SOFTWARE REQUIREMENTS SPECIFICATION

for

Dominion Logs Datamining

Prepared by
VER VALEM Willian
BAYOL Elmer
LOPEZ-FARFAN M. Liliana
TAGNAOUTI Khaoula

February 4, 2016

Contents

1 Introduction

1.1 Purpose

This Document has as purpose to clarify and describe as precisely as possible the requirements for the development of the program *Dominion DataMining*.
Dominion DataMining is a program which consists of 3 parts:

- **Data modeling** Responsible to read and remodel the data to be stored.
- **Data Storage** Store the parsed data
- **Data analyzer** Responsible to query the database and generate human readable statistics and plot charts.

1.2 Intended Audience and Reading Suggestions

This document is intended for any individual user ,developer, tester, project manager or documentation writer that needs to understand the basic system architecture and its specifications.
Here are the potential uses for each one of the reader types:

- **Developer:** The developer who wants to read, change, modify or add new requirements into the existing program, must firstly consult this document and update the requirements with appropriate manner so as to not destroy the actual meaning of them and pass the information correctly to the next phases of the development process.

- **User:** The user of this program reviews the diagrams and the specifications presented in this document and determines if the software has all the suitable requirements and if the software developer has implemented all of them.

- **Tester:** The tester needs this document to validate that the initial requirements of this program actually corresponds to the executable program correctly.

1.3 Project Scope

The *Dominion* card game has been running on a server from October 2010 to March 2013. The games played on this server have been recorded in *logs* that keep track of every players actions; Those *logs* have been made available to the public. But some information relative to decisions made by the players haven't been recorded.
A wiki about the game has been created, it offers expert advice in decision making during the game: it mainly offers advice on a strategy level. The goal of this project, submitted by Yvan Le Borgne, researcher at the Labri, is to process the *logs* (more than 12 million games) in order to answer several questions. It mainly consists in comparing the gathered data to the wiki's information in order to validate the usefulness of the ideas and informations given by experts. We will at least create validation tools offering enough efficiency. In order to fulfill this objective, the project's goal is to create a database containing all the games played and the to produce an analysis based on this information. To do so, our program will first have to extract the *log* data because they are too voluminous and too complicated to use directly (non-standardized structure). We will have to set up data structures allowing us to have a better data storage. A simple way to represent this data will be offered to the user. Some information is also missing in the *logs*(for example: cards drawn during a turn). We will try to find a method to rebuild this information and to add it to the existing information. We will also try to optimize the opeations made by the data analyzing by finding a balance between memorizing requests and re-computing. Finally, we will have to decide wich strategy is used with each game, or even recognize strategy changes during a game. We could also recognize ho applied a strategy first (who *inveted* it). If we can detect enough strategy related data, we can also detect the usual learning process of the players.

2 Overall Description

2.1 Product Functions

Following is a list of the expected functions to be available on the software.

- **User interface:**
  - Launch the log parsing
  - Statistics requests
  - Game data mapping
- **Data modeling:**
  - log decompression
  - parsing
  - create game-log
  - send game-log data
- **Data Storage:**
  - Create Relational Database
  - Connect to Relational Database
  - Create document-oriented Database
  - Connect to document-oriented Database
  - Compression
  - Load game-log
  - Save game-log
- **Data Analysis:**
  - Gather data
  - restore game-log
  - Plotting
  - Calculate ***ELO***
  - Recognize strategies
  - Recognize Greening

A more precise description of each functionality will be provided in Section 4 of this document.

2.2 User Classes and Characteristics

Physical Actors :

- **User :** The user will launch the parsing of the logs and demands analyses to be done on the stored data that will then be shown in the form of charts.
  The user's actions can be decomposed in two parts:
  - Precomputation step: This action will be performed once at the beginning and will launch the parsing phase and the standard analyzing of the games (strategy recognition, elo calculation)
  - User's data analyzing: This will be a loop in wich the user can ask for standard statistics display or create complex analyzing using python scripts and specific data queries. as long as the parsed data isn't erased or modified (new logs, new program, database deletion) this process can be repeated without having to reparse the data.

System Actors :

- **Parser :** The parser is the system that has write access to the data base and is responsible to store the parsed data on the database.
- **Analyzer :** The Analyzer is the system that will query the data base to collect data for analyzing, and will be able to create persistent queries.
- **Database:** The database will store the data extracted from the logs and answer to the queries of the other parts of the program

The Software to be developed is aimed to a personal use, so there is no privileged access, the user will have access to the intierty of the code and data.

2.3 Design and Implementation Constraints

This section presents an overview of possible issues that can limit the development of the Software:

- The Software will be developed for the operational system *Linux* and no other platform is necessary.
- The software has to run on hardware found on simple personal computers.
- The data to be analyzed is considerably big (400Gb) and the client will not provide a server to work with the data uncompressed what can impact on the performance of the software.

- The *Logs* provided by the client don't have an standard format and that can delay the time to develop the parser, as the problems that it can raise during the development are unknown at the moment.

3 The Logs

3.0.1 Data description

The data that was provided by the client is compressed in format tar.bz2 , one file for each day of *Log*, the total size of the compressed data is 13Gb.
The decompressed data makes a total of 400Gb.
Each *Log* file is in *HTML* format.
A *Log* should contain:

- **header** containing the game number and a winner.
- **match resume** containing the cards used on the match and how the match finished
- **player resume** containing each player victory cards, deck and points.
- **game log** that part contains all detailed moves made by the players during the match

Data inconsistency

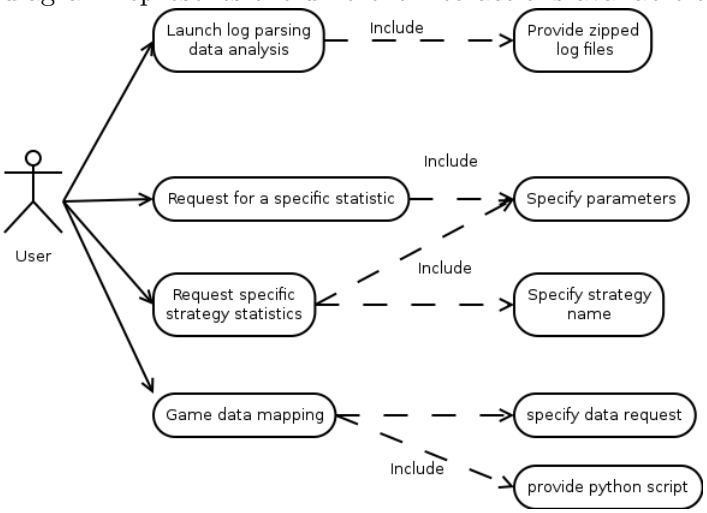
The *Logs* present some data inconsistency that can be problematic on the development of the parser. some problems found at the moment that this SRS is being write are:

- **Log number**
  The numeration of the logs aren't unique what implies that can't be used to identify a log.
- **User Names**
  The logs show that no restriction where made on the user names of the game and some user names have Keywords and special characters that can conflict with the parser.
- **Missing data**
  Some of the logs are missing part of the data (like: the header, users resume ...).
- **Log format**
  The syntax used to write the logs are not consistent and present some difference between logs.
- **Data compression**
  As the data is compressed and the equipment provided to work on the project can not handle the data uncompressed the project will have to work with the compressed data, and after a first benchmark, 4 hours was needed to decompress.

4 Functional Requirements

4.1 User interface

The program user interface will be composed of a executable(dominionmining) that will be executed with certain parameters to execute the desired action. The following diagram represents the different interactions available to the user:



Before the use of any features of the program , the user has to launch the parsing.

4.1.1 Launching the log parsing

Description and Priority

Priority level=high

- The user will have to place all the compressed logs (tar.bz2 files) to be parsed into a specific folder.
- Once the previous step is done, the user can execute the parsing by typing : *dominionmining parse (zlib/snappy)* if no compression is given the default compression is *snappy*.
- For testing purposes, in order to avoid having a 4 hours parsing step, the user can specify a percentage of the logs to be parsed. by specifing a value between 0 and 100. the parser will randomly select logs from the provided ones and parse them, creating a usable database for further testing.

- once the parsing is done a message will be shown on the console (parsing done).

4.1.2 Statistics requests

Description and Priority

Priority level=high
The request for a statistic can be made by typing *dominionmining stats (parameters).*
The possible strategy names are: bigmoney , penprovince , beyondsilver.
If other strategies are recognized by the program they will be added to the list.
The list of parameters that will be possible to use will be determined in the future.

The return of a statistic can be a chart displayed in a window or exported to a file. If its a single value, a name or a sentence it will be returned in the prompt.

4.1.3 Game data mapping

In order to allow the user to ask for specific statistics concerning a game, the program can be run in order to generate simplified game logs and then a python script will be applied to this log. This would allow more flexibility in the type of statistics displayable by the program.

- the program will be launched by using the following command: *dominionmining data_to_query user_script.py*
- the python script will be applied to the generated result and return the statistics asked by the user.
- The generated file containing the query results will have the following format:
  - The name of the produced file is *game.txt*
  - The simplified log will contain each action made by the player, one action per line. List of actions and their format:
    - Revealing a card: *player_x reveals card_name*
    - Drawing cards: *player_x draws n cards*
    - Buying a card: *player_x buys card_name*
    - Trashing cards: *player_x trashes n cards*
    - Puts a card: *player_x puts card_name*
    - Gaining a card: *player_x gains card_name*
    - Plays a card: *player_x plays card_name*

4.2 Data Modeling

4.2.1 Log decompression

Description and Priority

Priority level=high

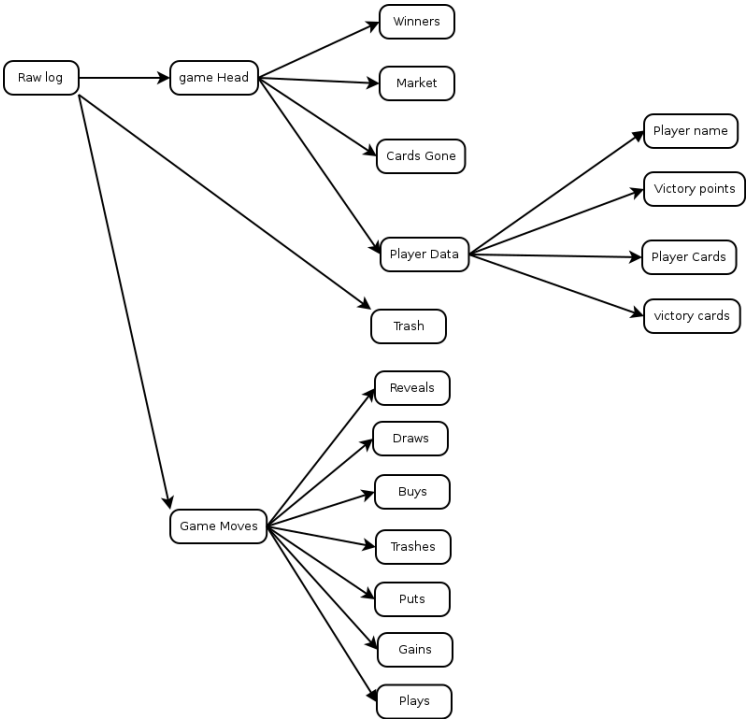
- The tar.bz2 files will be stored in a folder.
- The program will decompress one specified file from this folder in a temporary folder.
- The programm will delete the decompressed logs on demand by the parser.

4.2.2 Parser

Description and Priority

Priority level=high

Provided the issues revealed by the logs (like: the inconsistency of the syntax) a use of a parser made with tools like *YACC and LEX* is not recommended. For that a parser will be done by using the already present *HTML* and key words presents on the logs. The parser is responsible for reading the logs and collecting the important information without losing the structure of the log.
A glimpse of the data to be recognized by the parser is illustrated in this diagram (some data to be determined as all the possibilities inside a log haven't been revealed yet):



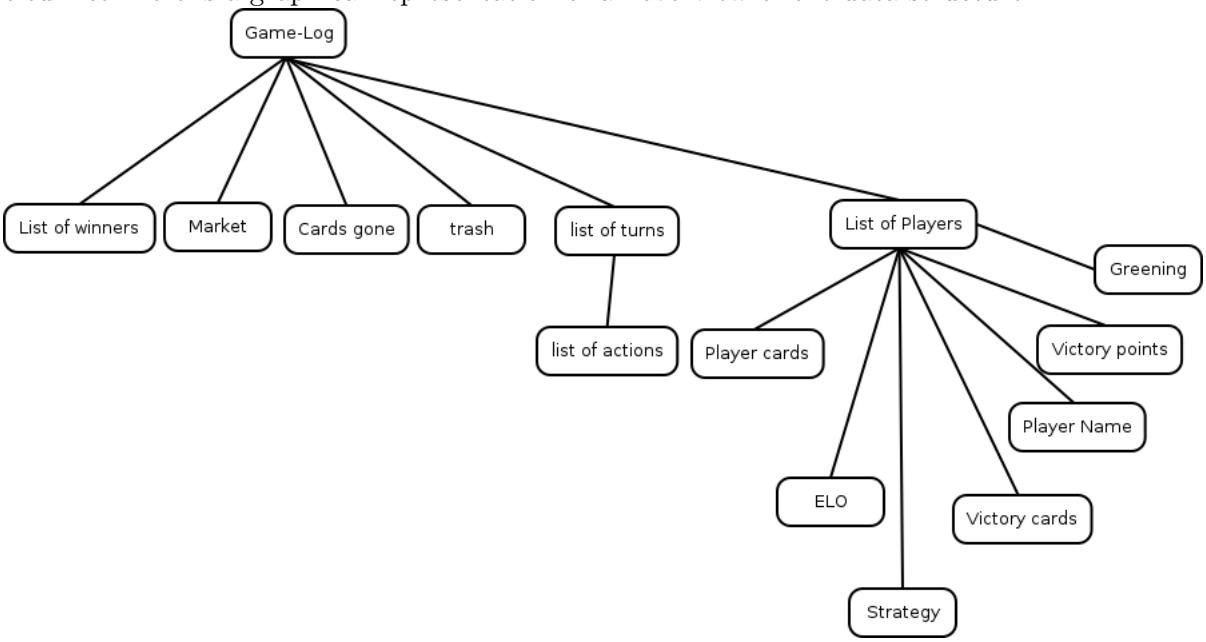
- Winners: The list of the winners of the game, it can be one winner or several winners if they are tied.
- Market: The list of 10 available cards to be bought by the players.
- Cards Gone: list of cards that have been entirely bought at the end of the game.
- Player Name: Name of the player.
- Victory points: Amount of points at the end of the game.
- Player cards: List of cards bought by the player with the amount they bought.
- Victory cards: List of victory cards the player with the bought.
- Trash: List of cards that that are discarded at the end of the game.
- Game Moves: Every action depending on this item correspond to an action performed during a player's turn.

The parser has to read every element shown in the previous graph as a part of the data when the user launches the parsing process.

4.2.3 Create game-log

Description and Priority

Priority level=high
This functionality is the creation of a data structure where the log data will be remodeled into. here is a graphical representation of an overview of the data structure:



4.2.4 Send game-log data

Description and Priority

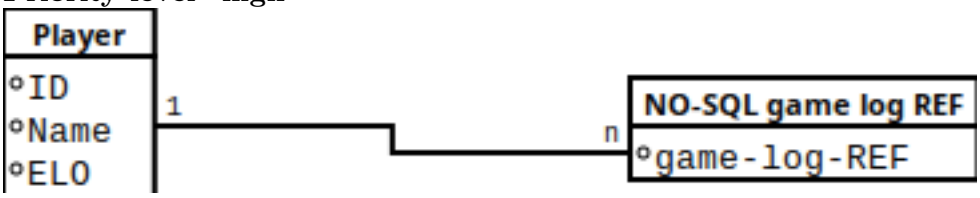
Priority level=high
The program will send the game-log data structure to the database part of the project.

4.3 Data Storage

4.3.1 Connect to Relational Database

Description and Priority

Priority level=high



- The program will be able to send data relative to the player (eg: ID, Name, Elo, Games played by the player)
- The program will be able to send requests to the database concerning every piece of data stored in it.
- The relational database will send the results of the requests made by the program.

4.3.2 Create document-oriented Database

Description and Priority

Priority level=high
This module will be responsible to create the document-oriented database containing data at the JSon format.

4.3.3 Connect to document-oriented Database

Description and Priority

Priority level=high

- The document-oriented database will be handling exchanges through a specific socket.
- The document-oriented database will receive game-logs at the JSon format.
- The document-oriented database will receive requests from the program concerning every piece of data contained in it.
- The document-oriented database will send results of the requests made by the program.

4.3.4 Compression

Description and Priority

Priority level=medium
The document-oriented database will hold most of the data and some level of compression will have to be applied. The database in focus to be used at the moment is *mongodb* and it offers 2 levels of compression *Zlib* and *Snappy*. During the development the *Snappy* will be used as it offers a better performance. But the final program should give the user the ability to chose the compression level when starting the log parsing.

4.3.5 Save game-log

Description and Priority

Priority level=high

- The document-oriented database will convert the game-log structure in a JSon formatted document.
- The produced data in JSon format will be stored in the database.

4.3.6 Load game-log

Description and Priority

Priority level=high
When asked for a game-log, the database will load the JSon formatted log in a game-log data structure indetical to the one described in the data modelling section.
If there is missing information in the JSon document, the corresponding field will be left empty.

4.4 Data Analysis

4.4.1 Gather data

Description and Priority

Priority level=high

- The data analyser will convert the query results in a format usable by the plotting module.
- If the data is simple to read (eg: a number, a name, a sentence), the data analyser will simply output the result in text format.
- The last query will be memorized with it's result in order to gain time if the next query made by the user is the same.
- If the user wants to apply a python script to game logs, the pogram will request the game logs and put them in a text file, this file will contain each action made by each player and the script will be applied to this file.

4.4.2 restore game-log

Description and Priority

Priority level=medium
This feature is responsible to restore the missing information on the parsed logs, by using

deduction based on the data obtained from the log.
In case of a missing game header or part of it missing:

- Winners/victory cards/victory points missing: the program can keep track of the cards owned by the players during the game in order to know how many victory cards they have at the end of the game.
- Market missing: The program can keep track of the cards bought during the game in order to partially or totally reconstruct the cards available in the Market.
- Cards Gone: As with a missing Market data, the program will keep track of cards bought and count the amount bought for each card, if the maximum amount of cards is bought, the card is gone at the end of the game.
- Player name missing: the game moves part of a log also keeps track of the player names, the program will simply restore them in the header.
- Player cards: the program will keep track of the actions performed during the game relative to the player cards and add them in this list.

4.4.3 Plotting

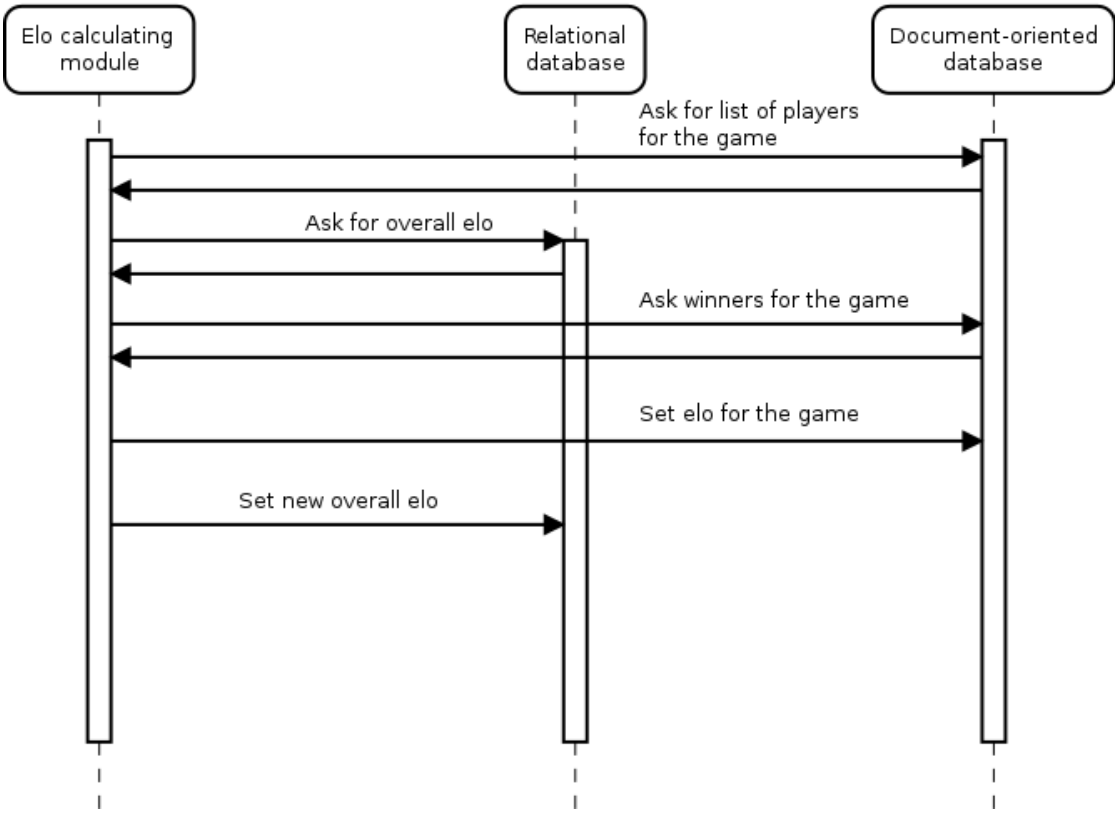
Description and Priority

Priority level=high
The plotting module will get the converted data obtained after the Gather Data process and will plot them in a graphical interface. This module will work in a similar way of *GnuPlot*.

4.4.4 Calculate ELO

Description and Priority

Priority level=high
The analyzer will have to calculate the final ELO for each player and what was a player ELO on a given game. For each game in chronological order, this module will request the initial elo of the players from the relational database and the winners of the game, then it will calculate the new elo of each player from the outcome of the game. The module will then set the calculated elo in the document-oriented database at the currently analyzed game and will also update the overall elo for each of the players participating in the game on the relational database.
More information about ELO can be found on https://en.wikipedia.org/wiki/Elo_rating_system
The following diagram illustrates the process of the elo calculation:



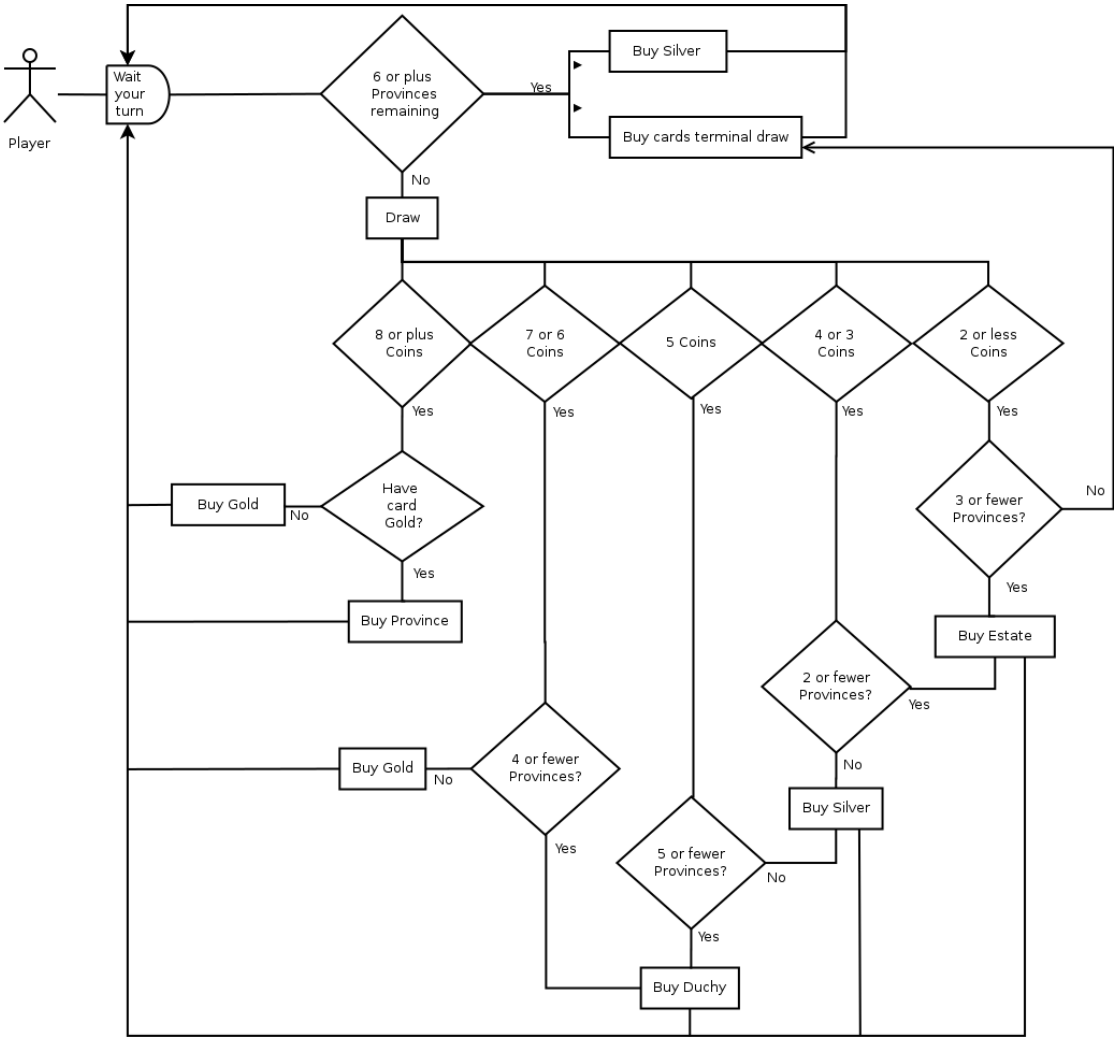
4.4.5 Recognize strategies

Description and Priority

Priority level=medium

The dominion wiki describes some strategies that can be used in the game .
For exemple :
Big Money
Beyond Silver
Penultimate Province Rule
More on http://wiki.dominionstrategy.com/index.php/Strategy

And the analyzer will have to be able to recognize what strategy was used on a given match (if any strategy was used) and generate statistics.
This image describes the decision process involved in the Big Money strategy



The analyzer has to recognize when a player buys only money cards and specific cards related to the big money strategy. It also has to keep track of the remaining provinces to be bought.
More on http://wiki.dominionstrategy.com/index.php/Big_Money

In order to recognize the strategy coined Beyond Silver, the analyzer has to recognize when certain type of cards are bought, the list of these cards can be found on. http://wiki.dominionstrategy.com/index.php/Silver#Beyond_Silver

In order to recognize that the Penultimate Province Rule is respected (as explained on http://dominionstrategy.com/2011/03/28/the-penultimate-province-rule/), the analyzer has to keep track of the amount of Victory Points of each player at every turn.

4.4.6 Recognize Greening

Description and Priority

Priority level=high
The analyzer has to be capable of recognize the ***greening*** moment on each match. More about ***greening*** on http://wiki.dominionstrategy.com/index.php/Greening.
The program will recognize when the greening happens by detecting when victory cards begin to be bought.

5 Nonfunctional Requirements

5.1 Performance Requirements

No specific performance requirement was made by the client. But for the analyzer a time of running should be less than an hour.
The user should be able to iterate the analysis process on a precise dataset without having to wait for the same query to be done again.

5.2 Reliability

The client demands that the maximum number of logs should be parsed , as the data provided has some inconsistencies and some logs wont be possible to parse. the client agrees that between 5 to 10% of the logs can be ignored.

The parser will have to parse 100% of the data present on the log and Restore any missing information if possible.

5.3 Software Quality Attributes

Even if just a command line is expected for the final program it should have a easy learn curve and be easy to use.
The data generated by the parser don't need to be human readable.
The game logs generated will have to be analyzeable by the user in order to allow as much flexiblity as possible in the treatment of the logs.