CDIO delopgave 2

'A Game Of Gold'

Udviklingsmetoder til IT-systemer 02313, Softwareteknologi

Gruppe Nr.: 31

Afleveringsfrist: Fredag 04/11-2016 23:59

Institut: DTU Compute

Vejleder: Ronnie Dalsgaard

Denne rapport er afleveret via Campusnet (der skrives ikke under)

Denne rapport indeholder 29 sider eks. denne sider.


Jia Hao Johnny Chen, s165543


Christopher David Carlson Chytræus, s165230


Bertram Christian Henning, s153538


Jonathan Yngve Friis, s165213


Thomas Kristian Lorentzen, s154424

Timeregnskab

CDIO delopgave 02							
Timeregnskab	Vers. 04/11/2016						
Dato:	**Deltager**	**Design**	**Impl.**	**Test**	**Dok.**	**Andet**	**I Alt**
04/11/2016	Christopher	**6**	**2**	**1**	**2**		
04/11/2016	Jonathan	**7**	**1**	**2**	**4**		
04/11/2016	Thomas	**5**	**2**	**1**	**3**		
04/11/2016	Johnny	**7**	**1**	**0**	**4**		
04/11/2016	Bertram	**4**	**3**	**3**	**1**	**3**	
	Sum	**29**	**9**	**7**	**14**	**3**	**62**

Abstract

The aim of this project has been to create a board game using dice, based on specifications brought by the customer and IOOuteractive. For this project, we have been using a Graphical User Interface (GUI) for interacting with the program. The system has been designed to be able to run on all the machines and databars on DTU. By using different diagrams for the interaction of the different classes in our source code, we have been able to make our development process much easier, with the diagrams giving a clearer image of how all parts connect. As a way of assuring ourselves of the functionality of our class methods, we've been running JUnit tests on them, which has been successful. In this report, our focus will be centered on explanations and depictions of our design models, explanations of different theoretical terms as well as a documentation and explanation of our test cases and JUnit tests. Overall, we can conclude that our project has been successful and that the end product is very functional.

Indholdsfortegnelse

Indledning

Emne

Videreudvikling af vores terningspil med formål at tilpasse det til et brætspil, hvor der bla. vises brug af GRASP, relationer og klasser til at løse opgaven.

Formål

Formålet med dette projekt er at udvikle et brætspil med 11 felter i JAVA som specificeres ud fra kravspecifikationerne som vi har udarbejdet i overensstemmelse med kunden. Rapporten indeholder en analyserende del bestående af diagrammer og modeller skabt ud fra standard UML samt brug af GRASP, som hjælper med at bryde problemløsningen fra krav til design og kode ned i flere overskuelige trin.

Kravspecifikation

Funktionelle krav:

1. Systemet skal kunne bruges på maskinerne i databarerne på DTU.
2. Systemet skal være et spil mellem to brugere på samme computer.
3. Systemet skal kunne simulere et slag med to 6-sidede terninger.
4. Systemet skal flytte en brugers brik til et af felterne nummereret fra 2-12 alt efter hvilken sum terningeslaget har.
5. Systemet skal kunne holde styr på en brugers pengebeholdning.
6. System skal sørge for at brugerens pengebeholdning ikke kommer under 0.
7. Systemet skal sørge for at ændre brugerens pengebeholdning alt efter hvilket felt han/hun lander på.
8. Systemet skal udskrive en tekst, der omhandler det aktuelle felt.
9. Systemet skal sørge for, at begge brugere starter med 1000 Gold coins.
10. Systemet skal afslutte spillet, når en bruger har opnået 3000 Gold coins og derefter kåre vedkommende som vinder af spillet.
11. Systemet skal have felter med unikke egenskaber og navne. Se bilag 1.

Non-funktionelle krav:

1. Systemet skal udvikles i Eclipse.
2. Systemet skal bruge UTF-8 som tegnsæt.
3. Systemet skal kodes i Java.
4. Systemet skal kunne vise resultat af kastet inden for 0.33 sekunder.

Konfiguration:

Programmet kræver en computer med java 1.7 eller nyere installeret og eclipse installeret.

Guide til importering af git repository:

1. Start browser
2. Gå ind på siden: https://github.com/BertramHenning/31_del2
3. Klik på “clone or download” og kopier derefter linket
4. Åben eclipse
5. Klik på “file” og vælg “import”
6. Vælg “Git-> Projects from Git”
7. Tryk “next”
8. Klik på “clone URI”
9. Klik på “next”
10. Klik på “next” igen
11. Vælg sti at gemme det på
12. Klik på “next”
13. Vælg “import existing eclipse projects” og tryk “next”
14. Sørg for at project mappen har et flyeben ud for sig eller klik i den tomme kasse
15. Tryk “finish”
16. Åben projektet i eclipse
17. Åben “src”
18. Åben pakken “aGameOfGold”
19. Åben aGameOfGold.java
20. Klik på den grønne knap med den hvide pil i.
21. Tillykke du har nu lært at starte programmet.

Navneords Analyse

- System
- Maskine
- Databar
- Spil
- Bruger
- Computer
- Slag
- Terninger
- Brik
- Felterne
- Numre
- Sum
- Pengebeholdning
- Tekst
- Gold coins
- Vinder
- Egenskab
- Navn
- Tegnsæt
- Java
- Resultat
- Sekund

Udsagnsord Analyse

- Kunne
- Bruge
- Være
- Simulere
- Flytte
- Holde
- Sørge
- Ændre

- Lande
- Udskrive
- Omhandle
- Starte
- Afslutte
- Opnå
- Kåre
- Udvikle
- Kode
- Viser

Begrebsdefinitioner[1]

BCE:

Boundary Control Entity, også tit refereret til som Entity-Control-Boundary.

GUI:

Står for Graphical User Interface, som er det der visualisere vores brætspil med tekst og grafik.

CRC:

Forkortelse for Class-Responsibility Card.

Et diagram som viser hvilke metoder som tildeles de forskellige klasser, samt hvilke andre klasser de arbejder sammen med.

[1] BCE og CRC afsnittet er taget fra CDIO1

GRASP:

Forkortelse for General responsibility assignment software patterns (or principles). GRASP er en retningslinje for tildele ansvar til klasser objekter i OOD som hjælper med at løse softwareproblemer.

Hypoteser

Vha. Eclipse og Github kan vi udvikle et brætspil som stemmer overens med kravspecifikationerne.

Teorivalg

Grasp:

Grasp står for: General responsibility assignment software patterns, og er et hjælpemiddel til objektorienteret design (OOD) og til tildeling af ansvar. GRASP består af en række mønstre, som hjælper med at tildele en eller flere klasse funktioner: controller, creator, information expert, high cohesion, low coupling, pure fabrication, indirection, protected variations og polymorphism. I vores opgave fokuserer vi dog kun på de fem første, altså controller, creator, information expert, high cohesion, low coupling.

Creator:

Noget af det første man kigger på når man skal lave et design, er hvilken klasse der står for at lave et eller flere objekt. F.eks. i dette spil instantierer klassen AGameofGold to Player objekter. Man kan nærmest se hvilken klasse der passer bedst til at instantiere et objekt, bare ved at kigge på det. F. eks så lyder det mere rigtigt at det er selve spil klassen AGameOfGold, hvor Player objektet bliver brugt, som instantierer objektet, i forhold til hvis det var klassen FieldDescriptions, der gjorde det. Ud over at det lyder forkert at bruge en anden klasse end AGameOfGold, kan man også spørge sig selv om følgende:

- Indeholder eller aggregerer klasse B, klasse A?
- Bruger klasse B meget fra klasse A?
- Har klasse B data som bruges til at instantiere A?

Jo flere af disse spørgsmål man kan nikke ja til jo bedre passer en klasse til at instantiere et objekt fra en anden klasse.

Information expert:

Hvis et objekt har lyst til at referere til et andet objekt, ved navn, så er det nødvendigt at vide hvor det første objekt kan få information om det objekt det første objekt gerne vil referere til. Til dette bruger vi information expert mønsteret. Med information expert mønsteret kigger man på hvilken klasse som er ansvarlig for information for et objekt, da man måske vil referere til et objekt ved navn. I vores opgave er FieldDescriptions informations expert, da denne klasse indeholder informationer omkring de forskellige felter. Ligesom med creator så kunne alle vores klasse have været informations expert, men det giver mere mening at have FieldDescriptions, som informations expert, da det er en klasse, som er lavet specifikt for at have information om felterne i spillet.

Low coupling:

Low coupling er en måde at sørge for at klasserne i et projekt, er så lidt afhængige af hinanden som muligt. Dette gør man fordi jo mindre afhængige klasserne er af hinanden, jo færre ændringer forekommer der i de andre klasser, hvis man ændrer en klasse. Dette gør koden nemmere at overskue, lettere at vedligeholde og lettere at ændre på. I vores spil har vi sørget for low coupling ved at have så få klasser afhængige af hinanden som muligt. F. eks kunne Player aggregere FieldDescriptions og så kunne AGameOfGold få adgang til informationen i FieldDescriptions igennem Player, men det giver mere mening at den bare henter informationen direkte fra FieldDescriptions.

Controller:

Controlleren er den første klasse som forbinder bruger interfacen (GUI) med selve spillet. Altså er controlleren den klasse, hvor selve spillet sker og bruger interfacen får information fra. I vores spil må controlleren derfor være AGameOfGold, da denne klasse er den første (og eneste), der sender information til bruger interfacen.

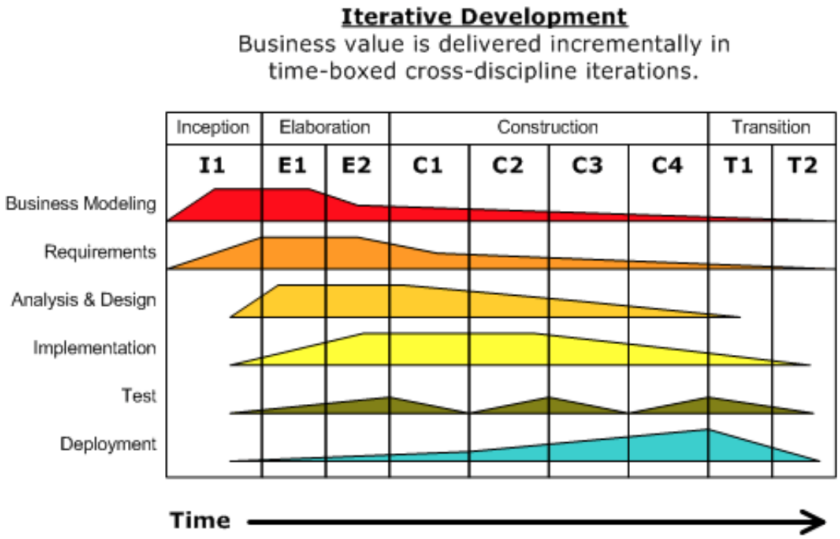
High cohesion:

I et software projekt har man udtrykket cohesion, som fortæller noget om i hvilken grad de forskellige dele af projektet hører sammen. I et high cohesion projekt er funktionalitet mellem elementerne stor og det er det man helst vil have, da dette både støtter low coupling og giver robusthed i programmet. En måde at opnå det på er at sikre sig at klasserne er “fokuserede”. At

klasserne er “fokuserede” betyder at de ikke har mange komplicerede metoder, men i stedet en, som de “fokuserer” på. Altså gælder det om at få fordelt metoderne til andre klasser. Dette hænger godt sammen med low coupling, da en eller flere af klasserne ikke er så afhængige af hinanden hvis metoderne er fordelt ud imellem dem. I vores opgave har klasserne kun de nødvendige metoder, og dermed har vi high cohesion i vores opgave.

UP

Unified process(hf. UP) er en iterativ udviklingsmetode, der har hjulpet os med at dele den længere udviklingsproces op i forskellige faser. Navnligt de fire hovedfaser; Inception, Elaboration, Construction og Transition. Inception-delen består normalt af idéfasen og skabelsen af kravspecifikationerne, men da vi ikke selv er kommet på idéen til programmet må kravspecifikationerne stå alene under den første fase. Det er dog væsentligt at bemærke, at de fire faser ikke fungerer som en vandfaldsmodel, hvor man slavisk arbejder sig igennem de forskellige faser, punkt for punkt. Derimod vil man indenfor UP benytte muligheden for at gå tilbage i faserne og lade dem ‘flyde’ sammen og derved skabe en mere dynamisk arbejdsproces. Derfor vil uddybningen af vores kravspecifikationer ligge under Elaboration. Analyserende modeller og diagrammer går hovedsageligt under punktet Elaboration da det er i denne fase vi i størst grad udvikler de centrale funktioner til vores software. I løbet af fasen Construction koder vi selve programmet. Det er her vi gør brug af vores analyserende arbejde og realiserer teorien bag spillet. Den sidste fase, Transition består af overgangen fra udvikler til drift. [2]



[2] Afsnittet om UP er taget fra CDIO 1

Metode

For at udvikle vores brætspil har vi startet ud med at lave forskellige diagrammer og use cases for at få en bedre forståelse og overblik over udviklingen af spillet samt hvordan diverse objekter, klasser og metoder skal fungere sammen. Derefter har vi kodet spillet ud fra vores usecases og diagrammer.

Hovedafsnit

Use Case og Use Case Diagram

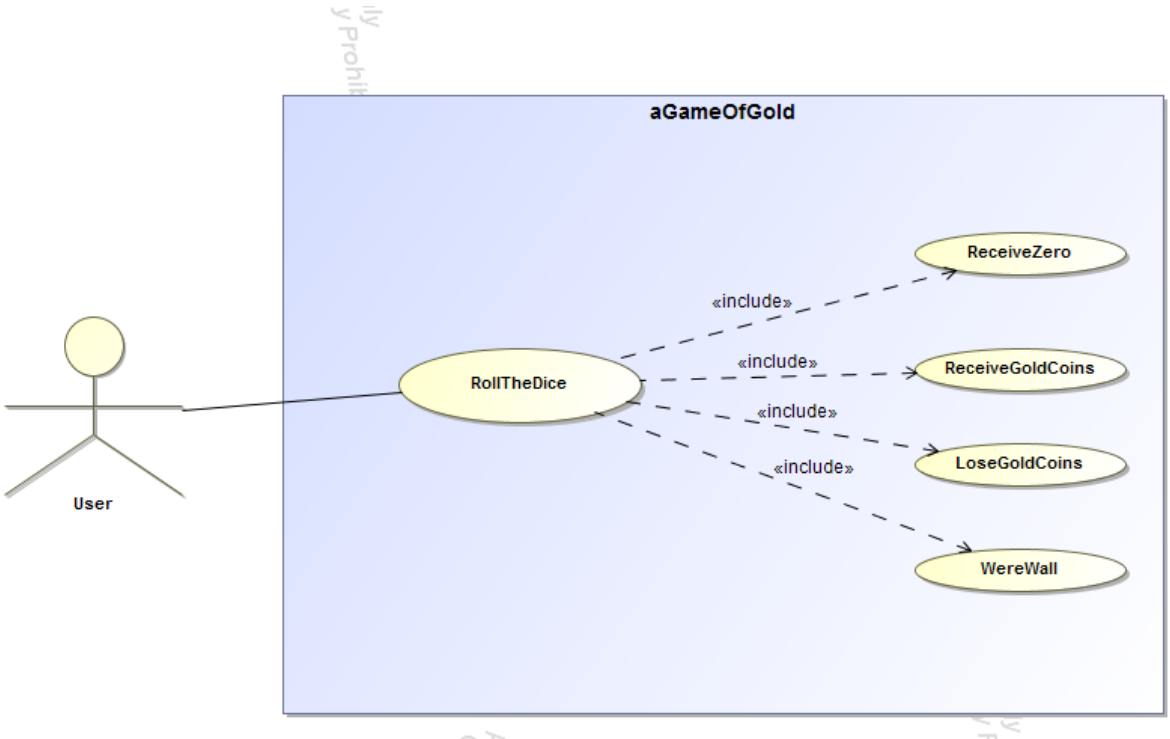
Diagrammet nedenfor viser hvordan aktøren bruger systemet. Aktøren beder systemet om at rulle terningerne. Vi kan også se de forskellige use case scenarios, som er inkluderet i RollTheDice:

-RecieveZero: Når man lander på feltet hvor der ikke sker noget (monastery)

-RecieveGoldCoins: Når man lander på et felt der giver goldcoins

-LoseGoldCoins: Når man lander på et felt hvor man mister goldcoins

-Werewall: Feltet hvor man udover får goldcoins også får en tur til



Use Case: RollTheDice
ID: 1
Brief description: A Player rolls two dice by pressing a button, then receives or loses gold coins according to the result.
Primary actors: - Player 1
Secondary actors: None
Preconditions: - Two Players - A computer
Main flow: 1. Player starts the game. 2. Player presses enter to roll dice. 3. The system rolls the dice. 4. The system sums the value of the dice 5. The system moves the Player to accordingly correct space. 6. The system adds or subtracts gold coins according to the space the Player landed on.
Postconditions: - The system successfully moved the played based on the dice roll and the according space - The system successfully added or subtracted to Players total gold coins.

Use Case Scenarios

Use Case Scenario: ReceiveGoldCoins
ID: 1.1
Brief description: The Player lands on a space where he/she receive gold coins.
Primary actor(s): The Player
Secondary actor(s): None
Preconditions: - The Player rolls the dice and gets a value moving him to a space where he/she receives gold coin.
Main flow: 1. Player presses enter to roll dice. 2. The system rolls the dice. 3. The system sums the value of the dice 4. The system moves the Player to the accordingly correct space that gives gold coins 5. The system adds gold coins according to the space the Player landed on.
Postconditions: The Player has rolled the dice and rolled a number that moves him from his previous space to a space that grants gold coins.

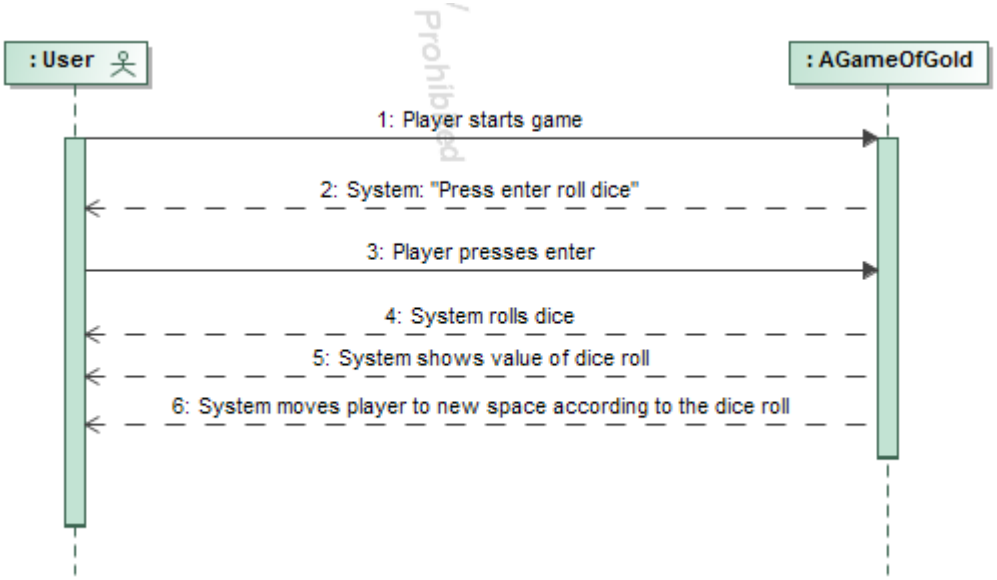
Use Case Scenario: LoseGoldCoins
ID: 1.2
Brief description: The Player lands on a space where he/she loses gold coin
Primary actor(s): The Player
Preconditions: The Player rolls the dice and gets a value moving him to a space where he loses gold coin.
Main flow: 1. Player presses enter to roll dice. 2. The system rolls the dice. 3. The system sums the value of the dice 4. The system moves the Player to the accordingly correct space that takes gold coins 5. The system takes gold coins according to the space the Player landed on.
Postconditions: The system successfully moves the player and subtracts the right amount of money from the Player's gold coin total.

Use Case Scenario: Monastery
ID: 1.3
Brief descriptions: This space does not add or subtract gold coins from a Player.
Primary actor(s): The Player
Preconditions: The Player rolls the dice and lands on the monastery space
Main flow: 1. The Player rolls the dice. 2. The Player lands on the monastery space. 3. The Player does not lose nor receive gold coins.
Postconditions: The Player's amount of gold coins is untouched.

Use Case Scenario: ExtraTurn
ID: 1.4
Brief description: The Player lands on the werewolf space
Primary actor(s): The Player
Preconditions: The Player rolls the dice and lands on the werewall space
Main flow: 1. The Player rolls the dice 2. The Player lands on the werewall space 3. The Player loses 80 gold coins 4. The Player starts his turn over and rolls again
Postconditions: The Player is left with 80 gold coins less than when his turn started.

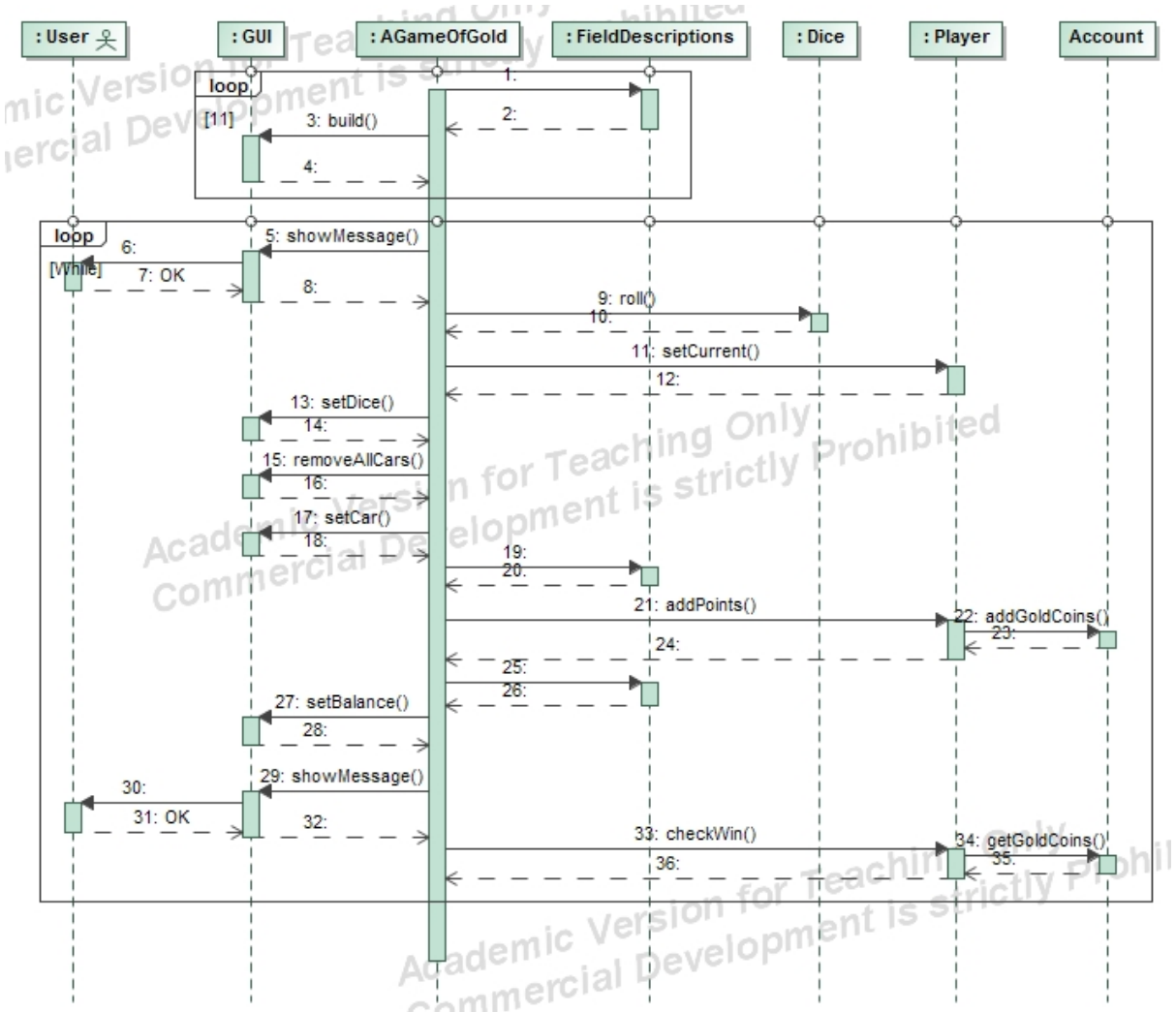
Sekvensdiagram

Her ses et sekvensdiagram der viser hvordan User og AGameOfGold kommunikerer sammen. Useren åbner spillet. AGameOfGold sender en besked tilbage og siger “Press Enter to roll”. Useren klikker på Enter for at starte sin tur. AGameOfGold ruller terningerne og viser værdien samt rykker han Useren til det felt med den givne værdi.



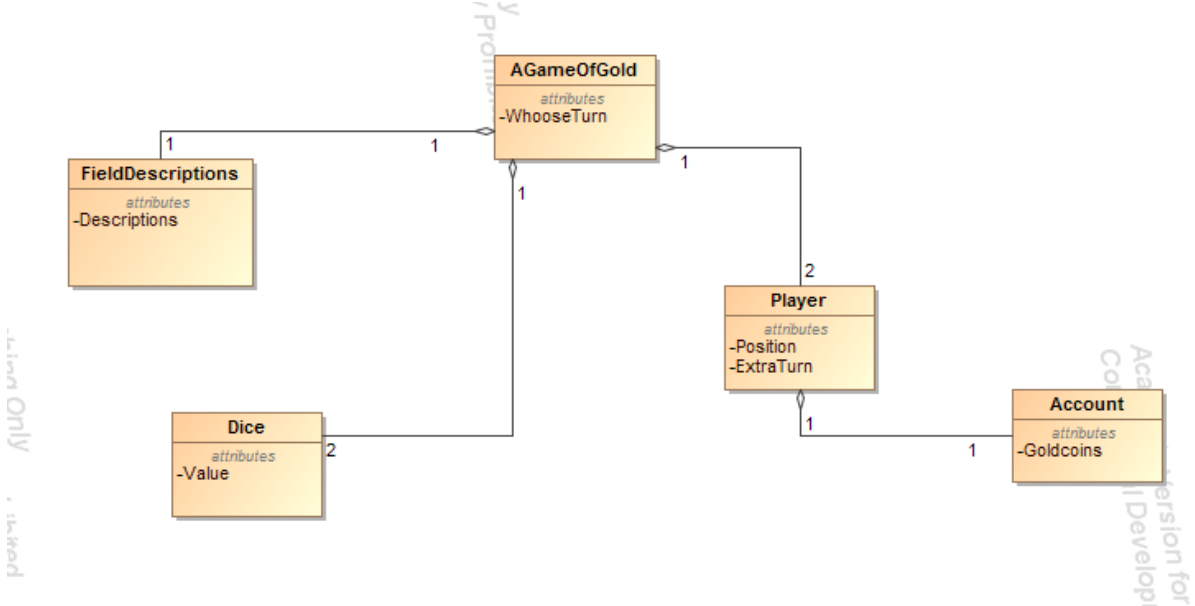
Design-sekvensdiagram

Design-sekvensdiagrammet viser en oversigt over hvordan brugeren samt klasserne kommunikere med hinanden. Systemet starter med at køre en for løkke 11 gange, hvor det opretter et nyt felt hver gang, med information det henter fra FieldDescriptions klassen. Så starter systemet en while løkke der kører en gang for hver spillers tur, det starter med at GUI'en viser en besked til brugeren og venter på at brugeren trykker OK. Systemet ruller så terningerne og gemmer resultatet i spilleren. Terningerne og bilen bliver så flyttet på GUI'en og spilleren bliver tildelt goldcoins for det felt der blev landet på og det bliver så også vist på GUI'en sammen med en beskrivelse af det felt der blev landet på. Så checker systemet om spilleren har vundet spillet og starter løkken forfra.



Domænemodel

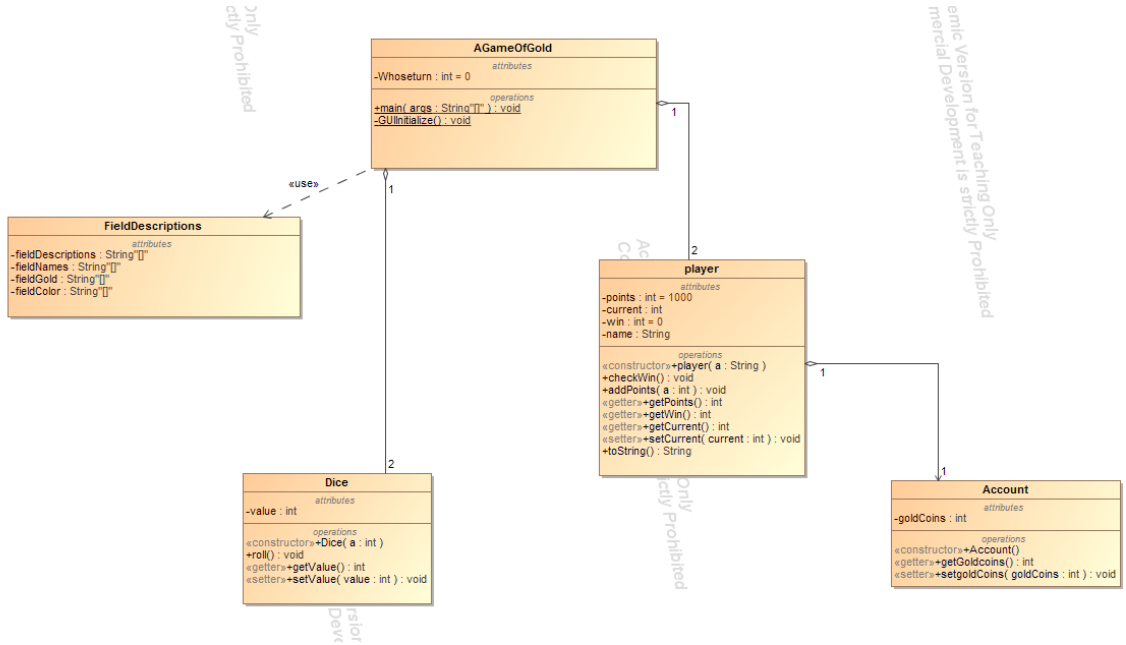
Denne domænemodel viser hvordan de forskellige klasser arbejder sammen i domænet og hvilke attributter de har. Her kan man se at AGameOfGold aggregerer med Player, Dice og FieldDescription hvor den kalder på deres metoder når de skal bruges. AGameOfGold står for hvis tur det er - Player 1 eller Player 2 - til at rulle med terningen. Man kan også se at vi benytter high cohesion og low coupling til vores klasser. Metoder fra FieldDescriptions, Dice, Player bliver kaldt fra AGameOfGold. Derudover aggregerer Player med Account, hvor Player kalder på metoden fra Account.



Design-klassediagram

Ligesom domænemodellen viser dette diagram relationen mellem klasserne. Udover dette viser design-klassediagrammet hvilke attributter og metoder, de forskellige klasser har.

F. eks kan vi se at klassen FieldDescriptions har attributterne for field, og at de alle er String arrays, men ingen metoder. Hvis vi kigger på Player klassen, kan vi se at den både har attributter og metoder.



CRC-kort

Nedenfor ses CRC fort for de forskellige klasser. CRC kortene viser, hvilke "ansvar" den enkelte klasse har, dvs. hvad de bidrager med af metoder og informationer i spillet. Collaborators fortæller om, hvilke andre klasser, den bestemte klasse afhænger af for at opfylde deres ansvar. Eksempel kan ses med nedenstående CRC kort for klassen FieldDescriptions. Dens ansvar er at give beskrivelser for f.eks. de forskellige felter i brætspillet. Dets "collaborator" er klassen AGameOfGold, da AGameOfGold er den klasse, som afgør hvor spilleren lander.

FieldDescriptions	
Responsibilities -Descriptions	**Collaborators** -AGameOfGold

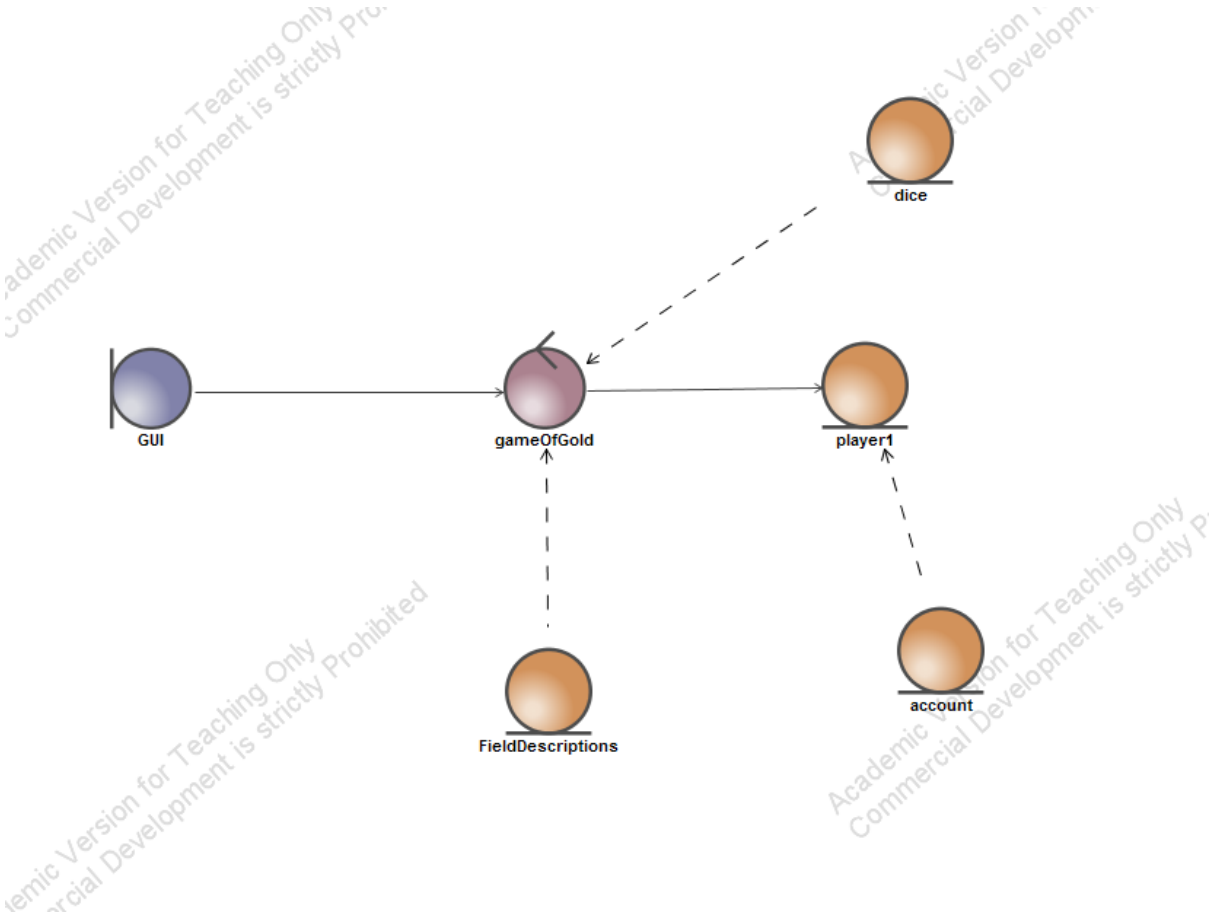
AGameOfGold	
Responsibilities -Whoseturn	**Collaborators** -Player -FieldDescription -Dice

Player	
Responsibilities -Position -ExtraTurn	**Collaborators** -AGameOfGold -Account

Dice	
Responsibilities -Value	**Collaborators** -AGameOfGold

Account	
Responsibilities -Goldcoins	**Collaborators** -Player

BCE-Model



I vores BCE-model (Boundary-Control-Entity) afbildes der, hvordan de forskellige klasser i spillet samarbejder for, at brætspillet kører optimalt. I denne model er vores grafiske brugergrænseflade (GUI) sat som vores boundary. Vores klasse AGameOfGold er sat til at være vores control, da det er den klasse, som spiller selve spillet, hvor den anvender informationer og metoder fra de andre klasser. Vores GUI er sat til at være associeret med AGameofGold, fordi vores GUI bruger oplysningerne fra AGameofGold klassen når spillet åbnes. Klasserne FieldDescriptions, Dice, Player og Account er alle sat som entity, fordi de interagerer med GameOfGold klassen. FiedDescription- og Dice klassen afhænger af klassen AGameofGold, da deres informationer og metoder er direkte associeret med vores "control"-klasse. AGameofGold er associeret med klassen Player, da det er klassen "Player" som interagerer direkte med klassen AGameOfGold. Klassen account afhænger af klassen Player.

Test

	jUnit1	jUnit2	jUnit3	jUnit4	jUnit5	TC1	TC2	TC3	TC4	TC5
k1						x				
k2							x			
k3		x								
k4								x		
k5	x									
k6	x									
k7					x					
k8									x	
k9			x							
k10				x						
K11										x

Test cases:

Navn	Scenarie	Input	Output	Forventet output	OK?	Kommentar
TC1	Krav 1	Klik på OK	Terningerne ruller	Terningerne ruller	Ja	Testet i databaren på DTU.
TC2	Krav 2	- Spiller 1 trykker OK - Spiller 2 trykker OK	Terningslag samt flytning af brik for begge spillere	Terningslag samt flytning af brik for begge spillere	Ja	Testet af to tilfældige personer på DTU.
TC3	Krav 4	Klik på OK	Brikken flyttes til det rigtige felt	Brikken flyttes til det rigtige felt	Ja	Brikken flyttes til feltet tilsvarende til terningkastet.
TC4	Krav 8	Klik på OK indtil brikken har passeret alle felter.	Tekst om feltet udskrives	Tekst om feltet udskrives	Ja	Eksempel: Brik lander på felt 2 -> teksten “A wild crater appears. You drop 100 gold coins into the crater.” udskrives.
TC5	Krav 11	Klik på OK indtil brikken har passeret alle felter.	Tekst samt egenskab af felt er unikt for hvert felt	Tekst samt egenskab af felt er unikt for hvert felt	Ja	Teksten og egenskaberne om det enkelte felt er unikt.

JUnit-test

jUnit 1:

I denne test trækker vi en masse gold coins fra spillerens account/beholdning, så meget at den ville gå under 0 hvis vi ikke havde skrevet kode som undgår dette.

jUnit 2:

I denne test ruller systemet terningen 60000 gange og tjekker derefter antallet af forekomster af de forskellige værdier. Terningen er god hvis hvert muligt slag er blevet slået 10000+-400 gange.

jUnit 3:

I denne test kalder systemet metoden points.getGoldcoins og tjekker om spillerens gold coins beholdning er 1000 gold coins, da dette er mængden spilleren starter med.

jUnit 4:

I denne test giver systemet spilleren 2000 gold coins for at tjekke om spillet slutter og om spilleren der har opnået 3000 gold coins får en besked om at de har vundet.

jUnit 5:

I denne test testes der om systemet giver spilleren den rette mængde gold coins afhængigt af hvilket slag spilleren har lavet.

Konklusion

Overordnet konklusion

Vi kan konkludere at gennem udviklingsværktøjet Eclipse har vi kunne producere et brætspil som opfylder kravspecifikationerne og dermed kan vi bekræfte vores hypotese. Vores diverse usecases og diagrammer har gjort arbejdsprocessen overskueligt og ud fra vores JUnit-Tests og Test-cases kan vi konkludere at spillet fungere som det skal.

Produktorienteret konklusion

Vi fik stillet en opgave hvor vi skulle udvikle et brætspil, som skal opfylde bestemte krav specifikationer. Ud fra kravene har vi analyseret og lavet diagrammer og use cases til at gå fra design til kode, hvilket har givet en indsigt på hvordan brætspillet skal kodes. Derudover har vi brugt GRASP metoden til at skabe overblik over designet til brætspillet og de klasser som vi har skabt samt hvordan de skal sættes sammen i vores kode. Dette har ført os til at vi har produceret det ønskede brætspil som stemmer overens med de specifikke kravspecifikationer der hører til.

Procesorienteret konklusion

Samarbejdet i dette projekt har været godt. Vi har brugt samme viden og metode fra CDIO-1 men også GRASP metoden til at løse opgaven. Github har også givet en god indsigt i hvordan man arbejder sammen kodemæssigt og dermed gjort det lettere at koordinere de forskellige arbejdsopgaver, dog har det været svært i starten af projektet at finde ud af hvordan Github fungerer.

Forslag til videre arbejde og forbedringer

- Man kunne evt. videreudvikle brætspillet til et større brætspil med flere felter og regler
- Gøre det muligt at være flere spillere

Litteraturliste

- https://www.campusnet.dtu.dk/cnnet/filesharing/SADownload.aspx?FileId=4298361&FolderId=1026792&ElementId=522003 - 02313, Udviklingsmetoder til IT-systemer. Lektion 06: Use case realisering, design. Forfatter: Ian Bridgwood, Henrik Bechmann - Afdeling for informatik, DTU Diplom. Sidst besøgt: 03-11-16
- https://www.campusnet.dtu.dk/cnnet/filesharing/SADownload.aspx?FileId=4291632&FolderId=1025146&ElementId=522003 - 02313, Udviklingsmetoder til IT-systemer. Lektion 05: Analyse, UML notation. Forfatter: Henrik Bechmann - Afdeling for informatik, DTU Diplom. Sidst besøgt: 03-11-16
- https://www.campusnet.dtu.dk/cnnet/filesharing/SADownload.aspx?FileId=4285254&FolderId=1023805&ElementId=522003 - 02313, Udviklingsmetoder til IT-systemer. Lektion 04: Use case modellering. Forfatter: Henrik Bechmann - Afdeling for informatik, DTU Diplom. Sidst besøgt: 03-11-16
- https://www.campusnet.dtu.dk/cnnet/filesharing/SADownload.aspx?FileId=4277841&FolderId=1021827&ElementId=522003 - 02313, Udviklingsmetoder til IT-systemer. Lektion 03: Rapportskrivning. Forfatter: Henrik Bechmann, Henrik Tange - Afdeling for informatik, DTU Diplom. Sidst besøgt: 03-11-16
- CDIO1 Aflevering
- Larman, Craig: “Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development”, tredje udgave, Addison Wesley Professional, Oktober 20 2004.

Bilag

Bilag 1:

1	Tower	+250	You've entered the tower and find a chest at the top containing 250 gold coins.
2	Crater	-100	A wild crater appears. You drop 100 gold coins into the crater.
3	Palace Gates	+100	You've found the Palace Gates! You gain 100 gold coins and receive blessings.
4	Cold Dessert	-20	Brace yourself! Winter has come and you are forced to spend 20 gold coins on a new coat.
5	Walled City	+180	A walled city saves you the trouble of spending 180 gold coins on a bodyguard.
6	Monastery	0	You enter the monastery. Nothing exciting here.
7	Black Cave	-70	A thief robs you of 70 gold coins.
8	Huts In The Mountains	+60	A pack of strange creatures are pleased with your visit and gifts you 60 gold coins.
9	The Werewall	-80, +tur	You've been bit by a werewolf and temporarily become a werewolf. Therefore you can roll at night. You must roll again to lose werewolf form but you lose 80 gold coins.
10	The Pit	-50	You stare down the pit. You are frightened and decide to run forgetting your sack of 50 gold coins.
11	Goldmine	+650	You dig deep into the mountains and discover a goldmine. You start to dig for gold and your efforts grants you 650 gold.