Requirements and Analysis Document for Dominion

This version overrides all previous versions.
Version: 1.0 Date: 20/3
Authors: Christoffer Medin, Pontus Doverstav

1 Introduction

Dominion is originally a card/boardgame, which we are making a digital version of. Our version will offer the same entertainment as the original, but will offer an easier and faster way to play.

1.1 Purpose of application

The purpose of this application is to provide entertainment to the user and deliver a smooth way of playing Dominion with friends and foes over the internet. It will strive to be true to the original game, implementing as many expansions as possible and following all the rules of the original.

1.2 General characteristics of application

The application for the user will be a client that connects to a central server. The server controls all the games and contains the logic. This means that the server does all the computation and the client will merely be displaying the data forwarded by the server in a graphical interface.

The game will be turn based and the active player must take actions for the game to progress. The next player is the next in order. The game will end when the rules state that a game is over.

See references for the rules of the game.

1.3 Scope of application

The application does not support an A.I. The game is meant to be played against other human players over the internet. Statistics are not planned to be tracked.

The game will contain only the base set of Dominion, unless ample time remains when that has been implemented.

1.4 Objectives and success criteria of the project

It should be possible to play a full game containing the base set of Dominion with 2 - 4 human players over the internet. It should also be possible to change some settings locally such as simple resolution settings and full screen.

1.5 Definitions, acronyms and abbreviations

GUI - Graphical User Interface.

View - See GUI.

Java - A programming language

Turn - When a player has the option to play and buy cards. A player can't do anything when it isn't their turn except when the active player plays a card that affect them.

Active player - The player whose turn it is.

2 Requirements

2.1 Functional requirements

The application must be able to:

- Have clients connecting to the server. If a client cannot connect to the server a game may never take place so it is almost the biggest functional requirement.

The application must be able to handle a players cards by:

- Storing cards in deck, discard pile, hand.
- Drawing cards and reusing the discard pile when the deck runs out.
- Handling cards that change zone, such as from hand to playing area, hand to deck etc.

The application must be able to handle a players resources such as:

- Money
- Actions
- Buys

The application needs to be able to handle the playing of cards which includes:

- Taking input from the player to play the card.
- Taking input from the player, when selecting an option presented by the card.
- Handling the players cards.
- Handling the players resources.

The application needs to be able to handle everything a player can do in a turn:

- Playing action cards.
- Playing treasures.
- Buying cards.

2.2 Non-functional requirements

2.2.1 Usability

Usability is very important since anyone should be able to use the application. Most users should not experience any difficulty using the application, however there are a few things required from the user:

- English proficiency

- Basic knowledge of the rules is helpful since the application will not feature a tutorial.

The application will strive to have clear graphical elements and presentation of data to make the user fully aware of what is going on.

2.2.2 Reliability

The application will require a working network since application does not support offline play.

2.2.3 Performance

Performance should not be an issue. Any computer with internet access that is able to run Java should be able to run the application.

The application will not be running any heavy graphical components so the graphical part should not limit the performance. Neither will any heavy computing be done and if such would be made it would be done by the server not the clients.

The application will be slightly limited by the clients response time to the server, however the application does not contain any real-time elements so the extra delay should never be an issue.

2.2.4 Supportability

The application will only support computers limited to those who run a Java version of 7 or higher. However since it is built in Java and the client is reduced to a graphical view modifying it to fit on some other platform should not be too much of an issue.

2.2.5 Implementation

To make the application easier and simpler for the user it is written in Java to ensure platform independency and not limit the user. This requires the user to have JRE installed.

2.2.6 Packaging and installation

Feel free to contact us if you need help with running the application.

Git

The repository can be found at: https://github.com/Glazastik/Dominion.git

Language

The game is programmed with Java 7

Frameworks

- Kryonet - http://code.google.com/p/kryonet/
- Slick2D - https://github.com/joshmarcus/slick2d

Main classes

- server.main.ServerMain.java
- client.main.ClientMain.java

2.2.7 Legal

Dominion is a copyright owned by Rio Grande Games.

2.3 Application models

2.3.1 Use case model

For a more comprehensive list, complete with links to each use case, please consult the summary

List of use cases: * Buying Cards
* Creating a game room
* Discard card(s) * Draw card(s) * Draw hand * Gain card(s) * Joining a game room * Play action card * Play treasure card * Starting the application *

Taking turn * Trash card(s)

2.3.2 Use cases priority

For a more comprehensive list, complete with links to each use case, please consult the summary

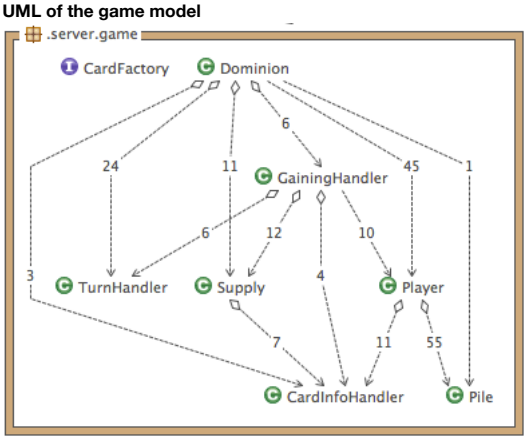
UC	Priority
Buying Cards	High
Creating a game room	High
Discard card(s)	High
Draw card(s)	High
Draw hand	High
Gain card(s)	High
Joining a game room	High
Play action card	High
Play treasure card	High
Starting the application	High
Taking turn	High
Trash card(s)	High

More UC to come

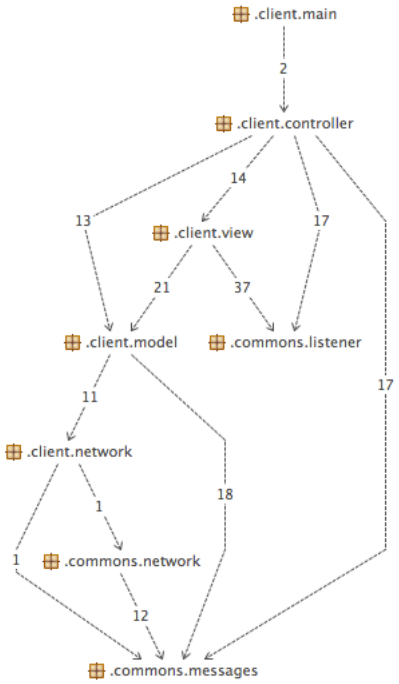
The most prioritized use cases are (in order): 1. Joining game room 2. Draw cards 3. Play cards 4. Buying cards

2.3.3 Domain model

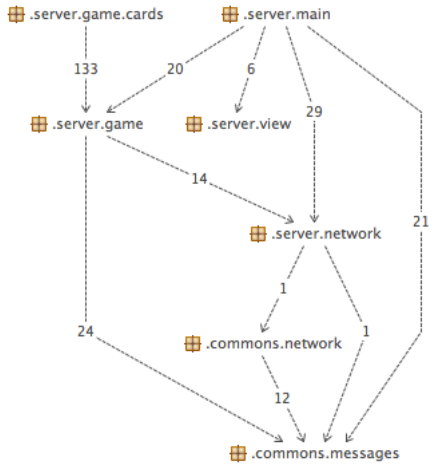
UML of the game model



UML of the client



UML of the server



Class summary:
RoomHandler
-A class solely supporting the server. -Handles rooms with clients.

Room
-A room the client joins before the game begins -Starts the game

GameModel
-The game model -Has a controller (GameController) -Has handlers(support classes)
-Has player objects that handles money, the players cards etc.

GameController
-Listens on the network and controls the model

Client (Serverside)

-How the server sees a client

-Is a socket that the client connects to

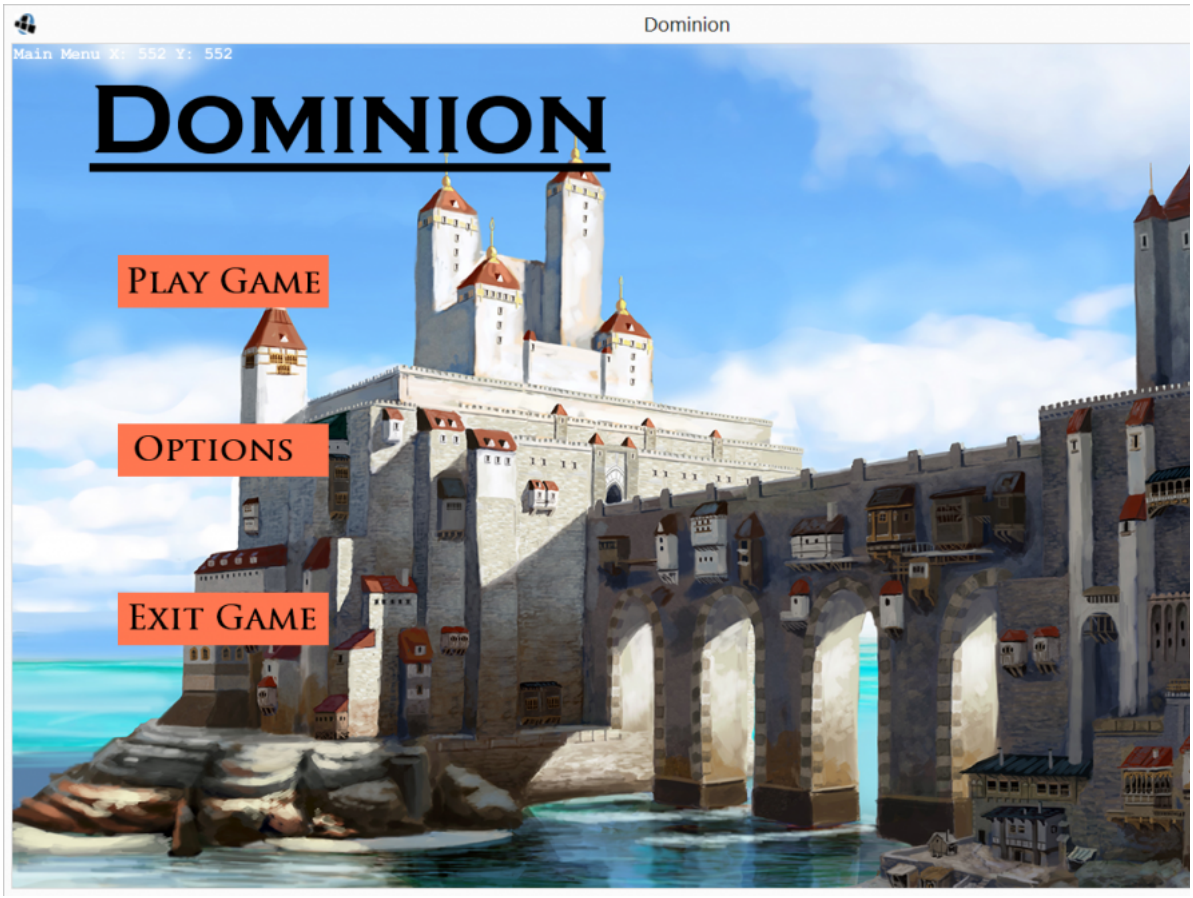
ClientHandler
-Handles clients -Support class to the server

Client (Userside)
-An instance of the client -Is solely a view and controller. -Has some user settings.

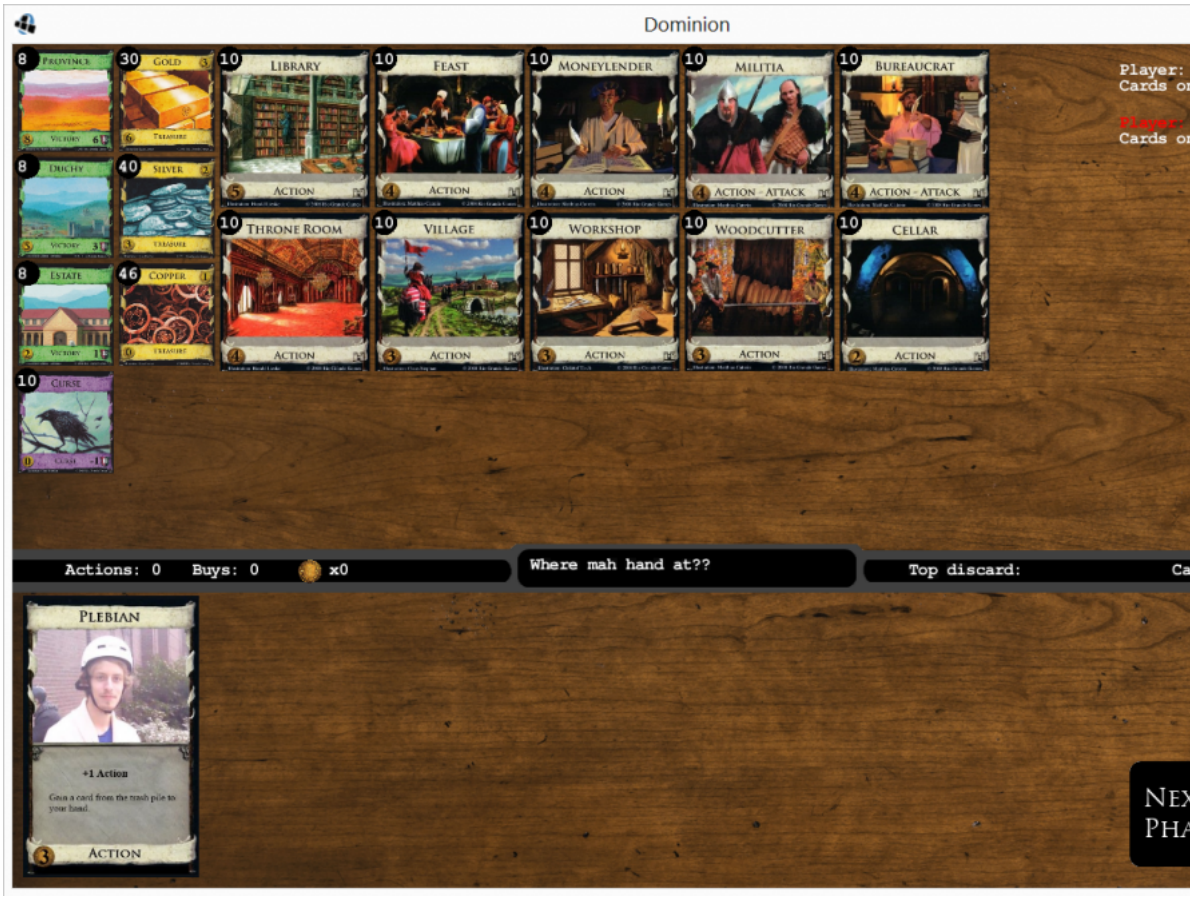
Views
-The clients different views to display menus, the game, settings etc.

2.3.4 User interface

Main menu

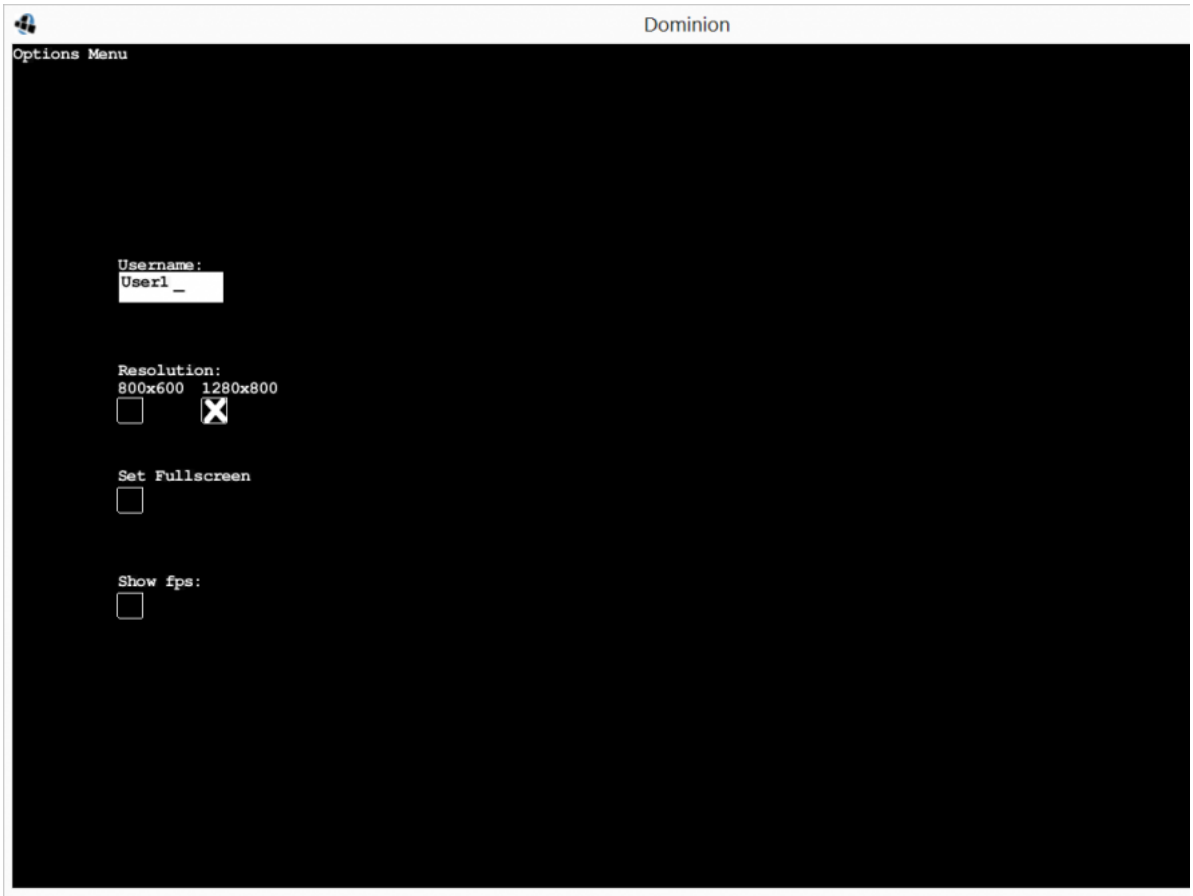


Ingame view


Dominion
Player:
Cards o
Player:
Cards o
Actions: 0
Buys: 0
x0
Where mah hand at??
Top discard:
Ca
Plebian
+1 Action
Gain a card from the trash pile to your hand.
Action
Nex
Pha

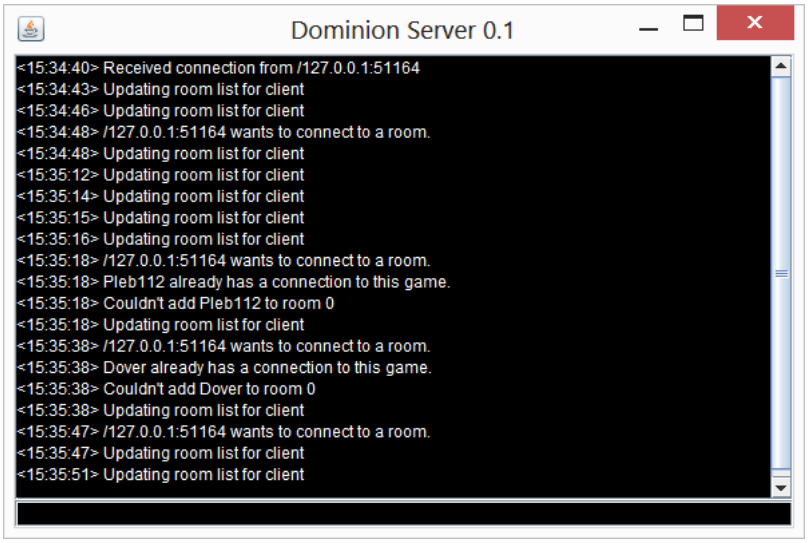
Lobby view

Options view



The above pictures are part of the client and are rather self-explanatory. The first screen that greets a player is the main menu. Every view of the client has been kept simple and straight-forward, aiming to be intuitive to use and easy to understand. This is so that a player that has knowledge of the rules of Dominion doesn't need to go through a tutorial to learn how to use our game, but can instead jump right in. Features and options are kept at minimum to achieve the same goal as above, but still to make sure that the games is playable for every user.

Server console



The console mostly serves as output for the servers internal workings. It reports events such as people joining rooms, new rooms being hosted and players disconnecting.

2.4 References

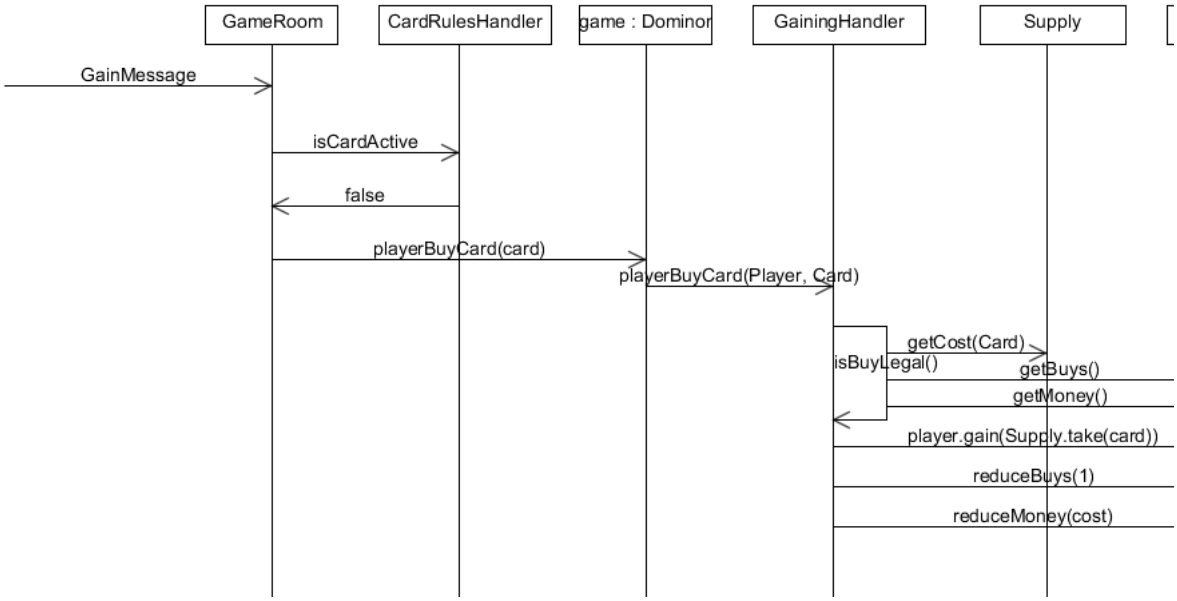
APPENDIX

1 Use case texts

For a shorter summary, complete with links to each use case, please consult the summary

1.1 Buying Cards

Sequence map of buyCard()



Normal flow of events:

---

1 Client: Highlight cards that are available for purchase
2 User: Choose which card to purchase
3 Client: Removes one from the supply pile of the selected card and puts it in the users discard pile

---

Alternate flow 2.1 Multiple buys

1 Client: Highlight cards that are available for purchase
2 User: Choose which card to purchase
3 Client: Removes one from the supply pile of the selected card and puts it in the users discard pile
4 System: Repeat first 3 steps, if user not done buying. (Additional buys)

Alternate flow 3.1 User doesn't want to buy anything

1 Client: Highlight cards that are available for purchase
2 User: Press "End turn button"
3 System: Ends turn

1.2 Creating a game room

Normal flow of events:

---

1 User: Starts a game room
2 System: Creates the game room and adds the user to it
3 System: Adds the game room to the room list

1.3 Discard card(s)

Normal flow of events

---

1 Actor: Choose which card to discard
2 System: Remove chosen card from hand, add chosen card to discard pile

---

Alternative flow 2.1 No cards to discard

1 System: Nothing special happends, moves on.

1.4 Draw card(s)

Normal flow of events

---

1 Actor: Draw x Cards
2 System: Remove x cards from deck, add x cards to hand

Alternate flow 2.1 No cards in deck

1 Actor: Draw x cards
2 System: Shuffle discard pile into deck
3 System: Remove x cards from pile, add x to hand

1.5 Draw hand

Normal flow of events

---

1 System: Remove 5 cards from deck, add the 5 cards to hand.

Alternate flow 2.1 Not enough cards in deck

1 System: Draw remaining cards in deck. 2 System: Shuffle discard pile into deck. 3 System: Remove card from deck and add it to hand until 5 cards in hand.

Alternate flow 3.1 No cards in deck

1 System: Shuffle discard pile into deck. 3 System: Remove 5 cards from deck, add the 5 cards to hand.

1.6 Gain card(s)

Normal flow of events

---

1 Actor: Choose which card to gain, according to action card.
2 System: Remove chosen card from buy stack and add to discrad pile.

Alternate flow 2.1 No cards available for gain (no legal targets)

1 System: Nothing happends, skips gain action.

1.7 Joining a game room

Normal flow of events :

---

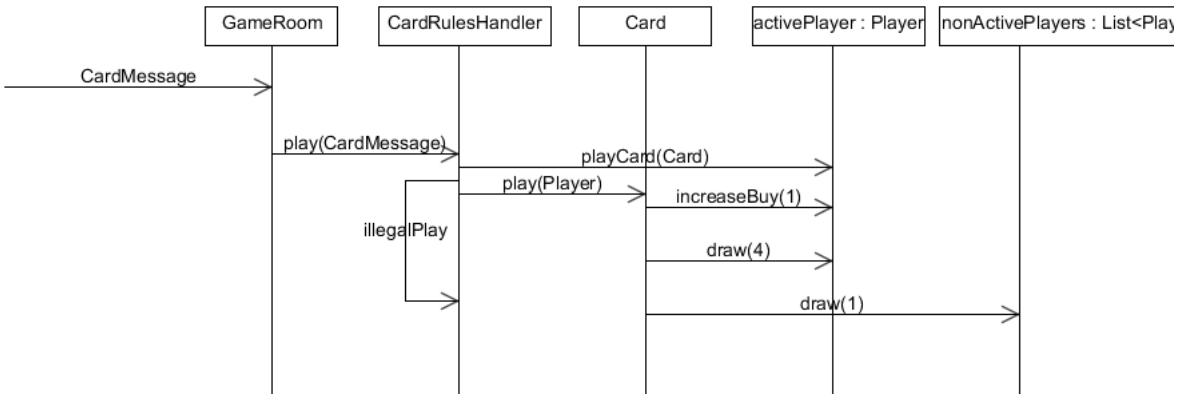
1 User: Joins a game room
2 System: Checks if there is space and if there is places the user in the room
3 Client: Displays the game room

Alternative flow of events 2.1 Game room is full

1 User: Joins a game room
2 System: Notices the game is full and rejects the action
3 Client: Displays an error message

1.8 Play action card

Sequence map of play(card)



Normal flow of events

---

1 Actor: Chose action card to play
2 System: Reveal chosen action card and put in play zone
3 System: Resolve card effect, ex: draw card, inc buy, +2 coins
Repeat 1-3 if possible and actor wants.
4 Actor: Move on to next phase

1.9 Play treasure card

Normal flow of events

Play all treasure cards
1 Actor: Press "play all treasure cards"
2 System: Reveals and moves all treasure cards from hand to play area.
2 System: Update available $

Alternate flow 2.1 Play treasure cards manually

1 Actor: Choose treasure card to play
2 System: Reveal and move chosen treasure card to play area.
3 System: Update available $
Repeat 1-2 until satisfied

1.10 Starting the application

Normal flow of events

---

1 User: Starts the application
2 Client: Starts up the graphical view
3 Client: Connects to the server

1.11 Taking turn

Normal flow of events:

---

1 Playing action cards
2 Playing treasures 3 Buying cards
4 Discard all cards on hand
5 Draw five cards
6 Ending turn

1.12 Trash card(s)

Normal flow of events

---

1 Actor: Choose card to trash
2 System: Remove chosen card from hand put into trash pile

Alternate flow 2.1 No cards to trash

1 System: Nothing happens, moving on