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

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.

We are planning on using a Model-View-Controller architecture. We want to separate our GUI from both the game rules and the state of the game via the controller. In essence we are likely to have a somewhat static model of the rules and a more dynamic and changing model of the state of the current game.

Frederik Lysgaard is the guy responsible for the design of our graphical interface. 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 Jensen is responsible for implementing the way the different cards interact with the state of the game when used. Christian is also the guy who will be looking into the networking portion of the project if/when it becomes relevant.

Jakob Melnyk is responsible for modeling the state of the game and the communication between the GUI and the model (in our model-view-controller architecture). Jakob Melnyk is also the "version-control-guy", the person with the final word in discussions and the general log-keeper for the group.

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 riogrande games. The walk-through 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 pic(clientinput) often one of the players will choose server and will then be able to give his fellow players who choosed client, a IP to connect to. pic(startserver) 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. pic(startgame) pic(startscreen)

4.1.2 Hot-Seat

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

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 there by giving you the edge in getting the most victory points which in the end determines who wins. I all ready 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 T

he user interface. pic (actioncard)

- 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 shown. Only actioncards can be drawn in this zone. When a turn ends the actionzone will be cleared and the actioncards will all be moved to the discard.
- 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 at new.
- 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 there by increase your decks size and strength, as we can see, we draw both the seven static victory/treasure cards and 10 extra kingdomcards, which also was one of our mandatory requirements.

4.4 e

nd of game. As stated by the Dominion game rules the game ends when either the province victory pile is empty or three kingdomcard piles are empty. When this happens we check which player has earned the titel of winer. If you are indeed the winner then you will be greated with a lovely congratulations message printed acrose the screen. pic(youarewinner) but if you loose you will be meet with disgust. pic(youloose) So to summarize our digital representation of Dominion let's you play with 10 preset kingdomcards in graphical interface with your friends either over hotseat or local area network. Which by total coincidence also is our mandatory requirements.

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`

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 Milestones

6.1 System analysis

6.2 System design

6.2.1 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.

6.2.2 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 exstensive 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 newbeginner 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 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 beforehand.

6.2.3 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 was 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.

6.2.4 Gamestate and Control (Game Logic)

Jakob Melnyk

6.2.5 BON specification

We have included all of our BON in section 7 on page 20.

6.3 System production

6.3.1 General

Our split into the different, very seperate 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.

6.3.2 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.

6.3.3 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 an 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 contains 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 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 GuiInterface, after removing the End of File text.

Then we are back at the GuiInterface, this is where things gets interesting. Depending on the message type, the GuiInterface 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 step: 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.

6.3.4 Gamestate and Control (Game Logic)

Jakob Melnyk

7 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
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 gam
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, han
        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
        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,
        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
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 gamest
        query
```

```
                "May I have the value 'v'?",
        constraint
                "The values allowed for this class are exactly one of 'DECK', 'DISCARD', 'HAND', 'SU
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 card
        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', 'BACKSID
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
        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 pi
                                  \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
                               \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
        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. Cost
        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 act
                                \ drawn this way; discard the set aside cards after the Player is fi
        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
        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
                                \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
        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
                                \and puts it on top of his deck (or reveals a hand with no Victory
        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 han
        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) r
                                \and the active Player decides to either put the card back or discar
        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 ca
                                \ the active Player can choose to trash one of them. The active play
                                \ 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
                                \ 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
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 /= voi
        end
        GetScores : SEQUENCE[INTEGER]

        --Ccmmands
        SetActivePlayer
            -> player : PLAYER
            require playerNumber >= 1 and playerNumber <= NumberOfPlayers and player /=
        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
        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
        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 (zone = DISCARD -> (GetDiscardSize /= 0))

                ensure GetTemporaryZone.Count = old GetTemporaryZone.Count + 1        and (z
                                and (zone = DISCARD -> GetDiscardSize = old GetDiscardSize -
        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
                ensure (GetTemporaryZone.Count = old GetTemporaryZone.Count - 1) and
                                (zone = DECK -> GetDeckSize = old GetDeckSize +1 ) and
                                (zone = DECK -> GetTopOfDeck = old GetTemporaryZone.Get(old
                                (zone = DISCARD -> GetDiscardSize = old GetDiscardSize + 1)
                                (zone = DISCARD -> GetTopOfDiscard = old GetTemporaryZone.Ge
                                (zone = HAND -> GetHand.Count = old GetHand.Count + 1) and G
                                (zone = HAND -> GetHand.Get(GetHand.Count - 1) = old GetTemp
                                (zone = PLAYED -> GetPlayed.Count = old Played.Count + 1) an
                                (zone = PLAYED -> GetPlayed.Get(Played.Count - 1) = old GetT
        end
        AddCardToZone
                -> card : CARD
                -> zone : ZONE
                require (zone = DECK or zone = DISCARD or zone = HAND or zone = PLAYED) and
                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)
                                (zone = PLAYED -> GetPlayed.Count = old GetPlayed.Count + 1)
                                (zone = DISCARD -> GetDiscardSize = old GetDiscardSize + 1)
                                (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
```

```
                                        and        (zone = HAND -> GetHand.Contains(card))
                                        and (zone = PLAYED -> GetPlayed.Contains(card))
                                        and (zone = DECK -> (GetDeckSize = 0 and GetDiscardSize = 0)
                                        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)
                                        (zone = DISCARD -> GetDiscardSize = old GetDiscardSize - 1)
                                        (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 =
                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 id
		--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 wante
			ensure Result = "COPPER" or Result = "GOLD" or Result = "SILVER" or
			Result = "CURSE" or Result = "ESTATE" or Result = "DUCHY" or Result = "PROVI
			Result = "CELLAR" or Result = "CHAPEL" or Result = "MOAT" or Result = "CHANC
			Result = "VILLAGE" or Result = "WOODCUTTER" or Result = "WORKSHOP" or
			Result = "BUREAUCRAT" or Result = "FEAST" or Result = "GARDENS" or Result =
			Result = "MONEYLENDER" or Result = "REMODEL" or Result = "SMITHY" or Result
			Result = "THIEF" or Result = "THRONE_ROOM" or Result = "COUNCIL_ROOM" or Res
			Result = "LABORATORY" or Result = "LIBRARY" or Result = "MARKET" or Result =
			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 player
                                        -> 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