



Technical University
of Denmark

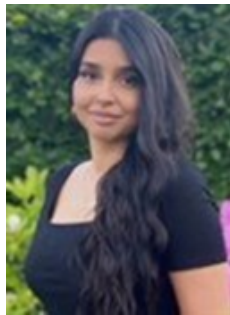
DANMARKS TEKNISKE UNIVERSITET

#62531 / #02314 / #62532

CDIO 3



Ali H. Y. Shanoof
s215716@dtu.dk
s215716



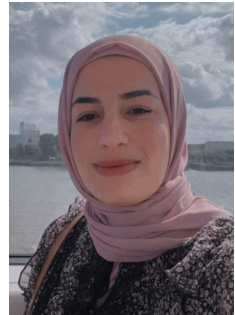
Anisa F. Riaz
s216237@dtu.dk
s216237



Berfin F. Turan
s215709@dtu.dk
s215709



Payam Madani
s192077@dtu.dk
s192077



Halah Ramadan
s193731@dtu.dk
s193731

Semester: E21
Gruppe: CDIO-09
Aflevering: 26-11-2021

Indholdsfortegnelse

1	Indledning	1
2	Kravspecifikation	1
2.1	Kravliste	2
2.2	Use case diagram	3
2.3	Use case beskrivelser	3
2.4	FURPS+	6
2.5	MoSCoW	7
3	Analyse	8
3.1	Domænemodel	8
3.2	Systemsekvensdiagram	9
4	Design	10
4.1	Designklassediagram	10
4.2	Sekvensdiagram	11
5	Implementering	12
5.1	playerTurn()	12
5.2	displayRandomChancecard()	12
5.3	gameFlow()	13
5.4	moveGUIPlayer()	13
5.5	buyField()	14
6	Dokumentation	15
6.1	Arv	15
6.2	Abstract	16
7	Test	17
7.1	Testcases	17
7.2	Brugertest	18
7.3	JUnit tests	19
8	Konklusion	20
9	Konfiguration	20

1 Indledning

I dette CDIO 3 projekt har vi arbejdet tværfagligt med fagene Versionsstyring og testmetoder, Udviklingsmetoder til IT-systemer & Indledende programmering. Denne gang har fokusset sat spids på sammensætningen af hele spillet ved brug af GUI. I projektet er der specifikt gjort brug af diverse artefakter, så som kravliste, use case, Domænemodel, system sekvensdiagram, sekvensdiagram samt design klassediagram. Artefakterne i dette projekt har dannet os et overblik over systemet, der gjort det tydeligt samt håndgribeligt for os at udvikle programmet - hvilket har medført til en bedre forståelse af dette projekt.

2 Kravspecifikation

I dette afsnit, tages der udgangspunkt i kravene for dannelsen af softwareprogrammet "Monopoly Junior". Herunder sættes der fokus på krav-dannelsen af diverse specifikationer og spillets behov, hvor der opstilles en kravliste med disse specifikationer. Kravlisten opstilles vha. den udleveret opgavebeskrivelse. Ud fra kravlisten, opstår muligheden for at afdække kravene, ved at opstille og anvende brugsmønstre teknikken, nemlig også kendt for "Use case". Ved hjælp af denne teknik, dannes der en use case diagram, som har til formål at give en opsummering af detaljer om systemet og brugerne i det system. Herunder dannes en use case beskrivelse. Ud fra use case beskrivelsen sorteres de forskellige krav vha. FURPS+ modellen, og dermed prioriteres ud fra MoSCoW metoden.

2.1 Kravliste

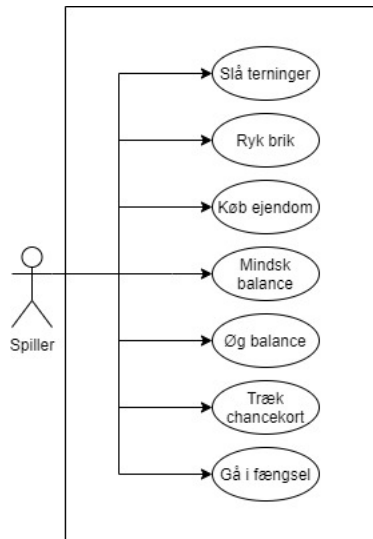
Nedenstående ses en kravliste, som er blevet opstillet ud fra den udleveret opgavebeskrivelse. Denne kravliste giver os et overblik over de diverse krav der tages udgangspunkt i dette spil. (Se Tabel 1)

ID	Krav Beskrivelse
K01	Programmet skal skrives i programmeringssproget Java.
K02	Programmet skal let kunne oversættes til andre sprog.
K03	Programmets kode skal være fleksibelt ifm. objekter.
K04	Spillet skal kunne spilles af 2 til 4 personer.
K05	Programmet skal kunne køres på DTU's databaser.
K06	Ingen bemærkelsesværdige forsinkelser.
K07	Der skal udarbejdes et overblik over programmets udvikling.
K08	Spillet skal indeholde to seks-sidet terninger.
K09	Spillet skal inkludere en GUI.
K10	Spillerne skal kunne købe grunde.
K11	Der skal være forskellige roller i spillet - hhv. en banker rolle og en alm. spiller rolle.
K12	Spillerene skal have en startbalance; 16 "dollars" hvis 4-spillere, 18 hvis 3-spillere og 20 hvis 2-spillere
K13	Der skal kunne skelnes mellem spillerne på GUI'en.
K14	Spiller med den højeste terningslag starter.
K15	Spillerens position skal kunne ændres ud fra terningslagenes resultat.
K16	Hvert felt skal have sin egne design, værdi og beskrivelse.
K17	Spillet skal indeholde chancekort.
K18	Der skal kunne hæves og indsættes penge på en spillers konto.
K19	En spiller skal kunne lande i fængsel.

Tabel 1: Liste over krav til projektet

2.2 Use case diagram

Nedstående ses en use case diagram som har til formål at give et overblik over hvordan spillets main use cases benyttes i systemet. Herunder ses en primær aktør som er en rolle i spillets system. Aktøren er 'spiller' og spillerens funktionaliteter i Monopoly Junior vises nedenunder:



Figur 1: Use Case diagram

2.3 Use case beskrivelser

For et grundigt indblik i kundens ønsker og forventninger, har man valgt at sortere og validere de forskellige krav, ved at benytte FURPS+ modellen som ses nedenstående i Tabel 1. Derudover beskrives de enkelte use cases som brief som ses nedenfor og dermed opstilles en fully dressed af main use casen.

2.3.1 Brief beskrivelser af use cases

Slå terning

Slå terninger funktionen i spillet tager udgangspunkt i at en spiller slår to terninger og dækker over funktionaliteten om at give nogle random værdier fra 2 til 12. Spillet beregner herefter terningernes samlede FaceValue.

Ryk brik

Ryk brik funktionen i spillet, anvender terningernes FaceValue (værdi), og rykker herefter spilleren på et af de 24 felter på gameboardet. Ryk brik dækker over metoden for bevægelsen af spillerens brik, som hænger sammen med slå terninger metoden. Når der bliver slået med terningerne, rykkes brikkerne dermed også.

Køb ejendom

Køb ejendom metoden går ud på at, når en spiller lander på 'GUI.Street' felterne. Kan spilleren enten vælge at trykke 'ja' for køb af ejendom, eller trykke 'nej' for fravælgelse køb af ejendom.

Mindske balance

I denne metode mindskes spillerens balance ved køb af et 'GUI.Street' felt. Derimod mindskes spillerens balance også hvis spilleren lander på en af medspillerens felter. Balancen mindskes også hvis spilleren trækker et chancekort, hvor de skal betale enten en medspiller eller banken.

Øg balance

I øg balance metoden, øges spillerens balance når de passerer start feltet, eller når en anden medspiller lander på et af de felter spilleren ejer. Balancen kan også øges, hvis spilleren trækker et chancekort, hvor de skal passere start.

Træk chancekort

Hvis spilleren lander på 1 ud af de 4 grå spørgsmåltegns felter på gameboardet. Modtager spilleren et tilfældigt chancekort ud af de 6 chancekort der findes i spillet. Chancekortene i spillet varierer meget fra hinanden, så spilleren kan enten være heldig eller uheldig.

Gå i fængsel

Gå i fængsel funktionen er som udgangspunkt ikke blevet implementeret ud fra de essentielle regler, dog virker den mere som et helligsted.

2.3.2 Fully dressed

Nedenstående vises vores fully dressed af vores use case ryk brik.

Fully dressed af Ryk brik	
Use case:	<ul style="list-style-type: none">• Ryk brik
Primær aktør:	<ul style="list-style-type: none">• Spiller
Stakeholders og interests:	<ul style="list-style-type: none">• Kunde
Preconditions:	<ul style="list-style-type: none">• Spilleren slår med begge terninger. systemet beregner resultatet og spilleren rykkes frem på en af de 24 felter.
Main flow:	<ol style="list-style-type: none">1. Spiller 1. kaster med begge terninger2. Systemet beregner terningernes samlet resultat.3. Spillerens brik rykkes frem på feltet4. Når spilleren rykkes frem på feltet, kan balancen enten mindskes eller forblive den samme.<ul style="list-style-type: none">- Købe/ betale husleje til ejendom- Trække et chancekort5. Spillerens balance ændres.6. Spiller 2-4 gentager.7. Brikkerne rykkes frem indtil en af spillerne har tabt.
Postconditions:	<ul style="list-style-type: none">• Spilleren er rykket frem på feltet, og ovenstående gentages indtil spillet er afsluttet.

Figur 2: Fully dressed

2.4 FURPS+

Nedenstående, har vi sorteret vores kravliste ind i en FURPS+ model, for at påvise dens effekt i hver af de kategoriseret beskrivelser.

Functionality	Usability	Reliability	Perfomance	Supportability
K01	K04		K06	K02
K03	K07			K03
K04				K05
K08				
K09				
K10				
K11				
K12				
K14	K13			
K15				
K16				
K17				
K18				
K19				

Tabel 2: Krav sorteret ud fra FURPS+

På baggrund af FURPS+ metoden kan man hermed anvende MoSCoW metoden som har til formål at prioritere kravne vha. fire kategorier af initiativer; Must have, Should have, Could have og Won't have.

2.5 MoSCoW

Nedenstående ses anvendelsen af MoSCoW.

ID	MoSCoW
K01	Must have
K02	Could have
K03	Must have
K04	Must have
K05	Must have
K06	Must have
K07	Must have
K08	Must have
K09	Must have
K10	Should have
K11	Could have
K12	Must have
K13	Must have
K14	Could have
K15	Must have
K16	Must have
K17	Should have
K18	Should have
K19	Should have

Tabel 3: Kravene fra kravlisten prioriteret vba. MoSCoW.

3 Analyse

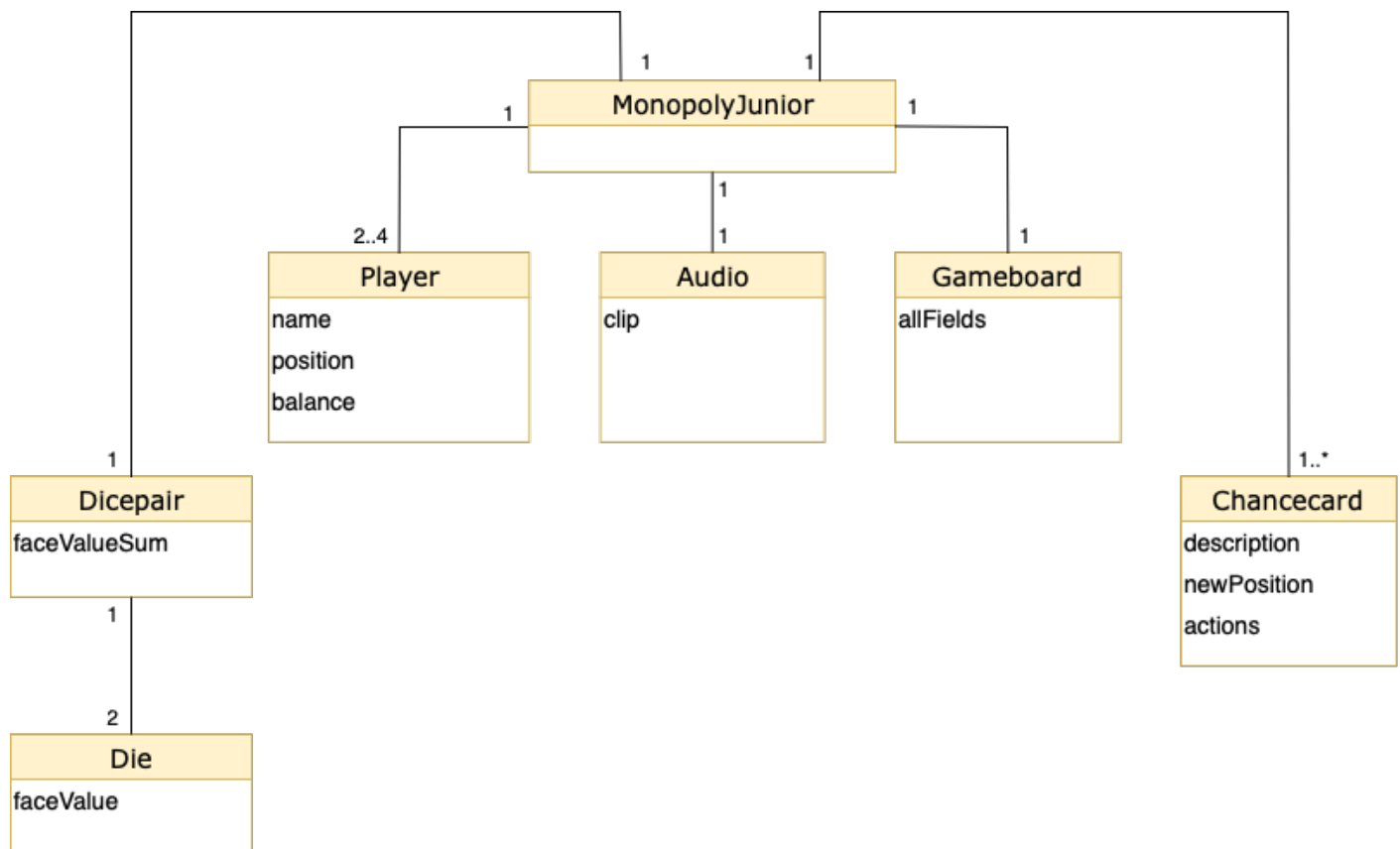
Under analysefasen af projektet blev der bl.a. udarbejdet en domænemodel, for at få et overblik over, hvad der skulle arbejdes med under implementeringsfasen. Der blev derudover udarbejdet et systemsekvensdiagram, som har til formål at vise hvordan en aktør interagerer med systemet ud fra en bestemt udvalgt use-case.

3.1 Domænemodel

Nedenstående kan den udarbejdede domænemodel ses.

Domænemodellen viser de klasser, som projektet skal indeholde samt deres attributter. Derudover viser domænemodellen multiplicitet, som siger noget om, hvad relationen klasserne er.

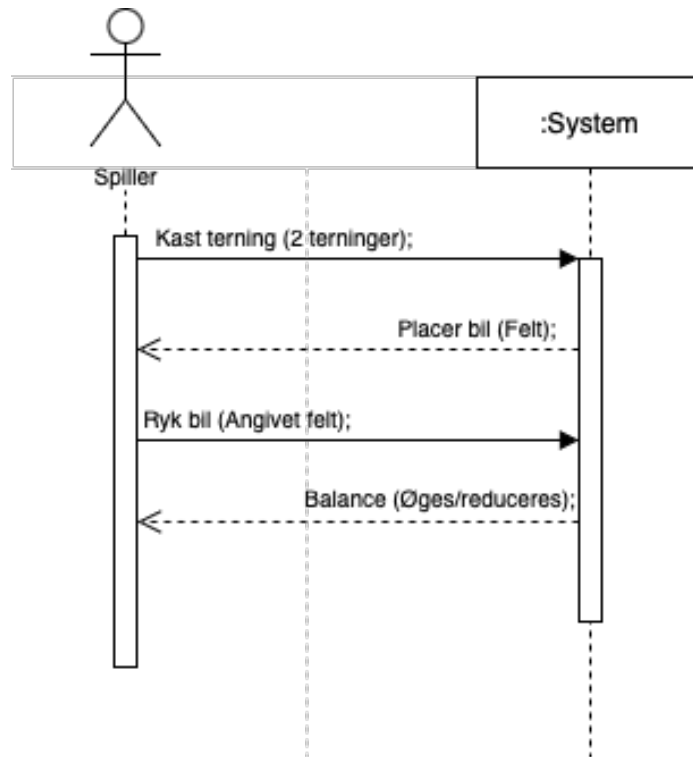
De nødvendige klasser blev fundet under kravspecifikationsfasen. Det kan ses på modellen, at MonopolyJunior har flest koblinger. Dette er grundet mængden af ansvar, som klassen har. MonopolyJunior klassen er den klasse, som opretter objekter af de andre klasser og dermed samler al information og logik, for at spillets funktionaliteter virker.



Figur 3: Udarbejdet domænemodel for projektet

3.2 Systemsekvensdiagram

På figur 3.2 er et System sekvensdiagram illustreret. Diagrammet har til formål at danne et overblik af en aktør, kaldt *Spiller* integration med systemet. Spilleren starter selve spillet ved at kaste 2 terninger, hvor systemet giver et modsvar udefra terningekastet, at placere bilen til det givet felt der stemmer overens med terningenssummen. Derfra rykker spilleren bilen til det givet felt, hvor systemet udregner spillerens kontobalancen, der kan afgøres negativt eller positivt.



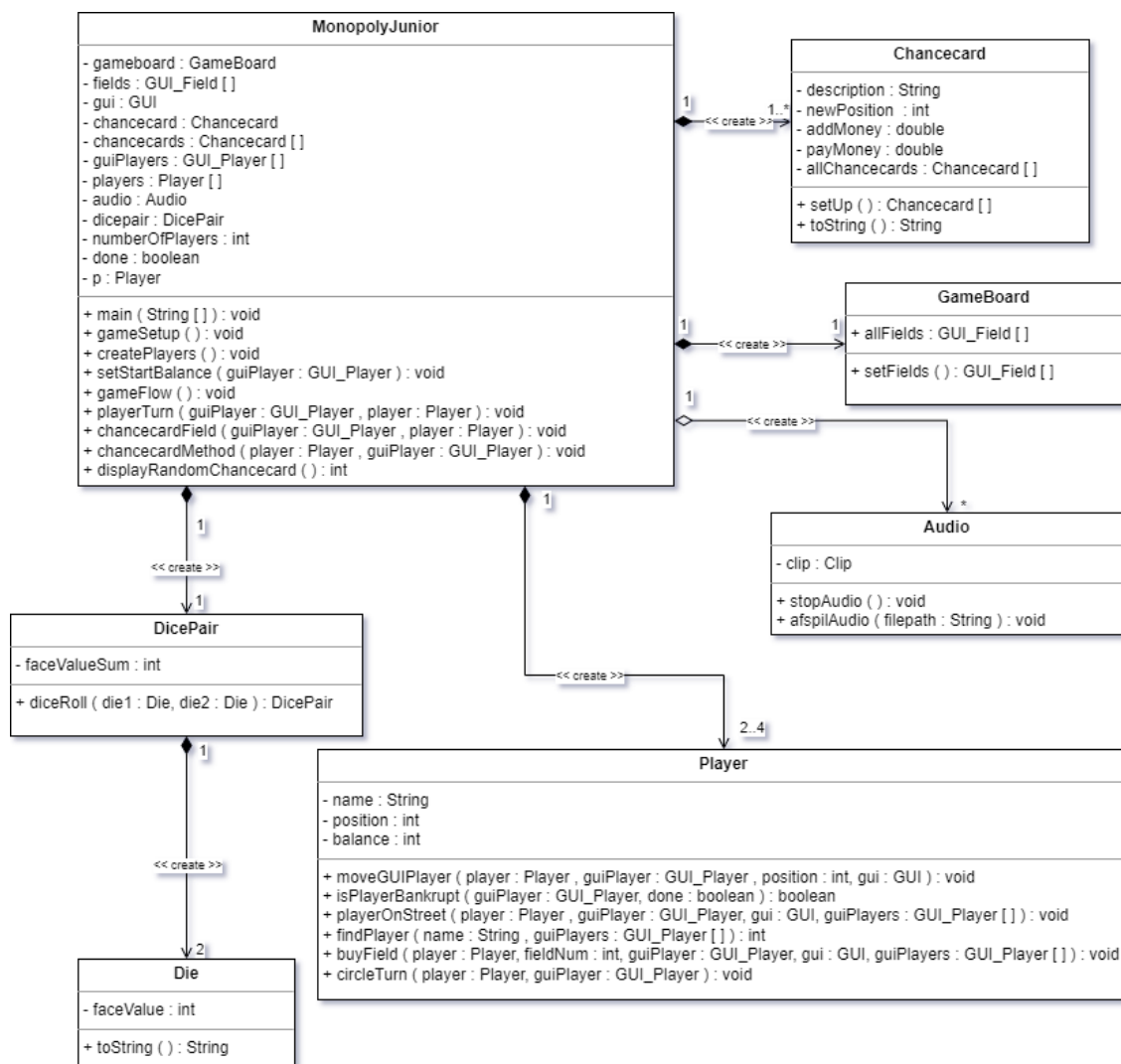
Figur 4: Udarbejdet systemsekvensdiagram for projektet

4 Design

Under designfasen af processen blev der udarbejdet hhv. et designklassediagram over systemet samt et sekvensdiagram for metoden `playerTurn()`. Disse anvendte design metoder er med til at give kunden et overordnet overblik over softwareprogrammets system, design og opbyggelsen af spillet, hvilket er essentielt at have med.

4.1 Designklassediagram

Designklassediagrammet er en detaljeret version af figur 3.1. På designklassediagrammet vises attributterne og metodernes access modifiers, typer samt evt. givne værdier. Desuden fokuseres der yderligere på associationerne mellem klasserne - herunder typen af associationer.



Figur 5: Designklassediagram over systemet

Der er udover en alm. associationspil bl.a. gjort brug af composition, for at beskrive klassernes samspil.

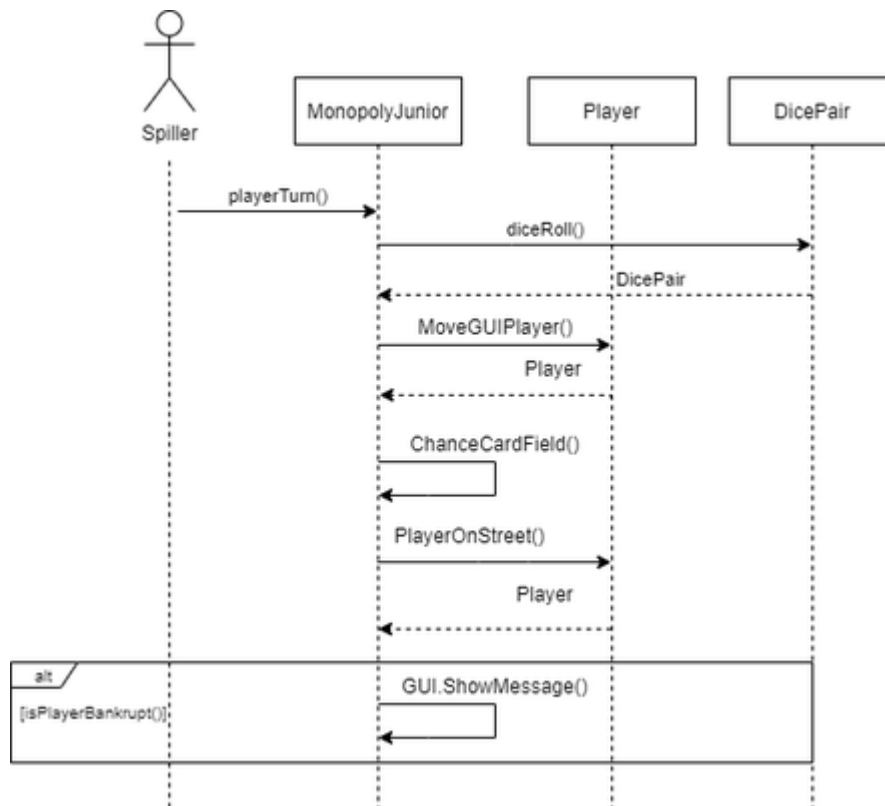
Designklassediagrammet giver et bedre overblik over de forskellige klassers indhold, samspil med andre klasser, samt ansvar.

I projektet har gruppen forsøgt at overholde GRASP ift. lav kobling og høj samhørighed så meget som muligt. Dette kan eksempelvis ses ved at der er forsøgt at dele ansvarsområder op så meget som muligt, så flere af klasserne ikke har mere end en association til andre klasser.

MonopolyJunior klassen har i dette projekt det største ansvar. Denne klasse instantierer mange andre, hvilket også betyder, at der gøres stor brug af composition ifb. med denne. Et gameboard ville eksempelvis ikke kunne eksistere uden MonopolyJunior klassens instantiering af GameBoard klassen for at oprette et objekt.

4.2 Sekvensdiagram

På figur 4.2 er et sekvensdiagram illustreret. Metoden `playerTurn()` er blevet valgt som værende et af de vigtigere metoder, da det er denne som sørger for, at funktionerne kan virke korrekt for de to til fire spillere. Denne metode gør desuden også brug af en række andre metoder i klassen, som sørger for, at spiller virker, som det skal. Dette gør `playerTurn` metoden til et af de mest centrale metoder i projektet.



Figur 6: Sekvensdiagram over metoden `playerTurn()`

5 Implementering

Del 3 af CDIO-projektet kan opsummeres som et MonopolyJunior spil, der involverer 2 - 4 spiller-objekter, to terninger, 24-feltobjekter inklusive beskrivelser, 6-chancekortobjekter samt tilhørende klasser. Terningerne skal lande på et tal mellem to og tolv tilfældigt, hvilket klares ved hjælp af matematikklassen. Spillere skal også registrere deres egne konti for at holde styr på deres spille-saldo, som kan påvirkes både positivt og negativt i spillets forløb. Derudover består spillet af et antal feltobjekter, som hver har sit eget ID, navn, indflydelse på førnævnte balance, position og beskrivelse, som tidligere nævnt. Sammenhængen mellem de forskellige metoder og objekter er kort beskrevet i fem listinger af den anvendte kode nedenfor.

5.1 playerTurn()

Herunder ses den første listing, Listing 1 som består af metoden playerTurn(). Denne metode har til formål at drive spillets bevægelse, spillernes position og spillernes funktioner. Denne metode er et af de mest centrale metoder som har diverse funktioner. Når terningerne smides, dannes der en timeout/pause(TimeUnit) på 0.5 sekunder mellem de slået terninger og bevægelsen af spillerens brik, hvor brikken rykker efter de 0.5 sek: (player.getPosition). Derudover dækker denne metode også over muligheden for at en spiller kan købe en ejendom, når en spiller lander på en GUI.street/ejendom (buyFields), og hvis en af de to til fire spillere ikke længere har positiv balance eller nul i balancekontoen (If(isPlayerBankrupt)) så slutter spillet (gui.showMessage...)

```
1 public void playerTurn(GUI_Player guiPlayer, Player player) throws
   InterruptedException {
2     dicePair.diceRoll(die1, die2);
3     gui.setDice(die1.getFaceValue(), die2.getFaceValue())
4     TimeUnit.MILLISECONDS.sleep(500);
5     moveGUIPlayer(player, guiPlayer, dicePair.getFaceValueSum()+player.getPosition())
6     chancecardField(player, guiPlayer);
7     buyFields(player, guiPlayer);
8     if(isPlayerBankrupt(guiPlayer)) gui.showMessage(...); }
```

Listing 1: Metode til spiller tur og bevægelse

5.2 displayRandomChancecard()

På nedenstående listing 2, vises metoden displayRandomChancecard som gør at en spiller kan trække nummer fra chancecard. På linje et kan man se, at der oprettes en variable randomNum der sættes lig med 0. Fra linje tre til syv anvendes en forloop, fordi den er mere kompakt at have med, især når man bruger tællevariable. Linje fire kan man se, at programmet returnerer et tilfældigt heltal. Math.random() metoden bliver brugt til at generere et tilfældigt randomNum.

```
1 public int displayRandomChancecard() { int randomNum = 0;
2     for (int i = 0; i < chancecards.length ; i++) {
3         randomNum = (int)(Math.random()*chancecards.length);
4         gui.setChanceCard(chancecards[randomNum].getDescription());
5         gui.displayChanceCard(); } return randomNum; }
```

Listing 2: Metode til at vise tilfældig chancekort

5.3 gameFlow()

På nedenstående listing 2 fremvises metoden gameFlow(). Den offentlige metode, GameFlow indeholder en while loop som tjekker om ikke 'done' er gældende, hvis dette er tilfældet vil der herefter tjekkes om spilleren er gået fallit. Første statement undersøger om PlayerTurn er større eller lig med antallet af spillere, hvis dette er sandt vil slå terningeknappen vises på GUI'en. I andet if statement tjekkes der efter om terningeknappen bliver trykket på, hvis dette er sandt indekseres PlayerTurn og næste spiller får sin tur. I det sidste statement, vises metoden for en spiller der er gået fallit, og spillet vil derefter afsluttes.

```
1  public void gameFlow() throws InterruptedException {
2      int playerTurn = 0;
3      while(!done) {
4          isPlayerBankrupt(guiPlayers.get(playerTurn));
5
6          if (playerTurn >= numberOfPlayers) playerTurn = 0;
7          String button = gui.getUserButtonPressed("Tryk på knappen, for at slå
8              å, " + guiPlayers.get(playerTurn).getName() , "slå");
9          if (button.equals("slå")) {
10              playerTurn(guiPlayers.get(playerTurn), players.get(playerTurn));
11              playerTurn++;
12          }
13          if (playerTurn >= numberOfPlayers) playerTurn = 0;
14      }
15      guiPlayers.get(playerTurn - 1).setBalance(0);
16      String input = gui.getUserButtonPressed("Spillet er slut", "OK");
17      if (input.equals("OK")) System.exit(0);
18  }
```

Listing 3: Metode til at vælge tur

5.4 moveGUIPlayer()

På nedenstående listing beskriver metoden kort og simpelt, hvordan vi får en spiller til at rykke til en given position, ved brug af *setCar*. *player.setPosition(position)* returnerer true så vores spillers bil, kan komme frem til den position uden at efterlade en trace

```
1  public void moveGUIPlayer(Player player, GUI_Player guiPlayer, int position,
2      GUI gui) {
3      gui.getFields()[player.getPosition()].setCar(guiPlayer, false);
4      player.setPosition(position);
5      circleTurn(player, guiPlayer);
6      gui.getFields()[player.getPosition()].setCar(guiPlayer, true);
7  }
```

Listing 4: Metode til at rykke spilleren på GUI'en

5.5 buyField()

På nedenstående listing kan metoden, som sørger for at en spiller kan købe et felt, ses. Metoden tjekker om feltet, som spilleren er landet på, er ejet. Hvis feltets *.getOwnerName()* metode ikke har en værdi, vil spilleren blive spurgt vba. en GUI metode, om spilleren vil købe feltet. Dette sker fra linje to til fire.

Fra linje fem til elleve ses koden som sørger for hvad der sker, hvis spilleren klikker ”Ja” til at købe et felt. Linje tolv og tretten sørger for, at der ikke sker noget med feltet, hvis spilleren klikker ”Nej”. Hvis feltets *.getOwnerName()* har en værdi, vil spilleren blive tvunget til at betale leje til den spiller med det navn, som er lig med den værdi, som *.getOwnerName()* returnerer.

Metoden er tre-og-tredive linjer lang, hvilket betyder, at den har mere ansvar, end andre mindre metoder. *buyField()* metoden ville kunne deles op i flere ansvarsområder, hvilket ville understøtte SoC konceptet, som er taget i brug af gruppen. Dette har der dog ikke været mulighed for i dette projekt, grundet ressourcemangel.

```
1  public void buyField(Player player, int fieldNum, GUI_Player guiPlayer, GUI gui
    , GUI_Player[] guiPlayers) {
2      if ( ((GUI_Ownable) gui.getFields()[fieldNum]).getOwnerName() == null) {
3          if (player.getPosition() == fieldNum) {
4              String button = gui.getUserButtonPressed("Vil du købe dette felt
                    ?", "Ja", "Nej");
5              if (button.equals("Ja")) {
6                  ((GUI_Ownable) gui.getFields()[fieldNum]).setOwnerName(
                        player.getName());
7                  ((GUI_Ownable) gui.getFields()[fieldNum]).setBorder(
                        guiPlayer.getPrimaryColor());
8                  if(fieldNum == 1 || fieldNum == 2 || fieldNum == 4 ||
                        fieldNum == 5)
9                      guiPlayer.setBalance(guiPlayer.getBalance() - 1);
10                 ... }
11                 if (button.equals("Nej")) {
12                     gui.showMessageDialog("okay");
13                 }
14             }
15         } else if (((GUI_Ownable) gui.getFields()[fieldNum]).getOwnerName() !=
            null && !((GUI_Ownable) gui.getFields()[fieldNum]).getOwnerName().
            equals(player.getName())){
16             if (player.getPosition() == fieldNum) {
17                 gui.showMessageDialog("Du skal betale lejen af dette felt.");
18                 String rent = ((GUI_Ownable) gui.getFields()[fieldNum]).getRent
                    ();
19                 guiPlayer.setBalance(guiPlayer.getBalance()-Integer.parseInt(
                        rent));
20                 int index = findPlayer(((GUI_Ownable) gui.getFields()[fieldNum])
                        .getOwnerName(), guiPlayers);
21
22                 guiPlayers[index].setBalance(guiPlayers[index].getBalance()+
                        Integer.parseInt(rent)); }
23         } else {
24             gui.showMessageDialog("Du er landet på et felt, som du ejer, så der er
                    intet at foretage her:"); }
25     }
```

Listing 5: Metode til at kunne købe et felt

6 Dokumentation

I dette afsnit beskrives koncepter i Java såsom arv samt abstract. Indholdet af dette afsnit skulle ydermere bestå af dokumentation for overholdelse af GRASP samt dokumentation for tests. I stedet har det gruppen besluttet at lave et separat afsnit til tests og dokumentationen af disse samt besluttet at beskrive brugen af GRASP i design afsnittet.

6.1 Arv

Arv er et element i Java, hvor en klasse kan nedarve de egenskaber, en anden klasse har. Objektorienteret programmering gør det muligt at definere nye klasser fra eksisterende klasser. Med arv kan man definere en generel klasse (superklasse) og bagefter udvide den til en mere specialiseret klasse (subklasse). Klassen hvis funktioner der nedarves fra, er kendt som en super klasse (eller parent klasse). Sub klassen er klassen der nedarver fra superklassen (child klassen). Subklassen kan nedarve fra Superklassen vba. keywordet *extends*. Et eksempel på dette kan ses på nedenstående listing.

```
1  public final class GUI_Jail extends GUI_Field {
2      ...
3  }
```

Listing 6: GUI_Jail klasse udsnit

Vi kan altså se på ovenstående eksempel, at klassen *GUI_Jail* arver *GUI_Field* klassens metoder. Arv og access modifiers har også et vigtigt forhold. Afhængigt af access modifieren for variabler og metoder, kan man bestemme hvad der skal nedarves og hvad som ikke skal:

Private: Nedarves ikke af subklasser

Default: Nedarves af subklasser som ligger i samme pakke som superklassen

Protected: Nedarves altid af subklasser

Public: Nedarves altid af subklasser

For at kalde på en superklassens konstruktør kan keywordet *super* bruges. Et eksempel på dette kan ses på nedenstående listing.

```
1  public GUI_Jail(){
2      this(PICTURE, TITLE, SUBTEXT, DESCRIPTION, new Color(125, 125, 125),
          Color.BLACK);
3  }
4  public GUI_Jail(String picture, String title, String subText, String
    description, Color bgColor, Color fgColor){
5      super(bgColor, fgColor, title, subText, description);
6  }
```

Listing 7: GUI_Jail klasse udsnit

6.2 Abstract

Abstrakte klasser, er klasser der er så abstrakte/generelle, at de ikke kan anvendes til at danne specifikke instanser.

Denne klasse designes for at nedarves af subklasser, som enten implementerer eller overrider den abstrakte klasses metoder.

Yderligere gælder det for abstrakte klasser at man ikke kan lave instanser af den (dvs. ingen new operator).

Klasser og metoder, som er abstrakte skal noteres med `abstract`. Et eksempel på dette kan ses på nedenstående listing.

```
1 public abstract class GUI_Ownable extends GUI_Field {  
2     ...  
3 }
```

Listing 8: GUI.Ownable klasse udsnit

En abstrakt klasse kan både indeholde metoder som er abstrakte og metoder som ikke er abstrakte, attributter og konstruktører. Konstruktører i en abstrakt klasse kan anvendes i de tilfælde, hvor en abstrakt klasse har attributter der skal instantieres (man ville derved anvende `super(...)` til at kalde konstruktøren).

Yderligere gælder det for en abstrakt metode, at de skal overrides af alle konkrete klasser der nedarver fra den abstrakte superklasse, udover abstrakte klasser som nedarver fra andre abstrakte klasser. Overriding af metoder betyder, at man i subklassen modificerer implementeringen af en metode der er defineret i superklassen. Detter er eksempelvis hvad der ville gøres brug af, hvis alle Field klasserne har en metode, som hedder `landOnField`, men at de gør forskellige ting.

Dokumentationen for tests ses i nedenstående afsnit 7.

7 Test

For at sikre projektets fremgang for et Monopoly Junior spil, har vi foretaget lang række test, der sikrer os, at projektet leverer til kundens forventning. Disse tests kan omdøbes i TestCases der fortæller en forventet status på projektets fremdrift. Dette kan også beskrives under Brugertest, for at give en konkret forståelse for brugervenligheden for programmet. Dette bliver til sidst pålagt i en JUnit Test.

7.1 Testcases

På nedenstående tabel, fremgår de diverse testcases. Der er blevet udarbejdet en testcase for DicePair, PlayerTest, GameBoard samt MonopolyJunior klassen. Disse test er foretaget for at sikre projektets fremdrift virker optimalt.

Test Case Type	Beskrivelse	Test Step	Forventet Resultat	Resultat
1	Kør Klassen	DicePairTest.java	DicePairTest.java	DicePairTest.java
2	Kast terning for at rykke til given felt.	Tryk "Slå terning"	Summen af Terning t1: 4 t2: 3	Pass; Sum: 7
3	Kast terning for at rykke til given felt	Tryk "Slå terning"	Summen af Terning t1: 2 t2: 6	Fail; Sum: 7

Test Case Type	Beskrivelse	Test Step	Forventet resultat	Resultat
1	Kør Klassen	PlayerTest.java	PlayerTest.java	PlayerTest.java
2	Indsæt følgende navn til profilen	Skriv dit navn på linjeteksten	Navn: Popsi	Pass; Name: Popsi
3	Indsæt følgende navn til profilen	Skriv dit navn på linjeteksten	Navn: Jens Hansen	Fail; Name: Jens Hansen

Test Case Type	Beskrivelse	Test Step	Forventet resultat	Resultat
1	Kør Klassen	GameBoard.java	GameBoard.java	GameBoard.java
2	Kontrol af felternes placering i spillet	Kør Gameboard Test klassen	Placering af felterne via array-listerne er korrekte	Pass
3	Kontrol af felternes placering i spillet	Kør Gameboard Test klassen	Fejl i arraylist	Fail

Test Case Type	Beskrivelse	Test Step	Forventet resultat	Resultat
1	Kør Klassen	MonopolyJunior.java	MonopolyJunior.java	MonopolyJunior.java
2	Køb grund eller spring over	Tryk mellem "Køb" eller "Spring over"	Spilleren får muligheden for at købe eller springe over	Spilleren ejer enten grunden eller har hoppet videre til næste given felt
3	Betal husleje	Reduces automatisk	Spiller rykker til et ejet grund	Spillerens balance reduceres for det given ejet felt

7.2 Brugertest

I dette projekt har brugen af en Grafisk Brugergrænseflade (GUI) været et krav. Dette betyder, at brugervenlighed er blevet vægtet højere end før af gruppen. For at teste brugervenligheden af de designvalg, som er blevet foretaget af gruppen, er brugertests en god metode at tage i brug.

To personer uden erfaring med programmering, softwareudvikling eller brug af en IDE blev udvalgt til brugertest.

Den første person er Carolina Georgotas, 20 år, studerende. Carolina mente at spillet var meget intuitivt, brugervenligt og at det mindede om de [funktioner] det rigtige Monopoly Junior spil har. Carolina tilføjede, at designvalget om baggrundsmusik var en god detalje, men at få ting afvigede fra det rigtige Monopoly Junior spil; såsom hvem der starter spillet.

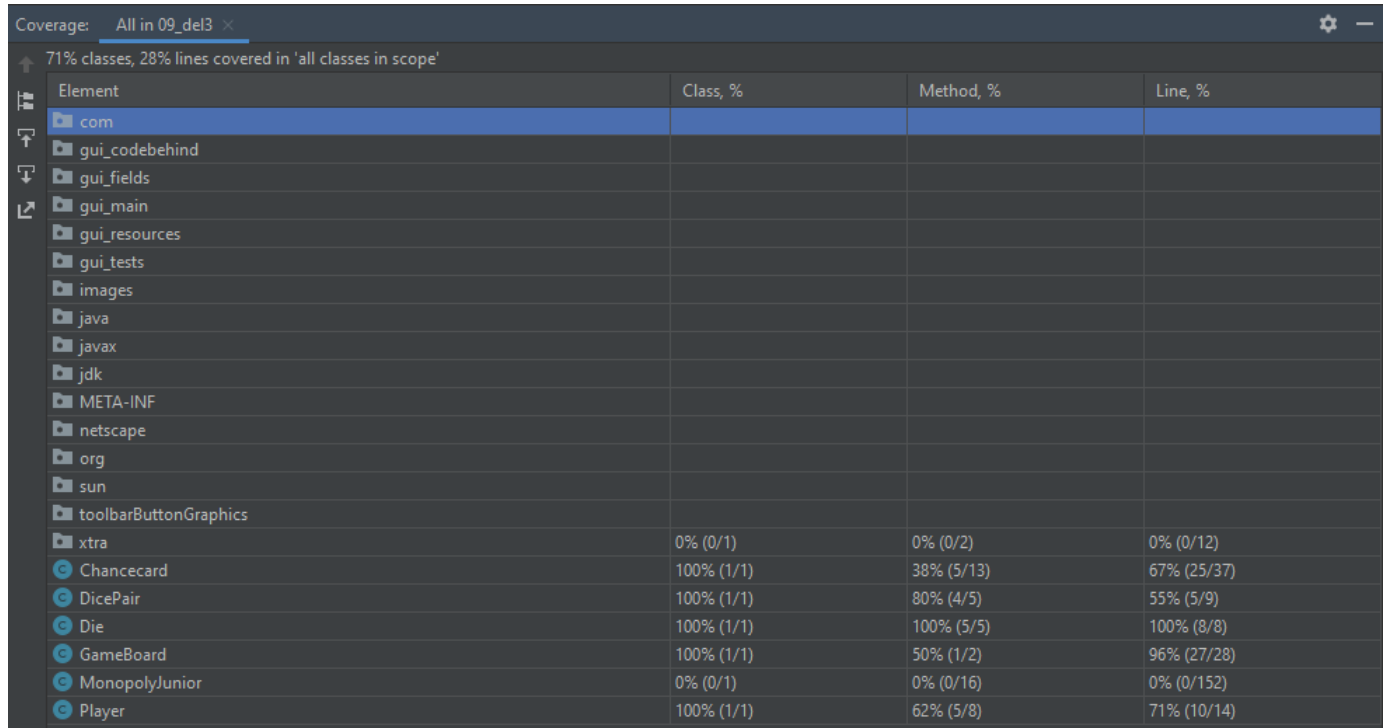
Funktionaliteter som disse har gruppen valgt ikke at implementere, grundet ressourcemangel samt anderledes prioritering.

Carolina kom hurtigt i gang med spillet, og da spillet var intuitivt for hende, var der ingen tidspunkter, hvor hun skulle bruge mere end et par millisekunder for at finde ud af, hvad hun skulle.

Anden brugertest blev udført af Bircan Turan Serdemir, 27 år. Spillet blev vurderet til at være brugervenligt, men endnu engang blev der givet kritik i form af at der kunne blive gjort brug af flere elementer fra det originale Monopoly Junior spil. Bircan kunne meget hurtigt finde rundt i spillet, og fangede desuden relativt hurtigt, at der kun var én terning som skulle bruges, og at det var den samme terning for hver spiller.

7.3 JUnit tests

For at teste små specifikke dele af projektet, er der blevet udført en række af JUnit tests. Disse tests er af de centrale metoder. Der er lavet en specifik package kaldet test, hvori en testklasse for hver java klasse i spillet ligger. Desuden er testen kørt med code coverage, som viser, hvor meget af projektet der er blevet brugt i de forskellige tests tilsammen eller enkeltvis.



Element	Class, %	Method, %	Line, %
com			
gui_codebehind			
gui_fields			
gui_main			
gui_resources			
gui_tests			
images			
java			
javax			
jdk			
META-INF			
netbeans			
org			
sun			
toolbarButtonGraphics			
xtra	0% (0/1)	0% (0/2)	0% (0/12)
Chancecard	100% (1/1)	38% (5/13)	67% (25/37)
DicePair	100% (1/1)	80% (4/5)	55% (5/9)
Die	100% (1/1)	100% (5/5)	100% (8/8)
GameBoard	100% (1/1)	50% (1/2)	96% (27/28)
MonopolyJunior	0% (0/1)	0% (0/16)	0% (0/152)
Player	100% (1/1)	62% (5/8)	71% (10/14)

Figur 7.3 Code coverage for alle Junit tests

På figur 7.3 kan det ses, at 71% af projektets klasser er blevet brugt i testene. De eneste klasser, som ikke er blevet brugt er Audio klassen samt MonopolyJunior klassen. Dette er i gruppen vurderet til at være en faktor, som kan overses, da Audio klassens funktionalitet sagtens kan testes ved at se, om der kommer lyd ud af enhedens højttalere. Derudover er afspilning af en lyd ikke et krav i projektet men et tilføjet designvalg af gruppen.

Da main metoden er i MonopolyJunior klassen, og det er den klasse, som beskriver hele spil-flowet samt instantierer de andre klasser, vil der ikke kunne laves en brugbar test, hvor MonopolyJunior klassen bliver instantieret. Dette er da der i gruppen er blevet prioriteret af lave tests af mindre dele af programmet, så specifik funktionalitet kunne blive testet.

Nogle af de mest centrale metoder, som er blevet testet er bl.a. playerTurnFlow metoden samt chancecardField metoden. Disse sørger for, at spillernes brikker kan rykkes ved terningekast samt at chancekort kan vises i centerfeltet, når en spiller lander på bestemte felter.

8 Konklusion

Ud fra den prioritering, som gruppen satte ifm. kravspecifikationsfasen af projektet, kan der konkluderes, at alle de krav, som var vurderet til at være af højeste prioritet er opfyldt.

Der har derudover også været ressourcer nok til at få opfyldt en række af 2. prioritets kravene, altså de krav, som blev vurderet til at være 'should have's.

Der har desuden været et fokus på at gøre brug af GRASP i så høj grad som muligt samt fokusere på brugervenligheden af designvalgende for den grafiske brugergrænseflade. På baggrund af de diverse design patterns har vi bl.a. især gjort brug af lav kobling samt høj samhørighed, som beskrives i afsnit 4.

9 Konfiguration

For at køre programmet skal repository'en først klones, hvilket gøres ved brug af følgende link:

`https://github.com/Berfin20/09_del3.git`

Den angivne repository skal klones vba. Git på eks. Windows, hvilket gøres med følgende kommando:

```
$git clone https://github.com/Berfin20/09_del3.git
```

Herefter logges der evt. ind på den pop-up skærm, som , muligvis bliver vist.

Når repository'en er blevet klonet, bruges nedenstående kommando til at tjekke ind på master-branchen:

```
$ git checkout <master>
```

Programmet afvikles ved at åbne hele projektet og køre main metoden i MonopolyJunior klassen. Først læses RM filen, herefter bygges projektet.

For at se loggen over projektet bruges nedenstående metode:

```
$ git log
```

Den nyeste version af git findes ved at skrive i terminalen: `git - version`. For at sikre sig, at versionen er opdateret, skal man skrive i terminalen: `git - update`. De forskellige versioner kan ses under historikken på GitHub, for at få et overblik over hvordan projektet så ud før og efter hver commit ændring.

Der var i gruppen problemer med merging konflikter, hvilket har betydet, at der var et behov for at oprette en ny repository, hvor en backup mappe af projektet blev uploadet som første commit til master branchen.