

**CDIO delopgave 3
'A Game of Coins'**

Udviklingsmetoder til IT-systemer 02313, Softwareteknologi

Gruppe Nr.: 31

Afleveringsfrist: Fredag 25/11-2016 23:59

Institut: DTU Compute

Vejleder: Ronnie Dalsgaard

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

Denne rapport indeholder 39 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

Abstract

The project described in this report is a board game based on the requirements brought by the customer and IOOuterActive. For this project, we have been using a GUI (Graphical User Interface) as a tool for the user to interact with the program. The program has been designed to be able to run on all machines and databars on DTU. To make our development- and coding process easier, we have been using different UML diagrams to give ourselves a more manageable perspective of our program structure. Additionally, we have been running JUnit tests on all of our relevant classes to assure ourselves that our program does not have any game breaking flaws. For this report the primary focus will be centered around our design models/diagrams (including theoretic explanations of those), test cases, JUnit tests (with documentations of those) as well as explanations of our theoretical terms. Lastly, we can conclude that our project has been very successfully developed and tested on the databars at DTU.

Contents

Abstract	1
Timeregnskab	3
Indledning	5
Emne	5
Formål	5
Kravspecifikationer	5
Navneords Analyse:	6
Udsagnsord Analyse:	6
Begrebsdefinition	7
BCE:	7
GUI:	7
CRC:	7
GRASP:	7
FURPS+:	7
Hypotese	8
Teori	8
UP	8
Grasp	8
Arv	9
Abstrakt klasse/polymorfi	9
Metode	9
Hovedafsnit	10
Domænemodel	10
CRC-kort	11
Use cases	13
Sekvensdiagram	19
BCE model:	20
Design Klassediagram	21
Design Sekvensdiagram	22
StartGame:	22
Land on Fleet:	23

Test.....	23
Formelle test cases	25
JUnit-tests	32
Acceptance test.....	34
Konfiguration:	35
Konklusion.....	36
Overordnet konklusion	36
Produktorienteret konklusion.....	36
Forslag til videre arbejde og forbedringer	36
Litteraturliste	36
Bilag.....	37

Timeregnskab

CDIO delopgave 03							
Timeregnskab	Vers. Uge 45						
Dato:	Deltager	Design	Impl.	Test	Dok.	Andet	I Alt
09-10/11/16	Christopher	5					5
09-10/11/16	Jonathan	5					5
09-10/11/16	Thomas	5					5
09-10/11/16	Johnny	4.5			0.5		5
09-10/11/16	Bertram	5					3
	Sum	24.5			0.5		25

CDIO delopgave 03							
Timeregnskab	Vers. Uge 46						
Dato:	Deltager	Design	Impl.	Test	Dok.	Andet	I Alt
16-17/11/16	Christopher	4					4
16-17/11/16	Jonathan	4	5				9
16-17/11/16	Thomas	4	2.5		2		8.5
16-17/11/16	Johnny	2	4.5		2		8.5
16-17/11/16	Bertram	4	5				9
	Sum	18	17		4		39.5

CDIO delopgave 03							
Timeregnskab	Vers. Uge 47						
Dato:	Deltager	Design	Impl.	Test	Dok.	Andet	I Alt
23-24-25/11/16	Christopher	3	0	2	5	1	11
23-24-25/11/16	Jonathan	2	1	2	5	1	11
23-24-25/11/16	Thomas	1	1.5	3	3		10.5
23-24-25/11/16	Johnny	2	2	1	4		9
23-24-25/11/16	Bertram	1	7	1	3		12
	Sum	9	11.5	9	20	2	53.5

Indledning

Emne

Videreudvikling af vores brætspil med formål at tilpasse det til et decideret brætspil samt udbygelse af flere og forskellige typer af felter, hvor der også vises brug af FURPS+.

Formål

Formålet med dette projekt er at videreudvikle brætspillet fra CDIO-2 med 2-6 spillere og 21 felter med forskellige funktioner, hvor man skal kunne gå i ring på brætspillet, som vi har udarbejdet i overensstemmelse med kunden. Rapporten indeholder en analyserende del bestående af diagrammer og use cases, som hjælper med give et overblik over processen og nedbryde diverse problemløsninger fra krav til design og kode.

Kravspekifikationer

Functional:

1. Systemet skal kunne simulere et slag med to 6-sidede terninger.
2. Systemet skal flytte en brugers brik til det rigtige felt på brættet alt efter summen af terningslaget.
3. Systemet skal kunne holde styr på brugerens pengebeholdning.
4. Systemet skal sørge for at ændre brugerens pengebeholdning alt efter hvilket felt han/hun lander på og overføre penge til en anden spiller hvis feltet allerede er eget.
5. Systemet skal give spilleren mulighed for at vælge om han/hun vil købe et felt, som han/hun er landet på.
6. Systemet skal sørge for, at brugeren går bankerot (dør i spillet) hvis brugeren går i 0 eller negativ pengebeholdning.
7. Systemet skal udskrive en tekst, der omhandler det aktuelle felt.
8. Systemet skal sørge for, at samtlige brugere starter med 30.000 gold.
9. Systemet skal afsluttes når alle på nær én er gået bankerot.
10. Systemet skal have felter med unikke egenskaber og navne. Se bilag 1.

Usability:

11. Systemet skal være et spil mellem to til seks brugere på samme computer.

Reliability:

12. Systemet skal have lige stor chance for at slå 1-6 med terningerne, som med en fysisk terning.
13. Systemet skal kunne fungere fejlfrit i minimum én time.

Performance:

14. Systemet skal kunne bruges på maskinerne i data-barerne på DTU.
15. Systemet skal kunne vise resultat af kastet inden for 0.6 sekunder.

Supportability:

- ingen krav

Implementation:

16. Systemet skal udvikles i Eclipse.
17. Systemet skal bruge UTF-8 som tegnsæt.
18. Systemet skal kodes i Java.

Navneords Analyse:

- Systemet
- Maskine
- Data-bar
- Bruger
- Terning
- Felt
- Bræt
- Sum af terningenslag
- Pengebeholdning
- Tekst (der omhandler felt)
- Gold
- Unikke egenskaber og navne
- Bilag

Udsagnsord Analyse:

- Bruges (på maskinerne)
- Være (et spil)
- Simulere (et slag)
- Flytte (en brugers brik)
- Holdet styr på (Pengebeholdning)
- Ændre (Pengebeholdning)
- Lande (På et felt)
- Går bankerot
- Går i 0
- Udskrive (tekst, der omhandler felt)
- Starter (30.000 gold)
- Afslutte (Når alle går bankerot, nær én)

Ved at lave en analyse af navne- og udsagnsord på baggrund af kundens vision om det ønskede produkt, kan man nemmere få sig et mere overskueligt blik over de forskellige klasser og metoder, som produktet bør indeholde.

Følgende navneord er blevet lavet som klasser i vores program:

- SYSTEMET: Er vores controller i programmet, som styrer hvordan vores spil kommer til at forløbe.
- TERNING: vores dice-klasse, som bruges til at få en tilfældig værdi ud fra de to terninger der bliver kastet i spillet.
- FELT: vores Field-klasse, som er en superklasse for de andre subklasser af felter, som hver afgør hvor mange coins du enten får eller mister.
- PENGEBEHOLDNING: vores account klasse, som bruges til at holde styr på, hvor mange coins spilleren har, får tildelt, og mister igennem spillet.
- BRUGER: vores player-klasse, som er den spiller der interagerer med programmet.
- BRÆT: vores GameBoard klasse, som bruges til at opbevare oplysninger om vores felter.

Følgende udsagnsord er blevet lavet til metoder:

- **SIMULERE** (et slag): vores rollDice()-metode, som bruges når vi skal få to tilfældige værdier af vores Dice-objekter.
- **ÆNDRE** (pengebeholdning): vores addCoins(int coins) metode, som i account-klassen tilføjer penge til brugeren afhængigt af hvilket felt man er landet på. Den samme metode gør sig gældende for udsagnsordet "HOLDE (styr på)", da det er det samme formål de tjener.
- **FLYTTE** (en brugers brik): vores movePosition(int amount) metoder, som er i Player-klassen, placere spillerne på de forskellige felter afhængigt af hvilket terning sum man har slået.
- **LANDE** (på et felt): vores landOnField(Player player) abstract metode, som bliver overrider i forhold til hvilken type felt det er. Og de bestemte type felter har hver sin bestemte regler.

Begrebsdefinition

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.

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.

FURPS+:

Forkortelse for Functional, Usability, Reliability, Performance, Supportability, hvor "+" står for følgende Implementation, Interface, Operations, Packaging og Legal. FURPS+ er et hjælpsfuld værktøj til af lave en checkliste over ens dækning af kravspecifikationerne. I vores opgave benytter vi Functional, Usability, Reliability, Performance, Supportability og Implementation.

Hypotese

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

Teori

UP

Vi har brugt Unified Process(hf. UP) til at dele projektet op i flere, mere overskuelige faser. Da selve spil-idéen er givet til os i forvejen har inception-delen ikke fyldt meget. Vi har skitseret kravspecifikationerne for videre at udarbejde og finpudse dem i løbet af elaboration-fasen. I denne fase inddrager vi yderligere teori i form af modeller og diagrammer, som beskriver, hvordan spilsystemet er bygget op; vores use case-diagrammer, domænemodel mm. Ud fra det analyserende arbejde koder vi selve spillet. Elaboration- og Construction-fasen har dog ikke været to adskilte processer, men derimod en dynamisk overgang mellem de to faser. Vi er i kode-fasen f.eks. gået tilbage og opdateret vores CRC-kort, da vi i udviklingsprocessen fandt ud af, at klasserne kunne interagere bedre, mere fornuftigt jf. low coupling/high cohesion. Til sidst har vi i Transition-fasen testet vores spil 'A Game Of Coins' på en computer i DTU's databar for at sikre, at implementeringen fungerede som planlagt.

Grasp

Creator:

I vores system er det klassen Controller, som er vores creator. Dette er fordi, at Controller er den klasse, som opretter alle objekter af de andre klasser og det er her selve "spillet" foregår. Udover dette giver det ikke mening at nogen af de andre klasser skulle være creator, da det f. eks ikke giver mening at PlayerList klassen skulle være den klasse, som opretter et GameBoard objekt. Det giver mere mening at have en klasse for sig selv, som opretter alle objekterne.

Information expert:

I vores system er det klassen Descriptions, som er vores information expert. Dette er fordi, at Descriptions klassen, er den klasse, som indeholder information omkring felterne, så når der bliver oprettet felt objekter, henter Controller klassen information fra Descriptions. Det er dog ikke kun Descriptions der indeholder information, både Player- og Account klassen indeholder information, men det er i et mindre omfang, så derfor har vi valgt Descriptions som vores Information expert.

Low Coupling:

I vores system har vi vidt som muligt prøvet at overholde low coupling. Vi har prøvet at sørge for at så få klasser, som muligt snakker med hinanden, så lidt som muligt. Vi har dog ikke helt overholdt dette, da klassen Field sender information til vores GUI, hvilket ikke stemmer helt overens med Low Coupling mønsteret. I vores krav fra kunden, som vi ikke ved hvordan man skal kunne overholde uden at bryde dette mønster. Vi har fået at vide at vi ikke må ændre i nogen af metoderne i den givne domænemodel, derfor føler vi ikke at vi har haft mulighed for at overholde low coupling helt.

Controller:

Controlleren i vores klasse må være GUIHandler, da dette er den klasse som er den første, som sender information til vores GUI.

High cohesion:

I vores system har vi high cohesion, da vi mener at hver klasse kun har de nødvendige funktioner i sig. En fingerregel vi har brugt er at man i hver klasse skal kunne ses al koden uden at skulle scrolle ned. Selvfølgelig afhænger dette af hvor stor skærm man har, men vi vil alligevel tillade os at sige at vi har high cohesion, da hver klasse kun har de nødvendige funktioner.

Arv

Arv er en relation mellem klasser. Det beskriver en "is-a" relation. Fordelen ved arv er man ikke behøver at gentage fælles kode. I vores kode har vi lavet nedarv fra vores super klasse Field. Her nedarver Ownable, Refuge og Tax. Territory, LaborCamp og Fleet nedarver fra klassen Ownable, hvor de har konstruktøren Ownable til fælles. Dette kaldes arv.

Abstrakt klasse/polymorfi

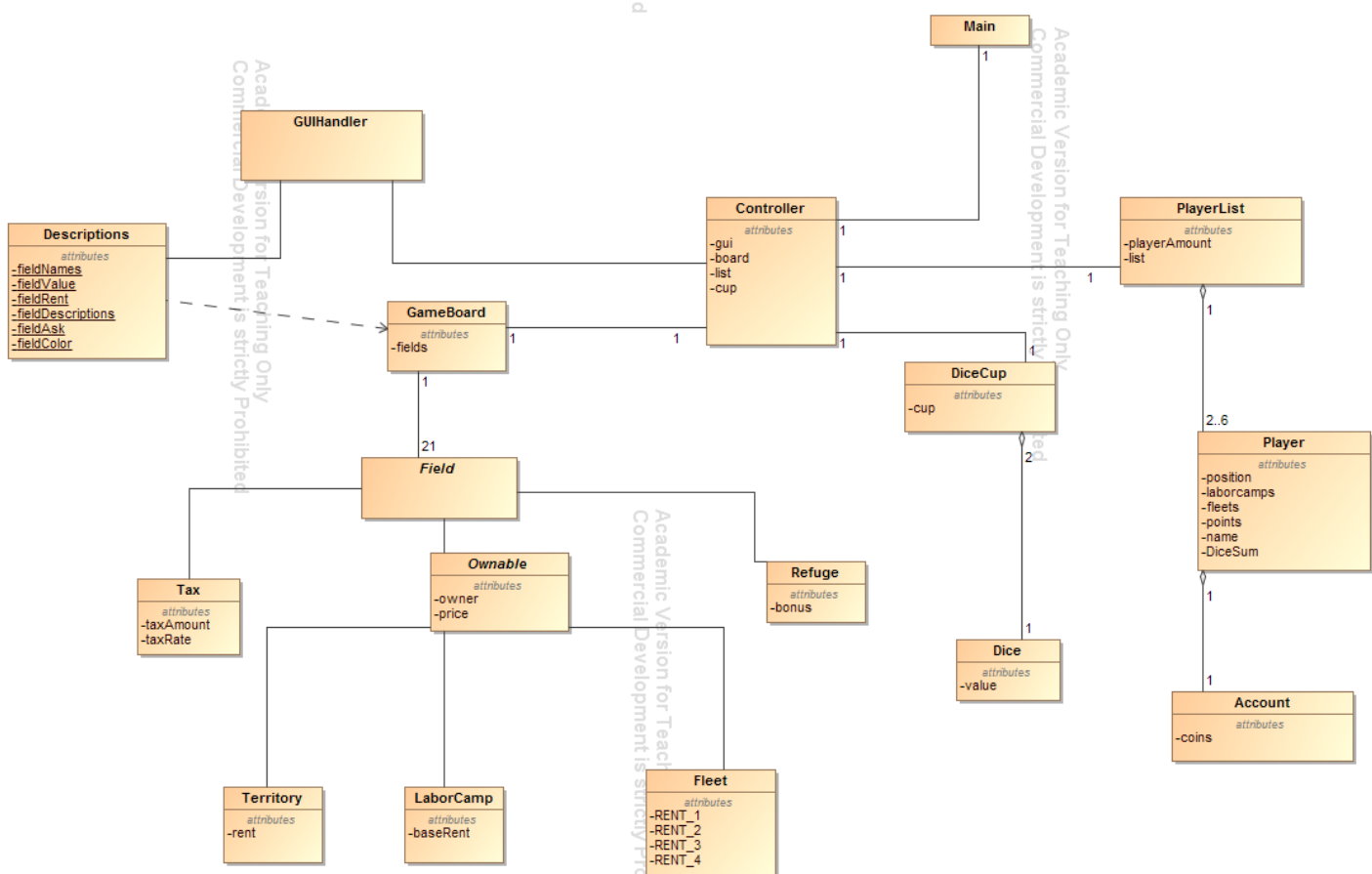
En abstrakt klasse er en klasse, som ikke kan instantieres, men som stadig godt kan have subklasser. I vores projekt har vi lavet Field klassen til en abstrakt klasse, da dette tillader os at override Fields metode LandOnField i subklasserne Tax, Territory, Refuge, LaborCamp og Fleet. Hvilket betyder at alle subklasserne har samme metode, men at metoden gør noget forskelligt alt efter hvilket klasse den er i. Dette er kaldet for polymorfi.

Metode

For at udvikle vores brætspil har vi brugt samme metode som i vores tidligere opgave CDIO-2. Vi har 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. En ændring siden CDIO-2 har været at vi har brugt en anden model til vores kravspecifikation, som hedder FURPS+. Derefter har vi kodet spillet ud fra vores usecases og diagrammer.

Hovedafsnit

Domænemodel



Domænemodellen viser et diagram over brætspillets forskellige klasser. Her kan man se hvem der arbejder sammen og hvilke attributter de har. Main starter spillet ved at invokerer Controller, som opretter spillet. Controller henter metoder fra de klasserne Gameboard og Dicecup, hvor de også henter informationer og metoder fra de klasser der er associeret. Ud over dette viser domæne modellen også multipliciteten. F. eks er der 21 fields til et gameboard. Vi har udviklet videre på systemet efter vi har lavet dette diagram, hvilket betyder at diagrammet ikke stemmer helt overens med det aktuelle system. F.eks. sender Field klassen information til vores GUI, hvilket ikke kan ses ud fra vores domænemodel.

CRC-kort

Nedenfor ses CRC kort 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.

Dice	
Responsibilities -Value	Collaborators -Dicecup

Account	
Responsibilities -Coins	Collaborators -Player -Playerlist

Player	
Responsibilities -Position -Assets -Fleets -LaborCamps	Collaborators -Controller -Account -Playerlist

Controller	
Responsibilities -StartGame -Whoseturn -addPlayers	Collaborators -PlayerList -DiceCup -GameBoard -GUIHandler

Field	
Responsibilities	Collaborators -Controller -Ownable, subclass -Tax, subclass -Refuge, subclass

Refuge	
Responsibilities -Bonus	Collaborators -Field, superclass

Tax	
Responsibilities -TaxAmount -TaxRate	Collaborators -Field, superclass

Ownable	
Responsibilities -Price -Owner	Collaborators -Field, superclass -Territory, subclass -LaborCamp, subclass -Fleet, subclass

Territory	
Responsibilities -Rent	Collaborators -Ownable, superclass

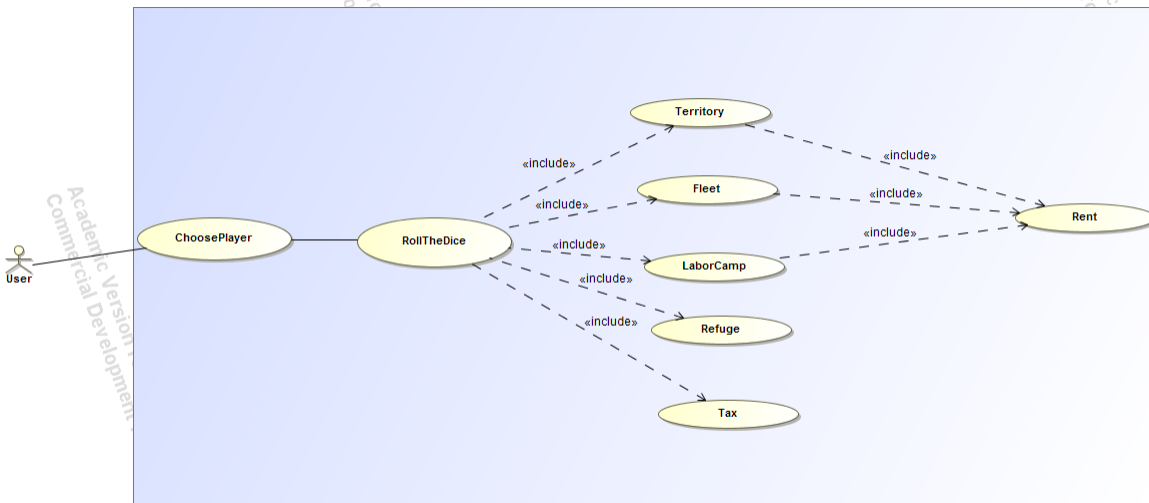
LaborCamp	
Responsibilities -BaseRent	Collaborators -Ownable, superclass

Fleet	
Responsibilities -Rent	Collaborators -Ownable, superclass

DiceCup	
Responsibilities -rollDice	Collaborators -Dice -Controller

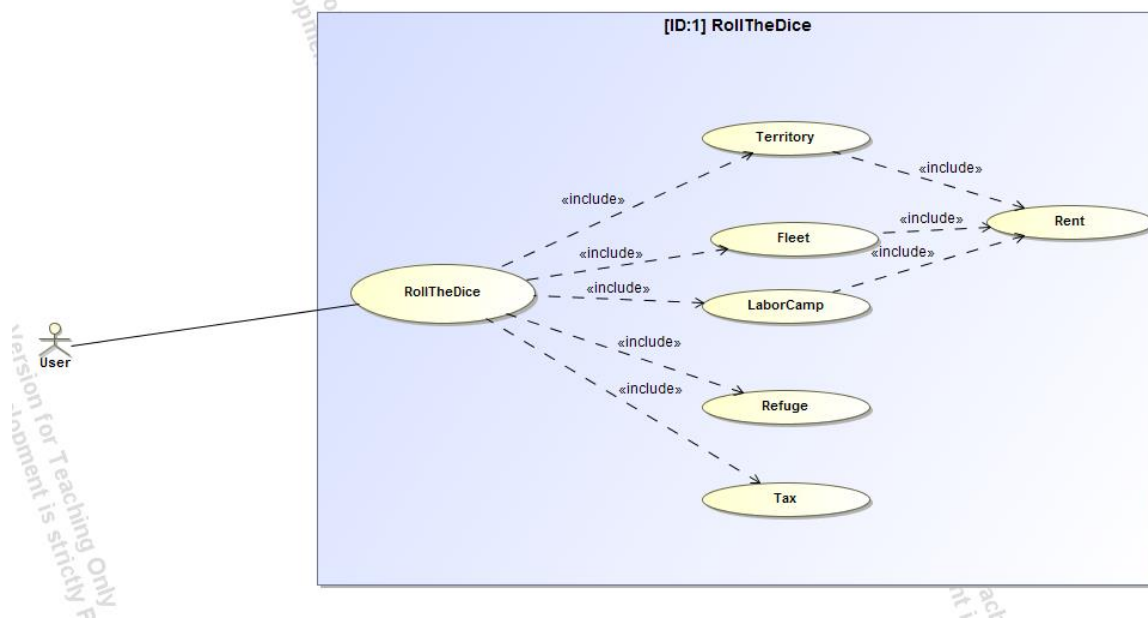
Use cases

Overordnet use case diagram for systemet:



Her ser vi use case diagrammet for hele programmet. En bruger kan to ting i dette system: Vælge spillerne, som består af at vælge antallet og navnene af spillerne, og at slå med terningerne. Dog skal man have valgt spillerne før kan rulle med terningerne. Når man har slået med terningerne lander spiller på et af følgende felttyper: Territory, Fleet, LaborCamp, Refuge eller Tax. Hvad der sker når man lander på en specifik type felt er beskrevet neden under med diagrammer og “fully dressed use cases”.

Use case RollTheDice:



Use Case: RollTheDice
ID: 1
<p>Brief description:</p> <p>A Player rolls two dice by pressing a button, then the system moves the player accordingly to the roll.</p>
<p>Primary actors:</p> <ul style="list-style-type: none"> • Player
<p>Secondary actors:</p> <p>None</p>
<p>Preconditions:</p> <ul style="list-style-type: none"> • Two to six players • A computer
<p>Main flow:</p> <ol style="list-style-type: none"> 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 accordingly correct space. 5. The system reacts accordingly to the field the player landed on.
<p>Postconditions:</p> <ul style="list-style-type: none"> • The system successfully moved the played based on the dice roll and the according space • The system reacts accordingly to field the player landed on.

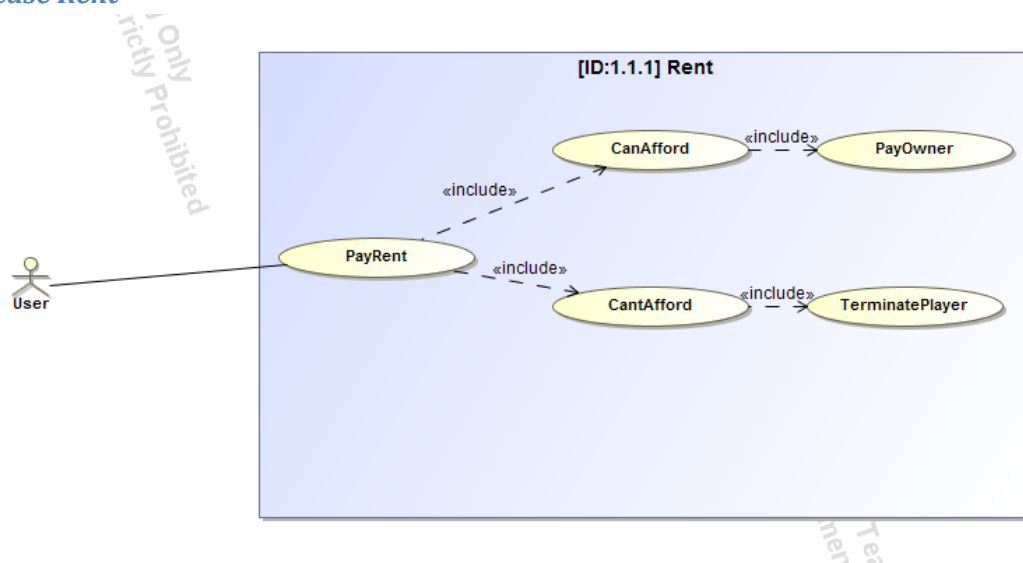
Use case Territory/Fleet/LaborCamp

Til denne use case har vi valgt at undlade et diagram, da vi følte det var så simpelt at det var spild af tid. Brugeren har nemlig kun et valg; at købe eller ikke at købe. Udover det har vi valgt at samle de tre use cases: Territory, Fleet og LaborCamp, da de har samme funktion, nemlig at man skal kunne købe dem.

Use case: Territory/Fleet/LaborCamp
ID: 1.1
<p>Brief description:</p> <p>If another player already owns this field look in Rent description to see what happens. If the player already own this field nothing happens. After rolling the dice the players lands on a Territory/Fleet/LaborCamp space, which means the player gets to choose yes or no to buy the space. If player cannot afford the space, the system replies "Insufficient funds!"</p>
<p>Primary actors:</p> <p>Player 1</p>

Secondary actors: none
Preconditions: -The player lands on a Territory/Fleet/LaborCamp space
Main flow: The player has sufficient funds 1. The player rolls the dice. 2. The system moves player to Territory/Fleet/LaborCamp space. 3. The player chooses yes 4. The system takes the accordingly price from the player.
Postconditions: -The system successfully takes the price from the player. -The player now owns the bought space and will receive rent from anyone who lands on his space.

Use case Rent



Use case: Rent
ID: 1.1.1
Brief description: If the player lands on a Labor camp, territory or fleet space, the players must then pay rent to the owner of the space, with the amount varying from space to space.
Primary actors: Player 1
Secondary actors: Player 2

<p>Preconditions:</p> <ul style="list-style-type: none"> -The player lands on a space that has rent. -Another player owns the space the player landed on.
<p>Main flow:</p> <ol style="list-style-type: none"> 1. The players lands on a space that has rent and is owned already. 2. The player pays rent to the owner. 3. The system transfer coins from player to owner. 4. The owner receives coins from player.
<p>Secondary flow:</p> <ol style="list-style-type: none"> 1. The player lands on a space that has rent and is owned already 2. The system tries to take the right amount of coins. 3. The player can't afford to pay his rent. 4. The system marks this player as bankrupt and the player is out of the game.
<p>Postconditions:</p> <p>Main flow:</p> <ul style="list-style-type: none"> -The system successfully takes the right amount of coins, according to the space, from the player and gives it to the owner of the space. <p>Secondary flow:</p> <ul style="list-style-type: none"> -The system marks the player as bankrupt if the player can't pay his rent.

Use case Refuge

I denne use case har vi også valgt ikke at tegne et diagram, da det er system som overfører penge til spilleren.

Use case: Refuge
ID: 1.2
<p>Brief description:</p> <p>After rolling the dice the players lands on a refuge space, the system grants the player coins.</p>
<p>Primary actors:</p> <p>Player 1</p>
<p>Secondary actors:</p> <p>none</p>
<p>Preconditions:</p> <ul style="list-style-type: none"> -The player lands on a refuge space

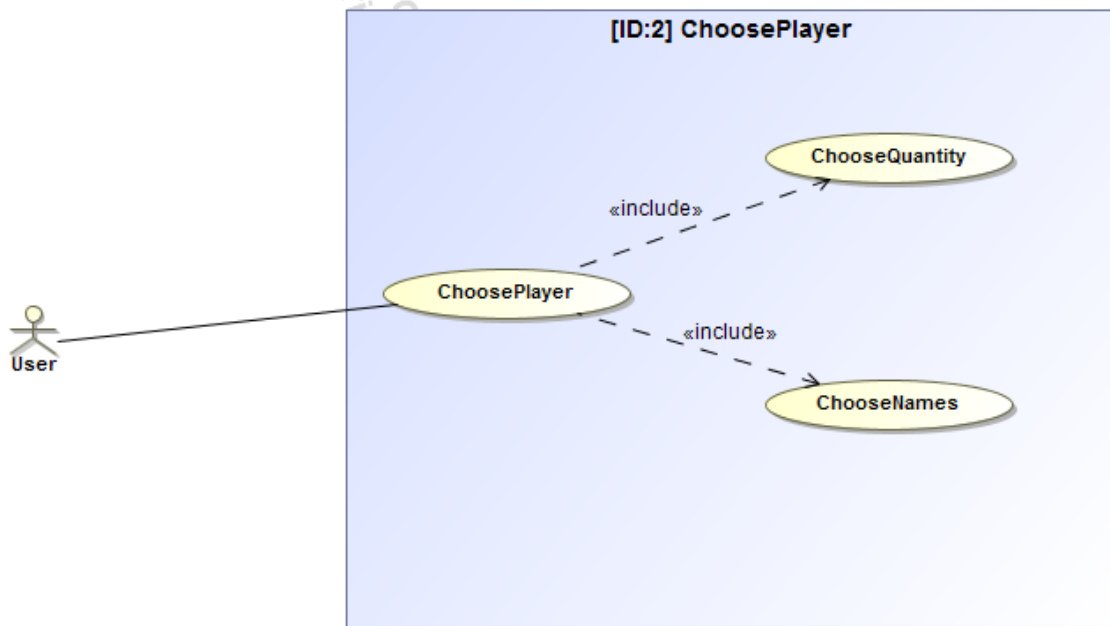
Main flow:

1. The player lands on a refuge space.
2. The system grants the player coins.

Postconditions:

-The system successfully gives the player coins
accordingly to the refuge space.

Use case ChoosePlayer



Use case: ChoosePlayer

ID: 2

Brief description:

The user chooses how many players there are and what their names are.

Primary actors:

The user

Secondary actors:

None

Preconditions:

-Two to six players
-a computer

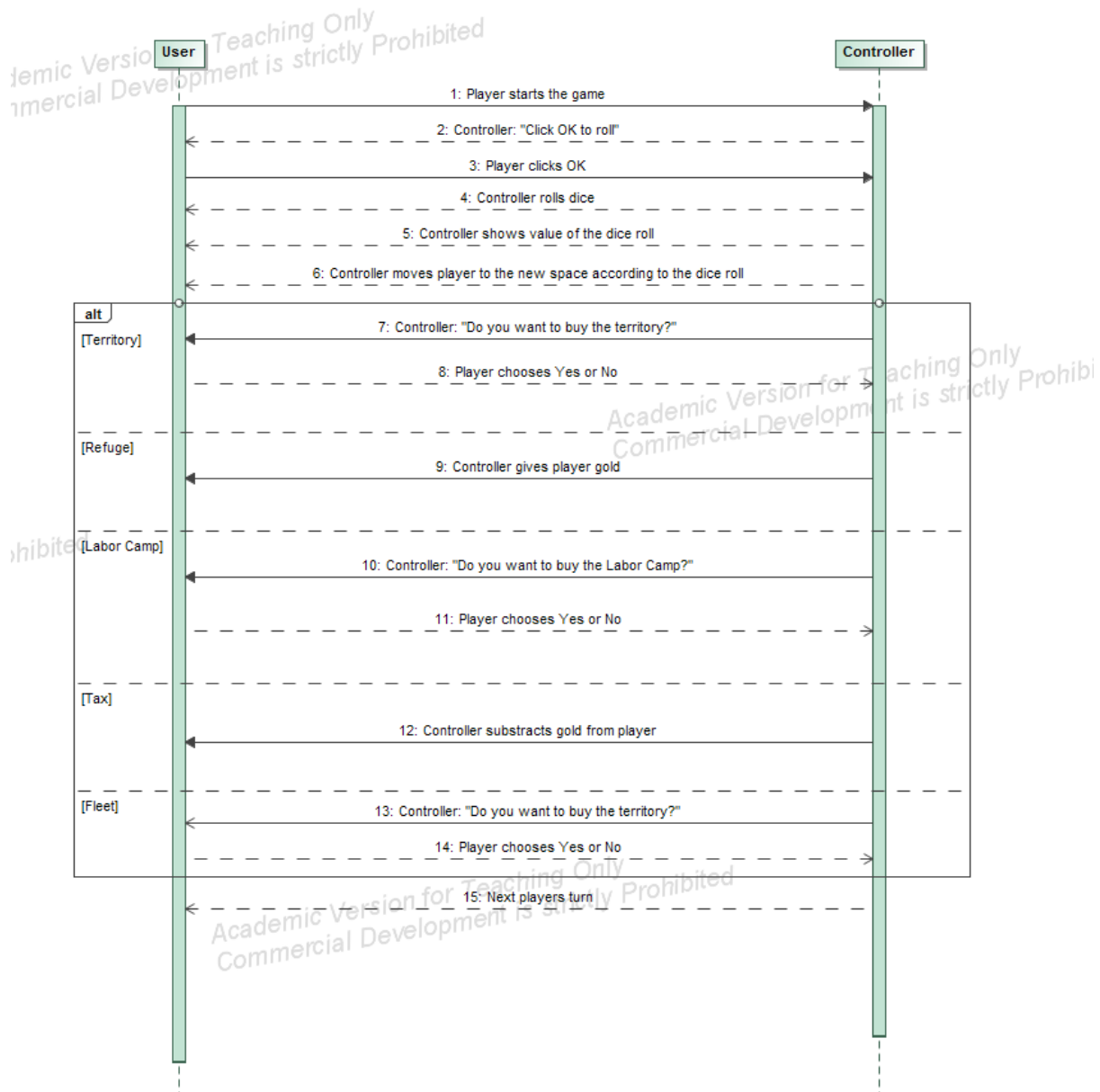
Main flow:

1. User presses "Choose players"
2. User decides number of players and tells the system.
3. System creates players.
4. User types in player names.
5. Game starts.

Postconditions:

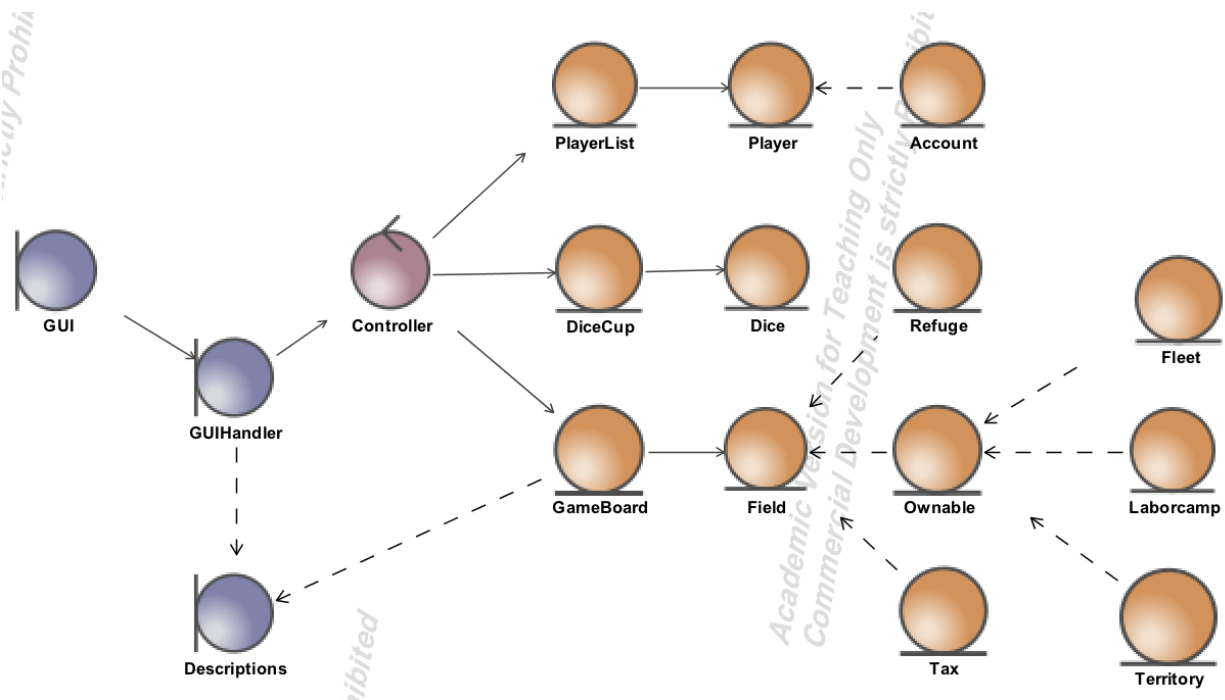
The system successfully created the right number of players the corresponding names and starts the game.

Sekvensdiagram



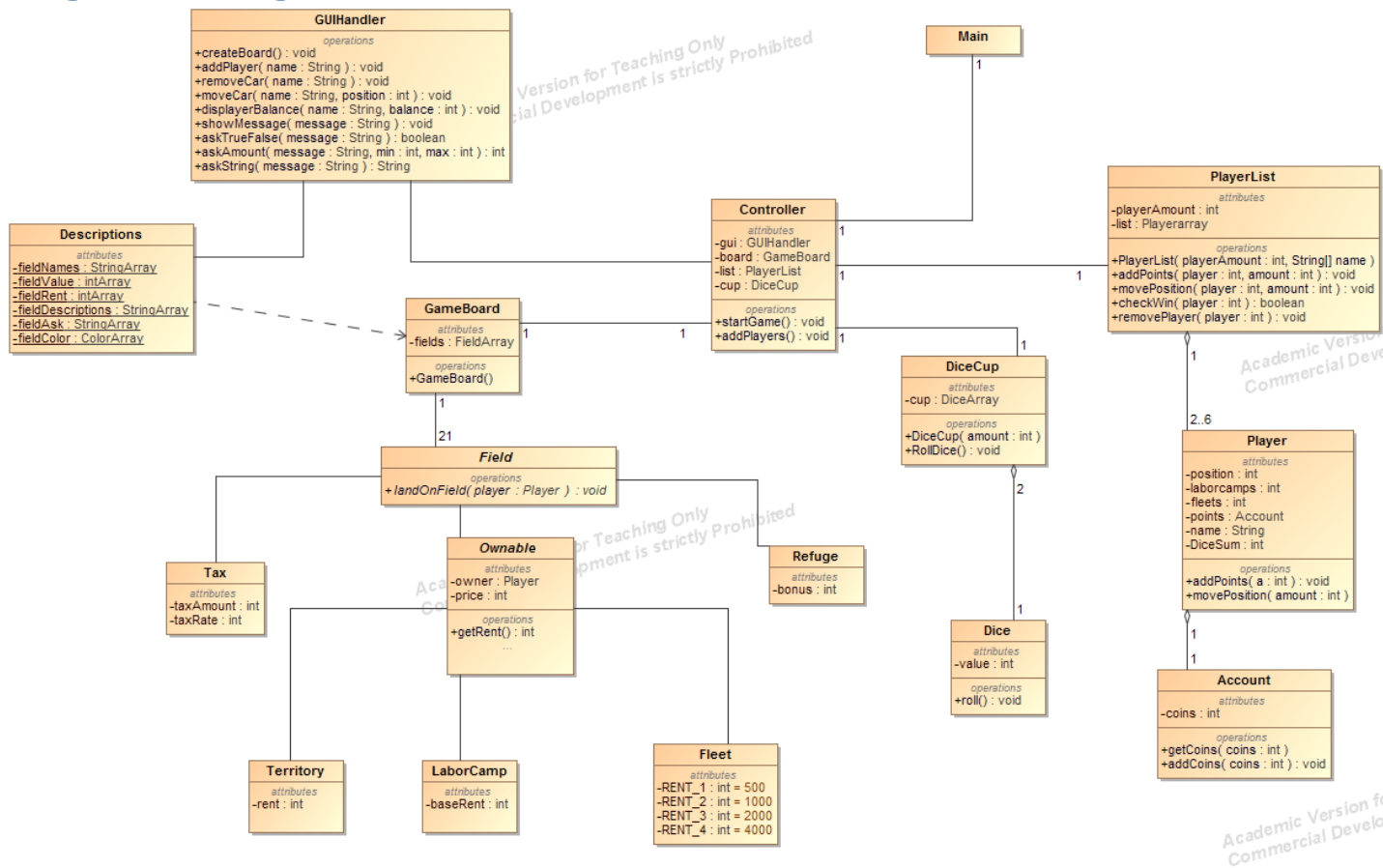
Her ses sekvens diagram der viser hvordan en User og Controller kommunikerer sammen. User starter spillet. Controller sender en besked tilbage til User og siger "Click OK to roll". User klikker "OK". Controller ruller terningerne og viser værdien, samt rykker User til det felt med den givne værdi. Herefter er der lavet en "If condition" som sender en bestemt besked afhængigt af hvad værdien er terningerne er. Efter "if condition" er ens tur slut.

BCE model:



I vores BCE-model vises der hvordan diverse klasser arbejder sammen for at kører brætspillet. Her er selve vores brugerinterface GUI, hvor GUI, GUIHandler og Descriptions er sat som boundary. Vores Controller er Controller, som opretter vores Gameboard, DiceCup og PlayerList entities. Controlleren henter metoder og informationer fra Entity klasserne, hvor den så sender informationerne videre til GUIHandler som opretter en grafisk interface (GUI) til brugeren.

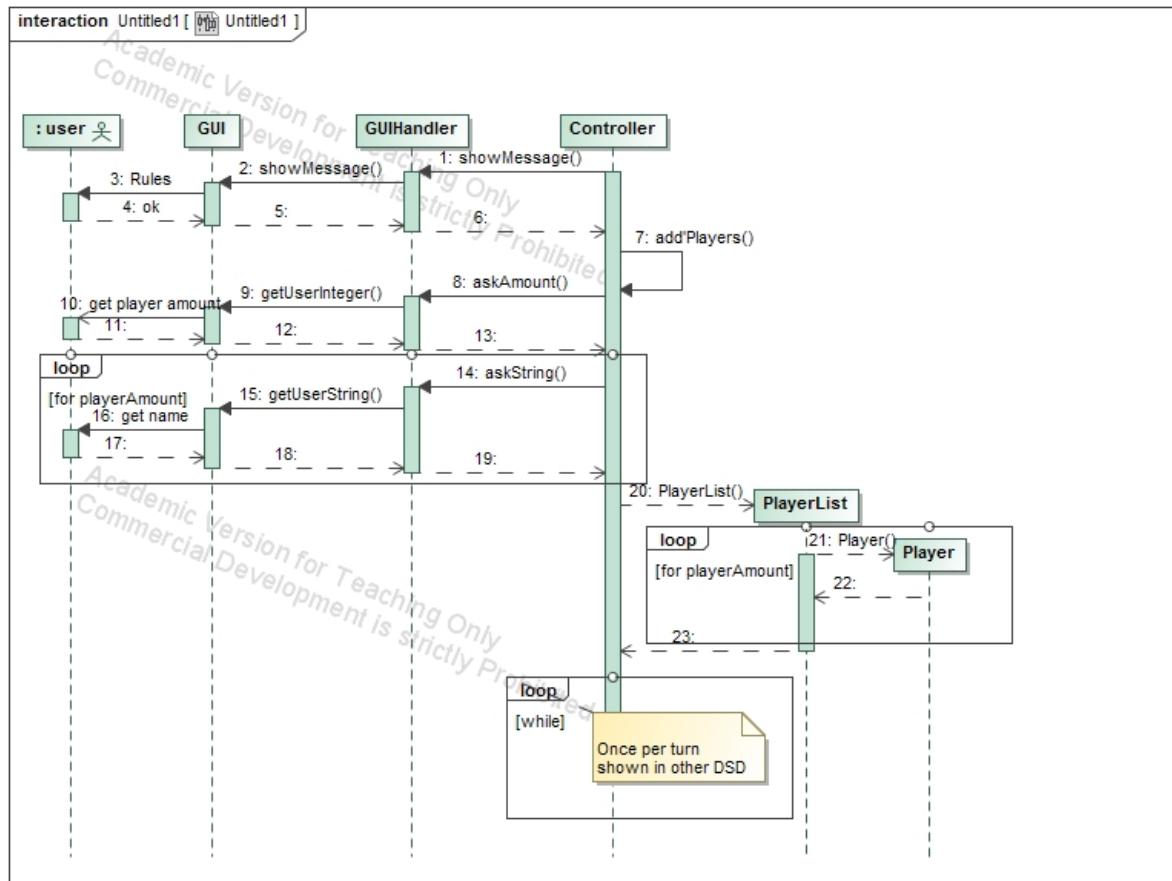
Design Klassediagram



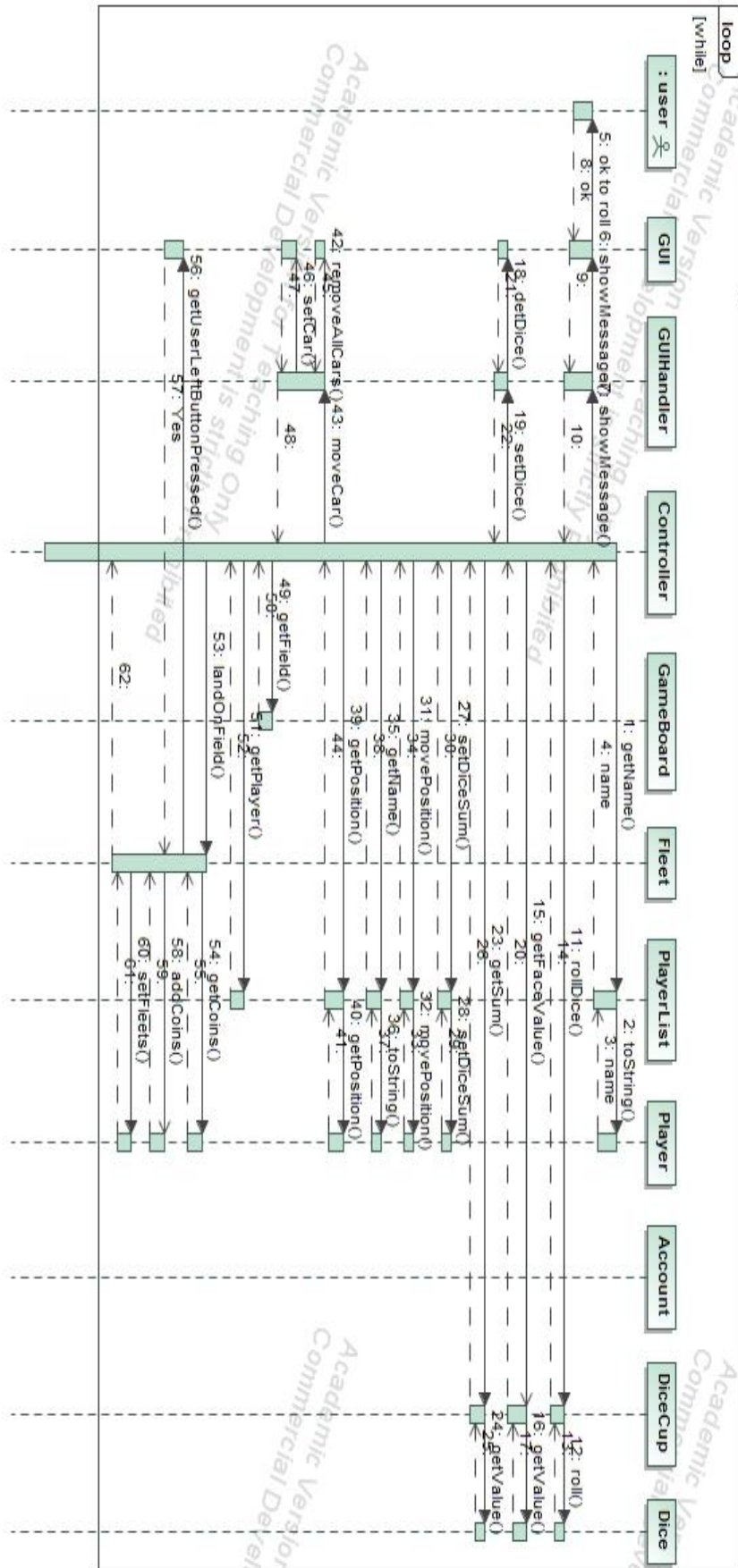
Her kan vi ses vores Design Klassediagram. Ligesom domæne modellen viser dette relationerne mellem klasserne. Udover det viser diagrammet også klassernes type af attributter og type af metoder.

Design Sekvensdiagram

StartGame:



DSD for hvad der sker når man starter spillet, hvor det der sker i while løkken er vist i det andet DSD diagram på den næste side.



Land on Fleet:

DSD der viser hvad der sker i en tur hvor spilleren lander på et fleet felt og køber det.

Test

	j U 1	j U 2	j U 3	j U 4	j U 5	j U 6	j U 7	j U 8	j U 9	T C 1	T C 2	T C 3	T C 4	T C 5	T C 6	T C 7	T C 8	T C 9	TC 10	TC 11	TC 12
k1	x									x											
k2											x										
k3		x																			
k4					x	x	x	x	x			x									
k5													x								
k6														x							
k7															x						
k8			x																		
k9																x					
k1 0																	x				
K 1 1																		x			
K 1 2				x																	
K 1 3																			x		
K 1 4																				x	
K 1 5																					x

Formelle test cases

(Tjek bilag for billeder af de forskellige test cases)

Test Case ID	TC01
Summary	Tests that the dice rolls
Requirements	R1
Preconditions	Player rolled the dice
Postconditions	The dice rolls
Test procedure	<ol style="list-style-type: none">1. Start the game2. Roll dice3. The dice rolls
Test Data	Dice roll value eg. 6
Expected result	The system successfully rolls 6
Actual result	The system successfully rolled 6
Status	Passed
Tested by	Christopher David Carlson Chytræus
Date	24-11-2016
Date environment	Eclipse 4.6.0 for Mac OS X El Capitan

Test Case ID	TC02
Summary	The car moves to the right field according to the dice roll
Requirements	R2
Preconditions	The player rolled the dice
Postconditions	The system moved the car successfully
Test procedure	<ol style="list-style-type: none">1. Start the game2. Roll dice3. See if the system moved the car correctly
Test data	Dice roll = 11, Field 11 = Walled city
Expected result	The system moved the car to Walled city

Actual result	The system successfully moved the car to Walled city
Status	Passed
Tested by	Jonathan Yngve Friis
Date	24-11-2016
Test environment	Eclipse 4.6.0 on Windows 10

Test Case ID	TC03
Summary	The system removes or gives players the correct amount of coins according to the field the players land on and transfer coins between players
Requirements	R4
Preconditions	A player lands on a field
Postconditions	The system removed og gave the player coins according to the field
Test procedure	<ol style="list-style-type: none"> 1. Start the game 2. Play the game 3. See if the players lose the correct amount of money if they buy a property, land on an owned property or land on a Tax field. 4. See if players gain the correct amount of coins if players land on a field they own and check if a player gains the correct amount of coins if he/her lands on a refuge field 5. Repeat until all fields have been checked
Test data	Example: Player 1 coins = 30000, Player 1 lands on Caravan, Player 1 coins = 26000.
Expected result	The system removes or gives the players the correct amount of coins according to the field
Actual result	The system successfully removes or gives the players the correct amount of coins according to the field
Status	Passed
Tested by	Jonathan Yngve Friis
Date	24-11-2016
Test environment	Eclipse 4.6.0 on Windows 10

Test Case ID	TC04
Summary	The system shall give the player the choice of purchasing the field he/she lands on
Requirements	R5
Preconditions	A player rolls the dice and lands on a field
Postconditions	The player was given the choice of purchasing the field he/she landed on
Test procedure	<ol style="list-style-type: none"> 1. A player rolls the dice 2. The player lands on a field 3. See if the player is given the choice of purchasing said field
Test data	Dice roll = 2. Field = Crater
Expected result	Player is given the choice of buying the field 'Crater'
Actual result	Player is given the choice of buying the field 'Crater'
Status	Passed
Tested by	Christopher David Carlson Chytræus
Date	24-11-2016
Test environment	Eclipse 4.6.0 on Windows 10

Test Case ID	TC05
Summary	The system removes a player when he/her goes bankrupt
Requirements	R6
Preconditions	A player went bankrupt
Postconditions	The system removes the bankrupt player
Test procedure	<ol style="list-style-type: none"> 1. Start the game 2. Played the game until one player went bankrupt 3. See if the system removes the bankrupt player
Test data	Player 1 coins = 0
Expected result	The system removes the bankrupt player from the game
Actual result	The system successfully removes the bankrupt player

Status	Passed
Tested by	Jonathan Yngve Friis
Date	24-11-2016
Test environment	Eclipse 4.6.0 on Windows 10

Test Case ID	TC06
Summary	The system prints a text in the GUI about the field the player landed on
Requirements	R7
Preconditions	The player rolled the dice
Postconditions	The system prints a text accordingly to the field the player landed on
Test procedure	<ol style="list-style-type: none"> 1. Start the game 2. Roll dice 3. See if the system prints a text containing information about the field the player landed on 4. Repeat until all fields have been landed on
Test data	Dice roll = 9, Field 9 = Huts in the mountain
Expected result	The system prints a text containing information about Huts in the mountain
Actual result	The system successfully prints a text containing price for the field and the rent if the field was already owned
Status	Passed
Tested by	Jonathan Yngve Friis
Date	24-11-2016
Test environment	Eclipse 4.6.0 on Windows 10

Test Case ID	TC07
Summary	The system ends the game when all but one player is bankrupt
Requirements	R9
Preconditions	All but one player is bankrupt

Postconditions	The system stops the game when all but one player is bankrupt
Test procedure	<ol style="list-style-type: none"> 1. Asks the GUI to create 3 players 2. Plays the game until all but one player is bankrupt 3. See if the system congratulates the winning player and ends the game
Test data	Player 1 coins = 0, Player 2 coins = 0 and Player 3 coin > 0
Expected result	The system ends the game when all but one player is bankrupt
Actual result	The system successfully ends the game when all but one player is bankrupt
Status	Passed
Tested by	Jonathan Yngve Friis
Date	24-11-2016
Test environment	Eclipse 4.6.0 on Windows 10

Test Case ID	TC08
Summary	Tests that field descriptions and properties are unique for each field
Requirements	R10
Preconditions	A player rolls the dice and moves to a field
Postconditions	The GUI has fields with unique descriptions as well as properties
Test procedure	<ol style="list-style-type: none"> 1. A Player rolls the dice 2. The car moves to a field 3. The field has a unique descriptions as well as properties
Test Data	Dice roll value = 4. Field = Second Sails
Expected result	Text about the properties and description of the field
Actual result	Text about the properties and description of the field
Status	Passed
Tested by	Christopher David Carlson Chytræus

Date	24-11-2016
Date environment	Eclipse 4.6.0 for Mac OS X El Capitan

Test Case ID	TC09
Summary	The GUI creates the wanted amount of player objects
Requirements	R11
Preconditions	2-6 is written in the "How many players" field within the GUI
Postconditions	2-6 player objects are created and
Test procedure	<ol style="list-style-type: none"> 1. 2-6 players want to play the game 2. The user inserts the wished amount of players in the GUI 3. The GUI creates the wanted amount of player objects and ask for their name
Test Data	Amount of players = 3. Player objects created = 3. Amount of names asked for = 3.
Expected result	The GUI asks for the names of the wished amount of players.
Actual result	The GUI asks for the names of the wished amount of players.
Status	Passed
Tested by	Christopher David Carlson Chytræus
Date	24-11-2016
Date environment	Eclipse 4.6.0 for Mac OS X El Capitan

Test Case ID	TC10
Summary	Tests that the program works for minimum an hour
Requirements	R13
Preconditions	2-6 players playing the game

Postconditions	The game successfully runs for one hour
Test procedure	<ol style="list-style-type: none"> 1. 2-6 players start the game 2. The game is played for at least an hour
Test data	.
Expected result	The game runs successfully for one hour
Actual result	The game runs successfully for one hour
Status	Passed
Tested by	Christopher David Carlson Chytræus
Date	24-11-2016
Test environment	Eclipse 4.6.0 on Windows 10

Test Case ID	TC11
Summary	Tests whether the game successfully loads and is playable on one of the computers in the DTU data bar
Requirements	R14
Preconditions	The necessary software is installed on the computer in the data bar.
Postconditions	The game successfully runs and is playable on the computer in the data bar.
Test procedure	<ol style="list-style-type: none"> 1. The necessary software is downloaded and installed on the computer in the data bar. 2. 2-6 players play the game to the end
Test Data	
Expected result	The game successfully runs on the computer in the data bar
Actual result	The game successfully runs on the computer in the data bar
Status	Passed
Tested by	Christopher David Carlson Chytræus
Date	24-11-2016
Date environment	Eclipse 4.6.0 for Mac OS X El Capitan

Test Case ID	TC12
Summary	Tests that the result of the dice roll shows within 0.6 seconds. We start and stop a stopwatch as fast as possible and if the timer on the stopwatch is less than 0.6 seconds we can conclude that the result is shown in less than 0.6 seconds since the GUI shows the result of the dice roll relatively instantly
Requirements	R15
Preconditions	The java project is downloaded and the game works
Postconditions	The GUI successfully displayed the result of the dice roll within 0.6 seconds
Test procedure	<ol style="list-style-type: none"> 1. A player starts the game 2. The player rolls the dice 3. Start a stopwatch at the exact same time and stop the stopwatch relatively immediately after 4. Check the stopwatch to see whether the result of the dice roll shows within 0.6 seconds
Test data	Stopwatch = 0.34 sekunder.
Expected result	Result of the dice roll shows within 0.6 seconds
Actual result	Result of the dice roll shows within 0.6 seconds
Status	Passed
Tested by	Christopher David Carlson Chytræus
Date	24-11-2016
Test environment	Eclipse 4.6.0 on Windows 10

JUnit-tests

JUnit1 Krav 1:

I vores første JUnit Test, som er bygget ud fra krav 1, har vi testet 2 terningekast og lavet en for-løkke, som simulerer 1000 kast, og udregner summen for hvert kast. Testens expected value er 1000, og dets actual er 0. For hvert kast tilføjes der 1 til actual, indtil man når 1000, såfremt alle slags sum er mellem 2 og 12 (minimum- og maksimumværdierne for summen af to terninger). Da testen har været positiv, kan man konkludere, at kravet er opfyldt.

JUnit 2 Krav 3:

I denne JUnit Test, som er bygget ud fra krav 3, har vi afprøvet metoden hvor der tilføjes eller fradrages coins fra brugeren. I vores eksempel i JUnit Testen har vi sat den forventede værdi til at være 20000 coins. Derefter tilføjes og fradrages der penge nogle gange, så man ender med at få trukket 10000 coins fra sin account. Testens actual værdi sættes så til at være den aktuelle pengebeholdning efter man har ændret beløbet flere gange. Da pengebeholdningen starter med at være 30000, går regnskabet op, og testen har også været positiv, hvilket vi kan bruge til at konkludere, at også denne metode virker som ønsket.

JUnit 3 Krav 8:

I denne test, som er bygget ud fra krav 8, har vi ganske enkelt afprøvet, om man starter ud med 30000 gold coins, som vi har sat som en startbetingelse. Vores expected value er 30000 coins, mens vores actual er den værdi, som aflæses fra brugerens account. Denne test har også været positiv, så igen kan man konkludere, at den har fungeret efter ønske.

JUnit 4 Krav 12:

I denne test, som er baseret på krav 12, 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. Siden denne også har været positiv, kan vi også her konkludere, at det fungerer optimalt.

JUnit 5 Refuge:

I denne test tjekker vi at refuge felterne opfører sig som de skal. Vi tester om de giver penge til spilleren hvis spilleren lander på et refuge felt der giver penge, tester om de tager penge hvis spilleren lander på et refuge felt der tager penge og tester om spillerens balance er uændret hvis spilleren lander på et refuge felt der intet gør.

Disse tests viste at refuge felterne virker og derfor kan vi konkludere at vores refuge tests har været succesfulde.

JUnit 6 Territory:

I denne test tjekker vi at territory felterne opfører sig som de skal. Vi tester om de ikke trækker penge fra en spillers beholdning hvis spilleren vælger ikke at købe feltet, tester om de trækker penge fra en spillers beholdning hvis spilleren køber spillet og tester om en spiller overfører penge til en anden spiller hvis den første spiller lander på en anden spillers felt.

Disse tests viste at territory felterne virker og derfor kan vi konkludere at vores territory test har været succesfulde.

JUnit 7 Tax:

I denne test tjekker vi om tax felterne virker som de skal. Her bliver der først testet om der bliver trukket den rigtige mængde coins. Herefter bliver der testet om den trækker den rigtige mængde af coin når man vælger 10% af ens pengebeholdning. Disse tests viste at det fungerede og dermed kan vi konkludere at vores Tax test er succesfuld.

JUnit 8 Fleet:

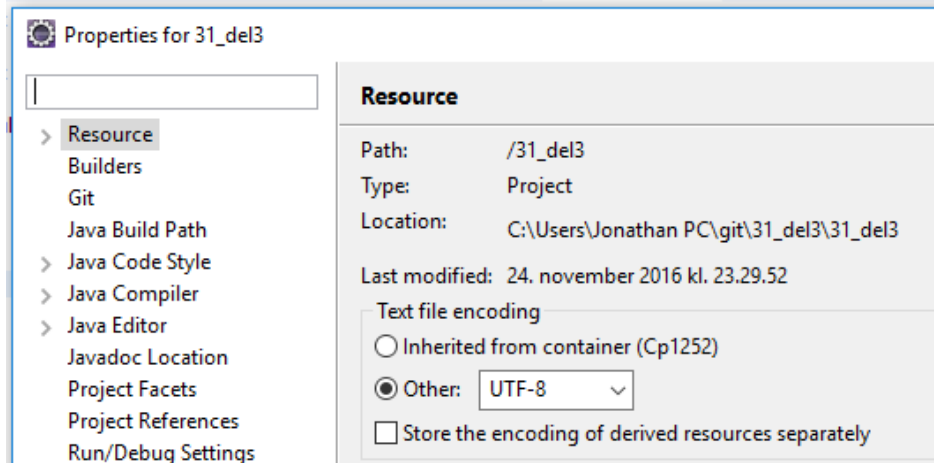
I denne JUnit Test tjekker vi om fleet-felterne virker som de skal. Vi har sat testen til at vise hvad der sker når spiller 2 lander på fleet felter, når de andre er ejet af spiller 1. Spiller 2 giver flere coins til spiller 1 afhængigt af hvilket nummer af fleet felt på spillebrættet han/hun lander på. Vi har testet alle 4 scenarier mht. hvor mange felter spiller 1 ejer, Siden vores test har været positiv, kan vi konkludere at vores fleet klasse fungerer efter kundens vision.

JUnit 9 Labor Camp:

I denne test tjekker vi at labor camp felterne opfører sig som de skal. Vi tester om de trækker penge fra en spillers beholdning hvis spilleren køber spillet og tester om en spiller overfører penge til en anden spiller hvis den første spiller lander på en anden spillers felt. Disse test viste at labor camp felterne virker og derfor kan vi konkludere at vores labor camp test har været succesfulde.

Acceptance test

Vi kan ud fra vores formelle test cases og JUnit tests se, at alle krav bortset fra krav 16, 17 og 18 er opfyldt. De sidste tre krav er implementeringskrav og kan som sådan ikke testes af os. Her er dog vedlagt et par billeder, som bevis på, at kravene er opfyldt:

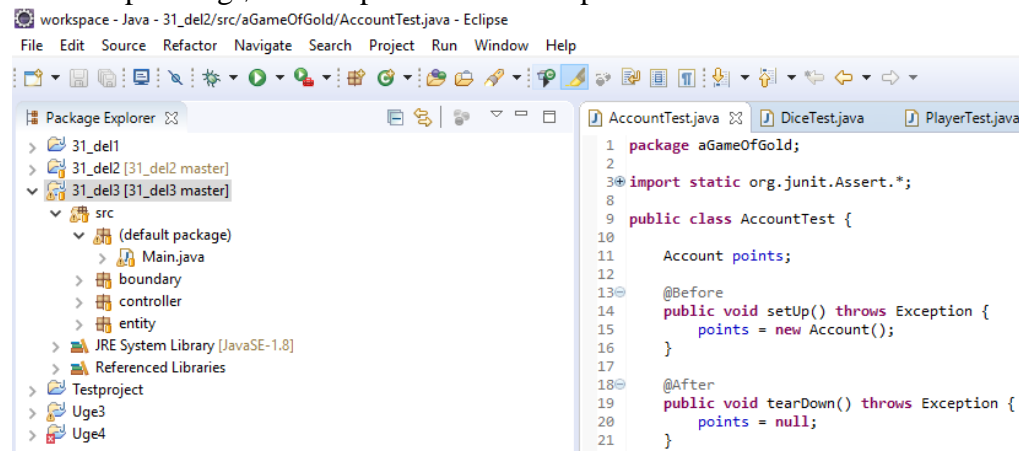


Konfiguration:

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_del3
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 URL”
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 flueben ud for sig eller klik i den tomme kasse
15. Tryk “finish”
16. Åben projektet i eclipse
17. Åben “src”
18. Åben pakken “(Default Package)”
19. Åben Main.java
20. Klik på den grønne knap med den hvide pil i.



Konklusion

Overordnet konklusion

Ud fra vores projekt kan vi konkludere at produktet vi har produceret gennem Eclipse opfylder kravspecifikationerne i overensstemmelse med kunden og dermed bekræfter vores hypotese.

Produktorienteret konklusion

Vi har brugt viden fra vores CDIO-1 og CDIO-2. Vores usecases og diagrammer har givet overblik og gjort arbejdsprocessen fra design til kode overskueligt. Vha. GitHub har vi det gjort det let at koordinere arbejdsopgaverne kodemæssigt. Ud fra JUnit og Testcases kan vi konkludere at spillet fungerer som det skal. Derfor kan vi konkludere, at produktet er blevet udviklet efter kundens ønske, da det opfylder de krav, som der er blevet stillet af kunden.

Forslag til videre arbejde og forbedringer

- Flere typer af felter, som gør brætspillet mere varieret.

Litteraturliste

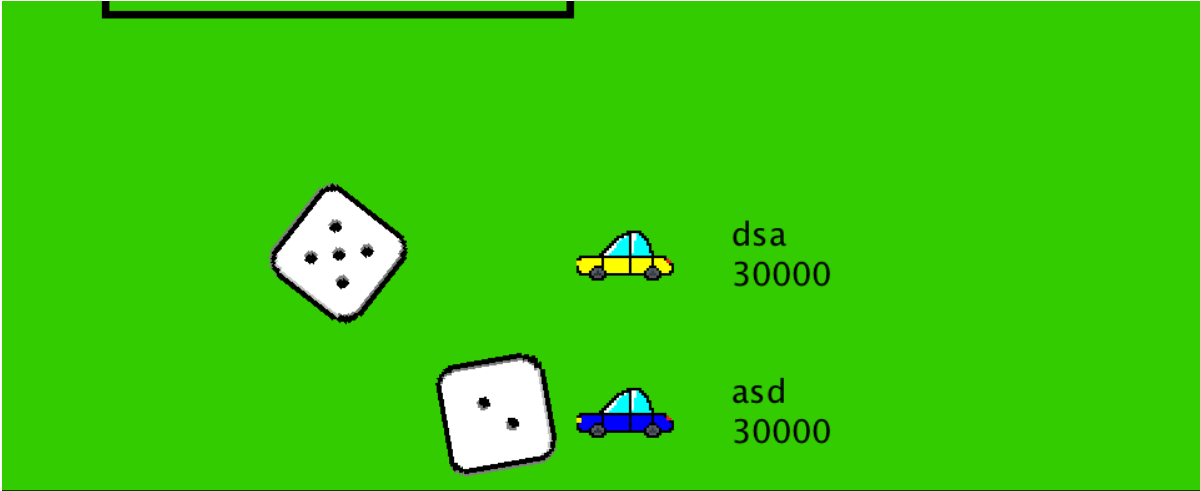

- CDIO-2 Aflevering
- Larman, Craig: "Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development", tredje udgave, Addison Wesley Professional, Oktober 20 2004.
- <https://www.campusnet.dtu.dk/cnnet/filessharing/SADownload.aspx?FileId=4298361&FolderId=1026792&ElementId=522003> - 02313, Udviklingsmetoder til IT-systemer. Lektion 06: Use case realisering, design. Forfatter: Ian Bridgwood, Henrik Bechmann - Afdeling for informatik, DTU Diplom. Sidst besøgt: 17-11-16
- <https://www.campusnet.dtu.dk/cnnet/filessharing/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: 17-11-16
- <https://www.campusnet.dtu.dk/cnnet/filessharing/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: 17-11-16
- <https://www.campusnet.dtu.dk/cnnet/filessharing/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: 17-11-16

Bilag

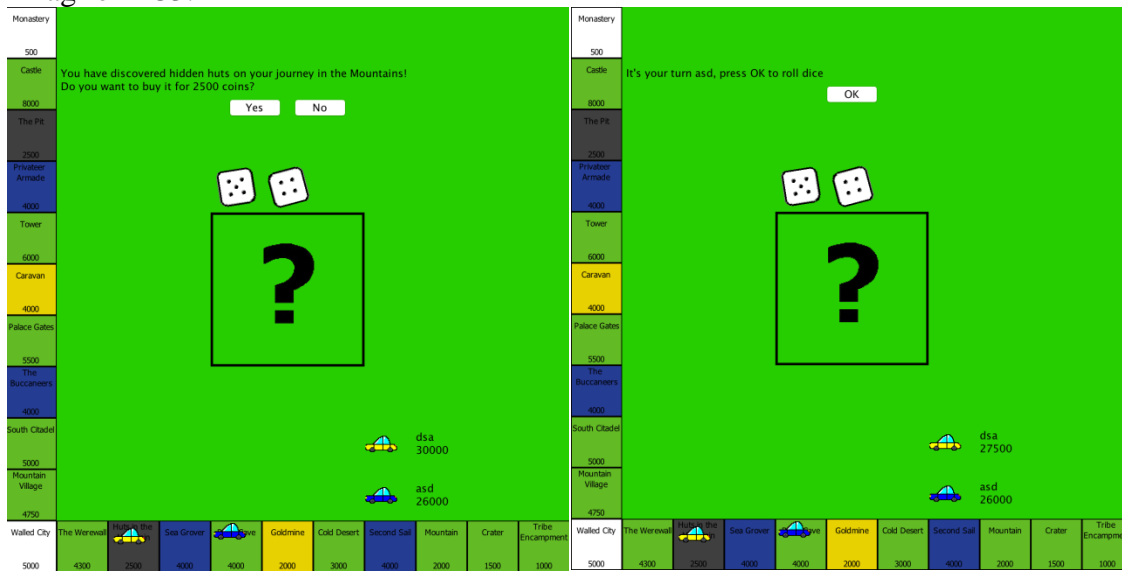
Bilag til beskrivelse af felter:

1. Tribe Encampment	Territory	Rent 100	Price 1000
2. Crater	Territory	Rent 300	Price 1500
3. Mountain	Territory	Rent 500	Price 2000
4. Cold Desert	Territory	Rent 700	Price 3000
5. Black cave	Territory	Rent 1000	Price 4000
6. The Werewall	Territory	Rent 1300	Price 4300
7. Mountain village	Territory	Rent 1600	Price 4750
8. South Citadel	Territory	Rent 2000	Price 5000
9. Palace gates	Territory	Rent 2600	Price 5500
10. Tower	Territory	Rent 3200	Price 6000
11. Castle	Territory	Rent 4000	Price 8000
12. Walled city	Refuge	Receive 5000	
13. Monastery	Refuge	Receive 500	
14. Huts in the mountain	Labor camp	Pay 100 x dice	Price 2500
15. The pit	Labor camp	Pay 100 x dice	Price 2500
16. Goldmine	Tax	Pay 2000	
17. Caravan	Tax	Pay 4000 or 10% of total assets	
18. Second Sail	Fleet	Pay 500-4000	Price 4000
19. Sea Grover	Fleet	Pay 500-4000	Price 4000
20. The Buccaneers	Fleet	Pay 500-4000	Price 4000
21. Privateer armade	Fleet	Pay 500-4000	Price 4000

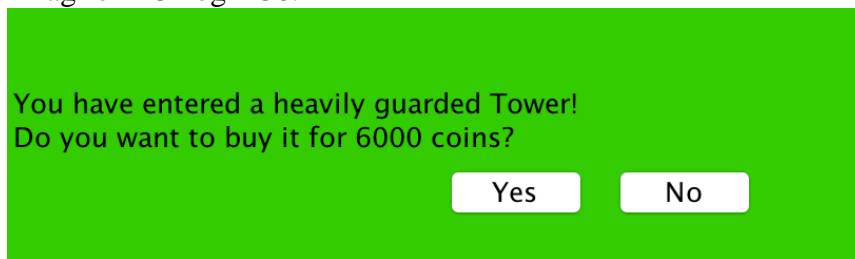
Bilag for TC1 og TC2:

							
Grover		Goldmine	Cold Desert	Second Sail	Mountain	Crater	Tribe Encampment
4000	4000	2000	3000	4000	2000	1500	1000

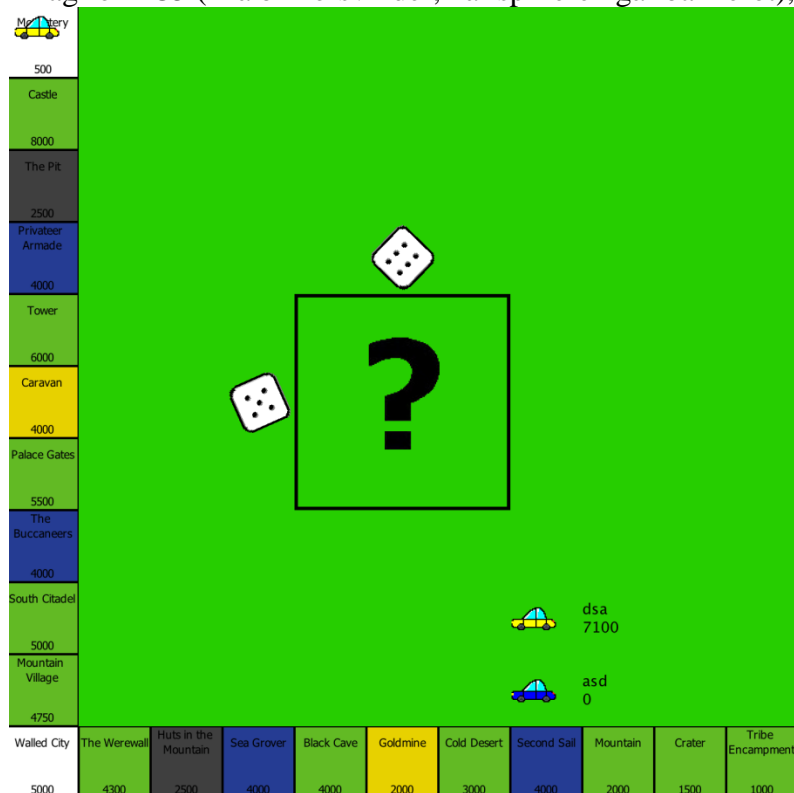
Bilag for TC3:



Bilag for TC4 og TC6:



Bilag for TC5 (Blå bil forsvinder, når spilleren går bankerot), TC7 (spillet er stoppet) og TC8:



Bilag for TC9:

Enter player amount between 2 and 6:

