Krav	R4
preconditions	Spillet spilles

postconditions	Spillet bliver stadig spillet
Test procedure	<ol style="list-style-type: none"> 1. En spiller bliver oprettet 2. De bliver sat i fængsel 3. Der bliver checket om de har et "kom ud af fængsel" kort 4. Deres balance bliver læst
Test data	Spillerens balance = 20
Forventet resultat	Spillerens balance forbliver det samme
Faktisk resultat	Spillerens balance = 19
Status	Fejlet
Testet af	Magnus Kjeldsen
Dato	22-11-2022
Test enviroment	IntelliJ IDEA 2022.2 (Ultimate Edition) Build #IU-222.3739.54, built on August 16, 2022 On Windows 11 Home

```

@Test
public void jailCardTest(){
    Player player = new Player(Token.Car);
    player.setAge(12);
    player.setID(0);
    player.setName("Test");
    player.setJailed(true);
    player.addOutOfJailCard(1);
    System.out.println(player.getOutOfJailCards());
    int prevBalance = player.getPlayerBalance();
    if (player.getJailed()){
        if (player.getOutOfJailCards() > 0){
            player.addOutOfJailCard(-1);
            player.setJailed(false);
        }
        if (player.getPlayerBalance() > 1){
            player.AddBalance(-1);
            player.setJailed(false);
        }
    }
    assertEquals(prevBalance, player.getPlayerBalance());
}
}

```

Denne test kører samme metode som test PTO2. Her giver vi dog spilleren et “kom ud af fængsel” kort. Dette kort gør at de ikke skal betale for at komme ud af fængslet, men i stedet kan bruge deres chance kort. Vi forventer at spillerens balance forbliver det samme som før de blev sat i fængsel. Det vi forventede var dog ikke det der skete. Selvom spilleren havde et “kom ud af fængsel” kort betalte de dog stadig 1 monopoly penge.


```
org.opentest4j.AssertionFailedError:  
Expected :20  
Actual   :19  
<Click to see difference>
```

Spillerens balance burde stadigvæk være 20, men den blev til 19. Vi kiggede så vores kode igennem for at se hvad der gik galt. I koden er der to if-statements der kontrollerer om spilleren har en mindst en på deres balance og om de har et “kom ud af fængsel” kort. Dette er en fejl da koden der tager penge fra deres balance kun skal køre når de ikke har et “kom ud af fængsel” kort.

Vi har så rettet koden ved at ændre det ene if-statement til et else if. Efter denne rettelse passere programmet testen.

```
if (player.getJailed()){  
    if (player.getOutOfJailCards() > 0){  
        player.addOutOfJailCard(-1);  
        player.setJailed(false);  
    }else if (player.getPlayerBalance() > 1){  
        player.AddBalance(-1);  
        player.setJailed(false);  
    }  
}
```

PT04. Hvis en spiller går bankerot stopper spillet så?

Test case ID	PT04
Resumé	Hvis en spiller går bankerot stopper spillet så?
Krav	R5
preconditions	Spillet spilles

postconditions	Spillet bliver stadig spillet
Test procedure	<ol style="list-style-type: none"> 1. 3 spillere bliver oprettet 2. 100 penge bliver taget fra spiller et balance 3. Metoden der tjekker om der er en spiller der er gået bankerot bliver kørt 4. Vinderen af spiller bliver læst
Test data	<p>Spiller 1 balance = -100</p> <p>Spiller 2 balance = 100</p> <p>Spiller 3 balance = 20</p>
Forventet resultat	Spiller 2 vinder
Faktisk resultat	Spillerens balance = 29
Status	Passeret
Testet af	Magnus Kjeldsen
Dato	22-11-2022
Test enviroment	<p>Intellij IDEA 2022.2 (Ultimate Edition)</p> <p>Build #IU-222.3739.54, built on August 16, 2022</p> <p>On Windows 11 Home</p>

```

69      @Test
70      public void bankruptTest(){
71          Player player1 = createPlayer( age: 1, ID: 0, Test: "One");
72          player1.AddBalance(-100);
73
74          Player player2 = createPlayer( age: 2, ID: 1, Test: "Two");
75          player2.AddBalance(100);
76
77          Player player3 = createPlayer( age: 3, ID: 2, Test: "Three");
78
79          Player[] players = {player1, player2, player3};
80          if (player1.getBankrupt()){
81              int max = 0;
82              int playerNum = 0;
83              for (int i = 0; i<players.length; i++){
84                  int balance = players[i].getPlayerBalance();
85                  if (balance > max){
86                      max = balance;
87                      playerNum = i;
88                  }
89              }
90              assertEquals(player2,players[playerNum]);
91          }
92      }
93

```

Når en spiller går bankerot skal den spiller med flest penge vinde spillet.

I denne test bliver der oprettet 3 spillere med 3 forskellige mængder penge på deres balance. Da spiller et har negative penge bliver de sat som have været gået bankerot. Pågrund af dette skal der så findes en vinder. Vi har så en metode der kigger alle spillernes balance igennem og ser hvem der har flest penge. Vi forventer at det bliver spiller 2 da de har flest penge. Det er også det resultatet bliver. Med denne test fandt vi ikke nogen fejl i koden.

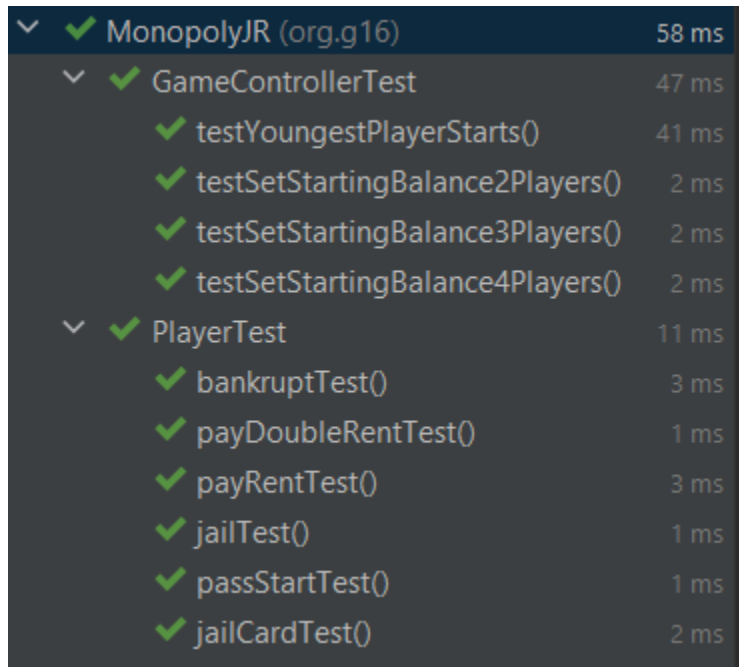
Udover disse 4 test har vi også lavet 6 andre junit test. 2 af dem tester funktioner på player klassen og 4 af dem på game controller klassen. På spilleren tester om de betaler den rette mængde penge også når begge felter er ejet af den samme spiller. På game controlleren tester vi om spillerene for den rette mængde penge når spillet bliver oprettet og at den yngste spiller starter.

GameController test

Test af setStartingBalance metode; 4 spiller, 3 spillere og 2 spillere.

Test af om den yngste spiller starter.

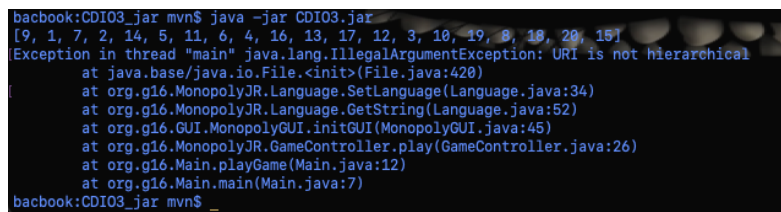
Her ses resultatet af alee junit testene.



✓ MonopolyJR (org.g16)	58 ms
✓ GameControllerTest	47 ms
✓ testYoungestPlayerStarts()	41 ms
✓ testSetStartingBalance2Players()	2 ms
✓ testSetStartingBalance3Players()	2 ms
✓ testSetStartingBalance4Players()	2 ms
✓ PlayerTest	11 ms
✓ bankruptTest()	3 ms
✓ payDoubleRentTest()	1 ms
✓ payRentTest()	3 ms
✓ jailTest()	1 ms
✓ passStartTest()	1 ms
✓ jailCardTest()	2 ms

Test af build til jar-fil

Et krav er, at vi bygger programmet til en jar-fil. Vi besluttede derfor at teste programmet, efter det var blevet buildet. Vi fandt en fejl, der gjorde at programmet ikke kunne køres. Vi fandt følgende fejl.



```
bacbook:CDI03_jar mvn$ java -jar CDI03.jar
[9, 1, 7, 2, 14, 5, 11, 6, 4, 16, 13, 17, 12, 3, 10, 19, 8, 18, 20, 15]
Exception in thread "main" java.lang.IllegalArgumentException: URI is not hierarchical
    at java.base/java.io.File.<init>(File.java:420)
    at org.g16.MonopolyJR.Language.SetLanguage(Language.java:34)
    at org.g16.MonopolyJR.Language.GetString(Language.java:52)
    at org.g16.GUI.MonopolyGUI.initGUI(MonopolyGUI.java:45)
    at org.g16.MonopolyJR.GameController.play(GameController.java:26)
    at org.g16.Main.playGame(Main.java:12)
    at org.g16.Main.main(Main.java:7)
bacbook:CDI03_jar mvn$ _
```

Vi kunne se at fejlen var i den måde Language klassen læser sprogfilerne. Kørte vi programmet fra IntelliJ, virkede programmet fint, da det læser filerne fra den stige, hvor resource-filerne ligger i. Efter at blive buildet til en jar-fil, blev filerne pakket ind sammen med jar-filen og kunne ikke længere læses som en fil. Vi fiksat problemet ved at læse resursen som en file stream i stedet for en fil.

Bruger test

Vi har lavet en bruger test, hvor vi fik to personer til at spille en runde af matador spillet. Vi opstillede et par test-punkter og bad om feedback til programmet.

Kan bruger navigere programmet

Brugere havde ikke de store problemer med at navigere programmet.

Er programmet intuitivt at bruge

Programmet navigeres ved at trykke på OK-knappen øverst i GUI'et. Det er nemt at bruge men det er også nemt bare at klikke videre uden at læse teksten. Desuden sker stort set hele spillerens runde på et enkelt tryk, så det kan igen være nemt at overse, hvad der er sket, hvis man kommer til at gå for hurtigt videre.

Feedback

- Teksten er meget lille, så det er nemt at overse hvad der står og bare trykke videre.
- Let at forstå.
- Kan være svært at følge brikkerne, når de rykker.
- Alt taget i betragtning et fint lille spil.

Spillerne fuldførte spillet uden at rende ind i nogen bugs. De havde ingen problemer med at følge instrukserne på skærmen og indtaste deres navn og alder. Derefter gik spillet rimelig ligetil, de skiftedes til at trykke på knappen og rulle terningen. De landede flere gange på chance feltet og spillet udførte hvad der stod på kortene uden problemer. Spillet endte med at spiller 1 vandt efter at spiller 2 løb tør for penge.

Alt taget i betragtning virker spillet som det skal og de fleste mangler kommer i at det er nemt at klikke for hurtigt videre og at der ikke er nogen animation på at rykke brikkerne.

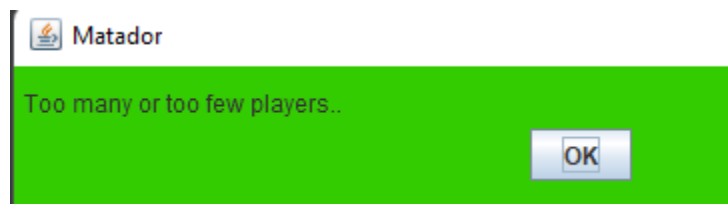
Begrænsningstest

Test case ID	PT07
Resumé	Test om spiller kan indtaste forskellige værdier ind i et navne felt og et alders felt.

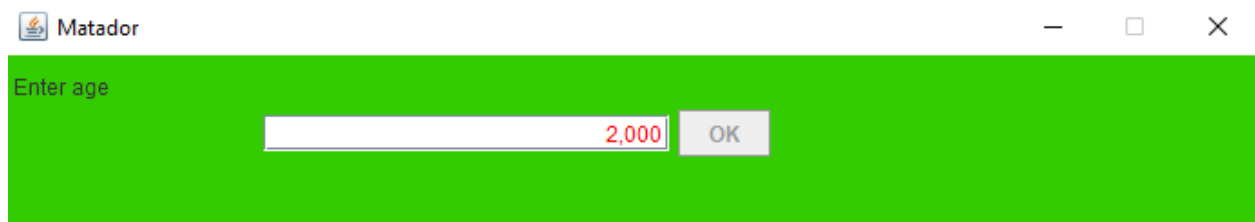
	Test hvor mange antal spillere kan spille spillet.
Krav	R1 og R3
preconditions	Spillet spilles
postconditions	Spillet bliver stadig spillet
Test procedure	<ol style="list-style-type: none"> 1. 4 spillere bliver oprettet. 2. Indtast antallet af spillere. 3. Metoden der tjekker om antal af spillere er under en begrænsning. 4. Hvis spillere antal er mindre eller mere end begrænsningen, skal der vises en error besked. 5. Indtast spillerens navn og alder. 6. Metoden der tjekker om spillerens navn og alder er gyldigt bliver kørt. 7. Hvis værdierne er ugyldige, så skal teksten blive rød og ok knappen er umulig at trykke på.
Test data	<p>Navn:</p> <ul style="list-style-type: none"> - Indtast navn med numeriske tegn. - Indtast andre sprog som ikke er latinske bogstaver - Indtast symboler og tegn i stedet for bogstaver så som @?!#% - Indtast mange bogstaver <p>Alder:</p> <ul style="list-style-type: none"> - Indtast 0 - 999,999,999 - Indtast bogstaver - Indtast tallet 1,000,000,000 - Indtast et tal som er mindre end 0 - Indtast tal med et punktum eller et komma <p>Antal af spiller:</p> <ul style="list-style-type: none"> - Indtast spiller antal 0 - 1. - Indtast spiller antal mere end 4.
Forventet resultat	<p>Navn og alder felter:</p> <p>Teksten skal vises i en rød farve og ok knappen skal være usynligt eller</p>

	<p>umulig at trykke på, når en ugyldig værdi bliver indtastet.</p> <p>Antal felt:</p> <p>En error besked skal vises “For mange eller for få spillere”, hvis der bliver indtastet et antal mindre end 2 eller mere end 4.</p>
Faktisk resultat	<p>Navne felt:</p> <ul style="list-style-type: none"> - Programmet accepter de special alfabet, bogstaver og tal i et navn felt. <p>Alder felt:</p> <ul style="list-style-type: none"> - Spilleren kan udfylde alderen fra 0 op til 999,999,999. - Hvis alderen bliver udfyldt med et punktum eller komma og bogstaver, så bliver teksten rød og ok knappen kan ikke trykkes på. <p>Antal felt: Virker som forventet.</p>
Status	Passeret
Testet af	Chyanee Christensen
Dato	24-11-2022
Test enviroment	<p>Intellij IDEA 2022.2 (Ultimate Edition)</p> <p>Build #IU-222.3739.54, built on August 16, 2022</p> <p>On Windows 11 Home</p>

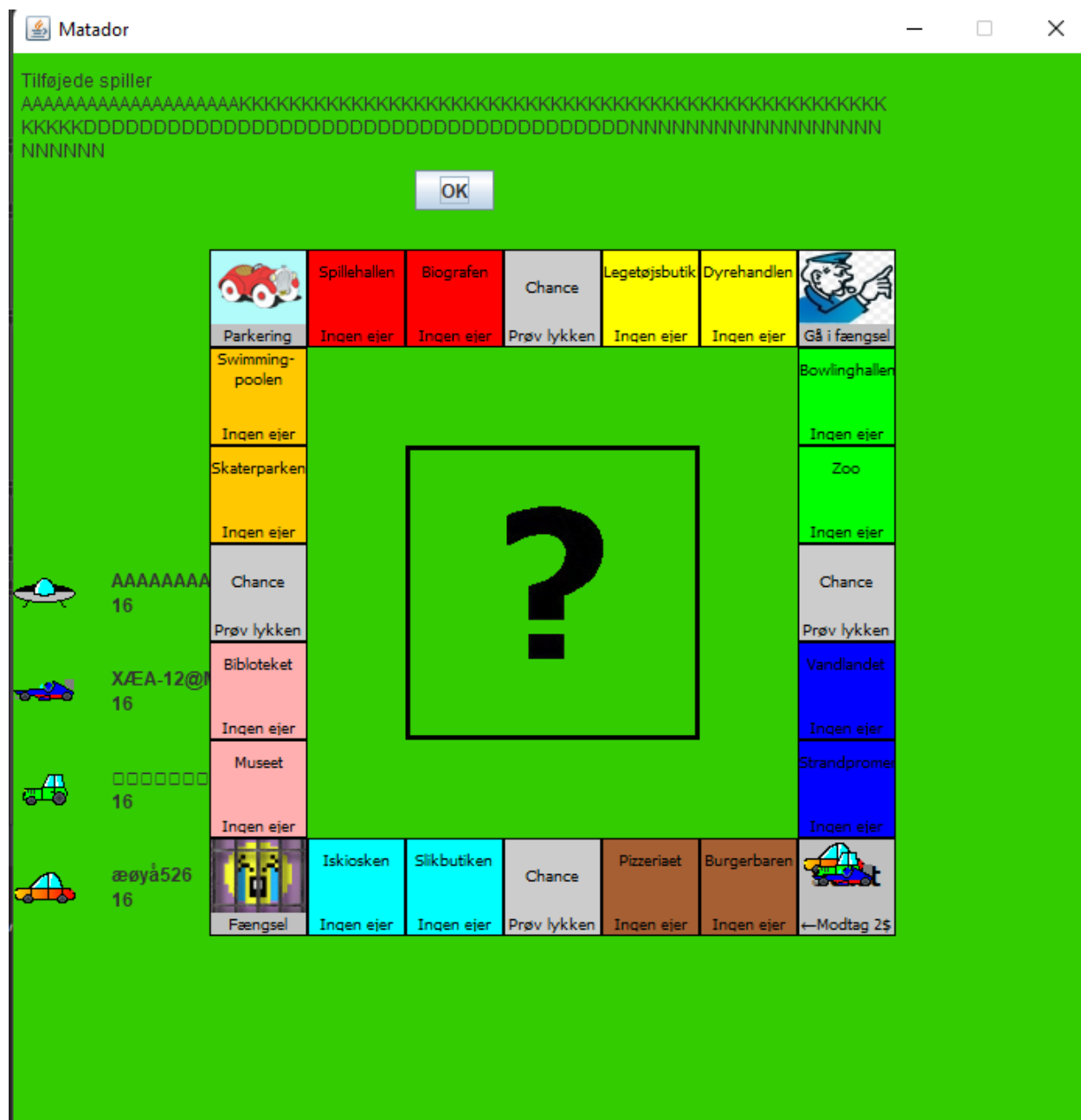
I spillet har vi har sat en grænse på antallet af spillere det skal være mellem 2 og 4 personer. Hvis antallet er mindre end 2 eller mere end 4, så vises der denne besked.



I alder feltet kan spillerene indtaste deres alder fra 0 op til 999.999.999 år uden at de for en error besked. Vi har sat en begrænsning så men ikke kan indtaste bogstaver og tegn i alders feltet.



I navne feltet er der ingen begrænsning for hvad man kan indtaste



I alders feltet har vi sat mange begrænsninger for hvad man kan indtaste da hvad spilleren indtaster her har en indflydelse på spillet. Vi mangler dog nogle begrænsninger i navnefeltet, men da det ikke har en indflydelse på spillet har vi valgt ikke at prioritere det.

Konklusion

Vi startede med at analysere kundens vision for at finde kravene. Vi fandt ud af at de funktionelle krav var at spillet skulle spilles efter de regler, som vi har fået udleveret af kunden. Udover de funktionelle krav, skulle spillet også have en brugergrænseflade. For at lave den, brugte vi det GUI-package som vi fik udleveret. For at kunne lave spillet så det overholder alle kundets krav, begyndte vi med at lave nogle diagrammer og artefakter, såsom en domænemodel og en use case beskrivelse. Efter vi havde lavet analysedelen gik vi igang med designe hvordan koden skulle opsættes. Her lavede vi et design klassediagram og system sekvensdiagram. Da vi lavede vores design diagrammer tænkte vi på hvordan vi kunne overholde samtlige GRASP-patterns. Vi fandt bla. ud af at vi kunne anvende polymorfisme til at lave spillets felter, de kunne så arve fra en overordnet field klasse.

Da vi kodede spillet genbrugte vi noget af koden fra vores tidligere spil, bla. terninge klassen da den fungerede på samme måde. Vi kunne også genbruge language klassen også bare lave nye strings til dette spil. Den nye kode, som vi skulle lave til dette spil var chance kortene, field klassen og vi skulle også lave en ny game controller. Vi skulle også lave en GUI controller. Vi har lavet vores chancekort med et array af integers og en switch case. For at lave vores forskellige field klasser har vi, som sagt brugt polymorfisme. Vi kodede gamecontrolleren, så spillet blev spillet efter det flow, som vi har designet i de forskellige diagrammer.

Vi har lavet nogle forskellige test for at sikre os at spillet overholder de krav, som kunden har sat. Vi har lavet JUnit tests, hvor vi har testet centrale metoder i "Player"- og "GameController" klassen. Vi kunne ud fra de forskellige tests se at vores spil overholder kundenskrav. Vi har også lavet en bruger test, for at finde ud af hvordan det føles at spille spillet. Vi fandt ud at spillet er nemt at spille, men at det kan være svært at se hvad der skete på ens tur hvis man trykker for hurtigt.

Vi kan konkludere at vi har lavet et spil, som overholder kravene. Vi har lavet forskellige diagrammer og artefakter til at designe og analysere spillet, og vi har anvendt samtlige GRASP-patterns. Vi kunne have optimeret hvor godt det føles at spille spillet.

Bilag

Link til Github repo. <https://github.com/Magnusrk/CDIO3>

Link til regelsæt: https://drive.google.com/file/d/1VjPh7bKIXeVWRJ2H_xsSgV_x_IoaAz9T

Link til chance kort:

<https://drive.google.com/file/d/1ymvoT5xWlvprTZkSO6DtWz9byFvZA9h3/view>