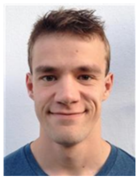
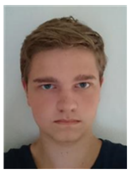
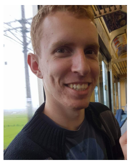
# CDIO Final

02312-14 Introductory programming, 02313 Development Methods for IT-Systems & 02315 Version Control and Test Methods.

Project title: CDIO Final
Group number: 17
Deadline: Monday 16th January 2017 - 12:00
Version: 1.0

This report contains 49 pages, including this page.

| Student nr., Surname, First name | Signature |
|---|---|
| S165248, Gadegaard, Theis  | ___________________ |
| S165232, Helstrup, Freya | ___________________ |
| S165208, Højsgaard, Tobias  | ___________________ |
| S165209, Jønsson, Danny [redacted] | ___________________ |
| S165227, Petersen, Gustav Hammershøi  | ___________________ |
| S165237, Poulsen, Joakim Thorum [redacted] | ___________________ |

## Work distribution

| Date | Participant | Analysis | Design | Impl. | Test | Doc. | Other | Total hours |
|---|---|---|---|---|---|---|---|---|
| 2017/01/15 | Danny | 9 | 5 | 10,5 | | 7 | 1,75 | **33,25** |
| 2017/01/15 | Freya | 11 | 3 | 16,5 | 6 | 9,5 | 2 | **48** |
| 2017/01/15 | Gustav | 10 | 7 | 6 | 5 | 8 | 2,4 | **38,4** |
| 2017/01/15 | Joakim | 7 | 2 | 1,5 | | 6,5 | 1 | **18** |
| 2017/01/15 | Theis | 8 | 2 | 17 | 4 | 2 | 4,4 | **37,4** |
| 2017/01/15 | Tobias | 4 | 6,5 | 6 | 16 | 4 | 2,4 | **38,9** |
| | **Sum** | **49** | **25,5** | **57,5** | **31** | **37** | **13,95** | **213,95** |

## Resumé

The objective of this project is to create a software product in the form of a danish “Matador” board game, based on our own observations of the physical version. This report’s purpose is to provide the documentation necessary to understand the thought process behind the software’s design and how it’s been implemented.

# Table of contents

# 1 Inception

## 1.1 Vision (Danish)

Lav et Matadorspil. Der tages udgangspunkt i et normalt matadorspil.
Kunden vil hellere have, at lidt er implementeret og alt virker, end at alt er forsøgt implementeret og ingenting virker.
Med udgangspunkt i 13 ugers delopgaverne ønskes der udarbejdet et ”færdigt” program til simulering af et Matador spil. Der ønskes udviklet en dansk udgave af spillet, dvs. reglerne følger de regler der gælder for et ”dansk matadorspil”. Man kan overveje at designe programmet således at der senere kan ændres til andre sprog.

# 2 Analysis

## 2.1 Requirements specification

### 2.1.1 Functional requirements

| | |
|---|---|
| 1.1 | The game shall have three to six players. |
| 1.2 | Player 1 shall always start. |
| 1.3 | The game shall consist of a board with 40 fields, numbered from 1 to 40. |
| 1.4 | The players shall start with a balance of 1500. |
| 1.5 | The players shall, in turns, throw two dice. |
| 1.6 | The player shall have a piece on the board, which will be moved around on the board. |
| 1.7 | The player shall move his piece the same number of fields as the sum of the dice thrown. |
| 1.8 | The game shall end after all players, except one, have lost all their assets. |
| 1.9 | The last remaining player, after all other players have lost all their assets, shall be the winner. |
| 1.10 | The board shall consist of the following fields having the specified type and color-group, if any: |

| ID | Field | Type | Color group |
|---|---|---|---|
| 1 | Start | Refuge | |
| 2 | Rødovrevej | Street | Cyan |
| 3 | Chance | Chance | |
| 4 | Hvidovre | Street | Cyan |
| 5 | Betal Indkomstskat | Tax | |
| 6 | Øresund A/S | Fleet | |
| 7 | Roskildevej | Street | Pink |
| 8 | Chance | Chance | |
| 9 | Valby Langgade | Street | Pink |
| 10 | Allégade | Street | Pink |
| 11 | Fængsel | Refuge | |
| 12 | Frederiksberg Allé | Street | Green |
| 13 | Tuborg | Brewery | |
| 14 | Bülowsvej | Street | Green |

| | | | |
|---|---|---|---|
| 15 | Gl. Kongevej | Street | Green |
| 16 | D.F.D.S. | Fleet | |
| 17 | Bernstorffsvej | Street | Gray |
| 18 | Chance | Chance | |
| 19 | Hellerupvej | Street | Gray |
| 20 | Strandvej | Street | Gray |
| 21 | Helle | Refuge | |
| 22 | Trianglen | Street | Red |
| 23 | Chance | Chance | |
| 24 | Østerbrogade | Street | Red |
| 25 | Grønningen | Street | Red |
| 26 | Ø.K. | Fleet | |
| 27 | Bredgade | Street | Magenta |
| 28 | Kgs. Nytorv | Street | Magenta |
| 29 | Carlsberg | Brewery | |
| 30 | Østergade | Street | Magenta |
| 31 | Gå til fængsel | Go to prison | |
| 32 | Amagertorv | Street | Yellow |
| 33 | Vimmelskaftet | Street | Yellow |
| 34 | Chance | Chance | |
| 35 | Nygade | Street | Yellow |
| 36 | D/S Bornholm | Fleet | |
| 37 | Chance | Chance | |
| 38 | Frederiksberggade | Street | Orange |
| 39 | Ekstraordinær Statsskat | Tax | |
| 40 | Rådhuspladsen | Street | Orange |

1.11A The player shall be able to buy one of the following fields for the price listed, if the player lands on the field and it is not already owned by another player:

- Rødovrevej, Hvidovre: 60
- Roskildevej, Valby Langgade: 100
- Allégade: 120
- Frederiksberg Allé, Bülowsvej: 140
- Gl. Kongevej: 160
- Bernstorffsvej. Hellerupvej: 180
- Strandvej: 200
- Trianglen, Østerbrogade: 220
- Grønningen: 240
- Bredgade, Kgs. Nytorv: 260
- Østergade: 280
- Amagertorv, Vimmelskaftet: 300
- Nygade: 320
- Frederiksberggade: 350
- Rådhuspladsen: 400
- Øresund A/S, D.F.D.S., Ø.K., D/S Bornholm: 200
- Carlsberg, Tuborg: 150

1.11B If the player refuses to buy the field the field may be set up for auction between all the players where the highest bidder will gain ownership of the field.

| | |
|---|---|
| 1.12A | The player shall pay an amount, as specified below, to the owner, if he lands on a field of the type street and the field is owned by another player: |

| Street name | 0 houses | 1 houses | 2 houses | 3 houses | 4 houses | Hotel |
|---|---|---|---|---|---|---|
| Rødovrevej | 2 | 10 | 30 | 90 | 160 | 250 |
| Hvidovre | 4 | 20 | 60 | 180 | 320 | 540 |
| Roskildevej | 6 | 30 | 90 | 270 | 400 | 550 |
| Valby Langgade | 6 | 30 | 90 | 270 | 400 | 550 |
| Allégade | 8 | 40 | 100 | 300 | 450 | 600 |
| Frederiksberg Allé | 10 | 50 | 150 | 450 | 625 | 750 |
| Bülowsvej | 10 | 50 | 150 | 450 | 625 | 750 |
| Gl. Kongevej | 12 | 60 | 180 | 500 | 700 | 900 |
| Bernstorffsvej | 14 | 70 | 200 | 550 | 750 | 950 |
| Hellerupvej | 14 | 70 | 200 | 550 | 750 | 950 |
| Strandvej | 16 | 80 | 220 | 600 | 800 | 1000 |
| Trianglen | 18 | 90 | 250 | 700 | 875 | 1050 |
| Østerbrogade | 18 | 90 | 250 | 700 | 875 | 1050 |
| Grønningen | 20 | 100 | 300 | 750 | 925 | 1100 |
| Bredgade | 22 | 110 | 330 | 800 | 975 | 1150 |
| Kgs. Nytorv | 22 | 110 | 330 | 800 | 975 | 1150 |
| Østergade | 22 | 120 | 360 | 850 | 1025 | 1200 |
| Amagertorv | 26 | 130 | 390 | 900 | 1100 | 1275 |
| Vimmelskaftet | 26 | 130 | 390 | 900 | 1100 | 1275 |
| Nygade | 28 | 150 | 450 | 1000 | 1200 | 1400 |
| Frederiksberggade | 35 | 175 | 500 | 1100 | 1300 | 1500 |
| Rådhuspladsen | 50 | 200 | 600 | 1400 | 1700 | 2000 |

| | |
|---|---|
| 1.12B | If the rent-collecting player, owns all the streets in a color-group, the rent shall be doubled. This effect is removed, if the field has houses. |
| 1.13 | The player shall pay the owner the sum of the dice thrown multiplied by 4, if he |

lands on a field of the type Brewery and the field is owned by another player.
If the owner owns both breweries, the sum shall be multiplied by 10.

1.14 The player shall pay a specified amount to the owner if he lands on a field of the type fleet owned by another player. The amount is based on how many fleets that the owner owns:

- The owner owns 1 fleet, the player pays 25 to the owner
- The owner owns 2 fleets, the player pays 50 to the owner
- The owner owns 3 fleets, the player pays 100 to the owner
- The owner owns 4 fleets, the player pays 200 to the owner

1.15 The player shall pay an amount, as listed below, if he lands on a field of the type tax:

- Indkomstskat: 200 or 10% of all assets*
- Ekstraordinær statsskat: 100

*) The player shall choose between the set amount and the percent (10% of the price of the player's owned fields + 10% of the player's balance).

1.16 If a player owns a field they shall be able to pledge it for a short term cash grab. If the player wants the field back they need to buy it back with added fees

| Field name | Pledge-Values | Raise-Pledges |
|---|---|---|
| Rødovrevej | 30 | 33 |
| Hvidovre | 30 | 33 |
| Roskildevej | 50 | 55 |
| Valby Langgade | 50 | 55 |
| Allégade | 60 | 66 |
| Frederiksberg Allé | 70 | 77 |
| Bülowsvej | 70 | 77 |
| Gl. Kongevej | 80 | 88 |
| Bernstorffsvej | 90 | 99 |
| Hellerupvej | 90 | 99 |
| Strandvej | 100 | 110 |
| Trianglen | 110 | 121 |
| Østerbrogade | 110 | 121 |
| Grønningen | 120 | 132 |

| | | |
|---|---|---|
| Bredgade | 130 | 143 |
| Kgs. Nytorv | 130 | 143 |
| Østergade | 140 | 154 |
| Amagertorv | 150 | 165 |
| Vimmelskaftet | 150 | 165 |
| Nygade | 160 | 176 |
| Frederiksberggade | 175 | 193 |
| Rådhuspladsen | 200 | 220 |
| Carlsberg, Tuborg | 75 | 83 |
| Ø.K. | 100 | 110 |
| D.F.D.S. | 100 | 110 |
| D/S Bornholm | 100 | 110 |
| A/S Øresund | 100 | 110 |

| | |
|---|---|
| 1.17 | If a player-piece lands on a “chance” field, the player shall draw a card and follow the cards description |
| 1.18 | When a player-piece passes the starting field the player shall receive +200 balance |
| 1.19 | When a player-piece lands on the “go to prison” they shall immediately go to the prison field and will not receive balance from requirement 1.18. The player will then suffer the penalties from being in prison:<br>● The player is unable to move out of prison. To escape, follow requirement 1.23.<br>If a player-piece lands on the prison field but has not been sent there by a card or field no penalties shall fall upon the player |
| 1.20 | If a player gets the same dice values two turns in a row they shall be granted an extra throw after their first throw. If the player proceeds to throw the same dice value 3 times in a row they shall be sent to prison and suffer the prison penalties |
| 1.21 | If a player is moved to an unowned field by a chance card then normal field purchase regulations shall apply to the player |
| 1.22 | If a player has been set in prison they can escape by:<br>● Pay a fee of 50 kroner<br>● Use an escape chance card<br>● Throw two dice and roll the same value on each dice<br>When throwing two dice of the same value you will move the amount |

| | |
|---|---|
| | corresponding to the dice throw and still get an extra turn form throwing two dice of the same value in a row. If three rounds have passed and the player have failed to escape prison they will either have to escape on the said round or be forced to pay the fee of 50 kroner and move according to the value thrown beforehand. The player put in prison still have the rights to buy houses and hotels and receive rent. |
| 1.23 | If a player owns all streets in a color-group, an opportunity to build/sell houses shall be granted to him at the beginning his round.<br>The prices to build and sell a single house are as following: |

| Color group | Buy 1 house/hotel | Sell 1 house |
|---|---|---|
| Cyan | 50 | 25 |
| Pink | 50 | 25 |
| Green | 100 | 50 |
| Gray | 100 | 50 |
| Red | 150 | 75 |
| Magenta | 150 | 75 |
| Yello | 200 | 100 |
| Orange | 200 | 100 |

| | |
|---|---|
| | The houses shall be build evenly. A player can't buy house nr. 2 on a ground, without having bought 1 house on each of the other fields, in the same color-group.<br>Buying a hotel on a street also requires 4 houses on each of the other streets in the same color-group. |
| 1.24 | At the beginning of a round, the player shall always be granted an opportunity to trade with another player, in attempt to receive new fields or money in exchange for their own fields or money.<br>This can be done by offering cash, fields and/or granting a 'get out of jail' card. |

### 2.1.2 Usability requirements

| | |
|---|---|
| 2.1 | The game shall be understandable enough that the average person should be able to play the game without the need for instructions outside of the game. |

### 2.1.3 Reliability requirements

None

### 2.1.4 Performance requirements

| | |
|---|---|
| 4.1 | The time between the input and result of a dice throw shall have a response time of no more than 333 milliseconds. |

### 2.1.5 Supportability requirements

| | |
|---|---|
| 5.1 | The game shall be installable and playable on Windows machines |
| 5.2 | The game shall be translatable by editing only one file. |
| 5.4 | The game shall be playable by three to six players. |

### 2.1.6 Additional non-functional requirements

| | |
|---|---|
| 6.1 | The game shall be developed using the IDE Eclipse and the Java Programming Language |

## 2.2 Noun/verb analysis

We have used the game rules to generate this list of nouns and verbs. These we can use in our further investigation and development of the game.

| Nouns | Verbs |
|---|---|
| 1. Player<br>2. Dice<br>3. Car<br>4. Board<br>5. Field<br>6. Start<br>7. Street<br>8. Refuge<br>9. Fleet<br>10. Brewery<br>11. Tax<br>12. “Feeling lucky”<br>13. Prison<br>14. Money | 1. Throw (dice)<br>2. Play<br>3. Move (car)<br>4. Land on<br>5. Buy (field/house/hotel)<br>6. Build (house/hotel)<br>7. Sell (house/hotel)<br>8. Demolish (house/hotel)<br>9. Pay (rent)<br>10. Get ownership<br>11. Trade<br>12. Pledge<br>13. Pass start<br>14. Go to (Prison) |

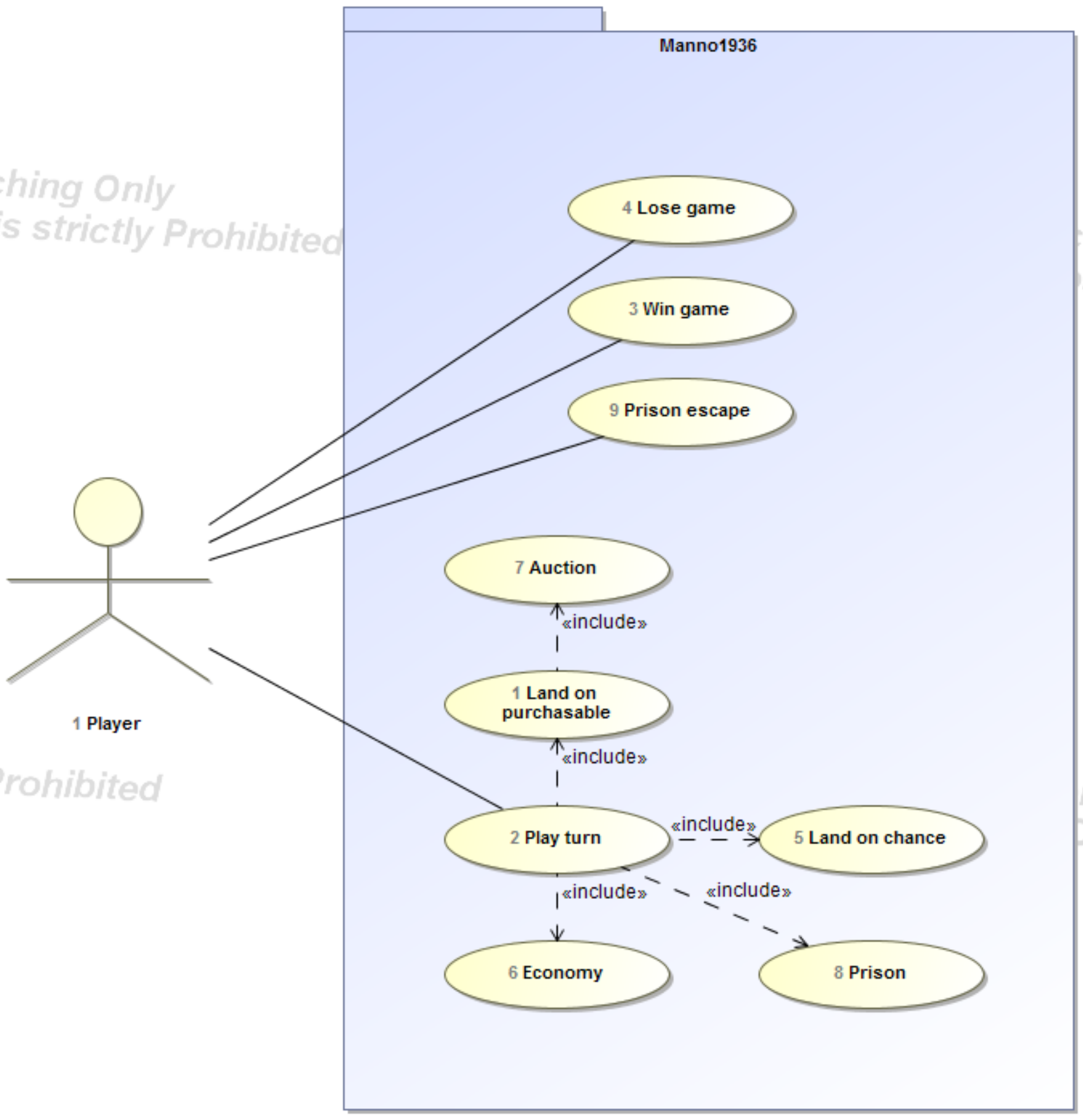
|  |  |
|---|---|
| 15. Positive/negative effect<br>16. Ownership (of a specific field)<br>17. Choice (between tax or amount)<br>18. Bankruptcy<br>19. Chance card<br>20. Extra turn<br>21. Color group<br>22. Rent<br>23. House<br>24. Hotel<br>25. Auction<br>26. Deed card<br>27. Assets<br>28. Bank<br>29. Fine | 15. Bid<br>16. Offer<br>17. Leave (prison) |

Matador can be *played* by *players throwing* 2 *dice* and *moving* their *car* on the *board* accordingly. The board consists of *fields*, which can have one of the following types: *start, street, refuge, fleet, brewery, tax*, *"feeling lucky"*, *prison* and *go to prison*. *Landing on* the fields can have a *positive or negative effect* on the player's *money*. One of the tax fields gives the player a *choice* between paying a tax rate or tax amount.
A player is able to *buy a field* and gain *ownership* of it, symbolised by a *deed card*, but if the field is owned by another player, he will have to *pay rent*. The player can *pledge* his owned fields to the *bank* if he is in need of money.
If a player owns all fields in a *color group*, then he can start to buy/*build houses* and *hotels* to gain an increased rent from the other players. The player can *sell/demolish* his houses and hotels and get money from the bank.
Players can *trade* their *assets* (money, deed cards and chance cards) internally, and thus *offer* their assets to the other players. Also, if a player chooses not to buy a field, the other players may *bid* on the field in an *auction* to get it at a discounted price.
The player can *go to prison*, and to *leave* prison he must throw two equal dice or pay a *fine*.
Throwing two equal dice gives the player an *extra turn*, but if the player throws two equal dice three times in a row, he is sent to prison.
When the player *passes start* he gains an amount of money.

# 3 Design

## 3.1 Use cases

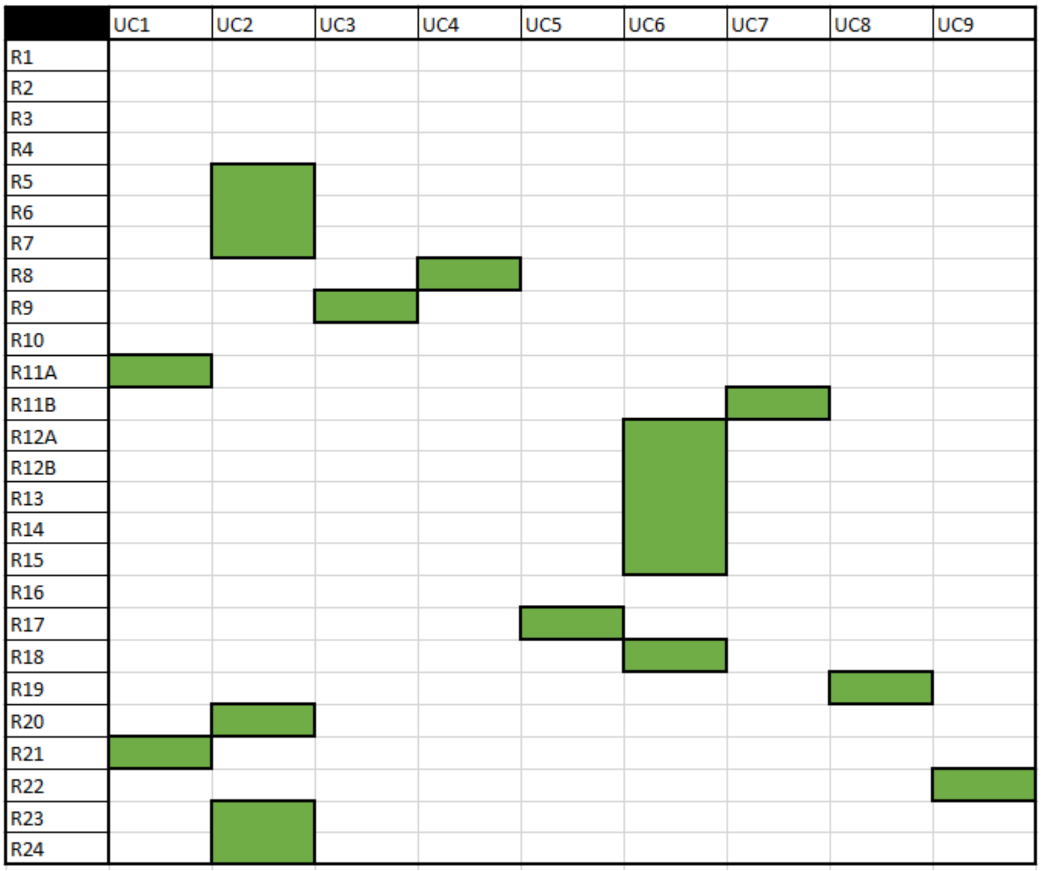
### 3.1.1 Use case diagram



Above is our Use Case Diagram that shows all of our use cases and actors. We have three primary actors, which all are players in the game.

## 3.1.2 Requirements traceability matrix

Based on our use cases, we’ve created a matrix, which only purpose is to assist us to determine the completeness of our use-cases. As seen below, most of our requirements are have an associated use case, while a few others doesn’t have one. The few requirements that doesn’t have a single use case associated, are simple functional requirements, that can be handled without a use case, for instance ‘The game shall be played with at least 3, and max 6 players’, or ‘player one shall always start’.



| | UC1 | UC2 | UC3 | UC4 | UC5 | UC6 | UC7 | UC8 | UC9 |
|---|---|---|---|---|---|---|---|---|---|
| R1 | | | | | | | | | |
| R2 | | | | | | | | | |
| R3 | | | | | | | | | |
| R4 | | | | | | | | | |
| R5 | | 🟩 | | | | | | | |
| R6 | | 🟩 | | | | | | | |
| R7 | | 🟩 | | | | | | | |
| R8 | | | | 🟩 | | | | | |
| R9 | | | 🟩 | | | | | | |
| R10 | | | | | | | | | |
| R11A | 🟩 | | | | | | | | |
| R11B | | | | | | | 🟩 | | |
| R12A | | | | | | 🟩 | | | |
| R12B | | | | | | 🟩 | | | |
| R13 | | | | | | 🟩 | | | |
| R14 | | | | | | 🟩 | | | |
| R15 | | | | | | 🟩 | | | |
| R16 | | | | | | | | | |
| R17 | | | | | 🟩 | | | | |
| R18 | | | | | | 🟩 | | | |
| R19 | | | | | | | | 🟩 | |
| R20 | | 🟩 | | | | | | | |
| R21 | 🟩 | | | | | | | | |
| R22 | | | | | | | | | 🟩 |
| R23 | | 🟩 | | | | | | | |
| R24 | | 🟩 | | | | | | | |

### UC1: Land on ownable

| | |
|---|---|
| **Use case ID:** | 1 |
| **Use case name:** | Land on ownable |
| **Description:** | When a player lands on an ownable field (Street, fleet or brewery) they shall be able to purchase it. |
| **Primary actors:** | Player |
| **Secondary actors:** | None |
| **Preconditions:** | 1. It is the player’s turn<br>2. The player rolls the dice and lands on an ownable field |
| **Postconditions:** | 1. The cost of the field gets subtracted from the player’s account |

| | |
|---|---|
| | 2. The ownership of the field goes to the player |
| **Main flow:** | 1. The player gets the option to buy the ownable field<br>2. The player chooses to buy the field |
| **Alternate flows:** | 1A: Another player has ownership of the field<br>1. The player is not given the option to buy the field<br>2. The player has to pay rent to the owner<br><br>1B: The player’s balance is less than the price of the field<br>1. The player is not given the option to buy the field<br>2. The player’s balance remains unchanged<br><br>2A: If the player chooses not to buy the field<br>1. The player’s balance remains unchanged.<br>2. (An auction begins **(Use Case ID: 7 (Auction))** To be implemented at a later date). |

## UC2: Play turn

| | |
|---|---|
| **Use case ID:** | 2 |
| **Use case name:** | Play turn |
| **Description:** | When a player plays their turn they will have to throw two dice and move their player piece accordingly. The player also gets the option to trade with other players and the option to buy houses/hotels if possible |
| **Primary actors:** | Player |
| **Secondary actors:** | None |
| **Preconditions:** | 1. It is the player’s turn |
| **Postconditions:** | 1. The player gets affected by the field they land on |
| **Main flow:** | 1. The player buys a house or trades<br>2. The player throws their dice<br>3. The player piece gets moved from its current position a set number of field corresponding to the dice throw forward upon the board |
| **Alternate flows:** | 1A: The player does not buy a house or trades<br>1. The player’s balance does not change<br><br>3A: The player throws two dice of the same value<br>1. The player gets given an extra turn<br>2. If the player throws two time the same value again<br>a. The player gets sent to prison<br>b. The player suffers the prison penalties<br>3. Else the player moves normally according to the throw |

## UC3: Win game

| | |
|---|---|
| **Use case ID:** | 3 |
| **Use case name:** | Win game |
| **Description:** | A player wins if there are no other players remaining in the game. |
| **Primary actors:** | Player |
| **Secondary actors:** | None |
| **Preconditions:** | 1. There are only two players remaining in the game.<br>2. Second player loses the game.<br>3. It is the current player’s turn. |
| **Postconditions:** | 1. A message will be shown, congratulating the winner and giving the choice of:<br>a. Exiting to the main menu<br>b. Play a new game |
| **Main flow:** | 1. The current player wins the game. |
| **Alternate flows:** | None |

## UC4: Lose game

| | |
|---|---|
| **Use case ID:** | 4 |
| **Use case name:** | Lose game |
| **Description:** | When the player’s balance would be reduced to less than the total value of the player’s assets, the player is declared bankrupt and loses the game. The games continues as long as there are at least two players in the game. |
| **Primary actors:** | Player |
| **Secondary actors:** | None |
| **Preconditions:** | 1. It is the player’s turn<br>2. The player encounters a scenario resulting in loss account balance |
| **Postconditions:** | 1. The player is declared as having lost the game<br>2. All houses/hotels have been removed from the player’s owned fields<br>3. The player has lost ownership of any owned fields |
| **Main flow:** | 1. The player throws dice and lands on another players property |

|  |  |
|---|---|
|  | 2. The player can't afford to pay the other player<br>3. The player goes bankrupt |
| **Alternate flows:** | None |

## UC5: Land on chance

|  |  |
|---|---|
| **Use case ID:** | 5 |
| **Use case name:** | Land on chance |
| **Description:** | The player land on a chance card field and draws a card |
| **Primary actors:** | Player |
| **Secondary actors:** | None |
| **Preconditions:** | 1. It is the player's turn.<br>2. The player land on a chance card field |
| **Postconditions:** | 1. The player's balance is increased with 2000. |
| **Main flow:** | 1. The player throws dice and lands on a chance field<br>2. The player draws the "Matador"-scholarship<br>3. The player has a sum of owned assets, that are under or equal to 750. |
| **Alternate flows:** | 2A: The player draws another type of card<br>1. The player's balance gets affected by the card.<br>2. The player gets to keep the card if it is a "Get out of jail"-card<br>3A: The player has a sum of owned assets, that are greater than 750.<br>1. The player's balance does not get affected. |

## UC6: Economy

|  |  |
|---|---|
| **Use case ID:** | 6 |
| **Use case name:** | Economy |
| **Description:** | The player lands on a field which subtracts balance from the player's account |
| **Primary actors:** | Player |
| **Secondary actors:** | None |
| **Preconditions:** | 1. It's the player's turn |

| | |
|---|---|
| | 2. The player has landed on a field owned by another player |
| **Postconditions:** | 1. The owner of the field gains the lost money from the player |
| **Main flow:** | 1. The player transfers an amount corresponding to the rent of the field (and its potential bonuses from houses or special field effects) to the field owner |
| **Alternate flows:** | None |
| **Note:** | There are also the tax field, start field and the refuge field, which gives, subtracts or does nothing but since these are lesser advanced versions of this case we will not include them as use cases. |

## UC7: Auction

| | |
|---|---|
| **Use case ID:** | 7 |
| **Use case name:** | Auction |
| **Description:** | A player refused to buy a field, and an auction for the field begins |
| **Primary actors:** | Player |
| **Secondary actors:** | None |
| **Preconditions:** | 1. It is the player's turn.<br>2. The player has landed on a field, which is not owned.<br>3. The player refuses to buy the field |
| **Postconditions:** | 1. The highest bid gets subtracted from the highest bidder's account.<br>2. The highest bidder gets the ownership of the field. |
| **Main flow:** | 1. Auction begins and all players are able to bid, starting from the next player.<br>2. The next player gets the choice to<br>a. Offer the first bid<br>b. Decline to bid and withdraw from auction<br>3. Next bidder*<br>a. Raises the bid and continues in the bidding<br>b. Declines and withdraw from auction<br>4. *This continues till only one bidder is remaining |
| **Alternate flows:** | 2A: The next player's balance is equal to 0.<br>1. Next player will automatically decline to bid, and bidding will continue to Next Bidder.<br>3A: Next Bidder's balance is less than highest bid.<br>1. Next Bidder will automatically decline the bid.<br>4A: Everyone declines the bid.<br>1. The field remains unchanged. |

## UC8: Prison

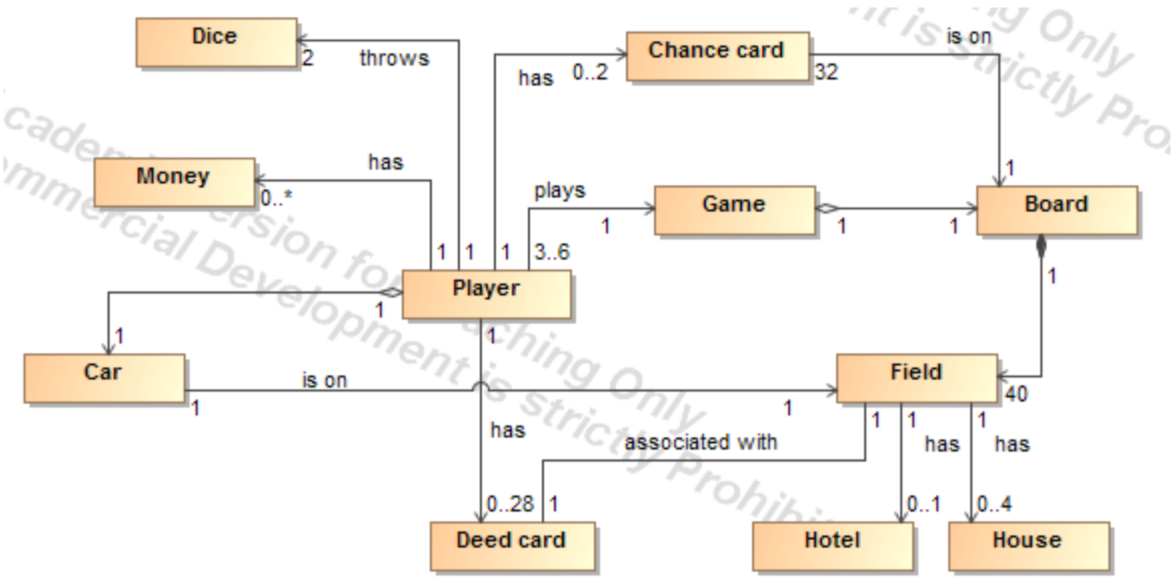
| | |
|---|---|
| **Use case ID:** | 8 |
| **Use case name:** | Prison |
| **Description:** | A Player triggers an effect which sends them to prison and they get jailed |
| **Primary actors:** | Player |
| **Secondary actors:** | None |
| **Preconditions:** | 1. It's the player's turn<br>2. The player triggers an effect in which they get jailed |
| **Postconditions:** | 1. The player moves to the prison field<br>2. The player suffers the prison penalties |
| **Main flow:** | 1. The player lands on the "go to prison" field<br>2. The player gets jailed |
| **Alternate flows:** | 1A: The player draws a go to jail chance card<br>1. The player gets jailed<br><br>1B: The player throws two equal dice three times in a row<br>1. The player gets jailed |

## UC9: Prison escape

| | |
|---|---|
| **Use case ID:** | 9 |
| **Use case name:** | Prison escape |
| **Description:** | A player escapes prison |
| **Primary actors:** | Player |
| **Secondary actors:** | None |
| **Preconditions:** | 1. It's the player's turn<br>2. The player is jailed |
| **Postconditions:** | 1. The player moves away from the prison field |
| **Main flow:** | 1. The player is given the choice of throwing the dice or paying 50.<br>2. The player chooses to throw the dice and rolls the same value for both dice.<br>3. The player moves the thrown value of tiles out of jail and |

| | |
|---|---|
| | gets an extra turn. |
| **Alternate flows:** | 1A: The player hasn't gotten out of jail in 3 turns<br>1. A: The player rolls the same value for both dice and gets out<br>1. B: The player doesn't roll the same value<br>a. The player pays 50 kr. from their balance<br>b. The player moves out according to the thrown value<br><br>2A: The player chooses to throw the dice, but does not roll the same value for both dice<br>1. The player stays jailed<br><br>2B: The player chooses to pay 50 kr.<br>1. The player gets 50 kr subtracted from their balance<br>2. The player moves out of jail as you would move normally |

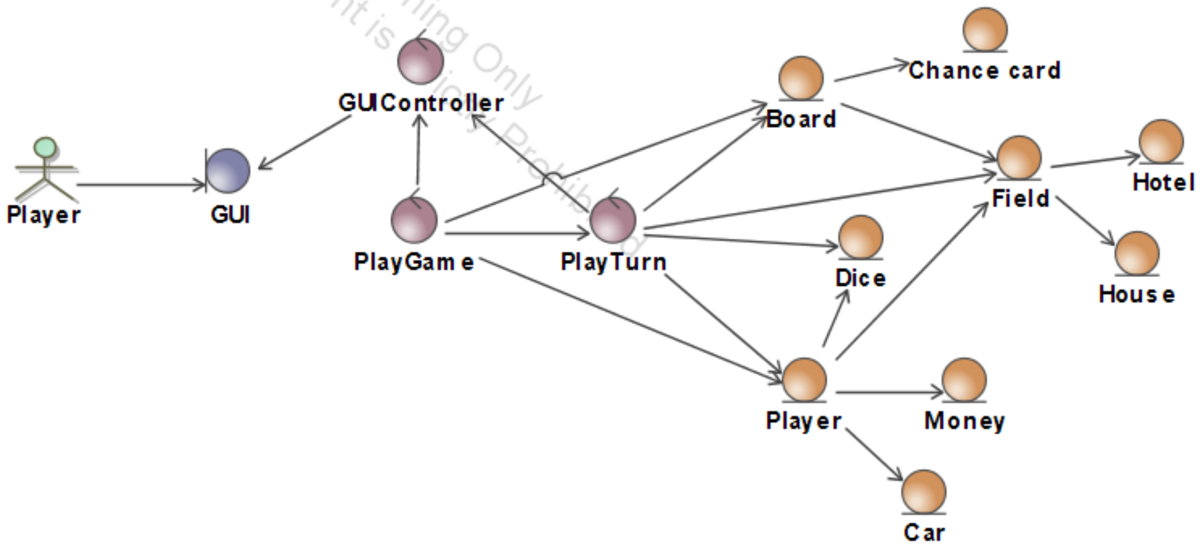
## 3.2 Domain model



Based on the game description and rules we have made the above definition of the domain. We have a *game* that is played by 3 to 6 *players*, which have *money* and a *car*. The player's car is placed on a *field*. There are two *dice* thrown by a player. There are *chance cards*, which a user can have up to two of, if he draws the "Get out of jail free"-cards. The 32 chance cards are placed on the *board*, that consists of 40 fields, which can have up to one *hotel* or up to four *houses*. The player can also have up to 28 *deed cards*, that are each associated with a field.

The deed card is associated with a field, which means that it has all the information about the field. Therefore we might be able to merge them into one.

## 3.3 BCE diagram



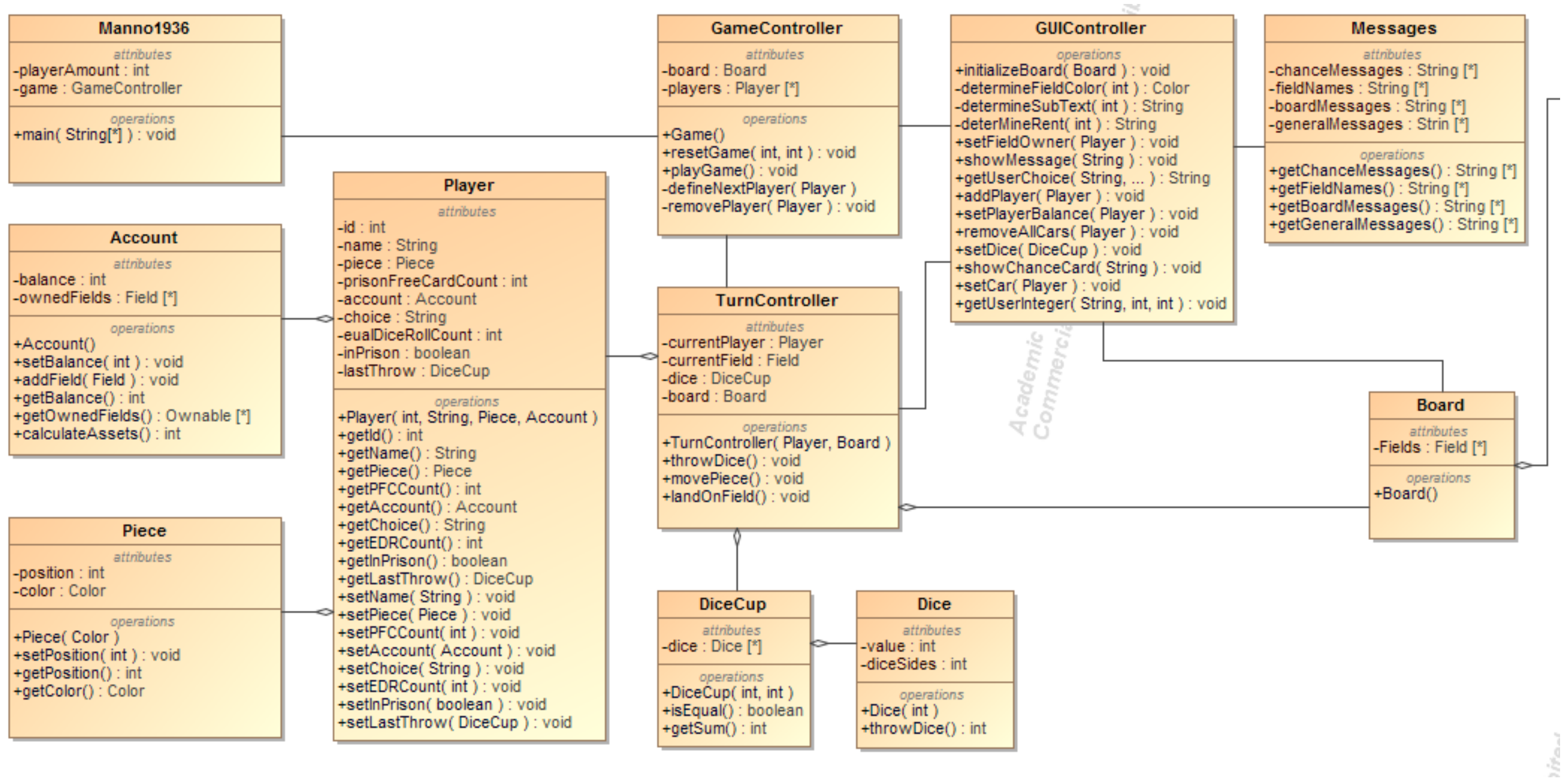
The Boundary Control Entity diagram shows the interaction between our system and the user. Usually our use cases would translate directly into controllers, but we have decided to merge some of them into the PlayTurn controller, because they are small. The entities are the same as we have in the domain model, except that deed card has been eliminated. As explained earlier, we can keep this information in the Field.

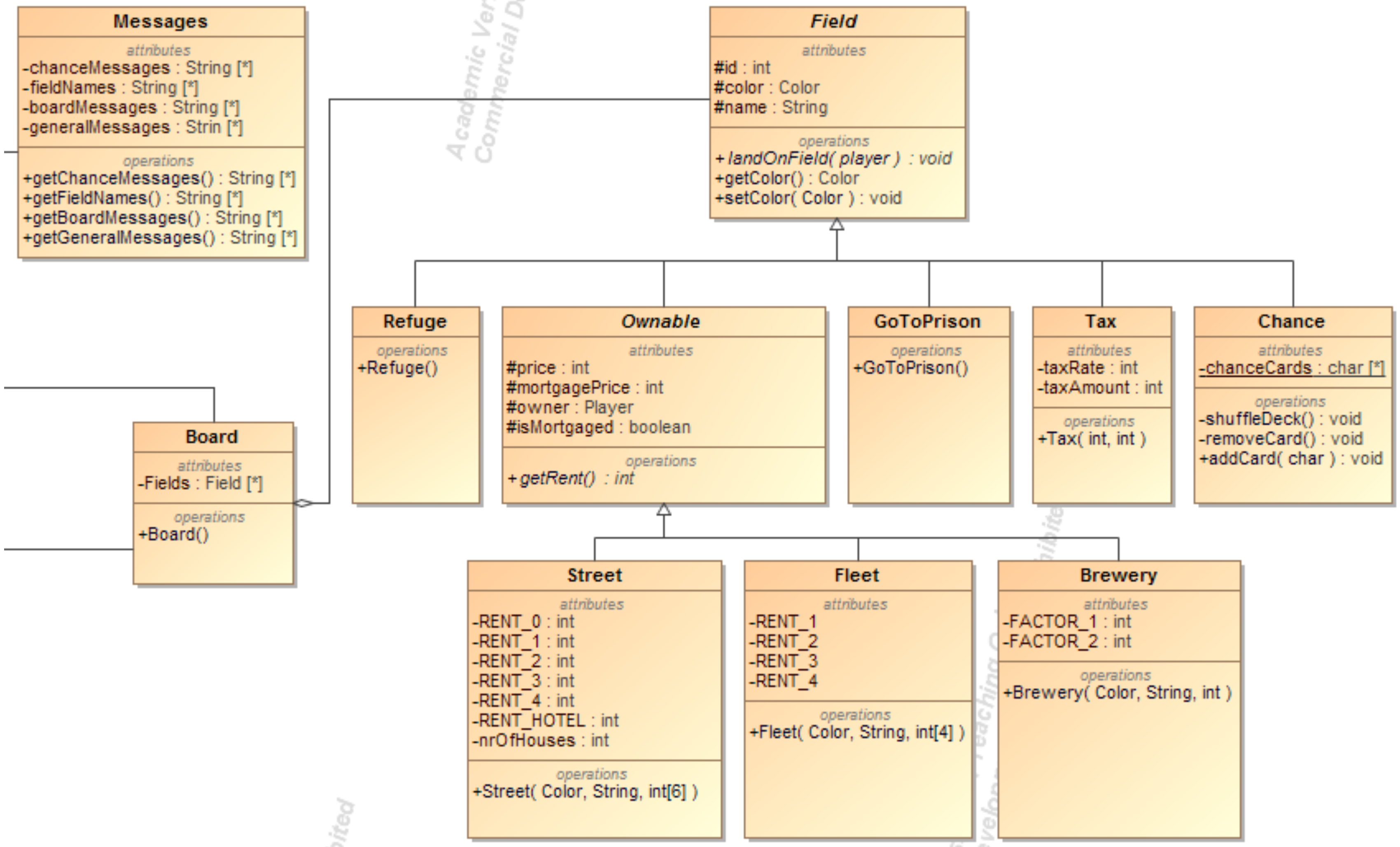
We have three controllers to handle information to the GUI. One is our GUIController which we can use to communicate with the GUI. Then we have PlayGame and PlayTurn, which both communicate with the GUIController. PlayGame communicates with PlayTurn, which communicates with most of our entities.

## 3.4 Analysis class diagram

Based on our system requirements, our domain model and BCE diagram we have made this class diagram. This is our first sketch of how the program should be written.

We have reused more than a few classes from earlier projects and their responsibilities are largely the same.


Manno1936
attributes
-playerAmount : int
-game : GameController
operations
+main( String[*] ) : void
Account
attributes
-balance : int
-ownedFields : Field [*]
operations
+Account()
+setBalance( int ) : void
+addField( Field ) : void
+getBalance() : int
+getOwnedFields() : Ownable [*]
+calculateAssets() : int
Piece
attributes
-position : int
-color : Color
operations
+Piece( Color )
+setPosition( int ) : void
+getPosition() : int
+getColor() : Color
Player
attributes
-id : int
-name : String
-piece : Piece
-prisonFreeCardCount : int
-account : Account
-choice : String
-eualDiceRollCount : int
-inPrison : boolean
-lastThrow : DiceCup
operations
+Player( int, String, Piece, Account )
+getId() : int
+getName() : String
+getPiece() : Piece
+getPFCCount() : int
+getAccount() : Account
+getChoice() : String
+getEDRCount() : int
+getInPrison() : boolean
+getLastThrow() : DiceCup
+setName( String ) : void
+setPiece( Piece ) : void
+setPFCCount( int ) : void
+setAccount( Account ) : void
+setChoice( String ) : void
+setEDRCount( int ) : void
+setInPrison( boolean ) : void
+setLastThrow( DiceCup ) : void
GameController
attributes
-board : Board
-players : Player [*]
operations
+Game()
+resetGame( int, int ) : void
+playGame() : void
-defineNextPlayer( Player )
-removePlayer( Player ) : void
TurnController
attributes
-currentPlayer : Player
-currentField : Field
-dice : DiceCup
-board : Board
operations
+TurnController( Player, Board )
+throwDice() : void
+movePiece() : void
+landOnField() : void
DiceCup
attributes
-dice : Dice [*]
operations
+DiceCup( int, int )
+isEqual() : boolean
+getSum() : int
Dice
attributes
-value : int
-diceSides : int
operations
+Dice( int )
+throwDice() : int
GUIController
operations
+initializeBoard( Board ) : void
-determineFieldColor( int ) : Color
-determineSubText( int ) : String
-deterMineRent( int ) : String
+setFieldOwner( Player ) : void
+showMessage( String ) : void
+getUserChoice( String, ... ) : String
+addPlayer( Player ) : void
+setPlayerBalance( Player ) : void
+removeAllCars( Player ) : void
+setDice( DiceCup ) : void
+showChanceCard( String ) : void
+setCar( Player ) : void
+getUserInteger( String, int, int ) : void
Messages
attributes
-chanceMessages : String [*]
-fieldNames : String [*]
-boardMessages : String [*]
-generalMessages : Strin [*]
operations
+getChanceMessages() : String [*]
+getFieldNames() : String [*]
+getBoardMessages() : String [*]
+getGeneralMessages() : String [*]
Board
attributes
-Fields : Field [*]
operations
+Board()
Academic
Commercia


Messages
attributes
-chanceMessages : String [*]
-fieldNames : String [*]
-boardMessages : String [*]
-generalMessages : Strin [*]
operations
+getChanceMessages() : String [*]
+getFieldNames() : String [*]
+getBoardMessages() : String [*]
+getGeneralMessages() : String [*]
Field
attributes
#id : int
#color : Color
#name : String
operations
+landOnField( player ) : void
+getColor() : Color
+setColor( Color ) : void
Board
attributes
-Fields : Field [*]
operations
+Board()
Refuge
operations
+Refuge()
Ownable
attributes
#price : int
#mortgagePrice : int
#owner : Player
#isMortgaged : boolean
operations
+getRent() : int
GoToPrison
operations
+GoToPrison()
Tax
attributes
-taxRate : int
-taxAmount : int
operations
+Tax( int, int )
Chance
attributes
-chanceCards : char [*]
operations
-shuffleDeck() : void
-removeCard() : void
+addCard( char ) : void
Street
attributes
-RENT_0 : int
-RENT_1 : int
-RENT_2 : int
-RENT_3 : int
-RENT_4 : int
-RENT_HOTEL : int
-nrOfHouses : int
operations
+Street( Color, String, int[6] )
Fleet
attributes
-RENT_1
-RENT_2
-RENT_3
-RENT_4
operations
+Fleet( Color, String, int[4] )
Brewery
attributes
-FACTOR_1 : int
-FACTOR_2 : int
operations
+Brewery( Color, String, int )

### 3.4.1 Responsibilities

Manno1936 is the main class responsible for starting the program.

GameController handles all logic related to game flow. It is responsible for removing players, shifting the turn to next player and declaring the winner.

TurnController manages all the logic inside the turn of each player.

GUIController is the class responsible for receiving inputs from other parts of the system and connecting these to the GUI library we have been given.

Dice stores a single value that can be determined randomly.

DiceCup is responsible for storing multiple dice, throwing these and storing relevant information about them.

Player is responsible for storing information connected to the players. Each player is given an ID, a name, a piece and an account.

Account is responsible for storing the player’s balance and owned fields.

Piece is a small class that is used to hold the position of the player on the board and the color of the car on the GUI.

Board stores all the necessary information on each of the fields we need for our game.

Field is the superclass in a inheritance hierarchy containing each of our different field types. It serves as the general structure for each of the more specific classes, meaning that each class under it must have a landOnField method. Each of the classes that have Field as their superclass is responsible for one type of field on the board.

Ownable is a subclass of Field that is worth noting as it also has several subclasses. It generalizes all of the field types that can be owned by the players, and as such contains methods that are relevant to ownership of a field. Each subclass of Ownable is a type of Field that can also be owned by a player.

Tax, Refuge and GoToPrison are subclasses of Field. Each of them are responsible for applying an effect when the player lands on them. Tax should subtract an amount from the player’s balance, while Refuge and GoToPrison should not affect the balance.

Fleet, Street and Brewery are subclasses of Ownable. Each is responsible for calculating and storing rents used when a player other than the owner lands on them.

Chance is a subclass of Field and is responsible for storing, shuffling and drawing from the deck of cards we use when landing on one of the Chance fields.

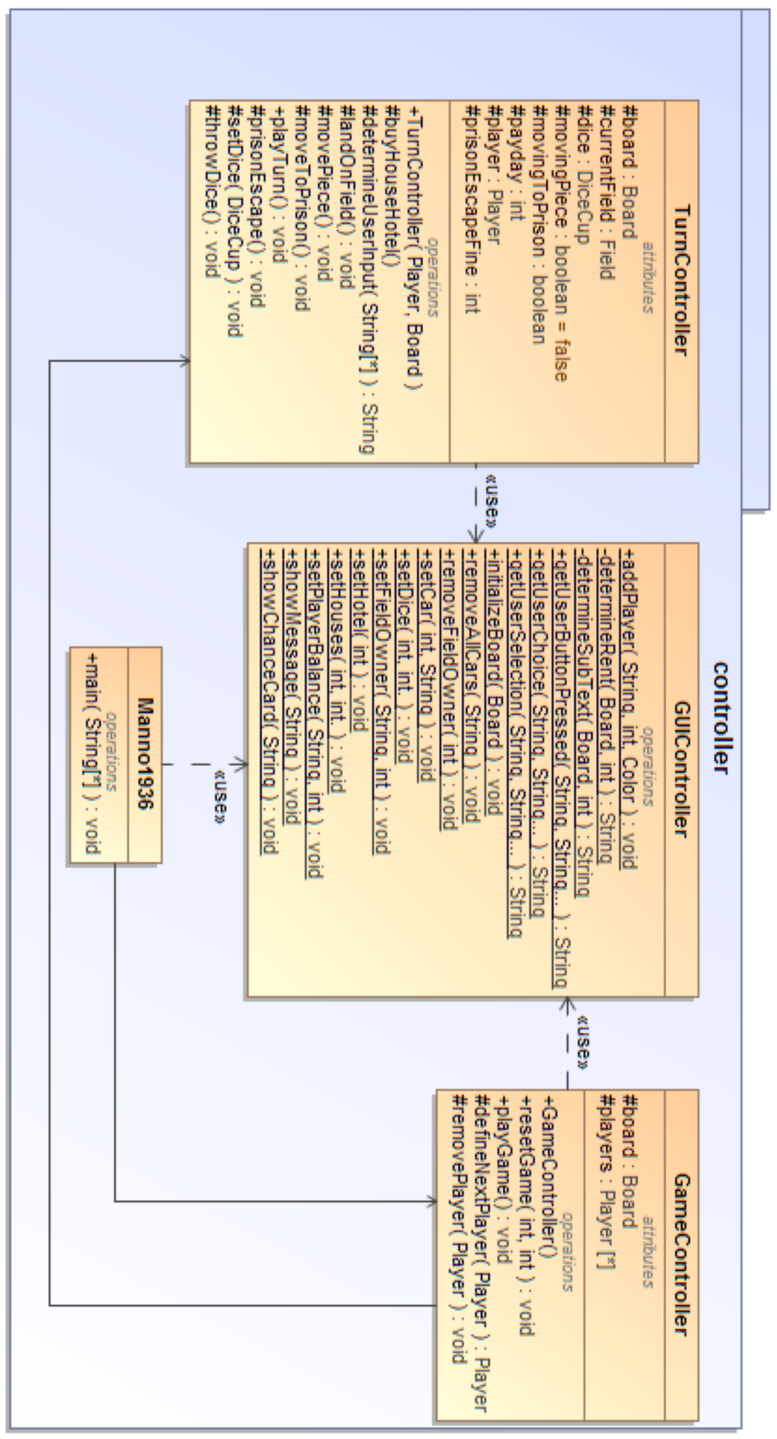
Messages is a class we use to ease translation of the game. It stores all messages that the GUI displays, including all the names of the fields. This ensures that a translator would only have to edit a single file in order to translate the game.

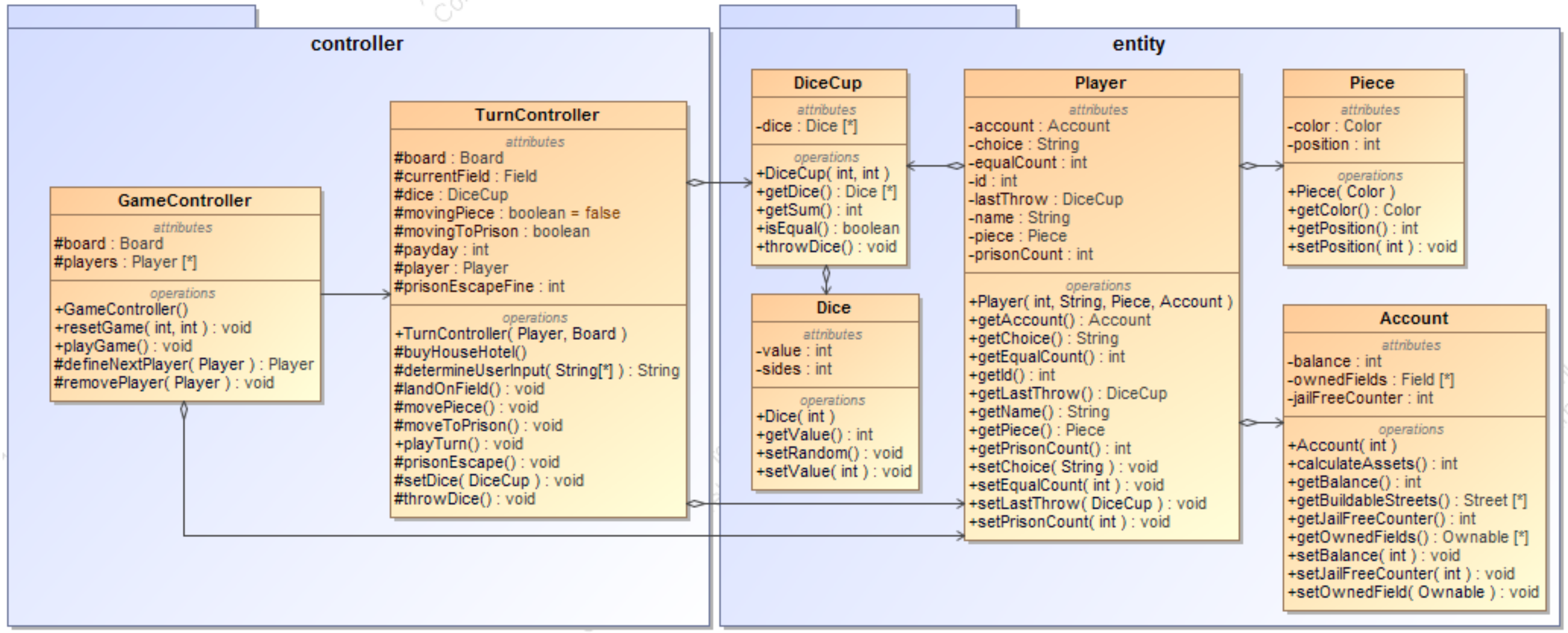
# 4 Implementation

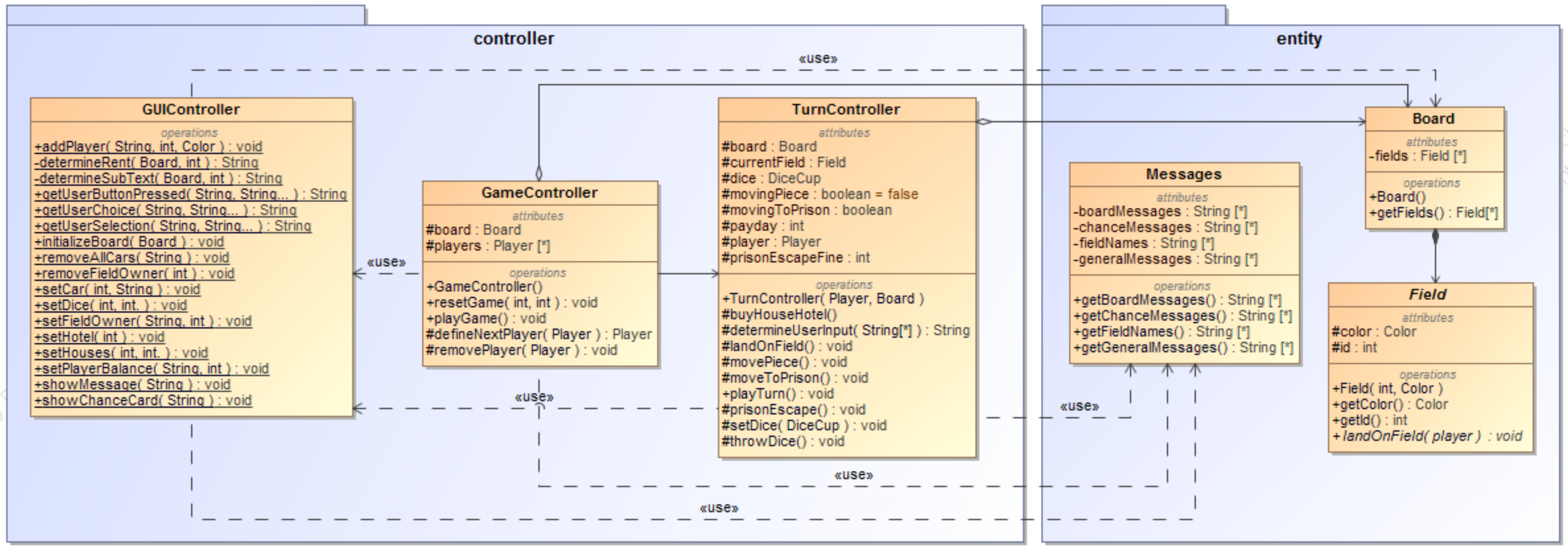
## 4.1 Diagrams

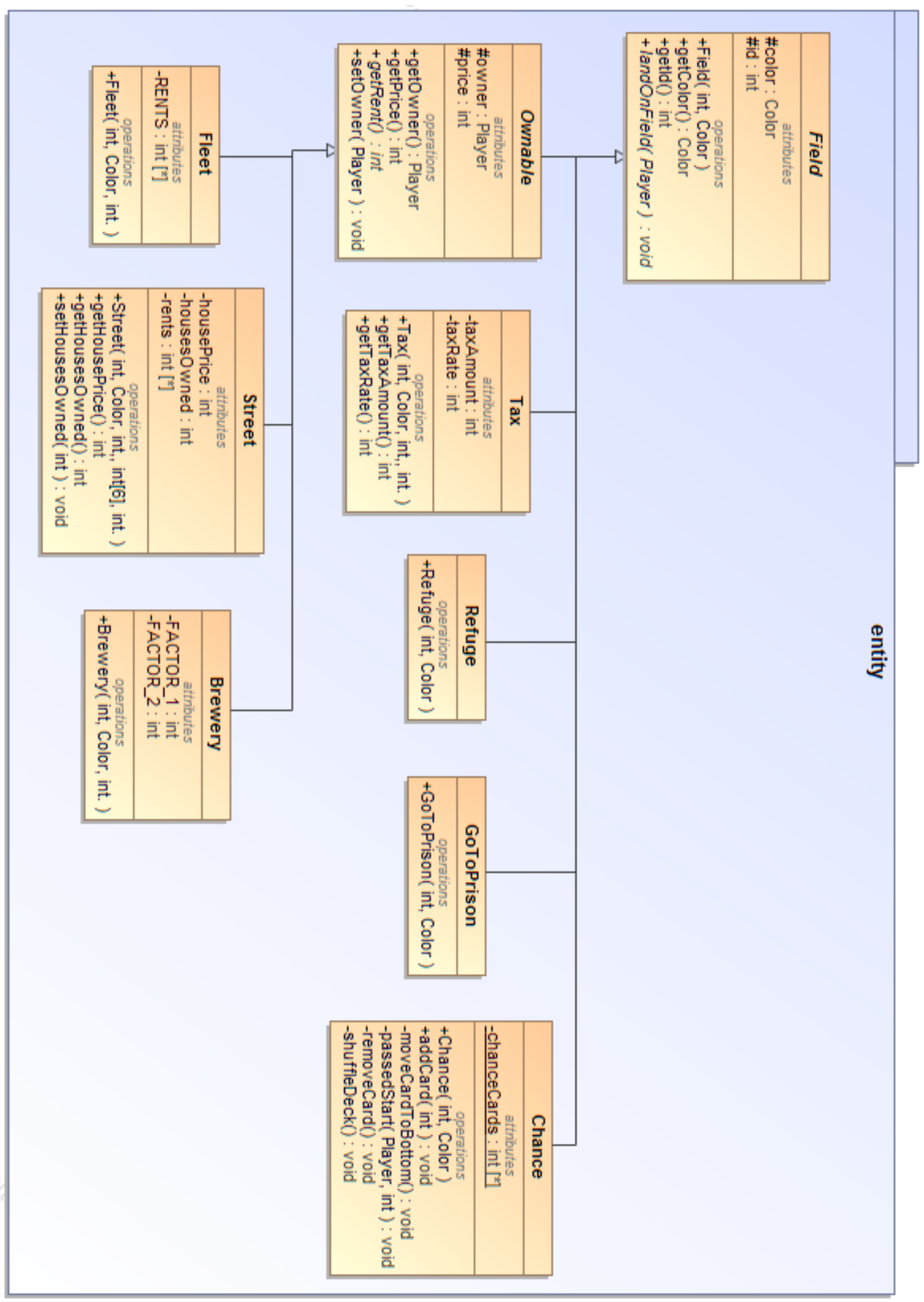
### 4.1.1 Design class diagram

We have created four class diagrams to show the classes and the relations between them.


controller
TurnController
attributes
#board : Board
#currentField : Field
#dice : DiceCup
#movingPiece : boolean = false
#movingToPrison : boolean
#payday : int
#player : Player
#prisonEscapeFine : int
operations
+TurnController( Player, Board )
#buyHouseHotel()
#determineUserInput( String[*] ) : String
#landOnField() : void
#movePiece() : void
#moveToPrison() : void
+playTurn() : void
#prisonEscape() : void
#setDice( DiceCup ) : void
#throwDice() : void
«use»
GUIController
operations
+addPlayer( String, int, Color ) : void
-determineRent( Board, int ) : String
-determineSubText( Board, int ) : String
+getUserButtonPressed( String, String... ) : String
+getUserChoice( String, String... ) : String
+getUserSelection( String, String... ) : String
+initializeBoard( Board ) : void
+removeAllCars( String ) : void
+removeFieldOwner( int ) : void
+setCar( int, String ) : void
+setDice( int, int ) : void
+setFieldOwner( String, int ) : void
+setHotel( int ) : void
+setHouses( int, int ) : void
+setPlayerBalance( String, int ) : void
+showMessage( String ) : void
+showChanceCard( String ) : void
«use»
GameController
attributes
#board : Board
#players : Player [*]
operations
+GameController()
+resetGame( int, int ) : void
+playGame() : void
#defineNextPlayer( Player ) : Player
#removePlayer( Player ) : void
«use»
Manno1936
operations
+main( String[*] ) : void

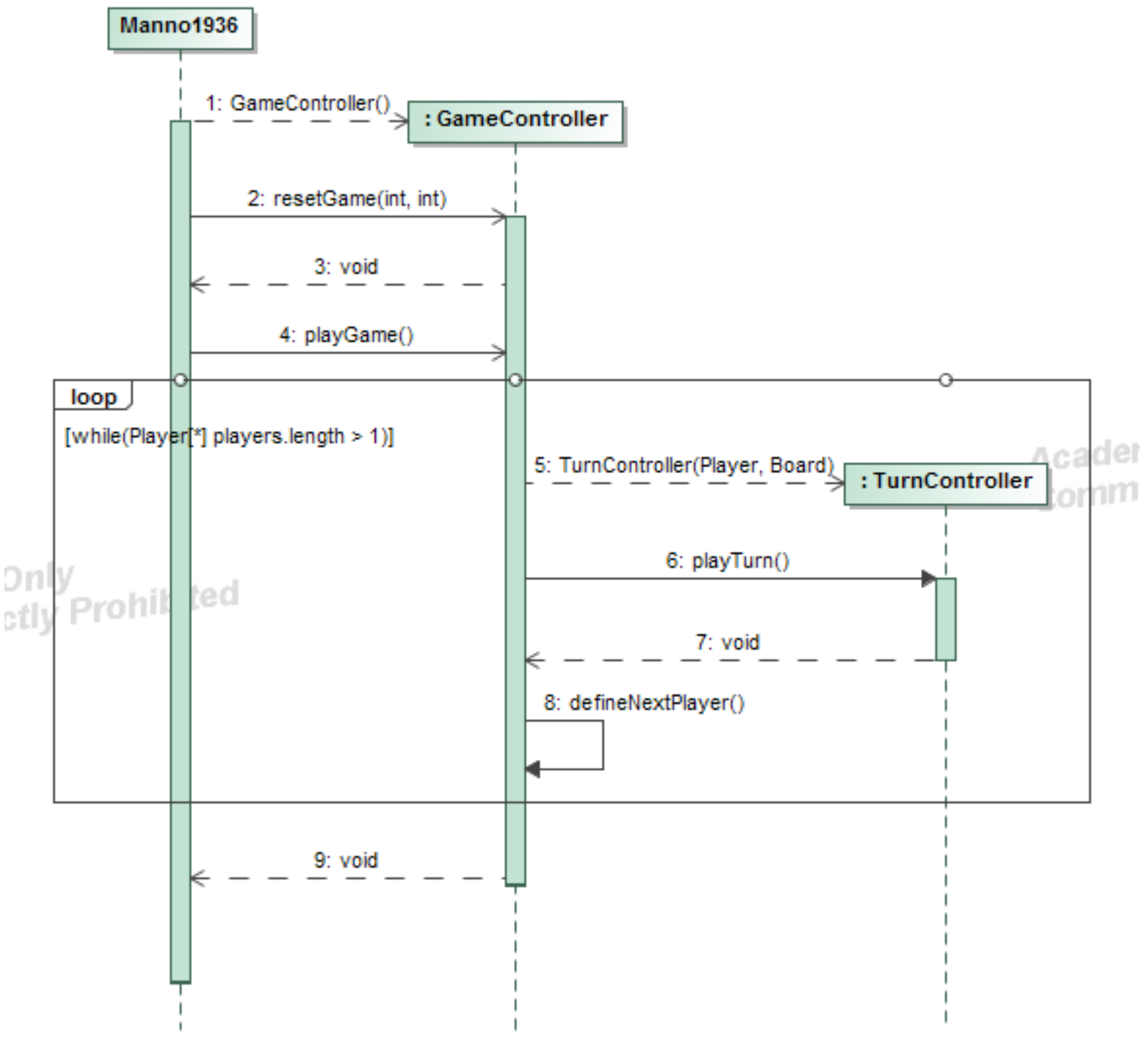

controller
GameController
attributes
#board : Board
#players : Player [*]
operations
+GameController()
+resetGame( int, int ) : void
+playGame() : void
#defineNextPlayer( Player ) : Player
#removePlayer( Player ) : void
TurnController
attributes
#board : Board
#currentField : Field
#dice : DiceCup
#movingPiece : boolean = false
#movingToPrison : boolean
#payday : int
#player : Player
#prisonEscapeFine : int
operations
+TurnController( Player, Board )
#buyHouseHotel()
#determineUserInput( String[*] ) : String
#landOnField() : void
#movePiece() : void
#moveToPrison() : void
+playTurn() : void
#prisonEscape() : void
#setDice( DiceCup ) : void
#throwDice() : void
entity
DiceCup
attributes
-dice : Dice [*]
operations
+DiceCup( int, int )
+getDice() : Dice [*]
+getSum() : int
+isEqual() : boolean
+throwDice() : void
Dice
attributes
-value : int
-sides : int
operations
+Dice( int )
+getValue() : int
+setRandom() : void
+setValue( int ) : void
Player
attributes
-account : Account
-choice : String
-equalCount : int
-id : int
-lastThrow : DiceCup
-name : String
-piece : Piece
-prisonCount : int
operations
+Player( int, String, Piece, Account )
+getAccount() : Account
+getChoice() : String
+getEqualCount() : int
+getId() : int
+getLastThrow() : DiceCup
+getName() : String
+getPiece() : Piece
+getPrisonCount() : int
+setChoice( String ) : void
+setEqualCount( int ) : void
+setLastThrow( DiceCup ) : void
+setPrisonCount( int ) : void
Piece
attributes
-color : Color
-position : int
operations
+Piece( Color )
+getColor() : Color
+getPosition() : int
+setPosition( int ) : void
Account
attributes
-balance : int
-ownedFields : Field [*]
-jailFreeCounter : int
operations
+Account( int )
+calculateAssets() : int
+getBalance() : int
+getBuildableStreets() : Street [*]
+getJailFreeCounter() : int
+getOwnedFields() : Ownable [*]
+setBalance( int ) : void
+setJailFreeCounter( int ) : void
+setOwnedField( Ownable ) : void


controller
entity
«use»
GUIController
operations
+addPlayer( String, int, Color ) : void
-determineRent( Board, int ) : String
-determineSubText( Board, int ) : String
+getUserButtonPressed( String, String... ) : String
+getUserChoice( String, String... ) : String
+getUserSelection( String, String... ) : String
+initializeBoard( Board ) : void
+removeAllCars( String ) : void
+removeFieldOwner( int ) : void
+setCar( int, String ) : void
+setDice( int, int, ) : void
+setFieldOwner( String, int ) : void
+setHotel( int ) : void
+setHouses( int, int, ) : void
+setPlayerBalance( String, int ) : void
+showMessage( String ) : void
+showChanceCard( String ) : void
GameController
attributes
#board : Board
#players : Player [*]
operations
+GameController()
+resetGame( int, int ) : void
+playGame() : void
#defineNextPlayer( Player ) : Player
#removePlayer( Player ) : void
TurnController
attributes
#board : Board
#currentField : Field
#dice : DiceCup
#movingPiece : boolean = false
#movingToPrison : boolean
#payday : int
#player : Player
#prisonEscapeFine : int
operations
+TurnController( Player, Board )
#buyHouseHotel()
#determineUserInput( String[*] ) : String
#landOnField() : void
#movePiece() : void
#moveToPrison() : void
+playTurn() : void
#prisonEscape() : void
#setDice( DiceCup ) : void
#throwDice() : void
«use»
«use»
«use»
«use»
«use»
Messages
attributes
-boardMessages : String [*]
-chanceMessages : String [*]
-fieldNames : String [*]
-generalMessages : String [*]
operations
+getBoardMessages() : String [*]
+getChanceMessages() : String [*]
+getFieldNames() : String [*]
+getGeneralMessages() : String [*]
Board
attributes
-fields : Field [*]
operations
+Board()
+getFields() : Field[*]
Field
attributes
#color : Color
#id : int
operations
+Field( int, Color )
+getColor() : Color
+getId() : int
+ landOnField( player ) : void



The first diagram shows our controllers, which we have put in a package called controller.
The second shows the relations from our GameController and TurnController to the Player class. The relations from other entities to the Player is also shown.
The third shows the relations from the controllers to the entities Board and Messages.
The fourth shows our superclass Field and its subclasses.

## 4.1.2 Design sequence diagram

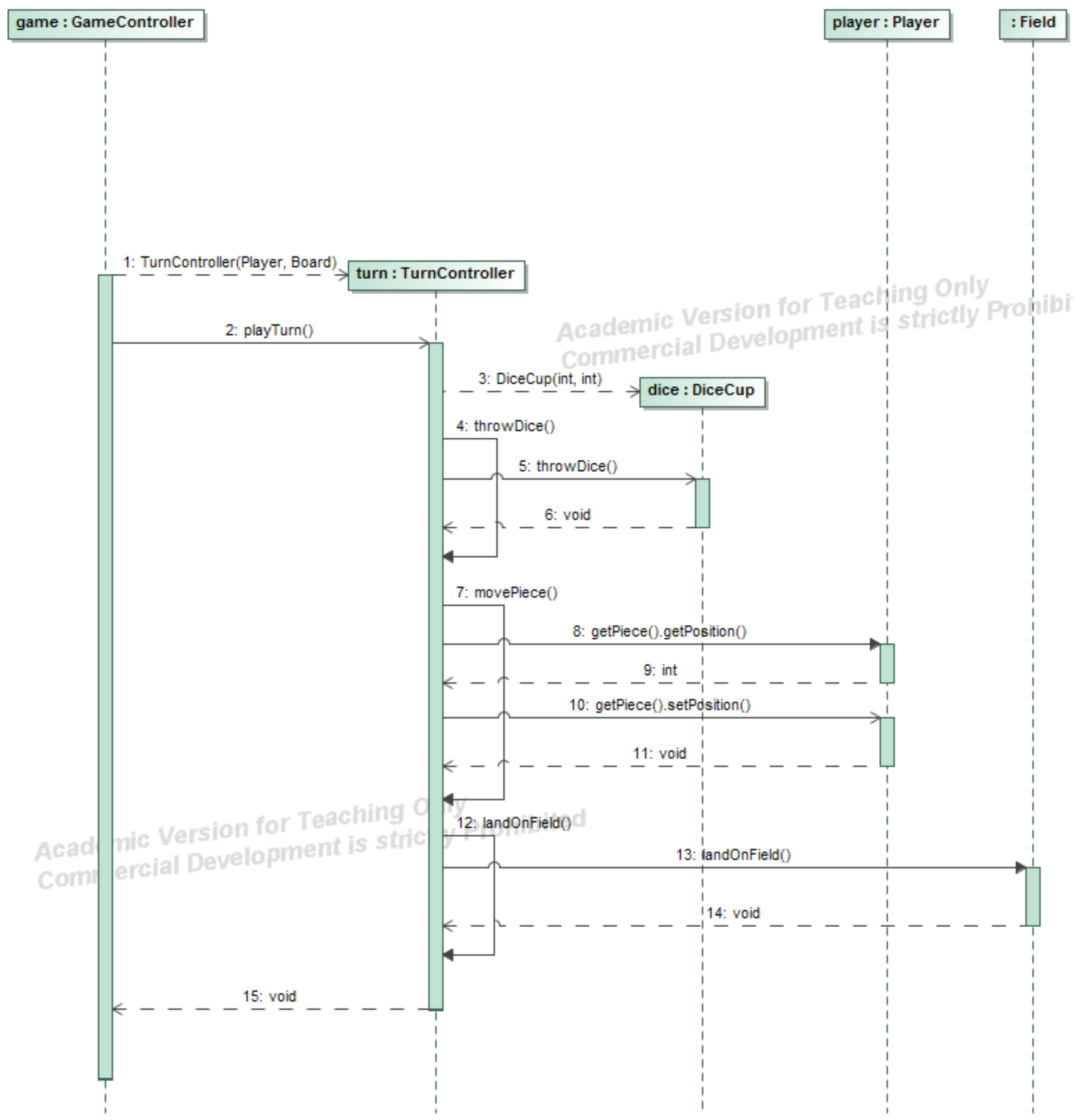


This diagram shows the basic overall sequence of method calls in our system. We have omitted almost all helping methods in the controller classes for the sake of simplicity.

When our game starts, we create a new instance of the GameController class and use the resetGame method to set up a new game with a specific amount of player and a certain starting balance for each of these players. Having the resetting method allows us a bit more flexibility with the class than we would get from having the constructor do the setup work.

Once we are satisfied with the game setup, we can start the game with the playGame method.

This method keeps creating new TurnController instances for each of the players in the player array of the GameController and then runs the method for initiating the turn logic. After a player is done with their turn, the game checks if there is a winner and if not, allows the turn to go on to the next player. It checks if there is a winner by checking if the player array has a length of 1, in which case there is only 1 player left.



In the TurnController, the basic program flow is that we first give the player the option to buy houses in their turn (Not pictured in the diagram as it is part of the playTurn method itself). After this we assign the DiceCup dice random values with the method throwDice(). We use this value to calculate the position of the player's piece and set it to this new position. After this we use the Board class as an information expert and fetch the Field corresponding to the player's new position. We then run the landOnField() method for the appropriate field.

### 4.1.3 GRASP

For this system we have based many of our design decisions around the patterns of GRASP.

**Low coupling:** Most of our classes have only one or two associations and do their communication with the rest of the program through their associated class. Doing this causes low coupling and means that we only end up having a few clear connections between different classes instead of having millions of methods that call to every single class in the program.

**High cohesion:** Our classes have focus around doing approximately one task, this causes high cohesion. Our Player class for example, is storing all the information on one player, such as an account, ID, position on the board, etc. The class Manno1936, is the class that initializes the game with a set amount of players. We have tried to keep the tasks as simple as possible, and every method and attribute in the class centered around that one task.

**Controller:** We decided to split our controller into four smaller controller classes, GameController, GUIController, Manno1936 and TurnController, in order to get a higher cohesion. Manno1936 is the main class that is responsible for starting the program. GameController is the one responsible for the logic to the game flow, it is responsible for, changing turns, removing players, and declaring the winner. GUIController is receiving input from other parts of the system, and then connecting them with the GUI library we have been given. Lastly the TurnController is the class managing all the logic inside the turn of each player. Such as throwing the dice, buying fields, going to prison and so on.

**Creator:** We mostly use the creator pattern in this system when:
- Creator contains the object class
- Creator closely uses the object class
- Creator has the necessary information for initialization of the object class.

Player is an example of a creator, since it contains the object class Account, another example could be Board, which is composed of Fields. An example of the last pattern is when we need to determine the amount of player the game should be played with, this is not determined upon initialization, but the class that creates the Game object, is the class that ends up setting the amount of players.

**Information expert:** Our Board class is a great example of an information expert. Board contains the information on all of the fields in our game, and is used to access the individual fields and fetch their information.

**Pure fabrication:** An example of pure fabrication would be the Messages class. This class contains all the messages we display on the GUI.

**Polymorphism:** This pattern is used in the inheritance between Field and the specific types of fields. This means the our landOnField() and getRent() methods take on different forms depending on which type of field you land on.

**Indirection:** This pattern have been used when making the board class. The pattern supports low coupling and reusability of certain parts of the system by making sure that the TurnController class has to get through an intermediate object in order to get the information on each of the fields.

We have not used the protected variations pattern for this project.

## 4.2 Code documentation

### 4.2.1 Manno1936

Manno1936 is the main class, that gets initialized when the program gets initialized. We call it the start menu of the game which has the only responsibility of starting a new game, or exiting the program. Whenever the player chooses to start a new game, the player gets to choose the amount of players (3-6 players), that shall participate. Once that is selected, the game is on and GameController takes over.

### 4.2.2 GameController

GameController is a controller class which bundles up other classes and run/resets the game when asked to. The GameController is also responsible for choosing which player is next in line for a turn in the game.

### 4.2.3 TurnController

The turn controller has the responsibility of handling everything that happens within a player’s turn. When it is a player’s turn, TurnController starts to check the player’s assets and if there are any ownable fields in his assets, he’ll be given the choice to build houses. To continue the turn, the player has to roll the dice. A call to DiceCup is made in order to get two values that will determine how far the player will move on the board. Every requirement of rolling pairs, or going to jail for rolling pairs too many times in a row, will also be examined, in order to get the correct outcome. TurnController also verifies the field that the player lands on, so that the player either pays rent to the correct owner, or gets the option to buy the field. If it turns out, that the player is serving jail time, he’ll get the option to either pay the fine, or roll the dice. If the roll is not a pair, he’ll stay in jail until his third attempt. If he fails to hit a pair in third attempt, he’ll be fined 50 and move the sum of the dice.

### 4.2.4 GUIController

The GUIController is our medium class for us to communicate with the interface of the game. The class have a method to initialize the whole board as well as a remove players, move pieces, set colors, interact with field owners and houses/hotel placements, etc. The GUIController also makes it possible for us to write messages (which we all have stored in a seperate class) in the game. We haven’t made our own GUI so we are borrowing a premade one and as to make our program easier to test we have made this class with the sole purpose of handling the commands which came with the premade GUI.

### 4.2.5 Board

Board is one of our entity classes and has the responsibility of containing all of the games field. When board is instantiated in another class the class will have access to all 40 fields of the game. The fields in the board class have in them a price for the field, a rent for the field (with the exception of some fields not buyable etc.), color for the field and all the fields are separated into types (fleet, street etc.).

## 4.2.6 Field

Field is a simple abstract class which defines a color and id to a field. The main point of this class is to let other field type classes inherit its functions so they can get assigned a color and id themselves

### 4.2.6.1 Ownable

Ownable is an abstract class which extends the Field class. Ownable makes it possible for a player to land on a purchasable field and purchase it (if it is not already owned). Ownable is also the class you will have to contact if you would want to use the rent of a specific field.

#### 4.2.6.1.1 Street

Street is a part of an ownable field, where the unique part is, that you can build houses. This class contain information regarding the house price, it’s rent and the number of houses that’s been constructed.

#### 4.2.6.1.2 Fleet

Fleet is also a part of an ownable field, but different than street. There are no possibility to place houses on a fleet, however the rent is doubled by the amount of fleets you own.

#### 4.2.6.1.3 Brewery

Brewery does not have a rent, but two factors only. The sum of the dice is multiplied by one of the factors. If the owner of the brewery owns only one brewery, the sum will be multiplied by factor_1, which is 4. if the owner of the brewery owns two breweries, the sum of the dice will be multiplied by 10.

### 4.2.6.2 GoToPrison

GoToPrison is an extension of the Field  class with the purpose of being able to send a player to the “Prison” field and make them stay in prison for a maximum of 3 turns by manipulating some values in the “Player” class

### 4.2.6.3 Tax

Tax is an extension of the Field class with the purpose of withdrawing money from the player when needed (when they land on a tax field)

### 4.2.6.4 Chance

Chance is an extension of the Field class and holds all of the Chance card effects. The player when landing on a chance card field will be affected by one of these stored effects chosen randomly (by shuffling the card deck at start) but it is not possible for the same chance card to be used twice since a used card will be discarded

### 4.2.6.5 Refuge

Refuge is a simple class that does nothing. It’s meant to be a sanctuary for the player.

### 4.2.7 DiceCup

DiceCup is a class we use to initialize a specified amount of dice (through an array) with a specified amount of sides each through the dice class. DiceCup is used to throw the specified dice and get a random value from them as well as get their sum and evaluate if the dice thrown is all of the same value.

#### 4.2.7.1 Dice

Dice is the class we use to get a specific die with a specific amount of dice sides. We use this to get the dice in the Dice

### 4.2.8 Messages

We wanted to create a game that can easily be translated. To do so, we had to create a class containing all the messages that was sent from the game, to the user. In the messages, we separated the messages into different type of messages.
All messages are saved in a String array. They are unchangeable and can be achieved using a get method in the class.
The messages has been separated into: fieldNames, boardMessages, generalMessages and chanceMessages.

### 4.2.9 Player

This is the Player class, that stores information regarding a player. There's id and name of the Player, which determines which player that's being looked into. Here we also store the last throw, amount of pairs in a row, and number of turns left in prison. From Player, we call two other classes, Piece and Account, to take care of other things, not regarding the player itself, but still related to the player.
The unique thing in this class is the String choice which is used to get past the GUI in the test phase.

#### 4.2.9.1 Piece

Piece is a class that has a small, but important responsibility. It is called through the Player class, which makes it a personal piece for the specific player. The information stored are the piece's color and the piece's position on the board.

#### 4.2.9.2 Account

Account is also called through the Player class to make it a personal account of the specific Player. It contains information of all the player's assets, which is the 'get out of jail' card, the player's owned fields, and it's balance. It contains a method to calculate the player's assets as well as get/set methods for all the information stored.

## 4.3 Version control

We have used GitHub to manage our code and version control during this project.

We had a master branch, which would contain our current “finished” version, after it had been developed and tested. If bugs were found, they would be fixed in the branch bugfixes, and then merged into our master branch.

For each feature we were developing, we made a new branch, so that changes regarding different features were kept separated until merging into master. Doing this made us have less conflicts during development of the program compared to having only one branch.

We still had to make sure that we always merged master into the different branches after a new version had been completed, so we were up to date with the current version of the program.

The master branch should only be updated with merges from other branches, except for changes to the Messages class, which was important to keep the same across all branches.

# 5 Test

## 5.1 Unit tests

### 5.1.1 Dice

We have tested the class Dice with two tests. The first test finds out whether the dice is “random”, meaning that one value is not more present than others when throwing the dice 60.000 times. The test succeeded.

The second test is testing whether the values returned by the dice corresponds to the number of sides on the dice. This means that the test will fail if a six sided dice returns the value seven.

### 5.1.2 Field

When testing the Field class and its subclasses, we are mainly interested in the landOnField and getRent methods implemented in the subclasses of Field.

We started with all of the classes that are subclasses to the Ownable class. This means Street, Fleet and Brewery.

Street is a flat rent which can be increased by buying a house on it. Testing a Street’s getRent method proved that it returned the correct rent with all of the different possible amounts of houses.

Fleet fields’ rents increase with the number of other Fleet fields that a given player owns. The getRent method for this class also returns the correct amounts based on our JUnit tests.

Brewery is a base rent multiplied with the player’s dice roll. The base rent increases when more than one brewery is owned, just like Fleet. The correct base rent is returned for each amount of owned Brewery fields, but we had to move the multiplication of the rent into the landOnField method of Ownable as getRent does not take a parameter.

Ownable’s landOnField method allows players to buy the field if it does not have an owner. When the field does have an owner, the player who lands on it must pay rent to the owner. The first player who lands on the field is given the option correctly, and the second player pays rent correctly based on our tests. We tested for the special case of Brewery and for a Street. Fleet is the same test as Street and is therefore trivial.

When testing the Tax field we test for both choosing having a percentage of a player’s assets taken or choosing a flat amount of money. Both of these work as intended and change the balance of a player’s account correctly based on our unit test of Tax.

## 5.2 Use case tests

## TC01: Land on ownable

| | |
|---|---|
| **Test case ID** | TC01 |
| **Summary** | Test to see if landing on an ownable field makes the moving player able to buy it, and when bought the withdrawal of money is correct |
| **Requirements** | R11A |
| **Preconditions** | The player stands on field 1. |
| **Test procedure** | 1. The player throws the dice<br>2. He lands on Rødovrevej<br>3. He chooses yes to buy the field |
| **Test data** | Dice roll: {1,2}<br>User choice: {"Slå med terningerne", "Ja"}<br>The price of Rødovrevej is 60. |
| **Expected result** | Player balance: 1440<br>Owner of Rødovrevej is the player<br>Rødovrevej is on the player's owned fields |
| **Actual result** | Player balance: 1440<br>Owner of Rødovrevej is the player<br>Rødovrevej is on the player's owned fields |
| **Status** | Passed |
| **Tested by** | Freya |
| **Date** | 13/01/2017 |
| **Test environment** | Eclipse Neon 1a Release (4.6.1) |

## TC02: Play turn

| | |
|---|---|
| **Test case ID** | TC02 |
| **Summary** | Test to see if player is offered to buy a house and whether he is subtracted the price of the house, and the field adds 1 to his counter of houses. |
| **Requirements** | R12A, R23 |
| **Preconditions** | 1. The player owns all streets in the color group.<br>2. The player has a higher balance than the cost of the house<br>3. All streets in the color group does not have hotels<br>4. It is the player's turn |

| | |
|---|---|
| **Test procedure** | 1. The player chooses “Køb hus eller hotel”<br>2. The player chooses one of the streets and clicks ok<br>3. The player chooses “Ja” |
| **Test data** | Dice roll: {1,2}<br>User choice: {“Køb hus eller hotel”, “Hvidovre”, “Ja”}<br>The price of a house on Hvidovre is 50 |
| **Expected result** | Player balance: 1450<br>Houses on Hvidovre: 1 |
| **Actual result** | Player balance: 1450<br>Houses on Hvidovre: 1 |
| **Status** | Passed |
| **Tested by** | Freya |
| **Date** | 13/01/2017 |
| **Test environment** | Eclipse Neon 1a Release (4.6.1) |

### TC03: Win game

| | |
|---|---|
| **Test case ID** | TC03 |
| **Summary** | Test to see if the player can win a game |
| **Requirements** | R9 |
| **Preconditions** | Two players are left in the game |
| **Test procedure** | 1. Two players are initialized with a balance of 100.<br>2. The first player is given a result that causes them to land on a tax field, where they choose to pay 200.<br>3. The first player’s balance is -100 and they are removed from the game |
| **Test data** | Dice roll: {3,1} |
| **Expected result** | 1. The second player is the only player remaining and has thus won. |
| **Actual result** | The second player is the only one remaining. |
| **Status** | Passed. |
| **Tested by** | Tobias |
| **Date** | 13/01/2017 |
| **Test environment** | Eclipse Neon 1a Release (4.6.1) |

## TC04: Lose game

| | |
|---|---|
| **Test case ID** | TC04 |
| **Summary** | A test to see if a player loses if their balance is reduced to a value less than 0. |
| **Requirements** | R8 |
| **Preconditions** | One of the players has a balance low enough that the next time they land on a field will reduce it to a value less than 0. |
| **Test procedure** | 1. Two players are initialized with a balance of 100.<br>2. The first player is given a result that causes them to land on a tax field, where they choose to pay 200.<br>3. The first player's balance is -100 and they are removed from the game |
| **Test data** | Dice roll: {3,1} |
| **Expected result** | The first player is removed from the game and has lost. |
| **Actual result** | The player array is reduced to 1 player and the only one in it is the second player. The first player has lost. |
| **Status** | Passed. |
| **Tested by** | Tobias |
| **Date** | 13/01/2017 |
| **Test environment** | Eclipse Neon 1a Release (4.6.1) |

## TC05: Land on chance

| | |
|---|---|
| **Test case ID** | TC05 |
| **Summary** | We test whether the chance card field effects function properly |
| **Requirements** | R17 |
| **Preconditions** | The player is in a position to land on a chance field |
| **Test procedure** | 1. We initialize a player with a balance of 1500.<br>2. We determine the dice roll to make the player land on a chance field.<br>3. We determine which effect the chance card should have.<br>4. We play the turn for the player. |
| **Test data** | Dice roll: {6,1} |

| | |
|---|---|
| | Position test: Effect 6 (Move the player 3 fields back)<br>Balance test: Effect 10 (Pay 100)<br>Scholarship test 751: Effect 17 & player is initialized with 751 as their balance insted of 1500<br>Scholarship test 750: Effect 17 & player is initialized with 750 as their balance insted of 1500<br>Scholarship test 749: Effect 17 & player is initialized with 749 as their balance insted of 1500 |
| **Expected result** | Position test: Player’s position is 5, Player’s balance is 1300 (Because they land on a tax field)<br>Balance test: Player’s balance is 1400<br>Scholarship test 751: Player’s balance is 751<br>Scholarship test 750: Player’s balance is 2750<br>Scholarship test 749: Player’s balance is 2749 |
| **Actual result** | All results are the same as the expected results |
| **Status** | Passed |
| **Tested by** | Freya, Tobias |
| **Date** | 15/01/2017 |
| **Test environment** | Eclipse Neon 1a Release (4.6.1) |

## TC06: Economy

| | |
|---|---|
| **Test case ID** | TC06 |
| **Summary** | Test to see if the player pays rent to the owner correctly. |
| **Requirements** | R12A, R12B |
| **Preconditions** | Player 2 owns field 4 |
| **Test procedure** | 1. Player 1 throws a dice roll of 3 and lands on field 4.<br>2. Player 1 clicks ok to pay rent to Player 2 |
| **Test data** | Dice roll: {1,2}<br>User choice: {“Slå med terningerne”, “OK”}<br>The rent on Hvidovre with no houses is 4 |
| **Expected result** | Player 1 balance: 1496<br>Player 1 balance: 1504 |
| **Actual result** | Player 1 balance: 1496<br>Player 1 balance: 1504 |

| | |
|---|---|
| **Status** | Passed |
| **Tested by** | Freya |
| **Date** | 13/01/2017 |
| **Test environment** | Eclipse Neon 1a Release (4.6.1) |

## TC07: Prison

| | |
|---|---|
| **Test case ID** | TC07 |
| **Summary** | Test to see if players get imprisoned when either landing on the “go to prison” field or hit equal dice 3 times in a row |
| **Requirements** | R19 |
| **Preconditions** | 1. The player has hit equal dice 2 times in a row |
| **Test procedure** | 1. Create 2 players in game<br>2. Move player 1 to the “go to prison” field<br>3. Simulate player 2 to have thrown 2 times equal dice<br>4. Simulate turn for player 2 with a throw of equal dice |
| **Test data** | Player 2 - setEqualCount(2) |
| **Expected result** | 1. Player 1 prisonCount = 3<br>2. Player 1 position = field id: 11<br>3. Player 2 prisonCount = 3<br>4. Player 2 position = field id: 11 |
| **Actual result** | 1. Player 1 prisonCount = 3<br>2. Player 1 position = field id: 11<br>3. Player 2 prisonCount = 3<br>4. Player 2 position = field id: 11 |
| **Status** | Passed |
| **Tested by** | Gustav Hammershøi Petersen |
| **Date** | 13/01/2017 |
| **Test environment** | Eclipse Neon 1a Release (4.6.1) |

### TC08: Prison escape

| | |
|---|---|
| **Test case ID** | TC08 |
| **Summary** | A test to see whether the player is able to get out of the prison either by paying or rolling two dice with equal values |
| **Requirements** | R22 |
| **Preconditions** | 1. The player's position is the prison field<br>2. The player has a prisoncount greater than 1<br>3. It is the player's turn |
| **Test procedure** | 1. The player chooses "Slå med terningerne"<br>2. The player chooses "Betal 50 kr."<br>3. The player chooses "OK"<br>4. The player chooses "Nej" |
| **Test data** | Dice roll: {1,2}<br>User choice: {"Slå med terningerne", "Betal 50 kr.", "OK", "Nej"}<br>Playerposition: 11 (Prison)<br>Prisoncount: 3 |
| **Expected result** | Player balance: 1450<br>Prisoncount: 0 |
| **Actual result** | Player balance: 1450<br>Prisoncount: 0 |
| **Status** | Passed |
| **Tested by** | Theis |
| **Date** | 15/01/2017 |
| **Test environment** | Eclipse Neon 1a Release (4.6.1) |

## 5.3 Integration test

We made a demo of the game to show that the game works as intended. We have used this as our integration test, because it runs through a game in its entirety and is showing the different possibilities on the fields.

## 5.4 Test conclusion

### 5.4.1 Unit tests

We made a few simple unit tests to check the validity of some of our classes which were not necessarily use cases. We tested the Dice class and the Field class as well as its hierarchy.

We tested the dice by setting its sides to 6 and threw it 60000 times. The value spread equally with a very low failure margin and the results did not go beneath or above the set amount of dice sides. We therefore evaluated our dice to be working.

We tested the Field class and its hierarchy by observing the rents when landing upon the fields. Through our tests we concluded that the rents were correctly calculated including the more complex rent fields like “brewery, fleet and tax”.

### 5.4.2 Use case tests

We have tested our use cases by constructing a special “TurnController test class”, where we can define what the dice should throw and what the user will click on in the GUI. This makes us able to test specific scenarios as described in the use cases.
Our tests are made as JUnit Test Cases, and they all succeeded. This means that we have tested the majority of our requirements, as described in our Requirements traceability matrix. Some requirements have not been covered in use cases since we didn’t have enough time to implement them as features. We have a use case covering an “auction” mechanic, but we did not have enough time to implement this feature, which in turn lead to us not having a test case for it.

### 5.4.3 Integration test

We have made a demonstration of our system, which through predetermined dice rolls makes a set of three players land on all of the different field types in the game and plays through the common scenarios that can play out during the course of the game, like paying rent or buying fields. Based on this demo and from running several games without predetermined results, we assess that our system works when we assemble its parts into its whole.

# 6 Conclusion

Based on the original Matador from 1936, we've made and fulfilled a requirement specification. We've further designed and created the board, in which players can move around with its piece, buy fields and build houses. Important features such as chance cards, prison etc. has been implemented. A few features, which is written in the requirement specification has not been implemented in the game due to the time limit. These features include selling or pawning of owned fields or houses/hotels, auctioning of fields, certain chance cards (such as get out of jail) and trading. Instead, we focused on bug fixing to ensure a fully functional game.
The structure of the game has been documented, and the game has been tested sufficiently, so that we can guarantee a finished and working product.

# 7 Literature

John Lewis, William Loftus: *Java software solutions, Seventh Edition*
Published 2012
Pearson Education, Inc., publishing as Addison-Wesley, Boston, Massachusetts
ISBN 13: 978-0-13-214918-1

Craig Larman: *Applying UML and Patterns: An introductions to object-oriented analysis and design and iterative development*
Published: 30. October 2004
Pearson Education, Inc. publishing as John Wait
ISBN: 0-13-148906-2

# 8 Appendix

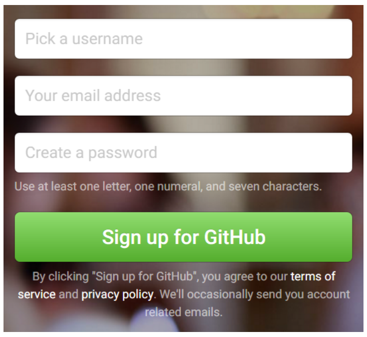
## 8.1 Appendix 1: Vision document / Customer description

Lav et Matadorspil. Der tages udgangspunkt i et normalt matadorspil.
Kunden vil hellere have, at lidt er implementeret og alt virker, end at alt er forsøgt implementeret og ingenting virker.

## 8.2 Appendix 2: Importing Git repositories in Eclipse Java Neon

We have used GitHub as our main code sharing tool - making it possible for us to merge our code and download fixes or updated code from each other. This chapter will guide you to find our git repository and import it to Eclipse Java Neon (from now on just referred to as Eclipse) - therefore if you want this guide to work we would recommend you have Eclipse installed.

0. Install Eclipse (if not installed)
1. Sign up as a user at: https://github.com/ (If not already signed up)



Pick a username

Your email address

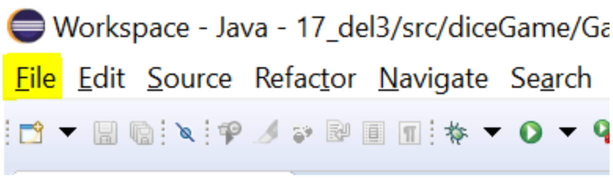
Create a password

Use at least one letter, one numeral, and seven characters.

Sign up for GitHub

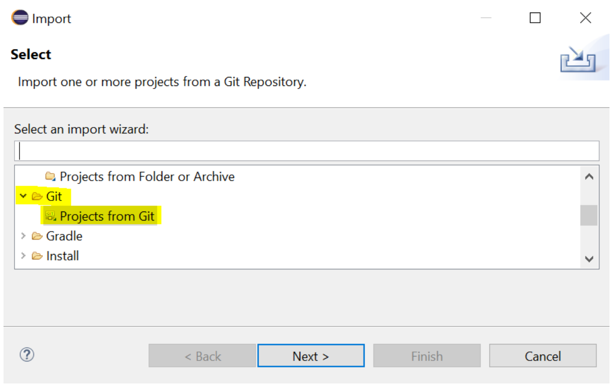
By clicking "Sign up for GitHub", you agree to our terms of service and privacy policy. We'll occasionally send you account related emails.

2. Open up Eclipse
3. Press file (Shortcut: Alt + f) up in the left corner



4. Press Import in the dropdown menu (Shortcut: i when dropdown open)

5. In the new window open up the Git folder and highlight “Projects from Git”



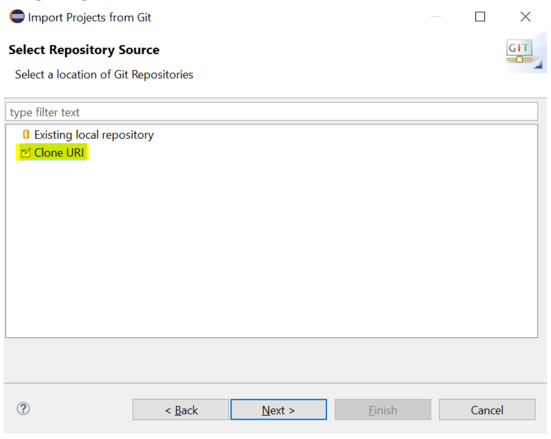
**Select**

Import one or more projects from a Git Repository.

Select an import wizard:

- Projects from Folder or Archive
- Git
  - Projects from Git
- Gradle
- Install

6. Press Next
7. Highlight “Clone URI”

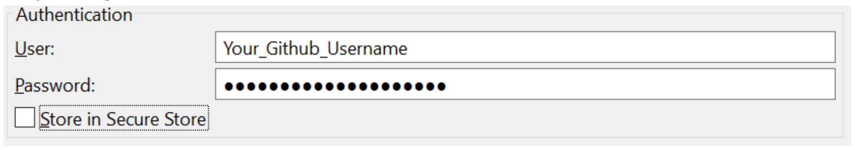


**Select Repository Source**

Select a location of Git Repositories

- Existing local repository
- Clone URI

8. Press Next
9. Copy (Ctrl + c) and paste (Ctrl + v) the following link into the URI input: https://github.com/freyahelstrup/17_final
10. The rest should fill out automatically except for authentication where you should write in your github user information



Authentication

User: Your_Github_Username

Password: ••••••••••••••••••••

☐ Store in Secure Store

   10.1. To skip this step in the future check the “Store in Secure Store” box, this will save your username and password to use in the future

11. Press next until the window disappears
12. You have now imported our GitHub repository to Eclipse

## 8.3 Appendix 3: Source code

The source code for this project can be found in the .zip file that was turned in.