



UNIVERSITETET I AGDER

Souls

by

*Per-Arne Andersen
and
Paul Richard Lilleland*

Supervisor: Halvard Øysæd

Project report in DAT-219, spring of 2014

Faculty of Engineering and Science
University of Agder

Grimstad, 18.05.2014

Status: **FINISHED**

Keywords: MMOCCG, pixi.js, .NET, WebSocket, rendering, WebGL



UNIVERSITETET I AGDER



An online collectible card game inspired by Hearthstone



ABSTRACT

In 1993, a man named Richard Garfield made a game later published by Wizard of the Coast. The game was called Magic: The Gathering and is the most popular card game in existence, played by approximately twelve million people as of 2011.

The game is played by two players where each player has their own deck of cards. The game features two wizards/summoners battling with their minions and spells.

Other similar games have been made where you battle as characters from Pokémon, Lord of the Rings and Star Wars. Some of these games have been digitized but remains most popular in their physical form.

Lately Blizzard released a free to play game called Hearthstone, probably the most popular digitized version of the card games at this time.



Preface

«Souls» is a student project by Per-Arne Andersen and Paul R. Lilleland in the course DAT-202 at University of Agder. The project has no specific work schedule and is developed from students home and the university campus. The project use GIT as versioning control and JIRA for work logging. These services are provided by the university at tools.uia.no

The game is found at: <http://souls.persoft.no>

Grimstad

May 2014

Paul R. Lilleland and Per-Arne Andersen

Contents

1. Introduction.....	7
1.1 Background.....	7
1.2 Problem statement.....	7
1.3 Priority Overview.....	8
1.4 Priority Specifications.....	9
1.4.1 Priority A.....	9
1.4.2 Priority B.....	10
1.4.3 Priority C.....	11
1.4.4 Priority D.....	13
1.5 Expected results	13
1.6 Literature review	14
1.7 Problem Solution.....	15
1.7.1 Project roles.....	15
1.7.2 General Approach.....	15
1.7.3 Tools and software	16
1.7.4 Tools Overview	16
1.8 Project Plan.....	18
2. Game Design.....	19
2.1 Login or register:	19
2.2 Welcome Page:.....	19
2.3 Queue:	19
2.4 Game:	20
2.5 Chat:	21
2.6 Admin panel	22
2.6.1 Players	22
2.6.2 Card Editor.....	23
2.6.3 Games.....	23
2.6.4 News editor	24
2.7 Game summary	25
2.8 The players	25
2.9 Card Shop	26
3. Technical requirements.....	27
3.1 Licensing	27
3.2 Diagrams.....	28

3.2.1 Class Diagram - Server	28
3.2.1 General Game Flow	29
3.2.2 Network architecture	30
4. Implementation	32
4.1 Game Textures	32
4.2 PixiJS (Rendering Library)	32
4.3 Asset loader	33
4.4 Sounds	33
4.5 Game Engine (client side)	33
4.6 Chat client	33
4.7 Cards	34
4.8 Players	34
4.9 Welcome Screen	34
4.10 WebSocket-sharp	35
5. Validation and testing	35
5.2 Unit Testing and Stress Testing	35
6. Results	37
6.1 "A" Priorities	37
6.2 "B" Priorities	38
6.3 "C" Priorities	38
6.4 "D" Priorities	39
6.6 Summary	40
6.7 Future Work	40
7. Support	41
7.1 Browser compatibility	41
8. References	41
9. Appendices	42
9.1 Code Standards	42
9.2 Team Contract	43
9.3 Meeting Reports	44
9.4 Time Sheet	Error! Bookmark not defined.
10. External dependencies	45

1. Introduction

1.1 Background

The project is a game based on Hearthstone, a popular online card game made by Blizzard. The developers use this as their main inspiration but the rules and gameplay is picked from various card games.

The purpose of this project is for the students to learn, and to show their competence in HTML, Visual Studio, C# and object oriented C#-classes. Databases, User authentication, ASP.NET, DHTML (JavaScript), XML, Ajax and Web services.

Numerous projects were taken into consideration, but Souls seemed to be the most interesting task. The project was designed to cover the whole curriculum and it quickly became a large project to complete. Time consuming aspects such as game balancing and heavy game content is and never was to be completed because of limited development time.

1.2 Problem statement

Multiple requirements were set in place to provide a good documentation and development framework for the project. The primary frames for the project period consisted of the Gantt diagram along with the priority list.

The group gathered a list of software and libraries to be used but this may be changed later in the project if necessary.

The priority list was considered the most important requirement and piece of pre-project documentation this project had. It explains roughly how Souls looks like, its functionality and what to expect in the first upcoming release.

1.3 Priority Overview

The goals in category “A” are primary goals and do in turn contain a list of goals prioritized according to necessity. “A” being the vital parts, “B” should be included. “C” is secondary features we hope to implement and “D” is room for extension. Unless we have a lot of extra time, we do not plan to implement features in this category.

Priority “A”

Web Server with resources	A
Game Server	A
Game Client	A
Web client	A
Two player game	A

Priority “B”

Music/sound effects	B
Visual effects	B
Limited and manageable cards	B
Chat client	B

Priority “C”

Vendor and currency	C
World map	C
Game Lore	C
One player game (with AI)	C
Admin panel	C

Priority “D”

3D version	D
Custom sounds	D

1.4 Priority Specifications

1.4.1 Priority A

Web Server with resources

Souls is a project written purely in .NET with C# as its language. In order to host the services, a web server will be needed, and the easiest option is Internet Information Services (IIS). This software is bundled with most Windows editions, and is also included on the deployment machine the group uses.

Instead of running the integrated IIS service in Visual Studio while developing, the group members have installed the standalone edition. This speeds up developing because of the reduced time to restart the server whenever a change has been made. Unfortunately, IIS does not ship with Compile on Change options, so it is important to rebuild the project on a change.

Game Client also runs on this web server, but it does not require any processing power from the server itself (JavaScript runs on the client).

Having a web server is not enough when it comes to multimedia projects. Souls uses a database to keep track of/ host resources for the game, ex the card textures. In order to make cards as dynamic as possible, this has to be indexed in a database.

The database's tasks also includes storage of player information, card data and game logging. This is something, which is handled by both the web server and the game server.

Game Server

The most essential priority in the project is the **game server**. This implementation will be the heart of the game. Its task is to host, process and serve game rooms for all of the clients connected to the game. This task requires a solid design in order to work properly, and it is something the group believes is possible to achieve.

The server will be written in C# .NET with additional dependencies, including a web sockets library and extended support for JSON. An important task is to keep the server logic as simple as possible. Doing so will decrease the need for debugging greatly, if it were to crash when the project goes live.

Our game client is developed in JavaScript, and this threatens the security of our game (Client modifications). The server will authenticate users with a sha256 hash before game network traffic is activated.

Starting development of the server, the group also made an "API Console" which tested the traffic the finished game client were to use. This is something that later would be the Souls game client.

Game Client

The game client's task is giving users a great experience when playing Souls. Even though it should be technically possible to play Souls via a Console, a graphical framework will be built on top of the network traffic. This will not only increase the gameplay experience, but also greatly improve the capabilities of the game, as it will allow development of multiclick interactions.

The game client, also called the **game engine** is the *body* of the game. The game engine does not do any computations, instead this is received from the game server. The game client implementation sounds trivial, but depending of the scale of the product, it might become relatively complex. This is due to the *limited* object oriented functionality in JavaScript. The game client/engine will be covered in detail later in the report.

Two player game

The *two-player* implementation is vital for testing gameplay when balancing and developing all of the game features. This is a part of the game server, which keeps a dictionary of connected players. One important note is that the game server does not expose information to the opponent, which should be hidden, for instance. Cards that the opponent has on hand should not contain information about which card it is. This should be achievable with identification of each of the players in the running game room. JavaScript does not prevent the user from modifying its content, so this is an important step in the project.

There should only be one player logged in on an account to prevent concurrency issues.

1.4.2 Priority B

Music/sound effects

Audio is an important step in setting the environmental mood in a game. Souls is a game which aim to set contrast and good and evil through images and audio, and it's therefore important to use the appropriate sounds in the game. The game should feature some moody effects in order to increase the entertainment value.

Visual effects

Like with sound and music the game would lose much of its end-user experience quality if every movement was a static dull A to B movement. With the help of Tweening a lot of visual effects are made possible without too much work.

Limited and manageable cards

Creating a game is not only about getting it to run, but also give great user experience. Dynamics and Player choices is something that is a big part of making this possible, and Souls provides this with **manageable cards**. This means that the player can purchase the desired card given that enough **soul coins** is present in the inventory.

For this implementation to fulfill its purpose, it will need to collaborate with the **Vendor and currency** priority.

Chat client

The idea of implementing a chat client came after the project planning. In group meetings this came up multiple times, and it felt natural to implement such a function in the game. Most games today have the ability to communicate between players, and whilst it is not a vital implementation, it's important for the completeness of the game.

The chat service will be separate from the game server. Souls is expected to have a basic chat room between the players, but the chat API should be extended above this for further development of this service.

Later expansions should include the ability to invite and kick players from chat, have general chat lobbies. Have cross game communications.

This task is trivial, but it remains to see whether sub protocols are preferable on web socket communications. More details in later chapters.

1.4.3 Priority C

Vendor and currency

The currency consist of "**Soul Coins**". These are used to buy additional cards for your deck, and can be obtained by winning games. The shopkeeper of the game is Hansel, the first champion of Souls. He sells a large amount of cards to any player who has the required amount of Soul coins.

Soul Coins should be stored in the database alongside the player information.

World map

A world map in Souls would give the ability to travel to different locations to fight against players of the same rank. This implementation is not important for gameplay, but may help increasing the total game duration. It is not expected for this feature to reach the first version of the game.

Game Lore

Most of the game lore is given at the first post on the homepage, there you may read the background how and why the game exists and about the main villain and the old hero who is retired and now runs the store.

One player game (with AI)

Souls is a game, which requires at least two users to play the game. It is then important to have an alternative if no players are present in the queue. This is not a high priority for the Souls project because AI does not make the gameplay itself any better. The developers of Souls thinks that implementing a rich content base comes before AI, and it is therefore set to a lower priority.

There are numerous ways to implement an AI in Souls:

- **Integrated AI** - Runs directly on top of the server layer, Better performance wise.
- **External AI** - Runs as a Client on external server, Connects to the server and acts as a player, Requires extra computation power for the WebSocket connection
- **Hybrid AI** - Locally connects through WebSocket, Marginally reduced load from external load, Better integration with the server, BUT bigger risk of concurrency issues (i.e. Multiple AI's reading from same reference)

The Hybrid and Integrated AI is best suited when creating unit tests, which is likely to happen for this project. It is therefore preferable with one of the two mentioned.

This task is likely to be implemented if enough development time is given.

Admin panel

The admin panel was planned relatively late in the project. First plan only included *game end* statistics, but this was later converted to a user feature instead of an administration feature.

The administration plans were therefore reworked, and came out as the following pages: "Players", "Games", "Card Editor" and "News editor".

Players should display all registered users, their online status and give the administrator the ability to kick, ban or reset the player. It should also be possible to promote or demote players. All player information should also show here.

Games should display all games done in souls. There will not be any demand for options in this page as its only purpose is to show games done by the players. A delete button may be added if seen fit.

In the "**Card Editor**", administrators may create custom cards that will be for sale in the store. The administrator needs to add a picture, along with all the stats. The card can then be saved to the database and used in the Souls arena. It will be available in the Shop section of the page as well.

Administrators can also modify existing cards to balance out the game if need be.

News Editor is the last page planned for the administration panel. This panel should give the opportunity to modify, show/hide or add new news to the front page.

1.4.4 Priority D

3D version

If all secondary tasks should be completed a future possibility would be to make the game three-dimensional. This was not intended for this project period but merely a future work idea. ThreeJS is the library of choice should this be implemented.

Custom sounds

Custom and detailed sounds is not something that is planned for the first version of Souls. This is something that is likely to be implemented in a 3D version of the game.

1.5 Expected results

The project is expected to be in a state where it is fully playable and have a relatively stable gameplay. It is not expected that the product is error free. In terms of priorities, it is expected that **A** and **B** priorities is implemented. There may be design changes preventing this from happening, but major parts of the priorities should be covered. **C** priorities will be implemented if the developers see the code base stable enough for further implementations. It should be noted that **C** priorities is not required for the game to work well.

D priorities is not expected to reach the v1.0 version of the game, nor the v2.0 version. These priorities may be branched as child projects when the game is stable enough for such implementations.

It is expected that **A** and **B** priorities is implemented halfway into the project period, these will then be polished until the end while **C** goals is implemented.

1.6 Literature review

Google

Google is the most popular search engine in the world. Google is not considered a literature source, but rather the referrer to the source of the literature. Google provides us with great hits on the page Stack Overflow that is regularly used when stuck on a problem.

Stack overflow

This can be considered the most regularly literature our project used. When an issue arise in the project, Stack overflow was there to help. Stack overflow is a forum where people meet discuss their main field of interest.

GoodBoyDigital

GoodBoyDigital is the home of pixiJS, our renderer library for the client. This is mainly used for documentation and examples. PixiJS is a relatively low-level framework, and is mostly used by other libraries to build game frameworks. It is therefore a very small community to help with problems.

Microsoft

Microsoft provides the documentation for all Microsoft products used. These products include Visual Studio, IIS, .NET and ASP.NET

CreateJS

CreateJS is a company, which offers TweenJS, SoundJS and EaselJS, which is used in our game client. They also have documentation for each of the libraries in question.

1.7 Problem Solution

A lot of the work in our project will be distributed on the fly as the group only consists of two members. Generally, both members can start work on whatever they like, but there are some ground rules and restrictions in place to ensure good practices.

1.7.1 Project roles

The project consists of two components, one for the backend server, and another for the front end game client. Paul Richard Lilleland will be in charge of the backend server while Per-Arne Andersen will take care of the front-end client.

Each of the team members are responsible that their part of the code remains compilable and is of “good quality”.

Both team members can work on both parts of the project by either making their own branch on GIT or committing to the development branch.

The 50/50 split in responsibility and “ownership” is done to give both team members better motivation in developing the project.

1.7.2 General Approach

Because the project consists mostly of .NET and JavaScript, the server will be running on Windows. There are ways to make it run on Linux but it is not well supported. There will be some performance limitations to the game because of the choice to use JavaScript as the language. Even though this is the case, it is important to keep the project as close as possible to the bounds set by DAT-219.

Developing happens on each of the developer’s computer, running Visual Studio with Standalone IIS installation. The usual flow of development is that task is being discussed by the team, and then implemented by the assigned member. A review phase then start when implementation is done which its purpose is to ensure code consistency and quality.

Developing will start by creating API console and server software, which should be able to communicate with each other. When the network framework is stable enough, it is appended to the game client and the network integration starts. This is a big task, and will consist of most of the game client work.

Code is shared between the developers by GIT. The code is also available on Dropbox for quick lookup if cooperation on a problem is required.

The above is how the group will attack the problem, and it is believed that this approach will give a good result when development and project hand-in is finished.

1.7.3 Tools and software

For development, we used Visual Studio 2013, NetBeans, Notepad++, and Navicat. We have a web server for login, game client, admin-panel and statistics. A server to communicate with the client, game logic computation and authentication (Also contains chat-server). The game also has a MS SQL Server as we use LINQ. Other than that C#, JavaScript, ASP, JSON, Ajax and WebSockets are used. Libraries used are NHibernate, JSON.NET, WebSocket-sharp, Newtonsoft, PIXI.js, jQuery, Bootstrap and RequireJS.

1.7.4 Tools Overview

Client side:

ASP.NET:

"ASP.NET is a server-side Web application framework designed for Web development to produce dynamic Web pages. It was developed by Microsoft to allow programmers to build dynamic web sites, web applications and web services."

PIXI.JS

"Pixi is a 2D rendering engine built on JavaScript using WebGL. With a lot of functionality and possibilities Pixi satisfies the graphical requirements of this project."

SoundJS

"The SoundJS library manages the playback of audio on the web. It works via plugins which abstract the actual audio implementation, so playback is possible on any platform without specific knowledge of what mechanisms are necessary to play sounds."

TweenJS

"The TweenJS JavaScript library provides a simple but powerful tweening interface. It supports tweening of both numeric object properties & CSS style properties, and allows you to chain tweens and actions together to create complex sequences."

EaselJS

"The EaselJS JavaScript library provides a retained graphics mode for canvas including a full hierarchical display list, a core interaction model, and helper classes to make working with 2D graphics in Canvas much easier. EaselJS provides straight forward solutions for working with rich graphics and interactivity with HTML5 Canvas"

jQuery

"jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript."

RequireJS

"RequireJS is a JavaScript file and module loader. It is optimized for in-browser use, but it can be used in other JavaScript environments, like Rhino and Node. Using a modular script loader like RequireJS will improve the speed and quality of your code."

Bootstrap

“Bootstrap is a front-end framework from Twitter designed to kick start the front-end development of webapps and sites. Among other things, it includes base CSS and HTML for typography, icons, forms, buttons, tables, layout grids, navigation along with custom-built jQuery plug-ins and support for responsive layouts.”

Server side:**WebSocket-sharp**

WebSocket is a protocol providing two-way communications over a single TCP connection. WebSocket-sharp is a C# library developed by STA (*Awesome Chinese guy*) at github. The library makes it easy to maintain WebSocket communication and use WebSocket features.

MariaDB

“MariaDB is a community-developed fork of the MySQL relational database management system. Being a fork of a leading open source software system, it is notable for being led by its original developers and triggered by concerns over direction by an acquiring commercial company Oracle”

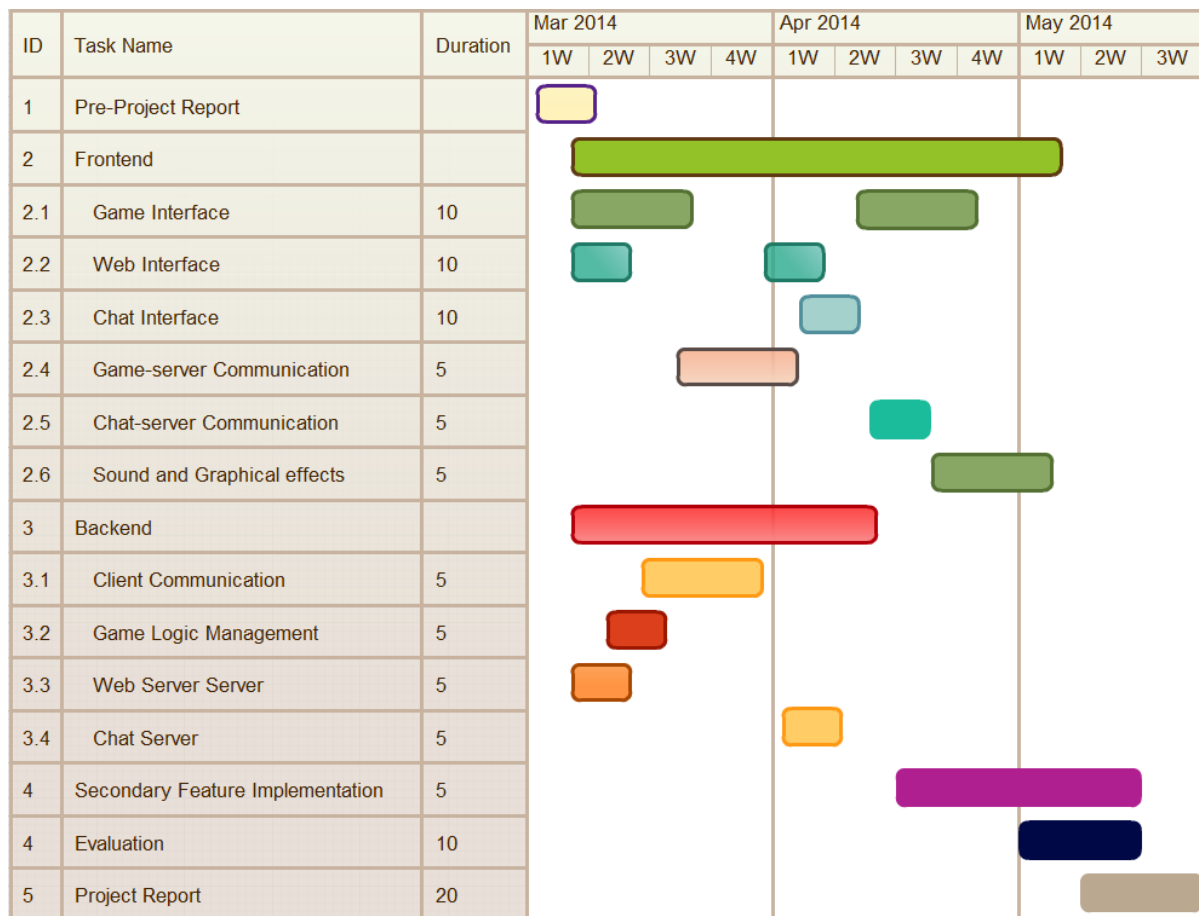
JSON.NET

“JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others.”

NHibernate

“NHibernate is a mature, open source object-relational mapper for the .NET framework. It has actively developed, fully featured and used in thousands of successful projects. “

1.8 Project Plan



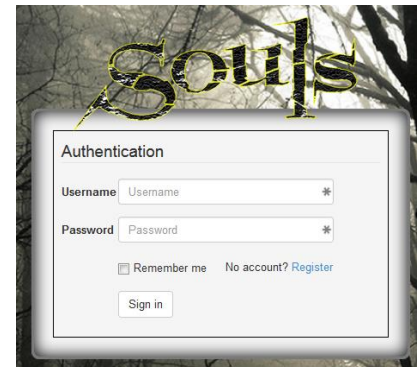
1.1 Screenshot of the gantt diagram

2. Game Design

2.1 Login or register:

The login page is displayed if a non-logged in user requests a new game or anything else that require an active session. This page consists of the typical login fields such as Username, Password, Remember and the Sign in button along with a register option for new players with some visual styling.

The register page consists of the defaults such as username, password with confirmation, email and submit. Since the player also needs a hero and cards, these options will be available from here only and cannot be changed afterwards. He or she will be able to choose from four different races and different heroes of this race. Only six cards will be available and the player have to choose five. The rest of the cards must be bought at the store.



2.1 Login screen

2.2 Welcome Page:

This will be the first a player sees when he or she enter the souls webpage or after a successful login. This page features multiple convenient and easy to read plugins. First, we have the bulletin board, which provides useful news and information to the players with easy to update functionality for the admins. Then we have the Server Status field, pings the database and game server whenever the page loads to notify the users in case one or more of the servers are down. Next is the Souls Updates plugin, which is connected to the games github repository. Any new fixes or updates will be shown directly to the user with a link to the commit. The last plugin is Recent Games. This shows the last 3 games played with a link to the post-battle screen.



2.2 Welcome page

2.3 Queue:

The queue feature works by matching players every 5 seconds. The reason for not instantaneously matching with the first available player is because of the rank. For example after 5 seconds there are 30 players in queue with 10 different ranks, the server will match those players with the player of closest rank.

Should there be a longer waiting time for any reason the player is also notified by the music when the game starts. This hopefully prevents boring waiting periods for the player as he may want do other things while waiting for a match.

2.4 Game:

This is the arena of the competing players. The rival player will always be on the top of your view and you on the bottom. Both players will have all the information shown on the screen except the opponent's handheld cards.

The gameplay is all about strategy. Each player plays half a round each, meaning that after both players have made their moves and ended their "sub-round" round number 2 begins. After each round, the mana replenishes for both players and increases with one. This means that in round one, players have one mana at their disposal and on round five, each player has five mana at their disposal. In addition to the mana replenishment for each sub-round the current player draws a card from their predefined deck.



2.3 Ingame screen

The most important part of the game is the cards. Each card costs a set amount of mana based on strength (stronger card costs more mana). Besides the mana-cost, each card has 6 stats, Name, picture, card-type, attack-damage, health and ability. The name and picture will be different for each card. The four different races are Darkness, Lightbringer, Vampiric and Feral. Attack, health and ability are different depending on the cards "strength".

At the beginning, the player has all their cards on hand, then placing them using mana as explained earlier. After the card is placed on the table, it cannot be returned or moved, only killed. Each player has their own set of cards/souls on the board able to attack their rival cards. When a minion attacks another minion both will deal and take damage according to their stats making it irrelevant, which is the attacking, or the defending player. (This enforces a sacrifice with each move you make)

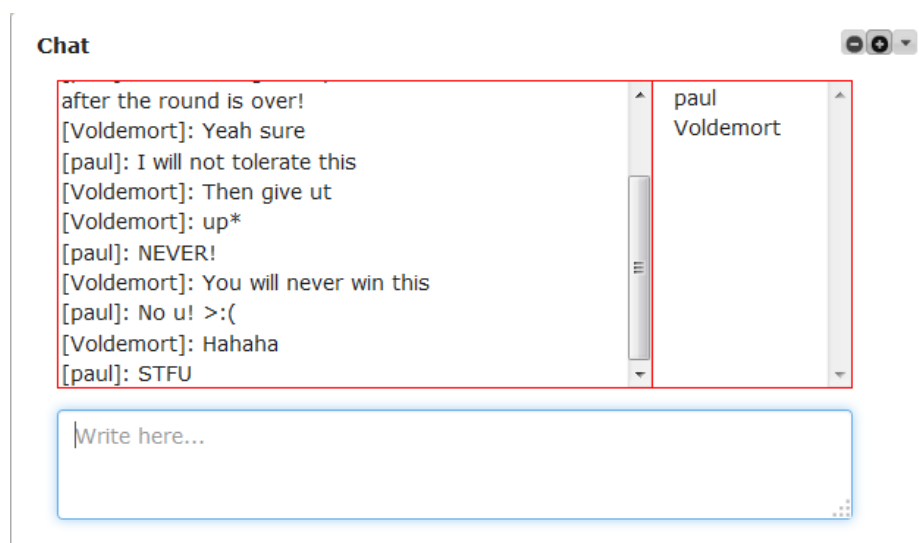
The character you chose as your hero is also a playable character on the board. When you or the opposite players health reaches zero the game is over and the surviving player wins (Should both heroes die at the same time the game ties). You have to carefully choose if you should attack the rival player or their souls. If you focus too much on the player, you may find yourself overwhelmed by enemy souls.

A feature that hopefully makes the game more interesting for the players is abilities. If a card have an ability, it may use this instead of attacking. As for now, the abilities are Heal and Sacrifice. These features are activated by holding the right mouse button and selecting a friendly or enemy player. The abilities targets may be restricted to only cards, friends, enemies or a combination of these. Like the attacks, the abilities are processed on the server and every ability requires its own function making this a time consuming implementation. This is why there is only two abilities at the time being.

2.5 Chat:

The chat client is built on the use of rooms with players being the members. For example, a general chat for players of rank 10 or a general chat for every player online. The players also have the ability to make their own rooms. The leader will be the player who made the room. This grants him the authority to invite or kick other players from it. (If the leader leaves, the leadership will pass down chronologically)

When two players are battling in the arena, a chat is automatically generated with no leader or other privileges. This can only be used by the competing players and is disposed of after the game has ended.



2.4 Ingame chat

2.6 Admin panel

2.6.1 Players

This page shows all the souls players along with their soul coins, rank, join date and online status. The status of the player is updated every 5 seconds where the webserver sends a request to the game server for all online players and updates the status of the players in this list.

Should the admin have any reason to kick, ban or reset a player this may be done from this screen as well.

Players						
Add Sample Users		Remove Sample Users				
id	Name	Money	Rank	Created	Permission Level	
1	BOT	1	1	26.03.2014 21:09:21	User Please Choose ▼	Offline Kick Ban Reset
23	per	0	1	08.05.2014 15:51:10	Admin Please Choose ▼	Offline Kick Ban Reset
24	paul	0	1	08.05.2014 15:51:51	Admin Please Choose ▼	Offline Kick Ban Reset
25	Harry	0	1	08.05.2014 16:17:03	Guest Please Choose ▼	Offline Kick Ban Reset
26	Slur	0	1	08.05.2014 16:17:24	Guest Please Choose ▼	Offline Kick Ban Reset
27	Hermine	0	1	08.05.2014 16:18:24	Guest Please Choose ▼	Offline Kick Ban Reset
28	Gygnid	0	1	08.05.2014 16:19:52	Guest Please Choose ▼	Offline Kick Ban Reset
29	Humlesnurr	0	1	08.05.2014 16:20:18	Guest Please Choose ▼	Offline Kick Ban Reset
30	Ronny	0	1	08.05.2014 16:20:49	Guest Please Choose ▼	Offline Kick Ban Reset
31	Snurp	0	1	08.05.2014 16:21:34	Guest Please Choose ▼	Offline Kick Ban Reset

2.5 Admin panel – Players page

2.6.2 Card Editor

This is the interface for generating or removing cards in Souls. On this page, the admins will have the possibility of customizing new cards for the store and should a card seem unbalanced in any way the admin will be able to edit it by selecting the card in the list below the editor, change the stats and click the update button.

Card Editor

Configuration

Card Portrait
 No file selected.

Ability Heal

Race Darkness

Cost

Health

Level

Attack

Price

Portrait URL

Name

Name	Cost	Health	Attack	Price	Level	Ability	Race
Rhino	5	25	5	20	1	Heal	Vampiric
Zebra	2	1	1	10	1	Sacrifice	Lightbringer
Per	4	4	6	30	1	Heal	Ferninus

2.6 Admin panel – Card editor

2.6.3 Games

This page features all the games played on the Souls server. If there is any suspicion of cheating or exploiting, the admins may use this to access the game-log for all games.

Games
Ronny vs BOT
Snurp vs BOT
Humlesnurr vs BOT
Gygrid vs BOT
Hermine vs BOT
Slur vs BOT
per vs BOT
paul vs BOT
Harry vs BOT

2.7 Recent games

2.6.4 News editor

At the news editor, the admins are able to add new posts or edit existing news-posts for bulletin board. The admins also have the ability to delete posts from the list. In addition, this is also possible to only disable a post from the front page if it should remain temporary invisible or created before publication.

News Editor

Title

Text

Source

B
I
U
S
x₂
xⁿ
I_x

Styles ▾
Format ▾
Font ▾
Size ▾
A ▾ A ▾

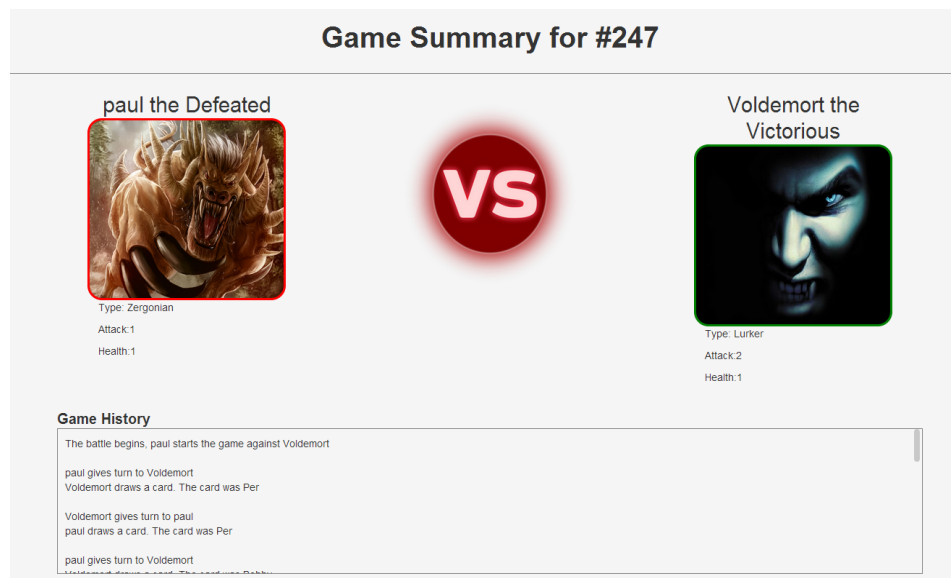
Create

Title	Author	Date	Enabled	Delete
Welcome to Souls!	paul	08.05.2014 16:36:30	Enabled	Delete
New Patch	paul	08.05.2014 16:41:39	Enabled	Delete

2.8 Admin panel – News editor

2.7 Game summary

After each battle has ended, the player is sent to the game summary screen where he sees his own and the opponent's stats together with a full log of the game's progress. This screen may be accessed afterward in the recent games list.



2.9 Game summary screen

2.8 The players

In the bottom center of the screen, you find your main character (The hero). Your objective would be to keep him alive until you defeat your opponent's hero. On the top, you have your opponent's hero. When you kill him, you win the game. The heroes are able to attack every card including the opposing hero.

Around the player portrait, you will see the available amount of mana or the remaining/required amount if you hover over a card. On the same object, you will also see the attack damage and health of the hero in the same way as the cards.

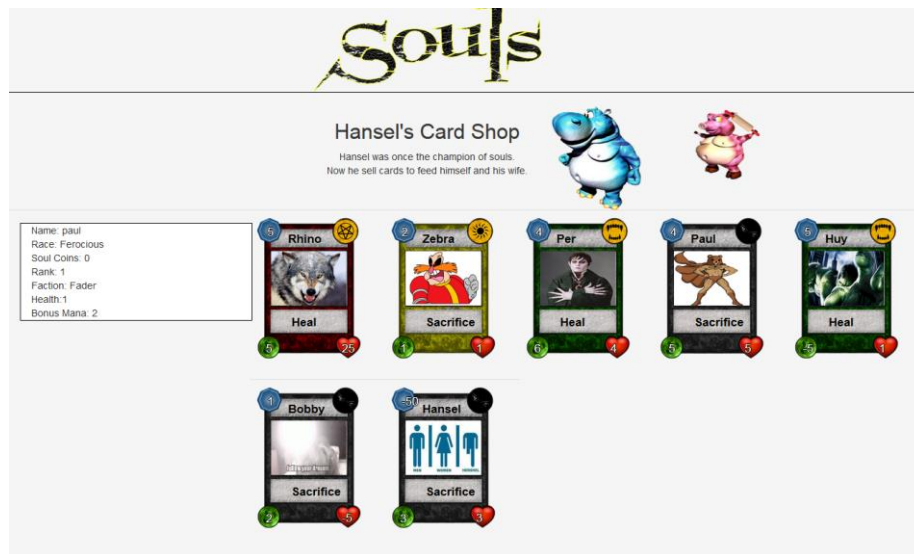


2.10 Enemy hero

2.9 Card Shop

When going to the soul shop you will be greeted by the old master of Greenheart, Hansel. He will help you gather more types of souls for your army if you can provide the soul coins.

This is where the custom made cards from the admin section appears together with the starting cards. The cards can't be refunded after purchase. Still, the admins may change the cards stats afterward to balance the game.



2.11 Card Shop

3. Technical requirements

In the pre project period there were multiple meetings and discussions regarding which technologies and libraries that were optimal for this project. The group did a lot of research to ensure that the project would be doable with the current technology in the browser market. An important part of the project is the communication between the client and servers, REST/AJAX, SOAP and WebSockets was the most appropriate for the task.

All of the technologies mentioned above are suited for the task, and we found pros and cons for all of the alternatives. We ended up with WebSockets-sharp as our communication library, which runs on a C# .Net platform. WebSockets-sharp had much better performance in both asynchronous tasks and general overhead on the network traffic, rendering REST and SOAP inferior.

The game server communicates with the web client running JavaScript and sends updates to the players containing the current game-data. The server will also receive requests to authenticate the players and their moves, then calculate the new game info and update the clients with the new data. All the game logic and game communication is done by this server, there is no game logic calculated client side. The main reason for this is to prevent cheating and modification of data before sending it to the server.

The web server manages the login (user authentication), making of hashes (for player identification and anti-spoofing), admin-panel and statistics. The game also have a MS SQL database for player, card, log and essential game info on the same host to reduce request delay.

“MySQL connector” does SQL connection with the ORM named NHibernate for managing and structuring.

3.1 Licensing

From the start of the project, we tried to use as much MIT licensed libraries as possible. MIT license is free to use. However, its only restriction is that the copyright notice is intact.

Luckily CreateJS[2], PIXI[3], Bootstrap[4], RequireJS[5], jQuery[6] all uses MIT licenses.

JSON only requires the permission notice to be distributed with the code and that the software only being used for good, not evil [7].

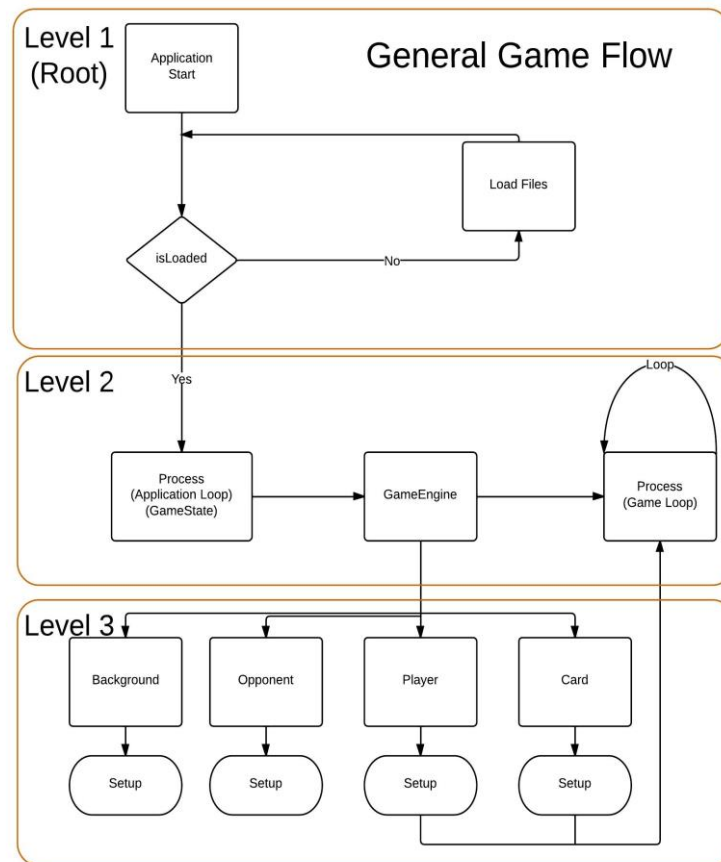
3.2.1 Class Diagram - Server



This is general everything that happen throughout the server loop.

Project report
DAT-219

3.2.1 General Game Flow



3.2 Game flow

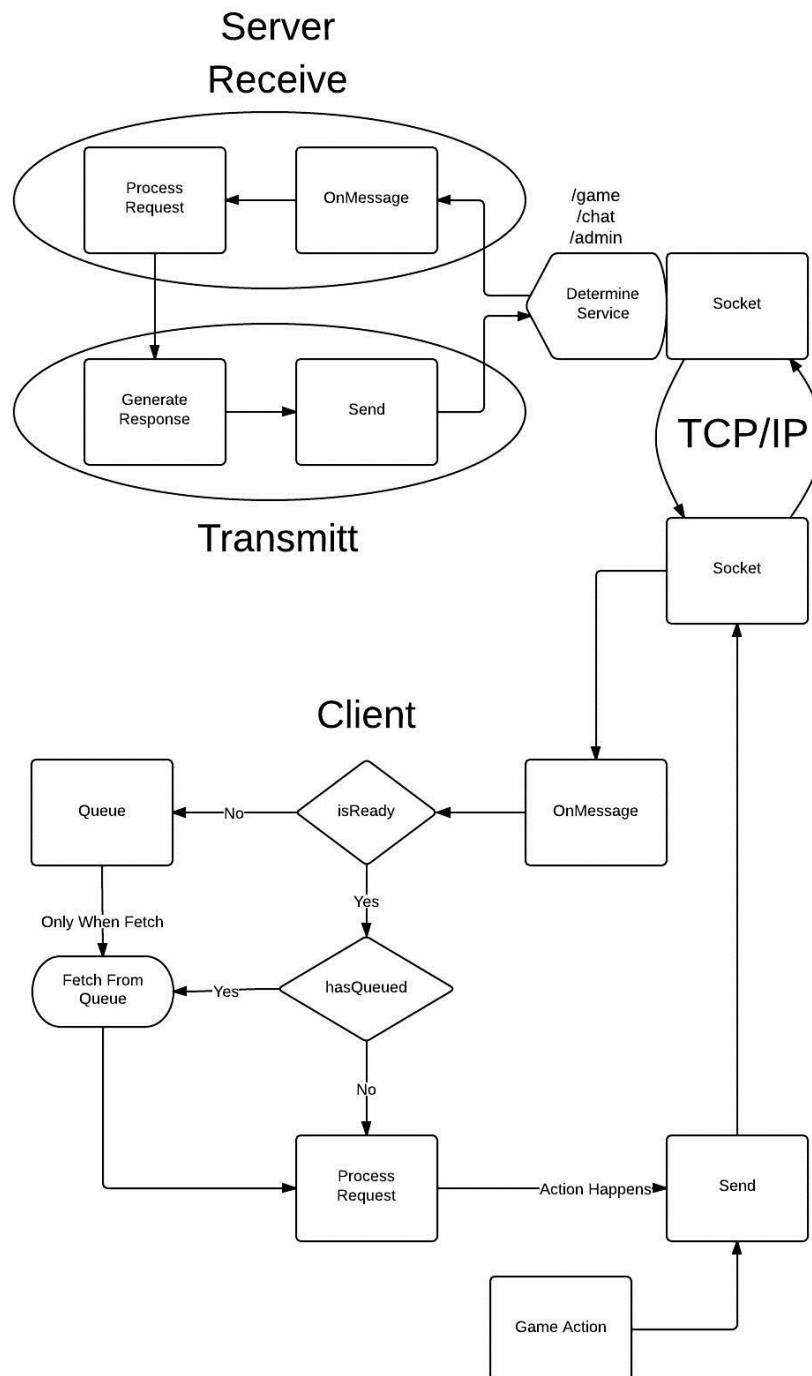
The client flow consists of mainly three levels, each of the levels have a task to perform before the next is executed.

Level 1 is start of the application. It starts of by creating a connection to the file server, and downloading all the game resources. This is a loop process, which stops when the game is fully loaded. The player can then select the appropriate game mode.

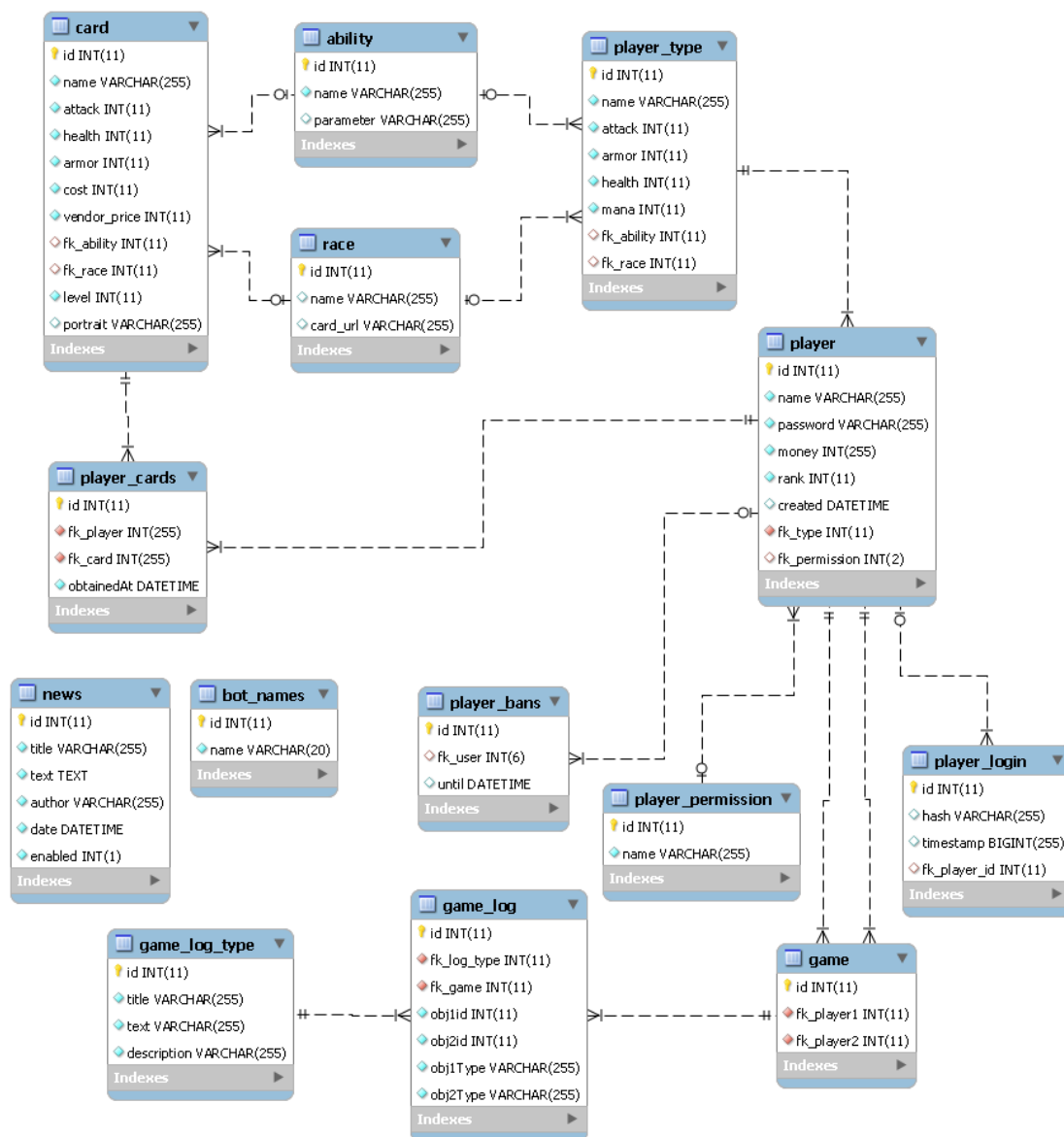
Level 2, the game loop sets up and runs the game until its ended. It is considered the middle of the application as it has no end-phase while the game is played. When a game is ended, Level 2 exits and the game client page is disposed.

Level 3 is the initialization phase of the game. This phase is very short and only takes care of creating, Player, Opponent, Background and Card objects. When this is done **Level 3** is done and goes back to the already mentioned **Level 2**

3.2.2 Network architecture



3.3. Network diagram



3.4 Database diagram

The database is relational having only two roots in its tree, **player** and **card**.

Player table contains all of the player information, used to authenticate and play games as a unique player. Card is root for card information whereas the relations gives information about ability and race, which is shared with the player_type table.

This implementation of the database is relatively optimal, but there are some things that could have been executed differently. For example, the news table is fully separated from rest of the table, it could have been optimized to use playerId instead of a varchar for name. Moreover, length of fields could have been optimized. An estimation of optimizing is below 1% improvement in terms of query speed.

4. Implementation

Souls use some new technology, specifically WebSockets-sharp for communication and WebGL for rendering. This made the project development a little unpredictable. This later resulted in a change we had to make from WebSockets to WebSockets-sharp. Some features started to make complications and we searched for a better alternative. Luckily, the migration to WebSockets-sharp was not too complicated. More about this in the WebSocket-sharp sub chapter.

When it comes to JavaScript there are many ways of structuring the code, this makes it hard to know what structure that will work best for the game client. After two major reconstructions of the code, the seemingly best structure was with RequireJS. Even though the code becomes hard to maintain in a structured way, RequireJS will take care of all of the required dependency injection in our application. The client itself consists of a game loop. This loop runs at a certain amount of times per second and updates the screen. Some of the game functions will also run each of these iterations to ensure the game is updated and fluent.

A very lightweight “state machine” will be used in our loading and process loop, which will determine what process and draw function that is going to be called.

The game, chat and admin servers are written in C#, using WebSockets to communicate with the client. The game server runs a “poll-queue” function at a predefined interval to match players. For every move made by a player client side the game-server receives the request, processes it and determines if the move is legal or illegal. Then the game-server responds by sending both players the updated game info.

4.1 Game Textures

Most of the game textures were obtained from different places on the internet and edited used Photoshop CS6. The reasons we edit the textures is to get an appropriate design and originality. Some textures like the mana crystals around the portrait is originally white but filtered into other colors by the client. The arena is originally a large water stadium with random signs on the sided but edited to sand with Souls logo on the sides.

4.2 PixiJS (Rendering Library)

Rendering is an important feature in our application, as we require Graphics for the user experience. It is important to note that PIXIJS is **not** considered a game library.

It does not have features for game development, so this was something developers had to make themselves.

This library gives very fast bindings to WebGL and Canvas and is the recommended library to use in 2D graphics in WebGL. There are game libraries using this renderer, namely PHASER.JS.

PixiJS takes care of all graphic objects in our project, IE Cards and Players and Text.

Cards are built up of multiple sprites, which at the end became one **Display Object**. This object is then rendered onto the screen with an interval of 60 frames per second.

4.3 Asset loader

The asset loader's job is to cache all the textures and sounds required by the game at any stage. This prevents delay on the client and excessive network usage on both web server and client.

On a fast connection, this feature is less noticeable but on slower connections, you may suffer from delayed sound and slow loading texture without an asset loader.

4.4 Sounds

All the sound and music used is downloaded or bought from different sites at the internet. No sounds are custom made at this point. All sounds are available at the web server and are being loaded to the client through the resource manager. After the audio is loaded to the client, all management of the sounds is done by SoundJS.

The only client manageable sound is the background music. The rest is played on events like attack or defend.

4.5 Game Engine (client side)

The game engine consists of the initialization code and processing code for the game. At initialization there are added 14 different PIXI groups, the purpose of these is to organize the sprites of the game in layers. (First group at the back, like background and cards further ahead in later groups). Another thing that is great about these groups is their possibility to define a property all items in the group, like visibility.

Then the engine makes two new sockets, one game and one chat and then connects them to the defined servers. All game and chat communication between server and client happens through these sockets.

After these are set up the game engine proceeds to instantiate, other classes needed and initialize these if necessary.

Besides, from the screen message, the last part of the game engine is the process loop. Like most games, this is a vital part and is needed to keep the game updated with fluent motions and right data. The engine has an own Process List containing all function that needs to be processed each tick. The process code for the engine is simply a for loop iterating through the process list functions and executes them.

4.6 Chat client

The chat-server runs parallel with the game-server. When the communication server gets a request it is defined by a "message type" if this is game or chat data. This structure makes it easy to implement more types of services using the same socket. The chat server then processes the request depending on if it is a "make-room", "message", "invite" etc. and runs it through the right functions before sending a response to the requester. (For example "room made by Karl with id 4" or "Tor kicked from room 4 by Karl")

4.7 Cards

All playable cards are stored in the database with their permanent values. The cards are loaded from the database to a dictionary at the game server each time the server start.

The main function of the cards are to serve as fighters for the player, each card/soul have their own mana cost depending on strength .Pixi has the ability to anchor images to each other which we used a lot at first, but the group later realized it was better to make finished cards for each race where the only objects coded to the card is the variables like mana, health, damage and image.

On the client, the card is recognized through the JSON string sent by the server and built and updated from those values.

When the card dies it rotates and floats towards the side of the screen, when it is outside it gets garbage collected.

Cards may also be added, deleted or edited through the card editor in the admin panel.

4.8 Players

Like the cards all players are stored in the database with their names, passwords, types, rank and money. When a player logs in the web server checks the database for authentication and generates a hash for the player from name and timestamp. Then stores in the database for the game server to authenticate with later.

When for example a page is loaded, the hash of the player (stored in the browser session) needs to be checked against the database to prevent session hijacking.

4.9 Welcome Screen

The welcome screen is made to accept partial html documents to make it work as a plugin page. The plugins receives data through AJAX in JSON format and styles it with the bootstrap table layout.

As for now the plugins made and used are the following:

News feed/bulletin board: For welcome messages, MOTD, updates and important information.

Server Status: Each time the page is loaded, it pings the servers to check if they are online.

Game Updates: When there is a new update committed through GIT it shows up in this view with a link to the commit.

Recent games: Shows a list of recently played games on the server with a link to the game summary.

4.10 WebSocket-sharp

As mentioned we started using a library called Alchemy but this made complications. These were mainly as follows:

1. Some packages did not make it through, making the data corrupt and unreadable due to incomplete encrypted messages.
2. WebSockets was not maintained as well as WebSocket-sharp
3. No support for different services at the same port (Like game, chat, admin all running on :8140)

Luckily, the migration was not complicated and after migrating things got easier and more structured. There still existed a problem with the server still sending messages after the connection was closed but this issue was fixed by the developers after the group member Per-Arne submitted an issue to the project's GITHUB repository[1].

When the communication server running WebSocket-sharp receives a request, it passes the request forward to the right service (game chat or admin-server). Then it gets a response from the game, chat or admin server to forward a response to the specific clients. It then checks what IP, port etc. that is connected to the specific players and sends the response.

All three servers are derived from a main service class that extends the WebSocket-sharp library. This requires the three servers to overload the service class's functions.

5. Validation and testing

5.2 Unit Testing and Stress Testing

The SoulsTest project is designed to unit test important features of the game. One of these is Attack validation. It is important to do such tests to keep consistency between Client and Server logic. Current implementation of testing is somewhat weak, this is due time restrictions. The group did however make a framework for further implementation of tests in both Client and Server application.

Stress Testing is necessary in order to assess the server's capabilities. Souls server has integrated stress tests which can be activated by running with the argument "debug" in the execution command ***SoulsServer.exe debug***.

Stress testing will hook all users registered in the database to bots and play against the general bot Entity. 200 Users in the database will equal 400 connections to the server. We estimate that our server will currently be able to handle 200 concurrent clients, which is an acceptable client count in the first server release.

5.1 Console log illustrating a stress test running 250 concurrent clients

5.2 Integrated Testing

Growth of Souls made developers think of Testing, It quickly became hard to manually test all of the functionality. Integrated testing takes all of the modules (Game and Client) and checks that it works well together.



There were no plans for advanced tests in the beginning of the project, and was therefore also not easy to implement the tests we wanted. This is because the design of our applications does not go well with such tests.

Souls is currently built and Tested in TeamCity - A Continuous Integration software developed by JetBrains.

Our test plan goes as follows

- **Stop Souls Server**
- Checkout Souls from GIT
- Build project via MSBuild package (Compile and Build)
- Run tests with MSTest package (Fails on errors)
- Run Grunt Script for minifying and uglifying the souls Client
- Publish Client on IIS
- **Start Souls Server**

This is the basic integration flow in our application. This saves hours when new builds are to be deployed.

Grunt is also an integration/build tool, which runs in a JavaScript environment via **Node.JS**.

6. Results

6.1 “A” Priorities

SERVER

As for the server there were two reconstructions. First, the group tried to find a good way to relate context (player socket information) to player and gameplayer. There were never a good solution to this with the current library we used (WebSockets). The other reconstruction was from WebSockets to WebSocket-sharp, which featured both a good way to structure the first problem and solved some issues mentioned earlier. After these changes the server structure became pretty good. Now the server consists of over approximately 20 classes where two of these are WebSocket-sharp and a generalized server. From this there are three servers derived from this that manages the game, chat and admin services.

WEB SERVER

The webserver runs ASP.NET MVC 4 with bootstrap. With 8 controllers, 10 views and 5 partial views (used as plugins for welcome page) every important aspect of the games web interface is covered. There were no accurate plans for any of the web pages but they all turned out well after some work. A great feature is the previously mentioned plugin feature for the welcome page. As this makes it possible to easily implement more partial pages the expansion possibilities of this page is theoretically limitless.

The other pages such as shop and all the admin pages use a combination of clean html, bootstrap and jQuery. There is not much work done on the design of these pages but the group is satisfied with the design produced in the project period.

GAME CLIENT

After two rewrites and three redesigns the projects game client were in a stable enough state to be released. JavaScript, which the game client runs on, is a very volatile language, mostly because it does not include **strict types** or any reasonable coding convention for **OOP** programming. In the beginning of the project, the client was made without any classes, and it quickly became a problem, because the scale of the project made it near impossible to control the data flow.

In the planning phase, a library called **RequireJS** was researched. It was not determined if this was something required for the project. But after the mentioned code rewrites, we saw no option but to implement it.

What RequireJS did was to separate each of the components in the game, IE Card, Player and Opponent, and gave us the opportunity to instantiate these as regular classes. It surely could have been done without RequireJS, but it worked very well with how we wanted our code design to be.

After **5197** lines of code and **27** modules, Souls game client was completed, and gave the results expected from the pre project phase. Not only did souls contain all features planned, but also additional features like Tweening.

TWO-PLAYER

The multiplayer feature came out well. Both players have the possibility to do exactly what they should be able to both at their turn and the opponents. An extra visual feature that was added is the ability to see the other player moving his or her card. There should theoretically be room for an unlimited amount of games though each game uses around 2-3 MB of memory. Mentioned later the game also have a single player functionality, which still is two players, but the opponent is a bot.

6.2 “B” Priorities

MUSIC/SOUND EFFECTS

At first a sound library named Buzz [8] was used but was later changed due to the limitations of the number of sounds played simultaneously being only one. This was a deal breaker for a game like this so the sound library was changed to SoundJS[9] which is a part of CreateJS used for tweening. After this, sounds and music work well and as intended.

VISUAL EFFECTS

The first tweening library used was from a single developer from github but after some issues that was not fixed even though they were reported [1] the tweening library was changed to Create JS's Tween JS which also requires EaselJS as a ticker for the tweens. There were a lot of bug fixing considering this cause to multiple and a little complicated visual effects happening at the same time. Most of these were resolved but some rarer still remain. The group have decided not to spend time on the rare bugs, as none of these is game breaking. Other than that, the result was better than planned because of multiple unplanned features. The only planned effect that was dropped is the arched styled card formation, but the ability to have a fluent card system was more important than having the arch formed design. It instead became a squared style, which was much easier to control when having different card configurations.

LIMITED AMOUNT OF CARDS

This was one of the last features implemented and was almost dropped due to prioritizing. Still it was implemented both because it was a B priority and the group had extra time. The cards for each player is stored in the database with player id and card id. This seems to work well.

CHAT CLIENT

The chat client was changed from secondary feature to a B priority. The server is built to be able to make rooms, invite and kick but as for now, only the game room is used. Which was the originally intended feature. The possibility to make own rooms is set to future work.

6.3 “C” Priorities

VENDOR

As for the “limited amount of cards” point the group considered dropping this feature as these two is dependent on each other. Still when making the limited card support this was also implemented. The vendor is also a main character in the game lore and has his own story.

WORLD MAP

World map was dropped because of unnecessary and the group later found out this would not be a feature worth implementing in the game. If it was implemented it should have shown a world map with different locations for each rank. Where the player traveled to new places each time he changed rank.

GAME LORE

At this time we have a game lore containing four races with a hero and a main villain. The players quest is to gain enough strength to defeat the villain, with the help of the retired hero who owns the shop. The group considered adding lore for all the cards and heroes but this was dropped as this was not originally intended and probably would not be worth the time it takes.

SINGLEPLAYER

Single player was another feature the group considered to drop due to no good solutions to the problem with implementation. At a later time a good solution for this was found and the feature got implemented. The bot do not have a very smart AI. It places out the card it can afford and attacks with it. The bot will not attack the players hero, which makes this more of a practice mode. The single player is also a good way of testing and bug fixing since you do not need two browsers or a friend to play with. Another thing about bots is the possibility of unit testing.

ADMIN PANEL

The admin was originally intended for statistics to implement XML into the project but was later expanded to have multiple administrative pages like the card editor, player management and news manager. The card editor is a GUI for writing new cards into the database, with a picture. Player management gives the ability to see what players are online and kick, ban or reset the player. News manager offers the option to add or delete news from the bulletin board.

6.4 "D" Priorities**3D**

As explained earlier this part is for the future work, this is not implemented and never was planned to be in this project period. However, it should be implemented would the first probably be a 3D arena followed by players and maybe UI.

CUSTOM SOUNDS

As there are plenty of sounds on the internet this probably will be the last priority in the games development. This would probably require a studio and some expensive equipment to get sounds of good quality.

6.6 Summary

Much time was put into this project, and it would not have been as successful if it were not for the team play in the group. The project can generally be categorized as a **Great Success** because we did not encounter any big problems while in development phase.

All of the expected priorities were met, and some additional features were implemented to make the game even better than originally planned.

It was an interesting project to work with, and the developers learned a lot from trial and error in the project time.

6.7 Future Work

To complete all “C” priorities the game is still pending a fully functional chat along with a world map. Should the game continue to be developed after the project period these functions would probably be the first to be implemented.

Along with more cards, heroes and lore, more abilities could be implemented to increase the games content. A search function for the admin panel functions would also be preferable.

When the implementation of all “C” priorities is completed the remaining “D” priorities is next. These are 3D graphics and custom sounds. There are libraries to make 3D in JavaScript but it is not that these will suffice for this games requirement.

Extended support for tablets. Like the abilities feature since it is not possible to right click.

Memory Management on the server is something that is a problem when 150 players or more connect to the game server. This is due to the way our logging system works the moment. What we would like to do is to create a self-dumping system that sends the logs in smaller batches. This will reduce the memory dramatically.

Database connection is also straining the game server in stressful times, so implementing the login server separated from the game server thread would also improve memory and load speeds.

Even though registration requires an email address there is no email server and no “forgot password” using this. To make this feature work a mail server could be implemented.

7. Support

7.1 Browser compatibility

Browser:	Issues:	Compatibility:
Internet Explorer	Fails on multiple JavaScript features.	Poor
Chrome	No known issues	Excellent
Firefox	Some minor caching problems	Good
Opera	No known issues	Excellent
Safari	Not tested	N/A

8. References

- [1] <https://github.com/sta/websocket-sharp/issues/34>
- [1] <http://tinyurl.com/lzuvxty> (The fix)
- [2] <https://github.com/CreateJS/TweenJS/blob/master/LICENSE>
- [3] <https://github.com/GoodBoyDigital/pixi.js/blob/master/LICENSE>
- [4] <http://getbootstrap.com/getting-started/#license-faqs>
- [5] <https://github.com/jrburke/requirejs/blob/master/LICENSE>
- [6] <https://jquery.org/license/>
- [7] <http://www.json.org/license.html>
- [8] <http://buzz.jaysalvat.com/>
- [9] <https://github.com/CreateJS/SoundJS>

9. Appendices

9.1 Code Standards

We have decided on the following standards and guidelines for this project:

- Keep the complexity as low as possible where applicable
- Code comments before the code
- Variable and function names in capital CamelCase
- “Useful” variable name wherever applicable
- Class names start with capital letters.
- Enumerators and static values: Capital letter only (ex. static final int TOP = 1)
- General code formatting using the tool standard (Fix indentations, remove blank lines etc.)
- Class, method or function start bracket on the next line
- Names of methods and classes should be as self-descriptive as possible
- One statement per line (not like this: a = 1; b = a*a)
- Flags are Boolean only.
- Recycling: If the same “large” function is used with various variables in multiple places the function has to be written as a method to limit redundancy
- No spaces in variable and method names. If spaces have to be used in other places, “_” is used instead.

Generally this is what’s described in the C# coding convention guide from Microsoft found here:
<http://msdn.microsoft.com/en-us/library/ff926074.aspx>.

9.2 Team Contract

Main goal:

Develop a well-tested and structured game within the project timeframe.
Deliver a product containing the specified A, B, C goals and hopefully more.
In order to fulfill this goal the group has agreed to the following:

1. Participation:

It is expected that the project tasks are equally distributed and that the person tries to complete their tasks in time. If a group member can't meet at the designated time he or she is required to inform at least another group member.

2. Responsibilities:

Every group member is expected to complete their task with their best effort and ask if there are uncertainties or difficult tasks.

3. No conflict policy:

Listen and respect the other group members. Only constructive criticism and no shit talk. On disagreements the group member in charge of the part being disagreed on will have the final word. The other members are obligated to respect this and adapt the code to this choice by best effort without being butt hurt.

4. Meetings:

Time should be agreed with each member of the group, to make everyone happy.
If someone didn't show up he or she will have to buy cafeteria dinner to the other group members

9.3 Meeting Reports

Refferat 04.03.14

- Skrivning av forprosjektrapport.
- Forklaring av prosjekt (en del javascript pga grafisk).
- Viste foreløpig demo.
- God start på prosjektet.

Refferat 18.03.14

- Forprosjekt ser bra ut (forslag til neste prosjekt: Legg til funksjonalitet tabell).
- Ok så lenge ASP spiller mer rolle enn login (Adminpanel, statistikk ++?).
- Viste foreløpig demo med 2 spillere.
- Fortsett i samme tempo.

Refferat 25.04.14

- Begynne på rapport.
- Fullfør spillet.
- Ikke prioriter nettside.

Refferat 6.5.14

- Avtale fremføringsdato.
- Rapport ser foreløpig bra ut.
- Max 3 nivå på innholdfortegnelse.
- Viste demo med ny welcome page og nyere funksjoner.

Refferat 13.5.14

- Ser bra ut.
- Legg til database diagram i rapport.

10. External dependencies

- Pre-report - "Forprosjektrapport DAT219-Souls.pdf"
- Power Point presentation - Souls-PP.pptx
- Git log - git-log.log
- Jira-Issues-Souls.png
- Jira-Timesheet-Souls.png