Dominion

Analysis, Design and Software Architecture

Jakob Melnyk, jmel@itu.dk
Christian Jensen, chrj@itu.dk
Frederik Lysgaard, frly@itu.dk

December 14th, 2011

Abstract

This project is about a virtual representation of the card game Dominion in C#. Dominion is a turn-based, deck-building game, where the objective is to gather more points than the other players. The game is played by 2 - 4 players.

Contents

1 Requirements

1.1 Mandatory

Must be able to play a full game of Dominion

- Must support 2 players in Hot-Seat configuration
- At least 10 Kingdom cards must work
- The game must be playable in a Picture-based GUI

1.2 Secondary

High priority

- Be able to play the game with 3 or more players
- Be able to use at least 20 Kingdom cards
- Be able to select Game Mode
  - Be able to play 'First Game' Card-set
  - Be able to play with 10 randomly select Kingdom cards
- Be able to see all Available Kingdom cards without scrolling

Medium priority

- Be able to view a Tooltip when mousing over any Card in the game
- Be able to play the game over LAN
- Be able to use all Kingdom cards (from the original version of the game)
- Be able to play all the Card-sets defined in the original rules

Low priority

- Be able to Draft Kingdom cards
- Be able to play the game over the Internet
- Be able to select different screensizes
- Be able to play in fullscreen

- Be able to create a User, that is saved across multiple games, with the following information:
  - Statistics
  - Options (if any)
  - Achievements (if implemented)
- Be able to support Extensions of the basic game
- Implement Achievements for funny and/or hard accomplishments

2 Overview

2.1 About

This project is about our virtual representation of the card game Dominion. Dominion is a turn-based, deck-building game. The objective of the game is to use Action cards to improve your chances or damage the opponent players and using Treasure cards to buy more powerful Action/Treasure/Victory cards to gain the upper hand.

Our version control repository is at `https://github.com/esfdk/BDSADominion` and is public.

Requirements that are fulfilled:

- All three mandatory requirements.
- Able to play with three or four players.
- Able to play the game over LAN.

Section 4 on page 13 walks through it in more detail.

2.2 Team members

Frederik Lysgaard is the guy responsible for the design of our graphical interface. He designed the interface using XNA, which he learned during the project. He is also the best Dominion player in our group. Because of this, he knows a lot of the usual strategies and is our general "go-to" guy when it comes to the tactics of the game.

Christian 'Troy' Jensen is our networking guy. He set up all of the networking with a server-client architecture and made it interface well with Control. He also did the proxy interface between the GUI and Control, so that the GUI could be replaced and Control would never know.

Jakob Melnyk is responsible for modeling the state of the game and the communication between Control and the Model (Gamestate), Control and GUI (the proxy-interface that Christian developed) and between Control and some parts of the network interface. Jakob is also the "version-control-guy", the person with the final word in discussions and the general log-keeper.

2.3 Architecture

We have used a Model-View-Control architecture to do a heavy separation of concerns in terms of the GUI, server, model and game logic. We planned on having a static model outside of the controller to contain the game logic (in terms of card attributes), but we decided to do it more simply in the controller and then focus on getting networking in the game.

The networking is done with a server-client architecture where the server has a number of clients and one or more of these clients can be itself. Currently we have only tested it on the server's local machine and on a LAN with the server. It should be possible to play over the Internet, but we have not tested it to any degree and as such cannot make any promises that it will work.

In the "Related Documents/Dependency Graphs" we have included dependency graphs for the Card Inheritance hierarchy, Control's dependencies and Gamestate's dependencies.

2.4 Known bugs

- It is very hard to click on the bottommost field in the Supply in the GUI.
- Sometimes a player will not be shown on the GUI at the end of the game whether he has won or lost the game. It is however shown in the console.
- While the hand can contain more than seven cards, it is not possible to see and use cards beyond the seventh card in the hand.
- If more than nine cards are played in any given turn, the tenth card and beyond will be drawn outside of the game window.
- The number of cards in the supply does not vary according to how many players are in the game.

2.5 Validation

2.5.1 Code Contracts and Pex

We have been debugging with Code Contracts enabled, but we do not have them enabled for the released version. We used contracts to not only have contracts on our classes, but also so that we could use Pex to generate some unit tests for us. The main amount of the contracts are in the Gamestate namespace, but there are a few contracts elsewhere in our project.

Pex could not generate as many unit tests as we would have liked within the time we had, but it covers about 8% of our code on its own. dotCover html report file of only the Pex tests can be found in "BDSADominion/dotCover Coverage Runs/Pex solitaire snap.html".

2.5.2 Screencast

We did a screencast of ourselves playing a couple of turns at the start of a game and of a player winning the game. These are called ”BDSADominion - Screencast pt 1” and ”BDSADominion - Screencast pt 2”, respectively. These can be found in ”Related Documents/DominionScreencast”.

Because we play a few rounds and the game continues doing the same turn-based play, we feel that showing that a couple of turns work can be used as validation for the functionality of our game. Because we also show that a game can be won in the second screencast, we have shown that the game can end.

2.5.3 Code Coverage of a game

While we have not done scenarios, we have instead played the game many times and so, in a sense, have validated it to some degree.

Besides the Pex code coverage, we also ran dotCover with ”Cover Startup Project”. This checks how much of the code is covered during the run of the Project’s Start.

The most code coverage we got in any single run using this way of covering code was Christian running his game as a server and then joining another client on his server from his own machine. The results of this can be see in ”BDSADominion/dotCover Coverage Runs/TroyServerRun1.html”. This report covered a lot of the code from the Connection and Server classes.

Another report merged from a solo player running the game ”BDSADominion/dotCover Coverage Runs/Merge_Report.html”, the Pex results and the game being run as a client on another computer, did not really do so well in covering the network area, but covered the other parts of the application better than ”TroyServerRun1” did. We should be able to reach at least 85% code coverage by our calculations.

2.6 Code Metrics

We did Visual Studio’s code metrics on our project and the exported CodeMetrics.xlsx (Excel file) can be found in ”Related Documents”. Of note in the code metrics is our Player class with the three methods with the highest cyclomatic complexity in the project (53, 48, 43). These are methods for adding and removing cards from a Player, which in turn make them heavy on contracts, which could explain the cyclomatic complexity.

2.7 Notes

- Given more time, we would definitely have used BON to better describe our architecture.

- We could have added more and better contracts, given more time. Especially invariants could have been done in more classes.
- We could probably have done without the inheritance as it looks now, because we never use the fact that something is a specific card, only ever its type (Action, Treasure, etc.)

3 Dictionary

3.1 General terms

This section describes the general ”out-of-game” terms.

Achievements An achievement is token rewarded for funny and/or hard accomplishments within the game.

Card-set A card-set is 10 different Kingdom cards. Card-sets are used to create a different play experience every time you play.

Dominion The card-game we are making a virtual representation of. A link to the full rules can be found at Rio Grande Games [2].

Draft Drafting is done by player 1 selecting one Kingdom card to be used in the game, then player 2 selects a Kingdom card, player 3 selects a Kingdom card, player 4 selects a Kingdom card, then back to player 1. This cycle repeats until a set number of Kingdom cards have been selected.

Extensions Expansion packs add additional types of cards to the pool of cards.

Game Mode There are different possible game modes: draft, random card selection and predefined card-sets. These are selected before the game starts.

Hot-Seat Hot-Seat is the act of having 2 or more players play on the same computer. The active player ”sits” in the hot-seat while playing, then passing the spot to the next player when his turn ends.

Message Type Messages of different types can be passed around in our server-client network.

Model-View-Control Often abbreviated MVC, Model-View-Control is often used to seperate something ”showing” data and the actual representation of the data on the disk. Control is usually the middle-link that takes care of the communication between the two.

Picture-based GUI A pictured-based GUI is a visual representation of the state of the game. The different cards are shown as pictures in the GUI.

Server-Client In a client-server design, the clients communicate with the server and the server then relays the information it was given by the client to the other clients.

Statistics Statistics such as number of games played, numbers of games won/lost, and other similar data about gameplay.

Tooltip A box with text describing something in the GUI in detail.

User A user is an entity storing statistics and achievements over the course of different games.

3.2 In-Game terms

This section describes the types of cards, supply and other "in-game" terms.

Available Available Cards are the Cards that can be bought from the Supply.

Action Phase In an action phase, a player have one Action, which he or she may use to play an Action Card. Playing an action card this way always costs one Action. Cards played may allow a player to receive additional actions. The Action Phase ends when a player has no more Actions left or chooses not to use his or her remaining Actions.

Buy Phase When a player's Action Phase ends, the Buy Phase begins. In this, the player receives a "Coin" amount, which is the combined value of all Treasure Cards in his or her hand and any Action Cards, that add "Coins". The player can then use a Buy to buy any Card they want from the Supply. Played Action Cards can allow more Buys. Bought Cards are added to the Discard Stack. After the Buy Phase, the Clean-Up Phase begins

Card A Card is the basic playing unit in Dominion. Everything you 'own' is represented by a Card in your deck. Cards are primarily added to the deck through the Buy phase. Each Card has a value, which represents what it costs to get during the Buy Phase.

- **Curse Card** A Curse Card is a special type of Victory Card, which gives a negative amount of Victory Points. While these cards can technically be bought by any player and added to his or her deck, they are usually given to other players by using Attack Cards against them.
- **Kingdom Card** Kingdom Cards are what make each game of Dominion unique. With one exception all Cards here are Action Cards (one is a special Victory Card) and there are no Action Cards which are not Kingdom Cards. Each game requires selecting 10 of the 25 Kingdom Cards to use.
  - **Action Card** An Action Card is used during the Action Phase.
    - **Attack Card** An Attack Card is a type of Card which affects other players negatively. All Attack Cards are Action Cards and the "Attack" actives when the Card is used as an Action.
    - **Action-Reaction Card** A Reaction card is used to respond to an Attack by another player. When an Attack Card is used against a player, that player may reveal a Reaction Card from his or her hand and do what the Reaction allows. Only one Reaction Card is in this game, 'Moat', which allows the player to negate the attack used against them.
  - **Kingdom Victory Card** A Kingdom Victory Card is a card that does generally not behave like usual Victory Card, but instead have special effects granting the player Victory Points.

Treasure Card A Treasure Card adds a number of "Coins" to spend in the Buy Phase. Note that a Treasure Cards value (the price to buy it) are usually different from what they cost to buy.

Victory Card A Victory Card gives a number of Victory Points at the end of the game. The player with the most Victory Points win the game.

Clean-up Phase The Clean-up Phase consists of putting all bought Cards, played Cards and Cards remaining in the Hand into the Discard Stack.

Deck A players Deck is his or her representation in the game. It consists of all the Cards that player started with and have bought during the game. A player's Draw Stack, Discard Stack and Hand is that player's Deck.

Discard Stack This contains previously played cards and any newly bought cards.

Draw Stack This contains face-down Cards for a player to draw. When there are no more cards available for a player to draw, the Discard Stack is shuffled and used as a new Draw Stack. Each player have their own Draw Stack and Discard Stack.

Hand The Hand represents a players current options in the following turn. These are drawn at the start of the game and each player draws a new hand after a turn has finished. When drawing a new hand, it always consists of 5 Cards.

Supply The Supply consists of 10 types of Kingdom Cards, 3 types of Treasure Cards, 3 types of Victory Cards and Curse Cards.

Round A game of Dominion consists of a number of rounds. Each Round is divided in to Turns, one for each player.

Trash Stack Sometimes a Card calls for itself or some other card to be Trashed. This means that it should be completely removed from the game and the Trashed Card is put on to the Trash Stack. All players share the Trash Stack.

Turn The player usually take turns in clockwise order. A players next Turn will be in the following Round.

4 Example

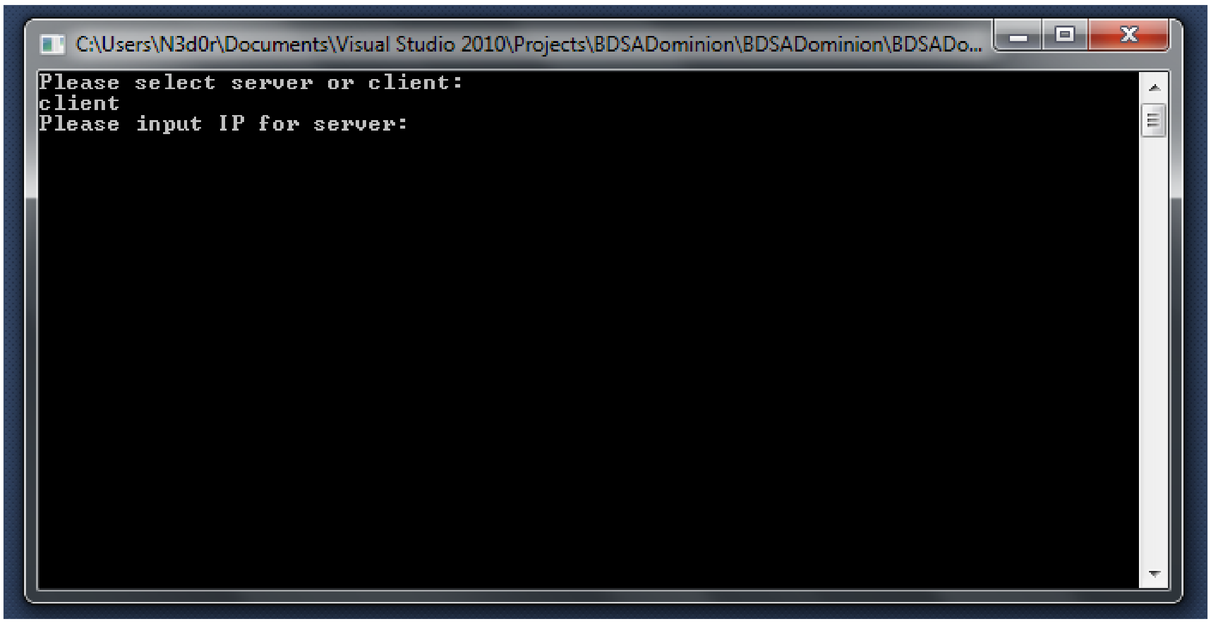
Frederik Roden Lysgaard This will be a example of our project, which is a graphical representation of the cardgame "Dominion", published by Rio Grande Games. The walkthrough will be built up around certain screenshots and will cover the following points:

- Starting the game, hotseat or LAN.
- Getting started, what is Dominion really about?
- The user interface.
- End of game.

4.1 Starting the game,hotseat or LAN.

4.1.1 LAN

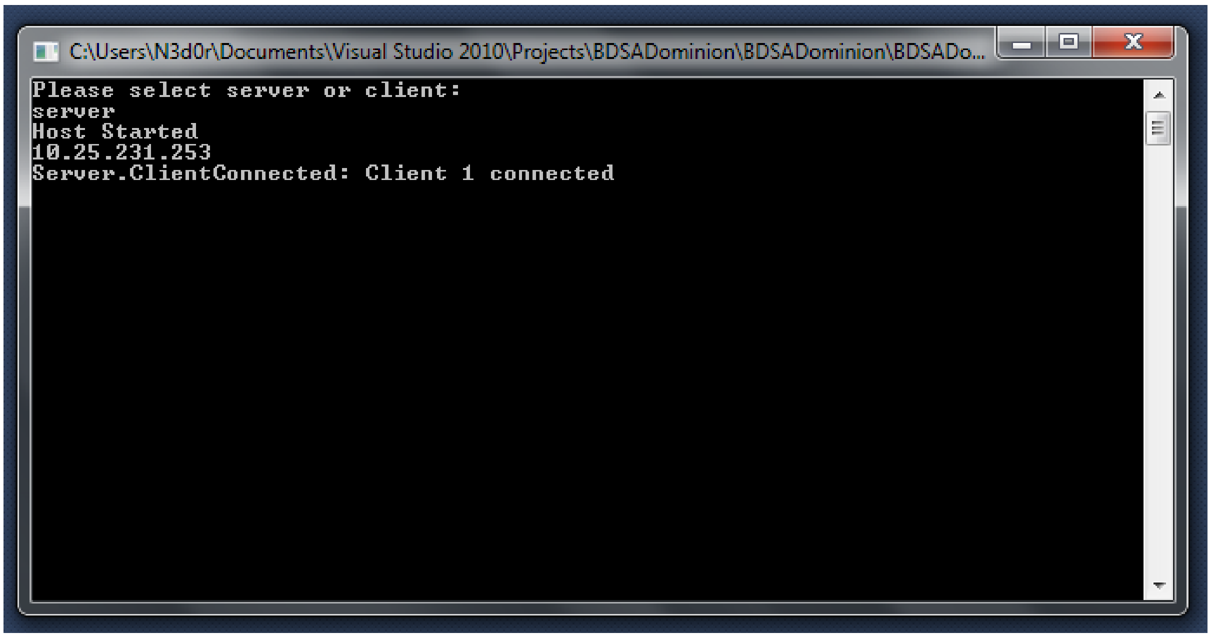
When starting the application you will be presented with a console window asking you to take the role of either Client or Server


C:\Users\N3d0r\Documents\Visual Studio 2010\Projects\BDSADominion\BDSADominion\BDSADo...

```
Please select server or client:
client
Please input IP for server:
```

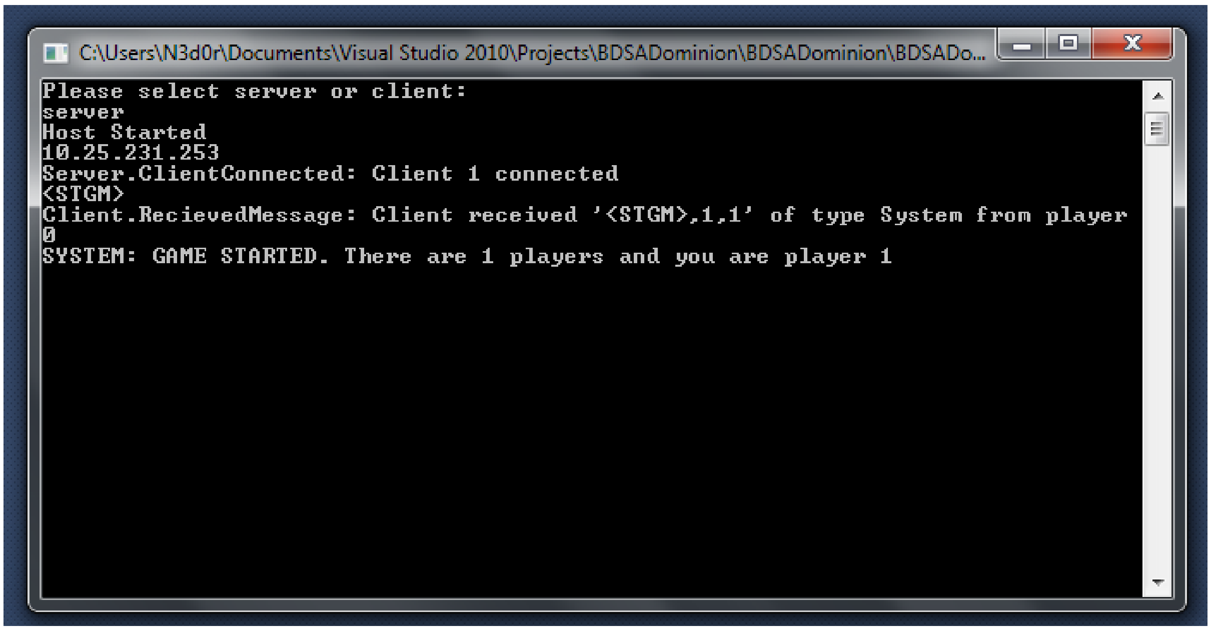
Figure 1: Client input

One of the players will choose server and will then be able to give his fellow players who chose client, an IP to connect to. When the appropriate amount of players has joined the server (usually 3-4) then the person with the server program runing will call startgame (command is <STGM>) and the game will then start.



```
C:\Users\N3d0r\Documents\Visual Studio 2010\Projects\BDSADominion\BDSADominion\BDSADo...
Please select server or client:
server
Host Started
10.25.231.253
Server.ClientConnected: Client 1 connected
```

Figure 2: start server



```
C:\Users\N3d0r\Documents\Visual Studio 2010\Projects\BDSADominion\BDSADominion\BDSADo...
Please select server or client:
server
Host Started
10.25.231.253
Server.ClientConnected: Client 1 connected
<STGM>
Client.RecievedMessage: Client received '<STGM>,1,1' of type System from player
0
SYSTEM: GAME STARTED. There are 1 players and you are player 1
```

Figure 3: start game

4.1.2 Hot-Seat

This is done almost identically to the procedure for a LAN game the only exception is, that instead of letting other computers create a client, you just run multi instance of the application on your computer like so: and after that it's just that same as with LAN games.

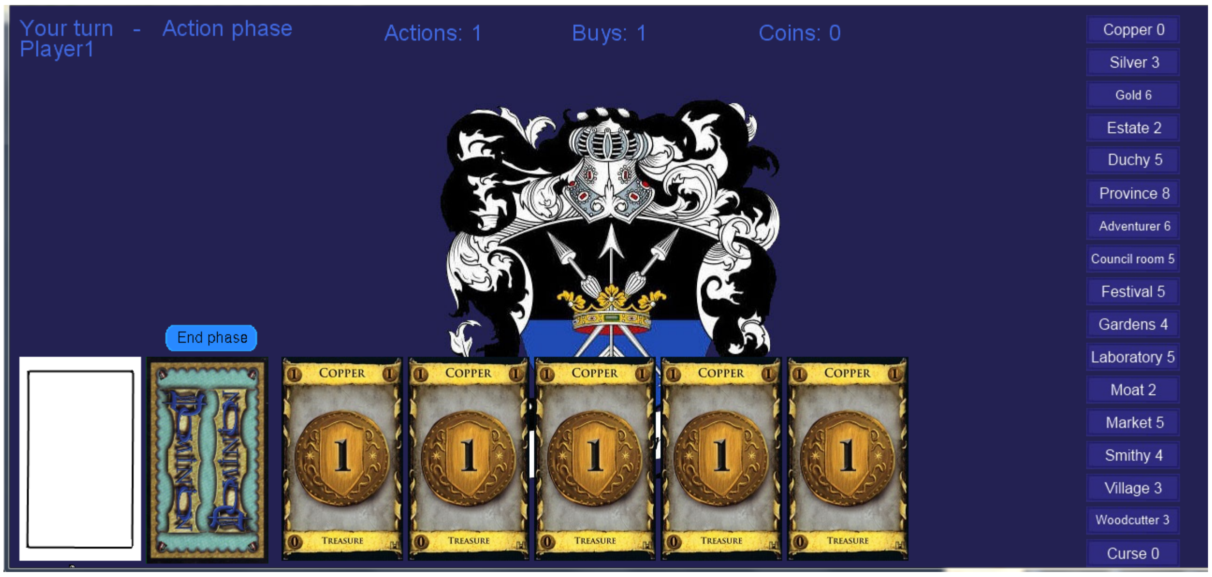
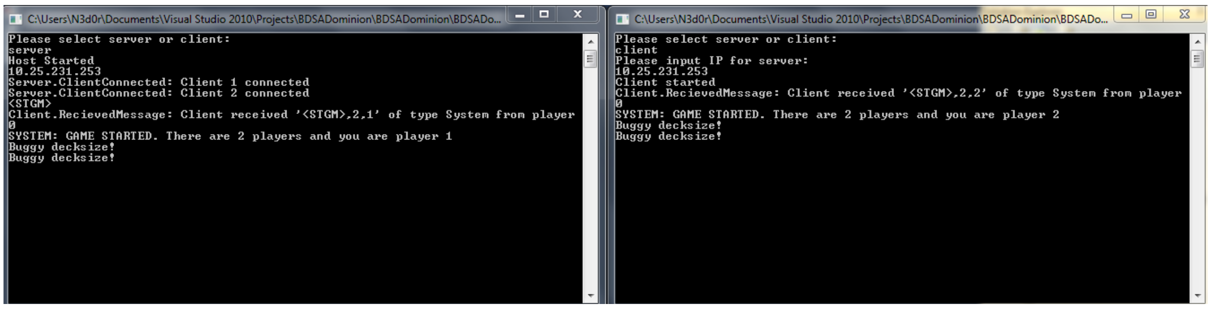


Figure 4: The start screen


C:\Users\N3d0r\Documents\Visual Studio 2010\Projects\BDSADominion\BDSADominion\BDSADo...
C:\Users\N3d0r\Documents\Visual Studio 2010\Projects\BDSADominion\BDSADominion\BDSADo...

```
Please select server or client:
server
Host Started
10.25.231.253
Server.ClientConnected: Client 1 connected
Server.ClientConnected: Client 2 connected
<STGM>
Client.RecievedMessage: Client received '<STGM>,2,1' of type System from player
0
SYSTEM: GAME STARTED. There are 2 players and you are player 1
Buggy decksize!
Buggy decksize!
```

```
Please select server or client:
client
Please input IP for server:
10.25.231.253
Client started
Client.RecievedMessage: Client received '<STGM>,2,2' of type System from player
0
SYSTEM: GAME STARTED. There are 2 players and you are player 2
Buggy decksize!
Buggy decksize!
```

Figure 5: hotseat, server-client

4.2 Getting started, what is Dominion really about?

Dominion is a deckbuilding game which means, that the object of the game is to build yourself a deck which will give you the best hands, and thereby giving you the edge in getting the most victory points which in the end determines who wins. I already introduced some of the game specific words and I will now show where they are placed on the playing board and what their responsibility is:

4.3 The user interface

hand The hand is where you see what you have drawn each turn.In Dominion there's three kind of cards: Treasure, Victory or Action all three kinds can be drawn into this field. If you click on a Action card while it's placed in hand and you got actions left then the card will be moved from the hand to the actionzone.

actionzone The actionzone is where the actioncards that is played from the hand is

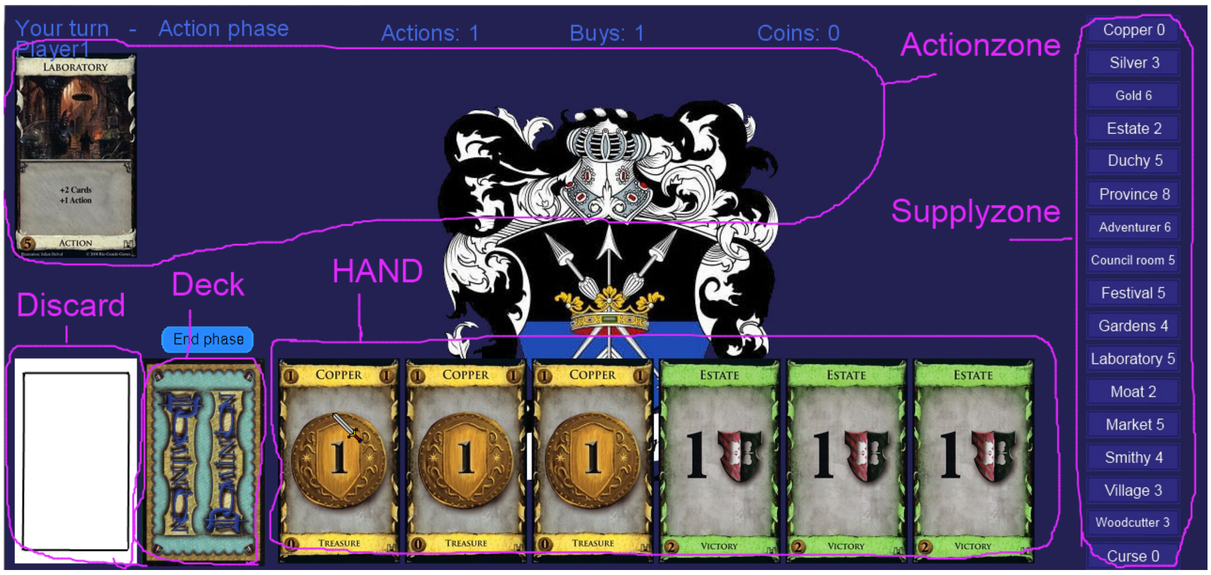


Figure 6: The playing board

shown. Only actioncards can be drawn in this zone. When a turn ends the action-zone will be cleared and the actioncards will all be moved to the discard zone.

discard The discard zone is where the cards go when they are not in use anymore, you can't click on cards while they are in the discardzone, while in the discardzone the cards can only wait to be shuffled into the deck again.

deck The deck is where the cards is held until they are drawn.

supply The supplyzone that are drawn to right side of the GUI is crucial to the game. This is where you can buy your kingdom cards and thereby increase your decks size and strength, as we can see, we draw both the seven static victory/treasure cards and 10 extra kingdomcards, which are also one of our mandatory requirements.

4.4 End of game

As stated by the Dominion game rules the game ends when either the Province card pile is empty or three kingdomcard piles are empty. When this happens we check which player has won. If you are indeed the winner then you will be greeted with a lovely congratulations message printed across the screen. but if you lose you will be met with disgust. So to summarize: Our digital representation of Dominion lets you play with 10 preset kingdom cards in a graphical interface with your friends either over hotseat or local area network.

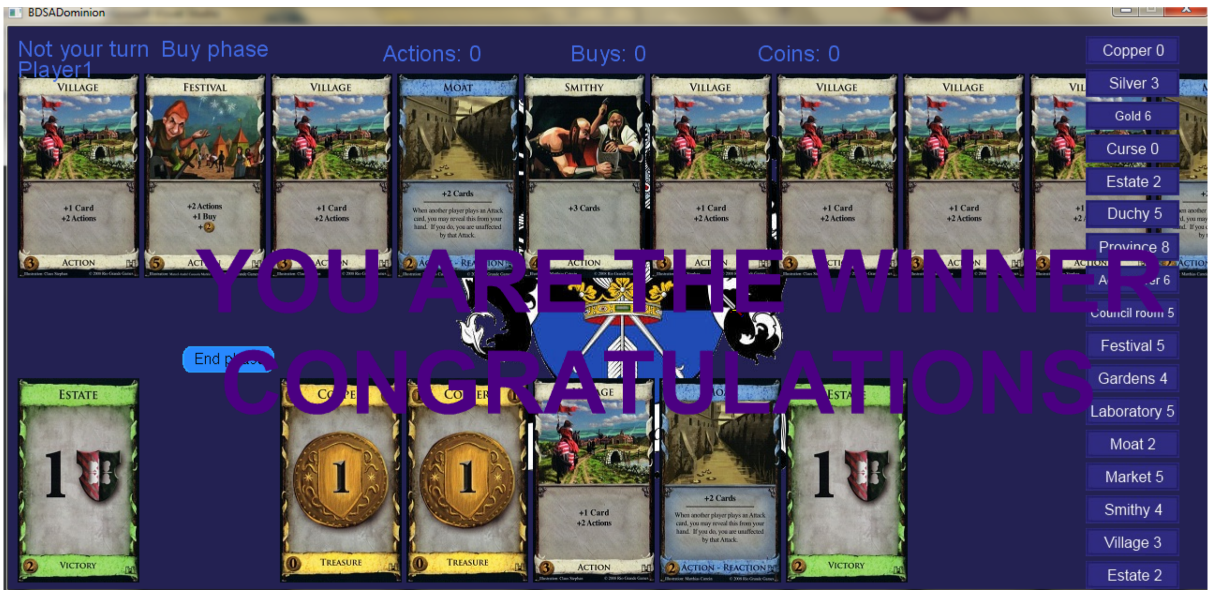


Figure 7: You are the winner!!

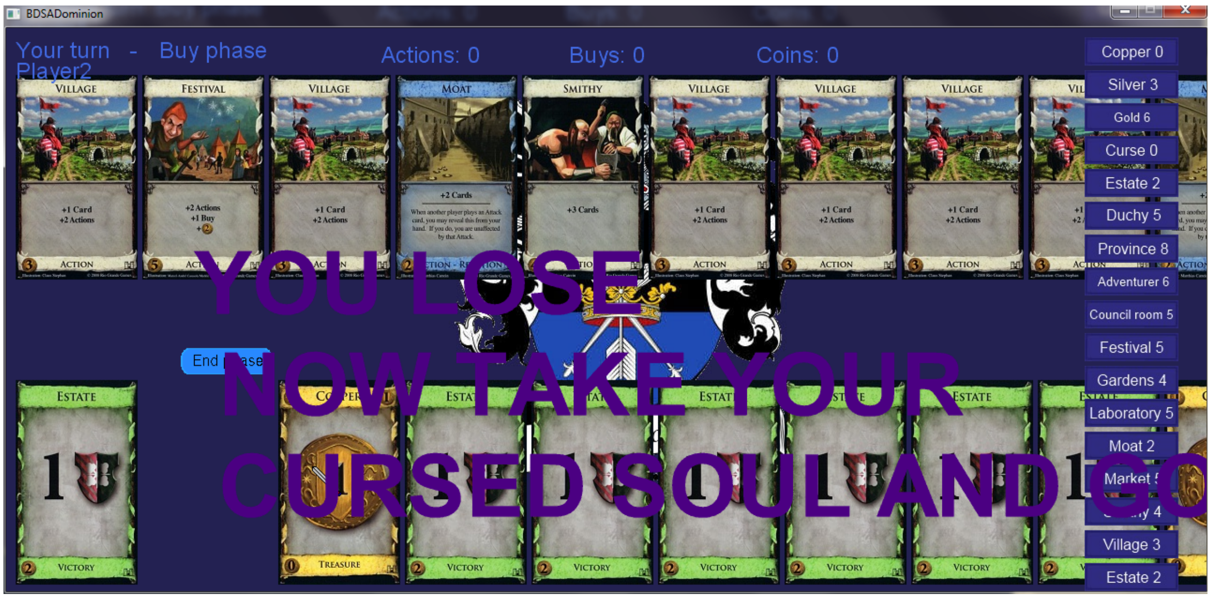


Figure 8: You loose!!!

5 Revision History

Some of our commits to our Github (`https://github.com/esfdk/BDSADominion`) with commit ID. The commits are sorted by time of commit, descending. Full commit log can be found at: `https://github.com/esfdk/BDSADominion/commits`

——————————————-FINAL COMMIT BEFORE HAND-IN——————

9bd4c41cf7f113b63175b338ec9d7b825e3db930 : Merge updated

318693d7f4543b3d50f105cee080a0d488e268c3 : Did system analysis

71ecb6c89c4ab14c426a904f4d308c3f696c42a2 : Did my System design and production and some history

19af7f669a71e5c262d5eaab540f1b51a591089f : SolitareRun added, code coverage looking good

e92c7c5381f65ca8f3deef9458922373c4da81de : Put BON files into Hand-in.

940473cec8b897f58774415fa99eb6fd50771b07 : Merged dotCover file added. System design and production of network added.

————————————————— Code freeze ——————————

9d86882ca255502d2cd07bdfbc13e84b13d50fb8 : Lots of work on hand-in getting done.

c2dcc6786210ecf5ac74a8d34bcf07e72cf8360c : pex snapshot

a397b2b79291e8349e106ef1773406e3c3a57e55 : made a dotcover solitaire run

cae05ae98caefbeaf2cf31fa4eb9473507486ce1 : contract

00782159d83baf89bf3a41acea7f87d6484e263f : network design done.

28cb2b5c8f343793662fa0cec0d8dae0e8476564 : Fixed bug in shuffling of cards. Not as random as it was before.

234ca16500ee52cbf8f37dff117de83c07febebe : informal is now legit

e22f402855a48cc2ce93ce651305bb801e121d75 : Key and control update

fe7e9ba83b41594a5ef91560af0fb505a55da627 : Some pex tests, better shuffle method, fixed adventurer and more.

3074885da9f22646bac0becbfe8bc45ae975723a : Made some changes to GUI it looks good now

e083766fe15b127f0a9ae03528e1eea956a6702e : dotCover solo run added

1951adb1e07b4d9735847e0bf3e622464380aa2f : Trying to get these weird bugs out of the game.

7c53affdf9aba583efc71dedab9476fe1441eeea : Done with some of the docs for the GUI

1173a01763032deb6b033d01471039be84a61f0f : Wrote some of system design and started on system production

a35c4939236770661fb751f378be4c7e5259d0fe : Formal bon 99% done for Melnyk.

97ae13e744b2283f0a0aecafae954b357d0e2c04 : Minor fix in Control to possibly prevent crashes. Also added some more Contracts to the formal BON.

aa1c0af808bc536cc50e5e988e185834ca34bd13 : GUI start update working now

9d4d9b2bcf0b8acda2544fd3c93ad926122237c4 : added the listeners

44f6bb367f5b56014069c2bfd8b1e8fd1a61da7c : Made files to be used in the hand-in.

911af507af6ec9e64a36d470280d1b4d7e9707d7 : Network starts the game

b7210739f2ed0b37875ab7d4a858559602e029b2 : Server PreGame messaging working. Still lots of WriteLines.

9a3c7bfc2334e04a0709a09eeb0d1b4a532327a7 : Much of Melnyk's part of Control is done! Mainly need communication with server!

3417bf70282d5b3bdfd39c8855f258eeb05e57a2 : Network seems to be working alright.

c710a8d4a93dfdc6623be4022bcb8da2a0259d4d : GUIInterface done. Network testing

bb1b5dbabccfc6c7d3c1d25ffa97fcf5e8d37b64 : Starting tests on network interface

537f2f25db57b263d63cac1ab298a69de5e80bf3 : GUI-Interface almost done, only need endPhase button in gamestate.

9f2a6f20d46e0037a2dfa2d917b3030b720551c3 : GUI Interface getting up and running. Console added :D

a883a6fd4d552fe71cb3aae641a7ec0af4037e06 : GUI now draws what needs to be drawn

7c368444740ed2bc8d49027e7a7d666674600ec2 : Finished gamestate and player for now

24454e804bde99435b2022244e6307d89a9c323a : GUI now updated to work with Card-Name

1ddb9247acd2f32b0e979f7dd561bd98edf4d570: Alfa GUI is done. LOCK AND LOAD!

5c28c48e556fea810b4dec2e3db4108f6c455ae4 : Diary and some gamestate stuff done.

857d25376348dd601579e20899eb76efc074a30d : Networking added. More or less direct copy of work done outside of project. Interfacing with Control added, but not completed

bec4f5f40ae4aca6189ab41078fbd9c0475ca8e5 : Started coding gamestate

523bd81b3a6e0e20185ff5e2b3aa5fce13ccbc25 : Arranged classes into folders for better overview.

ff7248e0612f9f2e53525fde41de536b22d624b7 : Started creating gamestate in code

5e858aff8cab0331b3a504f4d4280a4dbf774cc7 : Did lots of BON

1d7eb386c7839a9c97ac1b8efb286b12fdaf7d97 : Added all the necessary classes for the GUI part of the project

6af320925f540af6842e9b6156355738d3cc94b1 : Did a lot of informal BON

25ff80260b970ec14e36728249e3f9cb324052b0 : Did some BON and added some diary entries

e0ce1ea7ec8bd8345bfb6f66c77c3d908d40af03 : Put in BON files

86029d88fde9a403f777855e433cd3fd8775ff7e : Added diary files

533e909a8d36dbffb1c6953cf99639a5baa90fbc : Added GUI classes

905a2d3a02218dec666bff65c4738306e9e7877c : Starting project

6 Related Documents

- BPMN Model of a Dominion turn
- Dependency Graphs
- DominionScreenCasts
- CodeMetrics
- Originally Submitted Project Overview
- System Analysis (in this document)
- System Design (in this document)
- System Production (in this document)

7 Milestones

7.1 System analysis

We started by having a look at how a game of Dominion would flow. We realised that each turn is a seperation unto its own, which means that when you have modeled how one turn looks, you have pretty much modeled them all.

So we sat down and modeled a turn of a Dominion game, which came out as the subsection below describes. Further in we have a quick glance at what architectures we thought of using at the start.

7.1.1 Dominion

We used a tool to draw up a BPMN model of the flow of a turn in a game of Dominion. It does not completely follow the syntax for BPMN, but the flow chart should be sort of clear. Any path with a cross in it is an exclusive OR and any path in it with a plus is a split in two branches or two branches are joined together after having been split.

We include it mainly to show how we think a game of Dominion flows and we have set up our system to follow this pattern. The images of the BPMN model can be found in Related Documents.

7.1.2 Architecture

We did not really consider that many different architectures at this time other than Model-View-Control, so that is pretty much what we sort of stuck with all the way through. We did add the server-client architecture to our network communication later on, but those are the two primary architectures we have in our system.

7.2 System design

7.2.1 General

We decided to use an MVC pattern for our overall structure, because we felt we could seperate the model (the Gamestate), the View (GUI) and Control (Game Logic and more). We also use a Client-Server architecture to do the network communcation. We have all used BON to describe most of our architecture.

7.2.2 GUI

Frederik Lysgaard In terms of system design, there isn't really much to say to the GUI part since none of us knew the XNA framework, and therefore didn't have any knowledge of the limitations and benefits. This led to a very ad hoc way of designing the GUI system, for the first draft I used my extensive knowledge of the game to try and get a overview of the components needed to properly display and play a game of Dominion. This resulted in the basic design for the GUI, but since I was still a beginner to XNA, this led to a design with almost all the right classes but with some strange inheritance, all in all it felt weird which made me study XNA some more, and after having used some days getting familiar with the framework I came up with the second and final draft.

Even though this was the final draft, it was far from perfect. I would for example, if I had had more time with XNA before the project, suggested some inheritance between the zones since they all is made from the same sprite "template". So to summarize there wasn't really any general battle plan for the GUI system design at the start of the project, which was a challenge, that taught me one thing, if you know you're going to work in a new framework, learn it before hand.

7.2.3 Client & Server and Control (Server and start-up parts)

Christian Jensen When we set out to make Dominion, we knew that network should not be a mandatory requirement, yet here we are with it. There are several reasons for this, such as that even to be moderately able to play the game, multiple computers should be involved. However, on a more practical level, I lacked an area of responsibility after we decided to couple the Control and GameState tighter, and network seemed an obvious idea.

I had one problem however: I had little to no idea about how to build a network system that would be appropriate for the project (nor did any of my teammates), both in terms of C# and networks in general. This is the primary reason why I could not sit down and design a network off my head. I needed to study the options first.

I did what any respectable coder would do; I searched the internet to see if somebody already had coded a solution. I came by several solutions but they were either not doing what we needed or were too complex. An answer presented itself. I knew of a guy who is in a group doing a seperate project and who is a wiz at networks in general (Simon Henriksen, shen@itu.dk) and the network part for their project (Descent) was quite similar to what we needed. We had several inter-group meetings during the project period with his group and a third (Magic the Gathering) and I asked him to help me understand several concept related to networks and how C# could use them.

He often referenced to his own implementation and just like using an Internet example to provide the basis for an implementation, he graciously allowed me to use his implementation as the basis for mine. I am not peticularly proud of this point, but my options was either use some structure I didn't really understand or use this, which I will claim to have grasped quite well.

7.2.4 Gamestate and Control (Game Logic)

Jakob Melnyk At first we decided to split Control, the static game logic and the dynamic game model into three different pieces of the architecture, but after a lot of thought and effort went into how to create the Game Logic (or more specifically, the card rules) as a seperate entity, we could not come up with way to do it that did not seem very akward. Instead we decided to let the Control class handle what happens when, for example, a specific card is played in the game.

So instead of both a dynamic and a static model as two seperate entities, we now had the dynamic model and the Control class making the magic happen. Whilst Control seems a little akward and cluttered at the moment, we feel we have really achieved what we wanted to with the Model-View-Control.

Most of the BON I made for the Gamestate was done in a way so that it would be easy for an outside source to access what he needed to know to make the dynamic model change. However, instead of making a very vast interface for all the different collections, I instead made them public. This does not seem like a particularly good idea, but I chose to show some of the private stuff instead of making a huge interface(in terms of method count). I could have considered using a (richer) invariant to make sure that my privates were better shielded from harm.

I made no BON-specification for the Control class and how it works, because it has no public interface except for its constructor.

Considering how our architecture and our system works, the way I have designed the cards both as Enums and as Card objects can seem a bit overkill, especially the very large inheritance hierachy on the Cards. It worked out well in the end, because even though it is clunky and akward, it makes much of the implementation easier - especially with a Factory such as CardFactory.

7.2.5 BON specification

We have included all of our BON in section 8 on page 29. Most of our BON was started before we started coding, but a lot of the classes have since been expanded with more contracts and more queries.

7.3 System production

7.3.1 General

Our split into the different, very separate parts of the code, made it somewhat cumbersome to combine at the end, but once it actually combined, it was quite an easy ride home in terms of getting the game to play. The different parts of the architecture should be quite replaceable, especially considering the GUIInterface and NetworkingInterface concepts and the way Gamestate works.

7.3.2 GUI

Frederik Lysgaard The production of the GUI can be split into three parts:

- The initial idea.
- The attempt to write it.
- And at last the rewrite of it all.

So let's start at the begining. The initial idea of how to produce the GUI was that all drawn classes should inherit from a super Sprite class but as I began coding I realized that the idea wouldn't be so optimal, since we had different objects with different positions which at that point, in my XNA traning, semmed to make it all very hard to draw, atleast with different positions.

So after realizing that my first attempt of code was not going to work, I set to rewriting what I already had and try and reform it with my new knowledge of XNA. I then ended up with what is our end GUI which consist of a lot of zones where you can either draw buttons or cards sprites to, this seemed like a extremly easy straight forward solution, even though if I had had more time, I would have loved to code in some inheritance, espcially a super zoneclass that would act as template for the other zoneclasses.

7.3.3 Server and Control (Server and start-up parts)

Christian Jensen We had talked about networks and it was clear that we didn't want to communicate directly with clients and server. An interface was needed. Dominion has several cards that require a player to respond to how they want to react to it when another player plays it. This means that we needed to ensure that the other players has a chance to make a choice if such a card is played. The way that this works is though the TurnMessage method, through which all communication takes place.

In TurnMessage a arbitrary string is passed to the NetworkInterface. The only requirement is that it does not contain '|', '<' and '>', as these characters are used by the system. This is then wrapped in 2 pieces of information: a MessageType, which tells the server and the other clients what kind of message it is, and also added, is a piece of text that signifies the end of the message.

This is then sent to the server. The server communicates with the clients through a number of Connection objects, one for each client. These also contain the Socket objects on the server side. The Socket listens to incoming messages thanks to the BeginRecieve method, which then calls BeginRecieveCallback when a message is received. This method and the asyncResult it takes as a parameter, along with BeginRecieve, is the basis of the entire network.

The message is then passed to the server itself through an event. This ensures that the server can take any messages it receives at the "same" time. The message passed to the server no longer contains the End of File textpiece, but still contains the message type. The Server can then use the message type to determine how to act with it. In most cases, the server will simply forward the message.

The server keep all its Connections (which basically is its clients) in a Dictionary, with the Id of the player as the key. As keys are created incrementally as clients join before the game, 1 is defined by us as the Host, since the Host joins almost instantly after creating the server. The server can then use this Dictionary to forward the message, by excluding the Connection with the Id matching the Connection that the message was received from.

The message gets reattached with its message type and also joining it is the Id of the player that sent the message. The Client recieves it in almost the same way that the Connection does; through a BeginRecieve and a BeginRecieveCallback. The Client then passes the message to NetworkInterface, after removing the End of File text.

Then we are back at the NetworkInterface, this is where things gets interesting. Depending on the message type, the NetworkInterface reacts differently. System messages are passed on through the event MessageReceived. Action messages are also passed on, but they also flings a message of the third type back through the system, the Response. Response messages are stored in an array, where they await something to pick them up.

The Response message is how I have planned to implement cards that require a reaction from another player. The Idea is that on all normal action cards (i.e. cards that doesn't require another player to actively make a choice) a 'message received' Response, sent from every other player and this is then the 'insurance' that all players have received the message by the TurnMessage waiting for replys from all other players. These 'message received' returns are sent to all players, not just the active, so that everyone can be sure that everyone is still in the game. However, on a card that require a real reply, it would instead send a WaitResponse type message, which would tell the sender that they should wait, because a message from that player was coming later.

Almost the entire system for this is in the code, apart from the infrastructure to WaitResponse. We have however chosen not to enable any of it in the build. We found that for one, it causes bugs in rare cases. Network is a fiddly thing to test and bugs happened, one reason being that messages were received too quickly after one another, due to the Threads that lie behind the listening structure. Another reason to disable it is that with the current structure we have no cards which require replys from any player that is not currently active. Basically we prioritized our mandatory requirement, that the game should be able to be played, rather than implementing a prehaps overly fancy feature.

PreGame: The server need to set up the group of people playing. This is accomplished through these steps: first a person creates a server though the console at the beginning of the game. This command creates a server and a client object, the last of which instantly connects to the first. He is then shown his IP, which other people can connect to, by creating a client instead of a server and typing the IP address of the host. To begin the game, a special command is required. All players can type messages, which are sent with PreGameMessage, which uses System message type instead of action. They are sent to all players and shown in the Console. The server is hardcoded to wait for the command '<STGM>' from client 1 (the host). This will cause the server to send a system message to all players, who are in turn hardcoded to wait for it, because it contains 2 key pieces of information: The total number of players and the id of the player. After this is recieved, the game will begin.

Obviously a console start-menu is not optimal, but the primary reason for not having this as part of the GUI is because the GUI lagged behind and we needed to fullfill our mandatory requirement: The game must work.

While the grand design might not have been there from the start, thought has gone into how the network should run, as this text should show.

7.3.4 Gamestate and Control (Game Logic)

Jakob Melnyk Coding my original specification I made in BON was pretty simple, but once I got the code written for it, a lot of contract possiblities came to mind, so I implemented these as well, and then later updated my BON to reflect on this.

I was really glad when I finally got to code Control, because then I would get to see the fruits of my architectural labour. Implementing how each card worked was a breeze, considering the interface I had to work with in Gamestate. It got a bit more tricky once I had to communicate with the GUI and Server, but they have a pretty simple interface as well, so it did not take long for us to get started on testing the system.

8 BON-specification

```
system_chart BDSADominion
indexing
        author: "Frederik Lysgaard (frly@itu.dk), Christian 'Troy' Jensen,
        Jakob Melnyk (jmel@itu.dk)";
        supervisor: "Joe Kiniry";
        course: "BDSA-E2011";
        created: "28th November 2011";
        lastModified: "14th December 2011";
explanation
        "System chart for the BDSADominion project in the Analysis, Design
     and Software Architecture fall 2011 course."
cluster DOMINION_SYSTEM
        description "The Dominion game system."
end

cluster_chart DOMINION_SYSTEM
class CONTROL
        description "The man in the middle between the three parts of the architecture.
        Contains game logic, starts the program, etc."
cluster GUI
        description "Used to display the current state of the model and to interact with the user."
cluster GAMESTATE_CLUSTER
        description "The 'model' of the project. Remembers information about most of the states and
cluster NETWORK_CLUSTER
        description "Communicates between different instances of the application across LAN."
end
```

```
cluster_chart GAMESTATE_CLUSTER
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "The classes making up the 'dynamic' state of the game. This includes decks,
        hands, discard piles, etc."
        class GAMESTATE
                description "The overall 'state' of the current game."
        class PLAYER
                description "Represents a player and everything a player owns."
        class ZONE
                description "List of zones that can be targets of events."
        cluster CARD_CLUSTER
                description "The different kinds of cards."
end

class_chart GAMESTATE
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Keeps track of the players and everything the players share,
        such as the trash pile and the supply."
        query
                "May I have a new gamestate with this set-up?",
                "Who is the active player?",
                "How many players are in the game?",
                "Is the active player in the Action Phase?",
                "Is the active player in the Buy Phase?",
                "What cards are in the trash pile?",
                "What players are in the game?",
                "What does the supply look like?",
                "Number of actions left?",
                "Number of buys left?",
                "Number of coins left?",
                "How many points does each player have?",
        command
                "Make this player the active player!",
                "Begin Action Phase",
                "End Action Phase",
                "Begin Buy Phase",
                "End Buy Phase",
                "Do Clean-up phase",
                "Increase the amount of actions the active player has by this much!",
                "Increase the amount of buys the active player has by this much!",
                "Increase the amount of coins the active player has by this much!",
                "That player gains this card type in this zone from the supply.",
        constraint
                "Can have 2, 3 OR 4 players.",
                "Cannot begin Action Phase while in Action Phase or Buy Phase.",
                "Cannot end Action Phase while not in Action Phase.",
                "Cannot begin Buy Phase while in Action Phase or Buy Phase.",
                "Cannot end Buy Phase while not in Buy Phase.",
                "The active player cannot be made the active player."
end

class_chart PLAYER
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Each player is represented by a player object that keeps track of their
        decks, hands, discard piles, etc."
        query
                "May I have a new Player?",
                "What cards do you have?",
                "How many cards do you have in your deck?",
                "How many cards do you have in your discard pile?",
                "What card is on top of your discard pile?",
                "What card is on top of your deck?",
                "What cards do you have in your hand?",
                "What number are you?",
                "What cards have you played?",
                "What have you put in your temporary zone?",
        command
                "Move this card from that zone to the temporary zone!",
                "Move this card from the temporary zone to that zone!",
                "Move this card from the hand to the temporary zone!",
                "Add this card to that zone!",
                "Remove this card from that zone!",
                "Draw a card!",
                "Draw this many cards!",
        constraint
                "A player cannot have a card in his deck, discard pile, hand, or 'played field'
                that is not in his total set of cards."
```

```
end

class_chart ZONE
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Represents the values used to refer to the zones in the player class
        and gamestate class."
        query
                "May I have the value 'v'?",
        constraint
                "The values allowed for this class are exactly one of 'DECK',
                'DISCARD', 'HAND', 'SUPPLY', 'PLAYED', 'TRASH'."
end
```

```
cluster_chart CARD_CLUSTER
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Cluster showing how card system works."
        class CARD
                description "A card."
        class CARD_NAME
                description "The names of all the cards."
        class CARD_FACTORY
                description "Produces cards."
        cluster CARD_TYPES
                description "The different meta-types of cards."
        cluster CARDS
                description "Contains all the cards from the game."
end

class_chart CARD
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "A card is the representation of the cards within the game."
        query
                "What is your card name?",
                "What is your card number?",
                "Have you been initialized yet?",
                "Are you equal to this object?",
                "Are you and this other card the same?"
        command
                "Initialize yourself like this!"
end

cluster_chart CARD_TYPES
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "The different types of cards that exist."
        class TREASURE
                description "A card used to buy new cards."
        class VICTORY
                description "A card that grants the points used to win the game."
        class ACTION
                description "A card used to help the player buy more cards,
                get rid of unwanted cards, etc."
        class ACTION_ATTACK
                description "A card that is used to 'attack' other players."
        class ACTION_REACTION
                description "A card used to react to opponent attacks."
        class KINGDOM_VICTORY
                description "A special kind of victory card."
end

class_chart TREASURE
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "A card used to buy new cards."
        inherit CARD
end

class_chart VICTORY
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "A card that grants the points used to win the game."
        inherit CARD
end

class_chart ACTION
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "A card used to help the player buy more cards,
        get rid of unwanted cards, etc."
        inherit CARD
end

class_chart ACTION_ATTACK
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "A card that is used to 'attack' other players."
        inherit ACTION
end

class_chart ACTION_REACTION
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "A card used to react to opponent attacks."
        inherit ACTION
end

class_chart KINGDOM_VICTORY
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "A special kind of victory card."
        inherit VICTORY
end

class_chart CARD_NAME
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Represents the values used to refer to the card names."
        query
                "May I have the value 'v'?"
        constraint
                "The values allowed for this class are exactly one of \
                \'COPPER', 'SILVER', 'GOLD', 'CURSE', 'ESTATE', 'DUCHY', 'PROVINCE', \
                \'CELLAR', 'CHAPEL', 'MOAT', 'CHANCELLOR', 'VILLAGE', 'WOODCUTTER', \
                \'WORKSHOP', 'BUREAUCRAT', 'FEAST', 'GARDENS', 'MILITIA', 'MONEYLENDER', \
                \'REMODEL', 'SMITHY', 'SPY', 'THIEF', 'THRONE_ROOM', 'COUNCIL_ROOM', 'FESTIVAL', \
                \'LABORATORY', 'LIBRARY', 'MARKET', 'MINE',
                'WITCH', 'ADVENTURER', 'EMPTY', 'BACKSIDE'."
end

class_chart CARD_FACTORY
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Factory for producing cards with the correct values."
        query
                "Has the factory been set up?",
                "What cards have been made already?",
        command
                "Set up the factory with these cards!",
                "Give me a card with this name!",
        constraint
                "A card that has already been made cannot be made again.",
end
```

```
cluster_chart CARDS
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Contains all the cards from a standard Dominion game."
        class COPPER
                description "The Copper card."
        class SILVER
                description "The Silver card."
        class GOLD
                description "The Gold card."
        class CURSE
                description "The Curse card."
        class ESTATE
                description "The Estate card."
        class DUCHY
                description "The Duchy card."
        class PROVINCE
                description "The Province card."
        class CELLAR
                description "The Cellar card."
        class CHAPEL
                description "The Chapel card."
        class MOAT
                description "The Moat card."
        class CHANCELLOR
                description "The Chancellor card."
        class VILLAGE
                description "The Village card."
        class WOODCUTTER
```

```
                description "The Woodcutter card."
        class WORKSHOP
                description "The Workshop card."
        class BUREAUCRAT
                description "The Bureaucrat card."
        class FEAST
                description "The Feast card."
        class GARDENS
                description "The Gardens card."
        class MILITIA
                description "The Militia card."
        class MONEYLENDER
                description "The Moneylender card."
        class REMODEL
                description "The Remodel card."
        class SMITHY
                description "The Smithy card."
        class SPY
                description "The Spy card."
        class THIEF
                description "The Thief card."
        class THRONE_ROOM
                description "The Throne Room card."
        class COUNCIL_ROOM
                description "The Council Room card."
        class FESTIVAL
                description "The Festival card."
        class LABORATORY
                description "The Laboratory card."
        class LIBRARY
                description "The Library card."
        class MARKET
                description "The Market card."
        class MINE
                description "The Mine card."
        class WITCH
                description "The Witch card."
        class ADVENTURER
                description "The Adventurer card."
end

class_chart COPPER
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Worth one coin. Costs no coins."
        inherit TREASURE
end

class_chart SILVER
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Worth two coins. Costs three coins."
        inherit TREASURE
end

```

```
class_chart GOLD
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Worth three coins. Costs six coins."
        inherit TREASURE
end

class_chart CURSE
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Worth minus one victory point. Costs no coins."
end

class_chart ESTATE
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Worth one victory point. Costs two coins."
        inherit VICTORY
end

class_chart DUCHY
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Worth three victory points. Costs five coins."
        inherit VICTORY
end

class_chart PROVINCE
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Worth six victory points. Costs eight coins."
        inherit VICTORY
end

class_chart GARDENS
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Worth one victory point for every ten cards in your deck
        (rounded down) at the end of the game. Costs four coins."
        inherit KINGDOM_VICTORY
end

class_chart CELLAR
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Grants one action. \
                \Discard any number of cards - draw one card for each card discarded. \
                \Costs two coins."
        inherit ACTION
end

class_chart CHAPEL
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Trash up to four cards from your hand. Costs two coins."
        inherit ACTION
end

class_chart CHANCELLOR
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Grants two coins. The player may immediately
        put your deck into your discard pile. \
                                 \Costs three coins."
        inherit ACTION
end

class_chart VILLAGE
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Grants one card. Grants two actions. Costs three coins."
        inherit ACTION
end

class_chart WOODCUTTER
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Grants one buy. Grants two coins. Costs three coins."
        inherit ACTION
end

class_chart WORKSHOP
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Player gains a card costing up to four coins. Costs three coins."
        inherit ACTION
end

class_chart FEAST
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Player trashes this card.
        Gain a card costing up to five coins. Costs 4 coins."
        inherit ACTION
end

class_chart MONEYLENDER
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Player trashes a Copper card from his/her hand. \
                     \If the player does so, he is granted three coins. Costs four coins."
        inherit ACTION
end

class_chart REMODEL
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Player trashes a card from his/her hand.
        Player gains a card costing up to two coins more \
                                \than the trashed card. Costs 4 coins."
        inherit ACTION
end

class_chart SMITHY
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Grants three cards. Costs four coins."
        inherit ACTION
end

class_chart THRONE_ROOM
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Player chooses an Action card in his/her hand.
        That card is played twice. Costs four coins."
        inherit ACTION
end

class_chart COUNCIL_ROOM
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Grants four cards. Grants one buy.
        All other players are granted one card. Costs five coins."
        inherit ACTION
end

class_chart FESTIVAL
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Grants two actions. Grants one buy. Grants two coins. Costs five coins."
        inherit ACTION
end

class_chart LABORATORY
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Grants two cards. Grants one action. Costs five coins."
        inherit ACTION
end

class_chart LIBRARY
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Player draws until he/she has seven cards in hand.
        Player may set aside any action cards\
                                \ drawn this way; discard the set aside
                                cards after the Player is finished drawing."
        inherit ACTION
end

class_chart MARKET
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Grants one card. Grants one action.
        Grants one buy. Grants one coin. Costs five coins."
        inherit ACTION
end

class_chart MINE
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Player trashes a Treasure card from his/her hand.
        Player gains a treasure card costing \
                                \up to three coins more. Costs five coins."
        inherit ACTION
end

class_chart ADVENTURER
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Player reveals cards from his/her deck until
        two Treasure have been revealed. \
                                \Player puts the two Treasure cards into hand
                                and discard the other revealed cards. Costs six coins."
        inherit ACTION
end

class_chart BUREAUCRAT
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Player gains a silver card on top of deck.
        Each other Player reveals a Victory card from his hand \
                                 \and puts it on top of his deck
                                 (or reveals a hand with no Victory cards). Costs four coins."
        inherit ACTION_ATTACK
end

class_chart MILITIA
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Grants two coins.
        Each other player discards down to three cards in his/her hand. Costs four coins."
        inherit ACTION_ATTACK
end

class_chart SPY
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Grants one card. Grants one action. \
                \Each Player (including the active Player) reveals the top card of deck \
                \and the active Player decides to either put the card back or discard it.\
                \Costs four coins."
        inherit ACTION_ATTACK
end

class_chart THIEF
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Each other Player reveals the top two cards of his/her deck.
                                If any Treasure cards are revealed, \
                                \ the active Player can choose to trash one of them.
                                The active player may gain any or all of the trashed cards. \
                                \ The other revealed cards are discarded. Costs four coins."
        inherit ACTION_ATTACK
end

class_chart WITCH
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Grants two cards. Each other player gains a Curse card. Costs five coins."
        inherit ACTION_ATTACK
end

class_chart MOAT
        indexing
                author: "Jakob Melnyk (jmel@itu.dk)";
        explanation "Grants two cards. When another Player plays an Attack card and
        this card is in your hand, you reveal this card. Revealing \
                                \ this card makes you unaffected by that Attack. Costs two coins."
        inherit ACTION_REACTION
end
```

```
cluster_chart GUI
 indexing
        author: "Frederik Lysgaard (frly@itu.dk)";
 explanation " The graphical representation part of the project"
 class GAMECLASS description " The game class"
 class DECKZONE description " The deck class"
 class ACTIONZONE description " The card class"
 class DISCARDZONE description " The discardzone class"
 class HANDZONE description " The handzone class"
 class SUPPLYZONE description " The supplyzone class"
 class BUTTONSPRITE description " The buttonsprite class"
 class CARDSPRITE description " The cardsprite class"
 class PROGRAM description " The program class"
 class GUICONSTANTS description " The constant class"
 class GUIINTERFACE description " The interface class for the gui"
end

class_chart DECKZONE
 indexing
        author: "Frederik Lysgaard (frly@itu.dk)";
 explanation " responsible for representing the deck"
 command
        "Draw the content!"
end

class_chart BUTTONSPRITE
 indexing
        author: "Frederik Lysgaard (frly@itu.dk)";
 explanation " the basic class which all graphical objects should inherit from"
 command
        "Draw the content!"
end

```

```
class_chart CARDSPRITE
 indexing
        author: "Frederik Lysgaard (frly@itu.dk)";
 explanation " responsible for representing the cards"
  query
        "Is this cardsprite equal to this cardsprite?"
  command
        "Draw the content!"
end

class_chart PROGRAM
 indexing
        author: "Frederik Lysgaard (frly@itu.dk)";
 explanation " responsible for executing the game"
 command
        " Run a clinet!",
        " Run a Host!",
        " Start the GUI!",
end

class_chart GAMECLASS
 indexing
        author: "Frederik Lysgaard (frly@itu.dk)";
 explanation " responsible for creating the initial GUI with the components from the other classes"
 command
        " Initialize the content!",
        " Load the content!",
        " Unload the content!",
        " Update the game!",
        " Draw the content!",
end

class_chart GUICONSTANTS
 indexing
        author: "Frederik Lysgaard (frly@itu.dk)";
 explanation " responsible for keeping all the constants used in GUI i one place"
end

class_chart GUIINTERFACE
 indexing
        author: "Frederik Lysgaard (frly@itu.dk)";
 explanation " responsible for the interface between the GUI and the Controller"
 command
        "Run the game!",
        "Draw the hand!",
        "Draw the actionzone!",
        "Draw the discardzone!",
        "Draw the deck!",
        "Set actions!",
        "Set buys!",
        "Set coins!",
        "Set endgame!",
        "Set the turn!",
        "Set the phase!",
        "Set the playernumber!",
        "Make the supplyzone!",
end
```

```
cluster_chart NETWORK_CLUSTER
indexing
        author: "Christian 'Troy' Jensen, chrj@itu.dk";
explanation "The part of the program responsible for running the network"
class CLIENT description "A network client"
class SERVER description "A network server"
class CONNECTION description "A connection between a server and a client"
class NETWORKING_INTERFACE description "A network interface"

end

class_chart CLIENT
indexing
        author: "Christian 'Troy' Jensen, chrj@itu.dk";
explanation "Represents a player in a game of Dominion, one for each player"
query
        "Can I have the connection for this client?",
command
        "Begin recieving more messages!",
end

class_chart SERVER
indexing
        author: "Christian 'Troy' Jensen, chrj@itu.dk";
explanation "Responsible for managing the clients of a game, only one per game"
query
        "Can I have the IP of the server?",
        "Can I have a list of the known clients",
command
        "Start the server!",
        "Send this as a system message to this client!",
        "Send this as a system message to all clients!",
        "Forward this message!",
end

class_chart CONNECTION
indexing
        author: "Christian 'Troy' Jensen, chrj@itu.dk";
explanation "Responsible for holding all the information
                        on a client that a server has, one for each client"
query
        "Can I have the IP of the client",
        "Can I have  the Id of the Client",
command
        "Send this message!",
        "Begin recieving more messages!",
end

class_chart NETWORKING_INTERFACE
indexing
        author: "Christian 'Troy' Jensen, chrj@itu.dk";
explanation "The outward face of a networking session, keeps track of a client and maybe a server"
```

```
query
        "Is this interface running a server?",
        "Can I have the IP of the server?",
command
        "This is the number of clients!",
        "Send this message, and you better give me some answers!",
        "Send this message",
end
```

```
static_diagram GAMESTATE_CLUSTER
component
        class GAMESTATE
                indexing
                        author: "Jakob Melnyk (jmel@itu.dk)";
                feature
                --Queries
                ActivePlayer : PLAYER
                InActionPhase : BOOLEAN
                InBuyPhase : BOOLEAN
                GetPhase : NATURAL
                GetPlayers : SEQUENCE[PLAYER]
                GetSupply : TABLE[CARD_NAME, NATURAL]
                GetTrash : SEQUENCE[CARD]
                NumberOfPlayers : NATURAL
                        ensure Result >= 2 and Result <= 4
                end
                NumberOfActionsLeft : NATURAL
                NumberOfBuysLeft : NATURAL
                NumberOfCoinsLeft : NATURAL
                NewGamestate : GAMESTATE
                        -> numberOfPlayers : NATURAL
                        -> startSupply : TABLE[CARD_NAME, NATURAL]
                        require numberOfPlayers >= 2 and numberOfPlayers
                        <= 4 and startSupply /= void
                end
                GetScores : SEQUENCE[INTEGER]

                --Ccmmands
                SetActivePlayer
                        -> player : PLAYER
                        require playerNumber >= 1 and playerNumber <=
                        NumberOfPlayers and player /= void
                end
                StartActionPhase
                        require InActionPhase = false and InBuyPhase = false
                        ensure InActionPhase = true and InBuyPhase = false
                end
                EndActionPhase
                        require InActionPhase = true and InBuyPhase = false
                        ensure InActionPhase = false and InBuyPhase
                        = false and NumberOfActionsLeft = 0
                end
                StartBuyPhase
                        require InActionPhase = false and InBuyPhase = false
                        ensure InActionPhase = false and InBuyPhase = true
                end
                EndBuyPhase
                        require InActionPhase = false and InBuyPhase = true
                        ensure InActionPhase = false and InBuyPhase = false and
                        NumberOfBuysLeft = 0 and NumberOfCoinsLeft = 0
                end
                DoCleanUp
                        require InActionPhase = false and InBuyPhase = false
                end
                IncreaseActions
                        -> amount : INTEGER
                        require amount + NumberOfActionsLeft >= 0
                end
                IncreaseBuys
                        -> amount : INTEGER
                        require amount + NumberOfBuysLeft >= 0
                end
                IncreaseCoins
                        -> amount : INTEGER
                        require amount + NumberOfCoinsLeft >= 0
                end
                PlayerGainsCard
                        -> player : PLAYER
                        -> card : CARD_NAME
                        require player member_of GetPlayers and player /= void
                end
        end
        class PLAYER
                indexing
                        author: "Jakob Melnyk (jmel@itu.dk)";
                feature
                --Queries
                GetAllCards : SET[CARD]
                GetDeckSize : NATURAL
                GetDiscardSize : NATURAL
                GetHand : SEQUENCE[CARD]
                GetPlayerNumber : NATURAL
                GetTopOfDiscard : CARD
                        require GetDiscardSize /= 0
                end
                GetTopOfDeck : CARD
                        require GetDiscardSize /= 0
                end
                GetPlayed : SEQUENCE[CARD]
                GetTemporaryZone : SEQUENCE[CARD]

                --Commands
                MoveFromZoneToTemporary
                        -> zone : ZONE
                        require (zone = DECK or zone = DISCARD) and (zone = DECK ->
                        (GetDeckSize = 0 and GetDiscardSize = 0) /= true)
                                        and (zone = DISCARD -> (GetDiscardSize /= 0))

                        ensure GetTemporaryZone.Count = old GetTemporaryZone.Count + 1
                                and (zone = DECK -> GetDeckSize = old GetDeckSize - 1)
                                and (GetTopOfDeck /= old GetTopOfDeck)
                                and (zone = DISCARD -> GetDiscardSize = old GetDiscardSize - 1)
                                and (GetTopOfDiscard /= old GetTopOfDiscard)
                end
                MoveFromHandToTemporary
                        ->card : CARD
                        require GetHand.Contains(card) = false and card /= void
                        ensure GetHand.Contains(card) = false and GetTemporaryZone.Contains(card)
                end
                MoveFromTemporary
                        -> card : CARD
                        -> zone : ZONE
                        require (zone = DECK or zone = DISCARD or zone = HAND or zone = PLAYED)
                        and GetTemporaryZone.Contains(card) and card /= void
                        ensure (GetTemporaryZone.Count = old GetTemporaryZone.Count - 1) and
                                (zone = DECK -> GetDeckSize = old GetDeckSize +1 ) and
                                (zone = DECK -> GetTopOfDeck = old GetTemporaryZone.Get(old
                                GetTemporaryZone.Count - 1)) and
                                (zone = DISCARD -> GetDiscardSize = old GetDiscardSize + 1)
                                and
                                (zone = DISCARD -> GetTopOfDiscard =
                                old GetTemporaryZone.Get(old GetTemporaryZone.Count - 1)) and
                                zone = HAND -> GetHand.Count = old GetHand.Count + 1)
                                and GetHand.Contains(card) and
                                (zone = HAND -> GetHand.Get(GetHand.Count - 1) =
                                old GetTemporaryZone.Get(old GetTemporaryZone.Count - 1)) and
                                (zone = PLAYED -> GetPlayed.Count = old Played.Count + 1) and
                                (zone = PLAYED -> GetPlayed.Get(Played.Count - 1) =
                                old GetTemporaryZone.Get(old GetTemporaryZone.Count - 1))
                end
                AddCardToZone
                        -> card : CARD
                        -> zone : ZONE
                        require (zone = DECK or zone = DISCARD or zone = HAND or zone = PLAYED)
                        and card /= void
                        ensure GetAllCards.Contains(card) and
                                (zone = HAND -> GetHand.Get(GetHand.Count - 1) = card) and
                                (zone = HAND -> GetHand.Count = old GetHand.Count + 1) and
                                (zone = PLAYED -> GetPlayed.Get(GetPlayed.Count - 1) = card)
                                and
                                (zone = PLAYED -> GetPlayed.Count = old GetPlayed.Count + 1)
                                and
                                (zone = DISCARD -> GetDiscardSize = old GetDiscardSize + 1)
                                and
                                (zone = DISCARD -> GetTopOfDiscard = card) and
                                (zone = DECK -> GetDeckSize = old GetDeckSize +1 ) and
                                (zone = Deck -> GetTopOfDeck = card)
                end
                RemoveCardFromZone
                        -> card : CARD
                        -> zone : ZONE
                        require (zone = DECK or zone = DISCARD or zone = HAND or zone = PLAYED)
                        and GetAllCards.Contains(card) and card /= void
                                and        (zone = HAND -> GetHand.Contains(card))
                                and (zone = PLAYED -> GetPlayed.Contains(card))
                                and (zone = DECK -> (GetDeckSize = 0 and GetDiscardSize = 0)
                                = false)
                                and (zone = DISCARD -> GetDiscardSize /= 0)
                        ensure GetAllCards.Contains(card) = false and
                                (zone = HAND -> GetHand.Contains(card) = false) and
                                (zone = HAND -> GetHand.Count = old GetHand.Count - 1) and
                                (zone = PLAYED -> GetPlayed.Contains(card) = false) and
                                (zone = PLAYED -> GetPlayed.Count = old GetPlayed.Count - 1)
                                and
                                (zone = DISCARD -> GetDiscardSize = old GetDiscardSize - 1)
                                and
                                (zone = DECK -> GetDeckSize = old GetDeckSize - 1)
                end
                DrawCards
                        -> amount : NATURAL
                DrawCard
                        require GetDeckSize + GetDiscardSize /= 0
                        ensure GetHand.Count = old GetHand.Count + 1
                end

                --Invariant: A card cannot be in the DECK, DISCARD,
                HAND or PLAYED zones of a player
                --          if it is not in the 'ALL CARDS'.
        end
        class ZONE
                indexing
                        author: "Jakob Melnyk (jmel@itu.dk)";
                feature
                --Queries
                value : STRING
                        ensure Result = "DECK" or Result = "DISCARD" or Result = "HAND" or
                        Result = "SUPPLY" or Result = "TRASH" or Result = "PLAYED"
                end
                --Commands
        end
end
```

```
static_diagram CARD_TYPES_CLUSTER
        component
                class TREASURE
                        indexing
                                author: "Jakob Melnyk (jmel@itu.dk)";
                        inherit CARD
                end

                class VICTORY
                        indexing
                                author: "Jakob Melnyk (jmel@itu.dk)";
                        inherit CARD
                end

                class ACTION
                        indexing
                                author: "Jakob Melnyk (jmel@itu.dk)";
                        inherit CARD
```

```
		end

		class ACTION_ATTACK
			indexing
				author: "Jakob Melnyk (jmel@itu.dk)";
			inherit ACTION
		end

		class ACTION_REACTION
			indexing
				author: "Jakob Melnyk (jmel@itu.dk)";
			inherit ACTION
		end

		class KINGDOM_VICTORY
			indexing
				author: "Jakob Melnyk (jmel@itu.dk)";
			inherit VICTORY
		end
end
```

```
static_diagram CARD_CLUSTER
component
	class CARD
		indexing
			author: "Jakob Melnyk (jmel@itu.dk)";
		feature
		--Queries
		EqualsOtherObj : BOOLEAN
			-> obj : VALUE -- Object in C#.
		EqualsOtherCard : BOOLEAN
			-> other : CARD
		GetName : CARD_NAME
		GetNumber : NATURAL
		SetUp : BOOLEAN
		--Commands
		Initialize
			-> name : CARD
			-> number : NATURAL
			require SetUp = false
			ensure  SetUp = true
		end
	end

	class CARD_FACTORY
		indexing
			author: "Jakob Melnyk (jmel@itu.dk)";
		feature
		--Queries
		SetUp : BOOLEAN
		CreatedCards : SET[CARD]
		CardsMade : TABLE[CARD_NAME, NATURAL] --private

		--Commands
		CreateCard : CARD
```

```
                        -> Card : CARD_NAME
                        ensure Result.GetName = CARD_NAME
                end
                SetUpCards
                        -> cards : COLLECTION[CARD_NAME]
                        require SetUp = false and cards /= void
                        ensure SetUp = true
                end
                --Invariant commented because I could not get it to compile,
                --but below is a rough idea of what I wanted to express.
                --for_all c member_of CreatedCards it_holds c.GetNumber < CardsMade.get(c.GetName)
        end

        class CARD_NAME
                indexing
                        author: "Jakob Melnyk (jmel@itu.dk)";
                feature
                --Queries
                value : STRING --This looks very akward, but we felt it
                --best described what we wanted to express.
                ensure Result = "COPPER" or Result = "GOLD" or Result = "SILVER" or
                Result = "CURSE" or Result = "ESTATE" or Result = "DUCHY" or Result = "PROVINCE" or
                Result = "CELLAR" or Result = "CHAPEL" or Result = "MOAT" or Result = "CHANCELLOR" o
                Result = "VILLAGE" or Result = "WOODCUTTER" or Result = "WORKSHOP" or
                Result = "BUREAUCRAT" or Result = "FEAST" or Result = "GARDENS"
                or Result = "MILITIA" or
                Result = "MONEYLENDER" or Result = "REMODEL" or Result = "SMITHY"
                or Result = "SPY" or
                Result = "THIEF" or Result = "THRONE_ROOM" or Result = "COUNCIL_ROOM"
                or Result = "FESTIVAL" or
                Result = "LABORATORY" or Result = "LIBRARY" or Result = "MARKET" or
                Result = "MINE" or Result = "WITCH" or
                Result = "EMPTY" or Result = "BACKSIDE"
                end
                --Commands
        end
end
```

```
static_diagram CARDS_CLUSTER
        component
                class COPPER
                        indexing
                                author: "Jakob Melnyk (jmel@itu.dk)";
                        inherit TREASURE
                end

                class SILVER
                        indexing
                                author: "Jakob Melnyk (jmel@itu.dk)";
                        inherit TREASURE
                end

                class GOLD
                        indexing
                                author: "Jakob Melnyk (jmel@itu.dk)";
```

```
		inherit TREASURE
	end

	class CURSE
		indexing
			author: "Jakob Melnyk (jmel@itu.dk)";
	end

	class ESTATE
		indexing
			author: "Jakob Melnyk (jmel@itu.dk)";
		inherit VICTORY
	end

	class DUCHY
		indexing
			author: "Jakob Melnyk (jmel@itu.dk)";
		inherit VICTORY
	end

	class PROVINCE
		indexing
			author: "Jakob Melnyk (jmel@itu.dk)";
		inherit VICTORY
	end

	class GARDENS
		indexing
			author: "Jakob Melnyk (jmel@itu.dk)";
		inherit KINGDOM_VICTORY
	end

	class CELLAR
		indexing
			author: "Jakob Melnyk (jmel@itu.dk)";
		inherit ACTION
	end

	class CHAPEL
		indexing
			author: "Jakob Melnyk (jmel@itu.dk)";
		inherit ACTION
	end

	class CHANCELLOR
		indexing
			author: "Jakob Melnyk (jmel@itu.dk)";
		inherit ACTION
	end

	class VILLAGE
		indexing
			author: "Jakob Melnyk (jmel@itu.dk)";
		inherit ACTION
	end
```

```

        class WOODCUTTER
                indexing
                        author: "Jakob Melnyk (jmel@itu.dk)";
                inherit ACTION
        end

        class WORKSHOP
                indexing
                        author: "Jakob Melnyk (jmel@itu.dk)";
                inherit ACTION
        end

        class FEAST
                indexing
                        author: "Jakob Melnyk (jmel@itu.dk)";
                inherit ACTION
        end

        class MONEYLENDER
                indexing
                        author: "Jakob Melnyk (jmel@itu.dk)";
                inherit ACTION
        end

        class REMODEL
                indexing
                        author: "Jakob Melnyk (jmel@itu.dk)";
                inherit ACTION
        end

        class SMITHY
                indexing
                        author: "Jakob Melnyk (jmel@itu.dk)";
                inherit ACTION
        end

        class THRONE_ROOM
                indexing
                        author: "Jakob Melnyk (jmel@itu.dk)";
                inherit ACTION
        end

        class COUNCIL_ROOM
                indexing
                        author: "Jakob Melnyk (jmel@itu.dk)";
                inherit ACTION
        end

        class FESTIVAL
                indexing
                        author: "Jakob Melnyk (jmel@itu.dk)";
                inherit ACTION
        end

```

```
		class LABORATORY
			indexing
				author: "Jakob Melnyk (jmel@itu.dk)";
			inherit ACTION
		end

		class LIBRARY
			indexing
				author: "Jakob Melnyk (jmel@itu.dk)";
			inherit ACTION
		end

		class MARKET
			indexing
				author: "Jakob Melnyk (jmel@itu.dk)";
			inherit ACTION
		end

		class MINE
			indexing
				author: "Jakob Melnyk (jmel@itu.dk)";
			inherit ACTION
		end

		class ADVENTURER
			indexing
				author: "Jakob Melnyk (jmel@itu.dk)";
			inherit ACTION
		end

		class BUREAUCRAT
			indexing
				author: "Jakob Melnyk (jmel@itu.dk)";
			inherit ACTION_ATTACK
		end

		class MILITIA
			indexing
				author: "Jakob Melnyk (jmel@itu.dk)";
			inherit ACTION_ATTACK
		end

		class SPY
			indexing
				author: "Jakob Melnyk (jmel@itu.dk)";
			inherit ACTION_ATTACK
		end

		class THIEF
			indexing
				author: "Jakob Melnyk (jmel@itu.dk)";
			inherit ACTION_ATTACK
		end

		class WITCH
```

```
                        indexing
                                author: "Jakob Melnyk (jmel@itu.dk)";
                        inherit ACTION_ATTACK
                end

                class MOAT
                        indexing
                                author: "Jakob Melnyk (jmel@itu.dk)";
                        inherit ACTION_REACTION
                end
end
```

```
static_diagram GUI
component
        class GuiInterface
                indexing
                        author: "Christian 'Troy' Jensen, chrj@itu.dk";
                feature
                        --Commands
                        Run
                        DrawHand
                                -> cards : SEQUENCE[CARD]
                        DrawAction
                                -> cards : SEQUENCE[CARD]
                        DrawDiscard
                                -> card : CARD
                        DrawDeck
                                -> filled : bool --Whether there are any cards in the deck
                        SetAction
                                -> number : INTEGER
                        SetBuys
                                -> number : INTEGER
                        SetCoins
                                -> number : INTEGER
                        EndGame
                                -> playerId : INTEGER
                        YourTurn
                                -> yourTurn : BOOLEAN
                        SetPhase
                                -> phase : INTEGER
                        UsedCards
                                -> cards : SEQUENCE[CARD]
                        SetPlayerNumber
                                -> id : INTEGER
        end
end
```

```
--NOTICE: This network design is based heavily on code I got
--from Simon Henriksen (shen@itu.dk) and where there are similarities
--between our code, he deserves the full credit for its design.

--Receiving

static_diagram NETWORK_CLUSTER
```

```
component
        class CONNECTION
                indexing
                        author: "Christian 'Troy' Jensen, chrj@itu.dk";
                feature
                        --Queries
                        GetClientIp : IPADDRESS --C# object
                        GetId : INTEGER
                        --Commands
                        Send
                                -> message : STRING
                        BeginRecieve

        end

        class SERVER
                indexing
                        author: "Christian 'Troy' Jensen, chrj@itu.dk";
                feature
                        --Queries
                        GetIp : IPADDRESS
                        GetClientList : SEQUENCE[CONNECTION]

                        --Commands
                        Start
                        SystemMessageToClient --Sent to a particular client
                                -> message : STRING
                                -> CONNECTION : CONNECTION --C# object
                        SystemMessageToAll --Sent to all clients
                                -> message : STRING
                        ForwardMessage
                                -> message : STRING
                                -> clientId : INTEGER
                                -> type : MESSAGETYPE

        end

        class CLIENT
                indexing
                        author: "Christian 'Troy' Jensen, chrj@itu.dk";
                feature
                        --Queries
                        GetComm : SOCKET --C# object

                        --Commands
                        BeginReceive

        end

        class NETWORKCONST
                indexing
                        author: "Christian 'Troy' Jensen, chrj@itu.dk";
                feature
                        --All these are constants
                        GetEncoder : UTF8ENCODING --C# object
                        GetPort : INTEGER
                        GetBuffersize : INTEGER
        end

        class NETWORKINGINTERFACE
                indexing
                        author: "Christian 'Troy' Jensen, chrj@itu.dk";
                feature
                        --Queries
                                IsServer : BOOLEAN
                                GetServerIP : STRING
                                SetNumberOfClients
                                        -> TotalClients : INTEGER
                        --Commands
                                SendTurnMessage : SEQUENCE[STRING]
                                --Responses from the other players
                                        -> Message : STRING
                                SendPreGameMessage
                                        -> Message : STRING
        end
end
```

References

[1] http://www.riograndegames.com/games.html?id=278

[2] http://www.riograndegames.com/uploads/Game/Game_278_gameRules.pdf

[3] Simon Henriksen shen@itu.dk