CDIO3

Gruppe 16

10. november 2017

Mathias
Fager
s175182

Milishia
Moardi
s175193

Nicki
Christiansen
s170208

Semi
Seitovski
s175181

Simon
Pedersen
s175195

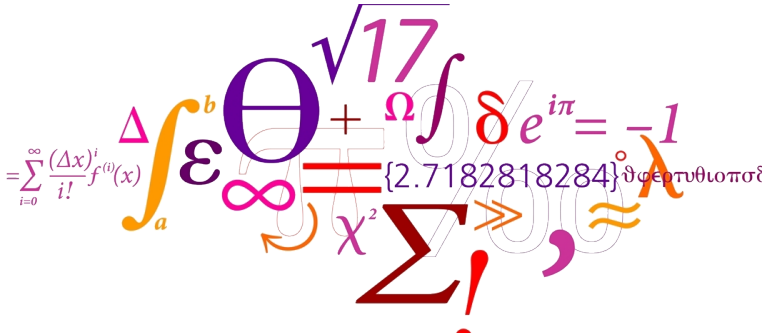
Thyge
S. Steffensen
s175176

Timeregnskab

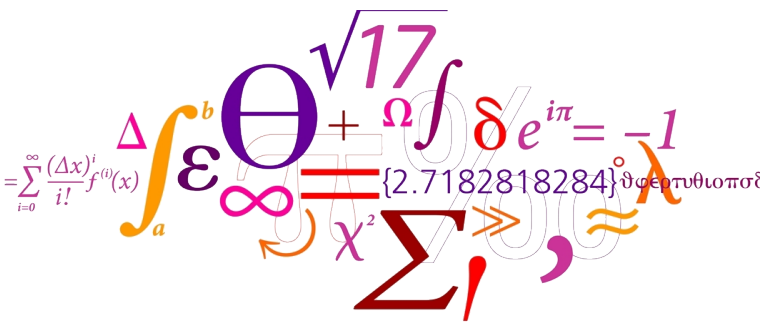
Timeregnskabet kan ses via linket her eller i billag XX.

Deltager	Total timer per deltager
Mathias Fager	3
Milishia Moradi	3,5
Nicki Christiansen	10
Semi Seitovski	6
Simon Steen Pedersen	5,5
Thyge Steffensen	14,5
Total	**42,5**

Figur 1: Overblik over Timeregnskab den 21/11/2017



Indhold



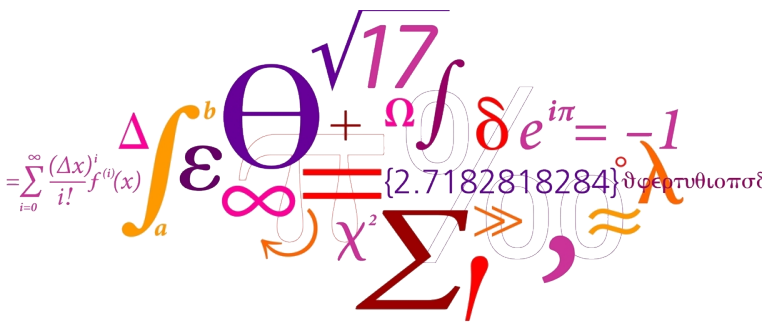
1 Analyse

1.1 Kravliste

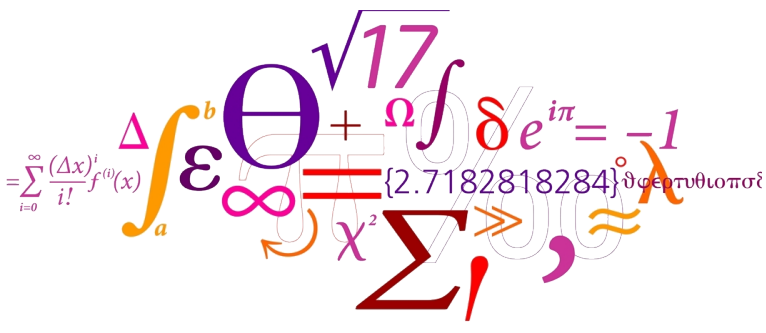
1. Spillet skal være mellem 2 - 4 personer
2. Spillerne skal slå terninger på skift
3. Der skal udskrives en tekst der omhandler det aktuelle felt, når spilleren lander på et felt
4. Hver felt skal have en effekt for spilleren
5. Spillerne starter med en balance på 30.000
6. Spillet slutter når alle undtagen en spiller er bankerot, altså når deres balance er nået 0
7. Spillerne skal kunne gå flere omgange rundt på spillepladen
8. Spillet skal kunne kører på DTU's databarer

1.2 UseCases

UseCase Section: Start af spil	Comment
Scope	Monopoly spil af IOOuterActive
Level	User-goal
Primær Aktør	IOOuterActive
Stakeholder og interessenter	IOOuterActive er interesseret i at spillerne skal kunne starte spillet
Forudsætninger	Spillet er installeret på enheden
Success guaranti	Spillet starter og spillerne bliver sendt videre til opsætning



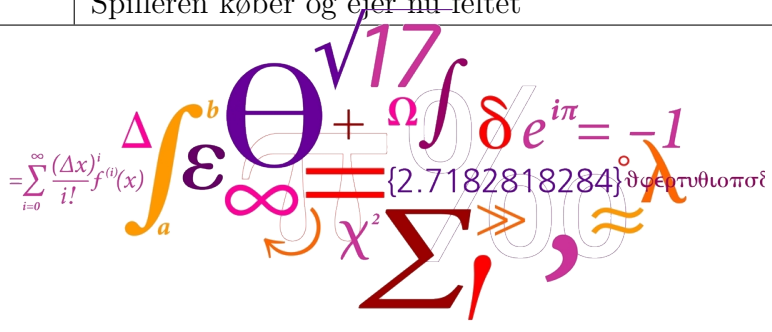
UseCase Section: Opsætning af spil	Comment
Scope	Monopoly spil af IOOuterActive
Level	User-goal
Primær Aktør	Spillerne
Stakeholder og interessenter	Spillerne er interesserede i at kunne vælge antal spillere og deres brikker
Forudsætninger	Spillet er startet op, og spillerne har nu mulighed for at vælge antal spillere og ønskede brikker
Success guaranti	Der er blevet valgt antallet af spillere, og hver spiller har valgt sin brik, herefter er spillet klar til at blive spillet



Fully dressed UseCase:

UseCase Section: Spillerne slår med terningerne	Comment
Scope	Monopoly spil af IOOuterActive
Level	User-goal
Primær Aktør	Spillerne
Stakeholder og interessenter	Spillerne er interesseret i at kunne trykke på en knap, og få et billede af to terninger med tilfældige værdier
Forudsætninger	Spillet er startet op, og spillerne har valgt antallet spillere og deres ønskede brikker
Success guaranti	Der er blevet valgt antallet af spillere, og hver spiller har valgt sit navn, herefter er spillet klar til at blive spillet
Hoved succes scenarie	Spillerne får udgivet en værdi af to terninger, og lander derefter på et felt
Alternative udfald	Negative udfald: - IOOuterActive har opdateret spillet, og derved opstår der en fejl når spillerne slå med terningerne, der kan ende i at der ikke bliver slået to terninger - Systemet blokerer for en spillers tur - En spiller hopper fra/på, og derved skal spillet startes om
Specielle krav	- Enheden som spillet kører på skal være kompatibel med Java - Spillerne skal kunne interagere med GUI'en ved brug af mus eller touch - Der skal være plads på enheden til at kunne hente spillet
Hyppighed	Hver tur bliver der slået med terninger

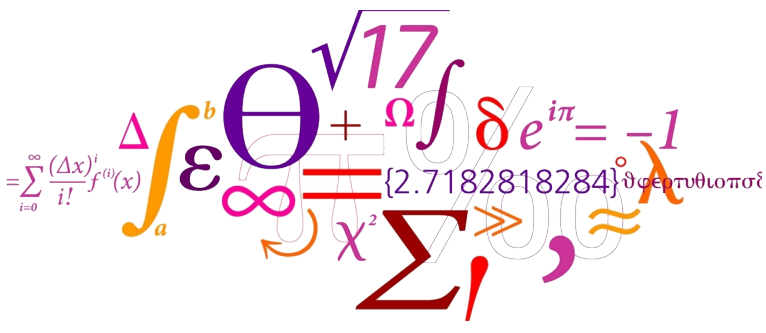
UseCase Section: Spiller køber et felt	Comment
Scope	Monopoly spil af IOOuterActive
Level	User-goal
Primær Aktør	Spillerne
Stakeholder og interessenter	Spillerne er interesseret i at købe det aktuelle felt
Forudsætninger	Spillet er i gang og en spiller har slået med terningerne
Success guaranti	Spilleren køber og ejer nu feltet



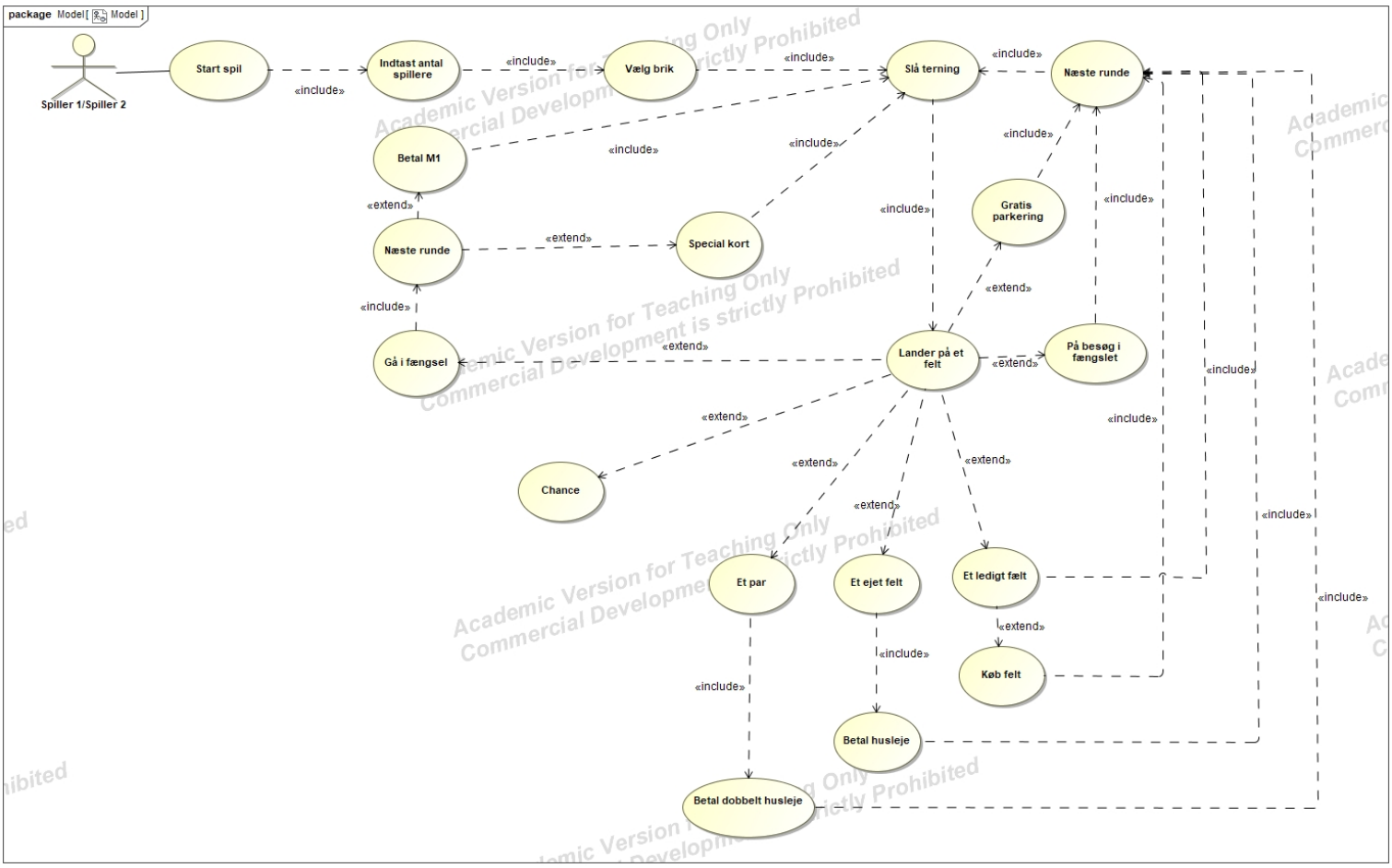
UseCase Section: Spiller lander på et ejet felt	Comment
Scope	Monopoly spil af IOOuterActive
Level	User-goal
Primær Aktør	Spillerne
Stakeholder og interessenter	****
Forudsætninger	Spillet er i gang og en spiller lander på et felt som er ejet af en anden spiller
Success guaranti	En spiller lander på et felt der er ejet af en anden spiller og får derfor en negativ effekt

UseCase Section: En spiller taber	Comment
Scope	Monopoly spil af IOOuterActive
Level	User-goal
Primær Aktør	Terningerne
Stakeholder og interessenter	Spillerne er interesseret i at deres balance ikke når 0, og dermed taber
Forudsætninger	Spillerne har slået med terningerne
Success guaranti	En spiller lander på 0, og er ude af spillet

UseCase Section: Spillet afsluttes	Comment
Scope	Monopoly spil af IOOuterActive
Level	User-goal
Primær Aktør	IOOuterActive
Stakeholder og interessenter	IOOuterActive er interesserede i at programmet viser en vinder og afsluttes
Forudsætninger	Alle spillere undtagen en, har fået en balance på 0
Success guaranti	Spillet viser en vinder og kan derefter afsluttes



1.3 UseCase diagram



Figur 2: UseCase diagram tegnet i MagicDraw

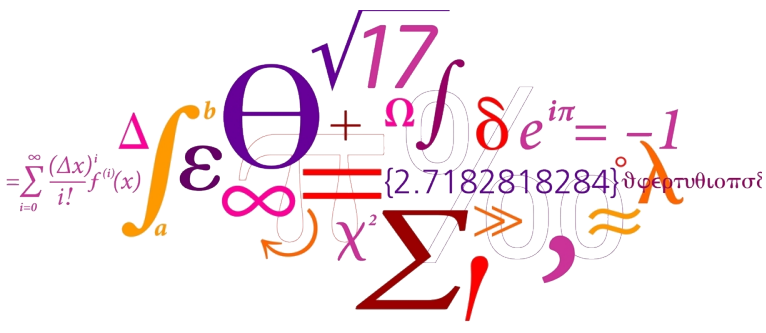
1.4 GRASP

GRASP står for General Responsibility Assignment Software Patterns. GRASP bruges til at give det rigtige ansvar til de forskellige klasser der bliver oprettet under udviklingen af et program. GRASP indeholder 9 patterns. Patterns bliver brugt til at strukturere et problem, samt at finde en passende løsning. De 9 patterns er:

1. Creator
2. Information expert

3. Low coupling
4. Controller
5. High cohesion
6. Indirection
7. Polymorphism
8. Protected variations
9. Pure fabrication

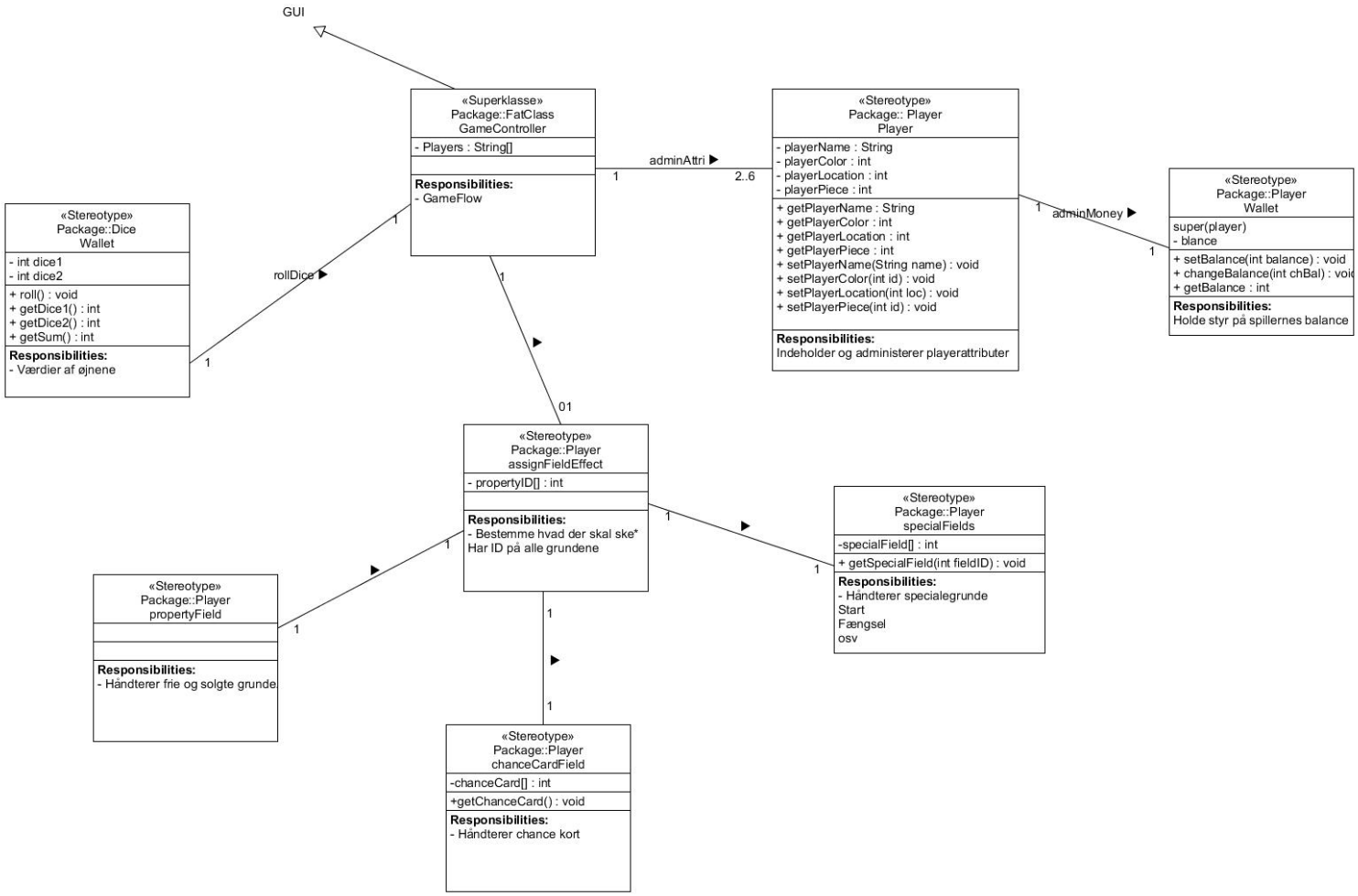
(Der skrives mere når vi er nået længere i projektet)



2 Design

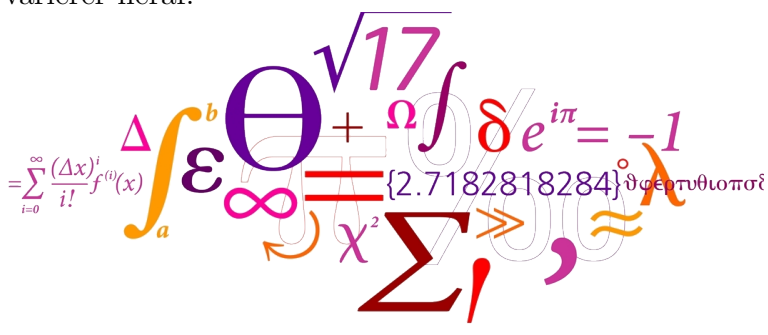
Herunder ses en række 'design steps', som skal hjælpe os med at lave Monopoly Junior spillet.

2.1 Klasse diagram



Figur 3: Klasse diagram tegnet i UMLet

Klasse diagrammet bygger på vores umiddelbare overvejelser, såvel som vores use case's. Dette er for at illusterer sammenspillet mellem vores klasser og deres associationer. Dog skal det siges, at dette er en skitse og den aktuelle programmering kan varierer heraf.



2.2 Sekvensdiagram

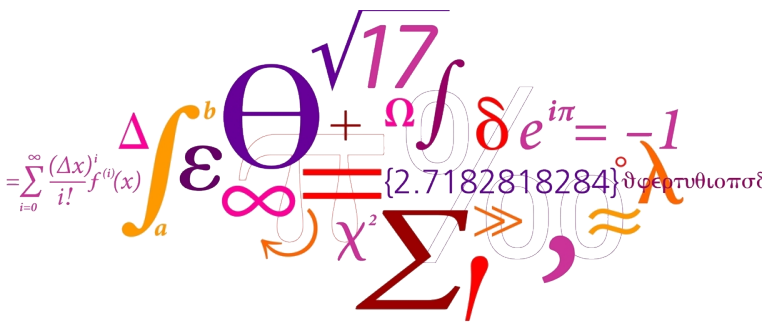
Vi har her lavet et sekvensdiagram, der skal skabe et overblik over hvordan aktøren, her spilleren, kommunikerer med spillet.

2.3 System sekvensdiagram

Vi har her lavet et systemsekvensdiagram for at forhøje gennemsigtigheden ved bruge af 'chanceCard' klassen.

2.4 Domænemodel

Ved hjælp af domænemodellen vil vi trække paraleller mellem den virkelige verden og programmeringen. Domænemodellen er en visuel repræsentation af konceptklasser og 'objekter fra den virkelige verden'. Ved hjælp af denne kan vi også oplyse kunden om, hvad vi vil lave.



3 Dokumentation

3.1 Forklar hvad arv er

3.2 Forklar hvad abtract betyder

3.3 Fortæl hvad det hedder hvis alle fieldklasserne har en landOnField metode der gør noget forskelligt

Hvis alle fieldklasserne gør brug af den samme landOnField metode er det fordi denne metode er en super metode nedarvet til de forskellige subklasser, dette gør at alle klasserne kan gøre brug af samme metode. Eksempeltvis, hvis vi har en masse dyr som klasser, kat, hund, kanin etc. så kan de alle nedarve super metoden eat() fra superklassen som hedder SurvivalRequirements, eftersom disse er metoder alle dyrene får brug for, så vil det give mening at lave det til en superklasse med supermetoder , så der holdes lav kobling og høj kohæsion, samt undgås kopiring af kode og høj mulighed for genbrug.

3.4 Dokumentation for test med screenshots

3.5 JUnit test

JUnit test er en autonomisreret testmetode. Her skriver/koder man selv en test, som tester java kode. Oftest opbygger man JUnit test ud fra Java klasser. Der er mange måder hvorpå man kan bruge JUnit testen. Man kan både skrive testen inden, man kan skrive den efter, man kan lave den på baggrund af indsigt i koden eller uden nogen form til kendskab af programkoden. Det to sidst nævnte kaldes Black- og Whitebox test. Her er et eksempel på et stykke udført JUnit test fra vores spil: Det ser på Figur 3.5 at koden ikke bestod testen, hvor vi også får vist den rette, selv skrevet, fejl besked.

3.6 Positiv negativ test

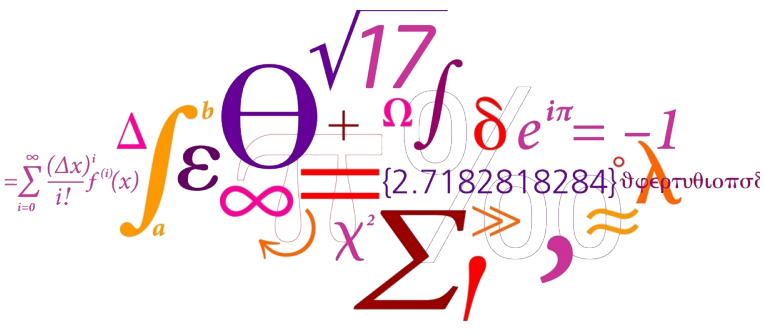
Denne test bruges for at teste om systemet kan håndterer 'ugyldige' input, dette kan både være negative som positive værdier, såvel som bogstaver og andre tegn. Denne test forbindes oftest med en eller flere ækvivalensklasse test.

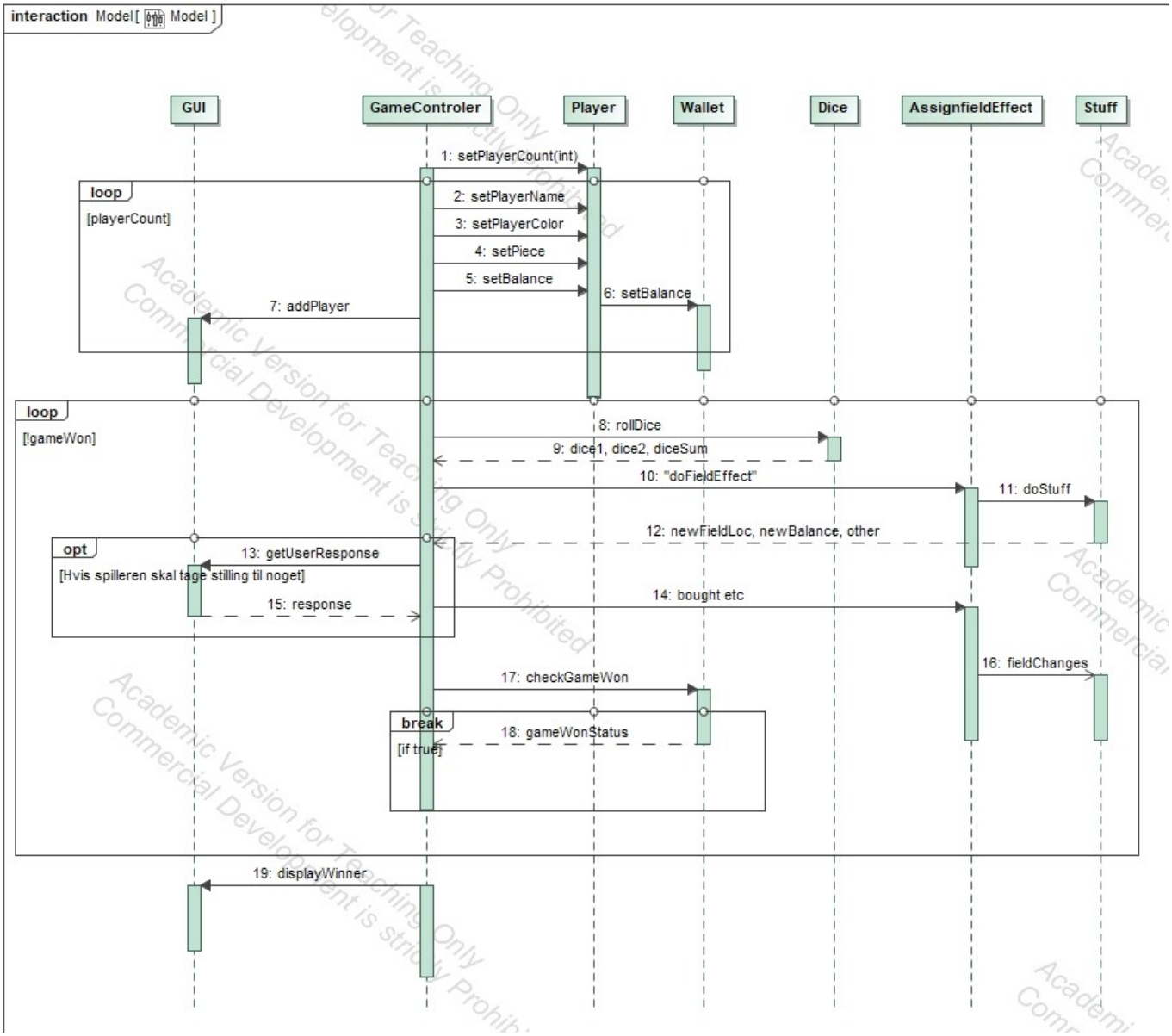
3.7 Black- og Whitebox test

Blackbox test: Her har du intet kendskab til koden, ud over hvad denne skal kunne gøre. Det er derfor –lige til højre benet– at teste det faktiske output,

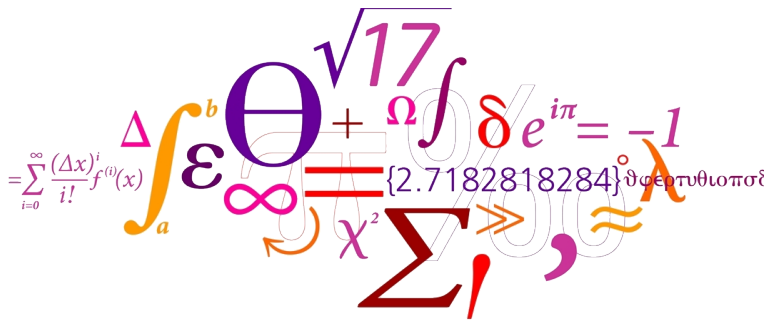
mod det forventede output og at teste ækvivalensklasserne. **Whitebox test**: Her opstiller man tests på baggrund af koden og dens opbygning. Man vil med disse tests teste hele koden, altså alle instruktioner, forgreninger og stier, som koden kan blive udført i. Dette er ikke altid muligt med *BlackBox testen* , da man her ikke har et indblik i koden.

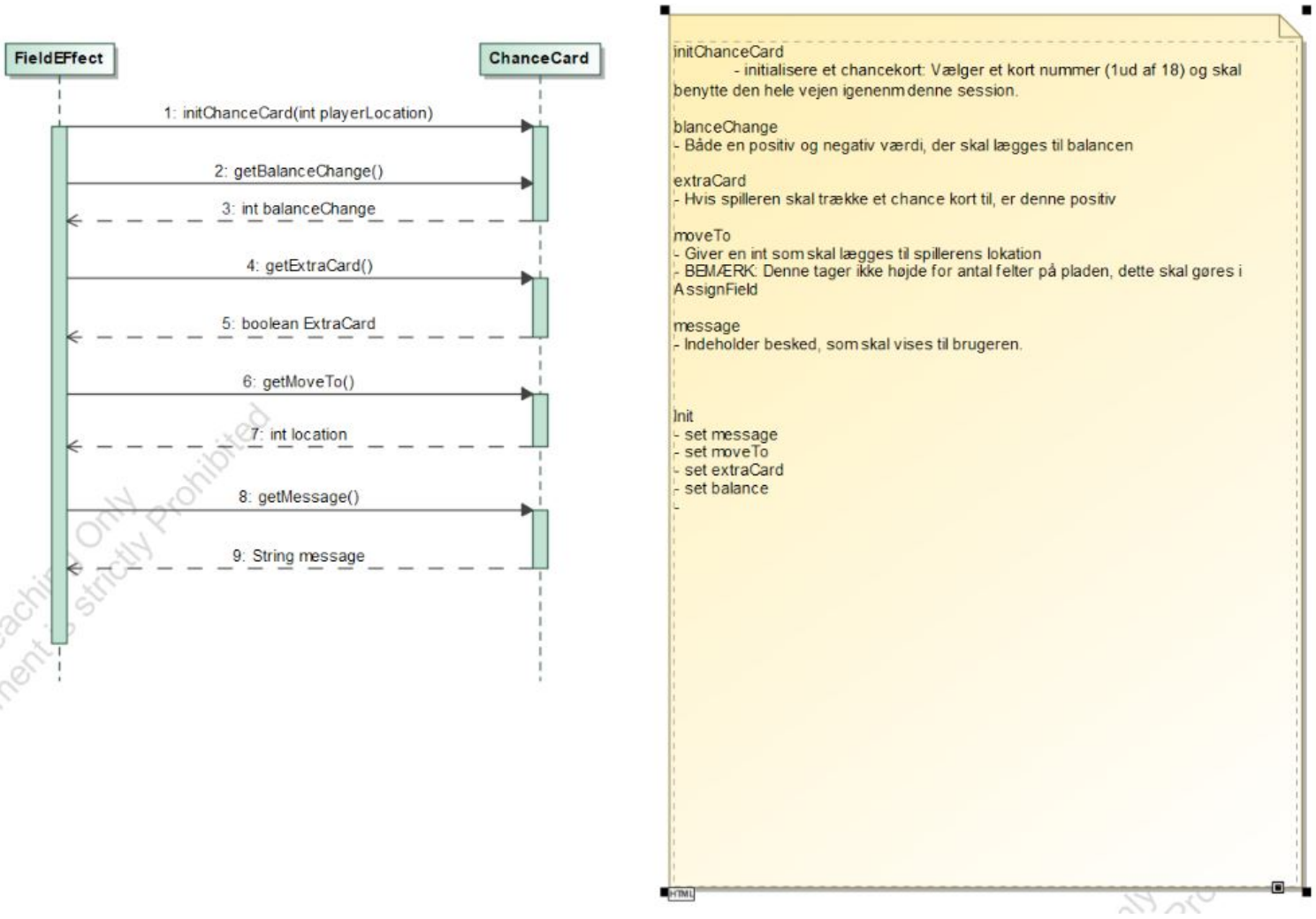
3.8 Dokumentation for overholdt GRASP



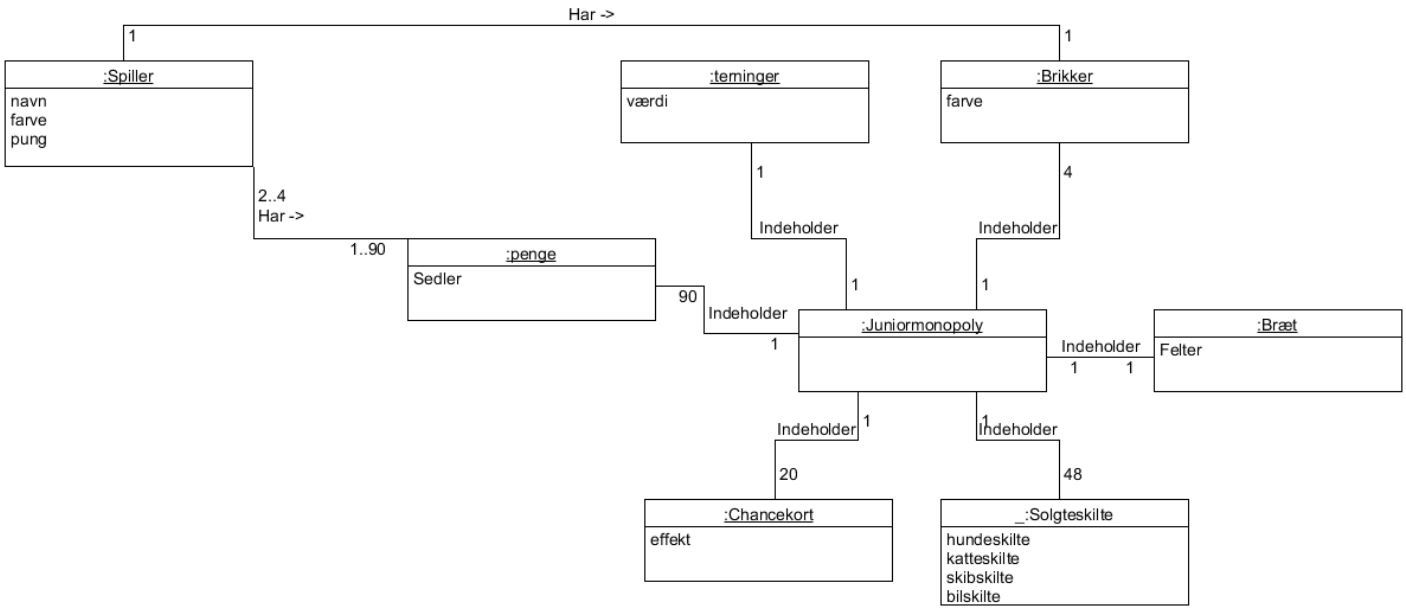


Figur 4: Sekvensdiagram tegnet i MagicDraw

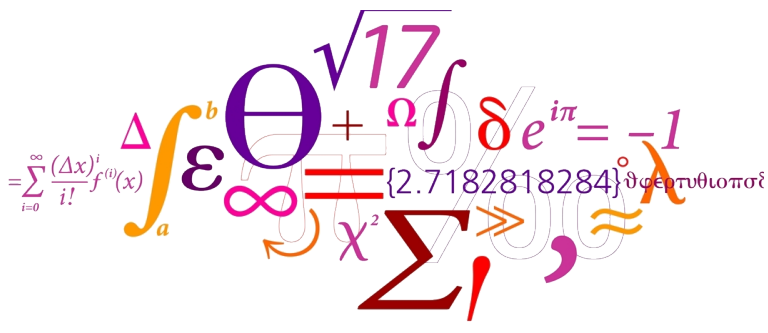


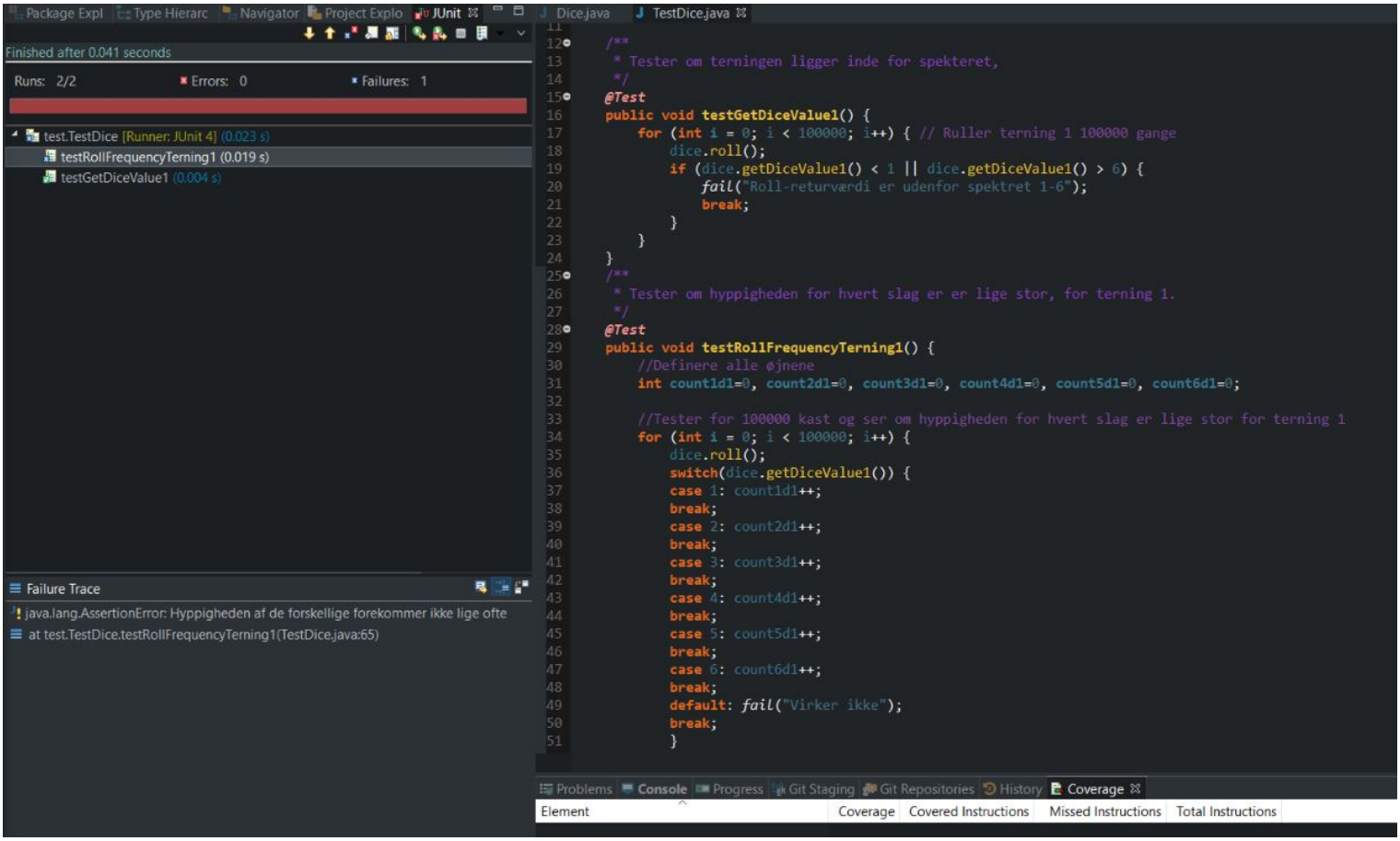


Figur 5: Systemsekvensdiagram tegnet i MagicDraw



Figur 6: Domænemodel tegnet i UMLet





Figur 7: JUnit test udført i Eclipse den 23/11-2017