CHALMERS


# Development of a web-based card game engine

**Master of Science Thesis in the Programme Software Engineering and Technology**

***ANDREAS THURESSON***
***LINUS HANSSON***

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, September 2010

Development of a web-based card game engine

ANDREAS H.J. THURESSON,
LINUS K. HANSSON,



Examiner: SVEN-ARNE ANDREASSON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden September 2010

## Abstract

This report covers the development of a web based card game and is an attempt to construct a game that is general enough so that it will be able to encompass all possible card games. The report covers the development process, how it was designed and the final system produced as well as an analysis regarding mistakes, problems encountered and what could be improved. The system was developed using a client server architecture and JavaScript together with the library jQuery, which is meant to help speed up the development, were used to develop the client. The server part of the system was also developed using JavaScript, this was done with the help of Node.js which is a system that allows executing JavaScript without a web browser to run in a similar way to regular computer programs. The communication between the client and the server is done using reverse Ajax with JSON formatted commands through a custom protocol. The project was done in part as an attempt to see what is possible in regards to developing highly interactive web applications. The project was also a learning experience as the previous knowledge of JavaScript was very limited.

**Keywords**: JavaScript, web, development, game, card, software, ajax, node.js, jQuery

# Table of Contents

## List of abbreviations

| | |
|---|---|
| **DOM:** | Document Object Model. |
| **JSONP:** | JSON with Padding, used by web pages to request JSON data from other servers than the primary one. |
| **W3C:** | World Wide Web Consortium. |
| **DOS:** | Denial of Service. |
| **Mutex:** | Mutual exclusion, used to make sure two or more entities are not able to use a specific resource at the same time. |
| **Spoof:** | To conceal your identity and/or take someone else's. |
| **HTML5:** | New version of HTML. |
| **XML:** | Extensible Markup Language is a protocol for storing and transporting data. |
| **Bitmap:** | Bitmap is a way of representing images. |
| **HTTPS:** | HTTPS is like HTTP but with encryption. |
| **Acid2/3:** | Is a test for web browsers to determine if they are following the standards. |
| **CSS:** | Cascading Style Sheets is a language used to describe the graphical presentation and formatting in a document. |

# 1. Introduction

This is a report covering the topic of developing a highly interactive web application. More specifically this report will attempt to convey why the project was started, how the work was done and finally what results were achieved.

This report will detail the development of a card game system meant to, with as high of a degree as possible, encompass all existing card games. In short this will be achieved by attempting to mimic the possibilities of having a real table and a deck of cards in front of you. The resulting system will then be one in which the user dictates the rules as well as enforcing them while the system only attempts to provide an environment for the games. The system will at the same time facilitate an easy way for the user to, through the game, interact with one or several opponents when appropriate for the game style chosen. All in all this will be the main goal of the system developed namely a GUI that is generalized enough to be able to accommodate many games. When switching between different types of games, there will only be minor changes to the GUI for example the number of possible players or the option of having a hand of cards.

## 1.1. Background

In recent years Internet usage has increased greatly and it continues to do so. To illustrate this fact one can look at the increase during the period of 2000/12/31 - 2009/09/30 during which we can see a 380% increase [1] for the number of Internet users. From looking at this data the conclusion that can be drawn is that the Internet will only keep growing and that at this rate everyone will eventually be using the Internet. Already we have seen Internet access getting declared as a legal right [2], which clearly show how important the Internet is in today's society.

Following close behind in the wake of this is all new emerging web standards and techniques for web development because of the immense popularity of the Internet. HTML5 is one example of a new web standard currently being developed, at the time of writing it is in a so called "Last call" state [3] meaning its nearing completion. In fact some parts of the standard are considered stable enough for implementation in major web browsers. With the release of web standards like HTML5 and techniques for use during web development a lot of new possibilities opens up for developers.

In the very beginning of the Internet webpages were limited to displaying text, as has already been implied web design has progressed at a very quick pace since then. Today webpages are much more graphically appealing and the content is much more dynamic. Dynamic in this instance refers to, amongst other things, that webpages accept user input and has content that automatically gets updated without any user interaction. This type of dynamic content opens up all of these new possibilities for developers. This can for example be something seemingly simple as a rolling piece of text displaying the latest stock price updates for your company. It could also be something a lot more advanced with interaction not only with just you and the server but also any other users,

which can lead to web applications like a chat or a multi user whiteboard. This brings us to the subject at hand and more specifically the purpose of this project.

### 1.2. Purpose

The main purpose of this project is to develop a highly interactive web application with support for multiple users and the possibility of interaction amongst all the users. Furthermore the project will be an attempt to create an application that behaves similarly to the regular programs used on a computer e.g. your web browser, image editor or instant messaging client. The reasoning behind this is that it would provide you with benefits from both types of applications. For example if you could turn an application into a web application there would no longer be any need for users to install, update or otherwise manage the application since the same application could be accessed anywhere and by everyone. While the company providing the application need only update the application on their own servers. This of course at the same time opens up possibilities for new business models where the users pay for access to the service instead of paying for the programs themselves. However, web applications come with restrictions, namely that it requires Internet access and a compatible web browser but these restrictions can to some degree be limited by for example the use of a new technique called web storage [4]. Web storage allows web applications to more easily store data on the clientside which in some cases can remove the need for a server, thus reducing the need for internet access. One thing that is worth mentioning is that you can skip the use of any restricted plug-ins such as Microsoft's Silverlight or other similar closed source technologies. This means you will get the advantage of having more control during the development and possibly an easier time deploying your system when you do not need to install additional software.

One of the larger influences for this work has been the collection of web applications developed by Google e.g. Google Docs and Calendar. Because of Google's popularity their products are likely to reach a lot of the Internet users. This means that for many people Google's web applications might be the first encounter with web applications exhibiting the same kind of interaction possibilities as regular computer programs. In particular one can look at one of their latest projects Google Chromium OS [5] which in short is an operating system built entirely around web applications.

Lastly it should also be noted that this project held interest for us on a personal level in part because of the reasons already stated. Since web applications are likely to continue to increase in popularity, spending time on a project developing a web application was an appealing option. Our education thus far has focused on making us able to adapt and solve problems in different situations and programming languages. The creation of a web application is new to us and gives us the possibility to do just that, namely learn to adapt to a new type of application development.

## 2. Requirements

Research on existing systems, similar to this project's, laid a foundation for the functional requirements. This foundation mostly consists of different types of interaction with cards and decks, in order to support a wide array of games. During research inspiration was taken from Magic Workstation [6] and Generic Collectible Card Game [7], both of which are regular software systems that requires installation. If a system contains the functionality of the previously mentioned systems it was considered to be able to support plenty of card games.

### 2.1. Functional Requirements

The website should provide the users with the possibility of creating a user account, connected to the account will be game-configurations, e-mail address and other similar user specific data. Once signed in on an account, users should be able to create game-configurations as well as create and import cards. Players should also be able to chat with other players in a lobby, when not in a game session. In addition, users should also be able to start a new or join an existing game session.

Once a player has joined a game session a number of things to do will be available to him. The user should be shown a player area on which he is able to move, select, flip, create, remove, rotate and see the contents of both cards and decks. In addition to this also be able to change the data of given cards, add, remove and manipulate counters for players (counters can for example be used to keep score). A console should also be visible at all times when playing, in which messages for actions taken by the players in a game should be shown as well as show chat messages from players.

Some of these requirements have lower priority because development efforts should be put on creating the functionality needed to be able to play the game. For example, a lobby is a nice feature but is not considered to be required. In addition to this, there should be focus on making sure the non-functional requirements are met, because the result could be a greatly diminished user experience if they are not. More detailed requirements that were used during the development can be found in Appendix II.

### 2.2. Non-functional requirements

#### 2.2.1. Usability

The user should not have to spend more than five minutes to be able to start a game and also understand how to invite other players as well as how interaction with the system works.

#### 2.2.2. Availability

The system should under normal conditions posses a high availability. Meaning it should be accessible to users and not prone to crashing or becoming unaccessible for

other reasons.

#### 2.2.3. Reliability

The system should under normal conditions most of the time perform required functions successfully.

#### 2.2.4. Security

User passwords should be stored safely, such that retrieving them is infeasible, even if the server is compromised.

#### 2.2.5. Performance

Updating the game-state should not take longer than five seconds. Meaning, five seconds from an action is taken the changes should have propagated to all players.

There are more possible requirements that could be added, these were however not considered to be as important. Mostly because of the given time frame the project has but also the fact that some of the considered requirements would put restrictions on the development. These restrictions could result in increased development time and possibly divert attention from the critical requirements.

# 3. Analysis

## 3.1. Development Process

Previous experience in this area directed the initial investigations regarding what development process to use for the project to the various agile processes. The reasoning here being that Agile processes are designed with a small group in mind and driven by close cooperation within that group. Since this project will consist of only two developers being able to utilize the process in a very small group was a very important factor. During this selection process many processes were investigated but as already mentioned the focus was on agile processes. So following that line of thought the processes we investigated more closely were Scrum, feature driven development, lean software development, test driven development and agile unified process. Some time was also spent investigating individual agile methodologies like the practices of extreme programming.

However having looked at the available alternatives the conclusion was that they are not optimal in this case with such a small team. When choosing a process you are supposed to customize the process to suit your needs but after having looked at the different alternatives, there was not one single process that fully match the needs for this project. The solution to get around this was to attempt to pick out the parts of one process that would suit the needs of the project well and then to do the same for the others, in essence combining several of the processes. This meant that the end result was a collection of practices which were found to be the most helpful for the development in a two man team.

Being a collection of practices it can not be stated precisely what process that was followed during the development but the core of the process was meant to be kept agile and in line with the agile manifesto [8]. Following are a few examples, to better illustrate the actual process, of practices that were deemed especially well suited for the project.

First, the practice that is possibly the most well known of all agile practices, pair programming. This choice was made mainly for the reasons of better produced code, less time spent working on more difficult parts of the code and also to facilitate a shared understanding of the code and the entire system. However it was also decided that this should not be used at all times, if it was shown to hamper productivity, this can for example be the case when implementing trivial parts of the system. In those cases working in parallel have the possibility of increasing productivity more than the benefits of pair programming would do.

Second, both a class diagram and a domain model was created and although these are not strictly agile practices a domain model could in this case work as a so called system metaphor. This is something that can be very useful when discussing further

development of the system. The class diagram on the other hand was more of a way to construct the initial design of the system and to make sure that the design did not miss any important parts. The reasoning was then that since the human brain is much better suited for analyzing something if you have a graphical representation to look at. If you instead were to attempt to keep track of all the information concerning the system design you would likely have a much harder time. This would be even further complicated by having to try to explain this mental model to a partner while avoiding misunderstandings.

### 3.2. Domain Model

Figure 3.1 shows the domain model of the systems which shows the entities in the system. Users of the system should be able to play the game through a client, which a user interface is considered to be a part of. The table and hand are entities where the user places their cards and decks. The server should be able to handle multiple clients and multiple games at the same time, as well as being able to import decks and cards into a game from the database. Cardgame will identify a group of players, each in the same game, and in combination with cards and decks this will represent the state of a game.

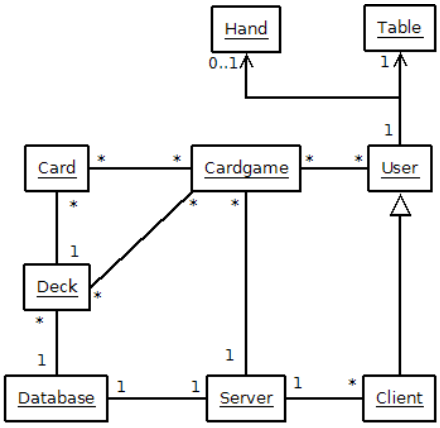



*Figure 3.1.* Domain Model

### 3.3. Theory

The following section contains information connected to this project, this includes programming languages, techniques and tools.

#### 3.3.1. JavaScript

JavaScript is an object oriented scripting language primarily used as part of web pages and the scripts are either included or embedded inside an HTML document. JavaScript supports the same structured programming syntax as C, for example if,while and for statements. JavaScript employs dynamic typing, this means that types are associated with values rather than variables. JavaScript uses prototype based programming, which basically means that inheritance is done through prototypes instead of classes. With prototypes you can add a property or method to all instances of an object, as well as create subclasses. This could be used on the JavaScript prebuild object String, you could for example add a function for printing the string in a special way e.g. upside down, backwards or randomly ordered. Objects are associative arrays where values can be either added, changed or removed, at runtime. Creating an object is simply done by prepending "new" before a function call. With JavaScript one can create mouse events, such as mouse over or click, quite easily. It also has functionality for accessing html elements such as pictures or frames, which can be used by developers to create highly interactive web pages.

JavaScript is normally executed on the client side, and there are many positive things

with this. For one, there is a lot of work that can be computed by each client instead of forcing the server do the computations. This increases the amount of users a server can handle at a time. It also increases how interactive a web page is, when for example hovering over drop down menus the server does not need to be involved. The client can simply handle it locally and whatever action is to be taken will take place instantly.

Client side JavaScripts does however not only provide advantages but also has some downsides. One downside is that it can be used for malicious purposes. JavaScript could be used to exploit vulnerabilities in a browser, one example of this is a buffer overflow attack. Other possible vulnerabilities can lie in browser plugins, for example video players. In other words clients should be conservative with what sites they trust, but the problem of security goes both ways. The JavaScript code is sent to the client, and because the client controls the execution of the code, the server can not trust that the data computed by the client was correctly computed. This limitation forces the server to in some cases inspect data supplied by a client if correctness is to be ensured. Another thing that should be mentioned is that JavaScripts are run in a sandbox by the web browser. The sandbox is there to limit for example file manipulation since most of the time this is not something you want a website to have access to. A sandbox is a technique that works by encapsulating a program, restricting it to a controlled environment.

JavaScript is normally not compiled but simply interpreted in the runtime environment supplied by the browser. Google has created a JavaScript engine known as V8 [9], which comes bundled with the Google Chrome Browser. V8 compiles JavaScript's to machine code, along with other optimizations this increases the speed at which JavaScripts run. V8 might not be suitable for all applications as it was specifically designed for high performance on large JavaScript applications.

JavaScript is widely supported by the major web browsers, however it is not strictly limited to being executed on web browsers where it's restricted by the JavaScript sandbox. JavaScript can also be used for other purposes, like for instance as a part of Firefox add-ons or as one of the supported scripting languages in OpenOffice. With help of the Windows Script Host one can also run scripts outside of the safe sandbox, able to run them as any application with similar functionality as a .bat or .vbs file. It is possible run JavaScript on a server as well, one can for example create JavaScripts and let them run as a server by using a system called node.js. This means that you can have JavaScripts running on both the client side and the server side which can simplify communication. Node.js will be explained in more detail later in section 3.3.4.

Because of the popularity of JavaScript a lot of frameworks and libraries have been developed over the years. The following sections will cover some of the more relevant ones, most of which were considered as a possible resource for this project.

#### 3.3.2. jQuery

jQuery is one of the many JavaScript libraries and it focuses on handling events, animation and user interaction [10]. Developers use it because it quite significantly

simplifies the coding process, many things can be done with just a few jQuery calls. As an example of how jQuery can simplify development there is something called a selector with which you can easily select an object of an HTML document, after selection it is just as easy to bind events or animations to it, modify its attributes or change the contents of it. An example of an event could be when you click on a given object.
Figure 3.2 shows an example of how to hide all the div tags in a document. The JavaScript code starts of by searching through the document for div's and stores the list in the variable 'divs'. By looping through all the elements and changing the css display

**JavaScript**

```
var divs = document.getElementsByTagName("div");
for(var i = 0; i < divs.length; i++) {
    divs[i].style.display = "none";
}
```

**jQuery**

```
$("div").hide();
```

*Figure 3.2.* JavaScript and jQuery example.

style to "none" the elements are hidden. With jQuery one can do the same thing with a much smaller amount of code. This illustrates the power of jQuery, animations for how the elements disappear is just as easy to add. The jQuery code also shows how to use the powerful selector, although this is a quite simple example selectors gives you a lot of ways to select precisely the elements you want to access.

### 3.3.3. Ajax

To be able to construct a more dynamic website you will need a technique for communicating with the server through JavaScript, otherwise you would be limited to only receiving new data when the page is reloaded. This is where the Ajax technique comes in, Ajax is short for asynchronous JavaScript and XML and by using Ajax your JavaScripts can request data from the server without the need to reload the page. This is done by creating something called a XMLHttpRequest which in turn lets you send a request to the server.

It is possible to take one more step to make the website even more dynamic and this can be done by reversing the Ajax technique. This means that when something happens on the server the client will know right away and update the web page accordingly. Something in this case could for example be when the price changes on your favorite stock exchange website or another client signs on to the community website you are currently using.

There exists several different methods for utilizing Ajax like this, one way of summarizing them is to arrange them under three different categories polling, long polling and streaming. The first one, polling simply works as follows, the client sends a request to the server, when it receives the reply the client process the reply and immediately sends another request. This way it constantly asks for new data it wants or checks if the data has been changed, understandably the result will be many requests being sent to the server. Sometimes this can mean there will be a delay of up to the same amount of time, that there is in between two requests, before the client recieves data that has been updated by the server. This can be seen in the polling example in figure 3.3 the first time new data is received by the server.

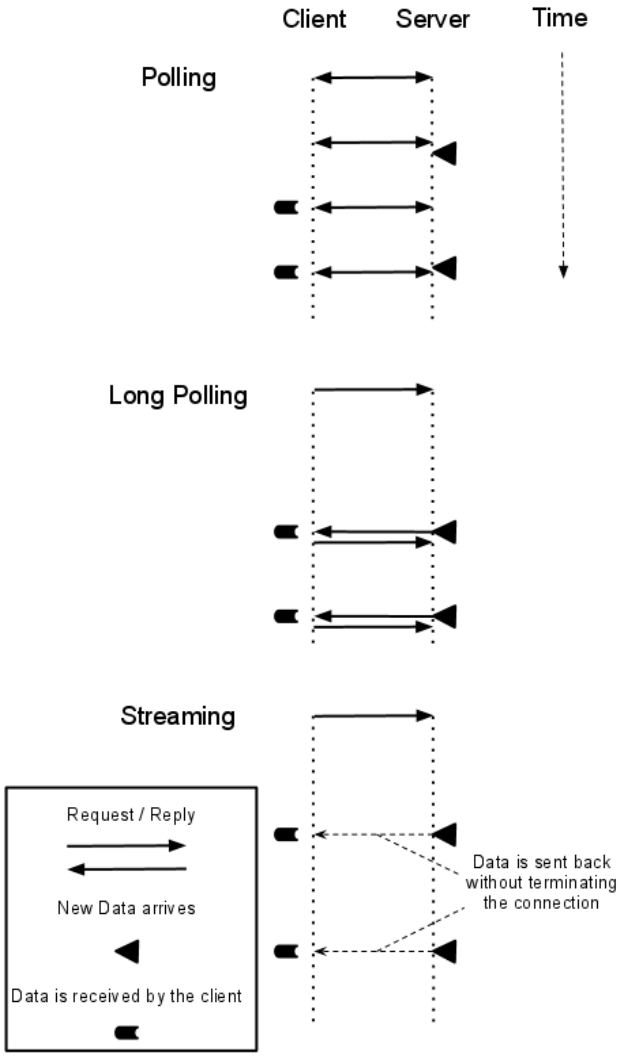



*Figure 3.3.* Illustration of AJAX techniques.

The second one, long polling uses a different approach, instead of the client continuously checking for updates by sending a lot of requests to the server only a single request is sent. For this to work the server keeps this single request open and does not respond to it immediately instead it waits until it has some data to send back to the client. This means the server will be able to push data to the clients and there will be a lot less traffic needed as illustrated by the long polling example in figure 3.3. It could even mean that you get a faster response from the server partly because the workload in terms of requests needing to be sent out is reduced. Another reason is that because the server will know precisely when new data arrives it can send the data to the client right away. When the client finally receives the response from the server it will again send a new request but in contrast to the first polling example this will only happen once for each piece of data sent.

Then finally there's streaming, to illustrate streaming one example of how it can be

implemented is very similar to long polling. The only major difference between the two implementations is that in the case of streaming the connection is kept open. Each time new data is received by the client it is processed but since the connection is not closed it is possible to receive more data from the server. Figure 3.3 has an example that shows how this works. In contrast to long polling this technique saves bandwidth by having to send less requests and time by not having to re establish the connection each time data should be sent. There is however some problems that can arise with this technique because of its implementation. Originally this was not how ajax requests were intended to be utilized so you will need to somehow handle cases where the connection is lost. This can happen for example if the user clicks the stop button on his web browser. This way of pushing data to clients might work better in the future with the implementation of web sockets [11] which will work in a similar fashion as streaming. Web sockets is a new standard being implemented for creating efficient bi-directional channels for communication between a server and a web browser.

### 3.3.4. Node.js

Node.js [12] is a system that lets you create network programs using JavaScript without the need for a web browser. This is done with the help of google's V8 , this as well as the use of JavaScript without a web browser was also mentioned in section 3.3.1, to provide performance comparable to regular computer programs. Node lets you write your program in JavaScript with the help of an API, you can then run the program with the help of Node which compiles the program before running it. Node was designed with a lot of focus being placed on scalability and to utilize non-blocking calls, for example if the Node program was going to preform file operations this will be a non-blocking operation for Node. Node also has the benefit of only using a very small footprint in memory for each new connection. This coupled with the non-blocking nature of the design provides a possibility to create programs that are very will suited for acting as a backend server for web applications. Because it is also using JavaScript as the programing language the communication between the web application and the server can become very easy to implement. Node is also well prepaired to handle the HTTP protocol which most web applications also use. Node is however still under development but it is far enough along so that it would be possible to use it in the project. Another thing worth mentioning is that Node gives you full control of the server implementation. This is not the case if you were to choose for example APE, that is covered in the next section, which provides a full implementation of the communication protocoll between the server and the client.

### 3.3.5 APE

APE is short for Ajax Push Engine and consists of a so called APE server and framework [13]. The framework will facilitate communications between client and server, in APE's own protocol. This framework however is optional and not needed to use the APE server, it is only a tool and can be reimplemented to better suit your needs or to better fit together with other JavaScript frameworks. The communication is done by a reverse Ajax technique which APE supports, for example long polling, streaming and JSONP. APE does not require any installation on the client side since it only uses JavaScript and thus supports most, if not all, web browsers.

In order to simplify sending data to multiple users you can create so called channels, any client who has joined it will be able to send and receive data. Both the commands sent to the server and data received are in the JSON(JavaScript Object Notation) format where JSON is a an open standard for human readable data. JSON is language independent but it is most often used in conjunction with JavaScript. The APE server comes with built in commands but it is possible to add your own through server side JavaScripts. In addition to this APE also supports MySQL out of the box.

### 3.3.6 Raphaël

Raphaël [14] is a JavaScript library for drawing SVG (Scalable Vector Graphics) vector graphics on web pages. Raphaël also supports VML (Vector Markup Language). Internet Explorer alone implements VML and other browsers support SVG. SVG is a platform for two dimensional vector graphics and consists of two parts, the open XML file format and the programming API. With SVG you can create three different types of objects, these are normal bitmap graphics, text and of course vector graphics. In addition to this the SVG language also contains basic syntax of PDL (Page Description Language) like for example PDF's. In order to better support hard copy printouts a specification known as SVG Print is in development [15]. Animations are possible through either scripting, css styles through the DOM or by using SMIL (Synchronized Multimedia Integration Language). SMIL is an XML markup language for timing events, animations and media embedding such as video and audio. ECMAScript is used as the default scripting language and is close to the same as JavaScript. The performance of SVG is quickly covered in the next section with a comparison to the similar technique called canvas of the HTML5 standard.

### 3.3.7 Processing.js

Processing.js [16] is a ported version of the programing language Processing, which in short is an open source programming language with focus on graphics. Processing.js is a port taking advantage of the new canvas element [17] that is being introduced with HTML5. The canvas element gives you the possibility to render 2D shapes in a defined area of your web page. As with HTML5 canvas is a new type of element but it is already supported by many of the major web browsers. In contrast to the similar technique SVG mentioned above which is based on vector graphics canvas is instead using a bitmap. Utilizing a bitmap can mean that you will gain performance in some areas but you might also loose some in others. As an example of this one can look at rendering times for for objects, a quite large difference can be seen when looking at the performance when rendering many objects versus using a larger area for rendering [18]. In the case when the number of objects rendered increases the performance of SVG starts to degrade however in the second case of increasing the drawing area the same can be said about canvas.

Processing.js provides a lot of methods that can be useful when you want to do graphical rendering. This can be for example methods for rendering 2D shapes such as a line, a rectangle or a triangle it also gives you the ability to render some 3D objects like a cube or a sphere. Processing.js provides methods for interaction as well like pressing a

key or clicking the mouse furthermore it also provides many more additional methods that can be useful, for example math functions. Processing.js is as its parent Processing an open language and it gives you a simple syntax to use for the development of anything from a diagram for visualising data to a web based computer game.

# 4. Design

## 4.1. Architecture

The system's architecture follows the client-server model which is commonly used when it comes to network applications. Another possible architecture that was considered is the peer-to-peer architecture, which brings benefits such as better use of available bandwidth and distribution of computing power. Implementing a peer-to-peer architecture could greatly increase the complexity of the system, firewalls might for example block traffic and this would need to be worked around. IP addresses for all players would have to be known by all clients, thus sending commands to another player would be very simple. In comparison to the client-server model quite a bit of security is lost as the data sent from a client cannot be trusted, simply because the client can be modified. In addition to this Ajax does not support communication outside of the website's domain, a solution for this could be a proxy server but the most of the advantages would then be lost. Another problem with peer-to-peer solutions is that they might require user involvement such as configuring routers or firewalls which is not considered a viable solution. With one server there is only a single point of failure, at the same time the clients are protected from each other and the possibility for cheating can be reduced. Centralization of data ensures that all clients will have easy access to the complete set of state data at any point in time, as well as knowing that the state is the correct one.

A multitier architecture of the system is shown in figure 4.1, in addition to the four tiers the figure is also split in half with the client to the left and the server to the right. A common architecture used within web application development is the so called three tier architecture which is the same type we use except for the addition of the communication tier. This type of architecture produces reusable modules and is perfect for when there a better library for creation and manipulation of the GUI has been released. All of the tiers are designed so that they can be changed or upgraded without affecting the rest of the system.

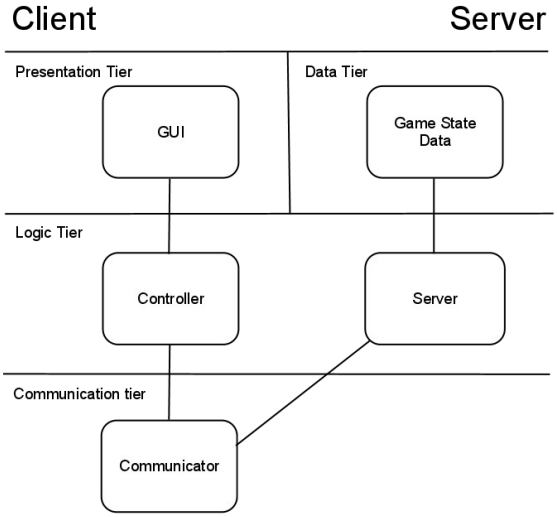



*Figure 4.1.* Illustration of the multitier architecture.

All of the computing and change of state will stem from some type of

action coming either from the server or, more likely, the client. Hence the systems follows an event driven architecture pattern where the client generates the event in the GUI. It is then up to the Controller to decide if the event needs to be forwarded to the server, through the communicator, or handled locally. Events generated by other clients will be received by the communicator and be processed by the controller. The GUI will continuously be updated as events are processed.

The system will be developed as a plugin, such that it is easy to embed it into another web page. An example of such is any game site with users and chats that can provide matchmaking opportunities for players wanting to use our system. The reasoning behind this decision was quite simple. Because we follow a multitier architecture the tiers should be quite independent, developing the system as an independent plugin originated from this.

### 4.2. Procedure

During the process of analyzing the available frameworks, tools and techniques in preparation for the development possible solutions was discovered. Each choice would be a weighing of the strengths of every technique against the potential weaknesses found. Following will be a text meant to go into detail about the decisions concerned with the design of the software system as well as the motivation and reasoning behind the decisions.

#### 4.2.1. Tools

The first decision that had to be made and at the same time possibly the most important one was the programming language to be used for the development of the software system. This is a very important decision because of the limitations it can place on the development but also the benefits it can provide. One benefit can be for example be how widespread it is since in the case of it being widely spread it would mean you will have access to a lot of work that has already been done e.g. libraries. This in turn might provide large benefits since you will have an easier time finding solutions to common problems. On the other hand if you run into a problem when using a language that is not as widely used you might be forced to solve the problem without any assistance.

For this project the choices were somewhat limited because of the need for the language to be able to be integrated into a web page. This means that there were three major candidates to choose from, namely JavaScript, Flash and Java applet. All of the three candidates are widely supported today, however out of the three JavaScript is the most supported since most web browsers support it without the need to install additional plug-ins. Flash and Java does require additional software to work which could cause some limits for the clients that are able to use the system. Flash is also a proprietary technology which could cause additional limits for both users and developers alike. From all of this information it was not possible to completely eliminate some alternatives. Therefore what likely had the largest effect on the outcome of the decision were an interest in developing using JavaScript in conjugation with some of the newer techniques made possible through for example HTML5.

The next decision in line was if any libraries were to be used. One of the most popular libraries for JavaScript is called jQuery, in fact when looking at the 10 000 most popular sites on the web as much as 30% are currently using jQuery [19]. Because of a lack of experience working with any JavaScript libraries previously this seemed like a very good argument for picking jQuery to help simplify development. The reasoning behind this is much the same as for why it can be a good idea to choose a popular language for development. Since jQuery is so widely used it also has a large community and many resources that will be very helpful when learning to work with a new library. Furthermore jQuery also has an impressive feature set which serves to convince on it's own as can be seen in section 3.3.2 jQuery is not the only option just to mention a few there is also Dojo, MooTools and Prototype. These libraries were also investigated but ultimately jQuery was chosen because of the reasons covered here and the benefits described in the jQuery section.

A great deal of time was also spent researching various libraries meant to help with the development of graphical web applications using JavaScript. The majority of the time was spent looking at the two competing techniques the first one was SVG, the second was the newer canvas element from the HTML5 standard. For both techniques libraries were found that provided similar functionality for making the development with the specific technique as simple as possible. The first library was called Raphaël which was a library for the former technique while the second library called Processing.js was implemented using the latter. Both of these libraries are discussed in greater detail in section 3.3.6-7 By looking closely at both of the techniques and examples of each the conclusion was drawn that SVG with the library Raphaël was the choice that performance wise seemed be the best fit for the card game scenario. However further investigation was done into the subject and studying of what would be required graphically by the card game. This finally resulted in the realization that both of these techniques provided much more in terms of graphical rendering than would be needed. Because of the relatively simple look of cards and decks of cards it was decided that standard HTML combined with CSS and JavaScript would suffice for this project. It is very likely that both SVG and canvas also would have worked very well but it would have meant a lot of extra work. Extra work in this case would be both learning to work with another library but it would also have meant spending a lot more time designing each graphical component. With the choice of using HTML with CSS it is possible to use the standard components that are provided and to with ease use jQuery to help with programming the user interaction.

The use of Ajax also required a decision to be made regarding which of the different variations to utilize for allowing the server to be able to push data to a client when new data is received e.g. when the server receives a command for moving one of the cards from a different client. To be more precise the decision is of which of the so called reverse ajax techniques to use of the ones explained in section 3.3.3 Since the system potentially will be serving a lot of simultaneous users polling might not be a good choice because of the bandwidth requirements. Also because we want to keep the delays very low since we are developing a game it would likely be better if the server could

have the possibility of sending out new data as soon as it becomes available. With polling this is not the case as the delay will be dependent on a combination of timing and the interval for each polling of new data. Timing in this case refers to the points in time when new data is obtained by the server. If the server were to get the new data right before it receives a new poll request it will be able to send out the new data right away. However if it happens right after a request it will be forced to wait for the next one.

This leaves the two similar techniques long polling and streaming both of which have the benefit of being able to send data as soon as it is received and the mentioned bandwidth disadvantages for polling. The reasons behind the final decision to choose long polling was firstly that it seemed like that solution would be a bit more robust. This is because it would not be as dependent on maintaining an open connection since if one connection is lost the server could just respond to the next one instead. Secondly long polling follows the standard more closely, keeping an ajax connection open for extended periods of time could potentially cause problems with firewalls.

Following from the use of Ajax there was also a need for setting up a server to handle the client requests. Either this had to be done by building a server from the ground up or finding a finished implementation for a server that will suit all or most of your needs. From the start the latter was considered to be the preferable choice of the two alternatives. The reason for this was that a server implementation was found that seemed to meet all our needs, this server was called APE and is covered in more detail in section 3.3.5. However upon further consideration it was decided that Node.js was likely a better solution. The reasons for this change was mainly that a quick test of APE itself revealed that the usability was not quite as good as was hoped which was needed because of how APE deals with session management. From investigating this more closely it was then discovered that if multiple clients were launched and then closed again, after a while this would prevent further clients from establishing a connection to the server before the first ones had timed out.

All of these small problems with APE led to the change of server implementation to Node.js. Switching to Node.js also means full control over how the server is going to work. This means that the implementation will not have additional features that are not needed which could serve to increase performance. It will also lead to a better understanding of all parts of the project and Node.js will also provide better possibilities to decide precisely how all functions are implemented. This can help to make the server and the client work even better together than would have been possible with APE.

Finally a decision had to be made regarding how the system was going to handle data storage. The main thing that needs to be stored are decks of cards since it would quickly become quite tedious to be forced to manually recreate the deck each time you want to play. The different options considered were to use a full fledged SQL database, a more simple key-value store or to use our own system and write the data directly to a file. Key-value store is simply a system for storing data that that is indexed with a key like an associative array. For the purpose of only storing data for decks of cards a very

simple solution would likely suffice although performance of the solution should also be considered. The system is meant to be integrated into a larger system which will need to keep track of for example user details. This means a more sophisticated solution will be a good idea to maintain a more structured data set. For this reason the solution that was finnaly decided upon was a MySQL database which will be able to support all the needs and enable storing of data in a uniform way. A MySQL database was also chosen because of how popular it is and because the development teams familiarity with it from before the start of the project.

### 4.2.2. Methods

This section documents a few additional design decisions that were taken in regard to development methods. The first one out of these decisions is concerning standards, and browser compliance or compatibility. Ideally compatibility between web browsers should not be an issue and this would be the case if all of them followed the same web standards, this is however not always the case. For example Internet Explorer which is the most widely used web browser has traditionally been considered bad at following web standards. This can easily be seen when looking at acid2 or acid3 tests which are meant for testing compliance to standards. Firefox on the other hand preforms better in these tests and is today the second most popular browser at 25% market share against Internet Explorer's 60% [20]. Firefox also allows for using an extension called Firebug which can simplify development and debugging. This is achieved by for example allowing on the fly editing and making more information easily available when you are debugging your code. It was decided that Firefox would be the primary environment used for testing the project. This decision was done since it would require a lot of extra time to construct work around to patch for example if Internet Explorer is show not to follow the standard correctly in some cases.

Since the system is meant to have a lot of user interaction a good user interface had to be designed. This was done with the help of simple sketches which helped to give insights in both how it could look when it is done and made it easier to spot parts of the design that might be missing. The sketch was used to iterate over the design many times by making several subsequent sketches until it covered all features in an aesthetically pleasing way. The design was also an attempt at making an as simple interface as possible as well as trying to make the interaction as intuitive as possible. These sketches served as simple prototypes of the system and was used as a basis for discussion about what features and functionality that needed be included. Also they were used to weigh the different alternatives against each other which is easier to do with a visual representation. It was also done to be able to convey the ideas of the different parties of the development team in a way that would reduce the risks of misunderstandings. As an example the most recent of these sketches can bee seen in figure 4.4, found in section 4.3.1.

Because of the agile style of the development process, the small size of the development team and how closely the team will work there will be a reduced need for documentation. A lot of the documentation that will be done will be in the form of code documentation to help with for example code re-factoring. However the same is true for

code documentation as with all other documentation and it will be attempted to in some cases forgo code documentation for self documenting code. This can be for example that instead of naming a method something short like "set" to save time. You could instead call it something along the lines of setCardName which in many cases can be enough to make you understand what the function does. In the latter case longer variable names can however cause an additional strain because the code will not be complied and it will be sent to the users while taking up bandwidth. This problem can be solved by manually changing all the names of variables or functions as a last step in the development. It is also possible to take it even further and remove all indentation new lines and spaces to save even more bandwidth. It is also possible to compress the code before sending it to clients either by the web server or the client itself can include a decompression algorithm for executing compressed code. There are also many tools available to do this kinds of compression automatically on the code which makes it very easy to utilize.

### 4.3. Detailed design

Figure 4.2 shows the class diagram of the server system and figure 4.3 shows the class diagram of the client side. Though both class diagrams show card and deck classes, the server will be keeping track of the full game state. The classes on the client side will be used to create representations of the real objects. For example, a card in another players hand will contain no values other than an Id for identifying it when actions are taken or made and decks on the client side will never contain its' cards, the important data will be the number of cards and possibly the topmost card if the deck is flipped. Put simply, the state of a game will be saved on the server side.

In order to successfully support reading and writing data, referring to cards inside a deck, some type of lock is needed. This is somewhat counter intuitive as the server does not have any concurrent threads. The base of the problem lies in that the contents of a deck is not a part of the client state e.g. if a client looks at the cards in a deck and another player draws any card from that deck, the first players knowledge of the deck will not be updated, hence they could try to draw the same card. The solution for this is to implement a lock in the deck which should follow a variation of the read-write lock pattern. This pattern is normally used within concurrent programming but in this case the behaviour consists of two parts. The first one is that a client should only be able to change the deck contents if it was the most recent client to conduct a read operation on that deck. Put simply, the last player to conduct a read operation will be in possession of the mutex, though any other

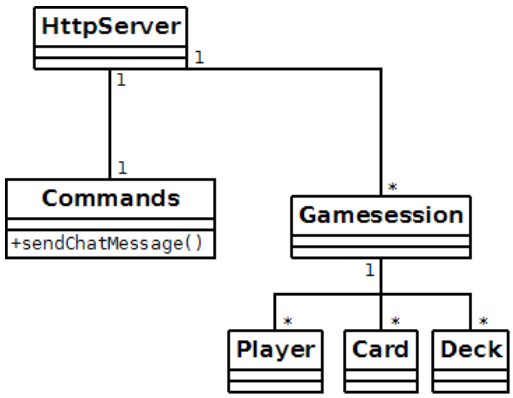



*Figure 4.2.* Class Diagram of the server.

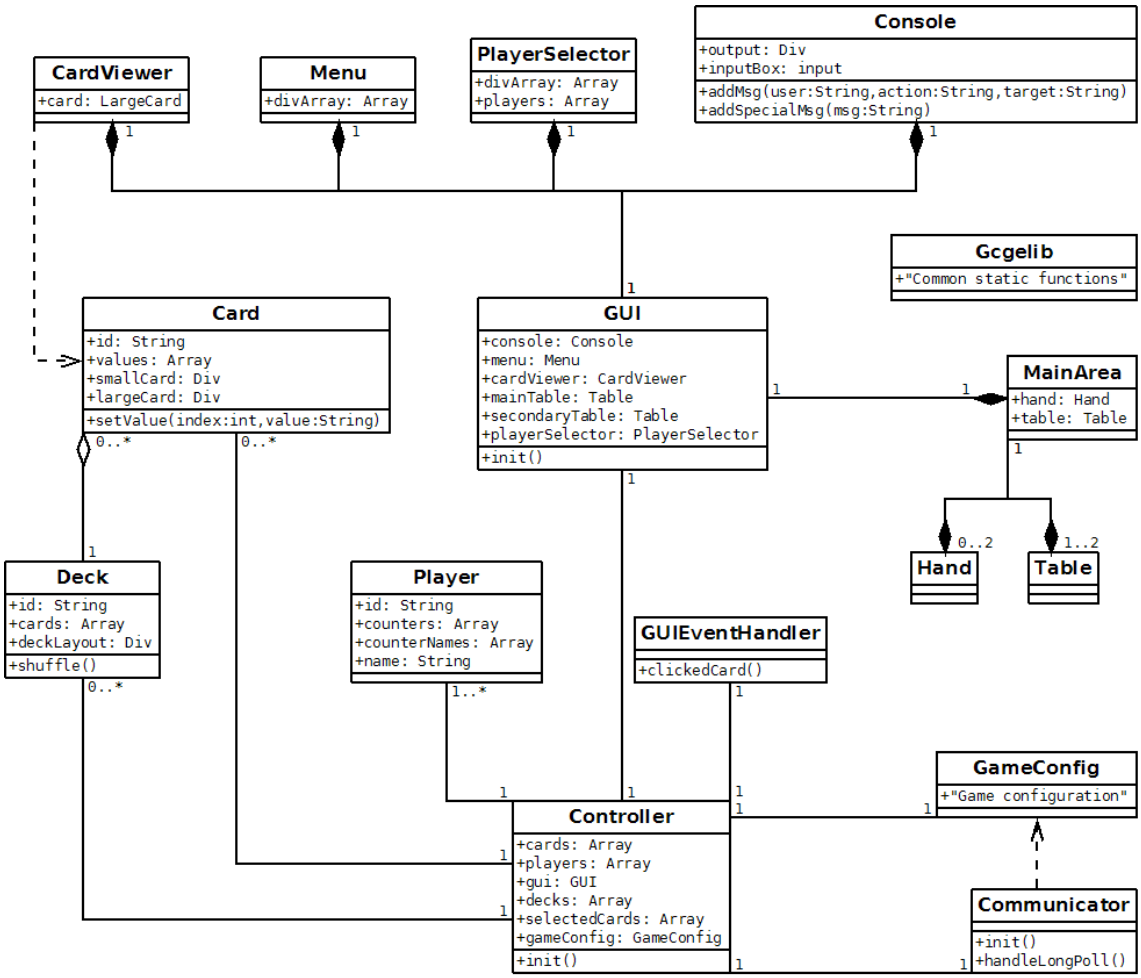



*Figure 4.3.* Class Diagram for the client.

client can get it by doing a read operation themselves. The second part is that the lock should only restrict actions that require the client to know the deck state e.g. changing the order of the cards or drawing a specific card. Actions outside of this restriction but that still change the deck state, for example drawing the topmost card, should make the lock owner less. To sum up, players will always be able to draw the topmost card from a deck, move the deck or flip it over. Players looking at the contents of a deck will only be able to do the restricted actions if the deck state hasn't been changed since he asked for its contents. The drawbacks of this lock pattern is possible starvation of a client wanting to do any of the restricted actions, as most card games are turn based this should not be a problem. A very important advantage however is that this type of lock will never end up in a deadlock state, because it can be unlocked by anyone at any time.

The read-write problem could be solved by keeping all clients that have asked for the deck state up to date, editing a deck can then be done concurrently by multiple clients. This solution was not chosen for a couple of reasons. The server would in this case need to save a state for every deck in order to keep track of what clients are currently looking

at the deck's contents. Each time a card is drawn this state would need to be checked for who to update the deck state for, doing this and sending updates would increase bandwidth, memory and processing power. There would also be issues with how to identify for example a specific card a player wants to draw. If clients were to only specify an index of a card in a deck, that index could be different if another client removes a card before the first one. The state could change before the command is received by the server. Using unique card Id's would solve this though. This solution might create vulnerabilities where data on which card was removed from a deck is sent to a client that should not receive it, hence locks would still be needed.

### 4.3.1. Interaction

Figure 4.4 shows a sketch of the user interface that was aimed for. The two biggest rectangles represent a players area, consisting of a table and a hand. The topmost one should show the cards of the currently selected opponent, while the bottom one will show the player's cards. These two areas form the playing field, and this needs to be configurable to suit the players needs. Examples of configurations are when players should share one common table or when players don't need "hands". Above this area is the player selector where players can select a player, the point of this is to be able to look at a specific opponents table. At the very top there should be a menu system for doing various things, such as leaving the game, creating a new one or inviting a friend to join the game session. The two rectangles on the right represent a chat window and and an area for showing all relevant data when hovering over objects like cards, decks and players. The chat area will also be used by the system to report things such as errors or actions taken by players.

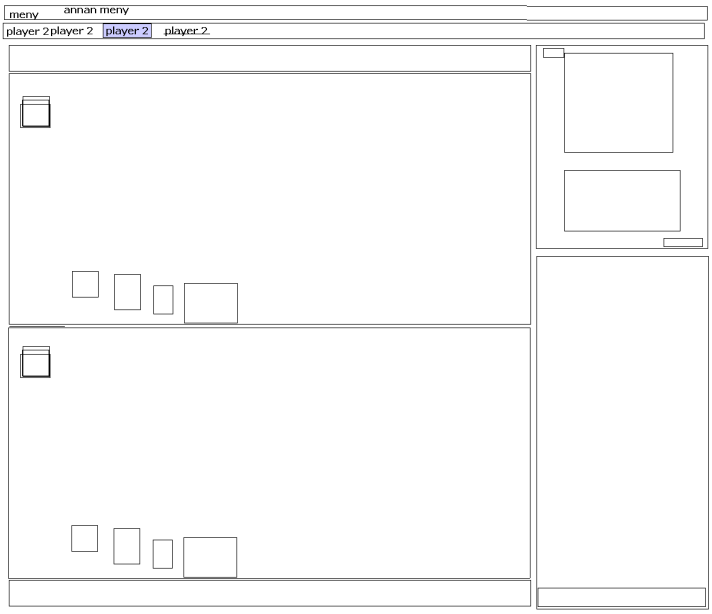


*Figure 4.4.* Early sketch of the interface.

This design was influenced by the interfaces for regular card game programs like Magic Workstation [6] and Magic the Gathering Online [21], this made it possible to establish a base that could be moulded to fit the requirements. Looking at other card games one can find functionality that this interface needs to support. Additions to the interface was needed, for example being able to play with multiple players, meaning more than two

players at a time. In addition to this, the GUI would also need to be quite general in order to allow for any type of card game to be played.

In order for players to be able to play with many cards at once the cards should be quite small, if the player hovers over a card with the mouse the card data needs to be shown. Moving cards around the player areas is done with the mouse, interactions with the card itself is done by right clicking on the card to bring forth a menu with options to for example remove, edit or rotate it. Adding cards to a table and similar actions can be done by right clicking directly on a player area. This implementation is meant to be as intuitive as possible. Adding a card for example, will place the card at the location of the right click. This would not have been possible if a menu at the top of the client had been used since it would not be possible to find a good location to place the card.

When an action has been taken by another player, for example when a card is drawn from a deck, players should be notified by a message in the chat window. The messages should be quite specific so that there is little or no room for misinterpretation. This is needed because some actions will be very hard for the players to spot by looking at the interface. A good example of this is if an opponent draws a card from a deck to his hand, and the player is not currently looking at this specific opponent, the action will go by unnoticed. Adding animations for this type of action would not solve this problem either, an action message solution was considered to solve all of these problems.

### 4.3.2. Communication

The client and server communication will be http based, with the use of Ajax long poll technique. The data sent will be in the JSON format, because it's quite easy to create and use, but it will also simplify debugging. Creating an object in JavaScript can look like this:

```
var json = {"command": "addCard", "card":{"ID":14, "value":"King", "type":"spades"}};
```

In this example the JSON has an object within itself, types like lists, boolean amongst others are also supported. When sending a command the object will be converted to a string, this can be done by simply using a JavaScripts built in function. This string will have the syntax used in the above example. When the receiver receives a command he will create an object from this string. JavaScript has a built in function that does this for us, in addition there are libraries that offer similar functionality as well.

The data sent will be interpreted as a command, where a string in the data uniquely identified the command given. The commands sent from and to the server may look quite similar, but the information required is quite different from each other. For example, the server will always require the command to contain a session-id and a personal id of the player in order to validate and verify the command. The personal id is something shared only between the server and one specific player, so unless it is shared no one should be able to spoof commands.

The http overhead on every command sent is something that can be reduced by sending multiple commands at once. This method is only applicable in certain situations. One

good example is when a client wishes to join an existing existing game, the server should then send a collection of commands in a single http response that represents the game's state. A collection of commands should also be sent when the client is not ready to receive data, meaning the time in between long polls e.g. when there is no http request for the server to respond to. This can happen when commands are being sent very frequently, in addition it will be more likely to occur if a client has a slow connection or host computer and when the server is occupied with many game sessions and players.

Figure 4.5 depicts the communication flow between clients and server, though this is only an illustration hence a lot of the actual data sent in a command is missing. In this instance the server has two different game states, the communication is represented by green and blue arrows respectively. The game states are separate from each other, so no communication will ever be sent in between different games. The figure illustrates two types of commands being sent from clients and how the server then sends similar commands to all clients that are a part of the concerned game state.

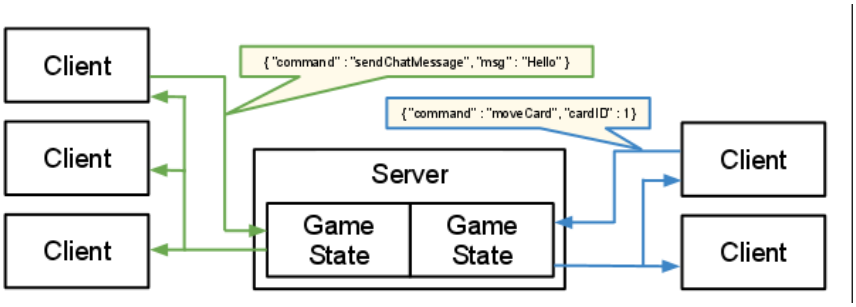



*Figure 4.5.* Illustration of the data flow within the system.

### 4.3.3. Security

Vulnerabilities and security issues is nothing new when it comes to web applications, so steps were taken in order to reduce risks concerning this. Although not entirely related to security, by reducing the amount of vulnerabilities of the system the possibility of cheating is expected to be reduced. One prominent vulnerability of the system are the different types of DOS attacks, a quite common vulnerability with web services which can be hard to protect against. Because of a vulnerability with JSON in combination with JavaScript, when the server creates objects from the JSON data produced by the clients, it could be possible to inject foreign code to be executed by the server. This very issue will be mirrored in the clients, because data in the commands sent from the server might have been produced by other clients. In addition to this there are other types of possible code injection vulnerabilities such as SQL injection, html injection and the special case of cross-site scripting.

In a way the client is inherently insecure, this is because the client code is sent to a host and the user can then modify the client in any way he likes, this is particularly easy

since JavaScript code is not compiled before it is sent. Because of this the data sent to the server can never be trusted, so this is where the system is the most vulnerable to code injection. The SQL database would be vulnerable to SQL injection but through the use of filtering methods, before the query is sent to the DBMS from the server, this can be prevented. There are other solutions as well but this one is quite suitable for this system as this can be done on the server with which there should not be any possibility to tamper with.

When a command is received by either client or server, an object will be created from the JSON data. In order to create this object JavaScript executes the JSON data, this makes for a vulnerability at both the server and client if the data is not validated as being in the JSON format. This is a problem with the built in JavaScript function "eval" and because this is a known vulnerability there are ways to avoid it. Measures can be taken in the form of for example using a library. In addition there is a newly implemented function that is a part of the ECMAScript language which JavScript is a dialect of, this function is designed to be a more secure version of "eval" when it comes to the JSON format.

Regarding DOS attacks, the server's location determines the degree of protection versus flooding attacks. These types of attacks should be detected and handled by the surrounding environment, by for example firewalls and routers. These types of DOS attacks can focus on exhausting the bandwidth of the victim, or by using up all available connections, the latter is known as a SYN flood. Possible DOS attacks that are more aimed against our system concerns exhaustion of the servers resources such as the memory or processing power. Preventing these types of attacks is possible by keeping a record over the users of the system, only allowing a player to join and create a certain amount of games and blocking otherwise. Players creating to many games could be temporarily banned. The automatic creation of a user account, and possibly a new game, likely used in a DOS attack could be limited by using a challenge-response test as proof that a user is human. The server will also need to be robust, it needs to be able to handle receiving invalid data and malformed commands without crashing. Concerning the commands that result in forwarding of data the server might need to verify the forwarded data, even though it is not intended to be used by the server in any way. As mentioned, code injection is possible on the server but the clients needs to be protected as well. The server is not vulnerable to html injection, hence it needs no protection against it. The client however does, and since processing power on the server is more valuable, protection can be delegated to the client.

In order to protect the client one can filter data, such as the text being printed in the chat, on the client side. This text can for example include html, ActiveX objects or scripts, all of which needs to be protected against. The filtering needs to be done as it originates not from the server but from other clients. This type of attack is known as cross-site scripting (XSS). The filtering mechanic can be removed by modifying the client, though modifying other clients through XSS should not be possible. In conclusion, the clients should filter incoming data not filtered by the server, and the server should only filter against vulnerabilities in the server.

Cheating in this system wont be prevented per se. However, no sensitive action should go by unnoticed by the other players, such as looking at the cards in a Deck. This means that all sensitive actions will need to involve the server, and the detection of an actual cheater will fall upon the participants of the game. Keeping track of the Id's of cards should be made impossible by giving generating a new Id when drawing a card from a deck. This way, no one but the player who takes the card to his hand will know which card it is.

The system will still suffer the vulnerabilities that comes with network applications, like for example man in the middle attacks. Guaranteeing data confidentiality and integrity could be provided with the use of https. Adding this would require a reasonable amount of work, this high level of security however was deemed unnecessary for this system.

# 5. Results

## 5.1. System description

Looking back at what has been achieved during the project we feel that the main goals have mostly been met. In it's current state the system have the most important features and can be used to play any card game. The system supports all major interaction such as moving cards around the virtual game table, from one table to another or to and from the players hands. The same is true for decks which also have the possibility of having cards placed inside them or taken out. Both cards and decks can also be created, removed or edited dynamically while playing, for cards editing entails changing the card data. For decks edit refers to viewing the content of a deck and either rearranging the card order or picking out specific cards from the deck. Decks also have the possibility of being used in a flipped state which means the top card will be visible, the opposite is possible for cards meaning it is possible to place them on the table facing down. There is also the player selector which can be used to switch which of the

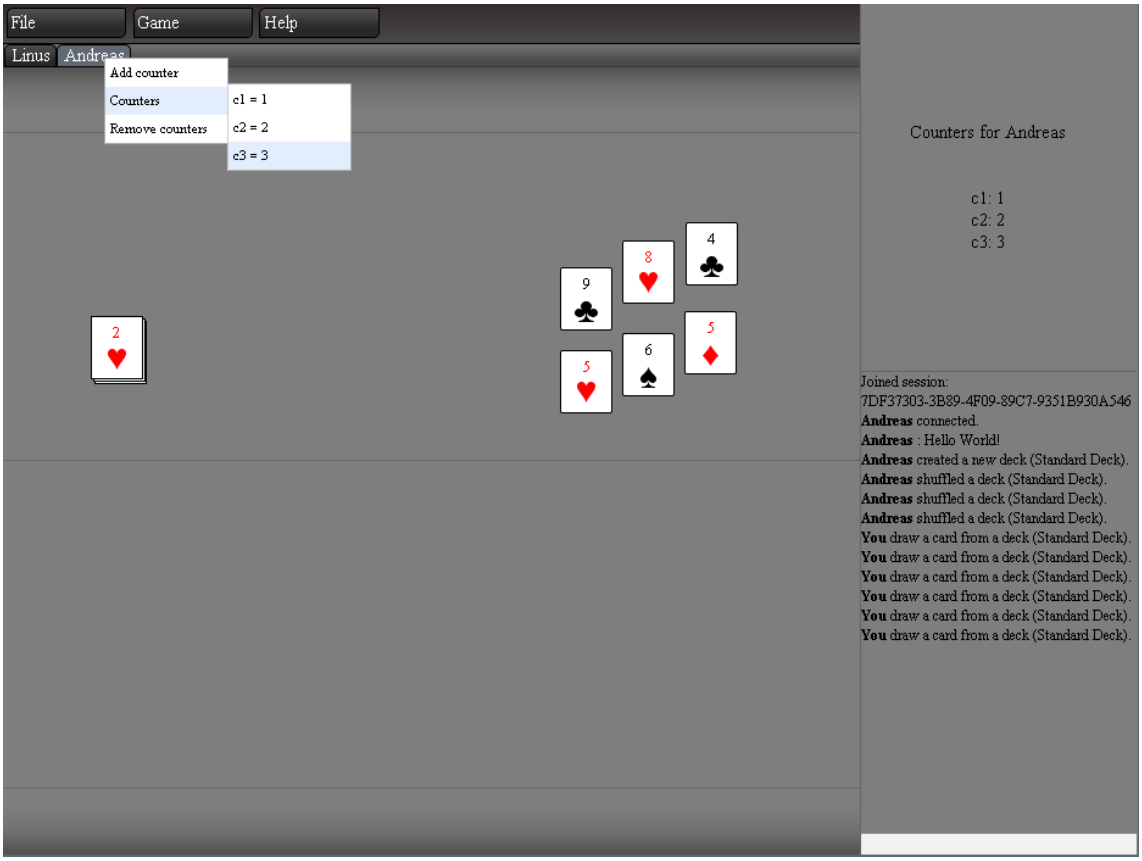



*Figure 5.1.* Screenshot of the client interface.

opponents tables are visible when there are more than one. The player selector can also be used to keep score, it can be seen at the top of the picture in figure 5.1 the screenshot also illustrates several of the previously mentioned features. It also shows the graphical design of the client which could use some improvements as it has not been one of the highest priorities during this project.

Many of these features are accessed with the right click menu which lets you apply an action on the selected object. For cards this action can also be done with multiple cards at once if more than one card has been selected either by area select or individually selecting each card with the help of a modifier key. Through the right click menu it is also possible to shuffle a deck or to rotate a card which can be useful in some games to indicate that the card has been used. It is also possible to load a complete preconstructed deck of cards and using a modifier key it is possible to move all cards in one deck into another by dragging on to the other. The panel on the lower right hand side, previously refereed to as a console, is able to present information to the user and report all actions taken by others players. The console is also used to communicate with the other players though text messages. Finally the system also have the ability to be launched with different configurations which can change the interface or some minor functions such as what should happen when a card is clicked twice. The game configuration can change the table setup to for example a single shared table and remove the hand areas. The game configuration also contains the layout for how a card should look so that it can be changed when the user wants to play a card game that does not use a standard deck of cards.

There are also a few features we would have liked to have implemented for which the time constraints were the largest reason that they are not currently in the system. First there is verification of the data received by the server inside a command this is not an issue during normal usage of the system. It can however become a problem if the user does something unsuspected including malicious activities such as modifying the client. There is currently some verification being done on the data received by the server but to be certain that no problems will occur there should be some rigorous testing done to find this type of problems. During the development the plan was to construct a separate component on the server that checked all the incoming commands before passing them along to be processed. Another feature missing on the server is a function for garbage collection. More specifically this would be a function that is supposed to make sure that no old game state objects gets left behind after a game session has ended. Without a function to handle this the system might eventually end up using all available memory and this will then cause a crash.

As it stands now the system is prepared for loading decks of cards from a database but it was discovered that the implementations existing for connecting a node.js program to a MySQL database have become outdated. This is likely because as has already been mentioned Node.js is still under development and since we are using the latest version some syntax has changed and some methods has been removed. So for now loading decks is instead handled in a much simpler way, decks of cards are stored in a separate JavaScript file as strings containing JSON formatted objects. This solution works well

for now but when adding a large amount of decks there might be some problems with performance and this solution means it will not be as easy to add new decks. Therefor this should be changed to MySQL when the libraries for it gets updated. There is also a few minor convenience features missing such as the ability to rename decks or to be able to draw several cards from a deck at once. Furthermore Several tests have been done on the system in it's current state to find bugs and several were fixed during this process. Testing has also been done continuously while developing the system to make sure added functionality works as it should and that it does not interfere with other parts of the system which could lead to new bugs being introduced. However some more testing should be done to try to further limit the number of bugs in the system. A few tests were also done to see if the system would work in another browser than Firefox which was used during development. From these test we saw that Internet Explorer 8 did not handle the client very well, it was possible to get some parts of the GUI working but there were a lot of problems. Google's Chrome on the other hand seemed to almost work completely and it might be enough with only a few changes to make the client work on both browsers.

### 5.2. Project evaluation

The project ended up being a great learning experience for us, as we got to design the system our selves and learn a new programming language. Even though the project was successful, looking back there were a couple of things that could have been done differently based on the new knowledge we now have. One such thing was in the design phase of the system, the decision to keep the game state on the server was taken later than we liked. The decision to use node.js over APE as well as using an SQL server together with node.js could have been tested more thoroughly as the sql library for node was outdated. These things increased the development time quite a bit and this could most likely have been avoided with a better and more thorough design as well as more research.

Some of the more successful choices we made was the use of programming language and libraries. Adapting to and learning JavaScript was to some extent quite painless though there was still a learning curve, our programming skills in JavaScript increased until the very end. Though no stress test of the server has been conducted we are quite pleased with the performance of node.js so far. We believe that JQuery helped quite a bit with reducing the development time of the GUI, more specifically with events and animations.

The development process suited the project well and we managed to closely follow the time plan. The three main iterations we had set out were kept to. However, smaller iterations in conjunction to what we had planned became a part of of the development. These smaller iterations could be as short as a day and as long as a week. These iterations consisted of first deciding on what should be implemented, then developing the set requirements and the last phase of testing and debugging. This made sense as it made the development process very agile. We were not however able to implement everything we wished, but that was expected and the requirements were prioritized accordingly so that the most important ones were done first. Conducting the

documentation of code that was a part of the development process was at some points lacking, it is however quite hard to determine what the consequences of this was. The lack of documentation was most prominent in the later stages of the development phase but made it possible to spend more time on implementing functionality. The trade-off is that some code could be slightly harder to understand, though in the end we think that we reached a good balance.

### 5.3. Future work

If one would want to continue the work of this project there are several areas where we think this effort could be directed. These mainly cover the areas of performance, usability and maintainability and the consists of ideas which were only to be considered if there were enough time or new ideas discovered during development. Firstly there is the additional systems that you want to have supporting the game system. Most importantly this it a system to manage users this entails tasks such as handling user accounts and connecting users with each other so that they can start a game session. This could for example be done with a system for listing all available game sessions which would require adding some support for this in our system or it could be done by allowing users to invite others to join their session. Something that is not as important but would improve usability is an additional web application for constructing and maybe editing decks or cards. A web application like that would only need to have access to the database and our system is already prepared for loading multiple decks. Similarly another web application could be developed for graphically designing cards our system currently have support for switching the layout but not for users to add their own. To save some resources the possibility of eliminating the standard web server should be investigated since the Node.js server could itself serve this purpose, this would also make server configuration simpler. This is because of the restrictions in Ajax that prevents requests being sent to a different domain or port which means the web server is forced to serve as a proxy.

Before publicly using the system it should be more rigorously tested for security e.g. it should be tested for injection attacks and it's ability to handle malformed data. This is something that would be very important if you were to expose it to the Internet since that could bring a large user base which might contain hackers. A feature that is not as important but was considered was to use sound as an indication for when actions are taken, like for example moving a card. This was however not given a lot of priority but we did look at it to determine the difficulty of such an implementation. From this we gathered that it is currently cumbersome to implement using only JavaScript. Problems can arise such as support for formats across different web browsers or you might need to install additional software. In the future this might become easier if more browsers start supporting the same audio formats for HTML5's audio tag. Another feature that was considered to complement the use of sound was to use animations of visual cues to also indicate what actions are taken. This could for example be animation of card movements initiated by other players or adding a border around the last few objects that there was some interaction with.

The code itself has some room for improvement since this project was developed as a

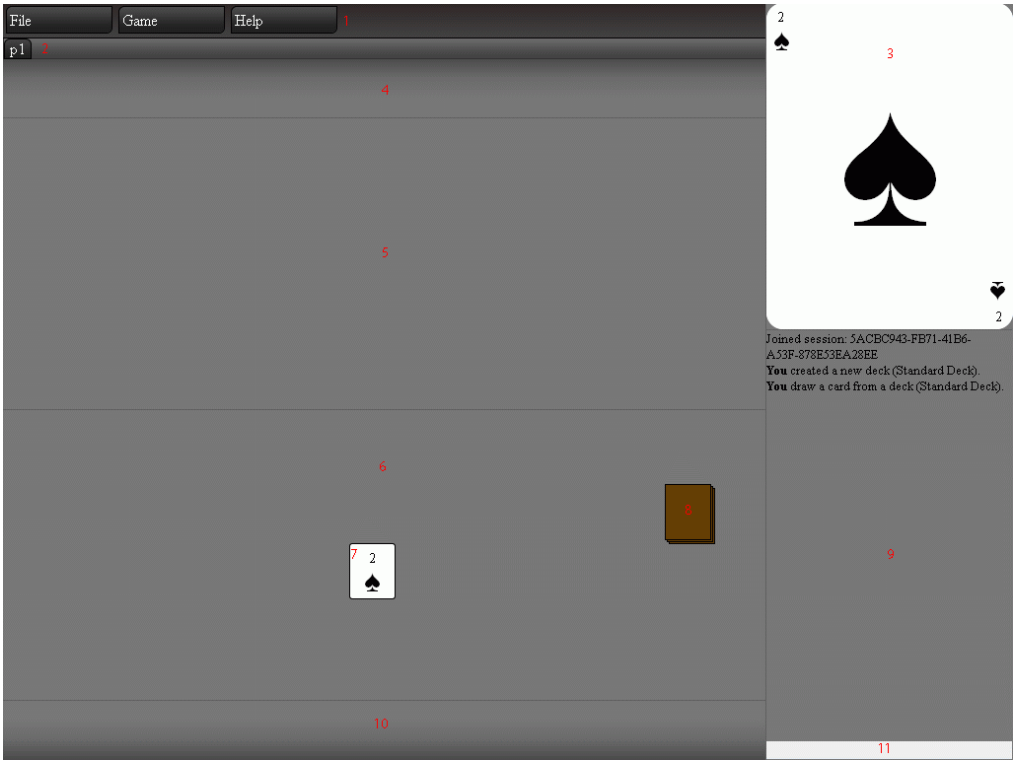
learning experience and a lot of knowledge was acquired during the work, some parts of the code could benefit from some refactoring. For a large scale deployment of the system this might be needed to improve performance. How much of an issue this is would have to be determined trough stress testing the system to see where the performance should be improved. At the same time one could look into web browser compability to see if some smaller issues could be fixed. In regards to optimization there is also work that can be done to reduce the bandwidth usage. As it stands now the commands that are being sent to the server are in a format that is easy to read for humans but they could be shortened a lot to be only one of two characters long. This could grant a rather large performance boost when the system starts being used by a lot of users at the same time. To reduce overhead further it could in the future be a good idea to switch the communication technique to web sockets which is part of the new HTML5 standard. Reverse Ajax which is currently being used is not used exactly as it was intended from the start web sockets on the other hand is created with this purpose in mind. It is also possible to add compression to the HTTP traffic which the web browser will then decompress before it is received by the client. However with compression there might be an increased delay because of the time it takes to both compress and decompress the data. It will have to be investigated if the bandwidth saved is worth the extra delay and extra computing power needed. It would also be possible to scale the system to multiple servers using a system for load balancing which would split up the different game sessions so. This would be quite easy to implement since all commands carries a game session id number and the different sessions do not need to communicate. There are other parts of our system that can be improved as well but these were the areas we felt would be best suited to focus on next.

## References


[1] Miniwatts Marketing Group, "World Internet Usage And Population Statistics," *Feb. 12, 2010.* [Online]. Available: http://www.internetworldstats.com/stats.htm. [Accessed: Mar. 23, 2010].
[2] Cable News Network, "Fast Internet access becomes a legal right in Finland," *Oct. 15, 2009.* [Online]. Available: http://edition.cnn.com/2009/TECH/10/15/finland.internet.rights/index.html. [Accessed: Mar. 23, 2010].
[3] I. Hickson, " HTML5 at Last Call (at the WHATWG)," *Oct. 27, 2009.* [Online]. Available: http://lists.whatwg.org/pipermail/whatwg-whatwg.org/2009-October/023849.html. [Accessed: Mar. 23, 2010].
[4] W3C, "Web Storage," May. 12 2010. [Online]. Available: http://dev.w3.org/html5/webstorage/ [Accessed: May. 12, 2010].
[5] Google, "Chromium OS". [Online]. Available: http://www.chromium.org/. [Accessed: May 12, 2010].
[6] Magi-Soft Development, "Magic Workstation™," [Online]. Available: http://www.magicworkstation.com/ [Accessed: May. 21, 2010].
[7] T. Ronkainen, "Generic Collectible Card Game," [Online]. Available: http://gccg.sourceforge.net/ [Accessed: May. 21, 2010].
[8] K. Beck et al., "Manifesto for Agile Software Development", 2001 [Online].

Available: http://agilemanifesto.org/ [Accessed: Jun. 3, 2010].
[9] Google, "V8 JavaScript Engine," [Online]. Available:
http://code.google.com/apis/v8/ [Accessed: Jun. 3, 2010].
[10] The jQuery Project, "JQuery," [Online]. Available: http://www.jquery.com
[Accessed: Jun. 3, 2010].
[11] W3C, "The WebSocket API," May. 12 2010. [Online]. Available:
http://dev.w3.org/html5/websockets/ [Accessed: May. 21, 2010].
[12] R. Dahl, "Node.js," [Online]. Available: http://nodejs.org [Accessed: May. 21,
2010].
[13] APE Enterprise Solutions, "APE Ajax Push Engine," [Online]. Available:
http://nodejs.org [Accessed: May. 21, 2010].
[14] D. Baranovskiy, "RaphaëlJavaScript Library" [Online]. Available:
http://raphaeljs.com/ [Accessed: May. 21, 2010].
[15] W3C, "SVG Print," Dec. 21 2007. [Online]. Available:
http://www.w3.org/TR/SVGPrint/ [Accessed: May. 21, 2010].
[16] J. Resig, "Processing.js," [Online]. Available: http://processingjs.org/ [Accessed:
May. 21, 2010].
[17] WHATWG, "The canvas element — HTML5," May. 12 2010. [Online]. Available:
http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-
element.html [Accessed: May. 21, 2010].
[18] B. Smus, "Performance of Canvas versus SVG," Jan. 19 2009. [Online]. Available:
http://www.borismus.com/canvas-vs-svg-performance/ [Accessed: May. 21, 2010].
[19] BuiltWith, "JQuery Usage Statistics," May. 18 2010. [Online]. Available:
http://trends.builtwith.com/javascript/JQuery [Accessed: May. 21, 2010].
[20] Net Applications, "Browser Market Share," Apr. 2010. [Online]. Available:
http://marketshare.hitslink.com/report.aspx?qprid=0 [Accessed: May. 24, 2010].
[21] Wizards of the Coast, "Magic Online," [Online]. Available:
http://www.wizards.com/magiconline [Accessed: May. 21, 2010].

# APPENDIX I - User Manual – Quick start guide




1. Menu
2. Player Selector
3. Viewer
4. Opponent's Hand
5. Opponent's Table
6. Your Table
7. A Card
8. A Deck
9. Console
10. Your Hand
11. Input Area

**Detailed descriptions**

1. The menu is used to access general actions such as “Quit game”.
2. The player selector is used to switch which opponent's hand and table you are currently viewing or to modify the players counters through the right click menu.
3. The viewer is used to display a larger view of cards, display information about a deck or a player.
4. This is the opponent's hand cards placed here will only be visible to the currently selected (see 3) player.
5. This is the opponent's table area cards placed here will not be visible if you select another opponent (see 3).
6. This is your table area cards placed here will always be visible to you at all times.
7. This is a card which can be moved around.
8. This is a deck which can be moved around and can contain cards.
9. The console is used to display chat messages, actions taken by players and system messages.
10. This is your hand area cards placed here will only be visible to you.
11. This is the input area which is used to send chat messages.

**Moving a card**
Cards are moved by dragging them using your mouse while holding down your left mouse button.
If multiple cards are selected all of them will be moved at once.
A card can be dropped on a deck to place it inside a deck and dragging a deck will let you draw the top card from the deck. If this is done while holding down **Shift** the card will be facing down after you have dropped it on a table or a hand. If this is done while holding down **CTRL**+**Shift** all cards can be moved from one deck to another.

**Selecting cards**
Cards can be selected one by one by holding down **CTRL** while using your left mouse button to click each card.
You can select many cards at once by left clicking anywhere on the hands or tables areas and dragging to form a square which will select everything within it.

**Moving a deck**
A deck can be moved it you hold down **CTRL** while dragging it using your mouse and left mouse button.

**Performing an action on an object**
Specific actions can be performed on decks and cards by right clicking them to bring up a menu where the action can be selected. For example shuffle a deck, rotate/flip/edit a card or view a decks content.

**Deck Viewer**
The content of a deck can be viewed by right clicking the deck and choosing how many cards you want to look at. Providing a negative number will allow you to look at cards starting from the bottom of the deck instead of the top. Not providing a number of cards that you whant to view will cause the system to display all cards in the deck.
Double clicking a card in the deck viewer will place it next to the deck on the table or hand area. Holding down **Shift** while doing this means it will be facing down.
Before pressing done you can choose weather you want the deck to be shuffled after you have viewed it with a check box.

**Edit card**
To edit a card right click the card and choose edit from the menu.
Each of the cards values will have a text field where you can input the new value.
The color for the card can be given using common names such as red, green, lightgrey. For a more specific color an rgb value can also be provided on the form “rgb(0,0,0)”, rgb stands for red green blue and must be withing the interval 0-255.

**Send a chat message**
To send a chat message simple write the message in the input area and push the **Enter** key.

**Switch currently viewed opponent**
The opponent you are currently viewing can be switched with the player selector where players will be automatically added when they join your game session. This is done with a left click using your mouse.

**Add/modify/remove a player counter**
A counter can be used to keep score or for something else you which to store for a specific player (all players have access to the same counters). This can be done by right clicking a player on the player selector and then choosing the option on the right click menu.

# APPENDIX II – System requirements

Following is a list of reqirements that were formulated at the start of the project, the ones colored green are implemented and the ones in red were not implemented.

**Gameplay**

- Select card/cards
  - Shift select
  - Ctrl select
  - Area select
- Move card from any possible card location to another
  - Put card into playarea from hand
  - Put card into playarea from deck
  - Draw card into hand from target deck
  - Draw x cards into hand from target deck
  - Discard card into discard pile
  - Discard random card into discard pile
  - Drag and drop selected card(s)
- Rotate or flip over a card in play
  - Rotate card
  - Flip card
- Pile cards/create deck?
  - Create new deck
- View content of a deck
  - Open popup with list of cards
  - Option to shuffle on closing window
  - Drag card to table
- Always see card name
- Display detailed card information
  - Display bigger card with all information on mouse over
- Increase/decrease a counter
  - Modify counter with value X
  - Add counter?
  - Remove counter?
  - Name the counter
- See all actions taken in a console
  - Print action function
- Change indicator for game phase
- Save current game session
- Add an indicator arrow
  - ctrl+alt + click+click = add arrow between clicks?
- Generate a random number in a specefied intervall
  - Flip coin
  - Roll dice
- Sort cards
  - Cascade

- Tile
- Add note to a card
  - Add token to card
  - Add note to a card
- Shuffle deck/hand
  - Shuffle target deck
  - Shuffle target hand
- Add token-card dynamically
  - Token creator popup
  - Temporary save last created token
- Group cards
  - (Interaction between cards)
  - Attach a card to another
- Modify card values
- Copy card

**Player interaction**

- Chat
  - Chat with his players in a game
  - Lobby chatroom (Chat with free players when not in an active game)
  - Send private message to another user
  - Join different channel for chatting than the default lobby channel
- Invite others to a game you created
- Join an active game session
- View list of active games
- Friendslist
  - Add player to list of friends
  - Remove player from list
- Filter active game sessions

**Manage**

- Game configuration
  - Create config
  - Save config
  - Load config
- Start a new game session
  - popup with option to choose game configuration and password
- Password protect a game session