02312

Indledende programmering

---

3 Ugesprojekt - Gruppe 16.

Thomas Arildtoft

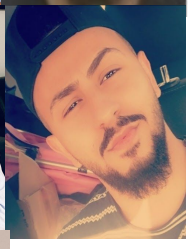
S193564

Omar Ebrahim

S163576

Ali Dadayev

s172858

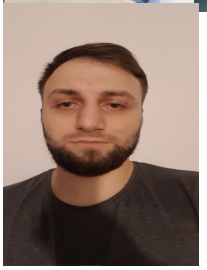


Salim Omar

s193472

David Lucas Møller

s173210



1. Timeregnskab.

Opgave / Navn	**David**	**Omar**	**Salim**	**Ali**	**Thomas**
Rapport	5	4	5	5	7
- Kravspecifikation	2	2	2	8	3
- Analyse	3	2	4	9	6
- Design	3	3	5	5	6
- Implementering	13	15	12	10	12
- Test	4	3	0	0	0
- Dokumentation	2	3	6	3	5
- I alt.	32	32	34	40	39

Tabel 1: Oversigt over det antal timer hvert medlem har brugt.

Github: https://github.com/salim963/Final_gr-16.git

Indholdsfortegnelse.

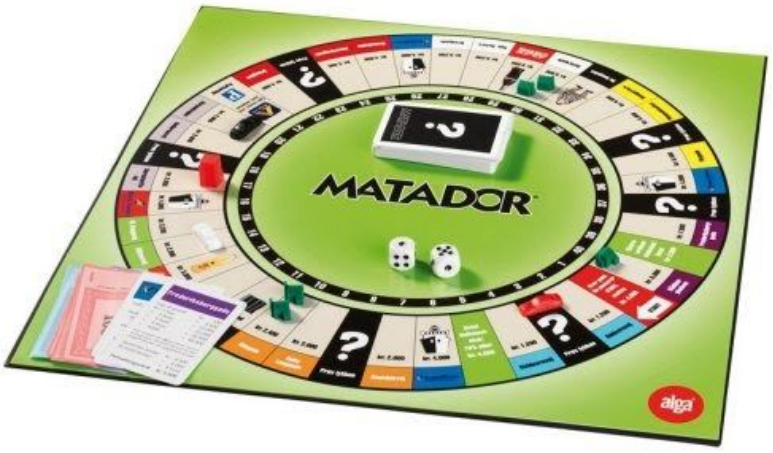
2. Indledning.

Rapporten er udarbejdet af gruppe 16 i 3 ugers forløbet til Indledende programmering - 02312. Vi er blevet bedt om at tage udgangspunkt i den fysiske udgave af Matador spillet. I den fysiske version er kravene til spillet:

- Skal være mellem 3 - 6 spillere.
- Slår med et terningesæt.
- Mulighed for rykke rundt på spillepladen.
- 40 Felter (8 forskellige typer af felter)

Spillerne starter alle sammen på feltet " Start " og den første spiller slår med terningerne, hvorefter at spillerens brik bevæger sig i urets retning, dette gentages for alle spillerne. Hvis man lander på en ejendom, har man muligheden for hvis ingen andre spillere ejer den, at købe ejendommen selv. Skulle man fravælge at købe ejendommen, så vil den komme på en åben auktion, hvor alle de andre spillere kan byde på den.

Lander man derimod på en ejendom, som en anden spiller ejer, skal man betale husleje til denne spiller. Huslejen kan variere alt efter hvor mange grunde man ejer i samme farve, det antal huse eller om der er et hotel på grunden. Skulle man være i den situation at man ikke kan betale huslejen eller nogle af de andre udgifter, der dukker op i løbet af spillet, så kan man pantsætte enten sine huse / hoteller eller ejendomme, dog kun til halve pris i forhold til hvad man selv betalte. Har man ingenting af værdi, så er man ude af spillet og resten af ens en beholdning, bliver betalt til ejeren af ejendommen, som man er landet på. Således fortsætter spillet indtil der kun er en spiller tilbage, som derved er vinderen af spillet.



Figur 1: Matador Spil.

3. Kravspecifikation.

Herunder inddeler vi forskellige krav, alt efter hvor vigtige de er for at have et funktionelt spil.

Timeregnskab. **1**

3.1 “ Must have ” krav.

1. Der skal være 3 til 6 spillere i spillet.
2. Hver spiller starter med 30.000 kr.
3. Spillerne skiftes til at slå med 2 terninger.
4. Der skal være 40 felter på spillebrættet.

5. Alle spillerne starter på feltet “ Start “.
6. Spillernes brik skal bevæge sig det antal øjne, som terninger viser.
7. Spillernes brik skal bevæge sig i urets retning på spillepladen.
8. Spillernes brik skal bevæge sig videre fra det felt, som de landede på ved sidste kast.
9. En spiller er ude af spillet når personen ikke kan betale en udgift og dermed er fallit.
10. Spillet fortsætter indtil der kun er én spiller tilbage, som ikke er gået fallit.

3.2 “ Should have ” Krav

1. Spillerne får 4000 kr. Hver gang de passere start feltet.
2. Man får et ekstra terningekast, hvis man slår 2 ens.
3. Slår man 2 ens, 3 gange i træk bliver man af den korteste rute sendt i fængsel, hænder det at man passere start, modtager man ikke 4000 kr.
4. Der skal være 28 felter, som er af typen “ Ejendomme “
    a. 22 felt grupper som som er tildelt forskellige farver, 2 af grupperne indeholder 2 felter og 6 af grupperne indeholder 3 felter.
    b. 4 af felterne er skibsredere (DFDS, Scandlines og Mols linjen)
    c. 2 af felterne er bryggerier (Tuborg & Coca-cola)
5. Der skal være 12 felter, som indeholder specielle aktioner.
    a. Der skal være 1 startfelt.
    b. Der skal være 6 “ Prøv lykken “ felter.
    c. Der skal være 1 “ På besøg i fængsel “ felt.
    d. Der skal være 1 “ Gå i fængsel “ felt.
    e. Der skal være 1 “ Gratis Parkering “ felt.
    f. Der skal være 1 “ Betal Indkomstskat “ felt.
    g. Der skal være 1 “ Ekstraordinær Indkomstskat “ felt.
6. Ubebygget ejendomme, rederier og bryggerier kan til enhver tid sælges imellem spillerne til en værdi de selv aftaler.
7. Hvis en spiller ejer alle ejendomme i sammen farve, er huslejen det dobbelte beløb og man kan nu bygge på ejendommene.
8. Man må ikke opgradere til et højere antal huse på ens ejendomme, før alle i farvegruppen har det samme antal huse på sig.

3.3 “ Could have “ Krav

1. Hvis man lander på et “ Prøv Lykken “ felt skal man trække et tilfældigt lykke kort og gøre som beskrevet.
2. Hvis man trækker et “ Benådningskort “ kan dette gemmes og eventuelt sælges til de andre spillere, for en sum man bliver enig om.
3. Hvis man lander på et felt som ikke ejes af anden spiller og man ikke ønsker at købe det så skal banken sætte det på auktion og alle spiller har ret til at deltage.
4. Når man har fire huse på alle ejendomme i farvegruppen, kan man opgradere til et hotel. Når man gør dette, giver man de fire huse tilbage til banken.
5. Man kan sælge sine huse til halv pris.
6. Man kan komme ud af fængsel ved at kaste 2 ens, men efter 3 forsøg skal man betale 1000. kr for derefter at rykke det antal felter, som øjnene viser.
7. Man kan ikke opkræve leje fra sine ejendomme imens man er i fængsel.

3.4 “ Wont have “ Krav

1. Maks antal balance
2. maks antal huse / Hoteller

3.5 Supplerende specifikationer FURPS+

Functionality:

- Spillet skal kunne spilles på DTU’s database.

Usability:

- Spillet skal være nemt.(spillet skal være klart og forståeligt).
- Den skal kunne spilles af børn eller en der ikke kender meget til teknologi.
  (spillet er underholdende, simpelt og har ikke brug for høj dygtighed)

Reliability:

- Spillet skal vise en besked hvis der er noget gået galt.
- Fejl skal forklares i beskeden.

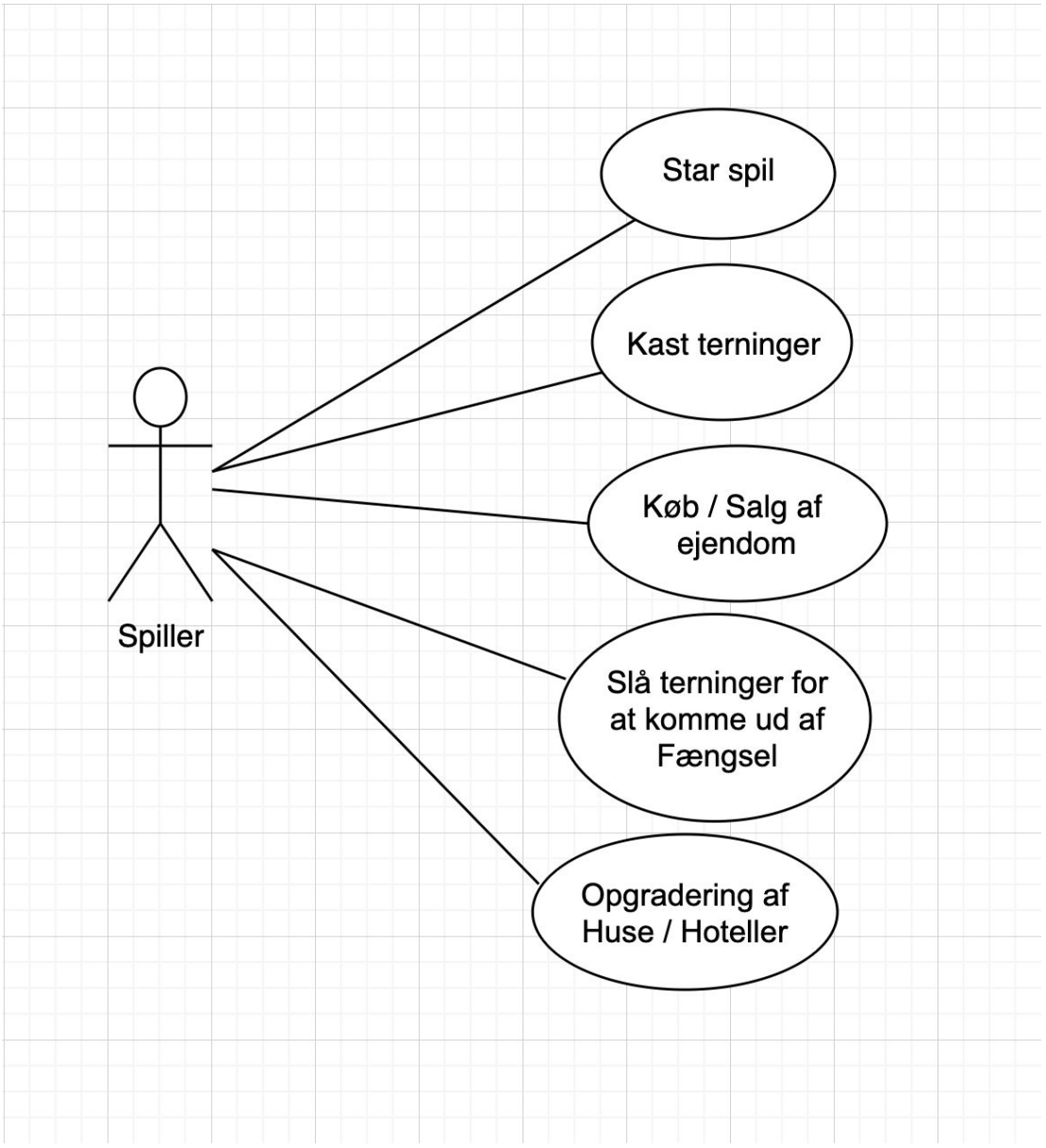
Performance:
- Spillet skal starte max efter 5 sekunder, når brugeren trykker på kast terninger.
- Man har maks 30 sekunder efter kast terninger for at vælge om man skal købe ejendom eller ej.

Supportability:
- Spillet skal være let at bruge og mulighed for at lave ændringer.
- Spillet skal kunne installeres uden høje krav fra brugeren da den køre på Java.

Implementation:
- Spillet udarbejdes i Java ved hjælpe af IntelliJ IDEA.

4. Analyse

4.1 Use case



Figur 2. Use case diagram.

Diagrammet herover, hvis vores primære aktør som er en spiller. Samtidig viser det også hvad vi anser, som de vigtige elementer i vores spil som bliver beskrevet nærmere i vores use cases nedenfor.

4.2 Use case beskrivelse.

Use case 1
Name: Start Spil.
Betingelser: Adgang til computeren os spillet tilgængelig
Goal: At starte spillet.

Beskrivelse: Spilleren åbner spillet op og indtaster spillerens navn. Hvor efter at spillet skal begynde.

Use case 2
Name: Kast terningerne
Betingelser: Adgang til computeren og spillet er startet.
Goal: Terningerne er kastet
Beskrivelse: Når spillet er gået i gang, vælger spilleren at trykke på "Kast terningerne". Efter han kaster terningerne kan brugeren se resultatet af sin kast.

Use case 3
Navn: Køb / Salg af ejendom.
Betingelser: At man enten ejer en ejendom eller lander på et felt ingen ejer.
Goal: Spilleren kan købe ejendomme og sælge ejendomme.
Beskrivelse: Spillerne skal kunne købe ejendommen på et felt, hvis ingen andre har købt det før ham. Spilleren skal også kunne vælge at kunne sælge sin ejendom til andre spillere hvis man ønsker det.

Use case 4
Navn: Slå terninger for at komme ud af Fængsel.
Betingelser: At man sidder i fængsel.
Goal: Brugeren kan slå op sig ud af fængslet.
Beskrivelse: Imens at brugeren sidder i fængsel, får personen 3 slag op til 3 ture, for at kunne slå ud af fængslet, dette gøres ved at få to terninger med samme antal øjne " Slå 2 Ens ".

Use case 5
Navn: Opgradering af Huse / Hoteller.
Betingelse: Spillet er åbnet, Brugeren ejer alle ejendomme af samme farve.
Goal: At kunne enten tilføje eller sælge bygninger på ens ejendom.
Beskrivelse: Hvis brugeren ejer alle ejendomme, enten af samme farve. Kan brugeren vælger at tilføje huse på ejendommene, dog kan der ikke tilføjes flere huse på ejendommene før alle har det samme antal dvs. 3 ejendomme med 1 hus giver dig tilladelse til at begynde at putte 2 huse på ejendommene.

4.3 Fully dressed use case.

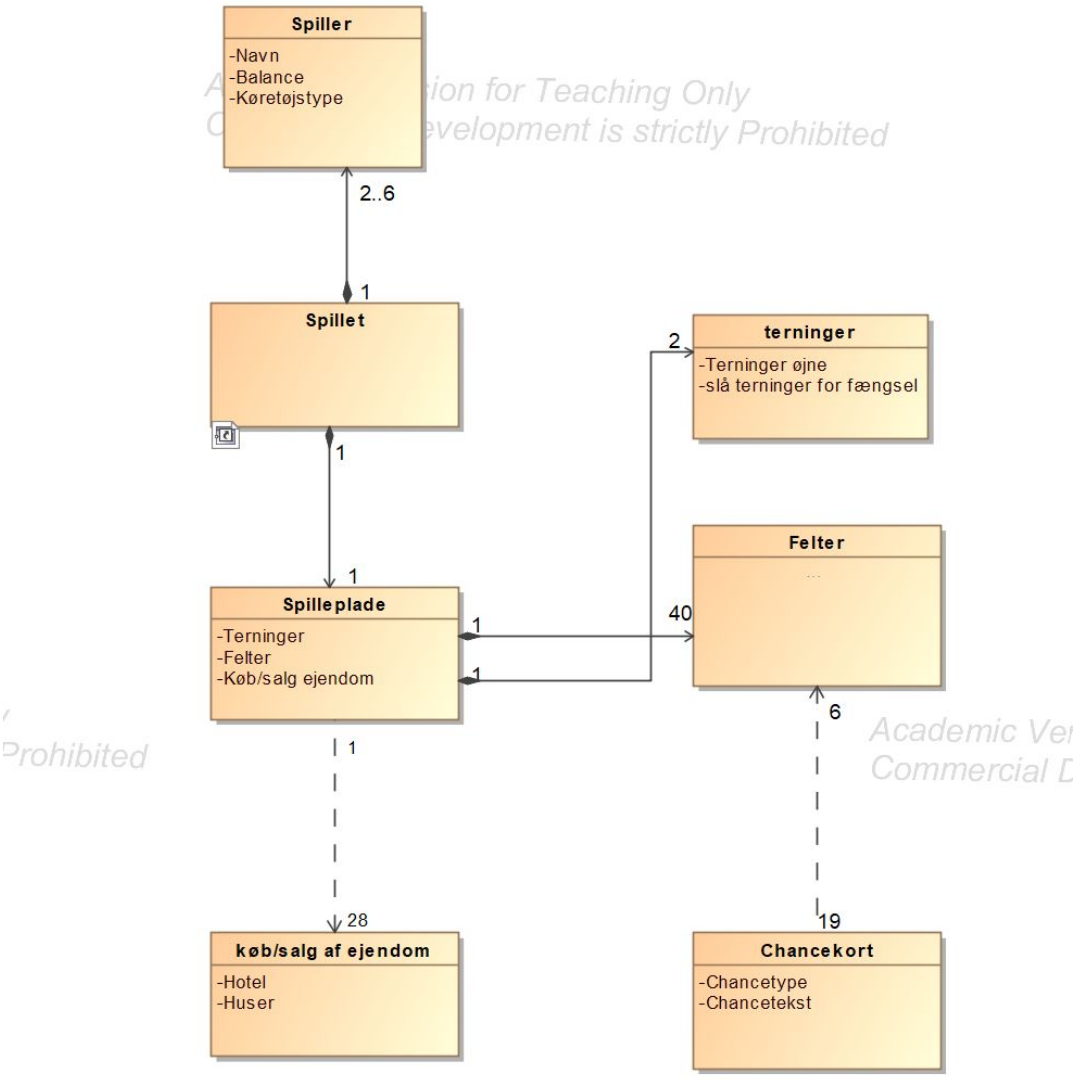
Use case sektion:	**Beskrivelse:**
Navn	Køb / Salg af ejendom

Scope	Matador spil
Level/niveau	
Primær aktør	Spilleren.
Andre interessenter	Ingen.
Forudsætninger/preconditions	At man ejer en ejendom eller at man lander på et felt ingen ejer.
Succeskriterier/postconditions	Spillerne kan købe ejendommen på et felt hvis dette ikke ejes af nogen. Spilleren kan sælge sine ejendomme til andre spiller.
Vigtigste succes scenarie	At spilleren kan købe eller sælge ejendomme.
Udvidelser	
Specielle krav	
Teknologi og dataliste	GUI_Ownable
Frekvens	Hver gang man ønsker det
Diverse	

4.4 Domænemodel.

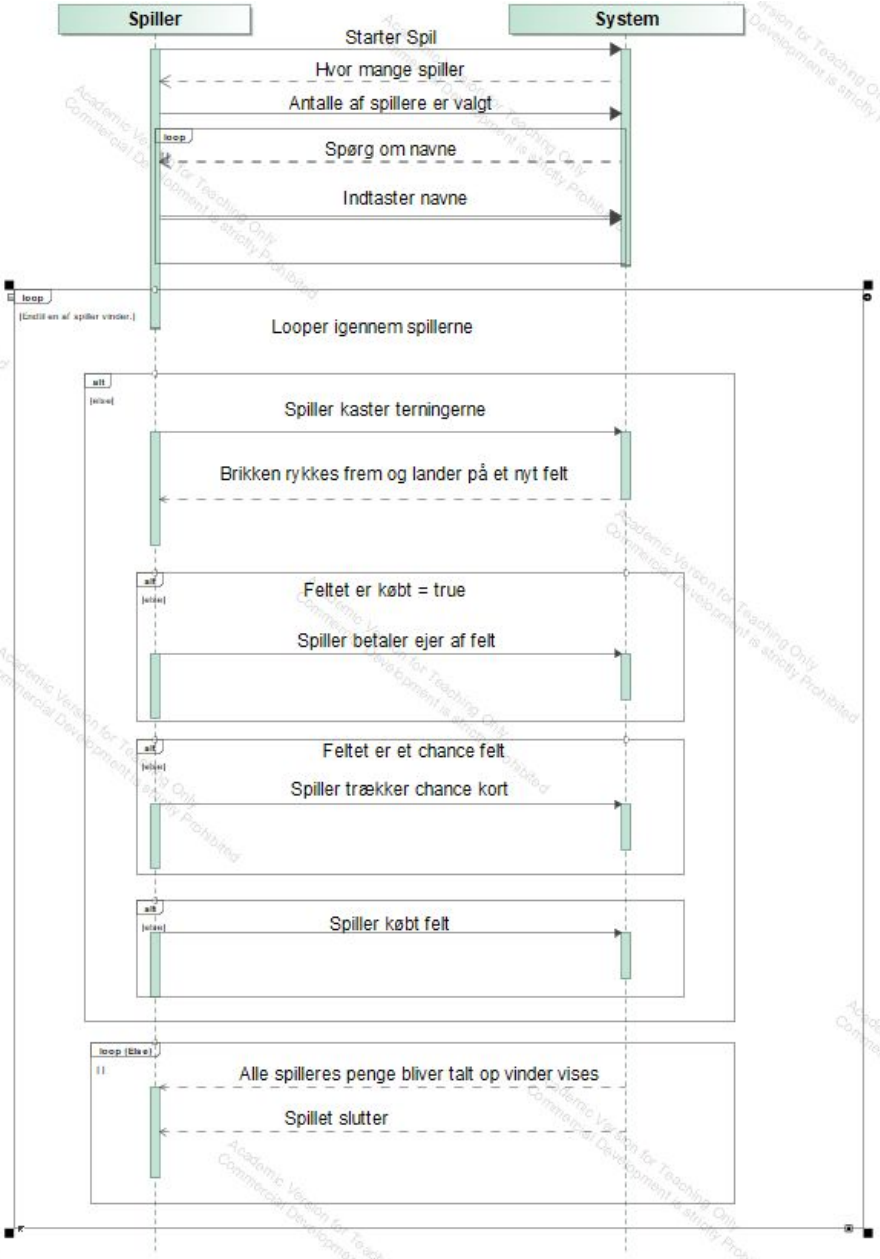
Kunden bad os om at lave et matadorspil, som følger de samme regler og retningslinjer, som det fysiske spil. Derfor har vi taget de fysiske objekter og inddraget i vores domænemodel.

I vores domænemodel kan det ses at alle vores klasser started ud fra “ Spillet “. “ Spillet “ klassen benytter “ Spiller “ klassen, hvor at hver spiller skal indtaste et navn og bliver tildelt en balance samt en køretøjstype. “ Spillet “ anvender derefter “ Spilleplade “ klassen som kalder på “ terninger “ for at slå med terningerne, for derefter at kalde på “ Felter “ for at rykke rundt på på spillepladen. Lander man på “ Chance Field “ feltet kaldes chancekort. Ligeledes hvis ønsker at købe / sælge ejendomme benyttes “ Køb / Salg af ejendom “



Figur 3: Domænemodel.

4.5 Systemsekvensediagram.



Figur 4: Systemsekvensdiagram.

5. Design

For at at kunne gøre vores arbejdsgang så simpel og effektiv, som muligt har vi valgt udarbejde et GRASP mønstre, et design klassediagram og et design sekvensdiagram. Disse er alle beskrevet og vist nedenfor.

5.1 Design mønstre og GRASP.

1. **Low coupling:**
   lav afhængighed, så påvirkningen ved ændringer reduceres, så genbrug og forståelse er lettere.
   det viser hvordan koblingen mellem klasserne er hvor den skal altid være lav,
   så blive det lettere at forstå delene og nemt at genbruge.
   -eksempel: vi har Player klasse som er koblet med Controller klasse og GuiView klasse

2. **High cohesion**
   Nemmere at forstå koden, og koden blive delt i forskellige klasser hvor der er tæt relaterede til hinanden. High cohesion hænge sammen med lov coupling.

   her vi skal være sikker at at hver klasse har kun de passende metoder (ensartet ansvarsområde)

   -eksempel: alle metode i vores kode er high cohesion

3. **Creator**
   Creator er ansvarlig for at oprette et objekt af en anden klasse.
   når vi har en klasse som er ansvarlige for at oprette en ny instanser af en anden klasse.
   -eksempel: vi har klasse GuiView som er ansvarlig for at oprette Gui og Gui-player.

4. **Controller**
   Controller er en klasse hvor alle handlinger foregår der og den er ansvarligt for modtage og koordinere inddata.
   controller adskiller logisk fra præsentation og holder systemet dele sammen.
   -eksempel på det er vores Controller klasse.

5. **Information expert**
   Information expert er den klasse da har relevante informationer der skal udføres opgaven.
   -eksempel: ControllerField

De 4 sidste punkter inden for GRASP som er:

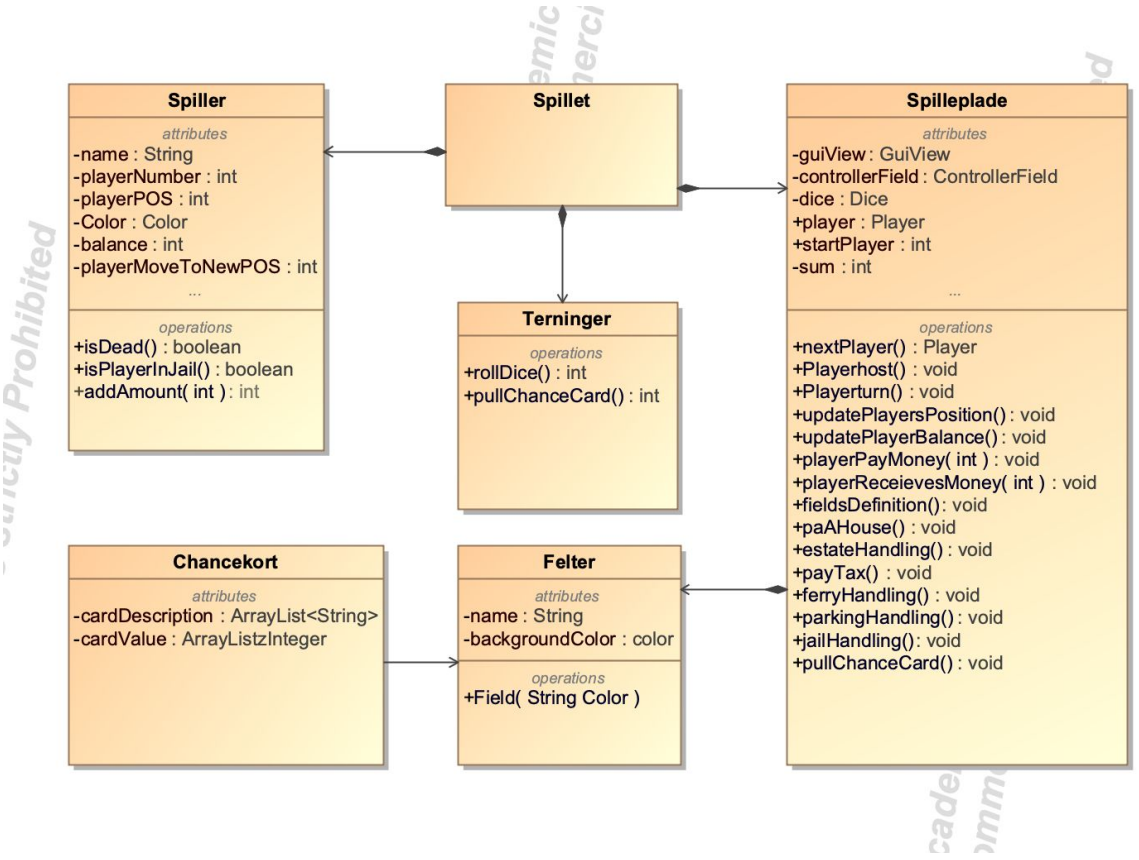
Indirection, Polymorphism, Protected variations og Pure fabrication.

5.2 Design Klassediagram

På baggrund af vores domæne model har vi opstillet et design klassediagram der viser implementering af interface.

Vores design klassediagram består af 5 dele: Spiller, Spilleplade, Terninger, Felter og Chancekort. I hver af delene er der vist de attributter og operationer de indeholder og hvordan de interagere med hinanden.

Den første klasse 'Spiller' indeholder de metoder der er nødvendigt for at oprette spillere, tilføje navne på spillerne, tildele spillebrikkerne, samt opdatere spillernes pengebeholdning. I vores 'Spilleplade' klassen er der implementeret metoder, der er ansvarlige for at ændre spillerens position på felterne og vise de nødvendige tekstoplysninger, når man lander på de forskelliger felter. Klassen 'Terninger' er ansvarlig for at der vises en tilfældig værdi efter man har kastet terninger og metoden, der er ansvarlig for denne handling, er også til stede. Der er 19 chancekort i vores Matador spil, så i klassen 'Chancekort' er der implementeret de metoder der ansvarlige for at der vises den specifikke information og værdi når man lander på et 'Prøv lykken' felt og der trækkes et chancekort. Og til sidst har vi klassen 'Felter' som nedarver fælles egenskaber fra package 'Fields'.



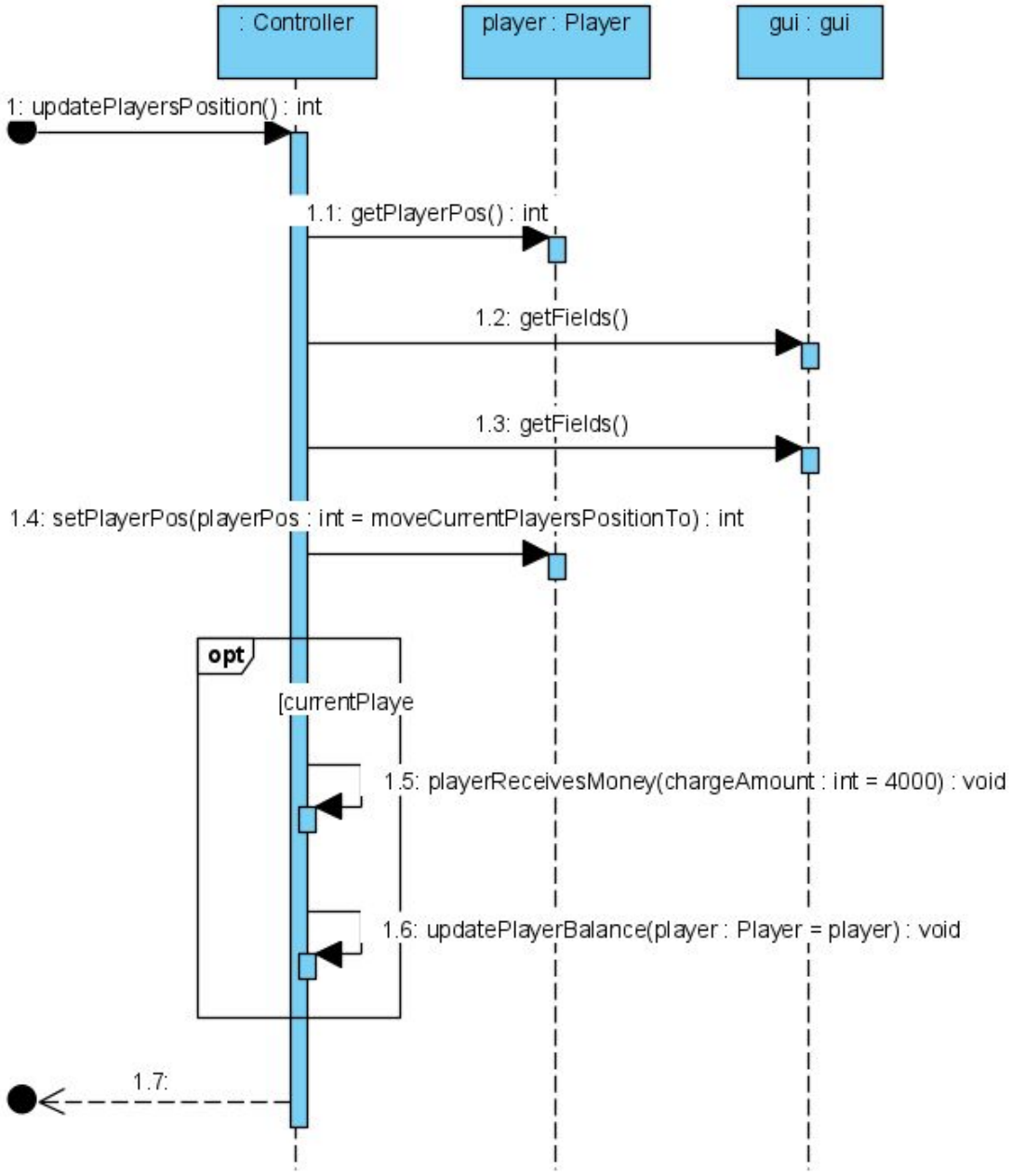
Figur 5: Design klassediagram

5.3 Design Sekvensdiagram

Da vi har rigtig mange metoder, som gøre forskellige ting, har vi besluttet, at det ikke giver mening at lave sekvensdiagrammer til alle metoder. Det gave hellere ikke mening at lave en overordnet fælles sekvensdiagram, fordi det var meget uoverskuelig og intetsigende, da vi har begyndt at lave den.

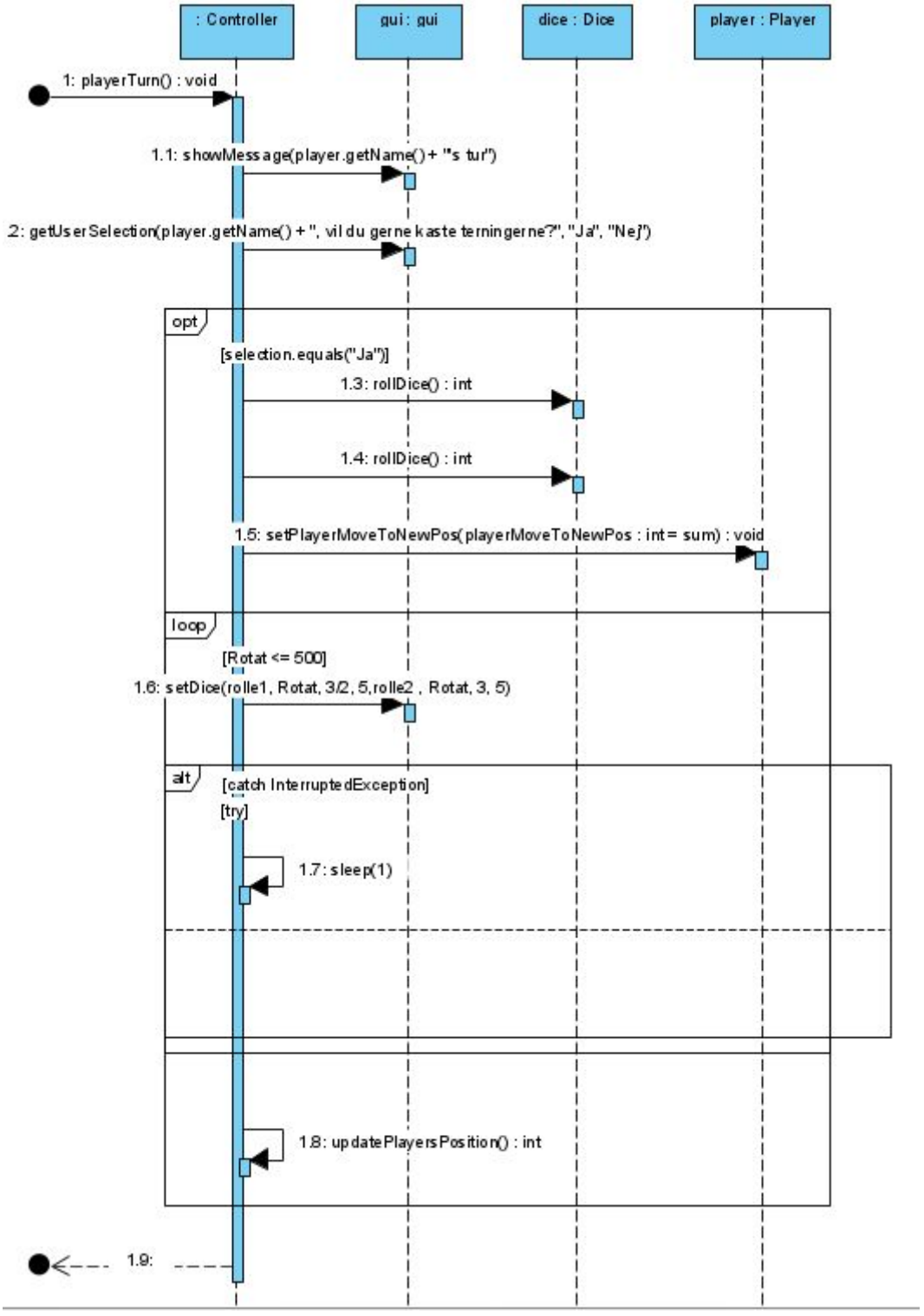
Vi har i stedet for besluttet at lave et par sekvensdiagrammer, som repræsenterer de vigtigste metoder i vores program, så man kan have indblik i hvad der foregår.

5.3.1 Sekvensdiagram til updatePlayerPosition():



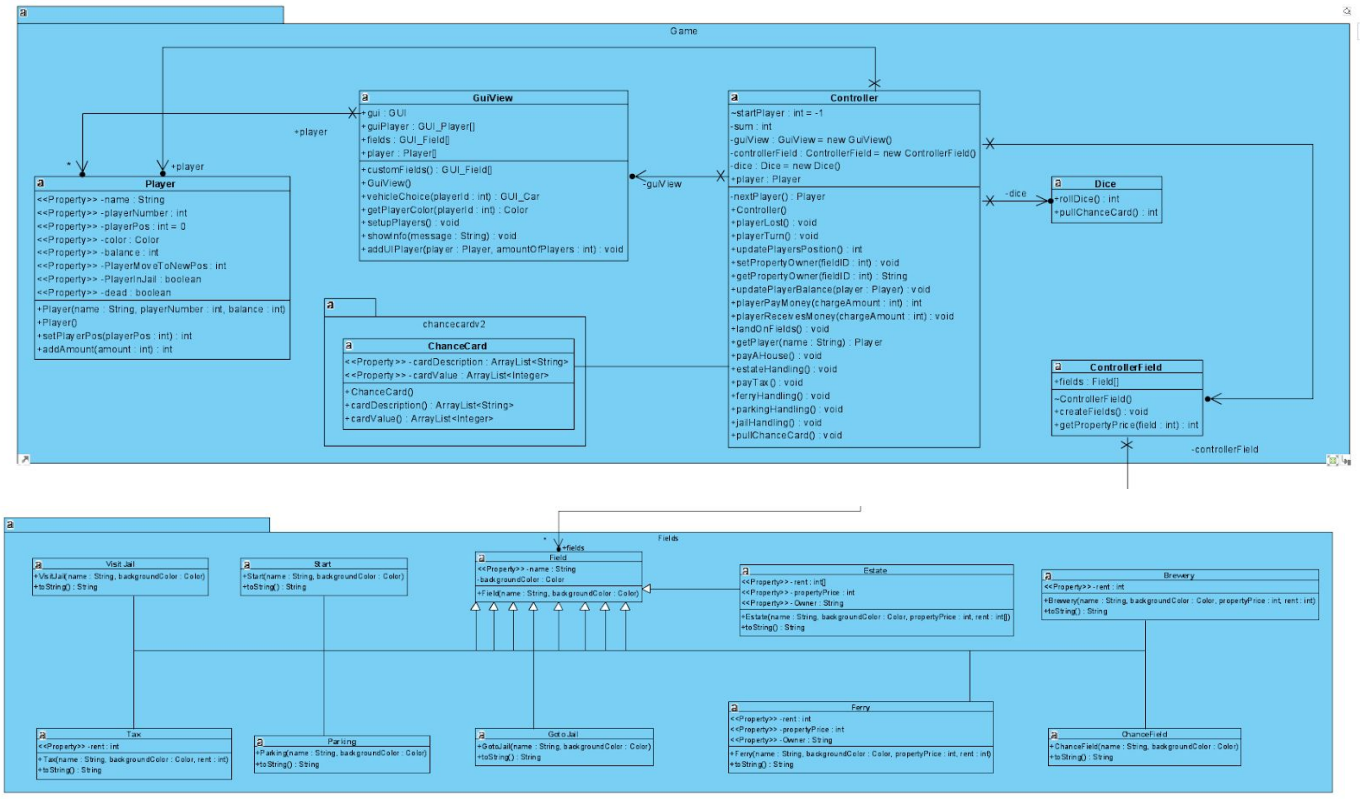
Figur 6: Design sekvensdiagram for updatePlayerPositon():

5.3.2 playerTurn() sekvensdiagram:



Figur 6.1: Design sekvensdiagram for PlayerTurn():

5.4 Klassediagram



Figur 7: Klassediagram.

Det er vores klassediagram efter vi var helt færdige med programmet. Vi har opdelt diagrammet i 2, så man kan se tydeligt hvad der sker og hvad der foregår. I starten troede vi, at vi kan nøjes med at bevare programmet simpelt, overholde god struktur og følge MVC pattern. Det viste sig at være lidt for optimistisk, fordi vores kode voksede hurtigt og da vi ikke kunne være sammen fysisk, kunne vi ikke altid at overholde hinandens kodestruktur. Vi blev ved med at opdatere diagrammet, så den der står ovenover er blevet vores endelig klassediagram.

6. Implementering.

Vores kode er består af 3 hoved packages, der ”chance card” og ”Fields” og ”Game”. Game package indeholder den centrale klasse og den hedder Controller , som styrer systemet , Gui View klasse arbejdes med en GUI som design af User Interface.

1. Game package. Består af

    a. Gui View klassen indeholder GUI der har ansvaret for den grafiske brugerflade der benyttes. GUI klassen bliver her benyttet til at opsætte det virtuelle spillebræt. GUI klassen har altså derfor kun noget at gøre med den grafiske brugerflade. Og indeholder også GUI_Player hvor vi kan opretter player.

    b. Controller klassen: styrer systemet og Controller indeholder logikken bag selve spillet som helhed.

    c. Controller Field klasse : Denne klasse består af en array hvor vi har putter felter værdier og lejer af felter.

    d. Player klassen : Player indeholder player, balance ,position. Denne klasse har ansvaret for spilleren og indeholder vedkommendes pengebeholdning, spillebrik, ejede felter og deres position på brættet.

    e. Dice: indeholder to metoder, som kan lave en klassisk 6 sidet terning eller terning med et vilkårligt antal sider alt. Der slås med terningen ved metoden "roll Dice", som int fra 1 - 6 returnerer et tilfældigt tal korresponderende til en af terningens sider.

2. Chancecard package. Består af

    a. Chance Card klassen: hvor der to Arraylist en er String som returnerer card Description og den anden arrayList Integer som også returnerer card Value.

    b. Ved Random metode sørger for at lagre logikken for hvilken handling skal foretages alt efter hvilket chancekort der trækkes.

3. Fields package. Består af

danner grundlaget for alle Field Actions. Der findes 10 typer af FieldActions der nedarver egenskaberne, står for at kontrollere hvilket type felt spillerne lander på.

a. Field : I denne klasse er blevet oprettet konstruktører og de andre klasser nedarver fra denne klasse.

b. GoToJail

c. Parking

d. Start

e. Tax

f. VisitJail

g. Ferry

h. Estate

i. ChanceField

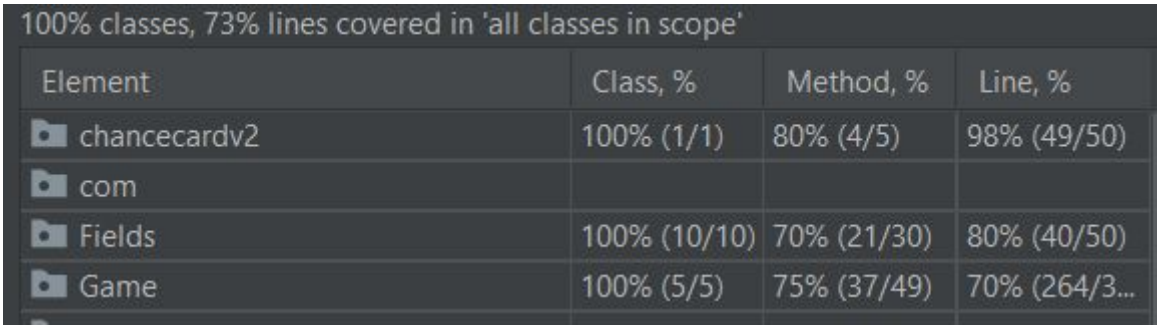
j. Brewery

7. Test.

Herunder vil i kunne se vores forskellige test.

7.1 JUnit test.

Det er vores oprindelig code coverage rapport, hvor man kan se hvor mange linje af kode bliver påvirket, når vi kører programmet:



100% classes, 73% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %
chancecardv2	100% (1/1)	80% (4/5)	98% (49/50)
com			
Fields	100% (10/10)	70% (21/30)	80% (40/50)
Game	100% (5/5)	75% (37/49)	70% (264/3...

Da mange af vores metoder er void, har vi ikke lavet test på alle metoder. Vi har i stedet for har valgt at skrive tests til de mest vigtigste metoder, samt med getters og setters. Vi har også lavet et par test cases på terningekast og lykkekort trækning, så vi er sikker på, at random værdier er altid i den ønskede scope. Vi har valgt at indstille kast på en måde, så den kaster en terning 10.000 gange.
Kodestykket står nederst:

```
void rollDice() {
    for (int i = 0; i < 10000; i ++) {
        int value = dice.rollDice();
        System.out.println(value);
        assertTrue(value >= 1 && value <= 6, "Success");
    }
}
```

Derudover har vi brugt Mockito Framework til at mocke nogle af metoderne, som vi ikke ønskede at de interagere med andre metoder, som ikke er med i testen. For eksempel kan man se en af de mock metoder nedenunder:

```
Controller controller = mock(Controller.class);
Player player = new Player();

@BeforeEach
```

```
void setUpPlayer() {
   player.setName("John");
   player.setBalance(20000);
   player.setPlayerPos(3);
}

@Test
void updatePlayerPositionTest() {

when(controller.updatePlayersPosition()).thenReturn(player.se
tPlayerPos(player.getPlayerPos() + 3));
   assertEquals(player.getPlayerPos(), 6);
}
```

Vi har mocket hele Controller klasse for at kunne kalde på metoder uden at påvirke andre metoder. Test case er også succesfulde, det vil man kunne se, når man kører test cases.

Alle andre test cases ligner dem vi allerede har nu, så man kan tydeligt se mønster i vores måde at lave test cases på.

8. Dokumentation.

8.1 Arraylist

Hvis man ønsker et alternativ til den statiske Java array, kan man i stedet bruge den mere dynamiske Arraylist. Forskellen mellem disse to er et Arraylist er mere fleksibel da man kan ændre størrelsen, hvor at hvis man ønskede at gøre det samme med et Java array, ville det kræve at man oprettede et nyt array i stedet for. Et eksempel på arrayliste fra vores eget program kan ses herunder.

```
1 package chancecardv2;
2
3 import java.util.ArrayList;
4
5 public class ChanceCard {
6
7   private ArrayList<String> cardDescription;
8   private ArrayList<Integer> cardValue;
9
10  public ChanceCard() {
11      this.cardDescription = cardDescription();
12      this.cardValue = cardValue();
13  }
```

Som det kan ses ovenover indeholder 'ChanceCard' klassen 2 arraylister, den ene er alle de Int værdier som de forskellige Chance kort har, fra linje 46 - 59 repræsenterer balance ændringer, hvorimod at at linje 61 - 66 repræsenterer " Ryk x felter frem / tilbage eller ryk til et specifikt felt "

```
43 public ArrayList<Integer> cardValue() {
44   ArrayList<Integer> cardValue = new ArrayList<Integer>();
45
46   cardValue.add(2000);   //0
47   cardValue.add(50);     //1
48   cardValue.add(200);    //2
49   cardValue.add(25);     //3
50   cardValue.add(20);     //4
51   cardValue.add(1);      //5
```

```
52   cardValue.add(50);     //6
53   cardValue.add(108);    //7
54   cardValue.add(100);    //8
55   cardValue.add(-100);   //9
56   cardValue.add(-100);   //10
57   cardValue.add(-10);    //11
58   cardValue.add(-20);    //12
59   cardValue.add(-20);    //13
60
61   cardValue.add(3);      //14
62   cardValue.add(-3);     //15
63   cardValue.add(39);     //16
64   cardValue.add(24);     //17
65   cardValue.add(5);      //18
66   cardValue.add(0);      //19
67
68   return cardValue;
69 }
```

Hvor at den øverste arraylist returnerede Int værdier, returnere den nederste String " Tekst " fra de tilhørende chancekort, som det kan ses i kommentarfeltet er de nummererede så at " //1 " i begge arrayliste passer sammen.

```
15 public ArrayList<String> cardDescription() {
16   ArrayList <String> cardDescription = new ArrayList<String>();
17
18   cardDescription.add("De modtager Matador legatet. Byens bogmester
overdrager dem en chack på 2000kr");                    //0
19   cardDescription.add("Modtag udbytte af deres aktier. Modtag
50kr.");                                                  //1
20   cardDescription.add("Værdien af egen avl fra nyttehaven udgør
200kr, som de modtager af banken.");                    //2
21   cardDescription.add("Grundet på dyrtiden har de fået
gageforhøjelse. Modtag 25kr. ");
//3
22   cardDescription.add("De har solgt deres gamle klude. Modtag
20kr.");                                                  //4
```

```
23   cardDescription.add("De har rettidigt afleveret deres
abonnementskort. Depositum 1kr udbetales dem af banken.");           //5
24   cardDescription.add("Manufakturvarerne er blevet billigere og
bedre, herved sparer de 50kr, som de modtager af banken.");  //6
25   cardDescription.add("Efter auktionen på Assistenshuset, hvor de
havde pantsat deres tøj, modtager de ekstra 108kr.");      //7
26   cardDescription.add("Deres præmieobligation er kommet ud. De
modtager 100kr af banken.");                                    //8
27   cardDescription.add("De har anskaffet et nyt dæk til deres vogn.
Indbetal 100kr.");                                          //9
28   cardDescription.add("De har kørt frem for Fuld Stop. Betal 100kr i
bøde.");                                                   //10
29   cardDescription.add("Betal for vognvask og smøring. 10kr.");
//11
30   cardDescription.add("De har været en tur i udlandet og haft for
mange cigaretter med hjem. Betal 20kr i told.");           //12
31   cardDescription.add("De har måttet vedtage en parkeringsbøde. Betal
20kr til banken.");                                       //13
32
33   cardDescription.add("Ryk tre felter frem");         //14
34   cardDescription.add("Ryk tre felter tilbage");      //15
35   cardDescription.add("Tag ind på Rådhuspladsen");    //16
36   cardDescription.add("Ryk fem til Grønningen");      //17
37   cardDescription.add("Tag med Øresundsbåden");       //18
38   cardDescription.add("Ryk frem til start");          //19
39
40   return cardDescription;
41 }
```

8.2 Arv

Vi vælger at bruge arvfunktionen for at gøre vores arbejde nemmere, når flere klasser har de samme basisfunktioner. Det vil sige at en eller flere klasser alle kan arve disse egenskaber, metoder og attributer fra en fælles klasse men derfra specialisere de sig individuelt.
Klassen som der arves fra kaldes for “ Superklassen “ og klasserne som arver kaldes for “ Subklasser “

Herunder kan vores “ Superklasse “ for fields ses.

```
1 package Fields;
2
3 import java.awt.*;
4
5 public class Field {
6
7   private String name;
8   private Color backgroundColor;
9
10  public Field(String name, Color backgroundColor){
11     this.name = name;
12     this.backgroundColor = backgroundColor;
13  }
14
15  public String getName() {
16      return name;
17  }
18
19
20
21 }
```

Her vises et eksempel på en “ Subklasse “, vi kan se i linje 5 at den “ Extends field “ så det vil sige at den arver fra “ Superklassen “

```
1 package Fields;
2
3 import java.awt.*;
4
5 public class Parking extends Field{
6
7  public Parking(String name, Color backgroundColor) {
8      super(name, backgroundColor);
9  }
10
11  public String toString() {
```

```
12      return "Parking";
13    }
14 }
```

8.3 Krav Gennemgang

8.3.1 Must have.

På nuværende tidspunkt har vi ikke fået implementeret et " Spillet er slut " eller " Vinderen er fundet " funktion, som spillet virker lige nu, kan den sidste spiller, som er tilbage fortsætte i det uendelige.

8.3.2 Should have.

Ekstra kast ved 2 ens, direkte i fængsel ved 3*2 ens, dobbelt husleje hvis man ejer alle ejendomme med samme farve samt opgradering af antal huse / hotel er ikke blevet implementeret.

8.3.3 Could Have.

Det er kun lykkedes os at få implementeret " Træk et chancekort, hvis du lander på et prøv lykken felt " Resten viste sig enten at være for omfattende for os, samt et tidspres.

Krav	Opfyldt	Ikke opfyldt	Krav	Opfyldt	Ikke Opfyldt
M1	X		S1	X	
M2	X		S2		X
M3	X		S3		X
M4	X		S4	X	
M5	X		S4.1	X	
M6	X		S4.2	X	
M7	X		S4.3	X	
M8	X		S5	X	
M9	X		S5.1	X	
M10		X	S5.2	X	

			S5.3	X	
C1	X		S5.4	X	
C2		X	S5.5	X	
C3		X	S5.6	X	
C4		X	S5.7	X	
C5		X	S6		X
C6		X	S7		X
C7		X	S8		X
W1		X			
W2		X			

9. Konklusion.

Vi fik stillet opgaven at udvikle " den fulde version af Matador " med det vi aflevere har vi gjort et ærligt forsøg på dette. Vi må konkludere at opgaven var lidt for stor når vi ser på gruppens medlemmer, vi er 4 personer med absolut ingen programmeringserfaring og en enkelt, som har en vis erfaring. Det har resulteret i en masse forsøg på at implementere og skrive kode, som i sidste ende måtte fuldstændig omskrives for at få et funktionelt program. Mange af vores diagrammer har vi, som følge af dette enten måtte lave om eller fuldstændig vente med at skrive indtil at vi var " Færdige " med skrive koden.

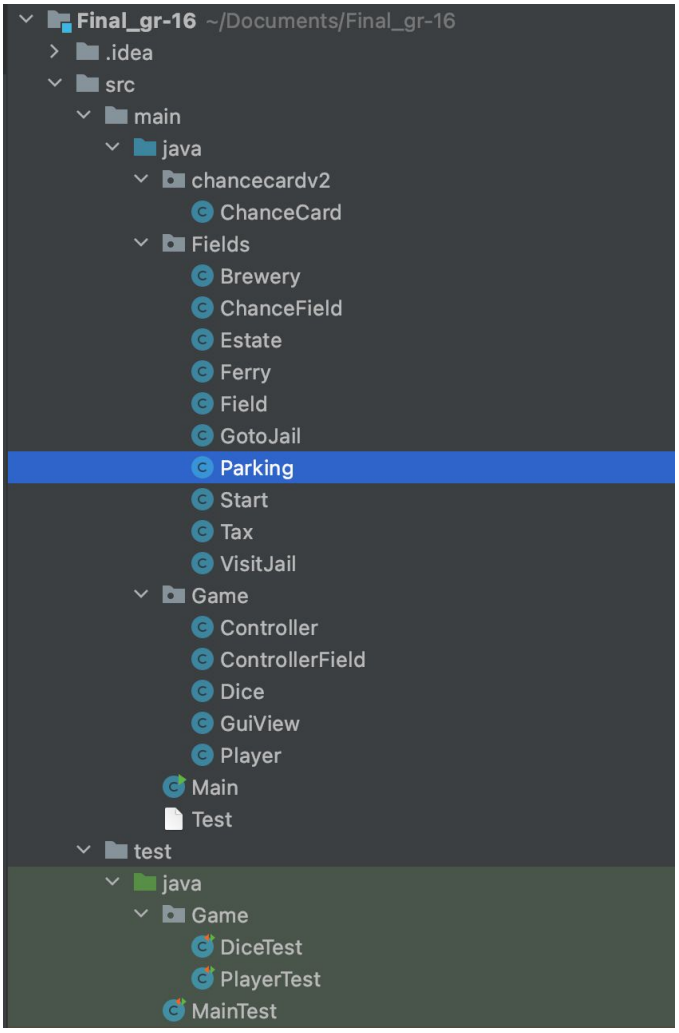
I sidste end, er vi dog endt ud med et funktionelt program, som faktisk virker og er bygget op, som det fysiske Matador, selvfølgelig dog med de mangler som er beskrevet i 8.3 krav gennemgangen.

10. Bilag

10.1 Github Repo

https://github.com/salim963/Final_gr-16.git

10.2 Kildekode indholdsfortegnelse.



Figur : Model + Controller oversigt.

10.3 Felt oversigt.

Felt	Type	pris	Leje af grund med huser eller hotel	Note
0	Start	*	Grund, 1 hus, 2 huser ,3huser, 4huser, hotel.	*
1	Street	600 kr	50, 250, 750, 2250, 4000, 6000	*
2	Chance	?	?	?
3	Street	600 kr	50, 250, 750, 2250, 4000, 6000	*
4	Tax	*	*	Betale 10% af indkomst eller 4000
5	Shipping	2000 kr	500, 1000, 2000, 4000	Scandlines
6	Street	1000 kr	100, 600, 1800, 5400, 8000, 11000	*
7	Chance	?	?	?
8	Street	1000 kr	100, 600, 1800, 5400, 8000, 11000	*
9	Street	1200 kr	150, 800, 2000, 6000, 9000, 12000	*
10	Jail	*	*	Du er bare på besøg
11	Street	1400 kr	200, 1000, 3000, 9000, 12500, 15000	*
12	Brewery	1500 kr	*	Man betaler 100 gange så meget som øjne viser
13	Street	1400 kr	200, 1000, 3000, 9000, 12500, 15000	*
14	Street	1600 kr	250, 1250, 3750, 10000, 14000, 18000	*
15	Shipping	2000 kr	500, 1000, 2000, 4000	Mols-Linien
16	Street	1800 kr	300, 1400, 4000, 11000, 15000, 19000	*
17	Chance	?	?	?
18	Street	1800 kr	300, 1400, 4000, 11000, 15000, 19000	*
19	Street	2000 kr	350, 1600, 4400, 12000, 16000, 20000	*
20	Parkering	/	/	/
21	Street	2200 kr	350, 1800, 5000, 14000, 17500, 21000	*
22	Chance	?	?	?
23	Street	2200 kr	350, 1800, 5000, 14000, 17500, 21000	*
24	Street	2400 kr	400, 2000, 6000, 15000, 18500, 22000	*
25	Shipping	2000 kr	500, 1000, 2000, 4000	Scandlines
26	Street	2600 kr	450, 2200, 6600, 16000, 19500, 23000	*
27	Street	2600 kr	450, 2200, 6600, 16000, 19500, 23000	*
28	Brewery	1500 kr	*	Man betaler 100 gange så meget som øjne viser
29	Street	2800 kr	500, 2400, 7200, 17000, 20500, 24000	*
30	Jail	*	*	Gå til fængsel
31	Street	3000 kr	550, 2600, 7800, 18000, 22000, 25000	*
32	Street	3000 kr	550, 2600, 7800, 18000, 22000, 25000	*
33	Chance	?	?	?
34	Street	3200 kr	600, 3000, 9000, 20000, 24000, 28000	*
35	Shipping	2000 kr	500, 1000, 2000, 4000	Scandlines
36	Chance	?	?	?
37	Street	3500 kr	700, 3500, 10000, 22000, 26000, 30000	*
38	Tax	*	*	Du skal betale 100 til almenvellet
39	Street	4000 kr	1000, 4000, 12000, 28000, 34000, 40000	*

Matador gr-16

Figur : Felt oversigt.