02314 62531 62532

Introductory Programming, Development Methods for IT Systems og Version Control and Testing Methods

CDIO 3

Group 22

s235068 - Casper Kielstrup

s235108 - Emil K. Petersen-Dalum

s235124 - Mirza Zia Beg

s235079 - Mohammed Ismail

s235771 - Younes Hamadi

s235084 - Nico Laursen

Deadline:

Friday, 24 November 2023

Indhold

Resume

Vi har fået til opgave og udvikle et Monopoly junior spil, hvor vi implementerer et spillebræt, chancekort og regler. Vores tilgang er baseret på principperne fra Introductory programming, development methods for IT-systems, og Version control and Test methods.

Timeregnskab

Hver person i gruppen har arbejdet 6 timer om ugen på Campus og mindst 5 timer om ugen i hjemmet.

Introduktion

Vi har videreudviklet på vores to forrige terningspil fra IOouterActive, og nu lavet et Monopoly junior spil. Vi har filføjet flere classes til felterne, chancekort og en playerchecker. Dette har vi gjort ved brug af vores viden fra Introductory programming, development methods for IT-systems og Version control and Test methods. I rapporten bliver der tilredelagt kravliste, use cases og passende UML-diagrammer.

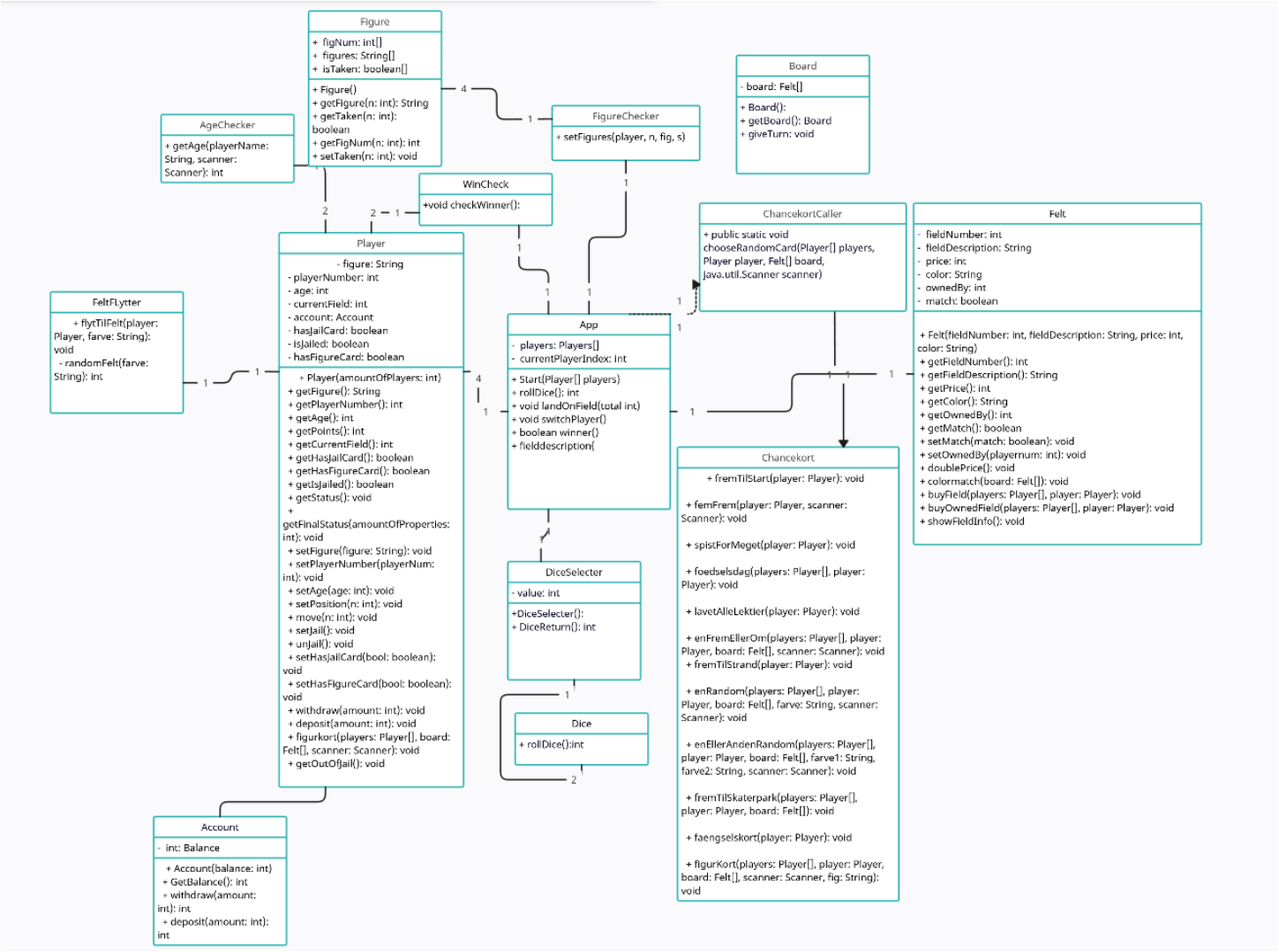
Projekt planlægning

Vi begyndte med at uddele opgaver til diagrammerne, og så begyndte vi at lave selve programmet ud fra diagrammerne. Undervejs dokumenterede vi også, hvad vi lavede.

Kravliste

- Spillet skal kunne indeholde 2-4 spillere.
- Implementer spillereglerne og chancekort.
- Man vinder spillet når en anden spiller går fallit og man har den højeste pengebeholdning.
- Spillerne skal hver starte med 20 penge.
- Hver gang en spiller passerer start, får de 2 penge.
- Spillerne skal kunne lande på et felt og fortsætte derfra i næste runde.
- Lav en klasse gameboard som indeholder alle felterne i et array.
- Skal udvikles efter OOP
- Lav mindst tre test cases med tilhørende fremgangsmåde.
- Lav mindst en Junit test til centrale metoder.
- Lav mindst en brugertest.

Class diagram:



Use case beskrivelser:

Use Case: Spil Monopoly Junior
Aktører
- Primære aktører: Spillere
- Sekundære aktører: Spillepladе, Terninger, Chancekort
Beskrivelse
Formål: Denne use case beskriver processen med at spille Monopoly Junior, hvor der er spillere som deltager i spillet for at vinde.
Hovedscenarie
1. Spillere starter spillet ved at vælge deres brik og placere den på startfeltet på spillepladen.
2. En spiller begynder sit træk ved at rulle to terninger.
3. Resultatet af terningekastet bestemmer, hvor mange felter spilleren skal flytte sin brik.
4. Spilleren flytter sin brik på spillepladen i overensstemmelse med resultatet af terningekastet.
5. Spilleren udfører handlinger baseret på det felt, de lander på, herunder køb af ejendom, betaling af leje, trækning af chancekort osv.
6. Spillet fortsætter i en cyklus, hvor spillerne skiftes til at trække terninger og flytte deres brikker.
7. Spillere har mulighed for at trække chancekort fra bunken i henhold til spillets regler.

8. Spillet fortsætter, indtil en spiller opfylder en forudbestemt vinderbetingelse, f.eks. at have en vis mængde penge eller ejendomme.
9. Når en spiller opfylder vinderbetingelsen, er spillet afsluttet, og vinderen bliver dermed fundet.
Alternative scenarier
- *Ingen terninger er rullet:* Hvis en spiller ikke ruller terningerne, går deres tur tabt, og spillet fortsætter med den næste spiller.
Forudsætninger
- Spilkomponenterne, inklusive spilleplade, terninger og chancekort, skal være korrekt opsat.
- Spillere er klar til at deltage i spillet og har valgt hver deres brik.
Efterspil
- En spiller vinder spillet og spillet afsluttes.
Specialkrav
- Spillet skal kunne håndtere 2-4 spillere.
- Spillet skal have en fyldestgørende mængde chancekort og ejendomme til at sikre variation og spænding.
- Reglerne for spillet skal følge Monopoly Junior-spillets officielle regler.

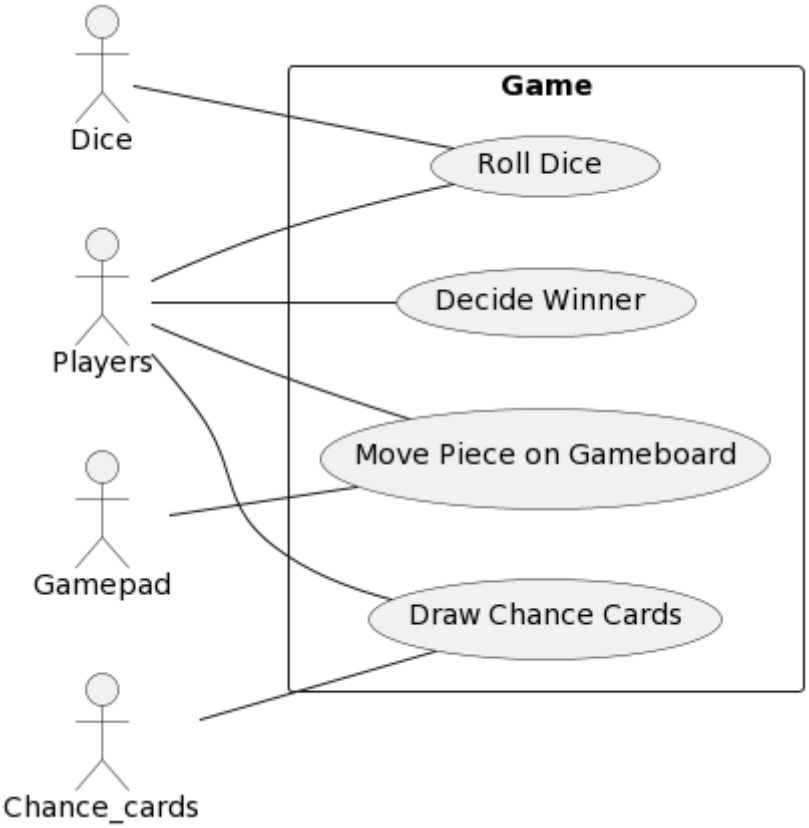
Use case: Rul terninger	
Aktører:	Spillere, Terninger
Beskrivelse:	Spillere skal have mulighed for at rulle to terninger for at bestemme, hvor mange felter de skal flytte deres brikker på spillepladen. Resultatet af terningekastet afgør spillets gang.

Use case: Flyt Brik på Spilleplade:	
Aktører:	Spillere, Spilleplade
Beskrivelse:	Når en spiller ruller terningerne, skal de have mulighed for at flytte deres brik på spillepladen i overensstemmelse med terningekastet. De skal kunne lande på forskellige typer af felter som har hver sin effekt.

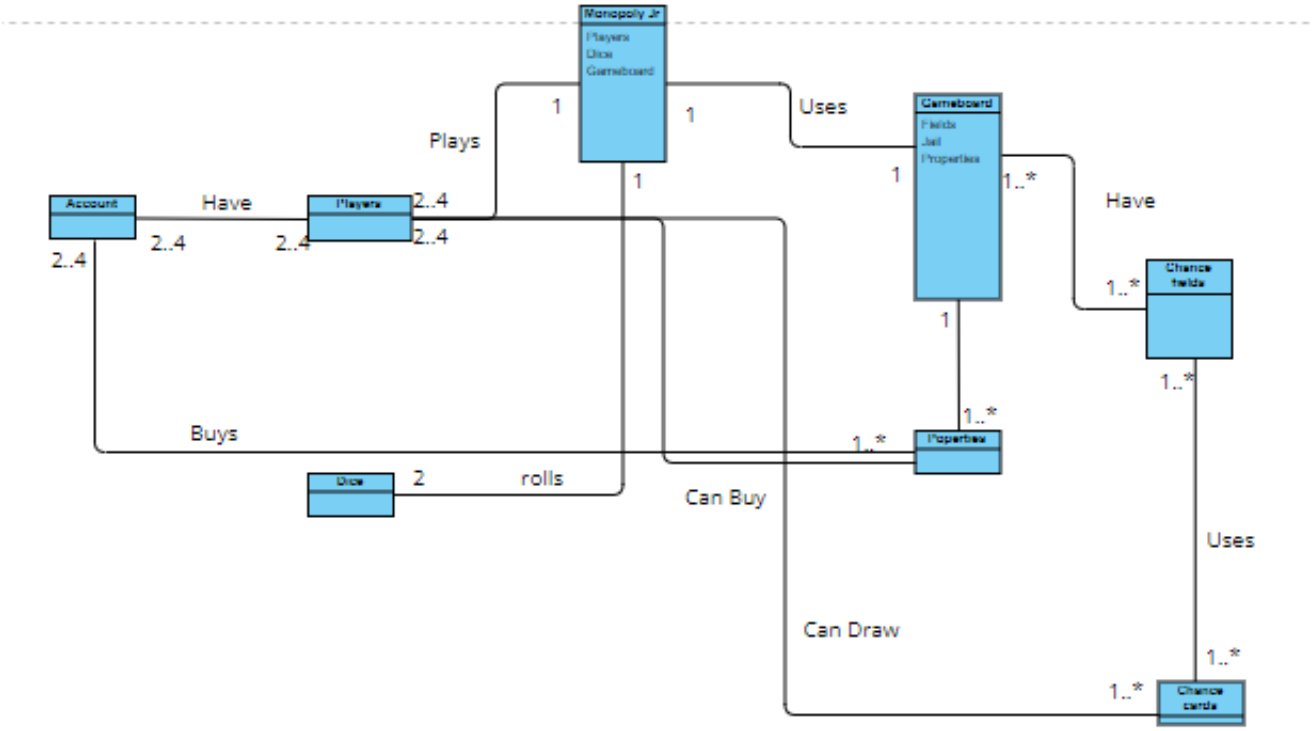
Use case: Træk Chancekort	
Aktører:	Spillere, Chancekort
Beskrivelse:	Spillere skal have mulighed for at trække chancekort fra bunken. Disse kort kan have positive eller negative konsekvenser for spillerne, såsom at give penge, kræve betalinger eller udføre specielle handlinger.

Use case: Afgør vinder	
Aktører:	Spillere
Beskrivelse:	Spillet fortsættes indtil en af spillerne når en bestemt mål, f.eks. at nå en vis mængde penge eller ejendomme. Når en spiller opfylder disse krav er spillet afsluttet, og vinderen afgøres.

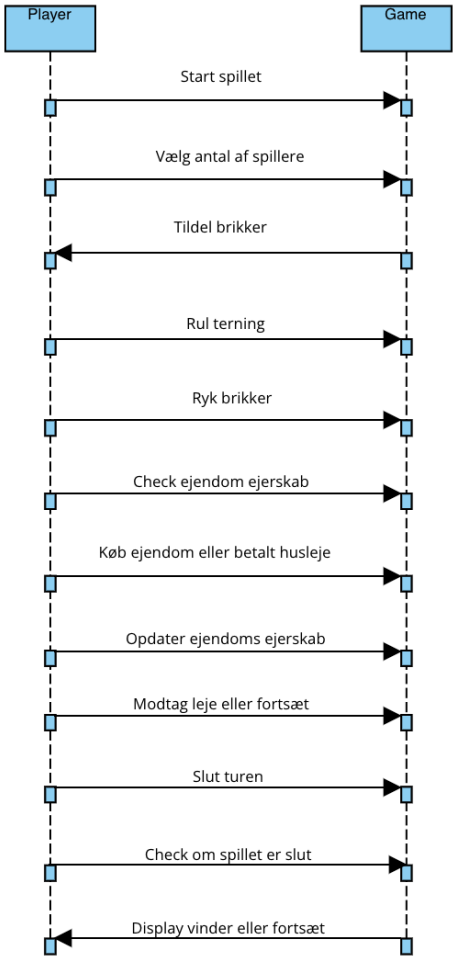
Usecase diagram



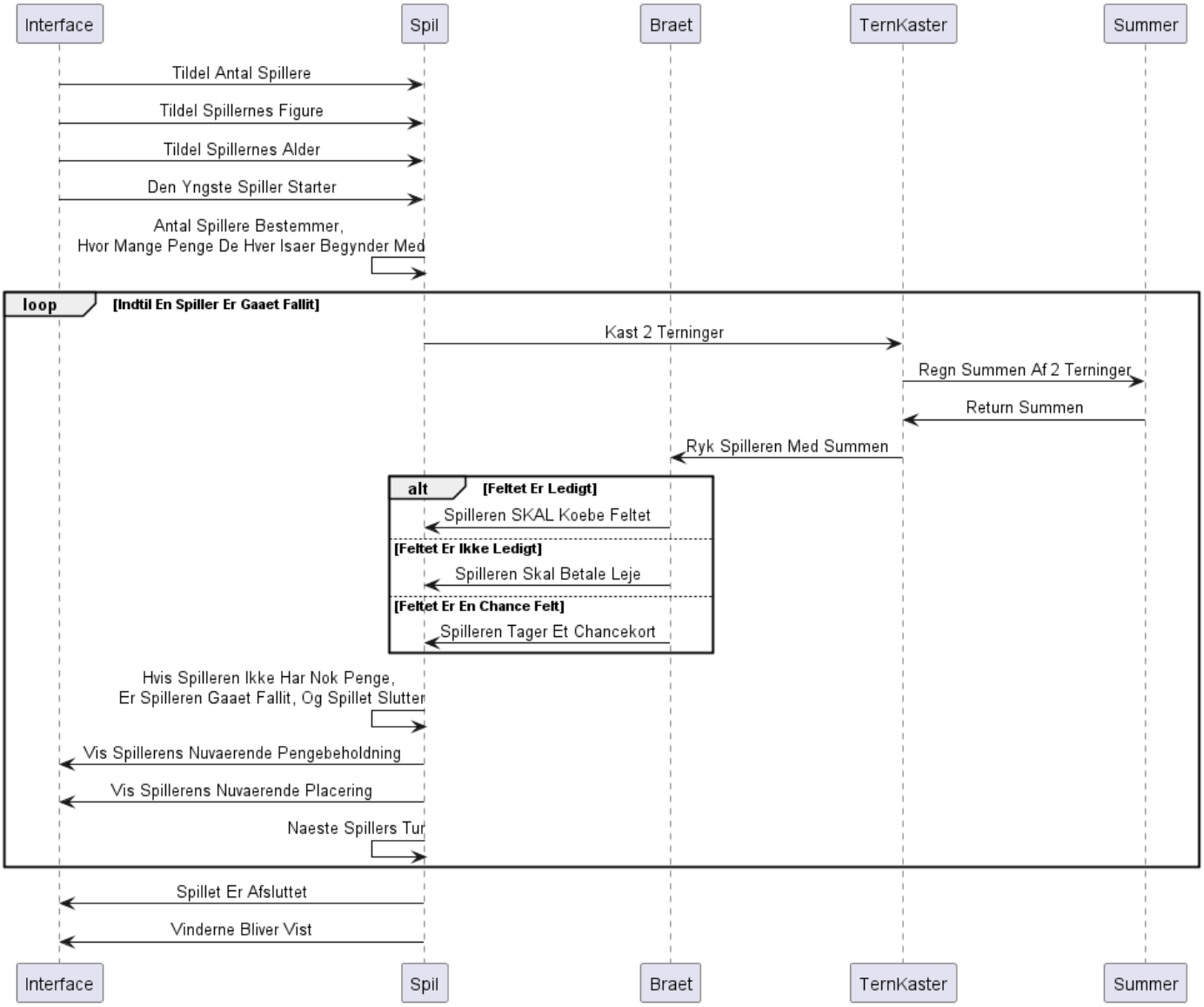
Domænemodel:



Systemsekvensdiagram:



Sekvensdiagram:



Implementering

Da det er et krav at programmet skal kunne køres på computerne i databarene på DTU, har vi valgt at implementere den ved brug af terminalen. Databarene hos DTU har Java 13 installeret, så programmet kan køres ved at kompilere filerne og så køre App. F.eks. hvis alle .java filerne er i én mappe, skal man åbne en terminal i den mappe, og skrive "javac *.java", og så "java App" efterfølgende.

Selve git repository kan importeres til Visual Studio Code (VS Code) ved at følge disse trin:

1. Åbne VS Code
2. Tast "ctrl + shift + p" (cmd + shift + p på Mac)
3. Vælg "Git: clone"
4. Tast https://github.com/ZabZZZZZ/22_CDIO_3 og tryk enter

Hvis du har fulgt disse trin, vil du kunne se alle filerne i vores git repository.

Testning

Test cases:

Test case	Action	Input	Expected output	Actual output	Test result
Age check (abnormal)	Enter age	4	Invalid input	Invalid input	Pass
Age check (extreme)	Enter age	120	Age = 120	Age = 120	Pass
Age check (normal)	Enter age	7	Age = 7	Age = 7	Pass
Figure check (abnormal)	Enter number between 1-4	0	Invalid input	Invalid input	Pass
Figure check (extreme)	Enter number between 1-4	1	Figure = "Bil"	Figure = "Bil"	Pass
Figure check (normal)	Enter number between 1-4	3	Figure = "Hund"	Figure = "Hund"	Pass
Figure check (abnormal)	Second player enters input when figure 1 is already taken	1	"Figure is already taken, pick another one"	Figure is already taken, pick another one"	Pass

Unit test:

Vi har lavet en test branch til JUnit tests, som kan blive testet i Visual Studio Code.

I denne test, tester vi Widthdraw og Deposit metoderne i Account klassen.

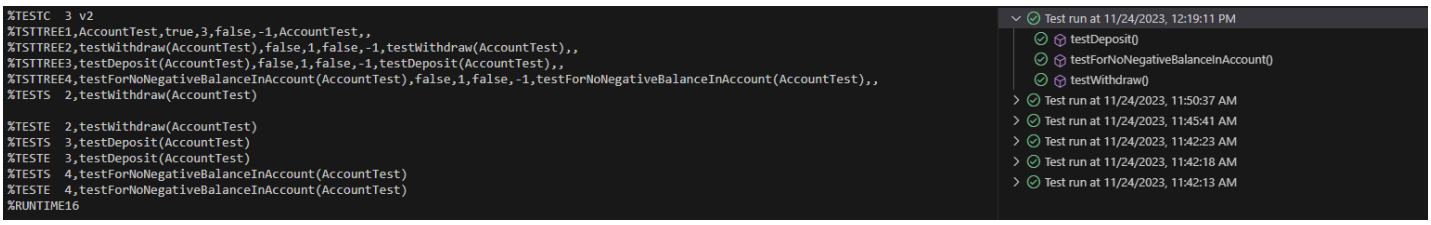
```
public class AccountTest {
    @Test
    public void testWithdraw () {
        Account account = new Account(balance:1000);
        account.withdraw(amount:200);
        assertEquals(800, account.GetBalance());
    }

    @Test
    public void testDeposit() {
        Account account = new Account(balance:1000);
        account.deposit(amount:300);
        assertEquals(1300, account.GetBalance());
    }

    @Test
    public void testForNoNegativeBalanceInAccount() {
        Account account = new Account(balance:1000);
        account.withdraw(amount:2000);
        assertEquals(0, account.GetBalance());
    }
}
```

1. I den første test, tester vi Widthdraw metoden af en konto med en balance på 1000, hvor vi trækker 200 fra kontoen. Forventet output er 800.
2. I den anden test, tester vi Deposit metoden af en konto med en balance på 1000, hvor vi indsætter 300 til kontoen. Forventet output er 1300.
3. I den tredje test, tester vi Widthdraw metoden af en konto med en balance på 1000, hvor vi trækker 2000 fra kontoen, for at tjekke om balancen går i minus, hvilket den ikke skal. Forventet output er 0.

Denne kode giver outputtet:



Tydeligvis giver testene de rigtige resultater.

Bruger test:

Vi gav den til vores forældre, som har ingen erfaring med programmering. Vi havde på forhånd implementeret nogle if og else conditions i de funktioner som krævede user input. Til håndteringen af forkert input, i de fleste tilfælde satte vi user-input funktionerne i et loop, hvilket tillod at en hel runde af spillet kunne gennemføres af vores "testere", uden fejl eller brud af reglerne. I det hele taget gik det fint med testen, og de kom igennem et helt spil uden nogle problemer.

Konklusion

Vi kan se, at de krav der blev stillet til opgaven, er opfyldt. De krav der er blevet givet, som er mere baseret på kode, er blevet opfyld i vores kode, så som antal spillere, gameboard og chancekort, samt at de givne start-værdier er brugt i koden. Vi kan også se, at vi har lavet en JUnit test på vores Withdraw og Deposit metoder i Account, og at denne test viser at metoderne virker som forventet. Vores Test Cases er baseret på de metoder der forventes før spillet kan startes, og de forskellige input spillerne kan lave. Vi kan se, at dissse også virker som de skal. Vi kan også se på vores brugertest, at spillere der ikke kender noget til hvordan koden virker, kan spille spillet uden problemer.
Vi kan derfor konkludere at alle krav er opfyldt.

Bilag

Litteratur:

Klasser fra vores forrige CDIO projekt:

1. Dice
2. Account

Klasser som vi har genbrugt, men ændret på:

1. Felt
2. WinCheck
3. Introduktion
4. Player

Kode:

- https://github.com/ZabZZZZZ/22_CDIO_3