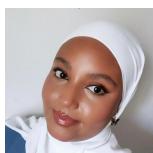


CDIO 3

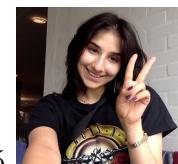
DEVELOPMENT METHODS FOR IT SYSTEMS - 62531 F22
VERSION CONTROL AND TEST METHODS - 62532 F22
INTRODUCTORY PROGRAMMING - 02312 F22

DTU - Danmarks Tekniske Universitet

Gruppe 32:



Amira Omar - s205821



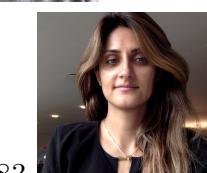
Besma Faris Al-Jwadi - s224325



David Rivera - s215461



Mélissa Woo - s224311



Negar Ostadhassan Salmani - s224283



Bayan Al-Dowairi - s224329

Afleveringsdato: 25. november 2022

Resumé

I denne opgave har vi fået til opgave, at digatilisere Monopoly Junior for kunden. Vi har taget kundens krav og reglerne for brætspillet som udgangspunkt til projektet. Disse krav har vi analyseret og prioriteret grundigt for at kunne komme ud med den bedste mulige udgave af dette spil.

Vi har udarbejdet et use case diagram, og et fully dressed usecase. Derudover, har vi en domænemodel, systemsekvensdiagram, som skal fortælle lidt om klassefordelingen og måden spilleren interagerer med spilsystemet. Diagrammerne er lavede før kodningsprocess, og er blevet ændret sideløbende. Dette system er udviklet iterativt og inkrementelt. Sammen med koden, udfører vi løbende nogle tests til koden for at se om det virker.

Timeregnskab

5 minutter

Indhold

Resumé	2
Timeregnskab	3
1 Indledning	5
2 Projektplanlægning	6
2.1 Prioritering	6
2.2 Konfigurationsstyring	6
3 Krav og Analyse	7
3.1 Kravspecificering	7
3.2 Use-Case: Brief	9
3.3 Use-Case: Fully-Dressed	9
4 Design	12
5 Implementering	14
6 Dokumentation	15
6.1 Versionsstyring	15
6.2 Hvad er arv?	15
6.3 Hvad betyder abstrakt?	15
6.4 Hvad det hedder hvis alle fieldklasserne har en landOnField metode der gør noget forskelligt.	15
7 Test	16
7.1 Dokumentation for test med screenshot	16
7.2 Test cases	16
7.3 Dokumentation for alle bestået tests	22
7.4 Bruger test	23
7.5 Dokumentation for overholdt GRASP	24
7.5.1 High cohesion and low coupling:	24
7.5.2 Creator:	24
7.5.3 Generalisering-Arv:	25
7.5.4 Information Expert:	25
7.5.5 Controller:	25
7.5.6 Polymorphism:	26
7.6 Yderligere kommentar:	27
8 Konklusion	27
9 Bilag	28

1 Indledning

Vi skal udvikle Monopoly Junior spil. Spillerne kan være mellem 2-4, hvor man på skift slår med en terning, og dermed skal lande på de forskellige felter på spillepladet fra 2-24, som skal ændre på vedkommendes kontobalance som resultat.

Spillerne starter med M20, M18 eller M16 på kontoen, og skal købe hver ledige ejendom de lander på. Når en spiller ikke har råd til at købe/betale eller går i minus, så er spillet slut, og vinderen er spilleren med flest penge på kontoen.

Dette system er bygget iterativt. Det skal vi tage et kig på i følgende segmenter.

Spillet bygger på vores forrige projekter, nemlig CDIO 1 og 2, derfor har vi genbrugt de relevante koder derfra til dette system.

2 Projektplanlægning

2.1 Prioritering

Vi afsætter den første uge af projektet til analyse. Vi vil først lave en liste over krav fra kundens vision og spillereglerne for Monopoly Junior. Vi vil derefter prioritere kravene med Moscow-metoden (Must have, Should have, Could Have og Will not have). Vi dedikerer anden uge til designfasen og implementering af kravene som har Must-have prioritet. Vi laver en Code-Freeze i slutningen af ugen. Hvis vi stadig har tid, før Code-Freezen, implementerer vi så Should-Have- og Could-Have-kravene. Den tredje uge er dedikeret til at teste vores kode og færdiggøre rapporten.

2.2 Konfigurationsstyring

1. Hvor er filerne?

De ligger på en remote repository på GitHub og på hver vores lokale repositories.

2. Hvordan finder vi den nyeste version?

Vi sørger for alle at pulle den nyeste version af produktet fra github, så vi alle kan holde os opdaterede.

3. Hvordan sikrer vi at vi ved om versionen er opdateret?

Hver gang vi laver en merge til master, skriver vi version nummer i kommentarer til com-miten.

4. Hvordan finder vi versioner som passer sammen?

NB I løbet af projektet blev det tydeligt, at der skal aftales en Git og/eller GitHub flow i starten. Det tager vi med til næste projekt sammen med en aftale om standarden for navngivning af krav (fx KR01, KR02...)

3 Krav og Analyse

3.1 Kravspecificering

Vi har indelt reglerne fra Monopoly Junior ved hjælp af MOSCOW-metoden ifølge, hvad vi mente var helt væsentligt til spillet, samt hvad vi syntes var realistisk at levere med den tid og kapacitet, vi havde.

Must-Have kravene til spillet:

1. Et spil mellem 2 til 4 spillere
2. Alle spillere skal starte fra startfeltet
3. Spillerne bevæger sig i ring omkring brættet, hvor de alle går med uret. Her skal spilleren være is stand til at lande på et felt, og fortsætte derfra på næste slag.
4. Spilleren rykker altid frem, aldrig tilbage
5. Man slår på skift med 2 terninger* (vi fandt først ud af at der kun burde være én terning, senere i processen, men vi tager det med til næste gang)
6. Nogle felter skal kunne købes
7. Spillet sluttes når én af spillerne ikke kan betale husleje eller bøde, eller købe den ejendom, de er landet på. Altså, når den første er gået bankerot.
8. Vinderen er den med den største værdi i sin konto, når spillet sluttes
9. Startbeløbet i spillerens konto afhænger af hvor mange deltagere der er. Hvis der er 2 spillere, starter man med 20M i kontoen. Er man 3 mand, starter man med 18M. for 4 mand, starter man med 16M.
10. Hvis en spiller lander på et ledigt ejendomes felt, skal feltet købes af spilleren.
11. Hvis et ejendomsfelt ejes af en anden spiller, skal spilleren betale husleje til ejeren, for at stå på vedkommendes grund.
12. Man slipper for at betale leje, hvis man er på besøg i fængslet, eller står ved parkeringsfeltet.
13. Computeren agerer som banken
14. Hvis spilleren lander på et chancefelt (?), skal aktionen på kortet udføres. Følgende kort hører under Must-Have Krav:
 - (a) chance 1: Move forward to GO. Collect M2
 - (b) chance 2: Move 5 squares forward
 - (c) chance 3: Move 1 square forward, or take one more chance card
 - (d) chance 4: Move forward to the Promenade
 - (e) chance 5: You have eaten a lot of candy. PAY M2 to the bank
 - (f) chance 6: It is your birthday! Each player will give you M1. HAPPY BIRTHDAY!
 - (g) chance 7: You have done your homework! Collect M2 from the bank
 - (h) chance 8: NO CHARGE SQUARE! Move forward The Skate Park to make the perfect grind! If no one owns it, then you get it for free. Or you have to pay the owner.
 - (i) chance 9: NO CHARGE SQUARE! Move forward to cyan or red square. If no one owns it, then you get it for free. Or you have to pay the owner
 - (j) chance 10: NO CHARGE SQUARE! Move forward to lightgrey or yellow square. If no one owns it, then you get it for free. Or you have to pay the owner

3 KRAV OG ANALYSE

- (k) chance 11: NO CHARGE SQUARE! Move forward to orange square. If no one owns it, then you get it for free. Or you have to pay the owner
- (l) chance 12: NO CHARGE SQUARE! Move forward to cyan square. If no one owns it, then you get it for free. Or you have to pay the owner
- (m) chance 13: NO CHARGE SQUARE! Move forward to red square. If no one owns it, then you get it for free. Or you have to pay the owner
- (n) chance 14: NO CHARGE SQUARE! Move forward to orange or green square. If no one owns it, then you get it for free. Or you have to pay the owner
- (o) chance 15: NO CHARGE SQUARE! Move forward to pink or dark blue square. If no one owns it, then you get it for free. Or you have to pay the owner

* Vi har opdaget for sent, at der faktisk står i reglerne, at Monopoly Junior spilles kun med én terning.

De "Should-Have"krav til spillet:

1. Spilleren modtager M2 når de passerer eller lander på startfeltet
2. Spilleren går direkte i fængsel uden at passere start og uden at modtage 2 MD.
3. En gratis parkering felt
4. Spiller skal betale 1MD til sin næste tur for at komme ud af fængslet Spiller spiller så sin tur som normale. Spiller må gerne modtage husleje mens de sidder i fængslet.

De "Could-Have"Krav til spillet:

1. Den yngste spiller starter
2. Placer et "Solgt"-skilt på det købte felt
3. Hvis uafgjort når spillet slutter, spillerens lægger værdien af deres ejendom til deres penge for at afgøre vinderen
4. Spilleren trække et chancekort og udføre action på kortet. Følgende kort hører under Must-Have Krav:
 - (a) "Get out of jail free"-kort
 - (b) Kortene, der skal veksles mellem spillerne

De "Will-Not-Have"Krav til spillet:

1. Hvis ikke nok penge til at betale husleje eller afgift fra chancekort, så spilleren betale gælden med sin ejendomme
2. En spiller skal betale sin gæld til en anden spiller med sine ejendomme, hvis spilleren ikke har nok penge
3. Hvis spilleren ikke kan betale sin gæld til banken, bliver spillerens ejendomme sat til salg igen

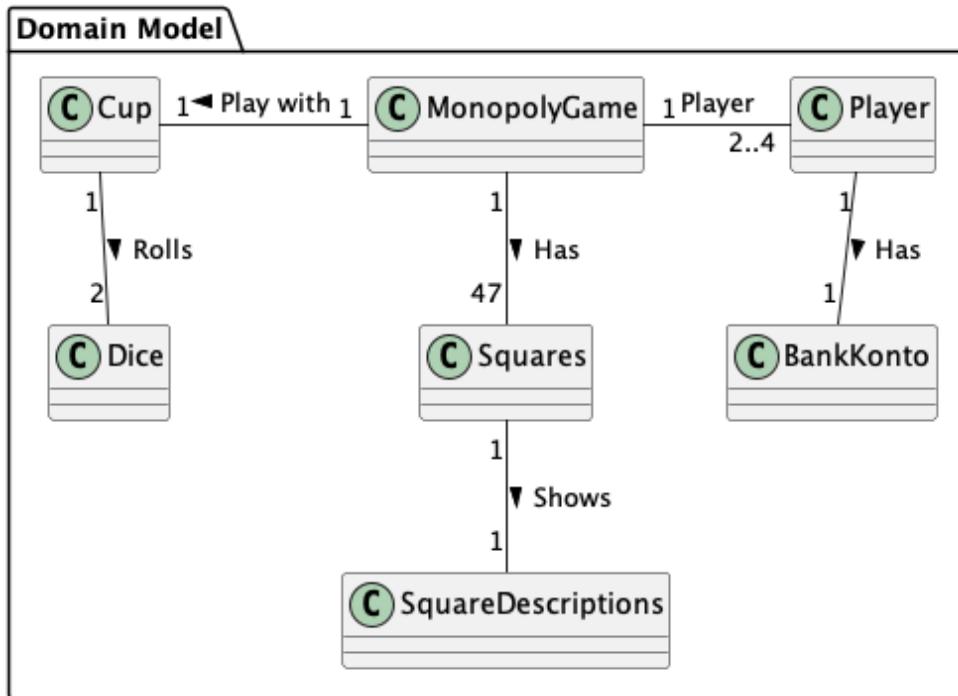
3.2 Use-Case: Brief

Use case 1	Spil spillet
Actor	Spiller
Basic flow	Brugerne trykker enter og starter spillet. Her vælges hvor spillere der spiller med her kan der vælges op til fire personer. Spillerne får hver især tildelt et startbeløb af vores system, som opdaterer deres bankkonto alt efter hvor mange der spiller med. Spilleren starter i et startfelt og forsøger at lande på en af de 47 felter som hver har deres egen funktion, som kan ende med et plus eller minus for spillerne. Man vinder spillet ved at være den spiller der til sidst står med flest penge vinder spillet når en af de andre spillere har mistet alle sine penge.

3.3 Use-Case: Fully-Dressed

Section	Comment
Case Name	Spil spil
Scope	Terninge Spil
Level	User-goal
Primary Actor	Spiller
Stakeholders and Interests	Kunden: De vil gerne have et system, der kan bruges på maskinerne (Windows) i alle databarerne på DTU. De kunne også tænke sig at spillet udbygges med forskellige typer af felter samt en spilleplade, så når spilleren lander på et felt skal de fortsætte derfra på det næste slag. Så man går i ring på spillebrættet. Nu skal spillet spilles mellem 2-4 personer.
Preconditions	<ol style="list-style-type: none"> 1. Spillet kan spilles af almindelige mennesker 2. Spilleren skal kunne spille uden en brugsanvisning

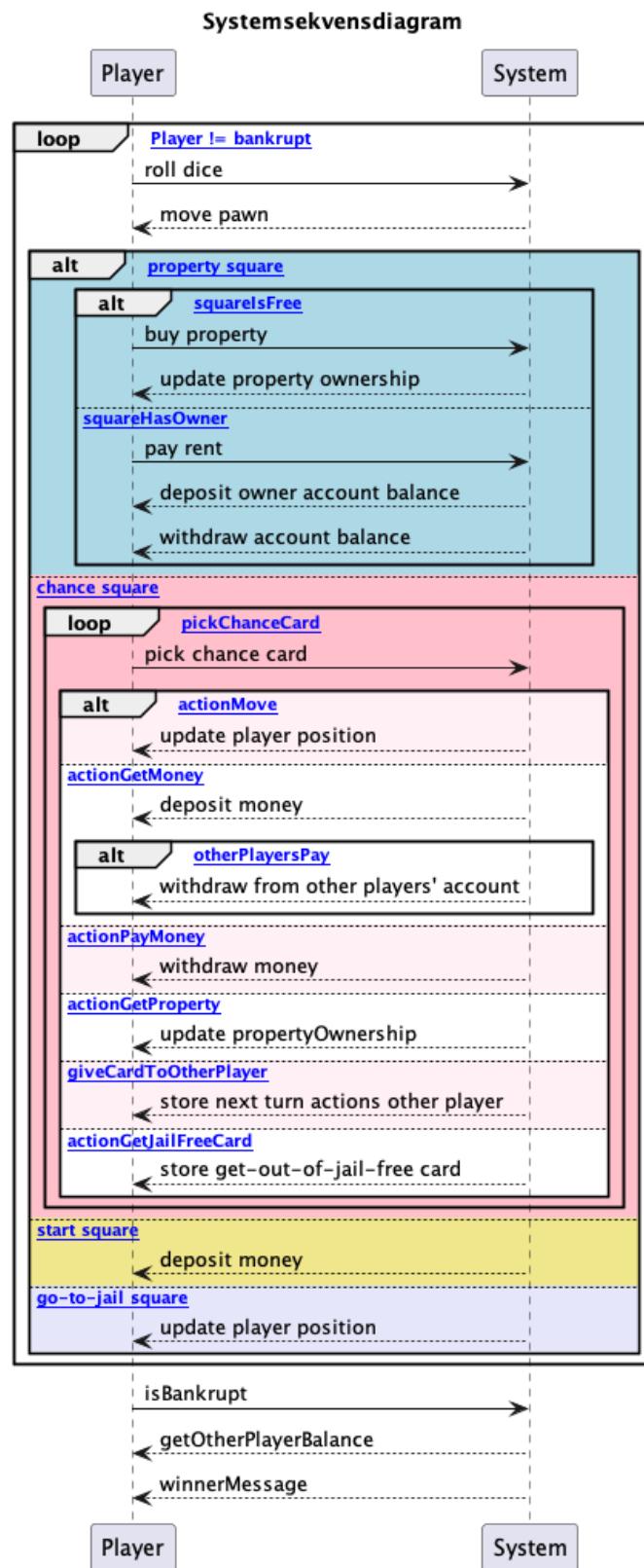
Success Guarantee	<ol style="list-style-type: none"> 1. Det er et spil der beregner spillernes pengebeholdning og indsætter det de optjener i deres konto. 2. Spilleren indtaster deres navne 3. Spilleren starter med hhv. m20/ m18/ m16 i kontoen, dette afhænger af hvor mange der spiller med. 4. Spillerne starter spillet i startfeltet. 5. For hvert af de 47 felter som spilleren lander på har hhv. en positiv eller negativ effekt på konto. 6. For hvert felt som spilleren lander på, udskrives der en beskrivende tekst. 7. Spillerne kaster terninger og lander på de ledige felter (ejendom) som de tilkøbe sig 8. Banken overserer transaktionen af købet af ledige ejendomme. 9. Hvis en spiller lander på et felt (ejendom) ejet af en anden spiller, skal de betale husleje til den spiller. 10. Spillerne kan også trække en af de 20 chancekort, her betales der også en afgift hvis man gør brug af dette. 11. Første spiller som har størst værdi af ejendomme sammenlagt (omsat til penge) tilbage, når en af de andre spiller er gået bankerot, vinder spillet.
Extensions	<ol style="list-style-type: none"> 1. Systemet skal fratake en spillers penge når de lander på en af de ledige 47 felter. 2. Systemet udskriver en beskrivende tekst for hver gang der landes på et felt og de har hver deres beskrivende tekst. 3. Systemet skal indsætte penge ind på en spillerens bankkonto, hvis en anden spilleren lander på et felt (ejendom) de ejer. 4. Systemet skal fratake penge fra en spillers bankkonto når der lander på et felt (ejendom) ejet af en anden spiller. 5. Systemet skal fratake penge fra spilleren, når de lander på et chancefelt. 6. Systemet skal beregne den største værdi af spillernes ejendomme sammenlagt når en af de andre spiller har fået frataget alle ejendele (dvs. gået fallit) og stoppe spillet når vinderen findes.



Figur 1: Domænemodel for spillet Monopoly Junior

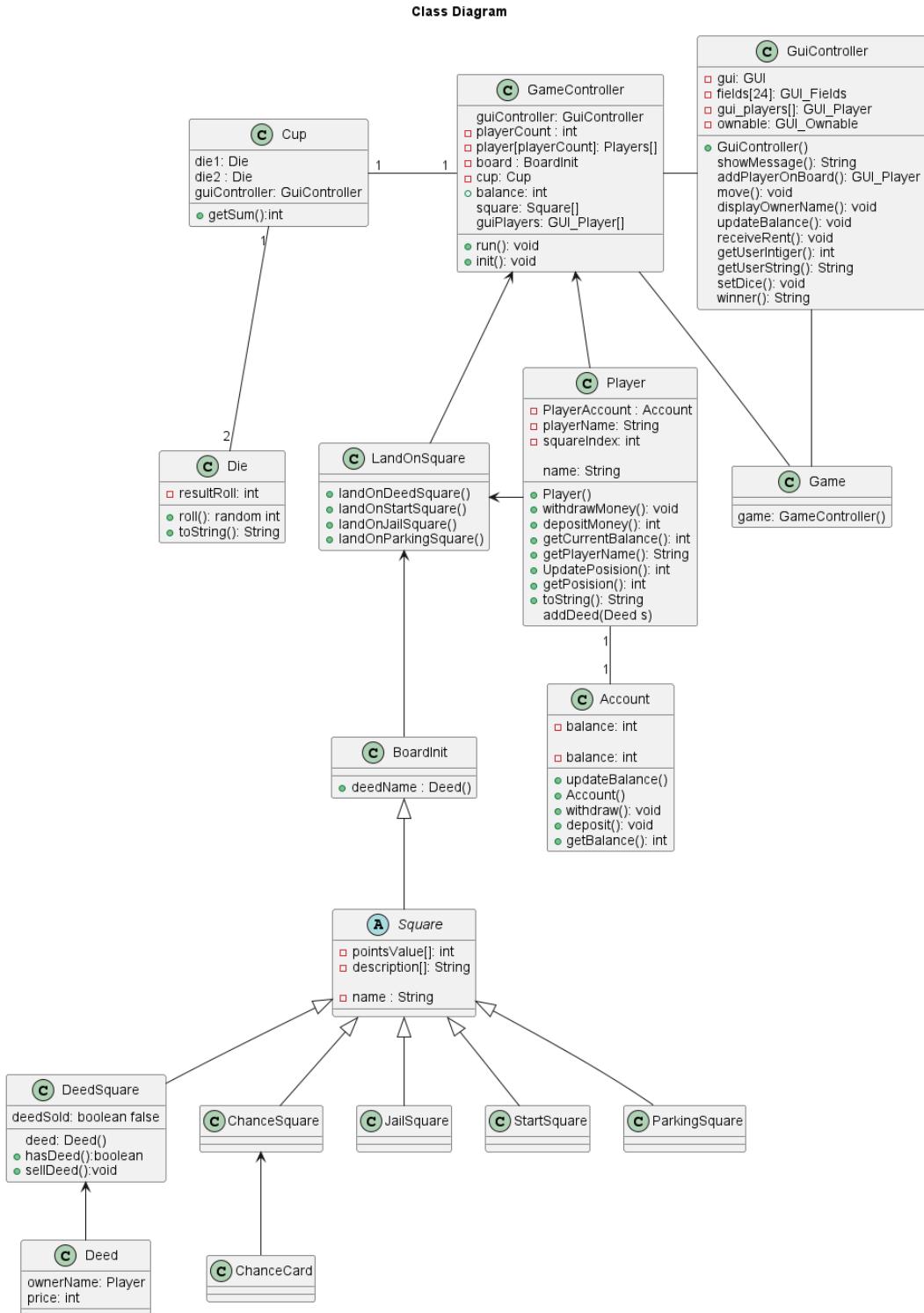
Vi har udviklet en domæne model, som skal vise hvad vores spil rent funktionelt skal indeholde for spilleren. Vi har en main monopoly game klasse, hvor man spiller med ét rafflebægre, som ruller med 2 terninger. Spillet har nogle felter, som indeholder nogle feltbeskrivelser, alt afhængigt af hvor man er i monopoly verden. Derudover, indeholder spillet også en spiller, karakteriseret ved en bil, som har en bankkonto, som holder monopoly valutaen M.

4 Design



Figur 2: Systemsekvensdiagram for spillet Monopoly Junior

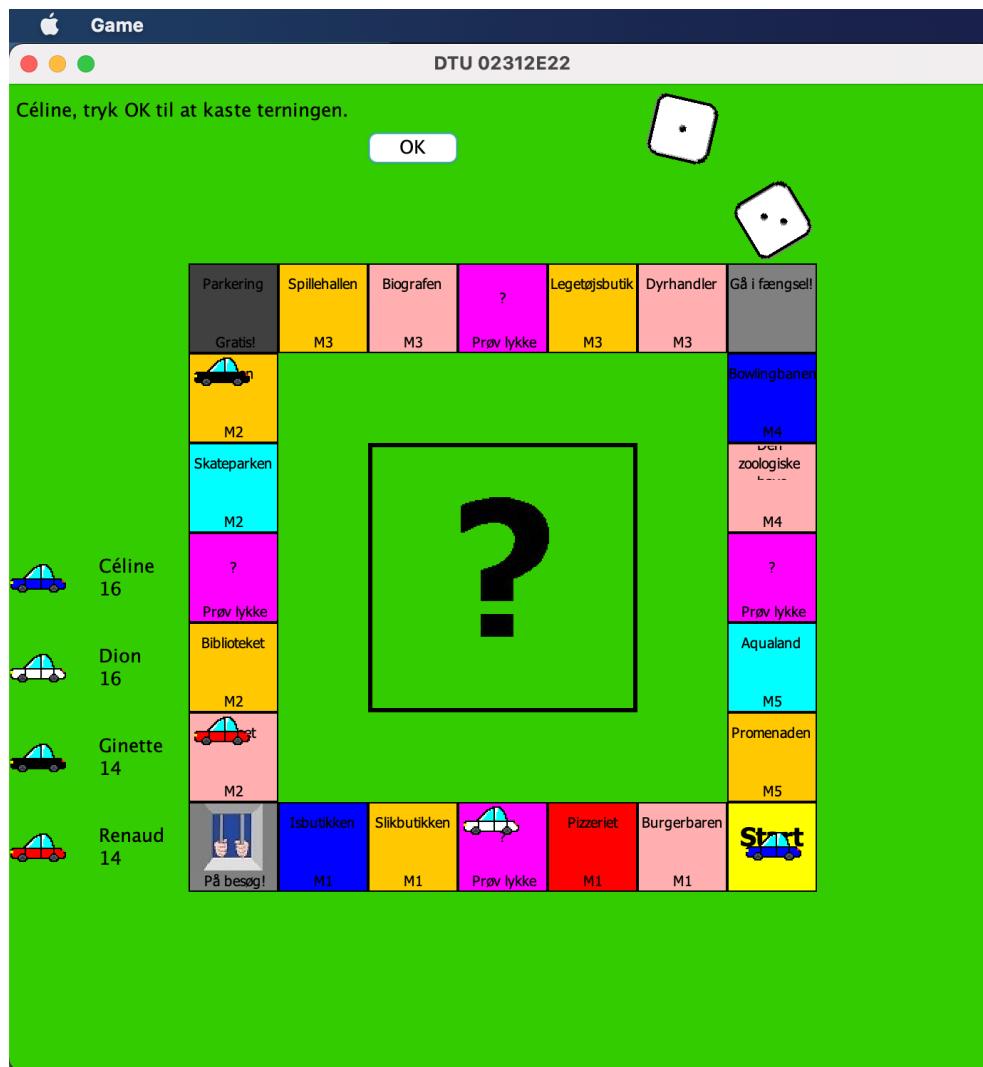
Nedenstående model er vores sidste udgave af klassediagrammet (se bilag til at se den første) Vi har forsøgt at overholde high cohesion ved brug af en GameController, som holder styr på de forskellige handlinger, der foretages i spillet. Med Controllers var vi nødt til at ofre lidt af de lav kobbling, fordi de skal kunne have adgang til mange dele af spillet, for at styre dem. Det var vi parat til at gøre, fordi det tillod os at undgå at gentage kode.



Figur 3: Klassediagram for spillet Monopoly Junior

5 Implementering

Dette er modellen af spillet via Gui, med felterne i en firkantet form, med terningerne ovenover. Spillerne indikeres af de små biler, på venstre side af spillet.



Figur 4: Skærmbillede af spillet med GUI

Se i GitHub repository: https://github.com/NegarOstad/G_26_CDIO_3.git

6 Dokumentation

6.1 Versionsstyring

Sådan importeres og køres koden fra GitHub:

1. Vælg den mappe, hvor spillet skal være.
2. Åbn terminal-programmet og nавiger til mappen.
3. Skriv: git init (dette initialiserer dit git-lager)
4. Repository'en et nu lokalt på computeren.
5. Åbn projektmappen i IntelliJ
 - (a) Hvis der er en meddeelse, der siger "Projekt JDK ikke defineret", klik på "Setup JDK" i det modsatte hjørne
6. Klik på Game.java i projektmenuen til højre på skærmen.
7. Klik på Run (den grønne trekant) for at spille spillet.

6.2 Hvad er arv?

Arv er et vigtigt begreb indenfor objektorienteret programmering. Arv sker når en forældreklasse (superklasse) videregiver dens attributter og funktioner til barnekasse (underklasse), (IS-A relation). Dette giver muligheden for at genbruge kodens funktionalitet og dermed sparer på implementeringstid.

6.3 Hvad betyder abstrakt?

Abstrakte klasser bruges til at forestille generelle begreber som ikke skal implementeres på dette niveau. Derudover kan de ikke instantieres. En abstrakt klasse indeholder abstrakte metoder, som kan defineres fuldt ud i en subklasse af den abstrakte klasse.

6.4 Hvad det hedder hvis alle fieldklasserne har en landOnField metode der gør noget forskelligt.

Det er polymorfi vi har at gøre med der.

7 Test

7.1 Dokumentation for test med screenshot

```

@Test
public void addPlayerOnBoard() {
    GUI_Player[] guiPlayer = new GUI_Player[3];
    for (int i = 0; i < 3; i++) {
        guiPlayer[i] = new GUI_Player(name: "ava");
        GUI gui = new GUI();
        gui.addPlayer(guiPlayer[i]);
    }
    assertEquals(expected: 3, actual: 3);
}

```

Figur 5: Junit test, Test addPlayerOnBoard methode in GuiController

```

@Test
public void updateBalance() {
    Player player = new Player();
    player.depositMoney(newPoints: 20);
    player.getCurrentBalance();
    int balance = 2;
    player.withdrawMoney(balance);
    assertEquals(expected: 18, player.getCurrentBalance());
}

```

Figur 6: Junit test, Test updateBalance methode in GuiController

7.2 Test cases

I det ideelle scenarie ville vi have opnået 100% coverage med vores tests. Til dette projekt har vi valgt prioritere en mindre del tests. Vi har valgt at skrive tests til de krav, som vi betragter som helt grundlæggende, for at køre spillet. Det følgende er et udvalg fra vores 21 tests. Det resterende tests kan ses i GitHub repository (link givet under afsnit 5 Implementering)

Test case ID	TC01
Summary	Write summary here

7 TEST

Requirements	Skriv krav, det tester her
Preconditions	skriv preconditions her
Postconditions	skriv post-conditions her
Test procedures	skriv test procedures her
Test data	skrive test data her
Expect result	skriv expected result her
Actual result	skriv actual result her
Status	skriv status her
Tested by	skriv navn her
Date	skriv dato her
Test environment	IntelliJ IDEA + add specifics (see slides 17-19 versionsstyring lektion 6)

Test case ID	TC01
Summary	Test, at det finder vinderen med den største værdi i sin konto
Requirements	At finde vinderen med største værdi i kontoen
Preconditions	En spiller har minus på sin konto
Postconditions	
Test procedures	Der er fire spillere med forskellige værdie i deres konto i en array. Spilleren med den største værdi i sin konto er vinderen
Test data	Spiller A har 12 point. Spiller B har 140 point. Spiller C har 18 point. Spiller D har 20 point.
Expect result	B
Actual result	B
Status	Passed
Tested by	Negar Ostad
Date	11-22-2022
Test environment	IntelliJ IDEA

Test case ID	TC02
Summary	Test, at det tilføje spillere på spillet
Requirements	At tilføje nye spiller
Preconditions	Brugeren har indtastet antallet af spiller
Postconditions	
Test procedures	Der tilføjes fire guiPlayer
Test data	antal player = 3

7 TEST

Expect result	3
Actual result	3
Status	Passed
Tested by	Negar Ostad
Date	11-22-2022
Test environment	IntelliJ IDEA

Test case ID	TC03
Summary	Test, at updatere spillernes konto
Requirements	At updatere balancen
Preconditions	Spilleren har kastet terning og landet på et felt og fået et point
Postconditions	Spillerns konto er uppdateret
Test procedures	Der er en spiller med 20 pointe i sin konto, og der trækkes 2 pointe fra spillerns konto
Test data	Balance = 20 point = 2 nye balance = 18
Expect result	18
Actual result	18
Status	Passed
Tested by	Negar Ostad
Date	11-22-2022
Test environment	IntelliJ IDEA

Test case ID	TC04
Summary	Tester at metoden sellDeed() opdaterer navnet på ejeren af Deed og at metoden getDeedOwner returnerer navne af spilleren, som ejer Deed'en.
Requirements	At nogle felt har et skød, der kan ejes af spillere
Preconditions	At spilleren har kastet terninge, er landet på et ejendomsfelt og har nok penge til at købe skødet
Postconditions	Spillet kører stadig

Test procedures	<ol style="list-style-type: none"> 1. Initialisér to Player objekter testPlayer1 og testPlayer2 2. Initialisér Square objekt testDeedSquare af type DeedSquare 3. Kald <i>sellDeed()</i> på testDeedSquare med testPlayer1 som argument 4. Kald <i>getDeedOwner()</i> på testDeedSquare, der returnerer spilleren, som ejer feltet 5. Kald <i>sellDeed()</i> på testDeedSquare med testPlayer2 som argument 6. Kald <i>getDeedOwner()</i> på testDeedSquare, der returnerer spilleren, som ejer feltet.
Test data	Til første sellDeed() kald: testPlayer1 Til anden sellDeed() kald: testPlayer2
Expect result	testPlayer1 og bagefter testPlayer2
Actual result	testPlayer1 og bagefter testPlayer2
Status	Passed
Tested by	Mélissa
Date	23 november 2022
Test environement	IntelliJ IDEA

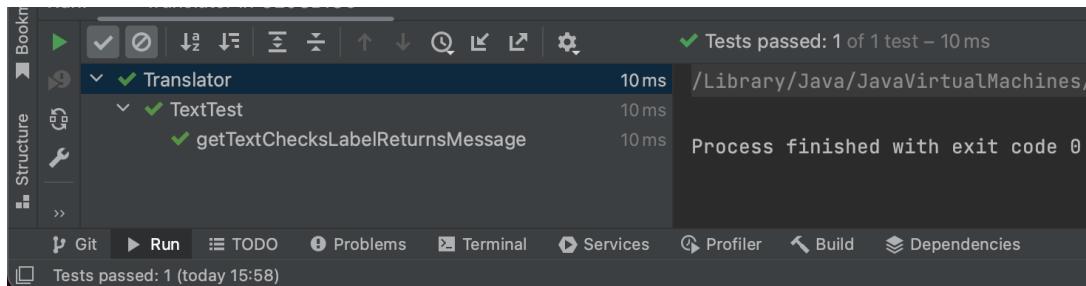
Test case ID	TC05
Summary	Tester, at spillerens konto opdateres, når de lander på et felt med et ledigt skøde og at en anden spillers konto opdateres, når de lande på et ejet felt, hvor de skal betale husleje til ejeren af skødet.
Requirements	At nogle felt har et skød, der kan ejes af spillere, samt at der betales husleje til ejeren, når andre spiller lande på deres felt
Preconditions	En spiller lande på et felt med et ledigt skød og en anden spiller lande på feltet, hvor det bestemte skød findes
Postconditions	At spilleren, som skal betale husleje har nok penge i banken

Test procedures	<ol style="list-style-type: none"> 1. Initialisér et Player[] testPlayers med to spillere og indsæt et startbeløb i hver deres konto 2. Initialisér en DeedSquare på index 1 i et testBoard Square[] array 3. Inialisér et LandOnSquare objekt playerTurnTest 4. Kald landOnDeedSquare() på playerTurnTest 5. Kald getCurrentBalance på spilleren, som har købt det ledigt skød 6. Kald landOnDeedSquare() på playerTurnTest 7. Kald getCurrentBalance() på spilleren, som skal betale husleje
Test data	<p>skrive Til den første landOnDeedSquare kald, argumenter til landOnDeedSquare: newPosition: 1, index af currentPlayer: 0.</p> <p>Til den anden landOnDeedSquare kald: newPosition: 1, index af currentPlayer: 1</p> <p>Startbeløb i konto: 20</p> <p>Prisen af skødet: 5</p>
Expect result	15 og 15
Actual result	15 og 15
Status	Passed
Tested by	Mélissa Woo
Date	23 november 2022
Test environement	IntelliJ IDEA

Test case ID	TC06
Summary	Tester at Text-klassens getText() metode returnerer den besked, som tilhører den label String, der passerer ind i metoden
Requirements	At spillet kan oversættes til flere sprog
Preconditions	Spillet er startet
Postconditions	Spillet kører stadig

Test procedures	<ol style="list-style-type: none"> 1. Initialisér et Text Objekt reader med en String, som er path til tekstufilen, der skal aflæses 2. Initialisér en test String 3. Kald metoden getText på reader med den ønskede label som argument 4. Se om test String er lige med det String, der bliver returneret via getText()
Test data	label String "enterPlayerCount" og tilhørende besked "Hvor mange spiller?"
Expect result	"Hvor mange spiller?"
Actual result	"får du det GRATIS! Ellers skal du BETALE leje til ejeren"
Status	Failed
Tested by	Mélissa Woo
Date	23 november 2022
Test environment	IntelliJ IDEA

Problemet, der blev opdaget under TC06 var, at linjerne i tekstuften var parsed med komma, men sætningerne selv indholdt kommaer på forskellige stedet. Derfor blev linjerne ikke delt ordenligt i to index, hvor index 0 skulle være alle de labels og index 1 skulle være alle de beskeder, der kunne returneres ved at kalde på metoden. Problemet blev løst ved at parse med semi-kolon i stedet for.

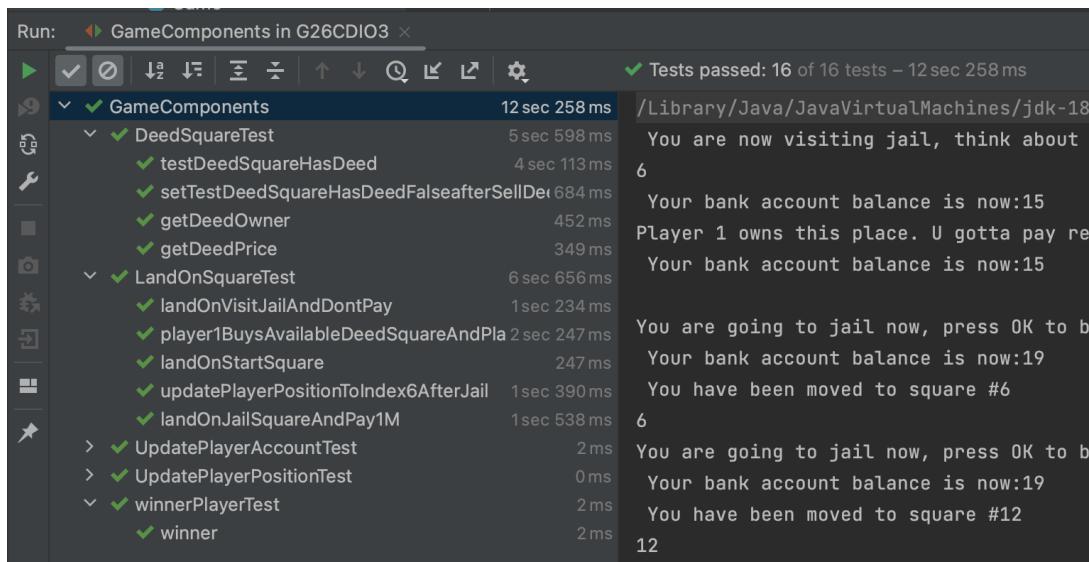


Figur 7: Text klasse text bestået

Test case ID	TC07
Summary	Tester om spilleren som lander på "gå til fængsel " (felt 18) bliver flyttet til fængslet (felt 6), og får sanktion på -M1 fra sin bank konto.
Requirements	At spiller bliver flyttet til det rigtige felt, og der hæves det rigtige beløb fra konto
Preconditions	Spillet er startet
Postconditions	Spillet kører stadig

Test procedures	<ol style="list-style-type: none"> 1. Initialisér player [] og square [] med en player 2. Initialisér et LandOnSquare objekt playerTurnTest 3. Initialisér player med navn 4. kald depositMoney() 5. definer fængselfelt i testboard[] 6. kald updatePosition() til spiller
Test data	Player 1 har til at starte med M20. new position 18 , expected position 6.
Expect result	M19 , og spille rykkes til felt fængsel
Actual result	spilleren rykkes til fængsel og spilleren har det rigtige beløb
Status	Success
Tested by	Bayan Al-Dowairi
Date	24 november 2022
Test environment	IntelliJ IDEA

7.3 Dokumentation for alle bestået tests



The screenshot shows the IntelliJ IDEA Test Runner interface with the following details:

- Run:** GameComponents in G26CDIO3
- Tests passed:** 16 of 16 tests – 12 sec 258 ms
- Test Structure:**
 - GameComponents** (12 sec 258 ms)
 - DeedSquareTest** (5 sec 598 ms)
 - testDeedSquareHasDeed (4 sec 113 ms)
 - setTestDeedSquareHasDeedFalseafterSellDeed (684 ms)
 - getDeedOwner (452 ms)
 - getDeedPrice (349 ms)
 - LandOnSquareTest** (6 sec 656 ms)
 - landOnVisitJailAndDontPay (1sec 234 ms)
 - player1BuysAvailableDeedSquareAndPla (2sec 247ms)
 - landOnStartSquare (247 ms)
 - updatePlayerPositionToIndex6AfterJail (1sec 390 ms)
 - landOnJailSquareAndPay1M (1sec 538 ms)
 - UpdatePlayerAccountTest** (2 ms)
 - UpdatePlayerPositionTest** (0 ms)
 - winnerPlayerTest** (2 ms)
 - winner (2 ms)
- Output Log:**

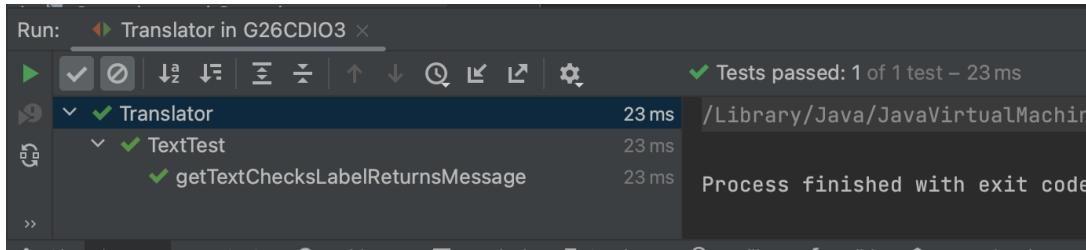
```

Tests passed: 16 of 16 tests – 12 sec 258 ms
/Library/Java/JavaVirtualMachines/jdk-18
You are now visiting jail, think about
6
Your bank account balance is now:15
Player 1 owns this place. U gotta pay re
Your bank account balance is now:15
You are going to jail now, press OK to b
Your bank account balance is now:19
You have been moved to square #6
6
You are going to jail now, press OK to b
Your bank account balance is now:19
You have been moved to square #12
12

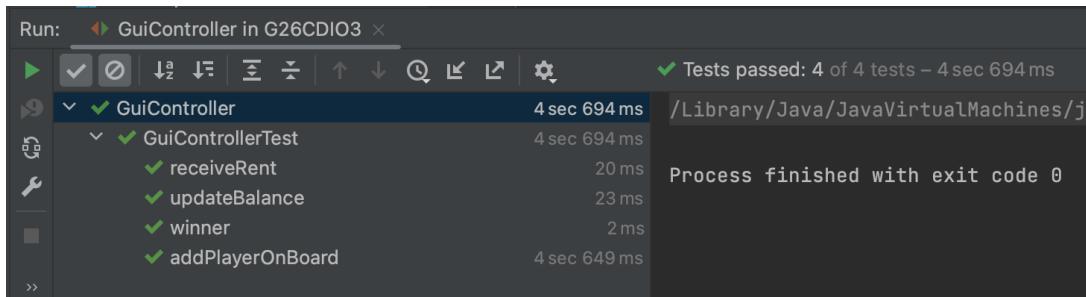
```

Figur 8: GameComponents tests beståede

7 TEST



Figur 9: Translator test bestået



Figur 10: GuiController tests beståede

7.4 Bruger test

Yderligere for at få en overordnet feedback på vores Monopoly Jr. fik vi udspurgt en helt almindelig 20 årig ung dame til at prøve spillet af. Før hun fik lov til at starte spillet, fik hun forklaret de specifikke regler for og formålet med spillet. Hun blev også gjort opmærksom på at spillet havde et lidt anderledes udseende end hun var vant til mht. GUI'en.

Efterfølgende blev hun sat igang, og efter at hun havde spillet cirka mere end 2 runder, blev der spillet nogle spørgsmål:

Hvad synes du om spillet?

Svar: Det er hyggeligt og sjovt at spille Monopoly på sådan en måde, og det føles samtidigt også mærkeligt at man styrer spillet via ENTER-knappen og bare observerer på skærmen hvordan der lejes, købes, mistes og rykkes rundt på spillerne. Dog en lidt langt spilletid, som ikke kunne ændres selv hvis man kun spiller med to spillere men synes at det er fedt at man selv kunne vælge sprog.

Er spillet let at forstå og gennemføre?

Svar: Det er meget simpelt og let at forstå, og det er meget genkendeligt og overskueligt grundet kendskab til det fysiske brætspil.

Gjorde du dig nogle iagttagelser under spillet der kunne forbedres det til næste gang?

Svar: Jeg er meget overrasket over af skærmen (mener GUI'en) var så skrigende grøn, og det er noget jeg blev vant til til sidst, men det irriterede mine øjne lidt. Under spillet når man kaster terningerne, så står de lidt i vejen for tekstdeskriptionsen når man gerne vil læse hvad der sker for den pågældende spiller.

7.5 Dokumentation for overholdt GRASP

7.5.1 High cohesion and low coupling:

Et eksempel på vores forsøg i at overholde dette, kan ses fra vores klasse diagram. Hvor vores klasser generelt har ensartet ansvarsområder, såsom Player der indeholder alt der vedrører spilleren. For eksempel, spillerns navn, hæve/indsætte penge, se til rådighed, opdatere position, gået fallit, og vinderbesked. Derudover har vi forsøgt at overholde low coupling ved at sikre lav afhængighed mellem klasserne ved at tildele ansvar mellem dem. For eksempel LandOnSquare som tildeler forskellige ansvar for de forskellige typer felter.

7.5.2 Creator:

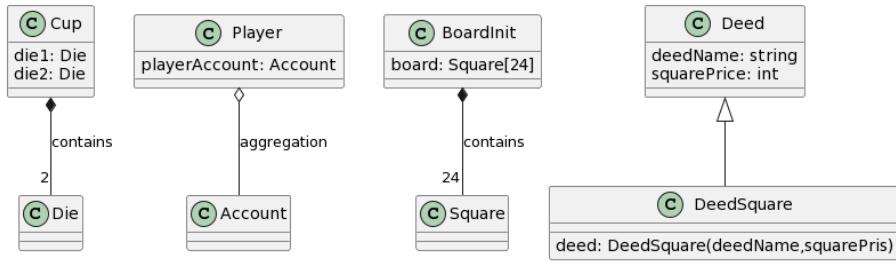
Creator er en del af GRASP patterns, som den identificerer, hvem har ansvar for at oprette andre klasser? I tabellet kan man se at hvilken klasser instansier objekter fra andre klasser.

Object	Which class create the object?
die	Class Cup has the responsibility to create two instances of class Die.
playerAccount	Class Player has the responsibility to create an instance of class Account.
board	Class BoardInit has the responsibility to create an instance of class Square.
deed	The constructor of the class DeedSquare has the responsibility to create an instance of class Deed.
fields	Class GuiController has the responsibility to create an instance of class GUI_Field.
guiPlayer	Class GuiController has the responsibility to create an instance of class GUI_Player.
ownable	Class GuiController has the responsibility to create an instance of class GUI_Ownable.
gui	Class GuiController has the responsibility to create an instance of class GUI.
guiController	Class GameController has the responsibility to create an instance of class GuiController.

Tabel 11: Creator

Eksempler på implementering af Creator Pattern:

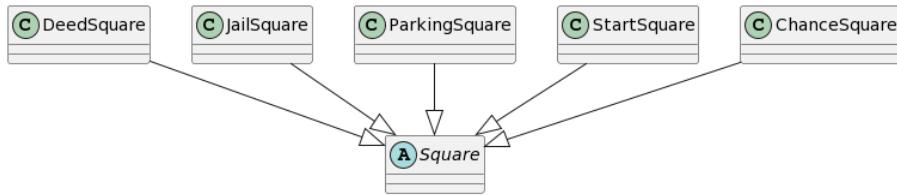
Her kan man se et eksempel af forskellige forbindelser mellem klasser, som arv, aggregation og komposition. For eksempel, Die og Cup har en komposition forhold, da Cup contains Die og Die kan ikke stå alene. Det samme gælder Boardinit som contains Square. Et eksempel på aggregation, er Account som er en del af player, men kan sagtens stå alene.



Figur 11: Creators

7.5.3 Generalisering-Arv:

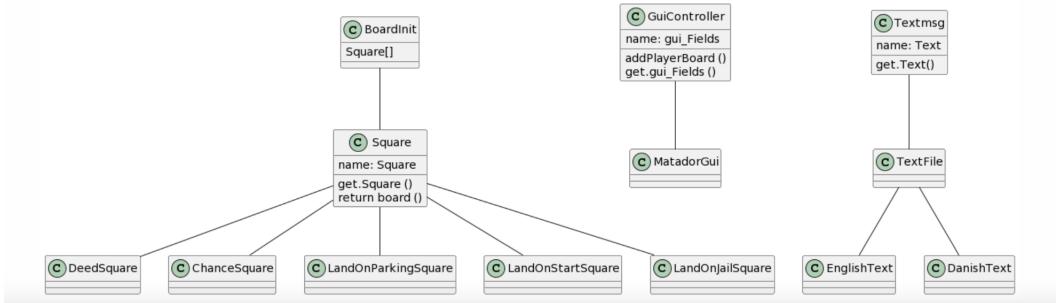
I billedet kan man se at der er 5 forskellige klasser der arver fra den abstrakte klasse ”Square”. De fem klasser er subklasser, og de skal ikke instantier super klassen `Square`.



Figur 12: Arv

7.5.4 Information Expert:

I vores projekt, har vi mange klasser som har fået tildelt ansvaret for de andre klasser, da det er dem som har informationen til de tilhørende klasser og dette visualiseres for neden. Da vi her kan se for `BoardInit`, `Gui Controller` og `Text msg` hvordan de holder alle informationerne for klasserne og dermed får tildelt ansvaret for dem.

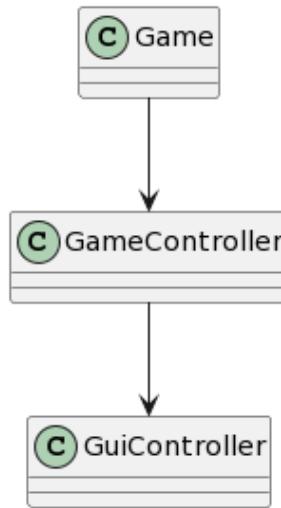


Figur 13: Information Expert

7.5.5 Controller:

En Controller agerer som en mellemmand mellem brugeren og systemets funktioner, som skal udføres.

I vores projekt, har vi to Controllers til vores spil, `GameController` og `GuiController`. `GameController` tager sig af at initialisere alle spillets komponenter og kalder på de forskellige komponenters metoder, for at få spillet til at køre. `GuiController` tager sig af alle de GUI komponenter fra `MatadorGui`'en, og indeholder nogle metoder, som kan bruges til at koble GUI'en til selve spillet bl.a. i `GameController`.



Figur 14: Controller

7.5.6 Polymorphism:

Billedet er et eksempel på brugen af subtype polymorphi i vores CDIO projekt(polymorphi med arv). Det viser at Square klassen kan gøre noget forskellige. Næste gang vil vi også bruge polymorfi til at håndtere alt det der sker efter spilleren er landet på et bestemt slags felt med en abstrakt landOn metode, som vil gøre noget forskelligt for hver subklasse af type felt. På den måde undgår vi også, at bruge ”instanceof”.

```

abstract class Square { ... }

class StartSquare extends Square{
    public StartSquare(String startSquare) {...}
}

class ParkingSquare extends Square{
    public ParkingSquare(String freeParkingSquare) {...}
}

class JailSquare extends Square{
    public JailSquare(String jailSquare) {...}
}

class ChanceSquare extends Square{
    public ChanceSquare(String squareName) {...}
}
  
```

Figur 15: Polymorphism

7.6 Yderligere kommentar:

section Vores færdige produkt er et funktionelt matador junior spil, men med nogle chance kort der er lidt buggy. Der opstod også en fejl til sidst, som nok opstod ved vores sidste merge til master, hvor bilerne sidder fast i GUI'en selv efter spilleren ikke er placeret der længere. Vi kunne nok have undgået det ved at bruge den nyere version af GUI'en.

8 Konklusion

Vi har udviklet Monopoly Junior Spil, hvor spillerne har en konto der påvirkes af hvor man lander på felterne. Det har vi gjort ved brug af vores faglige viden omkring analyse og design. Prioritering af krav var vores første opgave, så vi kunne starte med et stabilt grundlag til spillet og derefter tilføje de mindre essentielle dele af spillet.

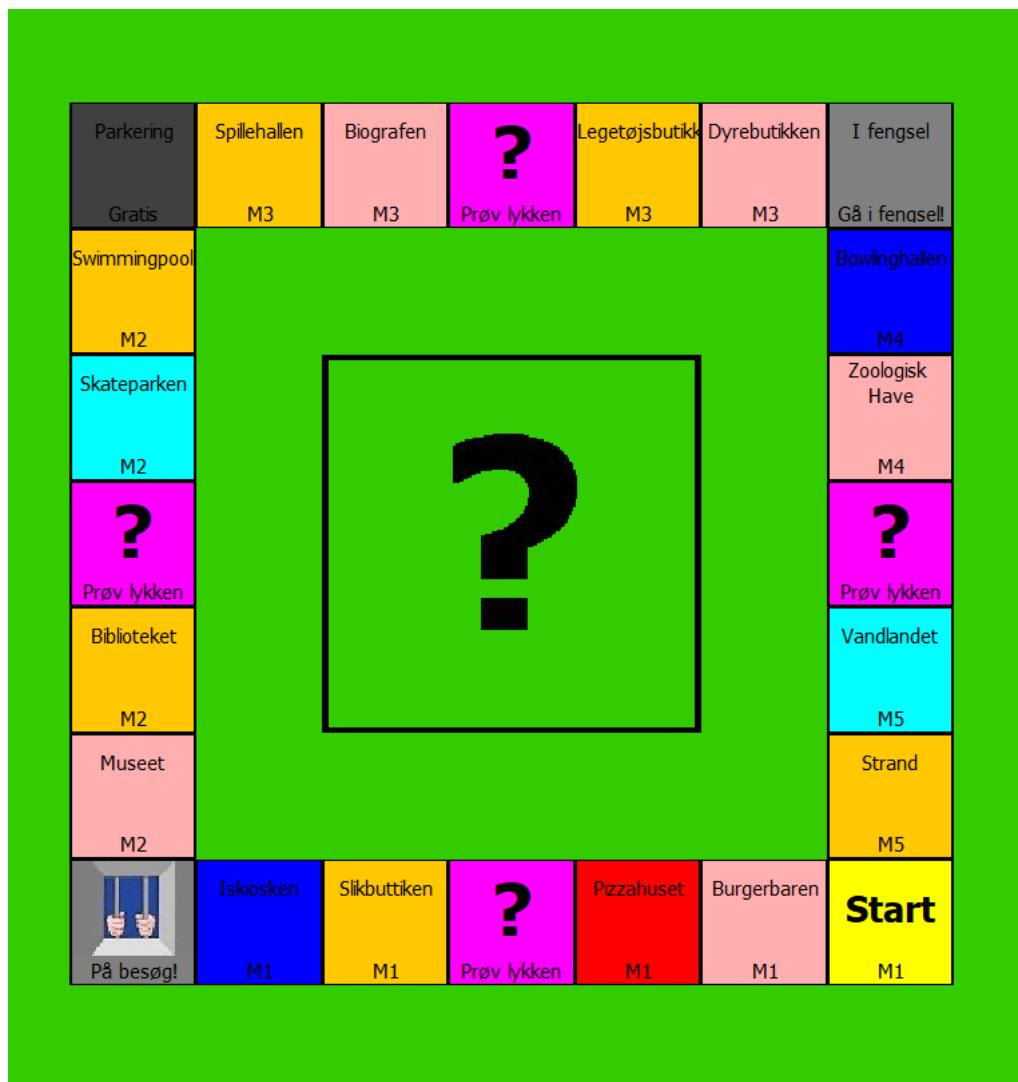
Dette system har vi udbygget gradvist, ved hjælp af Rational Unified Proces, altså ved brug af den iterative og inkrementelle metode. Vores kode er blevet bygget udefra de diagrammer vi har lavet, som selvfølgelig også var blevet opdateret løbende, for at undgå at ende ud i brugen af vandfaltsmetoden, som i vores tidligere forløb.

Opdelingen af arbejdet mellem gruppemedlemmerne, var ikke perfekt, men lettere overskueligt, idet vi arbejdede i klasser og forskellige segmenter uafhængigt af hinanden. Denne gang, har vi arbejdet uafhængigt af hinanden, med hver sin feature branch. Først da vi var sikre på at det hele virkede som det skulle, blev vores branches sammenfattet til master branchen. Næste gang vil vi dog vælge en bestemt gitflow på forhånd og en mere grundig konfigurationsstyring, for at strømline versionering og for at processen bliver gjort på samme måde hver gang.

Vi er nået alle vores Must-Have og Should-Have krav og ingen af de Could-Have og Will-Not-Have krav. Derfor kan vi konkluderer at vi har estimeret akkurat vores tid og evner.

9 Bilag

- Litteratur
 - Genbrug af kode fra vores CDIO 2, se : https://github.com/Besbes1/G_26_CDIO2
 - GRASP pattern er inspireret af <https://www.coursehero.com/file/26144650/GRASP-Monopoly-Examplepdf/>
 - Slides og materiale fra undervisningen (Udviklingsmetoder, Introductory Programming og Versionsstyring) er brugt til dokumentation.
- Kode: Gui modellen som vi fik uddelt fra lektoren



Figur 16: Gui Monopoly Game Board

-

Projektnavn	Gruppenavn	Afleveringsfrist			
CDIO 3	gruppe 26/32	25-11-2022 kl. 11:59 pm			
Opgaver	Timer	Prioritet	Måldato	Status	Noter
Projektplanlægning					
Projekt planlægning	0,5	2	04-11-2022	Complete	Melissa
Konfigurationsstyring	0,5	4	14-11-2022	Complete	Mélissa
Krav og analyse					
Kravspecificering	3	1	03-11-2022	Complete	Alle sammen
Usecase diagram	1	2	04-11-2022	Complete	Besma
brief usecase	0,25	2	04-11-2022	Complete	Amira
Fullydressed usecase	0,5	2	06-11-2022	Complete	Amira
Domæne model	0,5	2	04-11-2022	Complete	Negar
Systemsekvens diagram	0,75	2	04-11-2022	Complete	Melissa
Design					
Designklasse diagram	2	1	06-11-2022	Complete	Negar, Bayan, Mélissa
Sekvens diagram	3	2	06-11-2022	Complete	Negar, Bayan, Mélissa
Implementering					
<i>Lav et nyt projekt med klasser fra CDIO 2</i>				Complete	Mélissa
Account class (copying)	0	3	08-11-2022	Complete	
Die.class (copying)	0	3	08-11-2022	Complete	
Square.class (copy+mod)	1	3	08-11-2022	Complete	Mélissa
PlayerTurn.class (copy+mod)	0	3	08-11-2022	Complete	Will not be using this class afterall
Cup.class (copying)	0	3	08-11-2022	Complete	
Game.class	0,2	3	20-11-2022	Complete	Mélissa
<i>GameController</i>				Complete	Mélissa
GameController class	2	3	08-11-2022	Complete	Mélissa, Bayan
Mod Player class ifm GameController	1	3	08-11-2022	Complete	Mélissa
DeedSquare class	2	3	08-11-2022	Complete	Mélissa
BoardInit class	1	3	08-11-2022	Complete	Mélissa
Account class in relation to DeedSquare	0,5	3	08-11-2022	Complete	Mélissa
<i>Gui</i>				Complete	Negar, Mélissa
Gui til Chance kort	8			Complete	
Opsætning af Gui	12	4	14-11-2022	Complete	Negar

GuiController	19,5	4	21-11-2022	Complete	Negar
Gui (move car)	3,5			Complete	Mélissa
Gui (move car to jail)	2			Complete	Mélissa

Square subklasser

LandOnJailSquare	3	4	11-11-2022	Complete	Bayan
LandOnStartSquare	1	4	14-11-2022	Complete	Besma
<i>LandOnChanceSquare</i>			14-11-2022	Complete	David
<i>LandOnParkingSquare</i>	0,3	4	14-11-2022	Complete	Amira
Merging all the square features	1		08-11-2022	Complete	Mélissa

Translator

Writing text files for in-game text	2	3	14-11-2022	Complete	Bayan
Reader class (Text class)	9	3	20-11-2022	Complete	Mélissa
DeedInit.class	0,75	3	20-11-2022	Complete	Negar
Får spillet til at slutte	1			Complete	Mélissa, Bayan
Find vinder metode				Complete	Negar

Test

Sidste screenshot af alle tests bestået					Mélissa
Test	9,5	2	18-11-2022	Complete	Melissa, Besma, Negar, Bayan og Amira
Skriv Test Cases	5	3	23-11-2022	Complete	Negar, Mélissa, Bayan
Bruger test	1	4	24-11-2022	Complete	Amira

Rapportskrivning

Resumé	0,15	6	17-11-2022	Complete	Bayan
Indledning	0,5	6	17-11-2022	Complete	Bayan
Inddeling af krav MOSCOW	0,5	1	09-11-2022	Complete	Mélissa
Forklarende text til diagrammer	0,5	6	25-11-2022	Complete	Melissa, Besma, Negar, Bayan og Amira
<i>Dokumentation</i>		5	10-11-2022	Complete	Melissa, Besma, Negar, Bayan og Amira
Forklar arv	0,65	2	06-11-2022	Complete	Bayan
Forklar abstract	0,65	2	07-11-2022	Complete	Bayan
billededokumentation af test	0,7	5	16-11-2022	Complete	Negar
Forklar hvormen man importerer spillet fra GitHub (versionsstyring)	0,5			Complete	Mélissa

Forklar overhold GRASP	7	3	25-11-2022	Complete	Negar, Amira, Bayan og Melissa
Konklusion	0,6	6	23-11-2022	Complete	Bayan
Litteraturliste		6	25-11-2022	Complete	Bayan: jeg har sat vores cdio 2 som link
Bilag	0,2	6	25-11-2022	Complete	Bayan
Korrekturlæsning	2		25-11-2022	Complete	Besma

Diverse

Omdan til zipfil	timer	5	16-11-2022	Complete	Note
sikre dokumentation af artefakter		3	16-11-2022	Complete	Alle

Total 112,2