By

Per-Arne Andersen and Paul Richard Lilleland

Contents

1. Project description

Souls is a massive multiplayer online collectible card game (MMOCCG). As far as we know there do not exist any free to play version of a game like this that can be played in a browser.

Souls is a game developed for use in web browsers such as Google Chrome and Mozilla Firefox. The game uses various top notch JavaScript technologies for the game design. While JavaScript is a big part of this project, we also have a large portion of ASP.NET involved in the backend for the game client. For this game to work we need to find a good way for the server to communicate with the game client, the server shall be programmed in C#

Background

We settled on this idea after a lot of both good and bad suggestions. At first we decided to go for a media center type streaming application similar to Netflix but in a much smaller scale.

We had this technology working but decided to cease development because this project was simply not suited for this course.

The project we decided to start with is a massive multiple online card game (MMOCCG), similar to Hearthstone and Magic the gathering, played in the web browser using JavaScript and asp.net.

This is a project that requires hard work from all team members. It's easy to increase or decrease the functionality of such a project if needed.

Souls will not only meet the requirements of technologies in this course, but also go deeper into the web communications technology which hopefully will boost motivation.

Magic the gathering was the first modern collectable card game introduced in 1993 and is still very popular. The digitized and more simplistic version of this game, hearthstone is under development and soon to be released. This is the main inspiration for Souls, and we plan to use this as reference when implementing game mechanics, doing this will allow us to focus more on game technology resulting in a better end product.

Organization

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.

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 making their own branch on GIT.

A 50/50 split in responsibility and "ownership" will hopefully give both team members better motivation developing the project.

Code standard

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, that 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.

2. The project

Below is the description of project goals, limitations and prerequisites:

Goals

Game Server	**A**
Web Server with resources (textures/sprites/sound)	**A**
Game Engine	**A**
Web client	**A**
Two player game	**A**
Music/sound effects	**B**
Visual effects	**B**
Limited and manageable cards	**B**
Chat client	**B**
Vendor and currency	**C**
World map	**C**
Game Lore	**C**
One player game (with AI)	**C**
Admin panel	**C**
3D version	**D**
Custom sounds	**D**

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 don't plan to implement features in this category.

Limitations

Because the project consists mostly of .Net features and JavaScript, the server should preferably be a windows server. There are ways to run this on Linux but it will not be supported. Since the game client is a JavaScript and played in a browser there will be some graphical limitations compared to other languages but no features will suffer from this as far as we know.

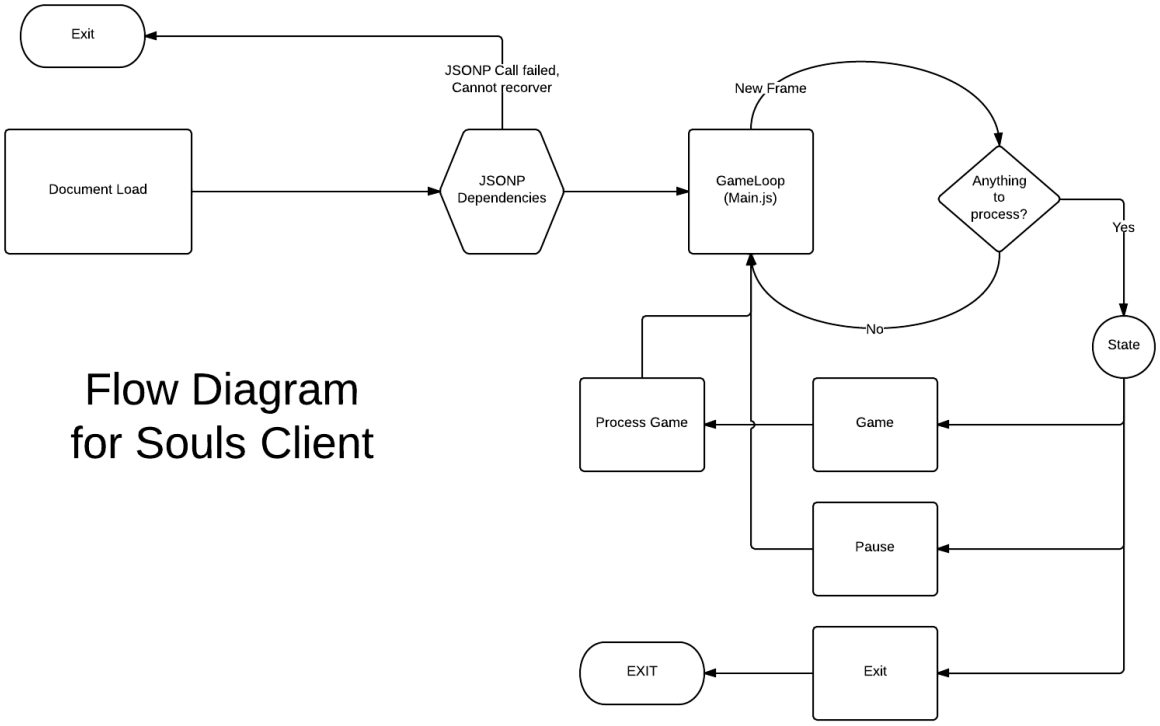
Prerequisites

For development we use Visual Studio 2013, NetBeans, Notepad++, Navicat. We will need 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 needs a MS SQL Server as we use LINQ. Other than that C#, JavaScript, ASP, JSON, Ajax and WebSockets are used. As of now we plan to use Alchemy, Newtonsoft, Pixi.js, jQuery, Bootstrap and maybe RequireJS libraries.

3. Tech design

Souls will be built with some new technology, specifically WebSockets for communication and WebGL for rendering. This makes it hard to know how the structure will turn out in our end product. Though we have planned the design, it may not be like this at the end of the project.

Game Client

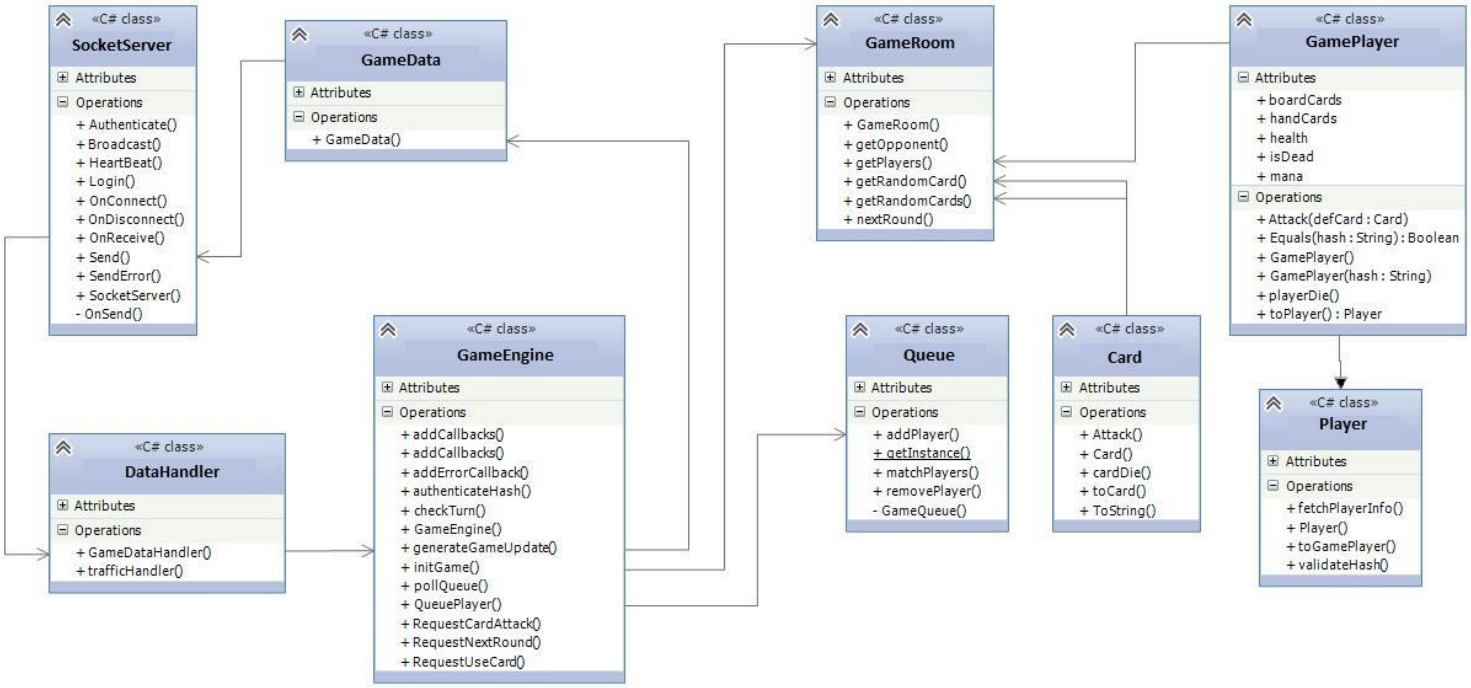


JavaScript is a language which comes with many ways of structuring code, this makes it hard to know what structure that will work best for the game client. Even though the code becomes hard to maintain in a structured way, RequireJS will take care of all of the required dependency injection in our application. The game itself will consist of a game loop. This loop will run at a certain amount of times per second and update the screen. Some of the game functions will also run each of these iterations to ensure the game is updated.

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

Features of the game client will most likely not be limited by the technologies used.

Game\Chat Server



The game and chat server is written in C# using WebSockets to communicate with the client. The game server runs a “poll-queue” function every 30 seconds 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.

The chat-server works 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 processes this before sending a response to the requester. (For example “room made by Karl with id 4” or “Tor kicked from room 4 by Karl”)

The communication server class is for the most part a template provided by the developers of the WebSockets library we are using. When it receives a request, it passes the request forward to the right service (game or chat-server). Then it gets a response from the game or chat 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.

4. Technologies

We had in our pre project period meetings and discussions regarding technologies which could be used for a project like we were about to begin. Research was done to ensure that the project would be doable with the current technology in the browser market. There were multiple alternatives, and some of these were REST/AJAX, SOAP and WebSockets.

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 as our communication to the server, which runs on a C# .Net platform. WebSockets had much better performance in both asynchronous tasks and general overhead on the network traffic, rendering REST and SOAP inferior to WebSockets.

The game server will communicate with the web client running JavaScript and send 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 players 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 web server will manage the login (user authentication), making of hashes (for player identification and anti-spoofing), admin-panel and statistics. The game also needs a MS SQL database for player, card, log and essential game info preferable on the same host for quick access.

We plan to implement a single player feature with basic AI. The quality of this is unpredictable.

Server side technology:

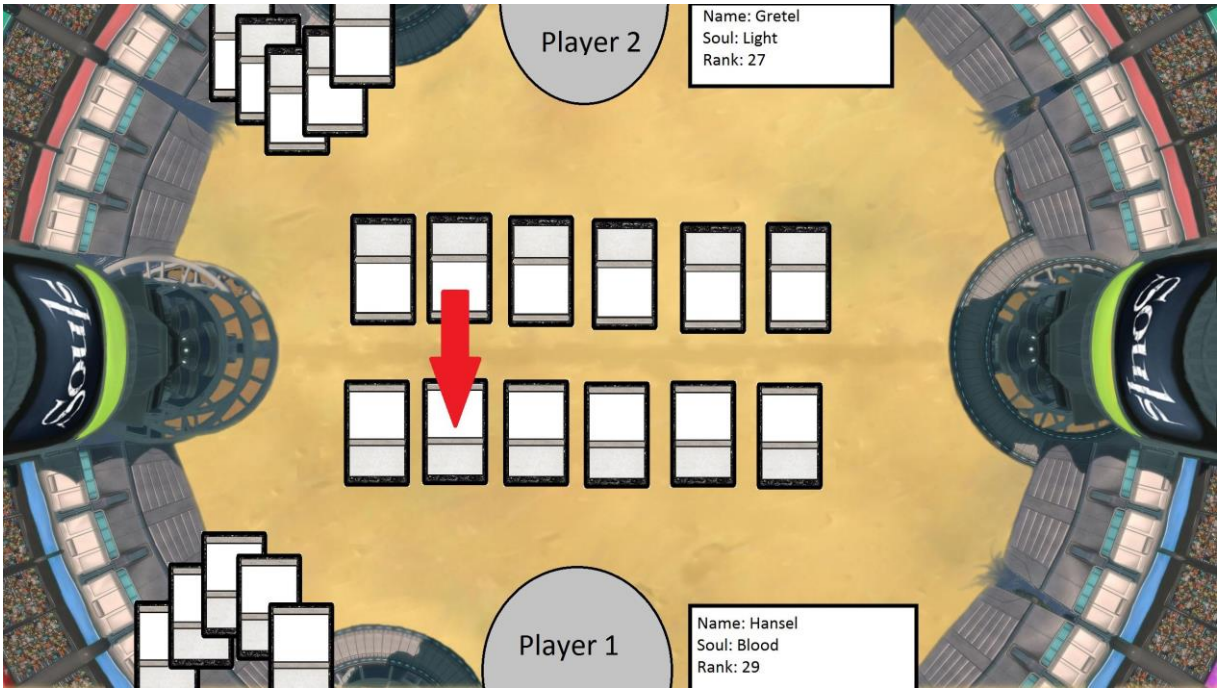
- C#
- WebSockets Server (websocket-sharp)
- REST API Server (WebAPI.net)
- MSSQL

Client side technology:

- ASP.NET MVC
- Ajax (JSON via GET/POST)
- JavaScript
- PIXI.JS
- SoundJS
- TweenJS
- jQuery
- jQueryUI
- RequireJS
- Bootstrap

4. Game design

Structure draft of the game client:



Login:

The first page that meets the player consists of a username, password and hopefully some graphics with audio.

Main Page:

This page will meet the player after login. This is where the player is able to check his profile, statistics, online players, ladder, and queue for a game and chat with friends or in chat rooms. A secondary feature for this page will be the world map showing the players geographical location based on his\her rank.

Queue:

The queue feature works by matching player every 30 seconds. After a player enters the queue there will spawn a timer showing the remaining time. An alternative option will be a tab\window autofocus feature when the timer reaches zero making the player able to do other things without worrying about missing the game.

Game:

The game page will look similar to the structure draft above. 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 opposite players 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 each player has one mana at their disposal and on round 5 each player has 5 mana at their disposal. (Spells or abilities may slightly change this). In addition at the “sub-round” start the current player draws a card from their (predefined) deck.

The most important part of the game is the cards. Each card costs a predefined amount of mana based on strength (stronger card costs more mana to use). 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 type is dark, light, blood or nature. Attack, health and ability are different depending on the cards “strength”.

At the beginning the player has all cards on hand, then placing them using mana as explained earlier. After the card is placed it can’t be returned or moved, only killed. Each player has their own set of cards\minions on the board being 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 choose is also a playable character on the board. When you or the opposite players health reaches zero the game is over and surviving player wins. You have to carefully choose if you should attack the rival player or their minions. If you focus on the player you may find yourself overwhelmed by enemy minions.

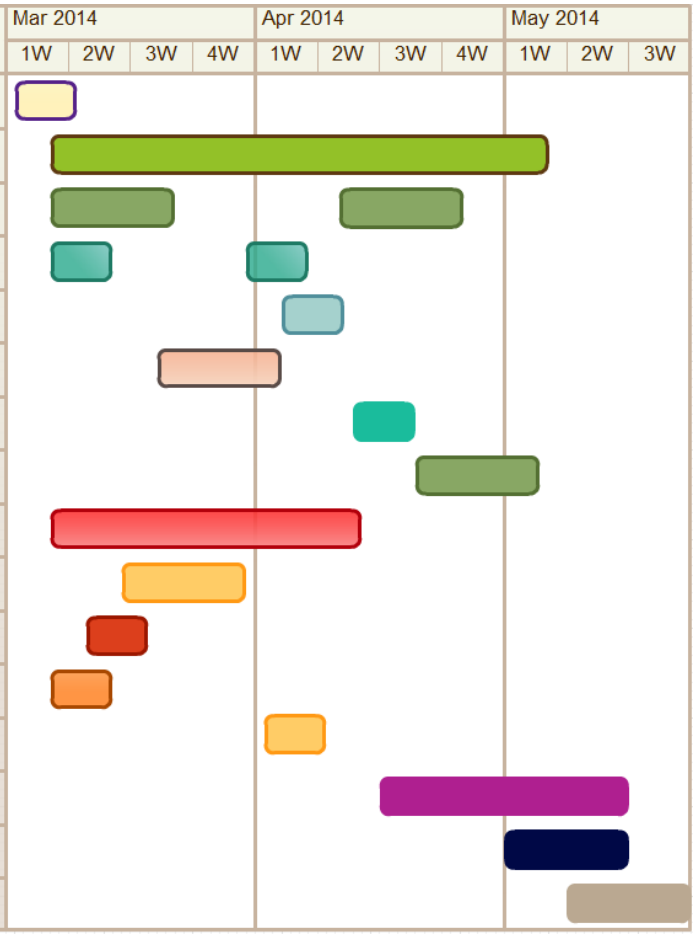
A feature that hopefully will make the game more interesting is abilities. For example if a card has the ability “deal one damage at spawn”, the card may deal one damage to an enemy minion specified by the player. Each player, depending on character and type also has their own abilities. For example a character of type “light” may have the ability to heal their minions.

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 will also have the possibility 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)

6. Progress

ID	Task Name	Duration
1	Pre-Project Report	
2	Frontend	
2.1	Game Interface	10
2.2	Web Interface	10
2.3	Chat Interface	10
2.4	Game-server Communication	5
2.5	Chat-server Communication	5
2.6	Sound and Graphical effects	5
3	Backend	
3.1	Client Communication	5
3.2	Game Logic Management	5
3.3	Web Server Server	5
3.4	Chat Server	5
4	Secondary Feature Implementation	5
4	Evaluation	10
5	Project Report	20



7. Group 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 butthurt.

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