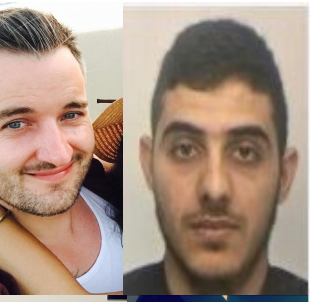


02312 - 62532 - 62531

Indledende programmering, Udviklingsmetoder til IT - Systemer og Versionsstyring og Testmetoder

---

CDIO 2 - Gruppe 16.

Thomas Arildtoft
S193564

Omar Ebrahim
S163576

Ali Dadayev
S172858

Salim Omar
S193472

David Lucas Møller
S173210

Jessica Meyer
S185192

Oktober 2020

Timefordeling:

Navn.	Timer.
Jessica	8
Salim	8
Omar	8
David	8
Ali	8
Thomas	8

Indholdsfortegnelse:

FURPS krav

Functional:
- Der skal være 2 spillere
- Der skal være felt og de skal være mellem 2-12
- Spilleren skal ikke slå selv terningerne, programmet skal gøre det for ham
- Felterne skal kunne reagere negativ eller positiv og den skal gå ud over spillerens pengepung.
- Der skal printes et teks ud når spilleren lander på de forskellige felter.
- Spilleren skal have 1000 når han/hun start (et start gevinst).

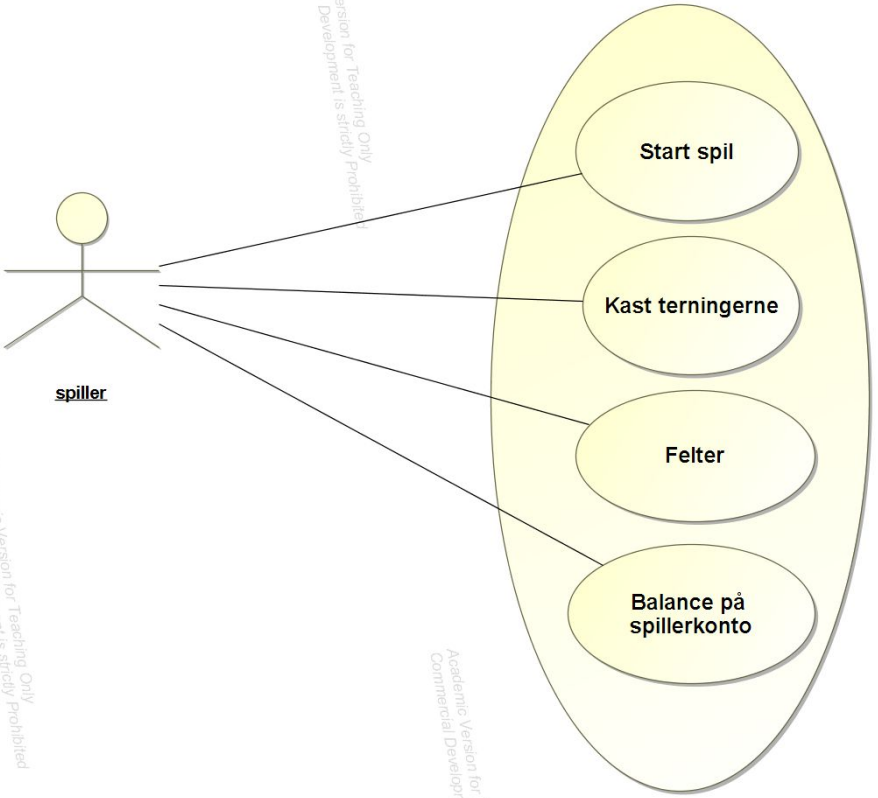
Usability:
- Spillet skal kunne spilles af en bruger, som ikke kender så meget til teknologi
- Spillet skal være intuitiv og forståelig

Reliability:
- Spillet skal kunne give besked, hvis der er noget der fejler, samt med fejlbeskrivelsen

Performance:
- Spillet skal starte op max efter 2 sekunder, når brugeren trykker på start
- Der skal maks. være 2 sekunder i mellem hver kast af terningerne og opnåelse af resultat

Supportability:
- Spillet skal være let at overholde og let at lave ændringerne
- Spillet skal kunne installeres let og uden høje krav fra brugeren

Use Case Diagram:



Use case (Brief)

1. **Start Spil:**

   Actor = Spillerne

   Goal = Vælge karakter, indtaste navn og starte spillet op.

   Brief = Spilleren skal vælge imellem spiller 1 og spiller 2 og tilføje karakteren et navn efter eget valg. Derefter skal spillerne tryk på start for at starte spillet op.

2. **Kast terningerne:**

   Actor = Spillerne

   Goal = Terningerne kastes og vi får vist en værdi mellem 2 - 12.

   Brief = Spillerne kaster terningerne og skal få vist en talværdi 2 - 12, derefter skal turen gå til den næste spiller, som så også skal kaste terningerne. Dette gentages indtil at en spiller har vundet.

3. **Spilleren lander på felter:**

   Actor = Spillerne

   Goal = At der er 10 felter som spillerne kan lande på.

   Brief = Når spillerne får givet et antal øjne, skal de lande på det korrekt felt med den samme talværdi som øjnene viser, hvert felt har en talværdi tilknyttet som påvirket pengebeholdningen hos begge spillere.

4. **Spillerens resultater ændrer balancen**

   Actor = Spillerne

   Goal = At holde styr på spillernes balance på deres konto.

   Brief = Alle spillerne skal starte 1000 i pengebeholdningen, denne beholdning bliver påvirket enten positivt eller negativt alt efter hvilket felt man lander på i løbet af spillet. Når en spiller opnår 3000 på deres konto er spillet slut og denne spiller har vundet.

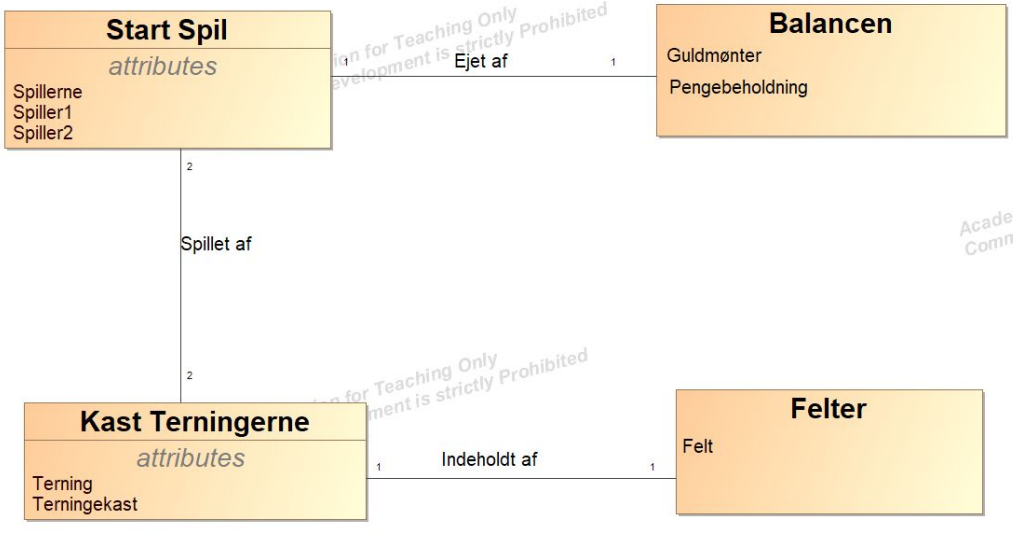
5. Fully Dressed Use Case (Spillerens resultater ændrer balancen)

Use case sektion:	Beskrivelse:
Navn	Spillerens resultater ændrer balancen
Scope	Balance
Level/niveau	Tager de værdier som er på felterne i spillet og enten addere eller subtraktion fra den samlede værdi på Balancen.
Primær aktør	Spilleren.
Andre interessenter	Ingen.
Forudsætninger/preconditions	At der står 1000 points på balancen.
Succeskriterier/postconditions	Ændring i balancen
Vigtigste succes scenarie	At balancen ændrer sig ved hvert terningkast.
Udvidelser	Spillet slutter når balancen opnår enten 3000 eller 0.
Specielle krav	
Teknologi og dataliste	Virke på alle OS og vigtigst på databarens computere.
Frekvens	Hver gang en spiller kaster terningerne.
Diverse	Skal kunne bruges i andre spil.

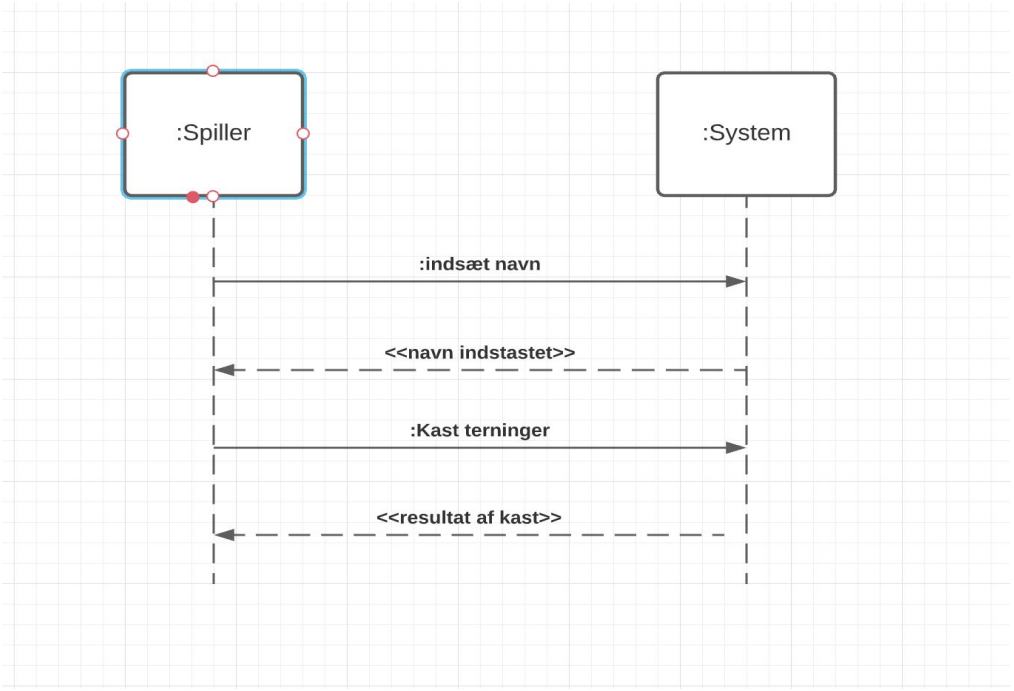
Navneord:

1. **Spil,**
2. **Spiller1**
3. **Spiller2**
4. **Terning,**
5. **Felt**
6. **Tur**
7. **Guldmønter**
8. **Pengebeholdning**
9. **Navn**
10. **Effekt**
11. **Terningekast**

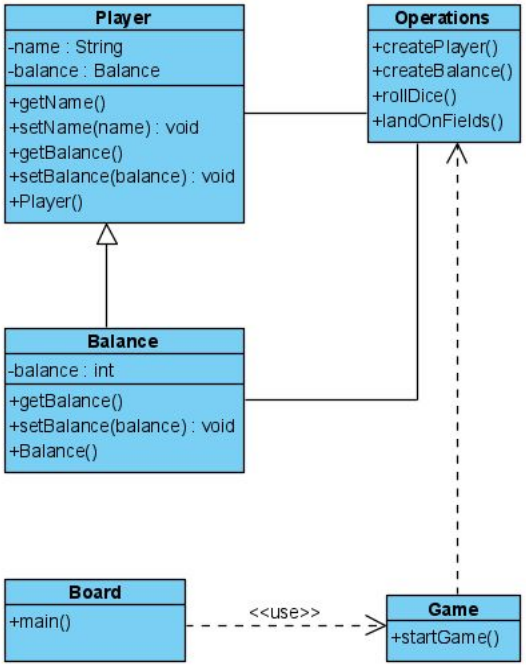
Domænemodel:



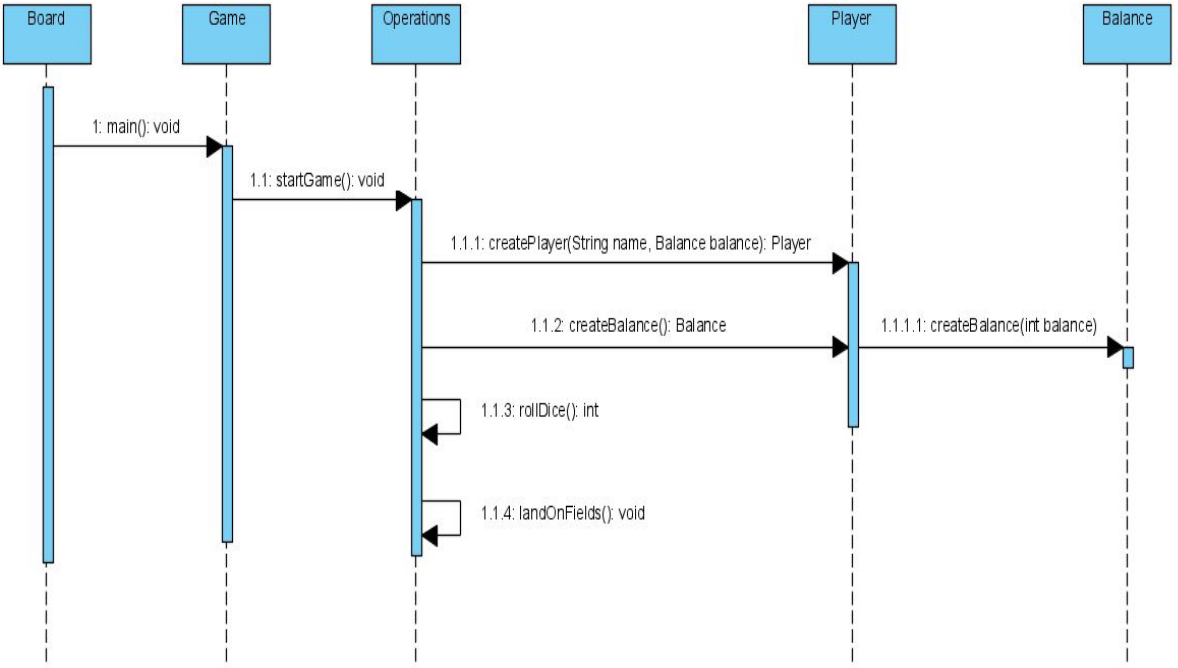
Systemsekvensdiagram:



Design Klassediagram:



Sekvensdiagram:



GRASP- mønster:

High cohesion:

Vi har oprettet vores objekter sådan, så de har begrænset område og de er meget klart. Af de hoved objekter har vi for eksempel Player og Balance og de begge har forskellige ansvarsområder. Balance er en attribut i Player klassen, samt med navnet. Så når man skal oprette en ny balance, som er tilknyttet til Player, kan man gøre det via Player objektet. Man kan også gøre oprette Balance separat og tilknytte det til Playeren senere.

Low coupling:

Vi har lavet vores program med Low Coupling, fordi alle klasse som eksisterer i vores program kan let fjernes, genbruges eller udskiftes.

Creator:

Vi har en Creator mønster i vores program og i vores tilfælde er det en klasse, som hedder Operations. Operations er ansvarlig for at oprette de mest nødvendige

objekter for programmet, såsom et nyt Player og ny Balance. Det er også en klasse, som opretter felter og kaster terninger.

Information expert:

Vi har også brugt Information Expert mønstret i programmet, og den hedder Game. Vores Game klassen indsamler alle de informationer, som vores Creator opretter, og udfører operationer.