CDIO Final

Projektopgave efterår 2015 – jan 2016
02312-14 Indledende programmering

Projekt navn: CDIO-Final
Gruppe nr: 39
Afleveringsfrist: 18 januar, 2016, kl. 12.00

Denne rapport afleveres via Campusnet (der skrives ikke under)

Denne rapport indeholder 26 sider incl. denne side.

Forfattere:

Korsgaard, Michael Nicolaj
s150348

Buur, Frederik Klibo
s133045

Larsen, Oliver Sonderegger
s147302
Kontaktperson / Projektleder (fag)

Mehlsen, Joachim Skov
s141207

Timeregnskab

CDIO Final							
Time-regnskab							
Dato	Deltager	Design	Impl.	Test	Dok.	Andet	I alt
04/01 - 2016	Michael N. Korsgaard	0	0	0	2	0	2
	Frederik A. K. Buur	0	0	0	2	0	2
	Oliver S. Larsen	0	0	0	2	0	2
05/01 - 2016	Michael N. Korsgaard	1	0	0	4	0	5
	Frederik A. K. Buur	3,5	0	0	1,5	0	5
	Oliver S. Larsen	3,5	0	0	1,5	0	5
	Joachim S. T. Mehlsen	3,5	0	0	1,5	0	5
06/01 - 2016	Michael N. Korsgaard	4	0,5	0	0,5	0	5
	Frederik A. K. Buur	1	4	0	0	0	5
	Oliver S. Larsen	1	4	0	0	0	5
	Joachim S. T. Mehlsen	2,5	2	0	0,5	0	5
07/01 - 2016	Frederik A. K. Buur	0	6	0	0	0	6
	Oliver S. Larsen	0	6	0	0	0	6
	Joachim S. T. Mehlsen	0	6	0	0	0	6
08/01 - 2016	Michael N. Korsgaard	1	4	0	1	0	6
	Frederik A. K. Buur	0	5	0	0	0	5
	Oliver S. Larsen	0	5	0	0	0	5
	Joachim S. T. Mehlsen	0	4,5	0	0	0	4,5
11/01 - 2016	Michael N. Korsgaard	0	0	1	0	0	1
	Frederik A. K. Buur	0	5,5	0	0	0	5,5
	Oliver S. Larsen	0	5,5	0	0	0	5,5
	Joachim S. T. Mehlsen	0	0	2	0	0	2
12/01 - 2016	Michael N. Korsgaard	0	0	2	0	0	2
	Frederik A. K. Buur	0	5	0	0	0	5

	Oliver S. Larsen	0	5	0	0	0	5
	Joachim S. T. Mehlsen	0	5	0	0	0	5
13/01 - 2016	Michael N. Korsgaard	0	0	6	0	0	6
	Frederik A. K. Buur	0	4	3	0	0	7
	Oliver S. Larsen	0	4	3	0	0	7
	Joachim S. T. Mehlsen	0	4	3	0	0	7
14/01 - 2016	Michael N. Korsgaard	0	0	3	3	0	6
	Frederik A. K. Buur	0	6	0	0	0	6
	Oliver S. Larsen	0	6	0	0	0	6
	Joachim S. T. Mehlsen	0	6	0	0	0	6
15/01 - 2016	Michael N. Korsgaard	0	0	0	7	0	7
	Frederik A. K. Buur	0	0	0	7	0	7
	Oliver S. Larsen	0	0	0	7	0	7
	Joachim S. T. Mehlsen	0	0	0	7	0	7
	Total	21	103	23	47,5	0	194,5

Indholdsfortegnelse

1.0 Indledning

CDIO final er det sidste projekt vi har gennemarbejdet i faget Indledende Programmering. Vi har efter 13 ugers undervisning fået en basal kendskab til programmering efter alle delopgaverne, og er nu blevet bedt om at udvikle en simulering af det danske brætspil matador. Vi har fået stille som opgave at implementere så mange af de originale regler som muligt.

2.0 Resumé (Abstract)

Vores opgave er blevet stillet af vores "kunde", som ønsker en java applikation der minder om brætspillet matador. Kunden har pointeret, at det er væsentligt vigtigere at vores applikation har nogle "hovedpunkter" implementeret som virker, i stedet for at have implementeret en masse, men som ikke virker. Herved skal vi inddrage kode som vi har arbejdet med/lært fra de foregående opgaver og projekter.

3.0 Hovedafsnit

3.1 Analyse

3.1.1 Kravspecifikation

Vores krav er skrevet ud fra en analyse, som vi har foretaget af manualen fra brætspillet Matador fra mærket Alga, som udkom i 2007.
Vi har valgt at inddele vores krav i følgende kategorier, som bliver identificeret foran kravet med understående bogstaver:

F = Funktionelle krav NF = None-funktionelle krav

M = Must have S = Should have C = Could have W = Would be nice to have

Krav markeret med rød tekst ***som denne*** *er krav der ikke nåede at blive gennemført i det endelige produkt.*

3.1.1.1 Funktionelle krav

- **FM1.** Spillet skal slutte, når der kun er 1 spiller tilbage.
- **FM2.** Der skal være en bræt med 40 felter.
- **FM3.** Der skal laves lykke kort til felterne "Prøv Lykken".
- **FM4.** Man skal kunne købe felter.
- **FM5.** Man skal kunne ryge i fængsel.

- **FM6.** Man skal kunne komme ud af fængslet, ved at kaste 2 ens terninger, mens man er i fængsel.
- **FM7.** Ejer man alle grunde med samme farve, skal man kunne bygge huse/hotel på de grunde.
- **FM8.** Kan en spiller ikke komme op på et beløb som er <= 0, gælder han som ikke at have flere penge, går fallit, og bliver smidt ud af spillet.
- **FM9.** Spillet skal være turn-based.
- **FM10.** Der skal være 2 terninger.
- **FM11.** Man rykker rundt på brættet med brikker, der repræsentere dem, og rykker et antal felter bestemt af antallet af øjne de slår med et sæt terninger.
- **FM12.** Feltet ”i fængsel” skal ikke smide en i fængsel, hvis man lander på det.
- **FM13.** “De fængsels” feltet skal smide en i fængsel.
- **FM14.** Antallet af spillere skal være 3-6 deltagere.

Grundet at spillet skal foregår foran en computer, har vi valgt at sænke minimum spillere til 2 spillere, for at sørge for at spillet kan appellere til flere personer.

- **FM15.** Man skal kunne bygge hus og hotel.

- **FS1.** Alle spillere gives 30.000 kr. i spillet ved spillets start.
- **FS2.** Spillerne starter på start feltet.

Vi vælger at spillerne ikke skal starte på startfeltet, men at det første “skridt” skal ske på startfeltet, for at gøre det muligt for spillere at kunne lande på rødovrevej i løbet af første runde, da det er det første felt som der er muligt at købe.

- **FS3.** Hver gang start bliver passeret, skal spilleren der passerer gives 4.000 kr. af banken.
- **FS4.** Når et lykke-kort er brugt, skal det placeres nederst i bunken af lykkekort.

Vi vælger at i stedet for at kortet bliver lagt nederst, skal kortet lægges ind igen og dækket blandes, for at sikre at spillerne ikke på et punkt i spillet blive i stand til at forudse hvad de kan trække igen.

- **FS5.** Hvis man kommer ud af fængslet ved at slå 2 ens, rykker man øjeblikkeligt det antal felter, som man slog for at komme ud.
- **FS6.** Man skal kunne komme ud af fængslet, ved at betale 1.000 kr. før man prøver at slå terninger.
- **FS7.** Hvis man har været i fængslet i 3 runder, uden at komme ud, så efter det 3. terningeslag skal spilleren øjeblikkeligt betale 1.000 kr., og kommer så ud af fængslet, og flytter det antal øjne, som terningerne fra det 3. slag viste.
- **FS8.** Man indkassere ikke 4.000 kr. for at passere start feltet når man rykker fra “de fængsles” feltet til “i fængsel”.

- **FS9.** Feltet “Indkomstskatten” skal give spilleren muligheden for at vælge mellem at betale 4.000 kr. eller 10 % af hans samlede formue.
- **FS10.** Spillere må ikke låne penge indbyrdes.

- **FC1.** Man får et ekstra kast / ekstra tur, hvis terningerne han slog for at rykke frem er ens i forhold til antallet af øjne de viser.
- **FC2.** En spiller ryger i fængsel, hvis han i sin ekstratur slår 2 ens, og i sin 2. ekstratur igen slår 2 ens (dvs. hvis 3. slag i turen også viser 2 ens, ryger man i fængsel).
- **FC3.** Man skal kunne komme ud af fængslet, ved at bruge et “kom ud af fængslet” lykkekort.
- **FC4.** Ejer man alle grundene med samme farve, fordobles lejen på grundene, hvis man ikke har hus eller hotel på grunden.
- **FC5.** Banken skal kunne købe huse tilbage, hvis ejeren ønsker at sælge dem, til den halve pris af husenes købspris.
- **FC6.** Man skal kun kunne pantsætte ubebyggede grunde til banken.
- **FC7.** Der skal bygges jævnt på grunde hvor der kan bygges huse (Det vil sige, at alle grunde af samme farve skal have 1 hus, før man må bygge 2 huse på et af dem, og så videre).
- **FC8.** Spilleren skal vælge betalingsmetoden for feltet “Indkomstskatten”, før han får oplyst sin samlede formue.

- **FW1.** Indbyrdes handel med en ubebyggede grund (dvs. intet hus/hotel på grunden) mellem spillere.
- **FW2.** Auto-spil mulighed.

3.1.1.2 Ikke-funktionelle krav

Performance Begrænsninger:

- **NF1.** Spillerne skal blive enige om, hvem der starter.
- **NF2.** Programmet skal kunne kører på en af DTU’s maskiner.

Projekt Begrænsninger:

- Projektet samt rapport skal være færdig d. 18/1 - 2016 kl. 12:00.
- Projektet skal kodes i programmet: Java - Eclipse.

3.1.1.3 Spørgsmål/svar

Andre krav-spørgsmål ud fra manualen til et matador-spil er (refereres til som Main (nr.)):

1. Skal "prøv lykken" kortene være de samme som der indgår i et almindeligt matador-spil.

    Hvis nej

    a. Skal der så være de samme typer af kort.
    b. Må der laves nye typer kort.

Nej, vi skal prøve at holde det tæt på et almindeligt matadorspil, men det er dog tilladt for os at lave vores egne kort. Vi har valgt at holde os til de kort der indgår i spillet, uden at tilføje nye typer.

2. Skal der være det samme antal "prøv lykken" kort som der er i et almindeligt matador-spil.

Nej, det behøves der ikke, men jo flere jo bedre. Vi har tilføjet en del lykke kort, men dog ikke alle fra det almindelige matador.

3. Til krav FS3, gælder det når man lander på Start-feltet, eller først når man er kommet forbi startfeltet (til feltet efter start eller videre)?

Begge dele kan accepteres, derudfra har vi valgt at gøre det sådan, at man får når man når startfeltet, om man så lander på det eller passere det, men ikke begge samtidig.

4. Til krav FS9, hvilken værdi tælles bygninger som at have? Det de blev købt for, eller det de kan sælges for?

Vi har valgt at det er den værdi de blev købt for.

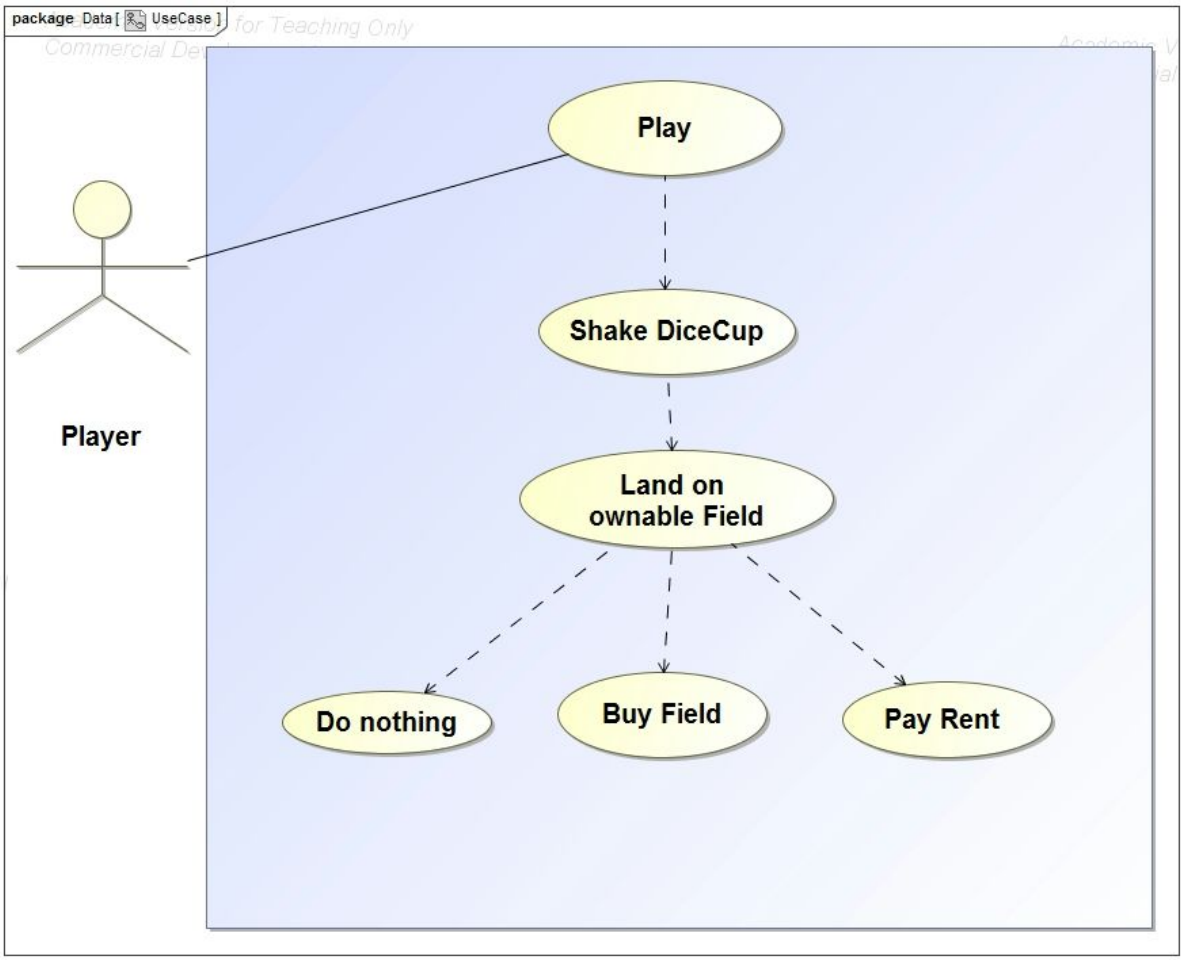
5. Til krav FS9, hvilken værdi tælles den trykte pris af grunden for? Det den blev købt for, eller det som den koster at leje, eller pantsætnings-værdien?

Vi har valgt at det er den værdi de blev købt for.

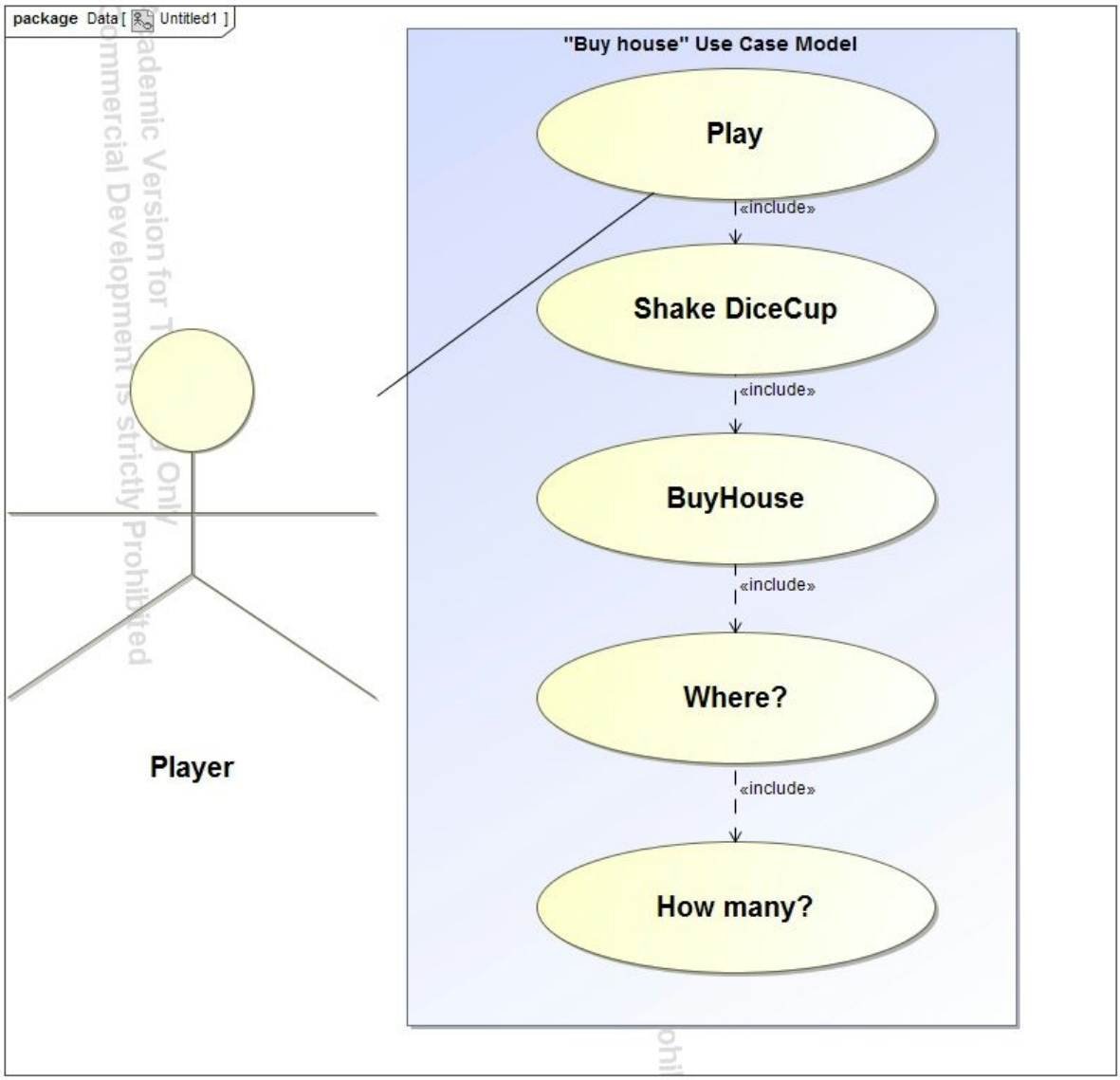
6. Skal lykke-kortene blandes inden et spil, så rækkefølgen er forskellige fra spil til spil?

Vores metode kommer til at tage et tilfældigt kort fra bunken, og bruger det, så rækkefølgen for dækket ikke har en betydning. Derfor behøves det ikke blandes.

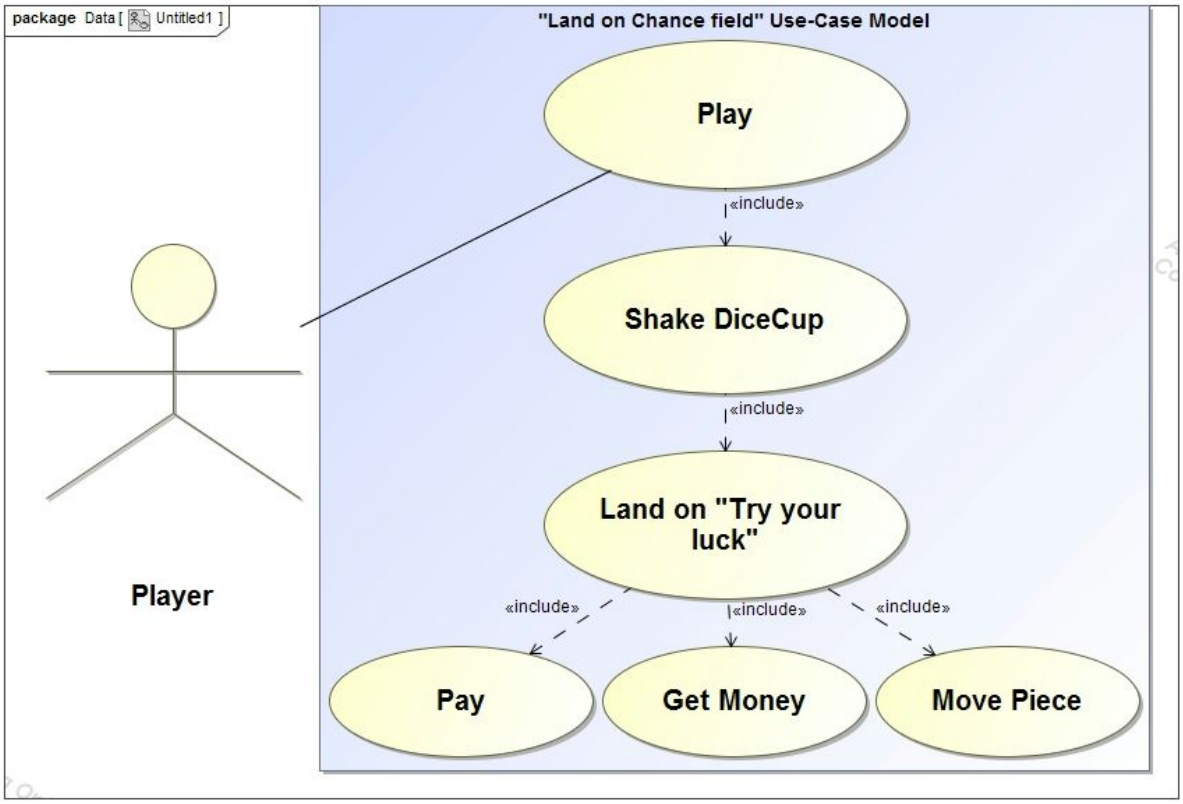
3.1.2 Use-Case Diagrammer



Figur 3.1.2.1: Use-case diagram der viser, hvad der står, når en player lander på et “ownable” felt.



Figur 3.1.2.2: Use-case diagram som viser, hvad spilleren kommer igennem, hvis han gerne vil købe et hus/huse til sin serie.



Figur 3.1.2.3: Her ses et diagram over scenariet “land on chance field”. Dvs. denne use-case tager udgangspunkt i, hvad der sker når man lander på “prøv lykken” feltet.

3.1.3 Use-case beskrivelser

Use Case: Land on ownable field
ID: 1
Beskrivelse: Spilleren lander på et felt der kan ejes
Primær Aktør: Player - (spiller)
Precondition: Spilleren har raflet og rykket sin brik
Main Flow: 1. Spilleren lander på et ejende Felt 2. Spilleren køber feltetFeltet(Alt1, Alt 2) 3. Turen gives videre
Postcondition: Spillerens brik er flyttet
Alternative flows:
Alt1: 1. Spiller har ikke penge til/vil ikke købe feltet. 2. Turen gives videre

Alt 2
1. Spiller lander på et felt der allerede er ejet.
2. Spiller betaler alt efter, hvad lejen bliver
3. Turen gives videre

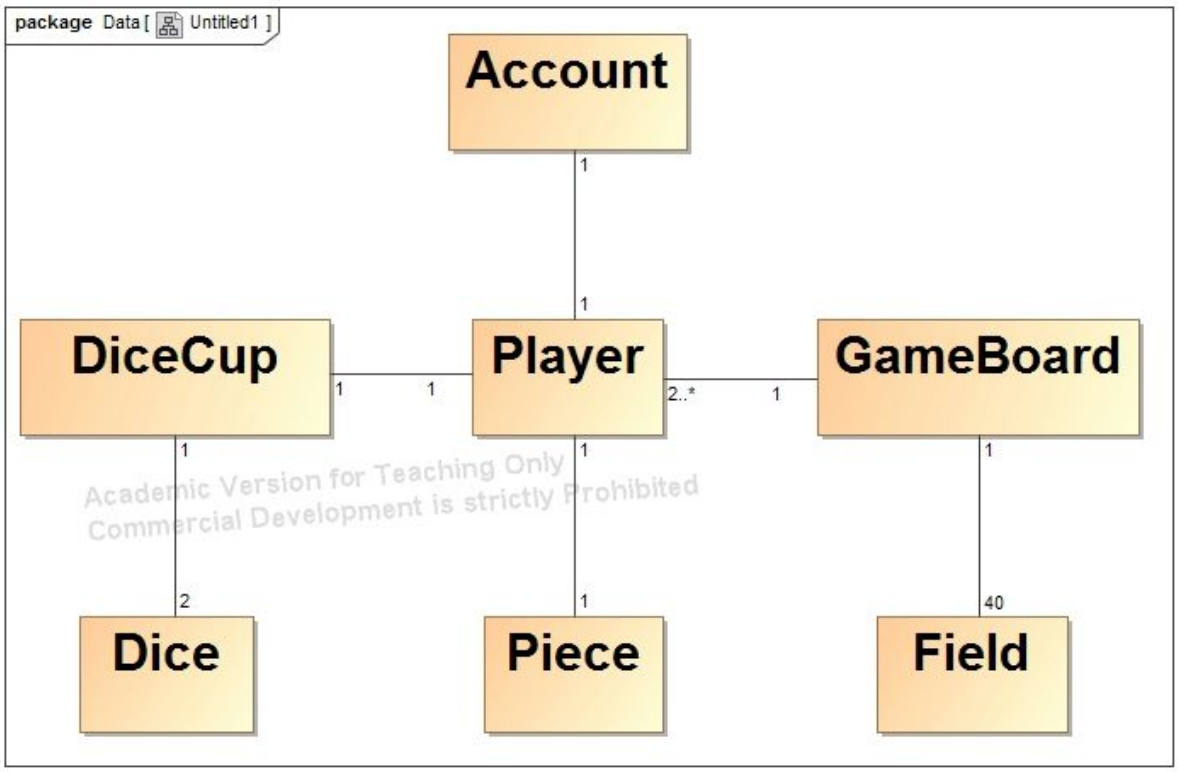
Alt 3
1. Spilleren ejer allerede feltet
2. Spilleren kan vælge at købe et hus/hotel
3. Turen gives videre

Use Case: BuyHouse
ID: 2
Beskrivelse: Spilleren ønsker at købe huse på en af sine grunde.
Primær Aktør: Player - (spiller)
Precondition: Spilleren har raflet og rykket sin brik
Main Flow: 1. Spilleren lander på et houseable felt. 2. Hvis spilleren ejer en serie af grunde, dvs. har mulighed for at købe huse, vil systemet spørge om spilleren vil købe huse. 3. Når spilleren har klikket “yes”, spørger systemet om hvilket felt det skal være. 4. Herefter vil systemet spørge om, hvor mange huse spilleren vil købe. 5. Når det er gjort, vil systemet igen spørge om spilleren vil købe flere huse.
Postcondition: Spillerens ønske af antal huse bliver betalt og placeret det korrekte sted.
Alternative flows:
Alt1: 1. Spiller har ikke en serie. 2. Turen gives videre. Alt 2 1. Spiller vælger et felt som spilleren ikke ejer. 2. Fejlbesked og systemet spørger igen om spilleren vil købe huse. Alt 3: 1. Spilleren vælger at købe flere huse end det maksimalt er tilladt. 2. Fejlbesked og systemet spørger igen om spilleren vil købe huse.

Use Case: Land on chance field
ID: 3
Beskrivelse: Spilleren lander på et “prøv lykken” felt.

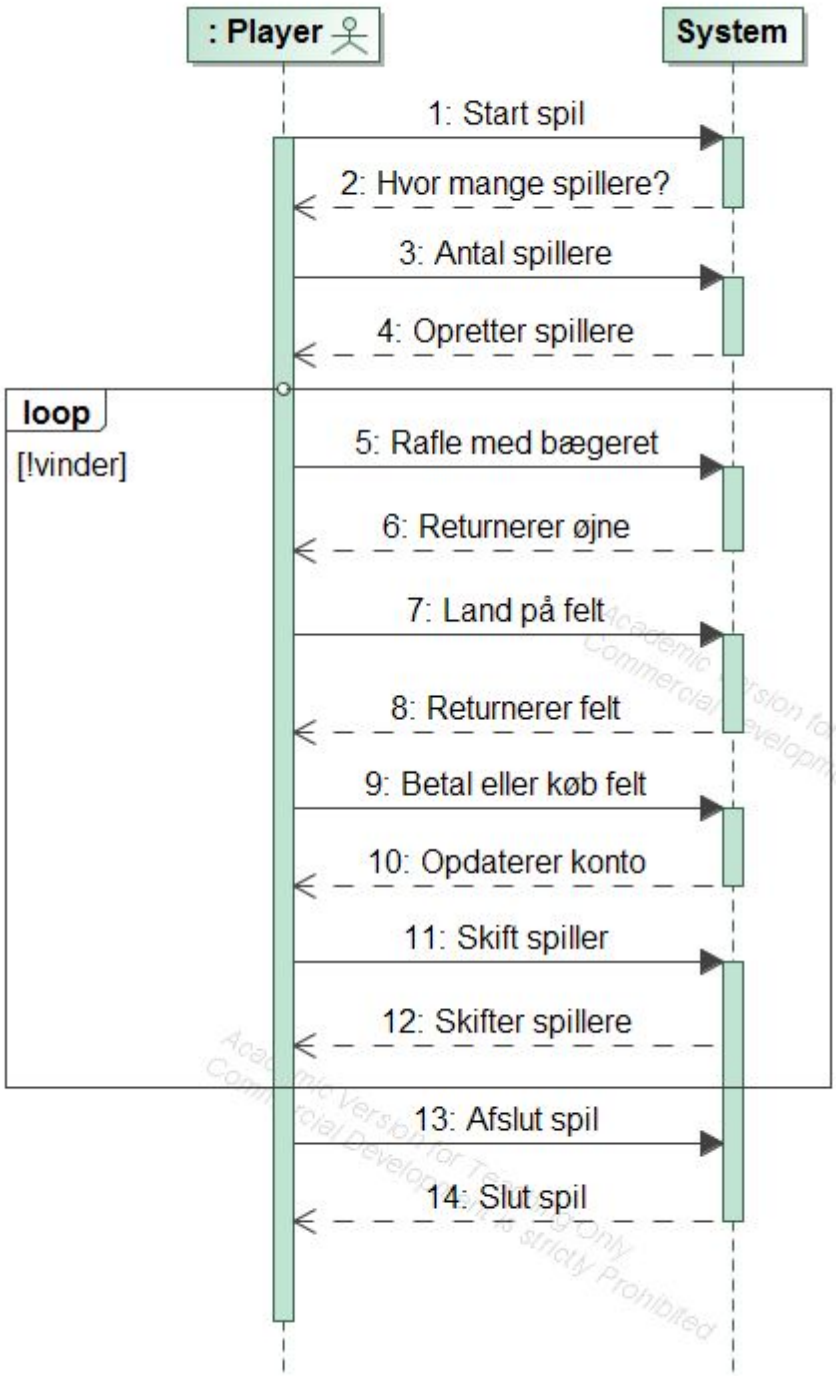
Primær Aktør: Player - (spiller)
Precondition: Spilleren har raflet og er landet på et “prøv lykken” felt.
Main Flow: 1. Spilleren lander på et “prøv lykken” felt. 2. Spilleren trækker et random kort fra bunken. 3. Alt ud fra kortets beskrivelse bliver udført. 4. Turen går videre.
Postcondition: Spilleren har enten fået penge, betalt penge, rykket nogle felter frem eller tilbage, eller spilleren bliver flyttet til et bestemt felt og landOnField bliver kørt.
Alternative flows:

3.1.4 Domænemodel



Figur 3.1.4.1: Ud fra et overordnet visuelt syn på et matadorbrætspil har vi udformet denne domænemodel. Dette giver os et overblik over, hvad vores java applikation i store træk skal indeholde.

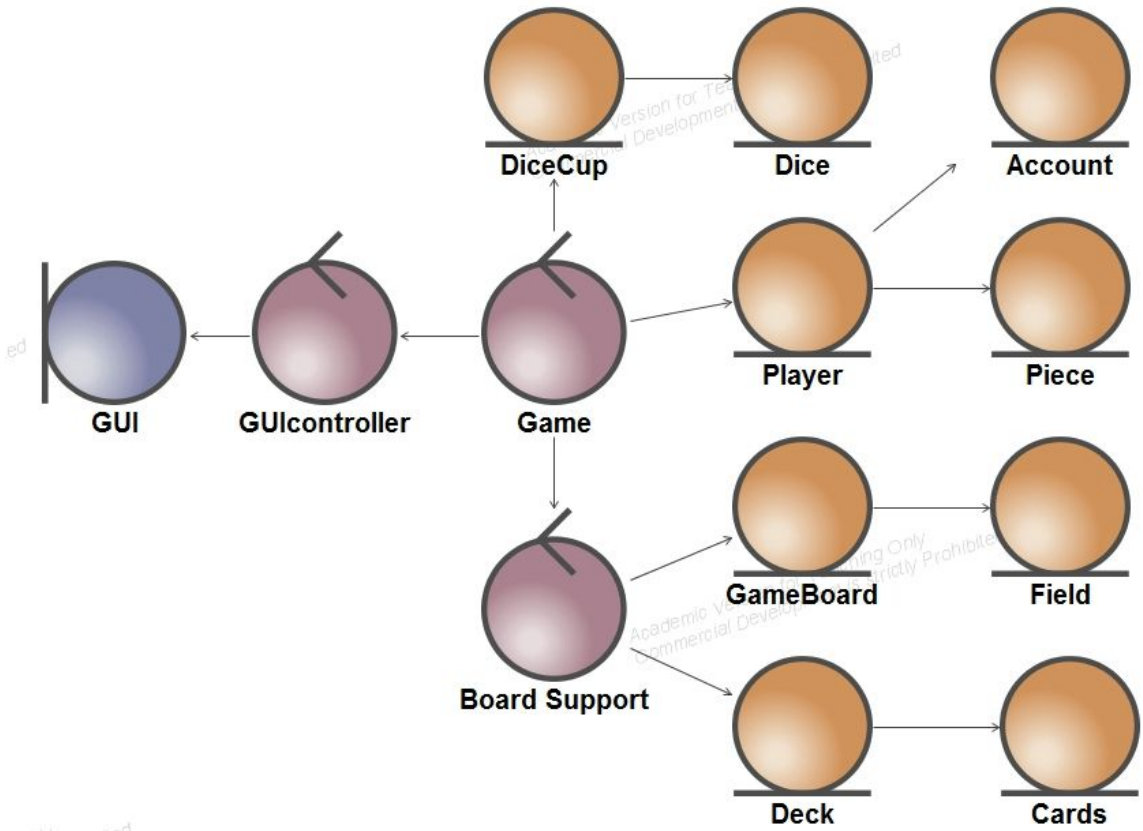
3.1.5 System sekvens diagrammer



Figur 3.1.5.1:System sekvens diagram over usecasen “land on ownable field”. Vi får her et blik på, hvilke spørgsmål systemet skal stille aktøren og hvad systemet skal returnere. Her ses hvordan en spiller integrere med systemet. Dette er vist med associations pile, som går frem og tilbage mellem aktøren og systemet. De fuldt optegnet pile viser, at det er spilleren der skal gøre en handling, og de stiplede viser systemets respons.

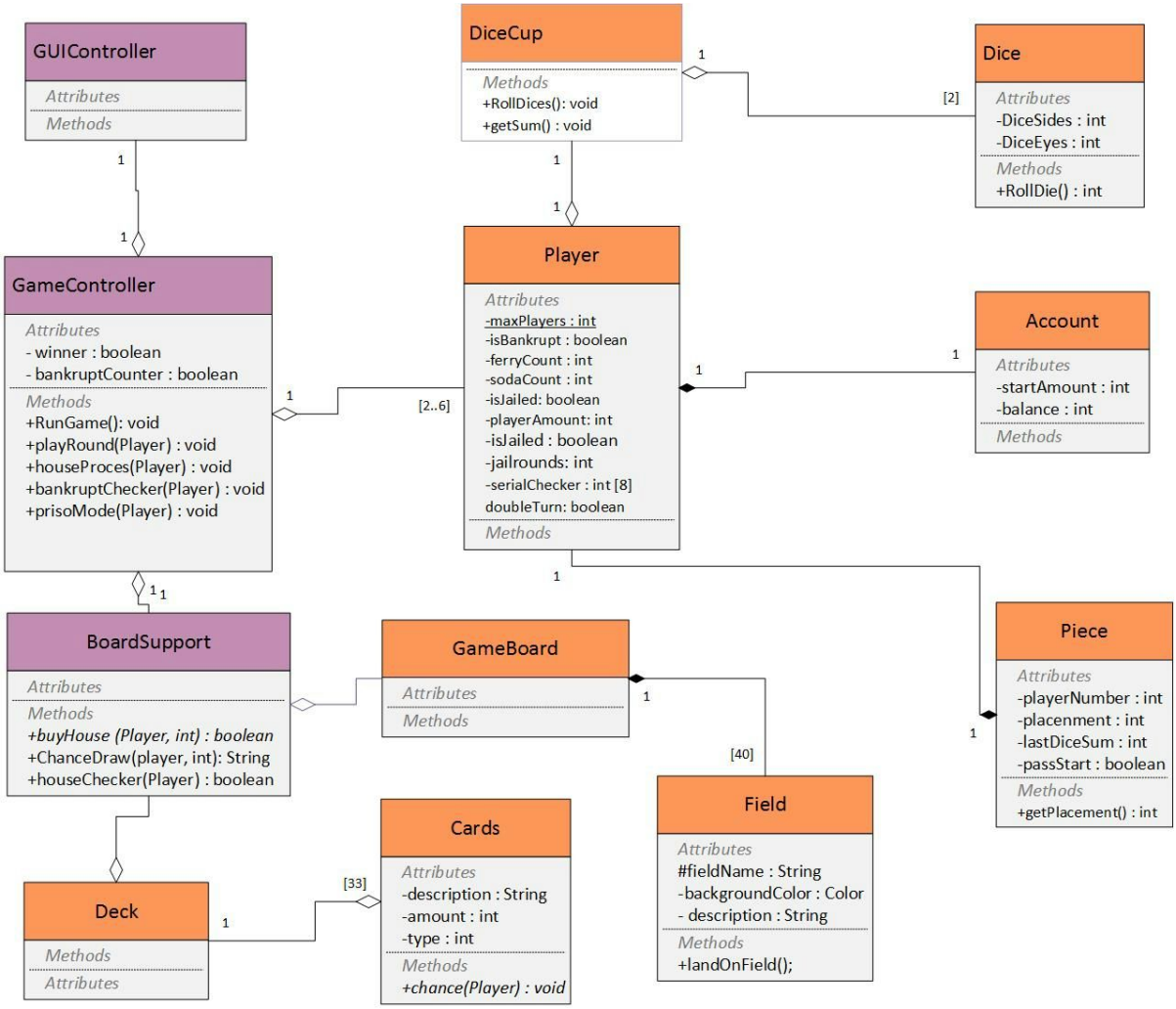
3.2 Design

3.2.1 BCE model

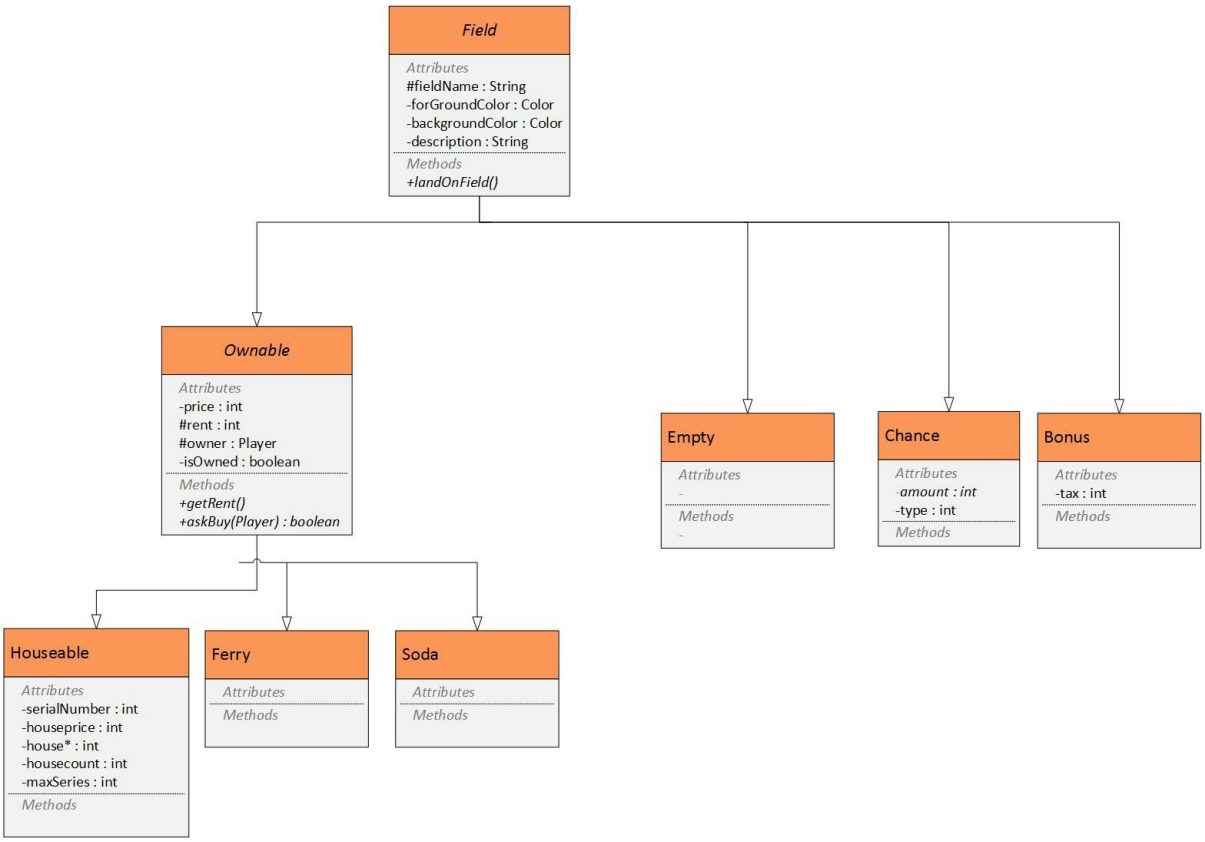


Figur 3.2.1.1: BCE model der fremviser klasse-fordelingen i vores program, og viser hvordan kommunikation generalt kører rundt. Boundary er interaktionen mellem systemet og brugeren, controller fordeler arbejde rundt i systemet ud fra det input boundary modtager fra aktøren. entiteterne indeholder data, samt sætter rammer for programmet.

3.2.2 Design klassediagram



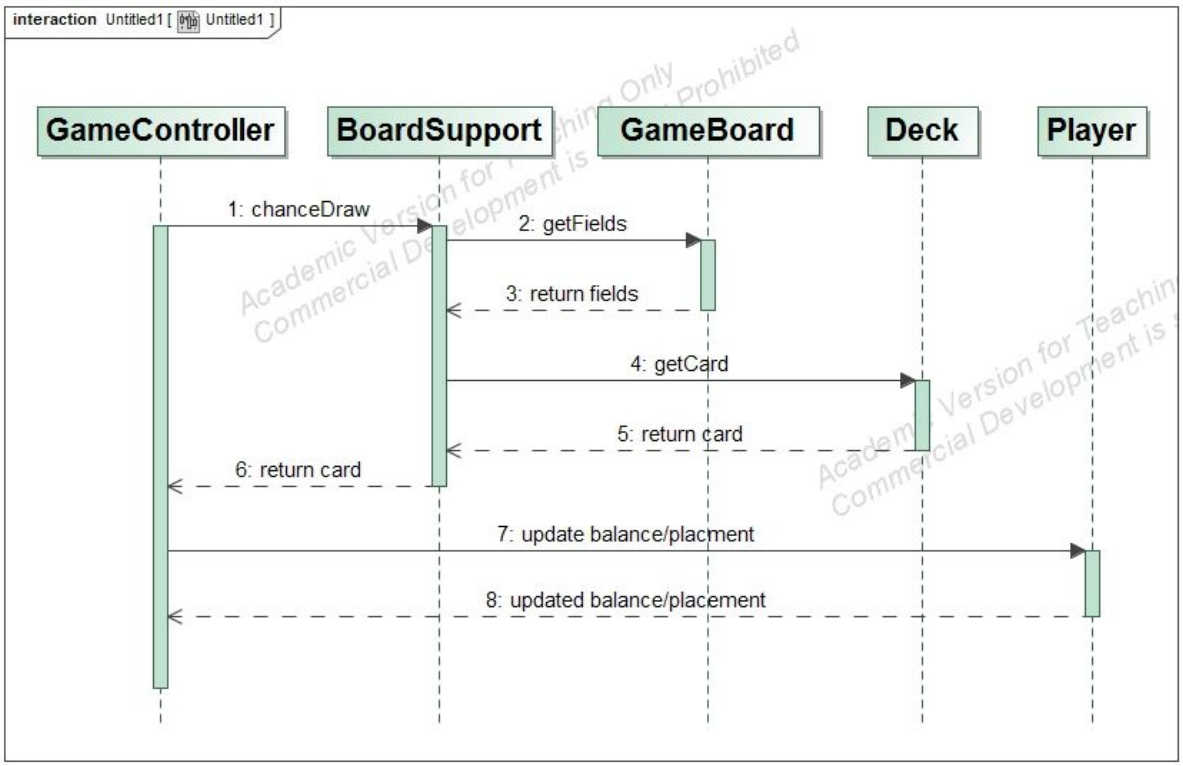
Figur: 3.2.2.1



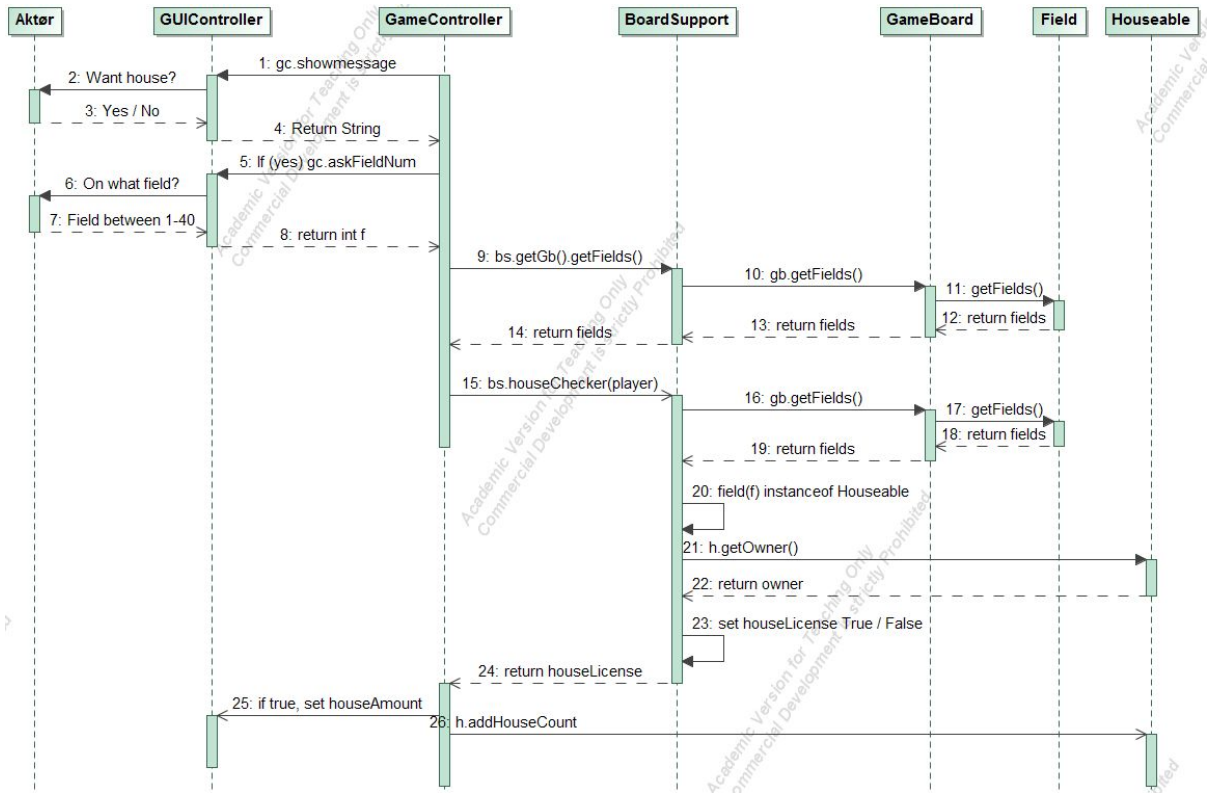
Figur: 3.2.2.2

Disse to design klasse diagrammer viser de forskellige controllere og entiteter der bliver brugt i applikationen. Her kan man se hver klasses attributter samt metoder, der viser relationerne mellem de forskellige klasser. Nummer to diagram viser alle felterne i uddybning af Field fra figur 3.2.2.1.

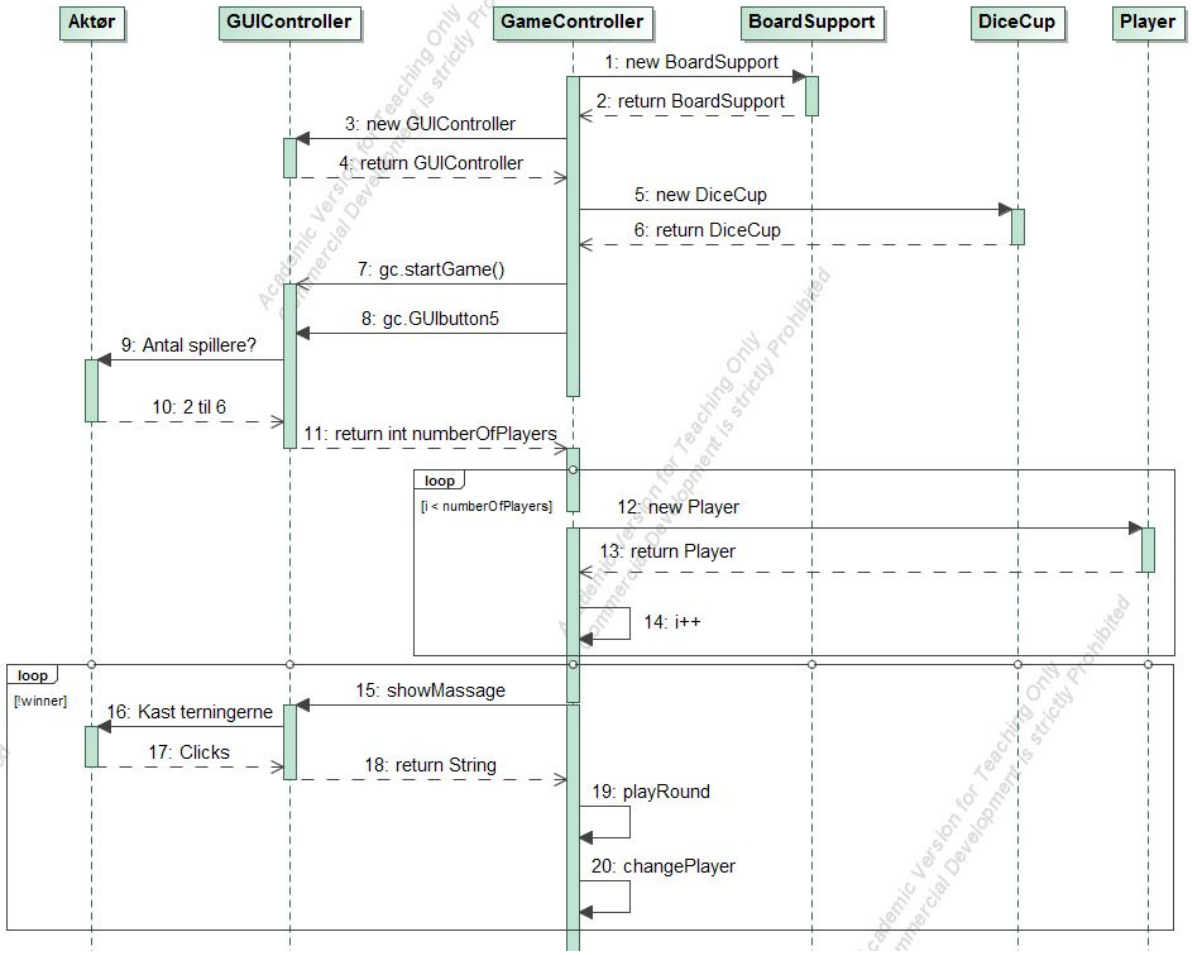
3.2.3 Design sekvensdiagram



Viser hvilke metoder der bliver kaldt, samt hvilke klasser det taler sammen når der skal trækkes et lykkekort.



Viser hvordan metoden houseProcess i GameController virker, som bliver kaldt i løbet af playRound.



Viser hvordan main metoden RunGame i GameController virker

4.0 Implementering

Vi viser her vigtige dele af koden, som kan være vigtige at forklare for at give en stærk idé om hvordan programmet fungere. Vi viser et udsnit af en del af koden, og kommentere på den del af koden nedenfor.

```
public void RunGame(){

    bs = new BoardSupport();
    gc = new GUIController(bs.getGb());
    dc = new DiceCup();

    gc.startGame();

    // Finder ud af hvor mange spillere der er, og returnere det som en int
    int numberOfPlayers = gc.GUIbuttons5("How  many players", "2", "3", "4", "5", "6");

    // Placere spillere ind i et array, og laver spillere tilsvarende til antallet Bruge
    playerArray = new Player[numberOfPlayers];
    for (int i = 0 ; i <= numberOfPlayers - 1 ; i++) {
        playerArray[i] = new Player();
    }
    //kører spilrunden for hver spiller så længe, der ikke er en vinder og springer dig
    while(!winner){
        for(int j=0;j<playerArray.length;j++){
            if(playerArray[j].isBankrupt())
                continue;
            if(!winner){
                showMessage("Player " + (j+1) +  " click OK to roll");
                playRound(playerArray[j]);
                if ( playerArray[j].isDoubleTurn() )
                    j--;
            }
        }
    }
}
```

Metoden der fra et visuelt synspunkt er hele spillets process. Den laver først et objekt af BoardSupport, som indeholder information om spillebrættets felter, samt dækket der indeholder de forskellige chancekort. GUIController (Til GUI'en, som laver det visuelle) og DiceCup, som indeholder information om terningerne.
Derefter beder den GUI'en, som spørger aktøren, hvor mange spillere der skal laves, som så bliver returneret som en integer, og den bliver brugt til at lave et array med antal spillere tilsvarende til, hvad aktøren trykkede. Derefter tjekker programmet om der er en vinder, og hvis der ikke er det, så kigger programmet gennem arrayet med spillerne, og hvis den spiller den kigger på er bankrupt, hopper den videre til næste spiller, og hvis spilleren ikke er bankrupt, så få den ham til at spille en runde, og hopper så videre til næste spiller, med mindre spilleren få en ekstratur.

5.0 Test

5.1 Tests

Vi har tilføjet nogle udsnit fra vores test i programmet, som fremviser hvordan vi bl.a. har testet vores program.

```
public class SodaTest {

    private Player player1test;
    private Player player2test;
    private Soda soda1;
    private Soda soda2;
    private DiceCup dctest;

    @Before
    public void setUpSoda() throws Exception {
        this.player1test = new Player("Player 1", 1, 30000);
        this.player2test = new Player("Player 2", 2, 30000);
        this.soda1 = new Soda("Soda1", Color.magenta, "Rip in pep(si)", 3000, null, 100, false);
        this.soda2 = new Soda("Soda2", Color.magenta, "Rip in pep(si)", 3000, null, 100, false);
        this.dctest = new DiceCup();
    }

    @After
    public void tearDown() throws Exception {
    }
```

Opsætter her entities som skal bruges til test af Field type Soda.

```
@Test
public void testEntities() {

    //Tester at alle Entities er blevet lavet.
    Assert.assertNotNull(this.player1test);
    Assert.assertNotNull(this.player2test);
    Assert.assertNotNull(this.soda1);
    Assert.assertNotNull(this.soda2);
    Assert.assertTrue(this.soda1 instanceof Soda);
    Assert.assertTrue(this.soda2 instanceof Soda);
    Assert.assertTrue(this.player1test instanceof Player);
    Assert.assertTrue(this.player2test instanceof Player);
}
```

Tester at entities er korrekt indsat og virker inden for deres type. Vigtig at vide at elementerne er korrekt indsat, for at sikre at vores test går ordentligt.

```java
@Test
public void testLandOnSoda() {

    //Starter med en grund balance for begge spillere, som testes ud fra.
    int expected1 = 30000;
    int expected2 = 30000;
    int actual1 = this.player1test.getBalance();
    int actual2 = this.player2test.getBalance();
    Assert.assertEquals(expected1, actual1);
    Assert.assertEquals(expected2, actual2);



    //Kører landOnField for soda1 som spiller 1, for at sætte ham som ejer.
    // TRYK PÅ KNAPPEN "JA" I GUI'EN HELE VEJEN GENNEM TESTEN
    System.out.println("TRYK JA FOR AT FORTSÆTTE TESTEN KORREKT");
    this.soda1.landOnField(player1test);


    //Tjekker at Spiller 1 har betalt for feltet.
    expected1 -= 3000;
    actual1 = this.player1test.getBalance();
    Assert.assertEquals(expected1, actual1);

    //Sikre at player1test er ejer af soda1, og at han kun ejer 1.
    Assert.assertTrue(soda1.getOwned());
    Player owner1 = this.soda1.getOwner();
    Assert.assertEquals(player1test,owner1);
    Assert.assertNotEquals(player2test,owner1);
```

Begynder at teste, hvor kommentarfelterne i koden kommentere, hvad der laves og hvad der testes.

```java
//Kører landOnField for soda1 som spiller 2, for at teste ham lande.
dctest.RollDices();
System.out.println("Terningerne rollede tilsammen: " + dctest.getSum());
player2test.getPiece().setLastDiceSum(dctest.getSum());
this.soda1.landOnField(player2test);


//Tjek om player1test har tjent penge, men ikke mere end 1200
expected1 = (27000 + (player2test.getPiece().getLastDiceSum()*100));
actual1 = this.player1test.getBalance();
Assert.assertEquals(expected1, actual1);
Assert.assertTrue(player1test.getBalance() >= 27200);
Assert.assertTrue(player1test.getBalance() <= 28200);

//Tjek om player2test har brugt penge, men ikke mere end 1200
expected2 = (30000 - (player2test.getPiece().getLastDiceSum()*100));
actual2 = this.player2test.getBalance();
Assert.assertEquals(expected1, actual1);
Assert.assertTrue(player2test.getBalance() <= 29800);
Assert.assertTrue(player2test.getBalance() >= 28800);
```

Tester noget mere, og fremviser flere ting der testes for. Kunne deles op i flere tests, hvis man gerne ville holde hver slags test af Soda field type mere separat.

5.2 Testcases

Vi har skrevet nogle testcases til programmet, som man bør kunne udvikle egne udgaver af test til programmet ud fra;

Soda Test

Denne Testcase fokusere på felt-typen Soda som nedarver fra Fields. Testen bør fokusere på landOnField metoden, under forskellige omstændigheder.

Preconditions:

Opret følgende objekter fra følgende klasser:

- *New Soda soda1*
- *New Soda soda2*
- *New Player player1test*
- *New Player player2test*
- *New DiceCup dctest*

Test:

Tjek følgende ting:

- *Alle entities er lavet, og hører til deres respektive klasse*
- *Tjek at spillernes balance er 30.000*

Få derefter player1test til at lande på Soda1, og gør ham til ejer.

Tjek følgende ting:

- *player1test har betalt for feltet*
- *soda1 er ejet*
- *player1test er ejer af soda1*
- *player2test er ikke ejer af soda1*
- *player1test ejer 1 soda-felt*

Få derefter player2test til at lande på Soda1.

Tjek følgende ting:

- *player1test skal have tjent penge når player2test lander på feltet*
- *player1test skal ikke have tjent mere end 1200.*
- *player2test skal have mistet penge når han lander på feltet*
- *player2test skal ikke have mistet mere end 1200.*

Få derefter player1test til at lande på Soda2, og gør ham til ejer.

Tjek følgende ting:

- *player1test har betalt for feltet*
- *soda2 er ejet*
- *player1test er ejer af soda2*
- *player2test er ikke ejer af soda2*
- *player1test ejer 2 soda-felter*

Få derefter player2test til at lande på Soda2.

Tjek følgende ting:

- *player1test skal have tjent penge når player2test lander på feltet.*
- *player1test skal ikke have tjent mere end 2400.*
- *player2test skal have mistet penge når han lander på feltet.*
- *player2test skal ikke have mistet mere end 2400.*

Postconditions:

- Fjern alle entities

Chance Test

Denne Testcase fokusere på “Prøv Lykken”-korten. Testen bør forkuserer på de forskellige korts virkninger på en spiller.

Preconditions:

Opret følgende objekter fra følgende klasser:

- *New Player p1*
- *New Deck d1*

Test:

Tjek følgende ting:

- *Tjek at spillerns balance er 30.000*

Få derefter p1 til at trække et kort af type 1 (change amount).

Tjek følgende ting:

- *Tjek at spillerens balance er steget*

Få derefter p1 til at stå på et felt, og derefter til at trække et kort af type 2 (Move to field).

Tjek følgende ting: ● *Tjek at spilleren står på det nye felt.* Få derefter p1 til at stå på et felt, og derefter til at trække et kort af type 3 (Move amount of fields). Tjek følgende ting: ● *Tjek at spilleren har rykket til det rigtige felt ud fra det felt han startede på.*
Postconditions: ● Fjern alle entities

6.0 Kildekode

- Indholdsfortegnelse til kildekoden f. eks. som bilag.
- Kildekode skal være formateret til læsbart format med linienumre.

7.0 Versionsstyring

Vi har brugt github til at styre versionering af vores applikation. Se bilag 1. Dette indeholder github link og ligeledes instruktion, om hvordan projektet hentes ned derfra.

8.0 Konfigurationsstyring

Programmet er kørt og testet på følgende opsætning:

- Eclipse Java EE IDE for Web Developers. Version: Mars.1 Release (4.5.1)
- Windows 10 Education 64-bit og 8.1 Education 32-bit
- Java version 8 update 65
- Java SE Develoment kit 8 update 65

9.0 Konklusion

I dette projekt har vi udformet en java applikation, som opfylder kundens centrale krav. Flere krav, som vi fandt frem til ud fra reference til kundens krav, blev også opfyldt, men nogle mangler at blive implementeret. Grundet tiden kunne vi ikke nå at implementere alle krav, som kunne aflæses ud fra reference til kundens krav. Vi har også lavet flere test for at fremvise programmets virkning, som testede forskellige type af felter samt klasser for spiller og mere. Vi har derudover lavet JUnit test på flere af vores klasser, for at sikre at de virker som de skal.

Under udviklingsprocessen har vi udformet diverse UML diagrammer, der dokumenterer vores fremgangs proces af projektet. De har hjulpet med at holde overblik over hele applikationen og gjort det muligt at have et blik på, hvad den færdige java applikation skulle indeholde.
Videreudvikling på programmet er muligt gennem kommentarer og dokumentation, og kan føre til opfyldelse af flere krav, som kunden kan se sig enige med, ud fra referencemateriale som bilag 1.
Programmet kan bruges i sin nuværende tilstand af aktører, og lever op til kundens krav.

10.0 Bilag

Bilag 1. Matador Manual
Bilag 2. Importering af projekt

11.0 Noter

GITHUB brugernavne::
FrederikBuur - Frederik Buur (s133045)
SondiDK - Oliver Sonderegger Larsen(s147302)
JSTM - Joachim Skov Thesbjerg Mehlsen (s141207)
Micniks - Michael N. Korsgaard (s150348)