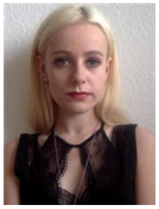
CDIO final

Emily Skovgaard Rasmussen, s153374
Mathias Tværmose Gleerup, s153120,
Sofie Freja Christensen s153932
Morten V. Christensen s147300
Simon Lundorf s154008
Jonas Larsen, s136335,
Group nr. 15 Classes: 02312

Deadline : January 16th 2017

January 3, 2017

Danmarks Tekniske Universitet



Contents

1 Hour registration

Table 1: Hour registration

	Emily	Jonas	Mathias	Morten	Simon	Sofie	Sum
Conceive	0	0	0	0	0	0	0
Design	13	13	13	13	7	13	72
Impl.	0	0	0	0	0	0	0
Test	0	0	0	0	0	0	0
Dok.	0	0	0	0	0	0	0
Andet.	0	0	0	0	0	0	0
Sum	13	13	13	13	7	13	72

2 Formalities

2.1 Abstract

2.2 Foreword

2.3 Git-strategy

In this project, the parallell configuration is expected to be handled as a combination of two types of patterns. These are the ”Branch per change” and ”Branch per environment” (These are seen in the appendix). This is used in a way, where there are three environments: The master, Change and Release. The change environment however, is furthermore branched. This meaning, a new branch for every new feature being created and implemented.

2.4 Risk analysis

- Understaffed: downgrading of the red and yellow points, and longer days for the remaining in the group.
- Not enough time during M2: A solution could be downgrading the chancecards, and make them parking spaces.
- Underestimating the emphazis of the project: A solution could downgrading the chancecards and pawning.
- Complications with the GUI: This is not downgraded, therefore chancecards and pawning is downgraded.
- The merging between the patterns (branch per release and branch per environment).

2.5 Procedure

Release 1

- First of all the Board, the players and the cars are ensured to work. This makes a game that can be played, and the features can aftwerwards be added.

Release 2

- Bankrupt
- Buy squares
- Taxes

Release 3

- Buy and sell hotels
- Double rent

- Tax (Determined price)
- Parking (Important, if chancecards and prison are not implemented, this is the replacement.

Release 4

- Startbonus
- Prison
- Effect with to equal die

Release 5

- Chancecard
- Pawn (+sale of houses and hotels to the bank) or sale to the bank
- Tax (10 %)

Release 6

- Inside trade
- Buy houses gradually

3 Systemrequirements

Functional

1. The game has to be played by 3-6 players.
2. The game shall use two die
3. The game has to be executable on the machines in DTU's databars.
4. The players are to take turns to roll the die.
   (a) If a player rolls two equal eyes, that player gets another turn.
   (b) If a player rolls doubles three times in a row, that player goes to jail.
5. The player has to have an account balance.
   (a) The balance starts at 30.000 kr.
6. The game must contain 8 types of squares.
   (a) *Street* - When a player lands on this square, owned by another player. The player has to pay the owner rent.
   (b) *Start* - The player is awarded a bonus of 4000 whenever, a player passes this square.
   (c) *Tax* - The player has to pay either a fixed amount, or 10 % of their fortune. This depends on which tax square, the player lands on.
   (d) *Brewery* - The player has to roll the die, 100x this amount is what the player has to pay the owner. If the owner of the labor camp, owns more than one, the amount is multiplied by the amount of breweries.
   (e) *Shipping* - The player has to pay the owner. Price determined by number of ships, as such:
      i. ship: 500.
      ii. ships: 1000.
      iii. ships: 2000.
      iv. ships: 4000.
   (f) *Parking* - Free zone, nothing happens.
   (g) *Jail* - The player can either choose to pay a fee of 1000 kr. or try to roll equal die. If the player cannot roll equal die, within 3 turns, the player automatically gets thrown out of jail, and has to pay the fee.

    (h) *Chance* - The player picks a chance card.

7. The game has to be played on a board with 40 squares developed from the former game CDIO3.
    (a) A table overview of the squares is listed in the appendix.
8. When a player owns all the streets of a certain color, this player can buy a house at the beginning of their turn
    (a) When 4 houses are owned on one lot, a hotel can be bought.
    (b) For every house and/or hotel built, the rent is increased.
9. Each players position has to be saved.
    (a) When the players get another turn, they continue from their latest position.
    (b) The player saves, how many rounds the player has been in jail.
10. The game is won, when only one player is left in the game.
    (a) A player loses the game, when they have gone bankrupt (account at 0 £ or under).
    (b) The other players continue to play.

Non functional

1. There has to be a minimum system requirements for the game.
    (a) This guide has to include a description of how to import the code from a Git repository.
2. A text has to be displayed, describing the effects of the square a player lands on.
3. The game must display a board.
    (a) This will be achieved by using a Graphical User Interface (GUI).

3.1 Noun and verb analysis

The customer want the game ”Matador” to be development, due to time constraints, this will be a simplified version of the game. Extracted from the rules of the game, a requirement specification has been formed. From this a noun and verb analysis is created.

3.1.1 Noun analysis

A noun analysis has been included, to develop and retrieve an overview of prospectable classes.

Game
Players
Die
Account balance
Squares
Street
Start
Tax
Brewery
Shipping
Parking
Jail
Chance
Hotel
House
Rent
Board

3.1.2 Verb analysis

A verb analysis has been included, to develop the expected methods.

Roll
Lands
Owned
Bankrupt
Buys

These are the key methods. However a lot more is to be generated.

3.2 Use case descriptions

Use case: Build hotel on street.
ID: 1
Brief description: The process of building a hotel on an owned street.
Primary actors: 1. Player.
Secondary actors: None.
Preconditions: It's the start of the players turn (Pre-rolling) Player wants to buy a hotel
Main flow: 1. Player clicks the buy hotel button. 2. The game checks if the player owns all the streets of the same type(color?) 3. The game checks if the player already owns four houses on the street. 4. The game checks if the player has enough money. 5. The four houses are removed and replaced by a hotel.
Postconditions: The game registers that the player now owns a hotel on the street.
Alternative flows: 1. The player does not meet one or more of the requirements. A message is displayed telling what requirement(s) was not met.

Table 2: Build hotel on street.
Describes the requirements a player must fulfill to be able to buy a hotel on a street. It also describes what we want our system to do, in case the player does not meet one or more of the requirements.

Use case: Owing money.
ID: 2
Brief description: What happens when a player does not have enough cash to pay what they owe.
Primary actors: 1. Player.
Secondary actors: None.
Preconditions: The player has to pay more cash than they current hold.
Main flow: 1. The game presents the player with a list of their assets (Houses, hotels and streets). 2. The player sells houses. 3. The player pawns property. 4. the player pays what he owes.
Postconditions: The player's turn is over. The pawned property does not collect rent. The pawned property is only rebuyable by the player that pawned it.
Alternative flows: If the player can not raise enough money: 1. The game presents the player with a list of their assets (houses, hotels & streets) 2. The Player sells houses. 3. The Player pawns property. 4. The Player still does not have enough money to pay debt (bankrupt) 5. The Player is removed from the game. 6. All of the eliminated players pawned properties are returned to the bank.

Table 3: Owing money.

Describes what a player must go through when in deeper debt than what his current cash amount can pay off. The player will get the option to sell houses for half the original building price. If the player does not raise enough money to pay his debt from selling houses/hotels, he may be forced to pawn property. Pawning a property removes the rights to collect rent, but will only be re-buyable by the player who pawned it. If the player is not able to raise enough money to pay his debt, the player goes bankrupt and is removed from the game. The amount the player was worth at the time of bankruptcy is collected by the Player that caused it.

Use case: Build hotel on street.
ID: 3
Brief description: The process of building a hotel on an owned street.
Primary actors: 1. Player.
Secondary actors: None.
Preconditions: It’s the start of the players turn (Pre-rolling) Player wants to buy a hotel
Main flow: 1. Player clicks the buy hotel button. 2. The game checks if the player owns all the streets of the same type(color?) 3. The game checks if the player already owns four houses on the street. 4. The game checks if the player has enough money. 5. The four houses are removed and replaced by a hotel.
Postconditions: The game registers that the player now owns a hotel on the street.
Alternative flows: 1. The player does not meet one or more of the requirements. A message is displayed telling what requirement(s) was not met.

Table 4: game

Use case: Build hotel on street.
ID: 4
Brief description: The process of building a hotel on an owned street.
Primary actors: 1. Player.
Secondary actors: None.
Preconditions: It's the start of the players turn (Pre-rolling) Player wants to buy a hotel
Main flow: 1. Player clicks the buy hotel button. 2. The game checks if the player owns all the streets of the same type(color?) 3. The game checks if the player already owns four houses on the street. 4. The game checks if the player has enough money. 5. The four houses are removed and replaced by a hotel.
Postconditions: The game registers that the player now owns a hotel on the street.
Alternative flows: 1. The player does not meet one or more of the requirements. A message is displayed telling what requirement(s) was not met.

Table 5: Game.

This use case description is an overview of our games flow. Its outlook is of a complete game sequence, from entering your name to winning or loosing. Since the use case is meant as an overview, the different actions of landing on a specific square is not included, but those will be fully described in seperate use cases.

3.3 Use case diagram

3.4 Domain model

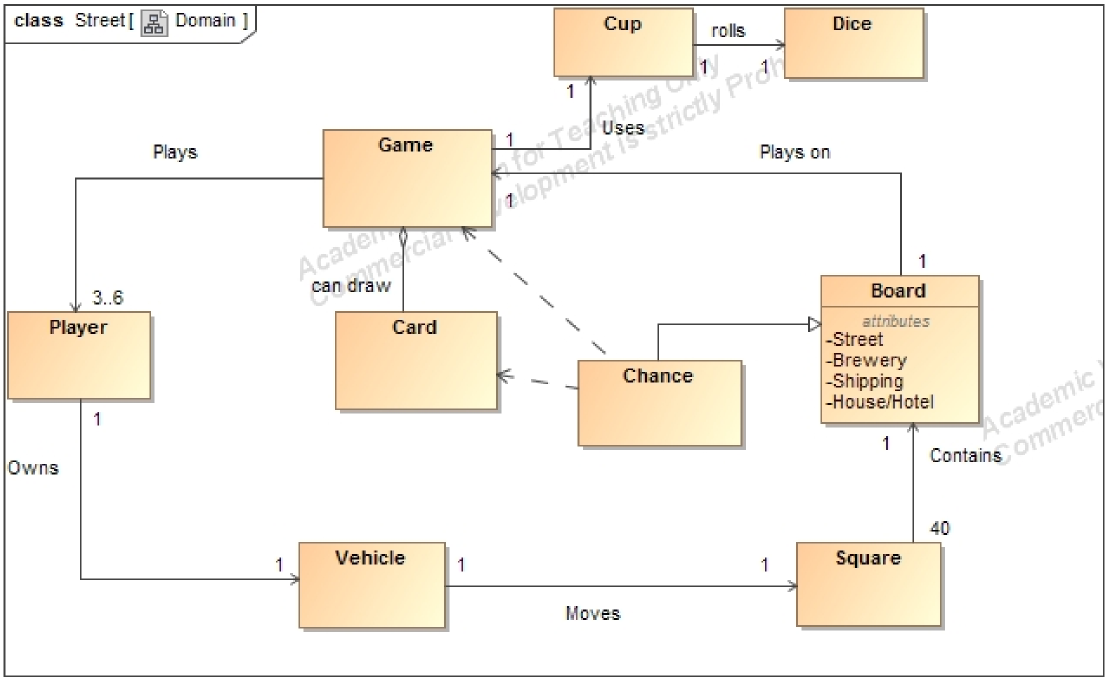


Figure 1: Domain model

Drawn from the prospectable classes, found in the Noun analysis, the following, has been picked out to be potential classes:

Game
Player
Card
Vehicle
Square
Board
Chance
Cup
Dice

These represent the basics of the game. Each player has an account balance, therefore the Account class is necessary. Furthermore for the visuals of the board, it is essential for the player to have a vehicle. This to see where the player lands, and where to move from next turn.
The square class, will contain different types of squares. Using polymorfism and creating classes for each type, thereby makes a inheritance hierarchy. This to encapsulate all the ownable squares and the needed methods for these.

4 Design

4.1 Design Class Diagram (DCD)

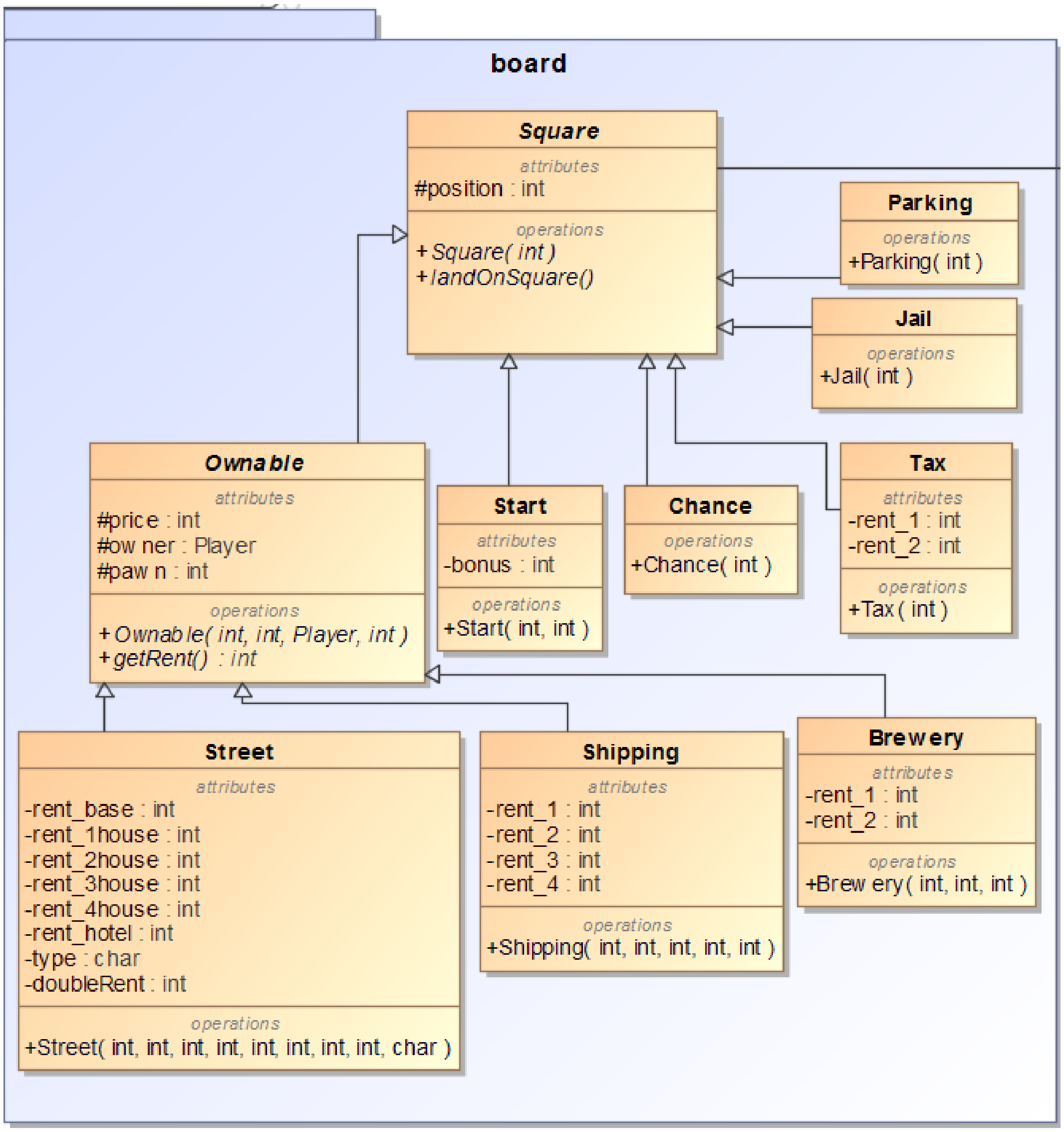


Figure 2: Pakken 'board'

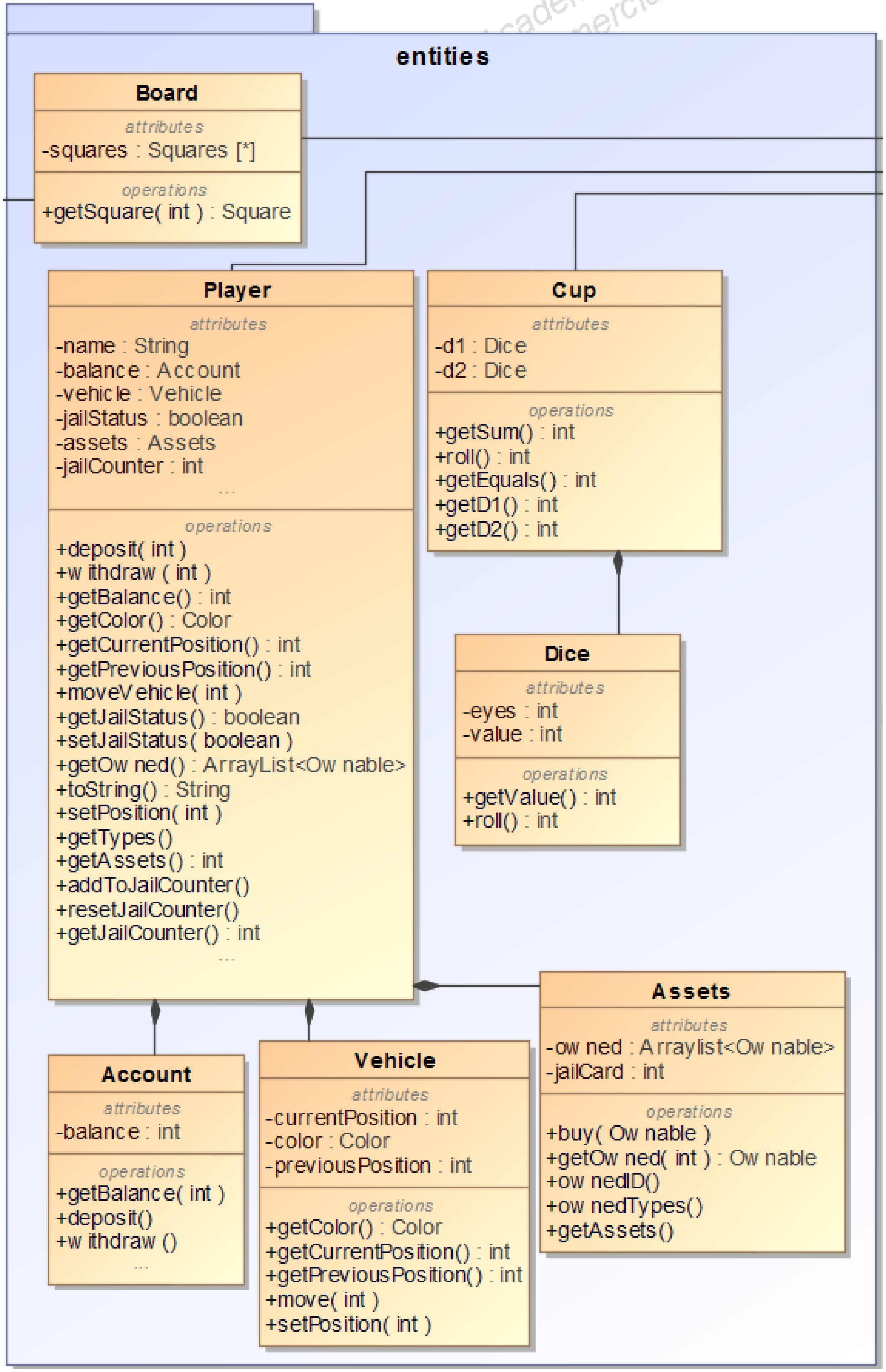


Figure 3: Pakken 'entities'

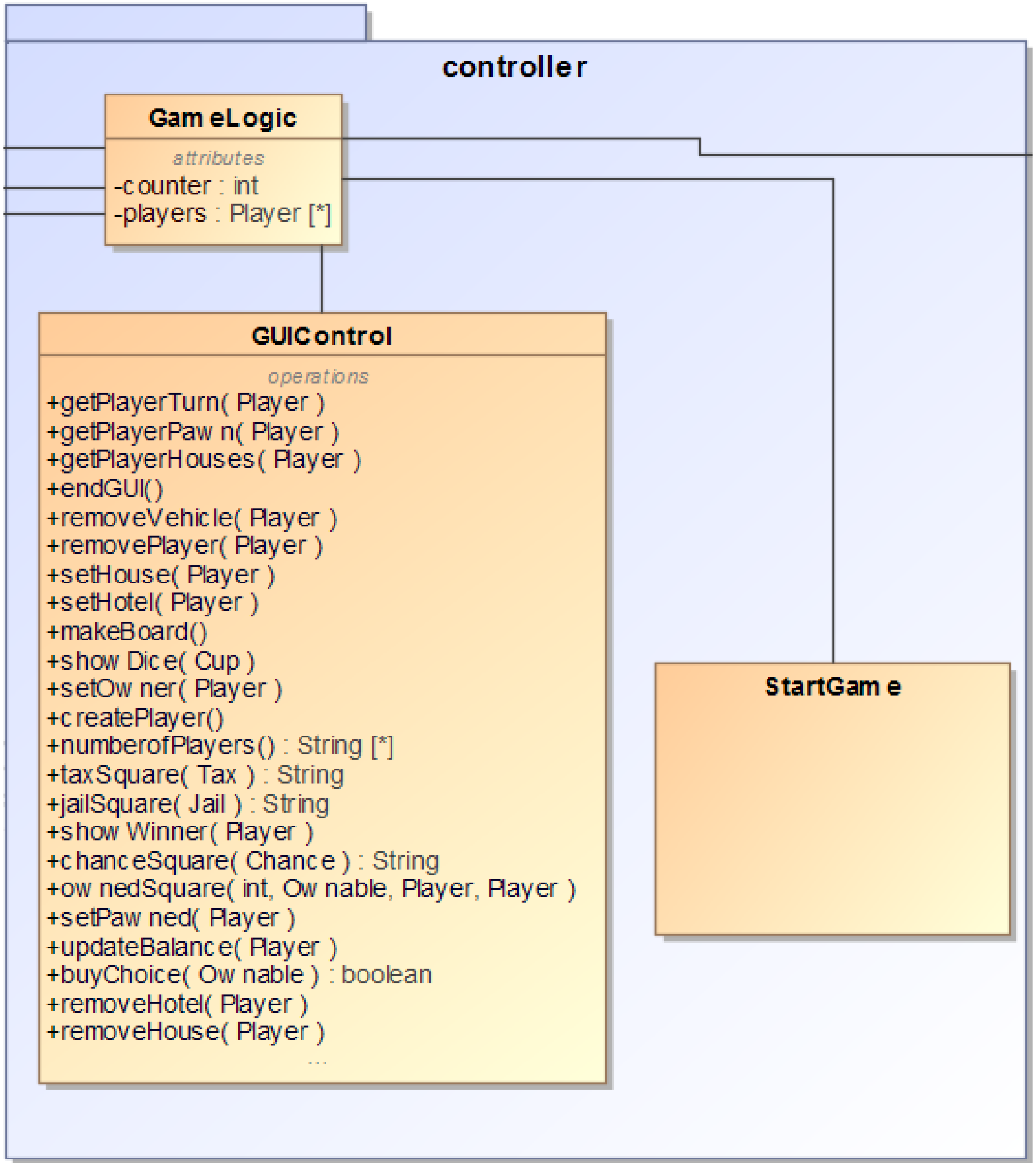


Figure 4: Klassen 'controller'

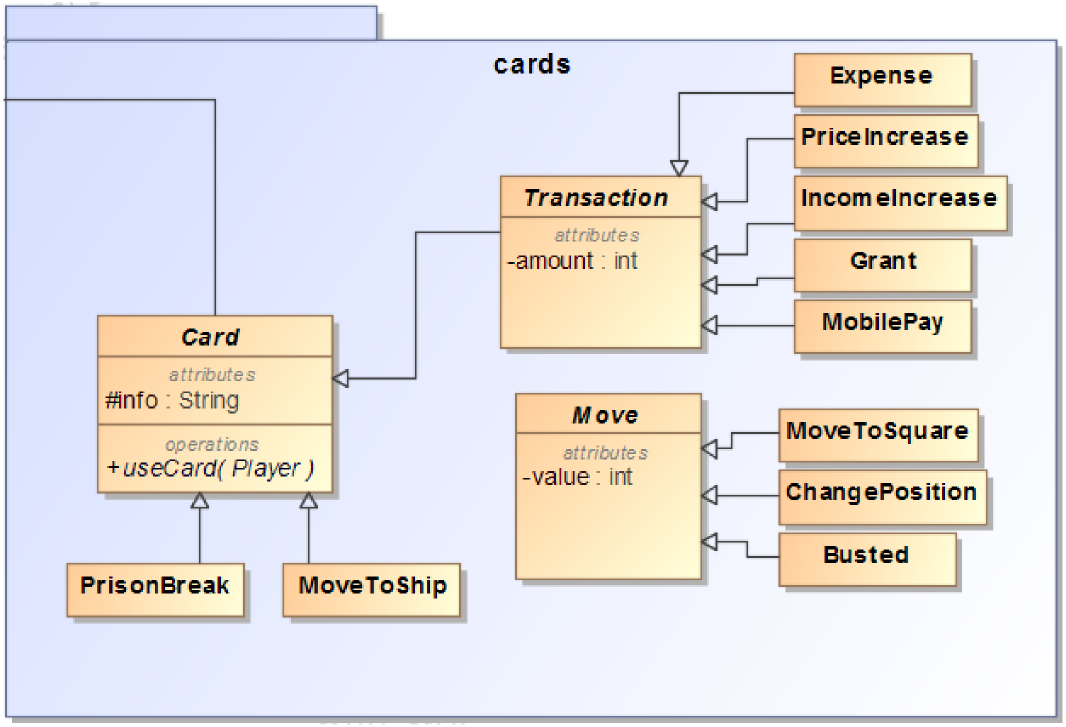


Figure 5: Klassen 'cards'

4.1.1 Centrale variable

I klassen Player har vi en variabel jailCounter, der tæller hvor mange runder den spiller har siddet i fængsel. Denne variabel bliver så tilgået af metoderne addToJailCounter, reset JailCounter og getJailCounter.
I klassen Asset er der en ArrayList over 'Ownables'. Denne ArrayListe holder styr på de grunde som spilleren ejer. Klassen Assets har også en variabel jailCard, der er en integer. Denne varibel tæller hvor mange "Get out of jail for free card" som spilleren har (man kan godt have flere).

4.2 Design Sequence Diagram (DSD)

5 Implementation

6 Test

6.1 Conclusion

7 Conclusion

8 Appendix

Table 6: List of features

Feltnavn	Pris	Leje	m/1 hus	2 huse	3 huse	4 huse	hotel	Hus pr	Hotel pr	Pant	udgifter
Start											+4.000
Rødovrevej	1200	50	250	750	2250	4000	6000	1000	1000	600	
Prøv lykken											
Hvidovrevej	1200	50	250	750	2250	4000	6000	1000	1000	600	
Indkomstskat											-2000
Scandlines H-H	4000	500	1000	2000	4000					2000	
Roskildevej	2000	100	600	1800	5400	8000	11000	1000	1000	1000	
Prøv lykken											
Valby Langgade	2000	100	600	1800	5400	8000	11000	1000	1000	1000	
Allégade	2400	150	800	2000	6000	9000	12000	1000	1000	1200	
Fængsel - besøg											
Frederiksberg Allé	2800	200	1000	3000	9000	12500	15000	2000	2000	1400	
Tuborg Squash	3000	x100	x200							1500	
Bülowsvej	2800	200	1000	3000	9000	12500	15000	2000	2000	1400	
Gl. Kongevej	3200	250	1250	3750	10000	14000	18000	2000	2000	1600	
Mols-Linien	4000	500	1000	2000	4000					2000	
Bernstorffsvej	3600	300	1400	4000	11000	15000	19000	2000	2000	1800	
Prøv lykken											
Hellerupvej	3600	300	1400	4000	11000	15000	19000	2000	2000	1800	
Strandvejen	4000	350	1600	4400	12000	16000	20000	2000	2000	2000	
Parkering											
Trianglen	4400	350	1800	5000	14000	17500	21000	3000	3000	2200	
Prøv lykken											
Østerbrogade	4400	350	1800	5000	14000	17500	21000	3000	3000	2200	
Grønningen	4800	400	2000	6000	15000	18500	22000	3000	3000	2400	
Scandlines G-R	4000	500	1000	2000	4000					2000	
Bredgade	5200	450	2200	6600	16000	19500	23000	3000	3000	2600	
Kgs. Nytorv	5200	450	2200	6600	16000	19500	23000	3000	3000	2600	
Coca Cola	3000	x100	x200							1500	
Østergade	5600	500	2400	7200	17000	20500	24000	3000	3000	2800	
Fængslet											
Amagertorv	6000	550	2600	7800	18000	22000	25000	4000	4000	3000	
Vimmelskaftet	6000	550	2600	7800	18000	22000	25000	4000	4000	3000	
Prøv lykken											
Nygade	6400	600	3000	9000	20000	24000	28000	4000	4000	3200	
Scandlines R-P	4000	500	1000	2000	4000					2000	
Prøv lykken											
Frederiksberggade	7000	700	3500	10000	22000	26000	30000	4000	4000	3500	
Ekstraordinær statsskat											-2000
Rådhuspladsen	8000	1000	4000	12000	28000	34000	40000	4000	4000	4000	

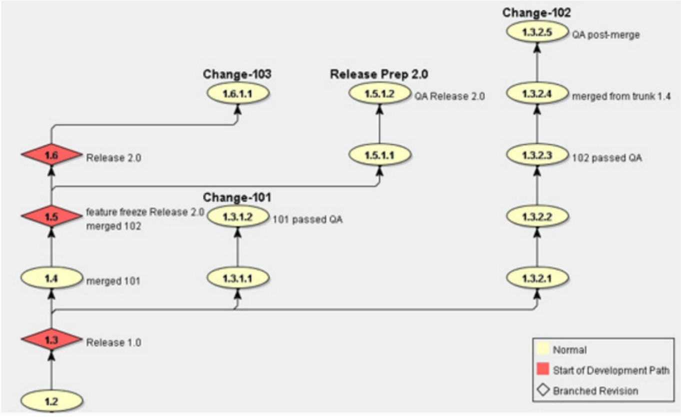


Figure 6: Branch per change

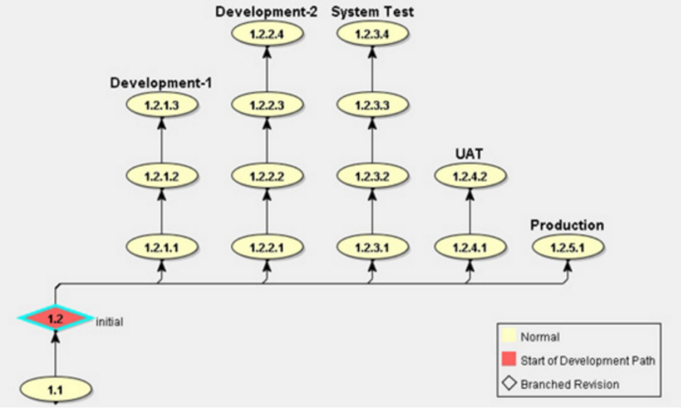


Figure 7: Brnach per environment

9 Litterature

References

[1] Craig Larman, Applying UML and Patterns 2004.

[2] Lewis and Loftus Java Software solutions 7th ed.