CDIO 2

02312 62531 62532
Indledende programmering, Udviklingsmetoder til IT-systemer og versionsstyring og testmetoder.

Emil Krøldrup
s224276

Matthias Bruun Kobbernagel
s214662

Filip Merlung
s180436

Anders Nørgaard
s224305

Noah Ravn Rasmussen Sewerin
s225822

28. Oktober 2022

GRUPPE 5
DIPLOM SOFTWARETEKNOLOGI

1 Resumé

Per bestilling fra kunden har vi udviklet et spil med 2 spillere, der efter tur slår med 2 terninger, og alt efter summen af terningerne lander de på felter de ændrer deres pengebeholdning. Hver spiller starter med en pengebeholdning på 1000 og for at vinde skal en spiller have 3000.

2 Indhold

Indhold

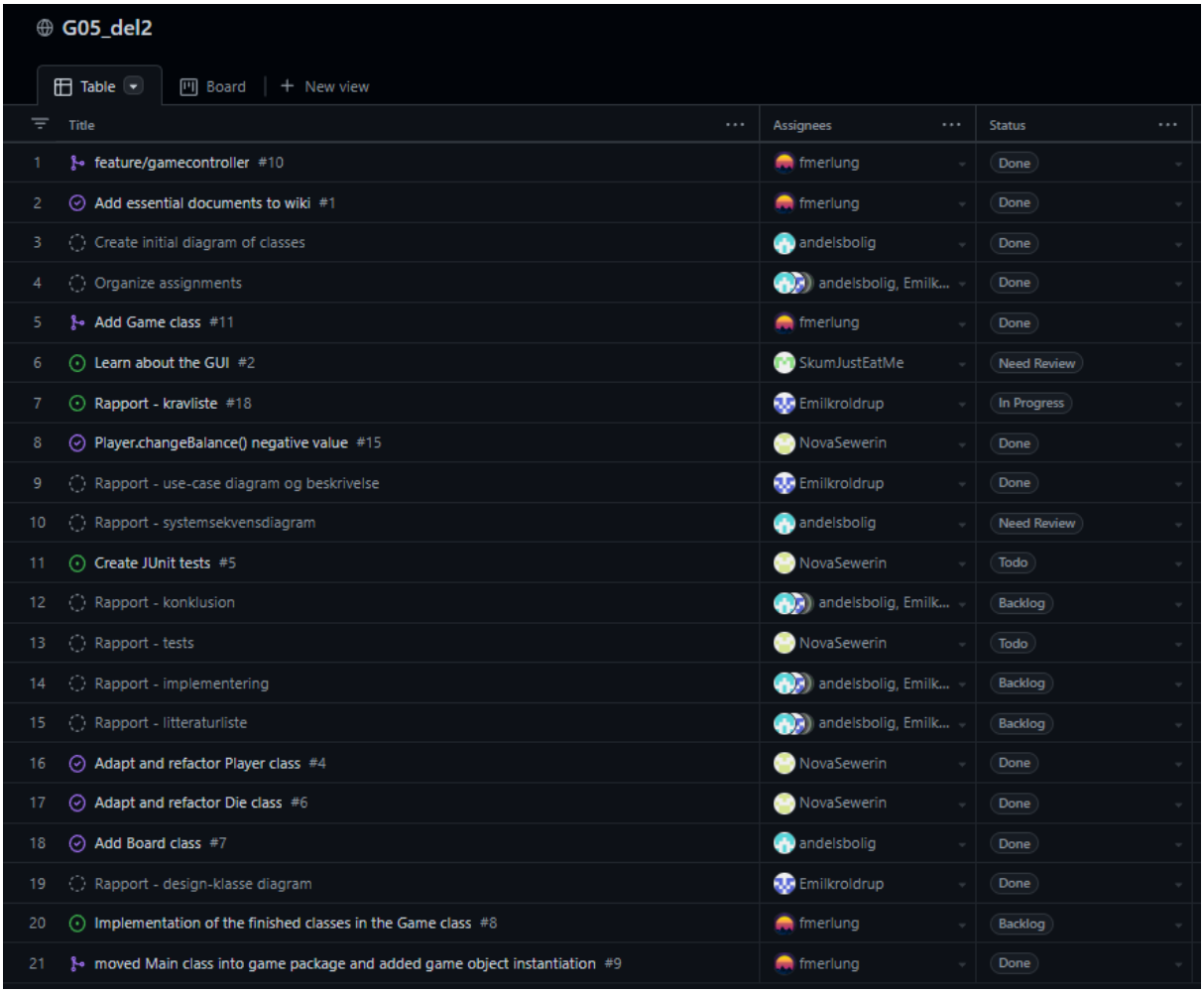
3 Indledning

Her er en kort rapport om et system, der kan spille et brætspil med nogle felter og terninger. I denne rapport har vi kort beskrevet vores metoder og lavet modeller ved UML. Så har vi lavet en kravspecifikation ud fra ønsker og samarbejde med projektleder og kunden.

4 Projektplanlægning

Inde på GitHub i vores repository har vi oprettet et projekt(se Figur 1), hvor vi i det projekt har oprettet en masse issues og opgaver, over hvad vores rapport og kode skulle indeholde. Derefter har vi fordelt os på de forskellige issues, og hver fået nogle ansvarsområder. Inde på projektet, er der status over hver issue og opgave, så vi har kunne have et overblik over:

- Hvilke opgaver der skal laves.
- Hvilke opgaver der er påbegyndt.
- Hvilke opgaver der er færdige.
- Hvilke opgaver der skal bruges hjælp til.
- Hvilke opgaver der skal laves når koden er færdig.



Figur 1. G05_del2 project in GitHub Repo.

5 Krav/Analyse

5.1 Kravspecifikation

Det skal være et system der kan bruges på DTU databarende (Windows computere)

Man skal kunne spille 2 spillere, der kan skiftes

Systemet skal være fleksibelt nok til at kunne skifte terningerne ud med andre terninger.

Systemet skal være nemt at oversætte til andre sprog.

Systemet skal have 11 felter, spillerne kan lande på. Der giver hver deres konsekvenser, god eller dårlig:

Feltliste

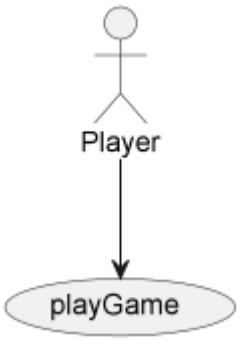
1. (Man kan ikke slå 1 med to terninger)
2. Tower +250
3. Crater -100
4. Palace gates +100
5. Cold Desert -20
6. Walled city +180
7. Monastery 0

8. Black cave -70
9. Huts in the mountain +60
10. The Werewall (werewolf-wall) -80, men spilleren får en ekstra tur.
11. The pit -50
12. Goldmine +650

Spillerne kan ikke gå under 0 i score/pengebeholdning

5.2 Analyse

En meget simpel use case til vores spil vil være:
Spil spil.
Det har vi også udarbejdet et meget simpelt use case diagram til(se Figur 2).



Figur 2. Use case diagram

BRIEF USE CASE BESKRIVELSER:

Board-klasse:
Main Success Scenario:
Gamecontroller får information om hvad der er blevet rullet med terningerne. Ud fra hvad der er blevet rullet vil “board” give information til gamecontroller om hvilket felt spilleren er landet på, hvor meget der skal ændres i spillerens pengebeholdning, og hvorvidt om spilleren får en ekstra tur.

Gamecontroller-klasse:

Main Success Scenario:
Gamecontroller sætter spiller 1 til at starte med at slå med terningerne ved at trykke på knappen der angiver man skal slå, hvorefter terningernes værdi bestemmer hvilket felt 1 lander på. Ud fra det vil spillerens pengebeholdning ændre sig, og måske vil spilleren få en tur til. Hvis spiller 1 ikke får en tur til gælder samme så foir spiller 2 bagefter. det fortsætter indtil spiller 1 eller 2 har opnået 3000 point og vinder.

Alternative scenarios:
Spiller 1 eller 2, trykker ikke på ‘enter’ og spillet går i stå.

Fully dressed use case.

Use case UC1: Spil spillet
Scope: CDIO 2 spil
Level: User Goal
Primary actor: Spiller
Stakeholders and interests:

- Spiller
    - Vil spille spil
- Kunde
    - Vil have et spil der opfylder krav, da de har smidt penge i projektet.
- IOOuterActive
    - Vil gerne have et spil der lever op til kundens behov, så de fremover kan udvikle flere spil for kunden og eventuelle andre kunder.

Preconditions:
Spillet er kørt.
Success guarantee:
Spillet fortsætter indtil spiller 1 eller 2 har vundet med 3000.
Main Success scenario:

1. Spiller # trykker på knap og slår med terningerne.
2. Spiller # rykker hen til felt der har samme nummer som terningesummen.
3. Spiller #'s pengebeholdning opdateres ud fra hvilket felt spiller # landede på.
4. Hvis spiller # slår 10, gentag step 1-4. Ellers fortsæt til step 5.
5. Spiller #'s tur er slut, næste spillers tur.
6. Gentag step 1-5 indtil en Spiller har opnået 3000, og vinder.

Extensions:

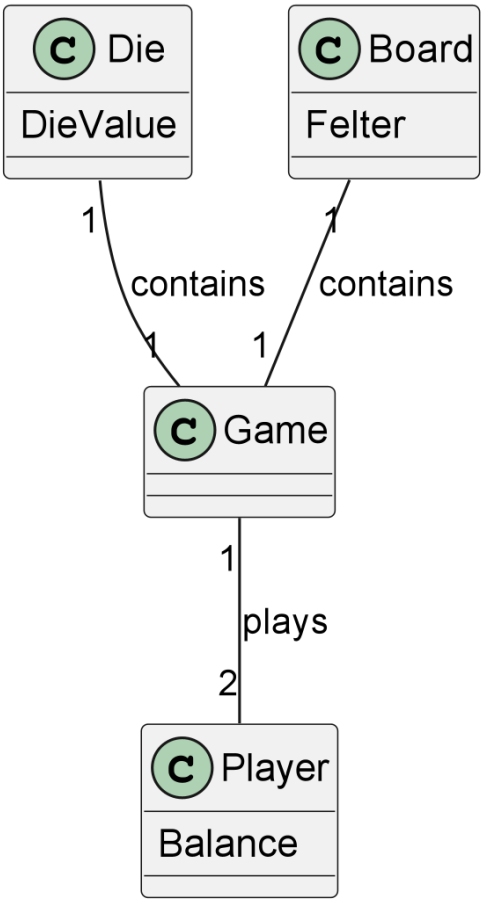
1. Spiller # trykker ikke ikke på knap.
   1.1 Spiller skal trykke på knap.

Special requirements:

- Systemet det køres på skal have Java version 17, eller nyere.

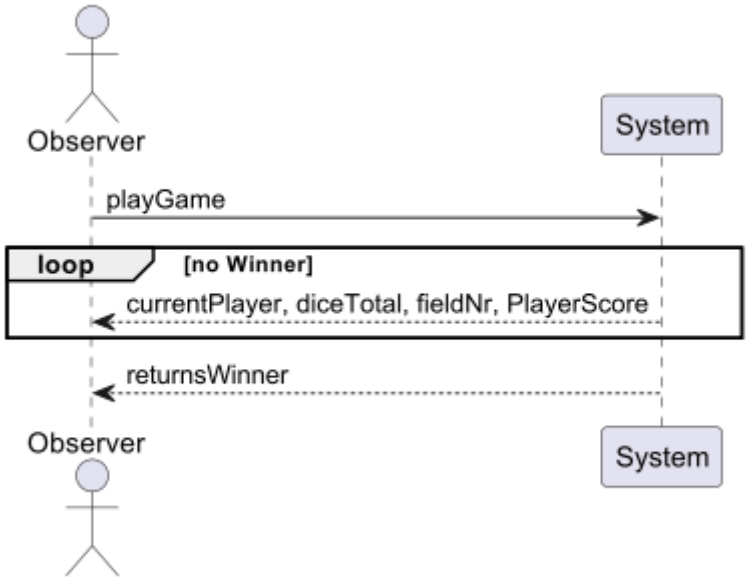
Analyse klassediagram

Vi har udarbejdet et analyse klassediagram(se Figur 3) som noget af det første for at danne os et overblik over hvilke klasser vi tænkte vi skulle bruge i vores kode.
Vi har vores 4 klasser, som består af
Die: Står for vores terningelogik og det at rulle med terningerne
Board: Står for det ønskede spils regler og logik
Game: Er vores controller der binder det hele sammen, og dermed kun henter den data den har brug for på daværende tidspunkt.
Player: Står for brugernes logik og pengebeholdning.



Figur 3: Analyse klassediagram.

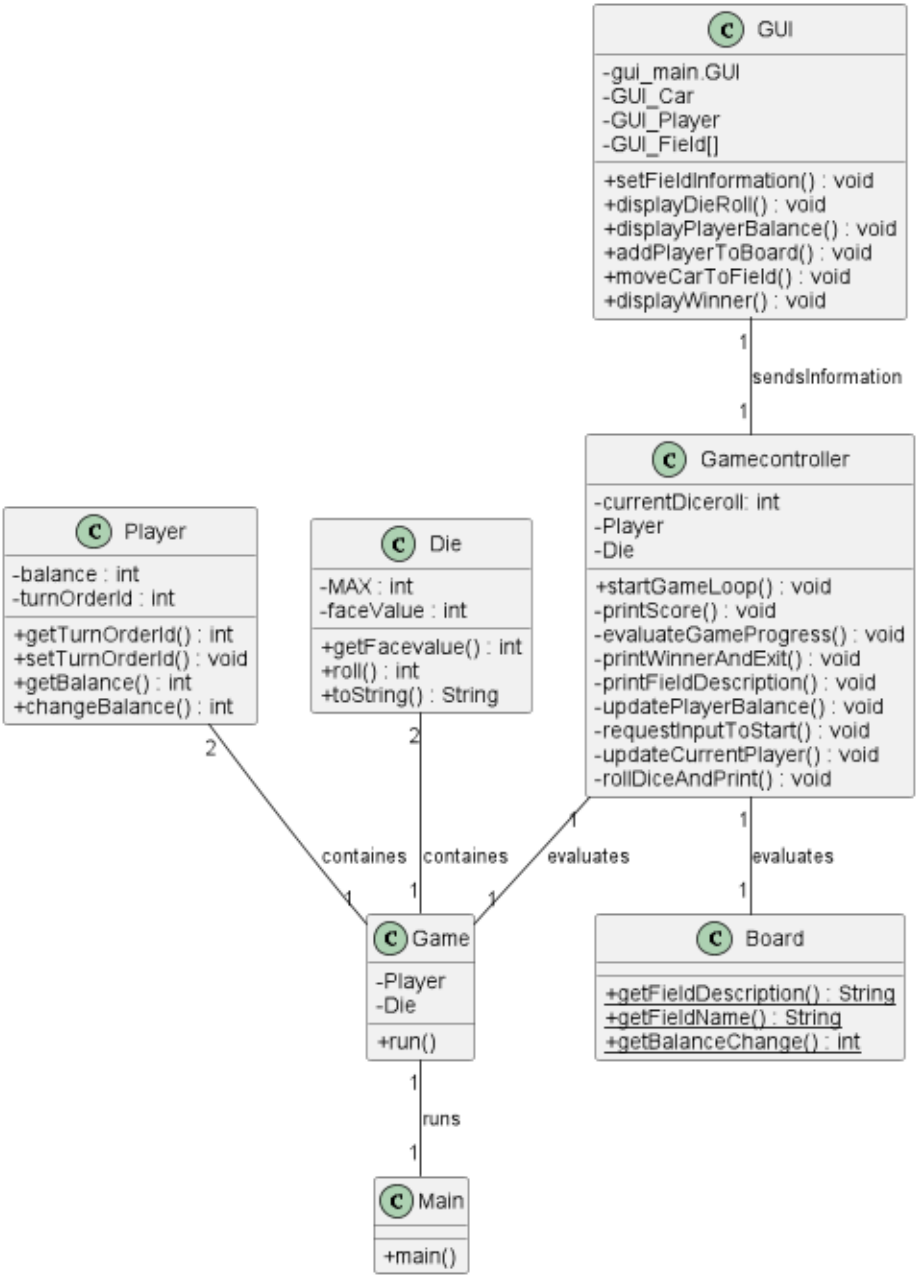
Herunder er oprettet et systemsekvensdiagram. Det første(se Figur 4) følger et meget simpelt use case over hele systemet “spil spil”, eller her “playGame”. Det kan så aflæses at spillet vil vende tilbage med nogle informationer om hvilken spillers tur det er, hvad de slår, hvilket felt de er landet på og hvad spillerens score er, indtil en vinder er fundet.



Figur 4. System sekvens diagram.

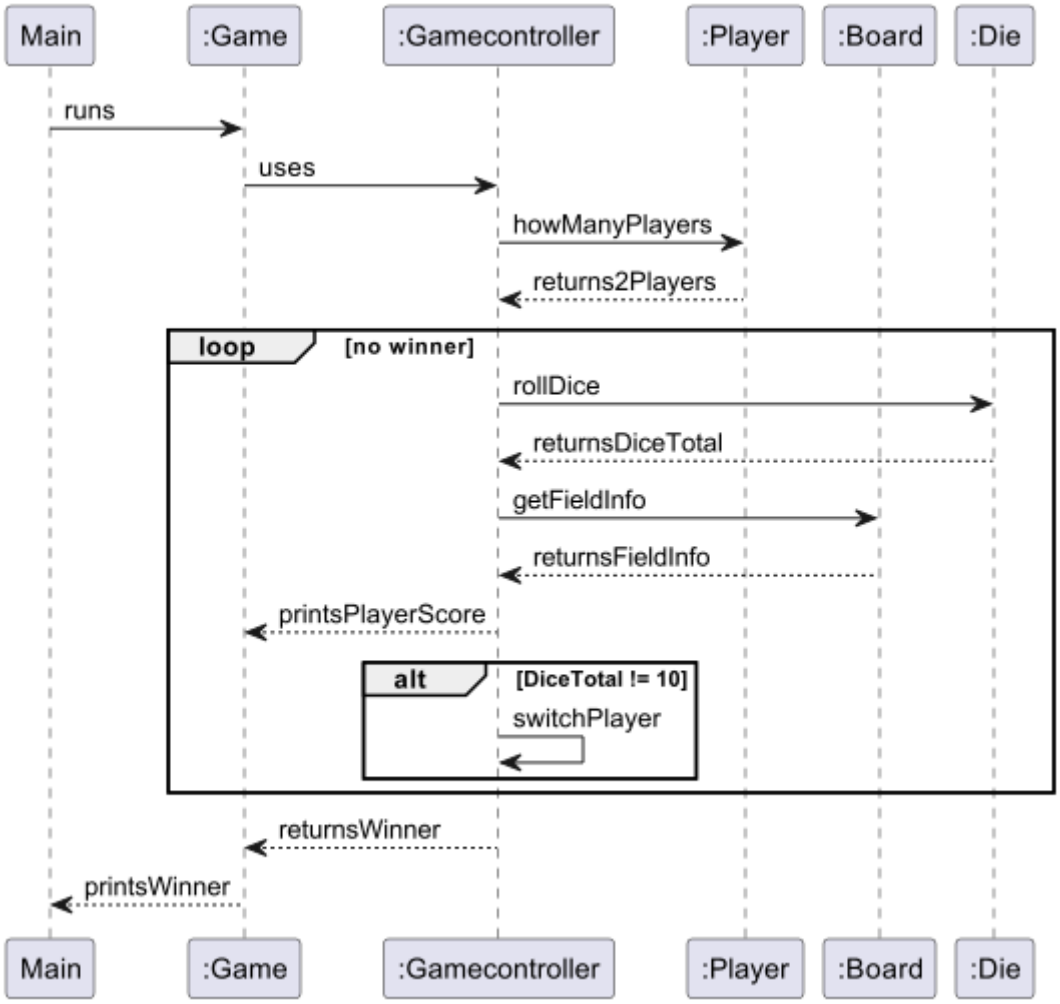
6 Design

Nedenfor ses vores færdige design klassediagram(se Figur 5):
Main kører spillet.
Game klasse, henter data fra Player og Die og videregiver dette til Gamecontroller.
Gamecontroller er vores overall logik i spillet, der får de forskellige klasser til at arbejde sammen.
Player, er der hvor spillerne er oprettet og bevarer deres progress
Die, er der vores terninger og rolls bliver genereret.
Board, styrer hvad der sker på vores GUI og effekterne for de forskellige felter.
GUI, er der hvor vi skaber funktionerne til det visuelle vi får lov at se med alt den data der kører bagved.



Figur 5. Design klassediagram.

Derudover er der lavet et sekvensdiagram(se Figur 6) der i grove træk viser hvordan de forskellige klasser skal arbejde sammen, men ikke et diagram over de reelle metoder i klasserne, men nogle pseudometoder som symboliserer de metoder der er i vores klasser. GUI'en er ikke indarbejdet i sekvensdiagrammet, den bliver påvirket af Gamecontroller hver gang der er kørt et loop og når en vinder er fundet.



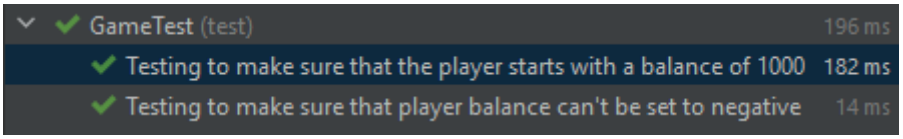
Figur 6. Sekvensdiagram.

7 Implementering

Programmet er skrevet ud fra GRASP principperne. Vi har et datalag af klasser som indeholder den nødvendige information, og en controller der kører spil-loopet og agerer som bindeled mellem user-interface og datalag. Vi har gjort brug af high cohesion og low coupling, så at hver klasse kun indeholder relevant information for at kunne udføre sin funktion, mens der er et minimum af overlap mellem klassernes indbyrdes dependencies.

8 Test

Der er lavet to tests for at sikre at programmet har den korrekte adfærd. Den første test undersøger om spilleren starter med en pengebeholdning på 1000. Den anden checker om spilleren kan få en pengebeholdning under 0, og giver en fejl hvis de kan. Det kan ses på skærmbilledet herunder at de to tests kører som de skal.



Koden til de to tests kan også findes i bilagsafsnittet.

9 Konklusion

Vi har udarbejdet det ønskede produkt. Vores spil kører efter hensigten og overholder/indeholder ovenstående krav. Yderligere har vi udarbejdet test der virker efter hensigten.

10 Bilag

10.1 Litteraturliste

Larman, C. (2005). *Applying Uml and patterns: An introduction to object-oriented analysis and design and the unified process* (3rd ed.). Prentice-Hall.

Lewis, J. & Loftus, W. (2018). *Java Software Solutions: Foundations of Program Design* (9th ed.). Pearson Education Limited.

CDIO Gruppe 5 (2022). *CDIO Del 1 - https://github.com/andelsbolig/o5_del1*

10.2 Kode

https://github.com/fmerlung/G05_del2.git