

Danmarks
Tekniske
Universitet



CDIO Del 3

AUTHORS

Group 1

Mohamed Zaabat - s235459

Julius Østerbye - s235462

Philip Amtrup Andersen - s226870

Anders Carmel - s235445

Asger Frandsen - s230577



24. november 2023

Indhold

1	Summary	1
2	Introduction	1
3	Requirements/analysis	1
3.1	Kravsspecifikationer	1
3.2	Krav til implementering	2
3.3	Krav til videreudvikling	2
3.4	Use case	3
3.5	Stakeholders:	4
3.6	Preconditions:	5
3.7	Success guarantee:	5
3.8	Main success scenario:	5
3.9	Alternative scenario:	6
3.10	Special requirements:	8
3.11	Technology and data variations:	8
3.12	Frequency of occurrence:	8
3.13	Open issues:	8
3.14	Sequence Diagram:	9
3.15	System Sequence Diagram:	10
3.16	Domain model:	11

4 Design	12
4.1 Design class diagram:	12
5 Testing	13
5.1 MonopolyTest:	13
5.2 Playertest:	13
5.3 PositionTest:	14
5.4 Test Cases	16
5.4.1 Player names	16
5.5 Brugertest	18
6 Configuration	18
7 Documentation	19
7.1 Hvad er arv?	19
7.2 Hvad er en abstract class?	19
7.3 Hvad hedder det hvis alle fieldklasser har en landOnField metode der gør noget forskelligt?	19
7.4 GRASP	19
7.4.1 Information Expert:	19
7.4.2 Creator	20
7.4.3 Controller	20
7.4.4 Low Coupling	20
7.4.5 High Cohesion	20

7.4.6	Polymorphism	20
7.4.7	Protected Variations	20
8	Conclusion	21

1 Summary

I denne opgave har vi lavet en digital udgave af Monopoly junior spillet. Det er et program, som er skrevet i Java med Visual Studio Code. Programmet er blevet sammensat af Anders, Julius, Phillip, Mohamed og Asger.

Her kan i se vores kode i et github repository:

https://github.com/Mozaab/G01_del3.git

2 Introduction

IOOuteractive har fået til opgave at lave en Monopoly junior spil. Spillet er en digital udgave af det fysiske brætspil Monopoly junior, som har 24 felter som man kan flytte sig rundt på. I spillet skal man kunne købe og eje felter, betale husleje til ejeren af feltet hvis man lander på det. Vi har ændret nogle ting i spillet for at gøre det mere interessant, hvor vi heriblandt har omskrevet felterne til at være forskellige fodboldklubber, og gjort således at man både kan vælge om man vil købe et felt eller ej, når man lander på det. Der er også nogle felter, chancefelter, hvor en bestemt handling skal udføres. Vi har også undladt at lave et time-out felt og istedet valgt at lave 3 "ferie felter" som skal repræsentere fodboldsæsonens pauser.

3 Requirements/analysis

3.1 Kravsspecifikationer

- Spilleren skal blive oplyst om hvad der sker.
- Der skal blive lavet en brugsanvisning i sammenhæng med spillet.
- En introduktion i starten med spillets regler.
- Mindst to spillere.
- Man kan købe felter.
- Man kan eje felter.
- Der skal være 24 felter.
- Én terning.
- Spillet skal fremvise konsekvenser.
- Felterne skal have forskellige egenskaber.
- Der skal være chancekort.
- Der skal være et Startfelt.
- Lander man på et ikke ejet felt kan man købe feltet.

- Lander man på et felt, en anden spiller ejer, skal du betale husleje. - Ejer du selv feltet, skal du ikke gøre noget.
- Man vinder ved at ramme 50 eller taber hvis man rammer 0.
- Spillet skrives på engelsk.

3.2 Krav til implementering

- Abstract class Monopoly - Public variabler - Skal kodes i Java - Felter der implementerer/ extender abstract classen.
- Get-funktioner, der viser spillerens position og om det er et chancefelt.
- String output, der giver en string ud fra hvilket felt der er landet på, og konsekvensen.
- Lost konsekvens, der viser hvis spilleren har tabt, (ramt 0) - Won konsekvens, boolean der viser hvis spilleren har vundet, (ramt 50) - Stringene osv skrives på engelsk

3.3 Krav til videreudvikling

Der skal være en junit test til de centrale metoder vi har i koden. Der skal være en brugertest fra en der ikke har noget viden om kodning.

Felter + værdi

- (OB - 1)
- (AGF - 1)
- (Lyon - 1)
- (Toulouse - 1)
- (Wolfsburg - 2)
- (Frankfurt - 2)
- (Brøndby - 2)
- (FCK - 2)
- (Man United - 3)
- (Chelsea - 3)
- (AC Milan - 3)
- (Juventus - 3)
- (Tottenham - 4)
- (Arsenal - 4)
- (Bayern - 5)
- (Real madrid - 5)

Chancekort

- Ryk frem til start og tjen: 2 kr.
- Ryk 5 felter frem!
- Du kommer for sent til træning betal 2 til manageren.

- Du scorer et hattrick og alle giver dig 3.
- Du får en belønning for at komme til alle træningerne og modtager 2.
- Ryk frem til Frankfurt.
- Ryk frem til Chelsea.
- Dine fans hader dig og smadrer din bil. Du skal betale 1 penge for at reparere den.
- Du renoverer en tribune på dit stadion og skal betale 2.
- Dit klubhus bliver evalueret til en højere værdi end før. betal 1 i ejendomsskat.
- En oliesheik fra Saudi-Arabien vil sponsorere dit hold. Modtag 6.
- Du har spillet godt på ungdomsholdet og får din debut på førsteholdet. Modtag 2.

3.4 Use case

Use case brief

Åbn spillet ved at opstarte programmet.

Vis brugsanvisning.

Vælg navn.

Spiller 1 vælger navn, efterfulgt af spiller 2,3 og 4.

Slå med terninger

Spiller 1 starter, efterfulgt af spiller 2,3 og 4. De lander på et felt.

Kan købe felt, hvis ikke det allerede er ejet.

Betal husleje, hvis anden spiller ejer det.

Træk chancekort, hvis du lander på chancefelt.

Hvis du lander på et feriefelt sker der ikke noget.

Hvis man lander på eller passerer start, udbetales 2 til spilleren.

Hvis spillerens formue rammer 0kr, går de fallit, og modstanderen med flest penge vinder.

Vi har lavet et use case diagram, som illustrerer spilerens interaktioner med systemet, samt de inbyrdes interaktioner mellem de forskellige dele af systemet.



Use case 1: (UC1) Fully dressed central

Scope: Monopoly junior spil

Level: User goal

Primary actor: Spillere

3.5 Stakeholders:

- Spilleren vil gerne have et spil, som er sjovt og nemt at spille. Eftersom spillet er Monopoly Junior og målgruppen er børn, skal spillet være intuitivt og nemt at forstå.
- IOOuteraktive vil gerne have at brugerne kan forstå spillet, og gøre kunden/ejeren tilfreds ved at lave et spil som nemt kan ændres, på et senere tidspunkt.
- Kunden vil gerne have et spil som kan underholde spillere, samt skal det også være muligt at kunne bygge videre på spillets kode.

3.6 Preconditions:

Spillerne skal kunne læse brugsanvisningen og forstå computeren godt nok til at kunne udføre instruktionerne skrevet til spillet, de har også læst brugsanvisningen til Monopolspillet og forstår reglerne der gør det muligt at spille.

3.7 Success guarantee:

Spillet er slut. Der er fundet en vinder eller taber, og spillerne har haft en sjov og hyggelig oplevelse med at spille Monopoly.

3.8 Main success scenario:

1. Spillere beslutter sig for at spille Monopoly junior på PC.
2. Spillerne stater programmet.
3. Spillet starter og reglerne til spillet bliver printet ud så spillerne kan læse reglerne.
4. Spillet spørger spillerne om antallet af spillere mellem 2-4.
5. Spillerne indtaster et tal imellem 2-4.
6. Spillet beder spiller 1 om at indtaste et navn + enter.
7. Spiller 1 indtaster sit navn.
8. Spillet udfører en iterativ process af step 6 indtil alle 2-4 spillere har indtastet deres navn, og har gennemført step 7.
9. Spillet beder spiller 1, rulle med terningen.
10. Spiller 1 ruller med terningen.
11. Spiller lander på et felt som kan købes.
12. Spillet spørger spilleren om de vil købe feltet.
13. Spilleren vælger at købe feltet.
14. Spillet udfører en transaktion og giver feltet over til spilleren, og tager imod penge for feltet.

15. Spillet beder næste spiller rulle med terningen.
16. Den næste spiller ruller med terningen og step 10-14 gentages.
17. En spiller lander på et felt som allerede er ejet af en anden spiller og betaler husleje til ejeren af feltet.
18. Spillet beder en spiller rulle med terningerne og de lander på et chancefelt. Spillet giver et output til spilleren om hvad der står på chancekortet der trækkes.
19. En spiller ruller med terningen og lander på et ferie felt. Derfor er det næste spillers tur.
20. En spiller passerer start og modtager 2 penge over start.
21. Step 9-20 gentager sig indtil en spiller ikke har flere penge og går fallit, eller en spiller får 50 penge på kontoen i hvilke tilfælde vinder de spillet.
22. En spiller vinder spillet.
23. Spilleren med flest penge på kontoen bliver meddelt som vinder af spillet, hvis der ikke opnås en saldo på 50.
24. Spillet er slut og spillerne lukker programmet.

3.9 Alternative scenario:

1. Spillere beslutter sig for at spille Monopoly Junior på PC.
 - 2a. Spillerne starter programmet.
 - 2b. Spillerne kan ikke starte spillet op ordentligt.
 - 3a. Spillet starter og reglerne til spillet bliver printet ud så spillerne kan læse reglerne. Spillerne læser reglerne.
 - 3b. Spillet starter og reglerne til spillet bliver printet ud så spillerne kan læse reglerne. Spillerne læser ikke reglerne.
4. Spillet spørger spillerne om antallet af spillere mellem 2-4.
 - 5a. Spillerne indtaster et tal imellem 2-4.
 - 5b. Spillerne indtaster en værdi som ikke ligger mellem 2-4 eller skriver eksempelvis ("fire"

el. “to”) istedet. Spillet beder spillerne prøve igen indtil scenarie (5a) er udført.

6. Spillet beder derefter spiller 1 indtaste et navn + enter.

7a. Spiller 1 indtaster sit rigtige navn.

7b. Spiller 1 indtaster andet end sit eget navn.

8. Spillet udfører en iterativ process af step 6 indtil alle 2-4 spillere har indtastet et navn, og har gennemført step 7a. el 7b.

9. Spiller1 bedes rulle med terningen.

10a. Spiller 1 ser beskeden med det indtastede navn og ruller med terningen.

10b. Spillerne ser beskeden med et navn men kan ikke huske deres indtastede navne.

11a. Spiller 1 lander på et felt som kan købes.

11b. Spiller 1 lander på et felt som ikke kan købes.

12. Spillet spørger spilleren om de vil købe feltet, hvis dette er muligt.

13a. Spilleren vælger at købe feltet!

13b. Spilleren vælger ikke at købe feltet

14. Spillet udfører en transaktion og giver feltet over til spilleren og tager imod penge for feltet.

15. Spillet beder næste spiller rulle med terningen.

16. Den næste spiller ruller med terningen og 10-12 gentager sig for spilleren.

17a. En spiller lander på et felt som allerede er ejet af en anden spiller og betaler husleje til spilleren som ejer feltet.

17b. Ingen spillere har købt et felt derfor kan en spiller blive bedt om at købe feltet. Step 12-14 gentages.

18. Spillet beder en spiller rulle med terningerne og de lander på et chancefelt. Spillet giver et output til spilleren om hvad der står på chancekortet der trækkes.

19. En spiller ruller med terningen og lander på et feriefelt og derfor er det næste spillers tur.

20. En spiller passerer start og modtager 2 penge over start.

21a. Step 9-20 gentager sig indtil en spiller ikke har flere penge og går fallit.

21b. Step 9-20 gentager sig for evigt fordi ingen spillere har købt et felt.

21c. Step 9-20 gentager sig, og en spiller får 50 penge og vinder spillet

22. Spilleren med flest penge på kontoen bliver meddelt som vinder af spillet, hvis der ikke opnås en saldo på 50.

23. Spillet er slut og spillerne lukker programmet.

3.10 Special requirements:

- En computer som kan køre et java program.
- En skærm som enten er stor nok til at 4 personer kan sidde omkring den, eller har en størrelse som kan passeres rundt til de forskellige spillere.
- Et tastatur til at interagere med programmet.

3.11 Technology and data variations:

- Indtil videre er spillet kun åbent til folk som har Java på en PC. Selvom der er variationer imellem styresystemer burde dette ikke være en udfordring.
- Spillet kan teknisk set bruges på en tablet men det vides ikke hvordan spillerne ville interagere med spillet.

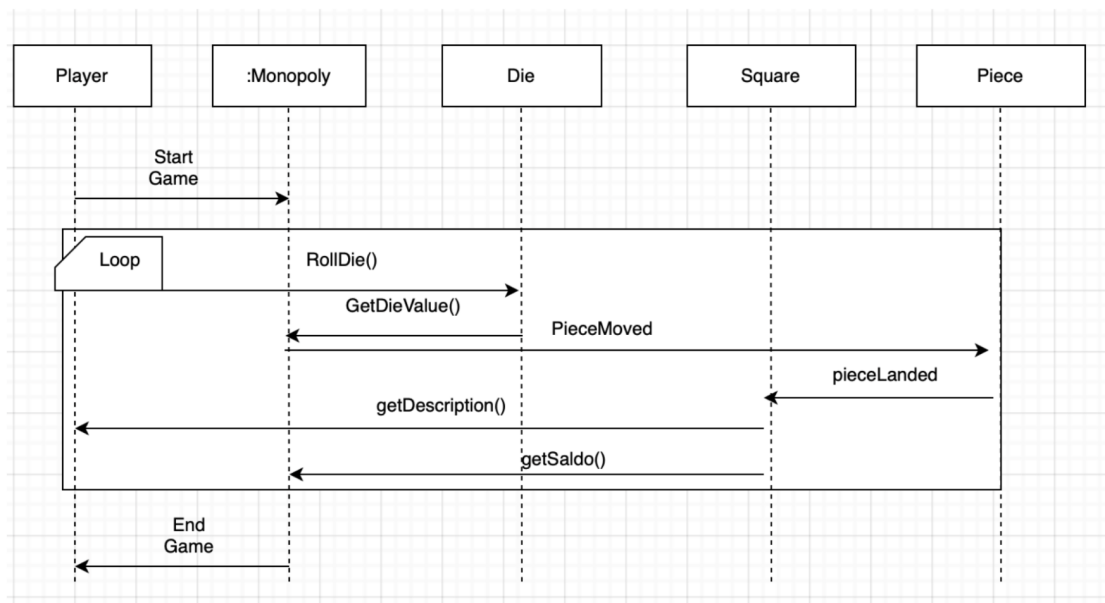
3.12 Frequency of occurrence:

Kan være næsten uendeligt så længe nogen spiller spillet eftersom at alle kan spille det på en normal PC med java.

3.13 Open issues:

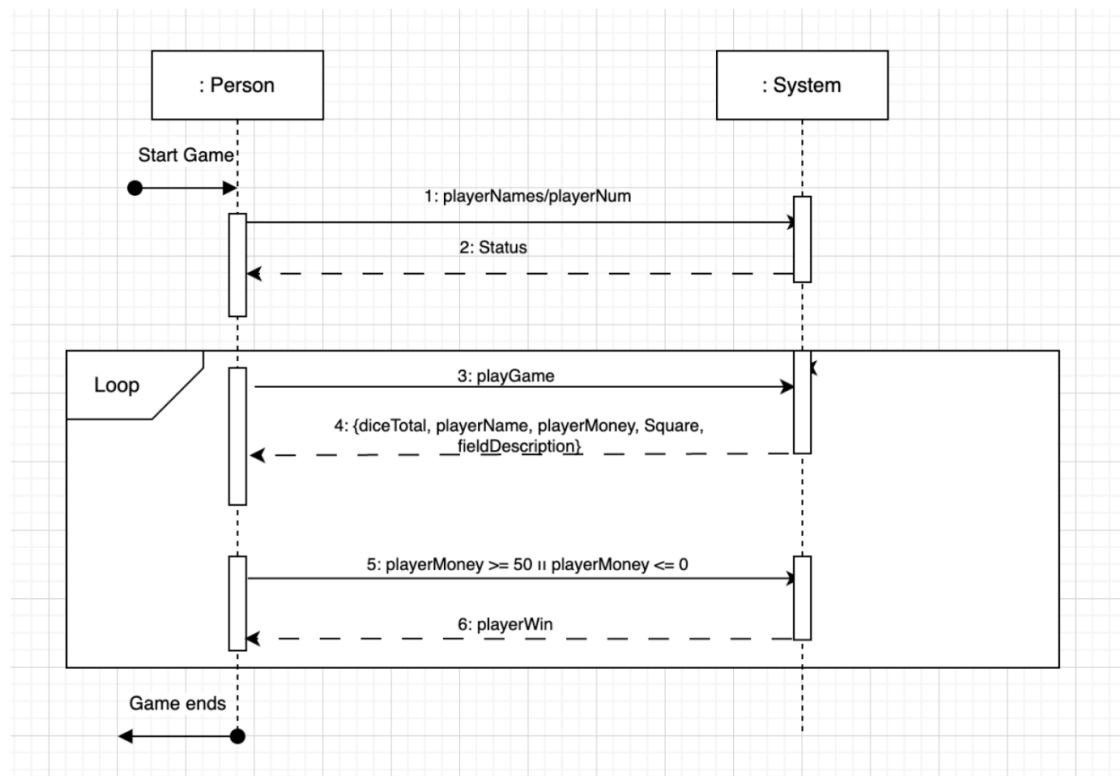
- Skal der være en øvre grænse for hvor mange penge der kan deles ud i spillet, således at en person kan vinde hvis spillet trækker ud? - Er det garanteret at spillet slutter?
- Ville det være mere optimalt hvis der var en tidsgrænse på spillet?

3.14 Sequence Diagram:



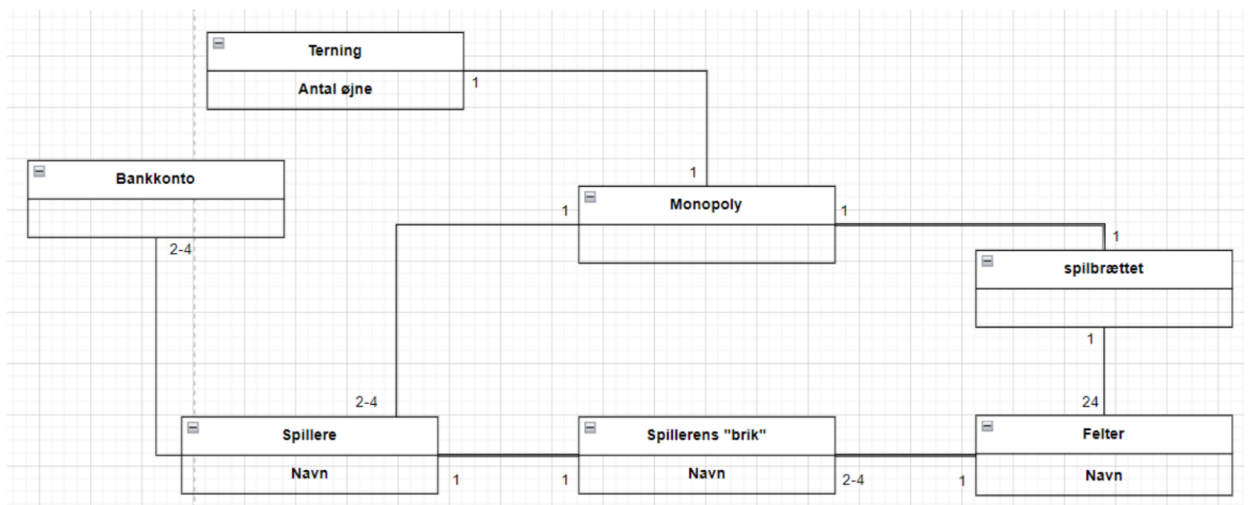
Ovenfor kan der ses et Sequence Diagram, hvor der bliver dannet et overblik over de forskellige sekvenser som findes i vores program. Derudover findes der inden i diagrammet eller i programmet, et loop som i sig selv er spillet der kører. Først og fremmest kan det ses at der fra 'Player' sker et input der starter spillet, hvor dette input går fra 'Player' til 'Monopoly', som i programmet er klassen 'MonopolyGame'. Derefter går programmet ind i loopet, hvor spillet går igang. Når terningen bliver slået, så får man en DieValue, hvor terningens værdi bliver sendt til selve spillet, som derefter skaber de konsekvenser der er, alt efter det respektive felt der landes på. Derefter får 'Player' en String som netop er en beskrivelse af hvad der er sket og hvad spilleren nu skal gøre. Alt efter hvad spilleren vælger af at købe eller sælge grunden, sker der en opdatering af saldoen, som spillet også får, og vurderer ud fra vores krav, hvorvidt spilleren har vundet eller tabt. Det resultere i at spillet så ender, og spillet selvfølgelig går ud af loopet.

3.15 System Sequence Diagram:



Som Sekvens diagrammet til dels også viser, så danner system sekvensdiagrammet et overblik over hvad der sker imellem system og actoren, som i dette tilfælde står som henholdsvis 'System' og 'Person'. Her kan der ses hvilke aktioner der sker undervejs, og hvilke informationer som bliver delt imellem systemet og actoren, hvor man i dette tilfælde kan se, at spilleren først starter spillet, og hvor der så bliver indtastet et navn som bliver opfanget af systemet. Derefter returnerer systemet en 'Status', der inkludere en beskrivelse. Derefter starter spillet, som vises i diagrammet ved at der sker et loop. Her spiller spilleren, hvor der går en 'playGame' aktion til systemet, som returnere et output af diceTotal, playerName, playerMoney, Square og fieldDescription, i form af integers og strings. Derefter vurderer systemet, hvorvidt spilleren har nået 50 eller 0 point, hvilket enten vil resultere i at spilleren har vundet eller tabt. Derfra vil systemet så give et output alt efter resultatet, hvor spillet herefter sluttet.

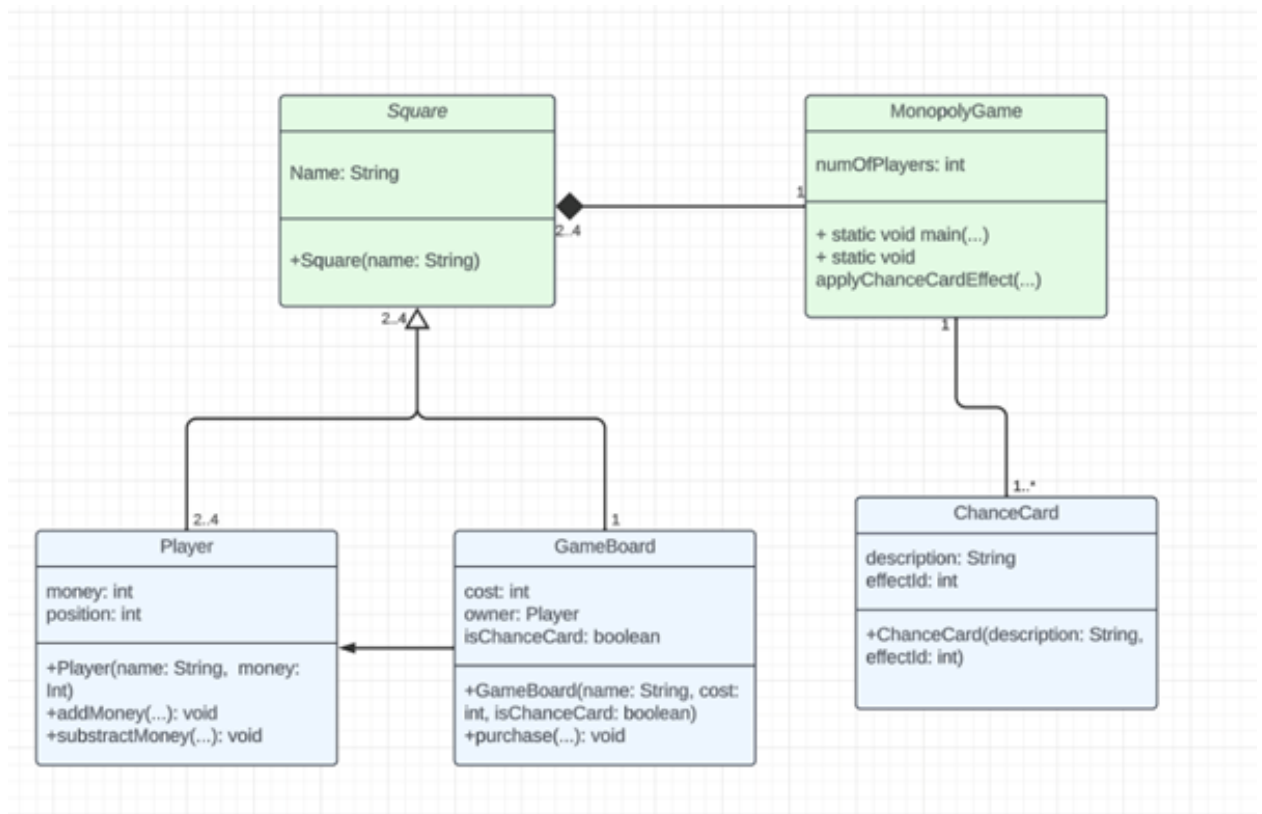
3.16 Domain model:



Domænemodellen ovenfor viser relationerne imellem de forskellige klasser og attributter der tilsammen udgør spillet. Her uddybes relationerne imellem hinanden, og i hvilket antal. Heriblandt kan det ses at man starter med en enkelt terning, der har et tilhørende attribut 'antal øjne', som går til et enkelt monopolyspil. Det går videre til et enkelt spilbræt som består af 24 felter, der har et attribut 'Navn'.

4 Design

4.1 Design class diagram:



Her har vi et UML class diagram, som beskriver vores program. Vi har inkluderet klasserne: Square, Player, GameBoard, Chancecard og Monopoly, sammen med deres attributter, metoder og forhold til hinanden.

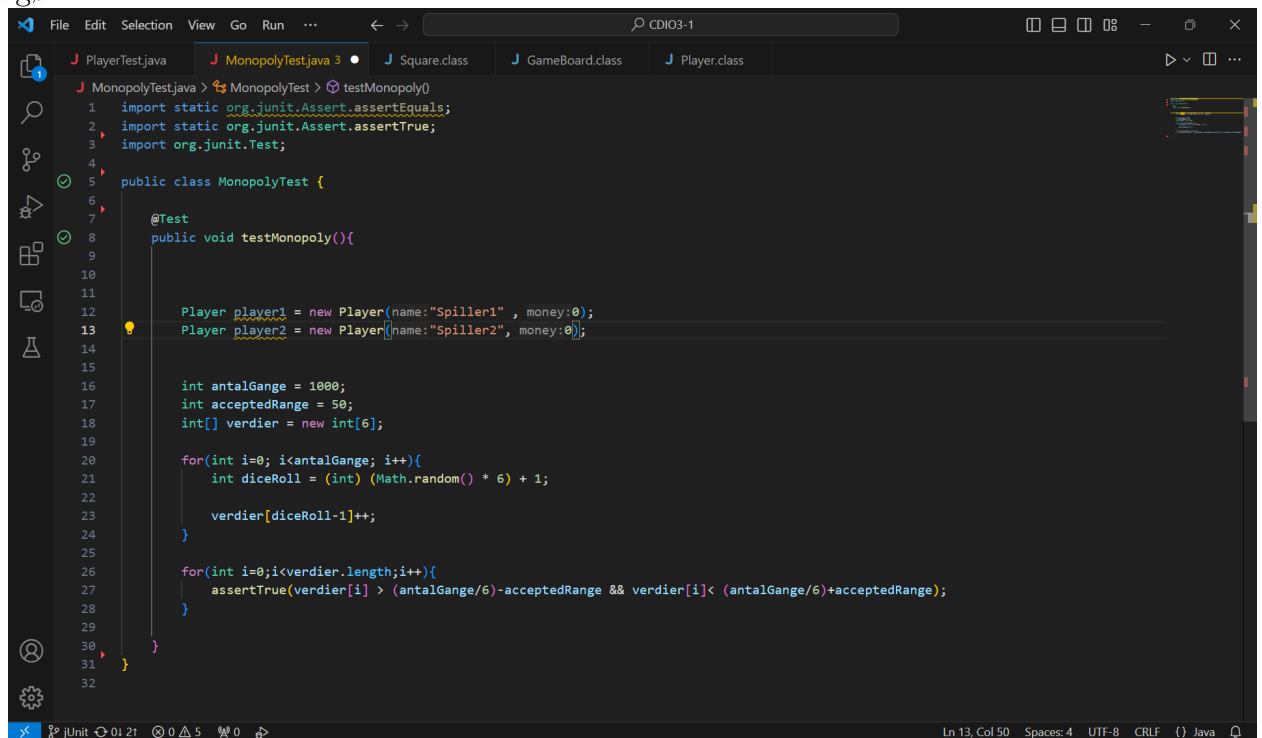
Som man kan se på diagrammet er Square-klassen en "parent class" for Player og GameBoard. Det er vist med "inheritance"pilen. Samt har Gameboard en "association"pil til player klassen gennem 'owner: player'. Vi har også en "composition"pil fra Monopoly til Square klassen, da Monopoly klassen ikke ville kunne stå alene, men eksisterer fordi Square klassen eksistere. Til den sidste klasse ChanceCard har vi valgt at bruge "association"pilen til/fra Monopoly, da Monopoly klassen skal trække fra ChanceCard, når der er blevet landet på et chancefelt.

Alt i alt giver det et bedre grafisk overblik over strukturen af vores kode og deres forhold til hinanden.

5 Testing

5.1 MonopolyTest:

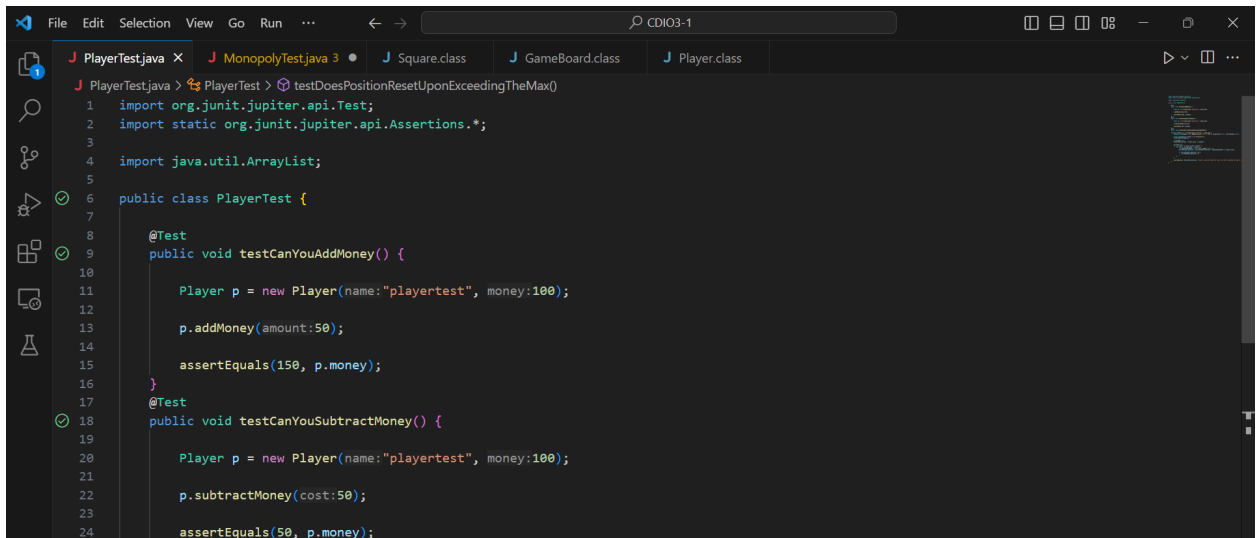
For at teste om vores kode virker, har vi lavet nogle forskellige unit tests, herunder PlayerTest.java og MonopolyTest.java. I MonopolyTest har vi testet, om terningerne virker som forventet, ved at simulere 1000 terningekast, ligesom når man spiller spillet. Testen slår med terningen 1000 gange og tæller hvor mange gange hver værdi fremstår. Vi har valgt, at 50 er en rimelig afvigelse, og tester derfor, om alle værdierne holder sig herunder, hvilket de også gør.



```
1 import static org.junit.Assert.assertEquals;
2 import static org.junit.Assert.assertTrue;
3 import org.junit.Test;
4
5 public class MonopolyTest {
6
7     @Test
8     public void testMonopoly(){
9
10
11
12         Player player1 = new Player(name:"Spiller1", money:0);
13         Player player2 = new Player(name:"Spiller2", money:0);
14
15
16         int antalGange = 1000;
17         int acceptedRange = 50;
18         int[] verdier = new int[6];
19
20         for(int i=0; i<antalGange; i++){
21             int diceRoll = (int) (Math.random() * 6) + 1;
22             verdier[diceRoll-1]++;
23         }
24
25         for(int i=0; i<verdier.length; i++){
26             assertTrue(verdier[i] > (antalGange/6)-acceptedRange && verdier[i] < (antalGange/6)+acceptedRange);
27         }
28     }
29 }
30
31
32
```

5.2 Playertest:

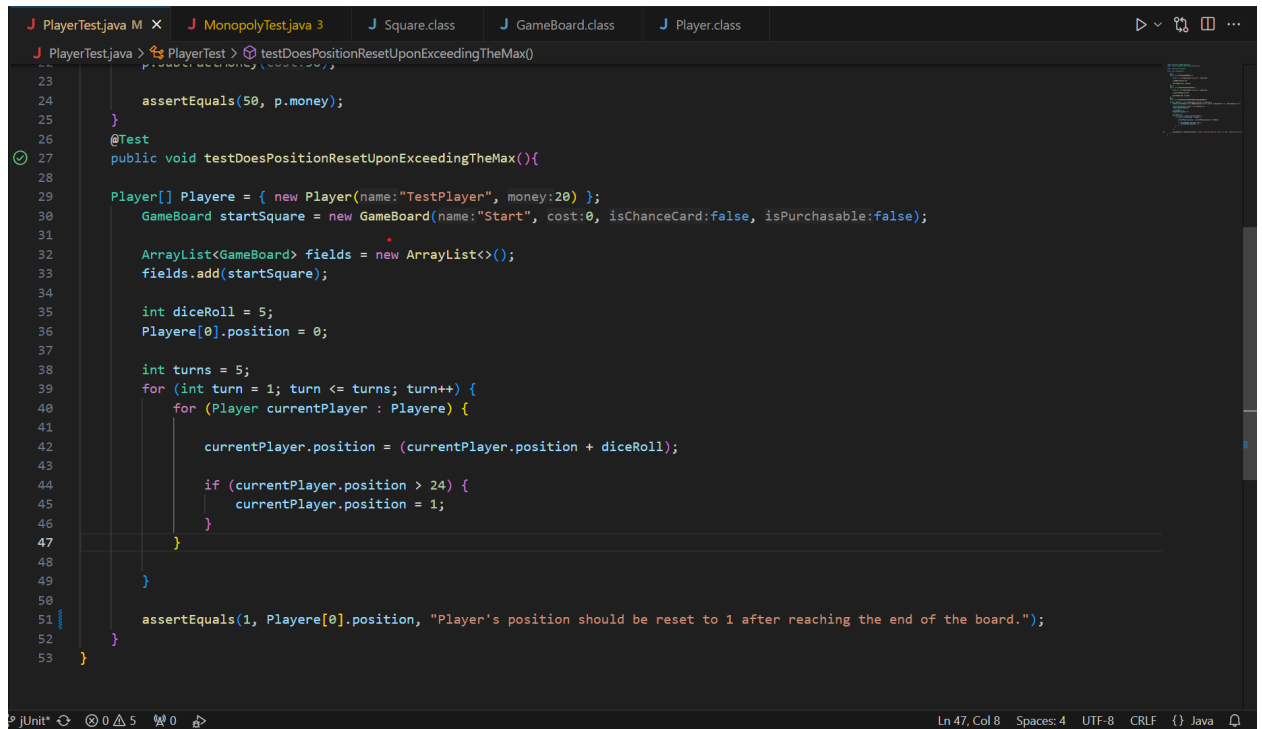
Classen PlayerTest indeholder 3 test, hvoraf 2 er simple test for at se, om vores AddMoney og SubtractMoney-metoder fungerer. Ved at lave et nyt objekt af classen Player, kan vi teste, om metoderne under denne, fungerer rigtigt og påvirker objektet som de skal.



```
1 import org.junit.jupiter.api.Test;
2 import static org.junit.jupiter.api.Assertions.*;
3
4 import java.util.ArrayList;
5
6 public class PlayerTest {
7
8     @Test
9     public void testCanYouAddMoney() {
10
11         Player p = new Player(name:"playertest", money:100);
12
13         p.addMoney(amount:50);
14
15         assertEquals(150, p.money);
16     }
17
18     @Test
19     public void testCanYouSubtractMoney() {
20
21         Player p = new Player(name:"playertest", money:100);
22
23         p.subtractMoney(cost:50);
24
25         assertEquals(50, p.money);
26     }
27 }
```

5.3 PositionTest:

Derudover har vi under PlayerTest lavet testDoesPositionResetUponExceedingTheMax, altså en PositionTest. Denne tester, om playerobjekters position sættes til nul, når man passerer det 24'ende og sidste feltt på spillepladen. Dette testes, ved at der skabes et nyt spillebræt, hvor der kun er start-feltet. Desuden skabes en spiller, hvis position er 0, men som i et for-loop rykker sig 5 pladser frem efter hvert slag. Efter det femte slag er positionen altså 25, hvilket går udover spillets rammer og derfor igen sættes til 1.



```
23     assertEquals(50, p.money);
24 }
25
26 @Test
27 public void testDoesPositionResetUponExceedingTheMax(){
28
29     Player[] Playere = { new Player(name:"TestPlayer", money:20) };
30     GameBoard startSquare = new GameBoard(name:"Start", cost:0, isChanceCard:false, isPurchasable:false);
31
32     ArrayList<GameBoard> fields = new ArrayList<>();
33     fields.add(startSquare);
34
35     int diceRoll = 5;
36     Playere[0].position = 0;
37
38     int turns = 5;
39     for (int turn = 1; turn <= turns; turn++) {
40         for (Player currentPlayer : Playere) {
41             currentPlayer.position = (currentPlayer.position + diceRoll);
42
43             if (currentPlayer.position > 24) {
44                 currentPlayer.position = 1;
45             }
46         }
47     }
48
49
50
51     assertEquals(1, Playere[0].position, "Player's position should be reset to 1 after reaching the end of the board.");
52 }
53 }
```

5.4 Test Cases

5.4.1 Player names

S.No	Action	Inputs	Expected Output	Actual Output	Test Browser/IDE	Test Result	Test Comments
1	Enter the numbers 2-4 and names with only letters from the alphabet	Number: 2 Name(s): Superman, Batman	"Velkommen til Monopoly!"	Welcome to Monopoly Jr.	Visual Studio Code	Pass	[Philip (dato)]: Success
2	Enter numbers out of 2-4 and special characters out of the alphabet	Number: 48 Names: Fred@#ik	Invalid number of players! Try again!	Invalid number of players! Try again!	Visual Studio Code	Pass	[Philip (dato)]: Success
3	Enter right number but instead of letters from the alphabet we enter numbers	Number: 4 Names: 0, 420, 578, 112	Names: 0, 420, 578, 112	Welcome to Monopoly Jr.	Visual Studio Code	Pass	Asger (21/11): Success
4	Enter right number but special characters	Number: 2 Names: %&//\, #####	Names: %&//\, #####	Welcome to Monopoly Jr.	Visual Studio Code	Pass	Asger (21/11): Success

Testcase: Køb af felter:

S.No	Action	Inputs	Expected Output	Actual Output	Test Browser/IDE	Test Result	Test Comments
1	Spiller 1 køber et felt og en anden spiller lander på det købte felt	Spiller 1 taster "y" og spiller 2 lander på det købte felt	This field is owned by "spiller 1". Paying rent: "cost of the field"	This field is owned by Spiller 1. Paying rent: \$2	Visual Studio Code	Pass	Asger (21/11) Succes
2	Spiller 1 har ikke nok penge, men prøver at købe feltet alligevel	Spiller 1 taster "y" for at købe felt med utilstrækkelige mængde penge	"Not enough money to buy the field"	"Not enough money to buy the field"	Visual Studio Code	Pass	Asger (21/11) Succes
3	Spiller 1 køber et felt og lander selv på det senere i spillet	Spiller 1 køber et felt og lander på det igen	"This field is owned by yourself. You can not buy this field"	"This field is owned by yourself. You can not buy this field"	Visual Studio Code	Pass	Asger (21/11) Succes

Test Case: Chance Felter.

S.No	Action	Inputs	Expected Output	Actual Output	Test Browser/IDE	Test Result	Test Comments
1	En spiller lander på et "Chance" felt og får penge fra chance kortene.	Spiller lander på chance felt	"Chance card: <u>chancecard</u> description + effect" og får tilføjet den korrekte mængde penge ind på konto	Chance card: an oil sheik from Saudi Arabia will sponsor your team. Receive 3. Spilleren fik også tilføjet 3 til sin "Money"	Visual Studio Code	Pass	Asger (21/11) Succes
2	En spiller lander på chance felt og skal rykke position	En spiller lander på chance felt	"Chance card: chance card description + effect" og rykker til det korrekte felt.	Chance card: Advance 5 spaces! Spilleren rykkede den korrekte mængde felter frem (5) fra den position han trak chance kortet	Visual Studio Code	Pass	Asger (21/11) Succes

5.5 Brugertest

Til at vurdere vores endelige spil, gjorde vi brug af en brugertest, hvor vi fik brugere uden programmeringserfaring til at teste og spille spillet. Der har vi opsat nogle spørgsmål som brugeren skulle svare på, hvilket gav os et indblik og en selvindsigt i, hvad der kunne forbedres i programmet, og på hvilke punkter. Heriblandt fik vi en feedback på at der manglede lidt flere outputs i forhold til hvad der skulle gøres, hvilket vi gjorde med det samme, og fik tilføjet flere outputs med hvad spilleren skal gøre ved hvert step.

Forstår du at købe eller ikke købe et felt?

Det forstår jeg godt. Jeg synes man bliver godt informeret om det.

Føler du spillet er brugervenligt, hvordan kunne det ellers gøres?

Det var egentlig super fint, jeg ved dog ikke hvordan det kunne skrives anderledes, men det der er nu virker fint . Det er svært at finde ud af det “design” (terminalen) spillet bliver spillet på, så det gør det lidt forvirrende.

Hvad synes du om at spille spillet?

Jeg har spillet sjovere spil, men det er fint, jeg ved hvad der sker ihvertfald.

Er det tydeligt at forstå, hvad spillet går ud på?

Ja, der er gode informationer

Hvad føler du, der mangler i spillet? Uddyb gerne med et eksempel fra din egen oplevelse

At chance kortene varierede lidt. Jeg føler, at jeg fik mange af de samme. Det er primært det visuelle i spillet som mangler. Det er ikke et særligt stort spil, så der er egentlig ikke så meget fylde på.

Hvordan har du det med at der ikke er et spillebræt?

Dårligt. Det påvirkede min oplevelse negativt. Jeg ville klart foretrække et spillebræt, og det ville helt sikkert gøre oplevelsen sjovere.

6 Configuration

Programmet kan køres med den seneste version af Java installeret, derfor er det et krav til vores program at styresystemet skal kunne køre med Java. Når man skal spille spillet kan det downloades fra vores github repository eller findes i den mappe der er lagt op på DTU Learn. Mappen G01_del3 skal derefter findes, hvor man så går ind på src mappen, hvor filen kan compiles med "javac MonopolyGame.java", efterfølgende skrives "java MonopolyGame", hvor spillet herefter påbegyndes.

7 Documentation

7.1 Hvad er arv?

Arv er et udtryk indenfor objektorienteret programmering, som beskriver, når classer overtager egenskaber fra andre klasser. I vores program har vi fx classen Monopoly, som har visse funktioner, der bruges af andre classes. Player-classen extender Monopoly, og arver altså fra den. I dette eksempel er Monopoly superclassen, og Player er en sub-class.

7.2 Hvad er en abstract class?

En abstract class er en klasse der for det første ikke kan stå alene, og har brug for nogle underklasser der extender den. Her bliver der defineret methods/constructors, som underklasserne skal implementere, for at abstract klassen kan gøre noget.

7.3 Hvad hedder det hvis alle fieldklasser har en landOnField metode der gør noget forskelligt?

Hvis man har et program hvor alle fieldklasser har én LandonField metode hedder polymorphism i objektorienteret programmering (OOP). Det refererer til muligheden for, at forskellige klasser kan have metoder med samme navn, men forskellige implementeringer.

7.4 GRASP

GRASP (General Responsibility Assignment Software Pattern) kan fortolkes som retningslinjer for tildeling af ansvar til klasser og objekter i objektorienteret design. I vores kode har vi også prøvet at opfylde disse retningslinjer.

7.4.1 Information Expert:

Vi observerede, at GameBoard-klassen effektivt håndterer købsprocessen, idet den har alle de nødvendige oplysninger om felterne. Dette er i overensstemmelse med Information Expert-princippet. Ligeledes er Player-klassen designet til at styre spillerens penge og position, da den besidder disse oplysninger.

7.4.2 Creator

I vores design har vi anvendt Creator-princippet, hvor MonopolyGame-klassen er ansvarlig for at oprette instanser af Player, GameBoard, og ChanceCard. Dette skaber en klar sammenhæng mellem klasserne og deres funktioner i spillet.

7.4.3 Controller

Vi har designet MonopolyGame-klassen til at fungere som en controller, der håndterer spillets "main-loop" og reagerer på spilhændelser, såsom terningekast, spillerture og chancekort effekter. Dette giver en centraliseret kontrol over spilflowet.

7.4.4 Low Coupling

Vores design fremviser en vis grad af lav kobling, især i forholdet mellem GameBoard, Player, og ChanceCard klasserne. Men vi erkender, at der er rum for forbedring, især i forhold til at mindske direkte interaktion mellem visse klasser. Her snakker vi bl.a. om den direkte manipulering af "player" attributten under GameBoard-klassen

7.4.5 High Cohesion

Vi har bestræbt os på at sikre, at hver klasse har et veldefineret og fokuseret ansvarsområde. Dette ses i den måde, Player håndterer spillerdata på, og hvordan GameBoard fokuserer på feltrelaterede operationer.

7.4.6 Polymorphism

Anvendelsen af abstrakte klasser og nedarvning, som ses med Square-klassen, er et eksempel på polymorfisme i vores design. Dette åbner for fleksibilitet og udvidelsesmuligheder i vores spilstruktur.

7.4.7 Protected Variations

Vi har forsøgt at beskytte vigtige dele af vores system mod ændringer i andre dele. Et eksempel er kapslingen af spillerdata i Player-klassen, hvilket isolerer disse data fra direkte ændringer i spillogikken.

8 Conclusion

Vi har valgt at arbejde således, at vi startede ud med at få et overblik over hvordan vi ville gå til projektet og opgaven. Her blev der også dannet et overblik over hvordan spillet skulle skrues sammen, og hvilke regler, fra det originale spil, vi ville beslutte os for at undlade. Vi ændrede i felternes egenskaber og navne, og har gjort det for at skabe et lidt anderledes spil, som i vores optik vil være mere spændende og tiltalende. De forskellige diagrammer spillede en vigtig rolle i at kunne få et overblik over hvordan spillet skulle bygges op, og gav os en lille ide om, hvor lang tid der skulle bruges på de forskellige dele af programmeringen. Ydermere var især brugertesten god til at give os en bedømmelse og selvindsigt i hvordan outputsene og spillets udseende var, i forhold til vores krav, som netop er at spillet skal være brugervenligt. Det var effektivt også i forhold til at træffe beslutninger i forhold til vores spil, heriblandt at fravælge et output af spillepladen, idet at det gjorde outputtet fra spillet mere uoverskueligt. Det resulterede i at vi lavede nogle ændringer, som til sidst udgjorde vores færdige spil, en modificeret version af Monopoly Game.