CDIO delopgave 2 efterår 2016

02312 - Indledende programmering, 02313 - Udviklingsmetoder til IT-systemer, 02315 - Versionsstyring og testmetoder.

Projektnavn: CDIOdel2
Gruppenr: 42
Afleveringsfrist: Fredag d. 4/11 2016 Kl. 23:59

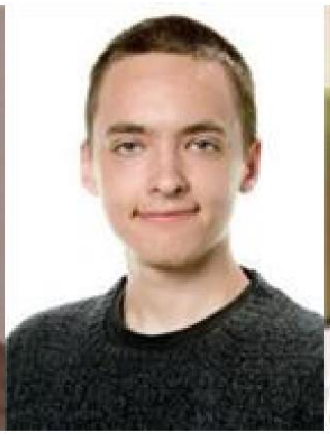
Denne rapport afleveres via Campusnet (der skrives ikke under).

Denne rapport indeholder 22 sider incl. denne side.

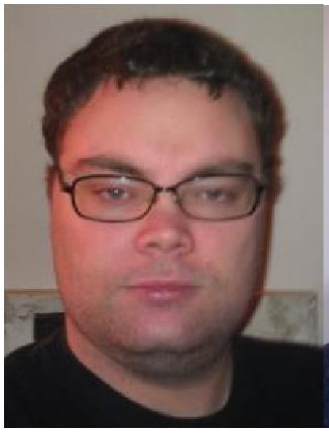
s165225; Al-Alak, Mehdi A.
s165224; Christensen, Ivan.
s165221; Jensen Lasse
s165216; Jokil, Michael
s165223; Jørgensen, Jimmy
s165206; Soelberg, Andreas.

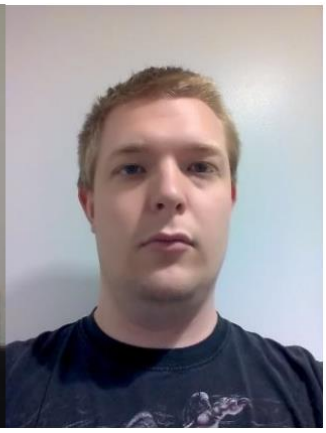

165225


165224


165221


165206


165223


165216

Timeregnskab

	total	Analyse	Design	Implementation	Test	Dokumentation
Jimmy	35.5	3	1	24.5	4	3
Lasse	28.25	4	6.75	8.5	5	4
Andreas	20.5	2.5	6	0	2	9
Mehdi	12.5	5.5	2	0	0	5
Michael	17	1.5	10.5	0	0	5
Ivan	0	0	0	0	0	0

Indholdsfortegnelse

Resume

Denne rapport omhandler en gennemgang af udarbejdelse for IOOuterActive's spil-vision. Indholdet af denne vision som vi modtog fra IOOuterActive skitserede løst rammerne for deres vision, men indholdet var for det meste op til os selv. Vi startede med at opsætte nogle krav, ud fra den vision vi havde fået. Ud fra kravene designede vi 1.version af et klassediagram og SSD-diagram. Herefter begyndte implementationen af programmet, hvor forskellige dele blev tilføjet pr iteration. Selve programmet er blev testet via JUnit og kørsel af programmet og de sidste fejl rettet.

Indledning

Det følgende program, som bliver præsenteret i denne rapport, er udarbejdet ud fra IOOuterActive vision omhandlende et spil. Vision af programmet gik ud på at skabe et spil mellem to spillere, hvor spillerne på skift kaster med 2 terninger, og herefter rykker frem til at repræsentative felt, afhængigt af terningernes værdi. Hvert enkelt felt har en effekt, som kan være positiv eller negativt ift. ens pengebeholdning. Efter nærmere specificering af kravene fra kundens side, er vi kommet frem til et resultat og et færdigt program, som vil blive præsenteret og dokumenteret i de følgende afsnit.

Hovedafsnit

Kravspecifikation

ID	Functional requirements	ID	Non-funuctional requirements
F1	The game shall be for 2 players	NF1	The game shall be playable at "DTU's data bars"
F2	The players shall roll with two dice	NF2	The game shall be able to run without any significant delays
F3	The players shall move around on a board	NF3	The system shall be written in Java
F4	The board shall feature 12 different fields*	NF4	The game shall be easy to translate (localization)
F5	The game shall feature 5 different roles***	NF5	The dice shall be easily changeable**
F6	Each field shall show a unique description		
F7	When the player lands on a field, the game shall print a certain message depending on his choice of character****		
F8	The player shall be able to roll a dice and continue from the field, on which the player landed in his/hers next turn		
F9	The player shall play the game via a GUI.		
F10	The players shall start with 1000 coins		
F11	If a player have 3000 coins or more, he wins the game and the game ends.		
F12	The game shall end, when a player wins		
F13	The game shall be en English		
F14	The fields shall have a permanent location		
F15	If the player has a balance of 0 or less, he shall lose his next turn and gain 100 coins.		
(*)	**References:**		
*	Fields: Start, Tower, Crater, Palace Gates, Cold Desert, Walled City, Monastery, Black Cave, Montain Huts, The werewall. The Pit, Goldmine.		
**	The GUI is only able to depict a die with 6 eyes.		
***	Roles: Wizard, Warrior, Dwarf, Elf, Thief.		
****	See 'Bilag' for messages		

Konfiguration

For at være i stand til at køre programmet, skal man have Java version 1.7 eller nyere og Windows 7,8 eller 10 installeret.

Guide - kompiler, installér og kør programmet

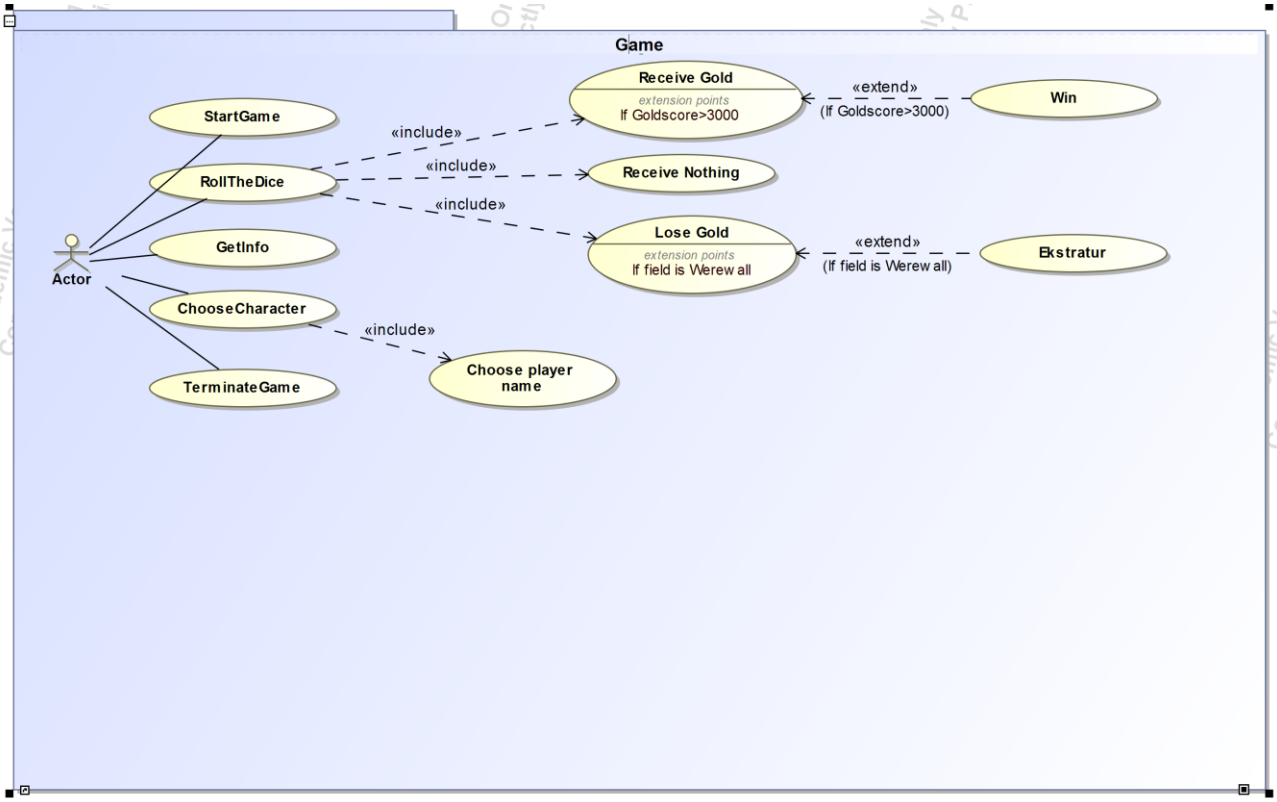
1. Open din browser
2. Gå til denne webside https://github.com/LasseJensen213/42_CDIO_02

3. Kopier klonings URl til højre (https://github.com/LasseJensen213/42_CDIO_02.git)
4. Åben ”Eclipse”
5. File -> Import
6. Tryk next
7. Git -> Projects from Git
8. Tryk next
9. Close URl
10. Tryk next
11. sæt den kopierede URl i feltet
12. Tryk next
13. Tryk next
14. Vælg din sti til at være dit workspace
15. Tryk next
16. Tryk finish
17. Åben projektet i Eclipse
18. Åben pakken kaldet ”Game”
19. Åben klassen kaldet ”Menu”
20. Tryk ”Run” (Den grønne pil i toppen)
21. Spillet åbner nu et nyt vindue.

Analyse/krav

Use-case diagram

Use-case diagrammet skal illustrere de forskellige funktioner som systemet indeholder, som er blevet designet og kodet, med tanke for kundes ønsker og krav, til det færdige system.



Use-case beskrivelser

Use Case:	**StartGame**
Id:	1
Kort beskrivelse:	Spilleren starter spillet
Primær aktør:	Spillerne
Sekundær aktør:	-
Forudsætning:	ingen
Main flow:	1. Use case starter når der vælges ”Start game” vælges i menu.
Efterfølgende:	Spillet starter

Use Case:	**ChooseCharacter**
Id:	-
Kort beskrivelse:	Spillerne vælger Karakter ud fra valgmulighederne: Elf, Wizard, Dwarf, Theif og Worrier.
Primær aktør:	Spillerne
Sekundær aktør:	-
Forudsætning:	Spillet skal være startet.
Main flow:	1. Use case starter når StartGame har kørt. 2. Systemet beder spilleren om at vælge en karakter. 3. Systemet beder spilleren om at vælge et karakternavn til den valgte karakter.
Efterfølgende:	Systemet giver nu mulighed for at kaste med terninger (Use case: RollTheDice starter)

Use Case:	**ChooseCharacter**
Id:	2
Kort beskrivelse:	Spillerne vælger Karakter ud fra valgmulighederne: Elf, Wizard, Dwarf, Theif og Worrier.
Primær aktør:	Spillerne
Sekundær aktør:	-
Forudsætning:	Spillet skal være startet.
Main flow:	1. Use case starter når StartGame har kørt. 2. Systemet beder spilleren om at vælge en karakter. 3. Systemet beder spilleren om at vælge et karakternavn til den valgte karakter.
Efterfølgende:	Systemet giver nu mulighed for at kaste med terninger (Use case: RollTheDice starter)

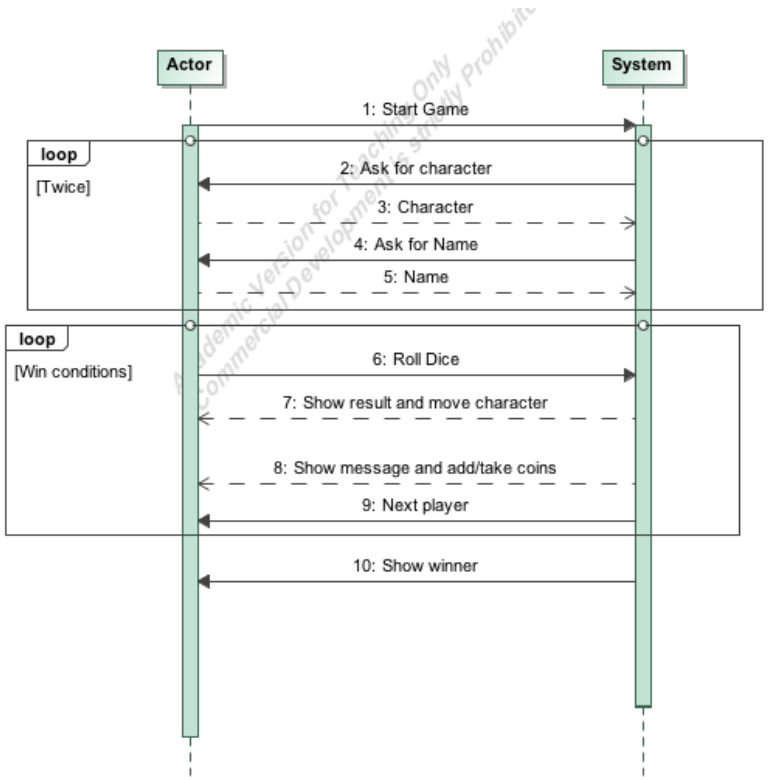
Use Case:	**RollTheDice**
Id:	3
Kort beskrivelse:	Spillerene slår med terningen og rykker det antal øjne terningerne tilsammen viser frem på pladen. Alt efter hvilket felt spilleren lander på, vil der ske et at følgende 4 scenarier: 1. Får tilført coins. 2. mister coins. 3. ingenting sker 4. lander på werewall, og mister coins, men får en ekstratur.
Primær aktør:	Spillerne
Sekundær aktør:	-
Forudsætning:	Spillet skal være startet, og spillerne skal ha valgt karakter og karakternavn.
Main flow:	1. Use case starter når ChooseCharacter har kørt. 2. Systemet beder spilleren om at slå med terningerne. 2.1. Systemet viser summen af terningernes værdi 2.2. Systemet rykker den pågældende spillers karakter frem til det felt, hvorpå distancen er lig summen af terningernes værdi. 3. Systemet udfører det pågældende felts effekt på spilleren. 3.1. Systemet udfører et af de 4 scenarier 3.2. Systemet returnerer en String
Efterfølgende:	Der skiftes tur, medmindre scenarie 4 er indtruffet

Use Case:	**GetInfo**
Id:	4
Kort beskrivelse:	Viser spilleren informationer omkring spillet
Primær aktør:	Spillerne
Sekundær aktør:	-
Forudsætning:	Systemet er startet.
Main flow:	1. Use case starter når der vælges ”Info” i menuen.
Efterfølgende:	Giver mulighed får at starte spillet eller afslutte systemet.

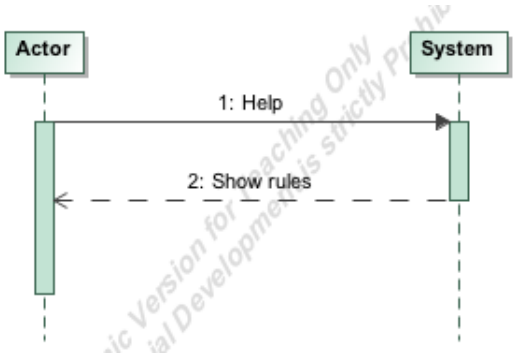
Use Case:	TerminateGame
Id:	5
Kort beskrivelse:	Afslutter systemet
Primær aktør:	Spillerne
Sekundær aktør:	-
Forudsætning:	Systemet er startet.
Main flow:	1. Use case starter når der vælges ”Exit game” i menuen.
Efterfølgende:	-

Systemsekvensdiagram

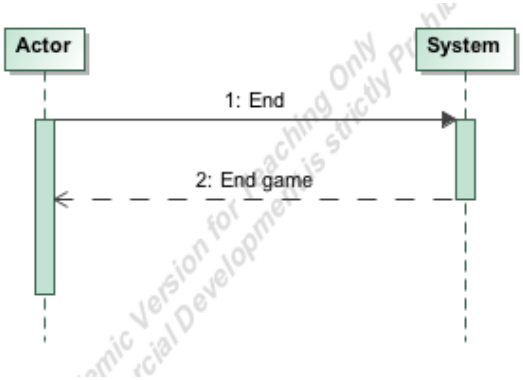
Efter at vi har lavet use cases og use case descriptions, kan vi nu lave vores system sequence diagrammer (SSD), disse diagrammer viser de forskelige interaktioner der er mulige mellem bruger og system. Vi har lavet 3 SSD modeller da spillet starter i en menu hvor man har 3 muligheder. Den første model er for den første funktion som starter spillet, her kan man se at spillet starter ved at brugeren trykker på funktionen ”Start”, derefter vil systemet spørge for hvilken karakter brugeren vil være og hvilket navn karakteren have, dette sker to gange da det er et krav at spiller skal være mellem 2 brugere, ved spillets start skal første spiller bruge funktionen RollDice for at kunne rykke sig på brættet, når denne funktion er brugt vil systemet vise terningernes resultat og automatisk rykke spilleren x antal pladser på brættet og derefter vil systemet vise en besked og tilføje eller tage coins fra en spillers konto, efter alt dette går turen til spiller 2 hvor spillet vil køre i et loop til en spiller har vundet.



Den næste SSD viser hvad funktionen ”Help” gør, her skal brugeren trykke på funktionen og derefter vil systemet vise reglerne for spillet.



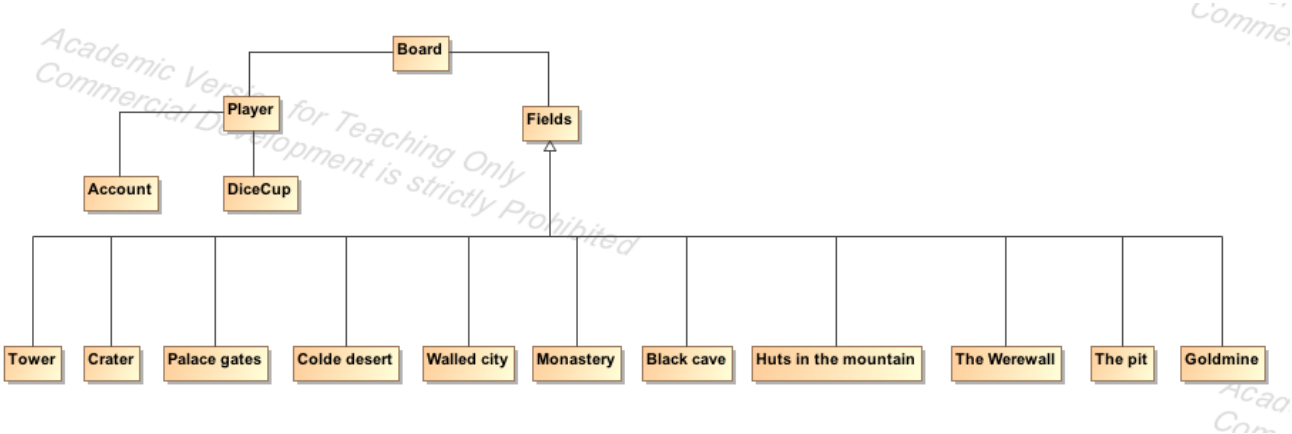
Den sidste SSD er for funktionen ”End”, hvis denne funktion bliver brugt at brugeren vil systemet slutte spillet.



Domæne-model

Efter at vi har fået alle krav og prioriteter på plads skal vi nu analysere spillet.

For at kunne lave spillet bedst muligt, skal vi først finde ud af hvilket fysiske objekter spillet skal indeholde og hvordan alle disse objekter er forbundet. Derfor har vi lavet den følgende domæne model:



Formålet med denne model er at få et overblik over de fysiske ting der kommer til at være i spillet, som en spiller kan se.

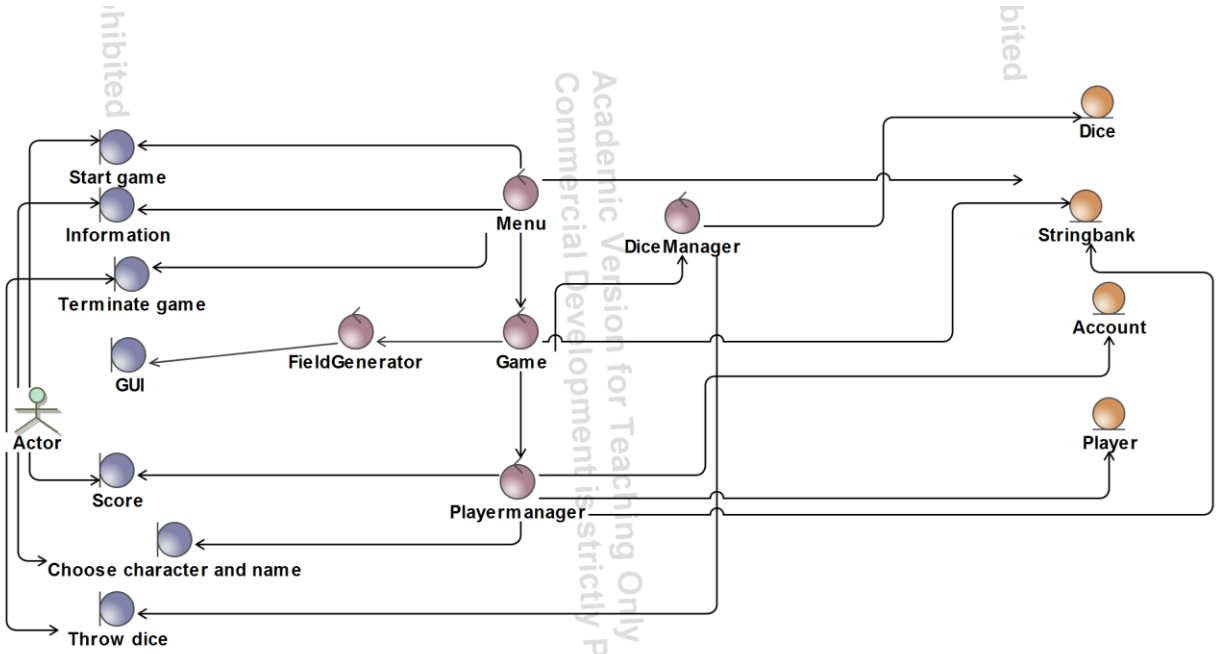
Ved at lave dette har vi fundet ud af at Board er det mest centrale ved spillet da alle andre fysiske ting høre til på Board og alle spiller interaktionen sker igennem.

Board indholdet også forskellige fields med hver deres funktion. Der er i alt 10 fields der enten tilføjer coins til en spiller eller trækker coins fra spillerens konto, og et felt hvor man ikke får eller mister coins.

Udover fields er der også player med en account som vil indholder antal coins for en spiller, og DiceCup som player kan bruge til at rulle terningerne. Med dette har vi så en oversigt over hvordan spillet skal struktureres

BCE

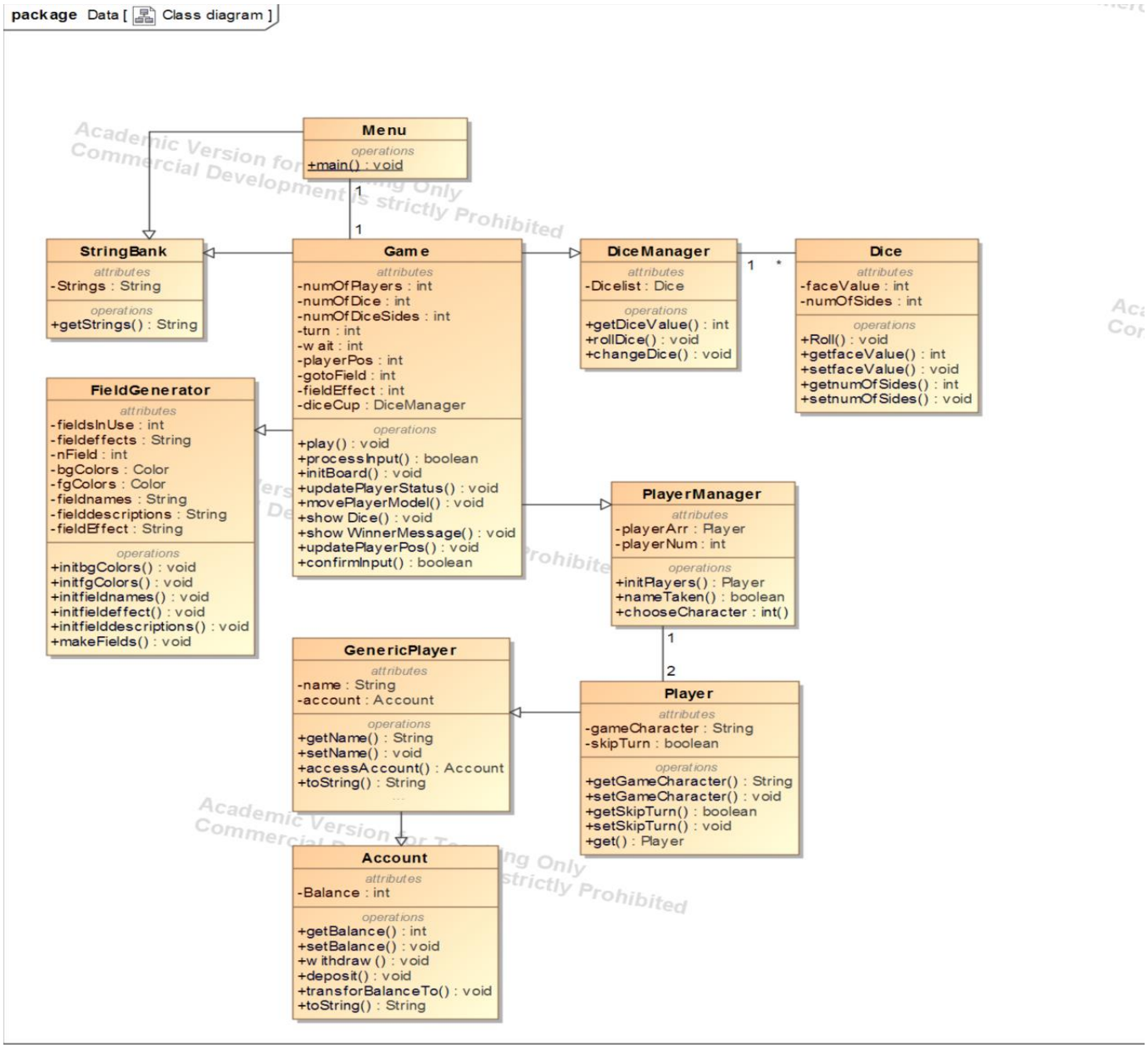
BCE-modellen illustrerer interaktionerne mellem bruger og system, samt system og klasser. I modellen har vi 7 boundary objekter, som er de funktioner brugeren kan tilgå i systemet ved brug. Derudover har vi 5 controllers som hhv. er en Menu-, Game-, FieldGenerator, Playermanager- og Dicemanagerkontrol. Hver kontroller har ansvaret for, at en pågældende forespørgsel fra er boundary objekt, bliver modsvaret med korrekthed, og det er dertil vigtigt at koblingen i systemet er korrekt, så at en kontroller kan hente de rigtige oplysninger i de individuelle klasser, og sende dem retur til boundary objektet og brugeren (evt GUI'en/TUI'en).



Design

Design-klasse-diagram

I vores design-klasse-diagram kan man se sammenhængen mellem alle de forskellige klasser i vores program, og deres attributer og operationer. Øverst har vi vores "Menu" klasse hvori vores main metode befinder sig. Denne kommunikerer direkte med vores "Game" klasse, som er selve spillet, og vores stringbank, hvor vores strings ligger.
Vores "Game" klasse håndterer så oprettelsen af "DiceManager", "PlayerManager" og "FieldGenerator".
"DiceManager" svarer til et raflebæger, som så opretter 2 "Dice" objekter.
"PlayerManager" håndterer så "Player", som indeholder hvilken spillerkarakter man har valgt og "GenericPlayer", som indeholder spillernavn og et accountobjekt.
"FieldGenerator" er den klasse som opretter selve spillebrættet



Design Sekvens Diagramme

Dette diagram viser interaktionen mellem bruger og programmet og dets klasser.
Det første der sker når brugeren åbner vores program er at programmet spørger om det skal starte et spil, afslutte et spil eller vise hjælpeinformation.
I dette tilfælde har vi valgt at vise hvad der sker under kørsel af programmet når brugeren vil spille et spil.
Når et spil startes oprettes et ” generic player” objekt, som er en general spiller klasse der indeholder et spillernavn og en ”account” der er lig nul. ”Generic player” opretter så et ”account” objekt og et ”player” objekt, der er specifikt til dette spil.
”Player” objektet sætter så account objektets værdi til startværdien for spillet.
Så beder programmet brugeren om at indtaste et spillernavn og vælge en ”game character”.
Efter det er gjort vil programmet rulle med terninger, hvilket resulterer i oprettelsen af en ”dicemanager”, som har samme funktion som et raflebæger, som opretter ”dice” objekter, som er selve vores terning.
Herefter går vi så ind i et loop, som svarer til en tur, der fortsætter indtil en vinder er fundet.
Først ruller vores dicemanager med to terninger, hvorefter resultatet bliver sendt til vores stringbank via vores main metode.
Vores stringbank returnerer så den string der passer til resultatet af terningrullet til main, og main ændrer so vores ”account” balance afhængigt af hvilken string der blev valgt. Derefter returnerer main stringværdien til brugeren.

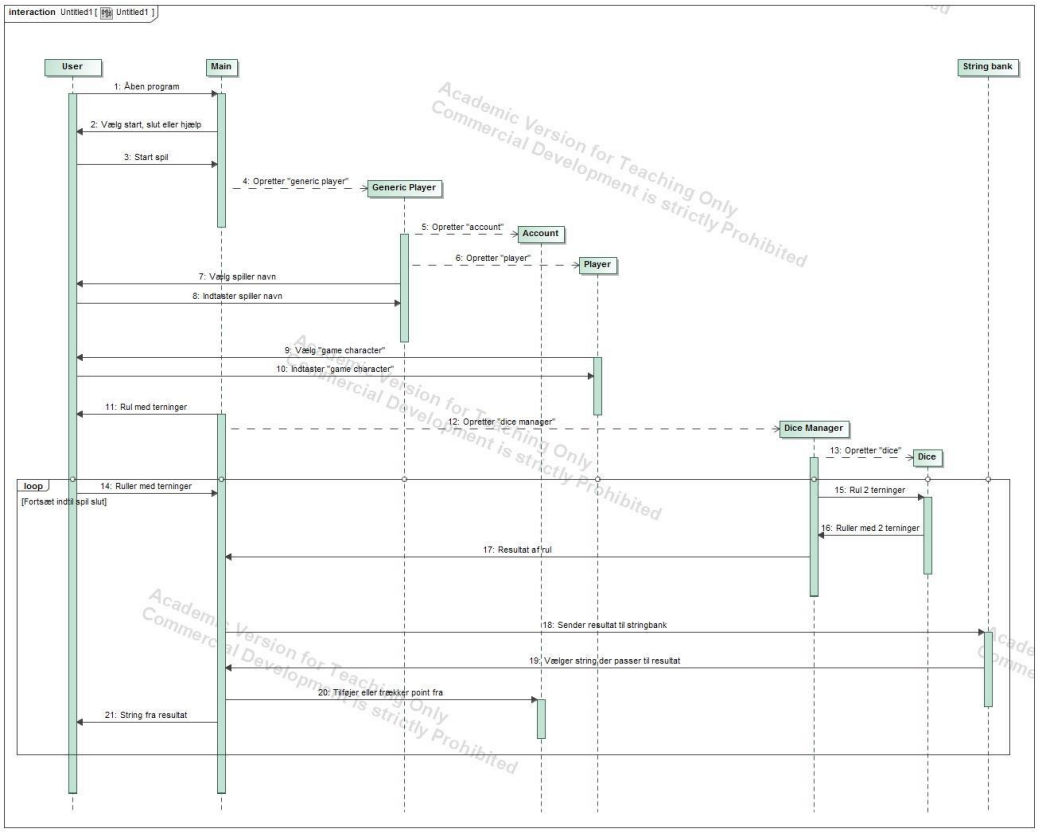


Diagram 1: Diagrammet afspejler ikke 100% det færdige program, grundet ændringer undervejs i processen.

Dokumentation

Implementering

Grasp

For at overholde GRASP(General Responsibility Assignment Software Patterns) lavede vi klasserne DiceManager og PlayerManager, som var både: information expert, og creator, og for at yderligere overholde princippet om low coupling fik spiller klassen ansvaret om at skabe et account objekt. Dette gøres for at nedsætte konsekvenserne, hvis man nu ændrer noget i en klasse, er der mindre chance for at hele programmet stopper med at virke. Klassen Game fungere som en controller, altså indeholder den selve spil logikken og koordinerer brugen af metoder. Vi lavede også til sidst en FieldGenerator klasse, der havde til ansvar at skabe arrays af felter. Før var det Game klassen der gjorde dette, men det understøttede ikke princippet om High-cohesion.

StringBanks

For at overholde kravet om at spillet skulle være nemt at oversætte for andre udviklere, samlede vi alle strings i en stringbank. Den indeholder kun strings, strings-arrays, og metoder til at få fat i dem. Vi startede med en string bank, men besluttede senere at brække dem op i flere stykker, så hver klasse, der bruger strings havde deres egen string bank.

Menu

Menu klassen er den første man kommer ind i. Her bliver GUI'en først initialiseret og man får 3 valgmuligheder("Start Game" / "Info" / "Exit game") Den bruger en række If/else sætninger i stedet for en switch case. Vi kunne ikke få switch casen til at virke da argumenterne i de forskellige cases skulle være konstante, altså de kunne ikke komme fra andre klasser. Selve menuen ligger i en while loop, hvori et Game objekt ligger, og vælger man at starte et spil, vil man returnere til dette while loop ved spillets afslutning.

Game

Selve spillet foregår i metoden play(), her bliver objekter af PlayerManager, og DiceManager oprettet. og Spillernes figurer bliver placeret på spil-brættet. Desværre havde vi ikke tid til at ændre på figurerne, så det passede mere med vores karaktere. I stedet får de en default bil-model. Når brættet er initialiseret, vil resten af spillet foregå i en while loop, der slutter når en vinder er fundet. i updatePlayerPos(), der trinvis rykker spillerens figur et felt frem, og springer start feltet over. Vi har sat en TimeUnit.MILLISECONDS.sleep() ind, så man kan se figuren rykke sig frem.

PlayerManager

PlayerManager har til ansvar at holde styr på spillerne i et Player-array, og at oprette spillere når den bliver instantieret. Den har metoder som nameTaken(), der kan se om et navn er taget, så man ikke begge to kan have det samme navn. i chooseCharacter() vælger man karakter, og alt efter hvilken karakter man vælger, vil man få forskellige beskeder, når man lander på et felt. De er for det meste til pynt og har ingen egentlig virkning på spillets forløb.

Player

Vi har to Player klasser: en superklasse GenericPlayer og en mere specifik subklasse Player, som indeholder attributter, der er relevante til vores spil. GenericPlayer indeholder et navn og er tilknyttet en Account klasse, som den selv instantiere. Player klassen har så yderligere en karakter fra spillet, og en boolean skipTurn, der viser om spilleren skal springe sin næste tur over. Først havde vi denne variabel i Game klassen, men det vidste sig at være for besværligt at implementere sådan at det virkede ordenligt. Grunden til at vi lavede en superklasse og subklasse var fordi kunden havde et krav om at de ville have en Player klasse og Account klasse, som de kunne bruge i andre projekter.

Test

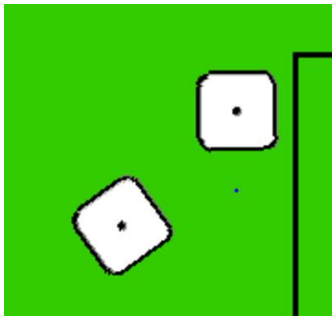
Test af krav specifikationer

Test ID: 1

Scenario	Input	Faktisk Output	Forventet Output	Ok?
F1	”start game” ”choose character” ”choose name”	Beder om karakter og spillernavne for 2 spillere.	Systemet beder 2 spiller om karakter og navn, da spillet er i mellem 2 spillere.	ja

Test ID: 2

Scenario	Input	Faktisk Output	Forventet Output	Ok?
F2	”start game” ”choose character” ”choose name” ”roll”	Systemet benytter 2 terninger.	Systemet benytter to terninger.	ja

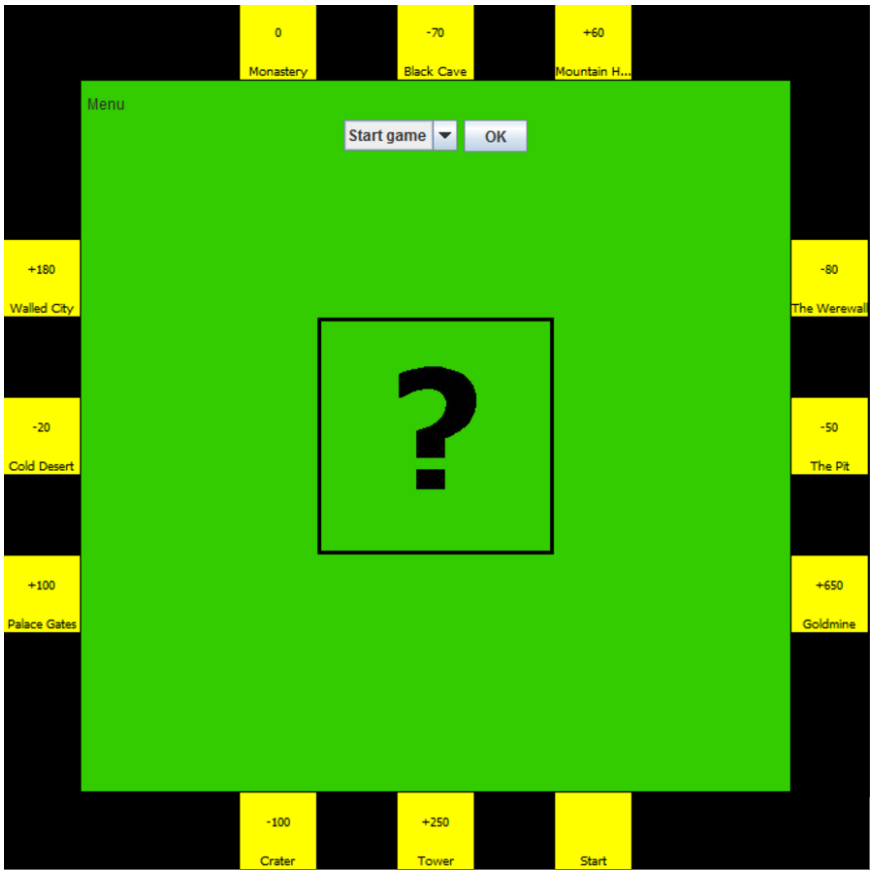


Test ID: 3

Scenario	Input	Faktisk Output	Forventet Output	Ok?
F3	”start game” ”choose character” ”choose name” ”roll” ”roll”	Spillerne rykker for felt til felt	Spillerne bevæger sig fra et felt til et andet, alt efter terningernes samlede værdi.	ja

Test ID: 4

Scenario	Input	Faktisk Output	Forventet Output	Ok?
F4		Systemet åbner en GUI med 12 felter	Systemet skal indeholde 12 felter.	ja



Test ID: 5

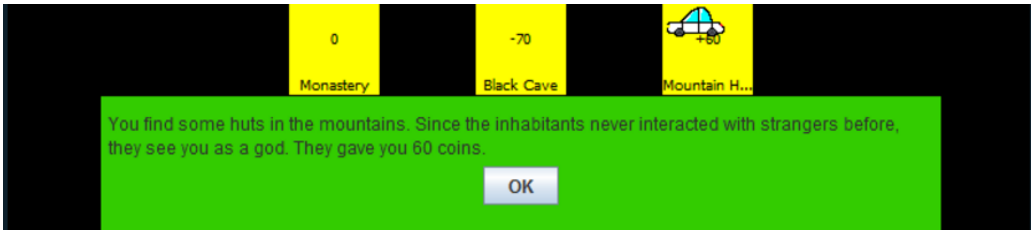
Scenario	Input	Faktisk Output	Forventet Output	Ok?
F5	”start game” ”choose character”		Man skal kunne vælge i mellem 5 forskellige karakterer	ja

Test ID: 6

Scenario	Input	Faktisk Output	Forventet Output	Ok?
F6	Hold musen over felterne og observér.	Hvert felt har sin egen unikke beskrivelse.	Hvert felt har en unik beskrivelse.	ja

Test ID: 7

Scenario	Input	Faktisk Output	Forventet Output	Ok?
F7	”start game” ”choose character” ”choose name” ”roll”	Systemet printer en besked som afhænger af vælg af karakter og hvilket felt, spilleren lander på.	Systemet skal printe en besked til brugen når han lander på et felt, som også er afhængig af felt og karakter.	ja



Test ID: 8

Scenario	Input	Faktisk Output	Forventet Output	Ok?
F8	”start game” ”choose character” ”choose name” ”roll” ”roll” ”roll”	Spilleren fortsætter med et rykke fremad fra det felt spilleren stod på inden kastet.	Spilleren rykker fremad fra det felt spilleren stod på da spilleren kastede med terningerne.	ja

Test ID: 9

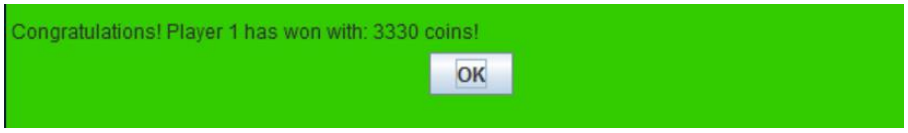
Scenario	Input	Faktisk Output	Forventet Output	Ok?
F9	observer	Spiller foregår i en GUI.	Systemet er visuelt repræsenteret i en GUI.	ja

Test ID: 10

Scenario	Input	Faktisk Output	Forventet Output	Ok?
F10	”start game” ”choose character” ”choose name”	Hver spiller starter med 1000 coins	Spillerne starter med 1000 coins.	ja

Test ID: 11

Scenario	Input	Faktisk Output	Forventet Output	Ok?
F11	”start game” ”choose character” ”choose name” ”roll” (”roll” until someone reach 3000coins)	Spilleren vinder når han opnår 3000 coins.	Spillet skal slutte når en spiller opnår 3000 coins eller over.	ja



Test ID: 12

Scenario	Input	Faktisk Output	Forventet Output	Ok?
F12	”start game” ”choose character” ”choose name” ”roll” (”roll” until someone reach 3000coins) “ok”	Systemet aflsuttes, når en spiller vinder, og det godkendes at spillets lukkes	Spillet slutter når en spiller vinder.	ja

Test ID: 13

Scenario	Input	Faktisk Output	Forventet Output	Ok?
F13	observér	Alle beskeder i brugerinterfacet er på engelsk	Spillet skal være på engelsk.	ja

Test ID: 14

Scenario	Input	Faktisk Output	Forventet Output	Ok?
F14	observér	Felter har en fast location i GUI'en	Felternes placering skal være statisk.	ja

Test ID: 15

Scenario	Input	Faktisk Output	Forventet Output	Ok?
F15	”start game” ”choose character” ”choose name” ”roll” (”roll” until someone reach 0 coins) “ok”	Spilleren får 100 coins, men mister sin næste tur (modsatte spiller slår 2 gange i træk)	En spiller som kommer under en coinværdi på 0, får 100 coins, men mister sin næste tur.	ja

Test/Krav	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15
1	X				X										
2	X	X			X										
3	X	X	X	X											
4				X											
5					X										
6						X									
7	X	X	X	X		X	X								
8	X	X	X	X		X	X	X							
9									X						
10	X									X					
11	X	X	X	X	X	X	X	X	X	X	X				
12	X	X	X	X	X	X	X	X	X	X	X	X			
13													X		
14														X	
15	X	X	X	X	X	X	X	X	X	X					X

Konklusion

I denne rapport har vi fremstillet et spil, ud fra den vision IOOuterActive har givet os. Vi har endvidere opsat vores egne krav, som har stået til bund for udviklingen af spillet. Spillet er et interaktivt virtuelt brætspil mellem to spillere, hvor det gælder om først at nå 3000 point. Vi kan ud fra de tests vi har lavet konkludere at spillet fungerer, som tiltænkt af kunden.