DANMARKS TEKNISKE UNIVERSITET

32_CDIO_Final

02312-13-14-15

CHRISTOFFER VOIGT (S154308)
KASPER BØGVAD NIELSEN (S175216)
CASPER KYSTER ANDERSEN (S175213)
THOMAS LØVENDAHL VESTERGAARD (S175219)
AUGUST CORNELIUS FROST SCHILLING HEDEGAARD (S160435)

Timeregnskab

Navn	Analyse	Design	Implementering	Test	Dokumentation	Diverse	I alt
Kasper	4	4	4	6	2	0	20
Casper	2	2	10	2	4	0	20
Thomas	2	0	0.5	0	0	1	3.5
Cornelius	2	3	18.5	0.5	0	0	25
Christoffer	5	1.5	6.5	1	2.5	3	19.5

Indhold

1 Indledning og problemformulering

1.1 Indledning

I dette projekt udvikles der et spil, som kan køres på en computer med Windows 7, eller nyere, installeret samt med Java installeret. Opgaven er udviklet på baggrund af undervisningsmaterialet for fagene "Udviklingsmetoder til IT-systemer (02313)", "Indledende programmering (02312)" og "Versionsstyring og testmetoder (02315) ". Rapportens formål er at dokumentere udviklingen og tankerne bag arbejdsprocessen for spillet. Der vil desuden blive lavet test for at sikre at spillet virker optimalt.

1.2 Problemformulering

Kunden ønsker et Matador spil, som så vidt muligt har samme regler som et rigtigt Matador spil, udviklet til at blive spillet på computeren. Det er vigtigere for kunden at spillet fungerer, frem for at spillet er en fuldstændig kopi af det rigtige Matador spil.

2 Analyse

2.1 Funktionelle krav

Must have:

1. Spiller-antallet skal kunne variere mellem 2-6 spillere.
2. Spillerne skal alle hver have en brik.
3. Spillernes brikker skal blive stående på det felt de er landet på til dennes næste tur, medmindre andet er anvist.
4. Spillernes brikker skal kunne gå i ring på spillepladen.
5. Pladen skal have 40 felter.
6. Startbeholdningen skal for hver spiller være 30.000.
7. Spillet skal have to 6-siders terninger.
8. Spillet skal have 45 chancekort.
9. Spillerne skal alle starte på feltet "Start".
10. En spiller skal rykke det antal felter frem, som de slår med terningeslaget.
11. Spilleren skal modtage kr. 4000 fra banken, når spilleren passerer eller lander på feltet "Start.", medmindre andet er angivet.
12. Lander en spiller på et felt, der er ejet af banken kan spilleren købe det såfremt spilleren har råd.
13. Lander en spiller på et felt, ejet af en anden spiller, betales det felt-bestemte lejebeløb til ejeren.
14. Lander en spiller på et "Prøv lykken"-felt, trækkes et chancekort. Spilleren følger en anvisning.
15. Lander en spiller på "De fængsles"-feltet, rykker spillerens brik til "På besøg"-feltet. Selvom spillere passerer start, modtager denne ikke 4.000.
16. Er en spiller "fanget" i fængsel, skal spilleren i næste tur slå to ens indenfor tre slag, betale 1.000 eller bruge "benådningskortet"for at komme ud.
17. Hvis en spiller har siddet i fængsel tre runder i træk SKAL spilleren betale kr. 1.000 eller bruge "benådningskortet"for at komme ud af fængsel.
18. En spiller går fallit, hvis spilleren ikke er i stand til at betale for en service.
19. Den spiller, der besidder de største aktiver når spillet afsluttes vinder.
20. Der skal være en spilleplade.
21. Ejes alle felter i en serie af samme spiller kan der bygges huse og hoteller på grundene.

Should have:

22. Ejes alle felter i samme farve af samme spiller, men uden bygninger, er huslejen dobbelt så høj.
23. En spiller fanget i fængsel kan IKKE opkræve husleje, når en anden spiller lander på et felt, ejet af spilleren.
24. Slår man to ens får man ekstra slag.
25. Slår man to ens tre gange rykkes man direkte i fængsel.
26. Chancekortene skal blandes før start.

27. Chancekort, der bruges, lægges nederst i bunken.
28. Når alle 45 chancekort er trukket blandes bunken.
29. Fordelingen af chancekort skal følge det, i bilag 1 vedlagte materiale.
30. Ejendom-felterne er farvet i forhold til hvilken serie de tilhører.
31. Spillet fortsætter selvom en spiller er gået fallit.
32. Man kan afslutte spillet på et hvert givet tidspunkt og finde en vinder.
33. Man kan kun bygge huse på ens egen tur, før man slår med terningerne.
34. Man kan bytte og sælge grunde med andre spillere.

Could have:

35. Kan en spiller ikke betale, skal spilleren betale gælden ved at sælge bygninger, pantsætte, sælge grunde eller erklære sig fallit.
36. Hver spiller vælger sin farve på brikken.

Want to have:

37. Den yngste spiller starter.

2.2 Implementeringsstrategi

Iteration 1 04/01 kl. 12:00 [Done]

F1 Brikkerne skal kunne rykkes korrekt ved at slå med 2 terninger.

F2 Det skal være muligt at have en balance med en startværdi på 30000, der kan ændres.

F3 Hvis ens balance er under 0, skal det ikke være muligt for spilleren at fortsætte i spillet.

F4 Det skal være muligt at købe grunde.

Iteration 2 07/01 kl. 12:00 [Done]

F5 Der skal være et spillebræt.

F6 Der skal slås med 2 terninger med 6 sider hver.

F7 Der skal være et ekstra kast hvis man slår to ens.

F8 Der skal betales til ejeren af et felt af spilleren der lander på feltet.

F9 Der skal modtages 4.000kr ved passering (inkl. landing på) af start.

F10 Der skal trækkes et chancekort ved landing på tilsvarende felt.

Iteration 3 09/01 kl. 12:00 [Done]

Ændring i kodens struktur for at overholde GRASP-principperne. Udvikling af controller og information expert for at holde styr på de rigtige metoder.

Iteration 4 11/01 kl. 12:00 [Done]

F11 Det skal være muligt at købe huse og hoteller.

- Det skal kun være muligt at bygge på ens grunde under ens egen tur før man slår med terningerne.

Iteration 5 15/01 kl. 12:00 [Done]

Bugfixes, test, samt rapportskrivning.

Hvordan vi ville have forsat med opgaven for at få alle funktioner med:

Iteration 6 18/01 kl. 12:00

F12 Der skal være tvungen betaling af indkomstskat ved landing på tilsvarende felt

- Indkomstskat kan enten betales som 4.000kr eller 10% af ens samlede værd (penge, ejendomme samt evt. byg)

F13 Der skal være en begrænset mængde huse og hoteller. 32 huse og 12 hoteller.

F14 Hvis man trækker et fængselschancekort, eller lander på feltet "gå i fængsel", skal man gå i fængsel.

- Man skal smides i fængsel hvis man slår to ens tre gange i træk
- Hvis man er i fængselsfeltet udenfor det ovenstående tilfælde (F11), skal man enten slå to ens med terningen, betale 1.000kr, eller bruge et "gå ud af fængsel" kort for at komme ud af fængslet.

Iteration 7 21/01 kl. 12:00

F15 Det skal være muligt at auktionere ejendomme mellem spillerne.

- Der skal kun være mulighed for at bytte/auktionere med ubebyggede ejendomme.

F16 Man skal kunne pantsætte ubebyggede grunde til banken.

F17 Hvis man ikke er villig til at købe en ejendom når man lander på den, skal banken auktionere ejendommen.

Tabel 1: Feltliste

Felt	Navn	Type	Serie	Pris	Note
1	Start	Start			Modtag 4.000 ved passering
2	Rødovrevej	Ejendom	S1	1.200	Købe, udvide eller betale leje
3	Prøv lykken	Chance			Træk et chancekort
4	Hvidovrevej	Ejendom	S1	1.200	Købe, udvide eller betale leje
5	Betal Indkomstskat	Skat			Betal 4.000 eller 10%
6	Scandlines Helsingør-Helsingborg	Rederi	S10	4.000	Købe eller betale leje
7	Roskildevej	Ejendom	S2	2.000	Købe, udvide eller betale leje
8	Prøv lykken	Chance			Træk et chancekort
9	Valby Langgade	Ejendom	S2	2.000	Købe, udvide eller betale leje
10	Allégade	Ejendom	S2	2.400	Købe, udvide eller betale leje
11	På besøg i fængsel	Fængsel			På besøg eller i fængsel
12	Frederiksberg Allé	Ejendom	S3	2.800	Købe, udvide eller betale leje
13	Tuborg Squash	Brygeri	S9	3.000	Købe eller betale leje
14	Bülowsvej	Ejendom	S3	2.800	Købe, udvide eller betale leje
15	Gl. Kongevej	Ejendom	S3	3.200	Købe, udvide eller betale leje
16	Mols-Linien	Rederi	S10	4.000	Købe eller betale leje
17	Bernstoftsvej	Ejendom	S4	3.600	Købe, udvide eller betale leje
18	Prøv lykken	Chance			Træk et chancekort
19	Hellerupvej	Ejendom	S4	3.600	Købe, udvide eller betale leje
20	Strandvejen	Ejendom	S4	4.000	Købe, udvide eller betale leje
21	Parkering	Parkering			Du holder her gratis
22	Trianglen	Ejendom	S5	4.400	Købe, udvide eller betale leje
23	Prøv lykken	Chance			Træk et chancekort
24	Østerbrogade	Ejendom	S5	4.400	Købe, udvide eller betale leje
25	Grønningen	Ejendom	S5	4.800	Købe, udvide eller betale leje
26	Scandlines Gedser-Rostock	Rederi	S10	4.000	Købe eller betale leje
27	Bredgade	Ejendom	S6	5.200	Købe, udvide eller betale leje
28	Kgs. Nytorv	Ejendom	S6	5.200	Købe, udvide eller betale leje
29	Coca Cola	Brygeri	S9	3.000	Købe eller betale leje
30	Østergade	Ejendom	S6	5.600	Købe, udvide eller betale leje
31	De fængsles	Fængsel			Gå til fængsel
32	Amagertorv	Ejendom	S7	6.000	Købe, udvide eller betale leje
33	Vimmelskaftet	Ejendom	S7	6.000	Købe, udvide eller betale leje
34	Prøv lykken	Chance			Træk et chancekort
35	Nygade	Ejendom	S7	6.400	Købe, udvide eller betale leje
36	Scandlines Rødby-Puttgarden	Rederi	S10	4.000	Købe eller betale leje
37	Prøv lykken	Chance			Træk et chancekort
38	Frederiksberggade	Ejendom	S8	7.000	Købe, udvide eller betale leje
39	Ekstraordinærstatsskat betal 2.000	Skat			Betal 2.000
40	Rådhuspladsen	Ejendom	S8	8.000	Købe, udvide eller betale leje

2.3 Use case Beskrivelser

Use-case: Spil

Brief
To til fire spillere sætter sig ved en computer og spiller et spil Matador. Spillerne starter med en penge-

holdning og skiftes til at kaste med to terninger og rykke en brik rundt på en spilleplade. Spillepladen har forskellige felter, der har forskellige funktioner, som kan påvirke hvad spilleren kan gøre. Når der kun er en spiller tilbage afsluttes spillet.

Casual
To til seks spillere sætter sig ved en computer og spiller et spil Matador. Spillerne starter med en bengeholdning på 30.000. Spillerne skiftes til at kaste med to terninger og rykke en brik rundt på en spilleplade. Spillepladen har 40 felter:

- Et start felt.
- Seks chance felter.
- Et besøg fængsel felt.
- Et gratis parkering felt.
- Et gå i fængsel felt.
- To skatte felter.
- 28 ejendomsfelter.

Når en spillers pengebeholdning bliver mindre end 0 er de ude af spillet. Vinderen er spilleren som er tilbage til sidst.

2.4 Supplerende specifikationer ((F)URPS+)

Functionality

- Spillet skal kunne installeres og anvendes på DTUs databarer.

Usability

- Brugervenlighed: Programmet er lavet brugervenligt, sådan at en bruger kun skal trykke på de knapper som spillet angiver, og ikke skal tænke på andet end at vælge mellem de fremstillede valgmuligheder.

Reliability

- Robusthed:

Performance

- Svartider: Vi ønsker at vores program har svar tid på $\frac{1}{3}$ sekund eller lavere fra at brugeren interagerer med systemet til der visuelt sker noget på skærmen.
- Nøjagtighed:
- Ydeevne:

Supportability

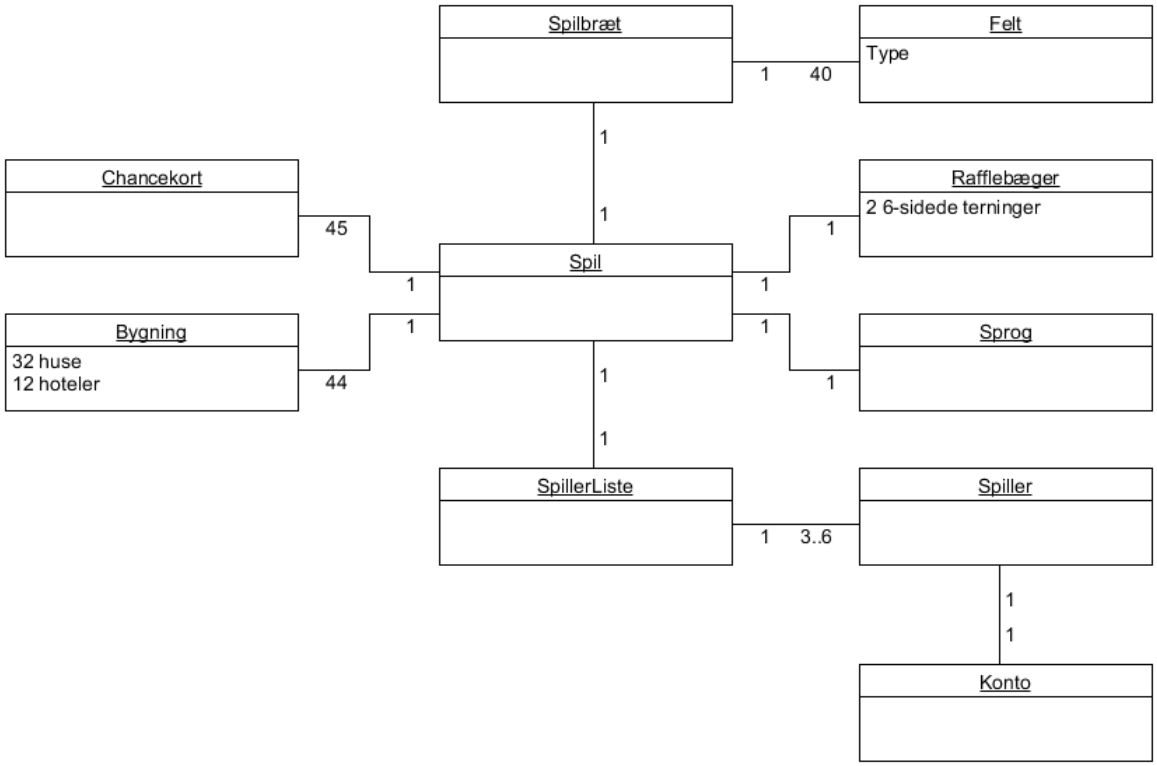
- Vedligeholdelse: Spillet bliver udviklet på en måde sådan at det efter udgivelse ikke har brug for yderligere support og kan fungere optimalt uden opdateringer.
- Anvendelighed: Spillet er lavet sådan at alle bare kan sætte sig ved en computer i databarene på DTU og spille uden nogen form for introduktion til hvordan de skal gøre.

Implementation

- Programmet udarbejdes i programmeringssproget Java

2.5 Domænemodel

Domænemodellen indeholder de vigtigste abstraktioner og informationer, der er nødvendige for at forstå domænet og de nuværende krav. De vigtigste klasser er afklaret. Desuden er navneordsanalyse anvendt på kravene for at udvælge navnene til henholdsvis klasser og metoder.



Figur 1: Domænemodel

2.6 Risikoanalyse

For at undgå så mange fejl som muligt, laves en liste over de mulige risici, som er involveret i udviklingen af projektet. Til de forskellige risici er der udarbejdet forslag til forebyggelse og håndtering, skulle risiciene indtræffe. Risiciene er farvekodet alt efter påvirkningskraft af projektet, så man ved, hvilke risici man skal fokusere på, bliver løst/forebygget. Det skal dog nævnes at ikke alle risici kan forebygges, men at man stadig klargøre løsninger skulle den pågældende risici indtræffe.

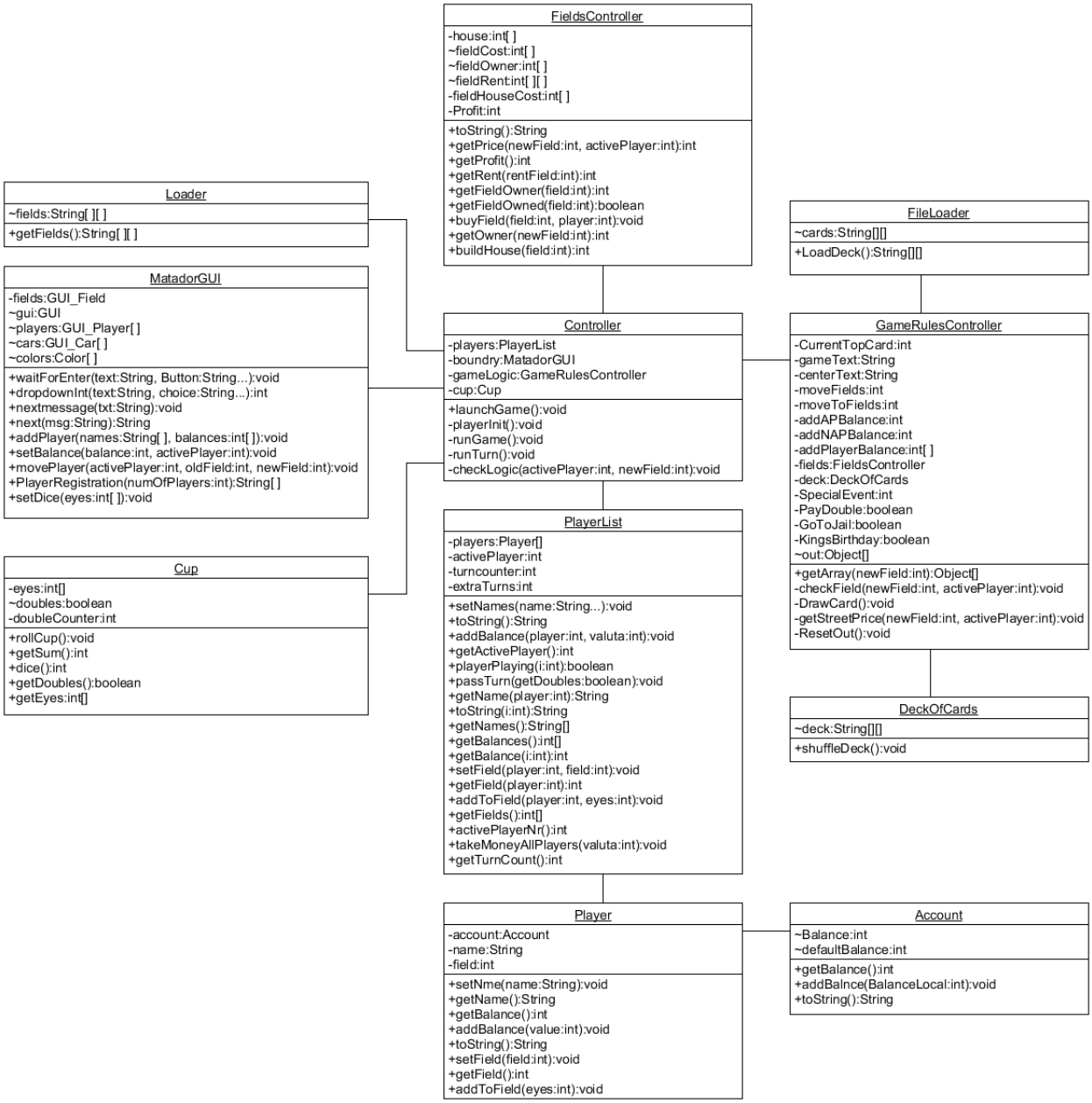
Tabel 2: Risikotabel

Risici	Sandsynlighed (0<= x >=1)	Skadevirkning (0<= y >=10)	Risikoindeks (x * y)	Forebyggelse og håndtering
Større ændringer af krav	0.1	10	1	Ofte revurdering af krav, så ændringerne foretages så tidligt i projektforløbet som muligt.
Mindre ændringer af krav	0.2	6	1.2	Ofterevurdering af krav, så ændringerne foretages så tidligt i projektforløbet som muligt.
Fejlestimering af tid til iterationer	0.3	4	1.2	Sørge for at estimering er grundigt analyseret og argumenteret
Miskommunikation	0.4	3	1.2	Gruppemedlemmer melder klart ud omkring hvad de laver, hvornår og så meget information som muligt.
Misforståelse af krav	0.1	7	0.7	Mundtlig gennemgang af krav mellem gruppemedlemmer.
Et gruppemedlem forlader projektet	0.05	6	0.3	Ligelig arbejdsfordeling samt. holde motiveringen oppe. Revurdering af arbejdsfordeling.
Hardwarefejl på computer	0.01	1	0.01	Ofte backup på online drev samt. eksternt lager
Softwarefejl på computer	0.05	2	0.1	Ofte backup på online drev samt. eksternt lager

Tabel 3: En tabel over de mulige risici, deres sandsynlighed for indtræden, skadevirkningen samt forebyggelsesmuligheder.

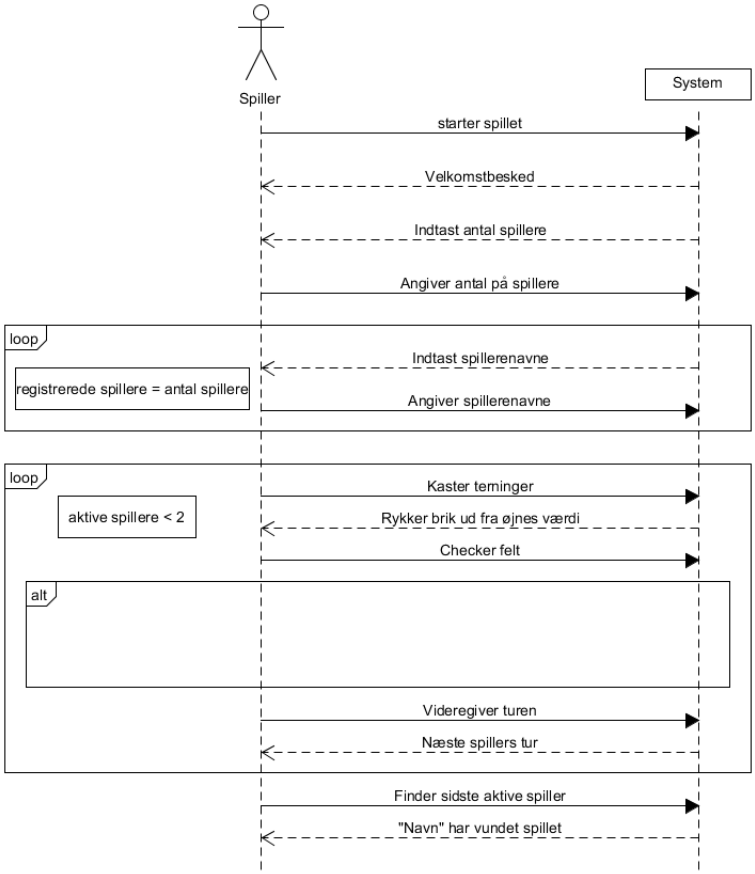
3 Design

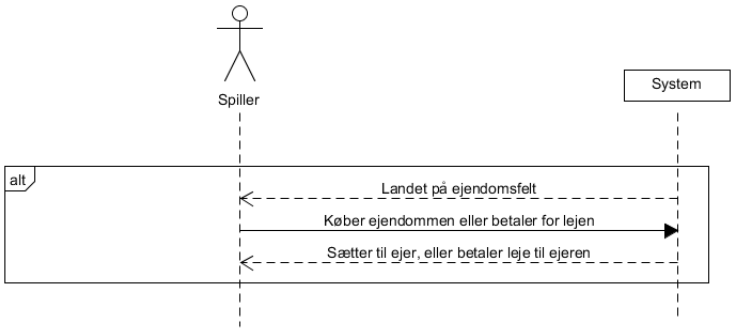
3.1 Design-klassediagram

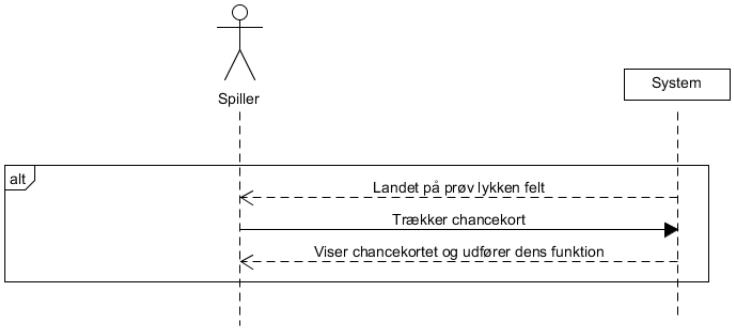


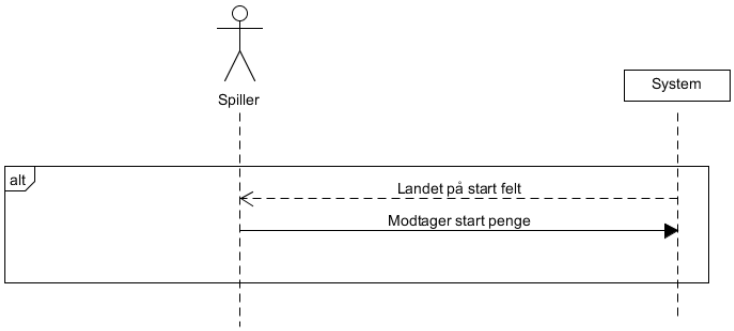
Figur 2: Et diagram som viser alle klasserne, deres forbindelser samt alle attributer og metoder, som klasserne indeholder.

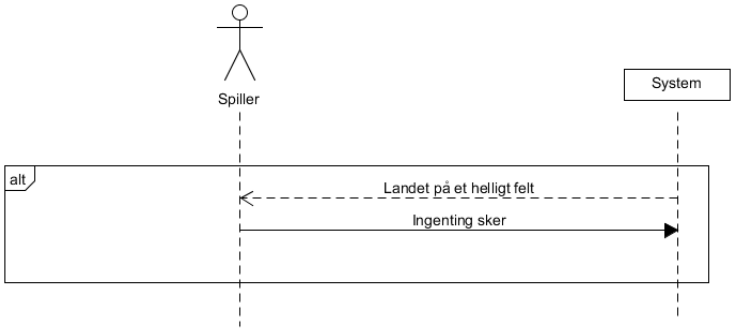
3.2 Systemsekvensdiagram



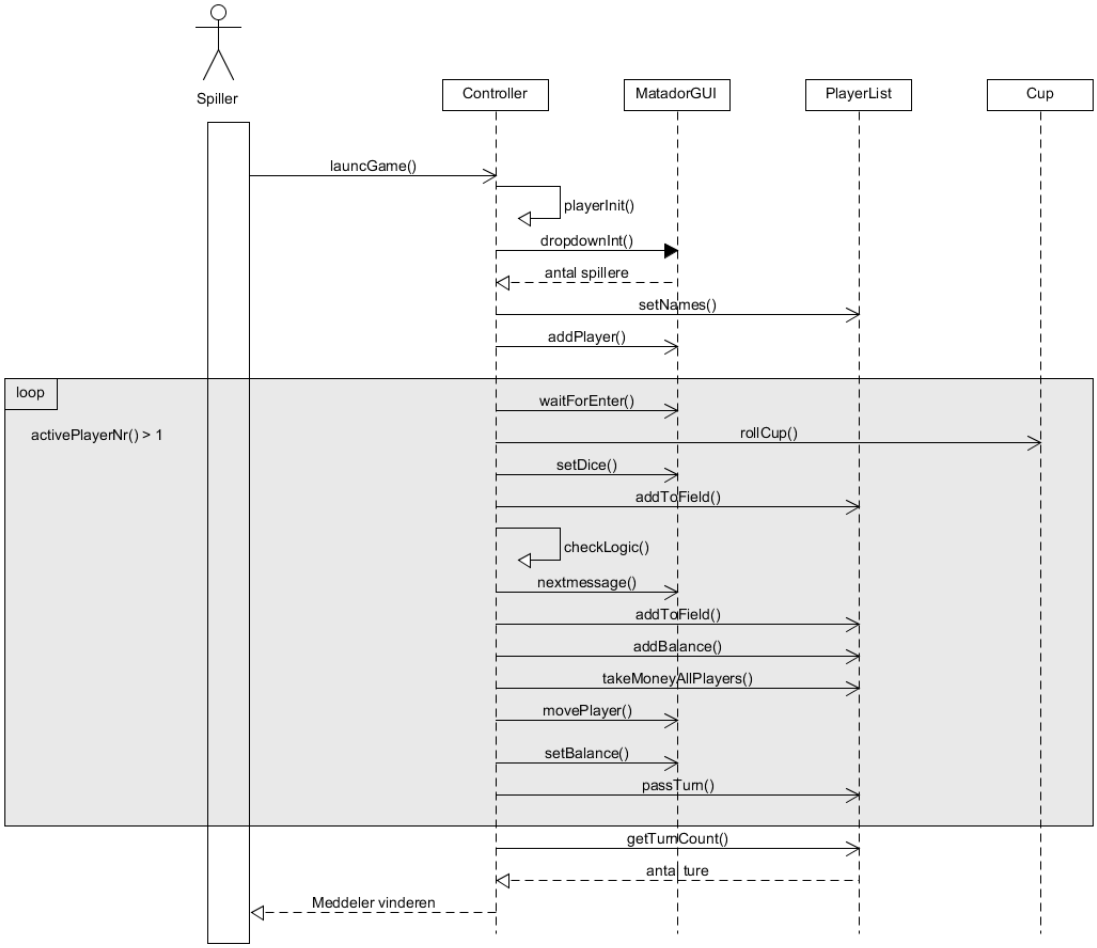





Spiller
System
alt
Landet på start felt
Modtager start penge


Spiller
System
alt
Landet på et helligt felt
Ingenting sker

3.3 Sekvensdiagram



4 Implementering

4.1 GRASP

Grasp (General Responsibility Assignment Software Patterns) består af metoder til fordeling af ansvar mellem objekter og klasser, indenfor objekt orienteret programmering. Spillet er udarbejdet efter udvalgte GRASP principper, til at fordele ansvar og opgaver blandt programmets forskellige klasser. Det udarbejdede designklassediagram viser resultatet af ansvarsfordelingen.

Creator
Creator mønsteret giver en beskrivelse til, hvor det vil være optimalt at instantiere objekter. Under dette princip betragtes, hvad der bruger objektet, hvad der har data til objektet, hvad der bliver gemt i objektet og om dette objekt er del af en komposition under et andet objekt. Kort sagt vil det sige at, hvis klasse A har ansvaret for objektet af klasse B, vil det være A's ansvar at instantiere B. "Cup"klassen er i programmets tilfælde klasse B og "Controller"klassen er klasse A.

Information Expert
Information Expert-princippet går ud på, som det ligger i navnet, at klassen som er ansvarlig for informationen, skal indeholde informationen. Princippet hjælper med at uddelegere ansvaret for beregninger og metoder til de klasser hvor de hører til. "Player"klassen kan beskrives som information expert-klasse, da den indeholder alle de informationer den selv skal anvende.

Controller
Her vil det sige at man blot skal have én klasse, som tager sig af at kontrollere de enkelte klassers sammenhæng for hvert use case. Dette bidrager til en lettere forståelig kode som også nemt kan redigeres.
I programmet er "Controller"controlleren som styrer programmet.

Low Coupling
Low Coupling princippet sikrer at så lav afhængighed mellem klasser som muligt, så påvirkningen af andre komponenter ikke sker ved ændringen i en en klasse, hvilket medfører en naturlig fleksibilitet ved udviklingen af programmet. Når et element ikke er afhængig af eller forbundet med for mange andre elementer, har det lav kobling. Ved udviklingen af programmet er den lave kobling eftertænkt gennem hele processen, og udført efter bedste evne. Den lave kobling mellem klasserne gør det let at forstå de forskellige dele, og er med til at gøre programmet let genbrugeligt.

High Cohesion
High Cohesion er med til at støtte den lave kobling i programmet, og bruges i programmet til at bevare objekterne fokuserede og forståelige. Programmet bliver også nemmere at genbruge til andre projekter. Ved High Cohesion kigges der på klasserne i forhold til ansvarsområde, om de har meget eller lidt ansvar. Programmet er udviklet efter tanken om, at ansvarsområdet for den enkelte klasse ikke må blive for stort. Dette er gjort gennem opdeling af klasser, så der fås flere klasser med mindre ansvar hver især.

4.2 Beskrivelse af klasser og objekter

Cup
Cup-klassen anvendes ved at oprette en seks-siddet terning som bruges til at slå med to gange i metoden "rollCup". Metoden tjekker også og men der bliver slået dobbelt og begrænser mængden af gange man kan slå dobbelt til tre gange.

PlayerList
Opretter et array med spillere fra klassen "Player". Klassen er ansvarlig for at sætte spillernes navn, ændre deres balance og at retunere en string med spiller navn og balance. PlayerList holder også styr på hvis spillers tur det er og sørger for at turen bliver givet videre. Klassen er også ansvarlig for at ændre felt placering.

Player
Klassen indeholder en spiller som bruges i klassen "PlayerList". Klassen indeholder metoder til at oprette en spiller med et navn og konto. Klassen er også ansvarlig for at "set-te og "get-te felt nummer, printe spillerens balance og bruge metoder fra klassen "Account".

Account
Opretter en konto for klassen "Player"med en default værdi på 30000. Klassen kan "get-te, printe og tilføje til kontoen.

Controller
En "Controller-klasse som indeholder metoder til at styre spillet spillet. Klassen instantierer objekter af andre klasser som bliver brugt til at udfører deres metoder. "Controller-klassen bruger metoderne "launchGame", "runGame", "runTurn"og "checkLogic"til at starte spillet og spørge om antallet af spillere og deres navne, til at checke antallet af aktive spillere og køre spillet, til at sørge for at spilleren gør de ting i sin tur som er påkrævet for at turen forløber korrekt samt at checke det nye felt og udføre dets påkrævede funktioner.

Loader
Konstruerer et String array med alle felter og deres værdier fra en .txt fil. Returnerer et String array.

FileLoader
Konstruerer et String array med alle chancekort og deres værdier fra en .txt fil. Returnerer et string array.

GameRulesController
GameRulesController klassen leverer spilleregler til spillet med "set"og "get"metoder samt metoder til at nulstille variabler.

FieldsController
Indeholder metoderne til at oprette et array med alle købbare felter, deres købs og lejepris og indeholder feltets ejer.

DeckOfCards
Indeholder metoderne til at oprette en string som indeholder et sæt kort og derefter blande dem. Indeholder også metode til at blande alle kortene i dækket igen skulle det blive nødvendigt.

5 Test

Et program kan have en del fejl, store som små, som er overset under produktionen eller ikke har været en fejl i producenternes øjne. Derfor er der gennemført en række test under selve udarbejdelsen, for at eliminere de væsentlige fejl i programmet, før den udleveres til kunden. Samtidig er der også udarbejdet en brugertest, udført af en person uden kendskab til projektet, for at teste brugervenligheden af spillet og for at finde fejl, som ikke blev bemærket af IOOuterActive. Testene har til formål at forbedre spillets kvalitet, således at brugeren får en god oplevelse samt, at virksomheden slipper for klager fra kunden.

5.1 White Box Testing

5.1.1 Junit test-1

Test Case ID	TC01
Referat	Tester at brikkerne rykkes korrekt
Krav	En spiller skal rykke det antal felter frem, som de slår med terningeslaget
Betingelser før	Spillerens felt er 1
Betingelser efter	Spillerens felt er 1+terningeslaget
Testens fremgangsmåde	1. Der oprettes en Junit test case. 2. Der opstilles en passende testmetode. 3. Junit test casen køres i Eclipse.
Test data	Terningøjne
Forventet resultat	Terningøjne + 1
Faktisk resultat	Terningøjne + 1
Status	Bestået
Testet af	Kasper Nielsen
Dato	07/01-2018
Testmiljø	Eclipse 4.7.0 på Windows 10

Tabel 4: Test case 1

```
@Test
/**
 * Checks the methods by comparing the expected field of the player with their
 * old field + the roll of the dice.
 */
public void testDiceToField() {
        this.player.setField(1);
        this.cup.rollCup();
        this.player.addToField(this.cup.getSum());
        int actual = this.cup.getSum();
        int expected = this.player.getField();
        assertEquals(actual + 1, expected);
}
```

Figur 3: Screenshot af DiceToField test

5.1.2 Junit test-2

Test Case ID	TC02
Referat	Tester spiller balance
Krav	Startbeholdningen skal for hver spiller være 30.000
Betingelser før	Spiller oprettet
Betingelser efter	Spillerens balance er 30.000
Testens fremgangsmåde	1. Der oprettes en Junit test case. 2. Der opstilles en passende testmetode. 3. Junit test casen køres i Eclipse.
Test data	getBalance()
Forventet resultat	30000
Faktisk resultat	30000
Status	Bestået
Testet af	Kasper Nielsen
Dato	07/01-2018
Testmiljø	Eclipse 4.7.0 på Windows 10

Tabel 5: Test case 2

```
@Test

/**
 * Tests the getBalance() method by comparing the starting balance which is set
 * to be 30000 with the number 30000.
 */
public void defaultBalanceCheck() {
        int expected = this.player.getBalance();
        int actual = 30000;
        assertEquals(actual, expected);
}

@Test
/**
 * Tests the addBalance() method by comparing the starting balance + 1000 with
 * starting balance + addBalance(1000).
 */
public void addBalanceCheck() {
        int defaultBalance = this.player.getBalance();
        this.player.addBalance(1000);
        int expected = this.player.getBalance();
        int actual = defaultBalance + 1000;
        assertEquals(actual, expected);
```

Figur 4: Screenshot af defaultBalanceTest

5.1.3 Junit test-3

Test Case ID	TC03
Referat	Tester at folk uden penge ikke kan fortsætte spillet
Krav	K18 En spiller går fallit, hvis spilleren ikke er i stand til at betale for en service
Betingelser før	Spillerens balance er negativ
Betingelser efter	Spillerens tur bliver sprunget over
Testens fremgangsmåde	1. Der oprettes en Junit test case. 2. Der opstilles en passende testmetode. 3. Junit test casen køres i Eclipse.
Test data	tilføj balance -30001
Forventet resultat	Spiller 2 får ingen tur
Faktisk resultat	Spiller 2 får ingen tur
Status	Bestået
Testet af	Kasper Nielsen
Dato	07/01-2018
Testmiljø	Eclipse 4.7.0 på Windows 10

Tabel 6: Test case 3

```java
@Test
/**
 * Tests if the player gets to be the active player when they have -1 in their
 * balance. Expected: They are NOT allowed to be the active player.
 */
public void PlayerNoTurnTest() {
        boolean actual = true;
        this.players.addBalance(1, -30001);
        for (int i = 0; i <= 10; i++, this.players.passTurn(false)) {
                if (this.players.getActivePlayer() == 1)
                        actual = false;
        }
        assertEquals(actual, true);
}

@Test
/**
 * Tests if the player gets to be the active player when they have 0 in their
 * balance. Expected: They are allowed to be the active player.
 */
public void PlayerNoTurnTestBoundry() {
        boolean actual = false;
        this.players.addBalance(1, -30000);
        for (int i = 0; i <= 10; i++, this.players.passTurn(false)) {
                if (this.players.getActivePlayer() == 1)
                        actual = true;
        }
        assertEquals(actual, true);
}

@Test
public void PlayerTurnTest() {
        int t = this.players.getActivePlayer();
        this.players.passTurn(false);
        int actual = this.players.getActivePlayer();
        int expected = t + 1;
        assertEquals(actual, expected);
}
```

Figur 5: Screenshot af EndGameTest

5.1.4 Junit test-4

Test Case ID	TC04
Referat	Tester at ejendomme kan købes
Krav	K12 Lander en spiller på et felt, der er ejet af banken kan spilleren købe det såfremt spilleren har råd.
Betingelser før	Spiller liste oprettet
Betingelser efter	Spiller 1 ejer ejendommen
Testens fremgangsmåde	1. Der oprettes en Junit test case. 2. Der opstilles en passende testmetode. 3. Junit test casen køres i Eclipse.
Test data	buyfield(1, 1) spiller 1 køber felt 1.
Forventet resultat	Spiller 1 ejer felt 1
Faktisk resultat	Spiller 1 ejer felt 1
Status	Bestået
Testet af	Kasper Nielsen
Dato	07/01-2018
Testmiljø	Eclipse 4.7.0 på Windows 10

Tabel 7: Test case 4

```
@Before
public void setUp() throws Exception {
        this.logic = new FieldsController(new FileLoader("src/gameRules/fieldsText.txt", 40, 8).LoadDeck());
}

@Test
/**
 * Tests by checking if the expected value of the field have changed ownership
 * to player 1 with a field where player 1 owns that field.
 */
public void testBuyProperty() {
        this.logic.buyField(1, 1);
        boolean expected = true;
        boolean actual = this.logic.getFieldOwned(1);
        assertEquals(actual, expected);
}
```

Figur 6: Screenshot af BuyPropertyTest

5.1.5 Junit test-5

Test Case ID	TC05
Referat	Tester at en enkelt terning er retfærdig i forhold til sandsynligheden med afvigelse på 2%
Krav	K7 Spillet skal have to 6-siders terninger.
Betingelser før	cup objekt oprettet
Betingelser efter	At sandsynligheden passer med en afvigelse på max 2%
Testens fremgangsmåde	1. Der oprettes en Junit test case. 2. Der opstilles en passende testmetode. 3. Junit test casen køres i Eclipse.
Test data	der bliver rullet en terning 100.000 gange.
Forventet resultat	16.67% sandsynlighed for hvert udfald +-2%
Faktisk resultat	16.67% sandsynlighed for hvert udfald +-2%
Status	Bestået
Testet af	Casper K. Andersen
Dato	09/01-2018
Testmiljø	Eclipse 4.7.0 på Windows 10

Tabel 8: Test case 5

5.1.6 Junit test-6

Test Case ID	TC06
Referat	Tester at to terninger er retfærdig i forhold til sandsynligheden med afvigelse på 2%
Krav	K7 Spillet skal have to 6-siders terninger.
Betingelser før	2 cup objekter oprettet
Betingelser efter	At sandsynligheden passer med en afvigelse på max 2%
Testens fremgangsmåde	1. Der oprettes en Junit test case. 2. Der opstilles en passende testmetode. 3. Junit test casen køres i Eclipse.
Test data	der bliver rullet to terninger 360.000 gange.
Forventet resultat	givne sandsynlighed for hvert udfald +-2%
Faktisk resultat	givne sandsynlighed for hvert udfald +-2%
Status	Bestået
Testet af	Casper K. Andersen
Dato	09/01-2018
Testmiljø	Eclipse 4.7.0 på Windows 10

Tabel 9: Test case 6

5.1.7 Junit test-7

Test Case ID	TC07
Referat	Tester at der betales leje til ejeren af feltet en anden spiller lander på.
Krav	K13 Lander en spiller på et felt, ejet af en anden spiller, betales det felt-bestemte lejebeløb til ejeren.
Betingelser før	Spiller liste oprettet
Betingelser efter	Lejen er betalt til ejeren af feltet
Testens fremgangsmåde	1. Der oprettes en Junit test case. 2. Der opstilles en passende testmetode. 3. Junit test casen køres i Eclipse.
Test data	Felt købes af spiller 1, og spiller 2 rykkes til feltet
Forventet resultat	Spiller 1 modtager leje, og spiller 2 betaler leje.
Faktisk resultat	Spiller 1 modtager leje, og spiller 2 betaler leje.
Status	Bestået
Testet af	Kasper Nielsen
Dato	09/01-2018
Testmiljø	Eclipse 4.7.0 på Windows 10

Tabel 10: Test case 7

```
@Test
public void test() {
        int payerActual = this.players.getBalance(1);
        int receiverActual = this.players.getBalance(0);
        this.logic.buyField(3, 0);
        this.logic.getPrice(3, 1);
        int payerExpected = this.players.getBalance(1);
        int receiverExpected = this.players.getBalance(0);
        assertEquals(payerExpected, payerActual - this.field3rent);
        assertEquals(receiverActual, receiverExpected - this.field3price + this.field3rent);
        System.out.println(this.players.getField(1));
}
```

Figur 7: Screenshot af PayRentTest

5.1.8 Junit test-8

Test Case ID	TC08
Referat	Tester at spiller får ekstra kast ved at slå to ens
Krav	K24 Slår man to ens får man ekstra slag.
Betingelser før	Spiller liste oprettet
Betingelser efter	Spilleren får et ekstra slag
Testens fremgangsmåde	1. Der oprettes en Junit test case. 2. Der opstilles en passende testmetode. 3. Junit test casen køres i Eclipse.
Test data	passTurn(True), passTurn(false)
Forventet resultat	Aktive spiller er 0
Faktisk resultat	Aktive spiller er 0
Status	Bestået
Testet af	Casper K. Andersen
Dato	09/01-2018
Testmiljø	Eclipse 4.7.0 på Windows 10

Tabel 11: Test case 8

```
@Test
public void test() {
        int a = player.getActivePlayer();
        System.out.println("Active player is " + player.getActivePlayer());
        player.passTurn(true);
        System.out.println("Active player is " + player.getActivePlayer());
        assertEquals(a, player.getActivePlayer());

        int b = player.getActivePlayer();
        System.out.println("Active player is " + player.getActivePlayer());
        player.passTurn(false);
        System.out.println("Active player is " + player.getActivePlayer());
        assertNotEquals(b, player.getActivePlayer());

}
```

Figur 8: Screenshot af ExtraTurnTest

5.1.9 Junit test-9

Test Case ID	TC09
Referat	Tester at spillere får 4.000 ved passering af start
Krav	K11 Spilleren skal modtage kr. 4000 fra banken, når spilleren passerer eller lander på feltet "Start", medmindre andet er angivet.
Betingelser før	Spiller liste oprettet
Betingelser efter	4000 er tilføjet til spillerens konto
Testens fremgangsmåde	1. Der oprettes en Junit test case. 2. Der opstilles en passende testmetode. 3. Junit test casen køres i Eclipse.
Test data	spilleren rykkes 40 felter frem.
Forventet resultat	Spillerens balance er forhøjet med 4000
Faktisk resultat	Spillerens balance er forhøjet med 4000
Status	Bestået
Testet af	Kasper Nielsen
Dato	09/01-2018
Testmiljø	Eclipse 4.7.0 på Windows 10

Tabel 12: Test case 9

```
@Test
public void StartMoneytest() {
        int expected = this.players.getBalance(0);
        this.players.addToField(0, 41);
        int actual = this.players.getBalance(0);
        assertEquals(actual, expected+this.startmoney);
}

@Test
public void StartMoneyBoundrytest() {
        int expected = this.players.getBalance(0);
        this.players.addToField(0, 40);
        int actual = this.players.getBalance(0);
        assertEquals(actual, expected+this.startmoney);
}
```

Figur 9: Screenshot af StartMoneyTest

5.2 Black Box Testing

5.2.1 Brugertest

Vi ville gerne have haft udført følgende brugertest på en eller flere personer, som har spillet Matador før, men ikke på computeren.

Tabel 13: My caption

Spørgsmål	Svar	Uddybende besvarelse
Ud fra din kendskab til matador kunne du så finde ud af spillet?		
Var der brug for uddybende vejledning i hvordan du skulle foretage en handling?		
Syntes du at spillet var brugervenligt?		
Hvordan var spillængden i forhold til hvor lang tid du gerne ville spille?		
Var der nogle fejl og mangler?		

6 Brugervejledning

Programmet anvendes ved at køre filen 32_final.JAR. Dernæst indtastes spillernes navne og spillet er nu klar til at begynde.

6.1 Minimumskrav

Følgende skal minimum være til rådighed for at køre programmet:
To brugere, mus, tastatur, skærm, 5 MB ledigt lagerplads på harddisken og Java 7 eller nyere.

6.2 Import af programmet i Eclipse (Oxygen)

1. Åbn Eclipse
2. Tryk på "file"og vælg "import"
3. Under "General"vælg "Existing Projects into Workspace"og tryk "Next"
4. Tjek om der er et hak i "Select archive file"
5. Find filen på computeren og vælg den
6. Tryk på "Finish"

6.3 Import fra Github

1. Åbn Eclipse
2. Tryk på "File"og vælg dernæst "Import"
3. Under "Git"vælges "Projects from git"og herefter på "Next"
4. Vælg "Clone URL"og tryk så "Next"
5. Indsæt https://github.com/MegaRandomCake/CDIO_Finale i feltet "URL"og tryk derefter på "Next"
6. Vælg hvilke branches du vil importere
7. Vælg hvor du vil have projektet
8. Tjek at der er hak i "Import existing Eclipse projects"
9. Tryk på "finish"

6.4 Kør programmet i Eclipse

1. Åbn Eclipse (programmet er installeret og importeret til Eclipse)
2. Vælg projektet i Package Explorer (venstre side i Eclipse standard view)
3. Vælg controller pakken under projektet "32_final"
4. Højreklik derefter på SpilSpil Klassen og vælg "run as..."og vælg "1 Java Application"

7 Konklusion

7.1 Proces

Vi har i gruppen haft en god arbejdsproces med at analysere, designe, implementere, teste og dokumentere. Processen har været nogenlunde flydende og har overholdt de fleste af internt satte deadlines.

7.2 Produkt

Programmet som er blevet lavet, simulerer et Matador spil. Produktet er blevet godt og opfylder de vigtigste krav som der er blevet stillet, sådan at det minder om et rigtigt matadorspil.

7.3 Perspektivering

Sammenlignet med et rigtigt matadorspil mangler der nogle muligheder såsom betaling af skat ved landing på skattefelt, en begrænset mængde huse og hoteller, fængselsfunktion, auktionsfunktion, samt pantsættefunktion. I vores implementeringsstrategi har vi indskrevet hvordan vi ville arbejde videre med opgaven, hvis der havde været mere tid, for at få implementeret de manglende funktioner.

7.4 Samlet

Samlet mener vi at vi har lavet et godt produkt uden fejl, dog med nogle manglende funktioner som ville kunne implementeres simpelt, skulle der have været givet mere tid.

8 Bilag

8.1 Kildekode

Kildekode til vores projekt der er afleveret som 32_CDIO_FINAL:
`https://github.com/MegaRandomCake/CDIO_Finale`