# Requirements Specification
# for
# SENG 299 “Go” Project

**May 2016**
**Version 0.1**

**Prepared By**
**Alex Laing <lainga@uvic.ca>**
**Alex Rebalkin<alexreba@uvic.ca>**
**Charlie Friend <cdfriend@uvic.ca>**
**Jia Xu<xujia@uvic.ca>**
**Tyler Harnadek <tylerhar@uvic.ca>**

# Table of Contents

# 1 INTRODUCTION

## 1.1 Purpose

The purpose of this document is to facilitate the production of a “Go” web application. This application will allow users play and configure the traditional Chinese game Go.

## 1.2 “Go” Background

The “GO” game originated in China in ancient times. “GO” was one of the four cultivated arts in China. The game reached Korea by the 5th century, and reached Japan in the 7th Japan. By the early 20th century, it had spread in western countries. Google DeepMind developed a computer program AlphaGo, an automated program to beat Lee Sedol, a professional human Go player. AlphaGo’s algorithm makes its every move based on its previous knowledge, specifically by deep learning method and extensive training from computer and human play.

## 1.3 Rule Set

Our implementation of Go will utilise a traditional ruleset. The rules are as follows.

### 1.3.1 Playing Field

1. Go has two players; one plays black stones, and the other plays white.
2. The board is a square grid, consisting of horizontal and vertical intersecting lines. The board is 9x9, 13x13, or 19x19 horizontal and vertical lines.
3. There is no limit to the number of stones that either player can play during a game.
4. The game begins with an empty board. A handicap may be applied, in the form of as many as 9 stones placed on the board prior to the start of the game. The handicap is established over repeated games against the same opponent. The handicaps are placed at set locations on each size of board.
5. Black places the first stone. Players alternate thereafter.

### 1.3.2 Playing The Game

5. Any intersection on the board exists in one of three states: empty, occupied by black, or occupied by white.
6. Two stones of the same colour placed adjacent to each other are said to be connected. Many stones can be connected, forming a chain.
7. An empty space next to a placed stone is called a “Liberty”. Liberties are shared across connected chains.
8. Stones are captured when all of their liberties are occupied by opposing stones.
9. A stone cannot be placed in a location with no liberties and no connected stones of the same colour.
10. A stone cannot be placed in a location that recreates a previous position in the game.
11. Players can pass on their turn at any time. When both players pass consecutively, the game ends.

### 1.3.3 Scoring

The application will implement three types of scoring.

1. Stone scoring is simply counting the number of stones on the board. The player with the most stones wins.
2. Area scoring counts all of a player's stones, plus any empty spaces that are entirely surrounded by a player's stones. The sum of the two is a player's score - highest score wins.
3. Territory scoring is the most complicated. A player first counts their area score, and then subtracts the number of stones that their opponent has captured. As before, the player with the highest score wins.

To compensate for black moving first, a "Komi" is given to white to compensate. A Komi consists of a number of points, usually 7.5, added to white's score automatically. The Komi does not apply if a handicap was used at the beginning of the game.

## 2 SYSTEM DESCRIPTION

The system that will be built is essentially a web application version of the traditional Chinese board game "Go". The system will be built primarily with JavaScript and if requirement will use a MongoDB database. The application will be presented in a GUI that is visually customizable by the user. In the application itself, the user will have three options to play the game. The first will allow the user to play against the provided AI and the second will allow two human users to play each other locally in a hot seat format. Finally, the third option will allow the user to play against another human user over a network. In addition to the three game play options the user will be able to select from three different board sizes, create user account and view game replays after a match has been completed.

## 3 FUNCTIONAL REQUIREMENTS

**[1.0]** The system shall allow sets of two users (automated or human) to play Go.

**[1.1]** The system must enforce the rules of Go detailed in section 1.3.

**[1.2]** The system shall end the game after both players choose to pass their turn.

**[1.3]** Upon ending the game, the system shall calculate the score of each player according to the method detailed in section 1.3.3.

**[1.4]** The system shall have a configurable timer system.

**[2.0]** The system shall support "Hot Seat" style play.

**[2.1]** The system must give a prompt after each turn, indicating that it is the next player's turn.

**[3.0]** The system shall support the use of 9x9, 13x13, and 19x19 playing boards.

**[3.1]** Board size must be chosen before the start of a game.

**[4.0]** The system shall be capable of interfacing with 3rd party go AIs, and an in-house AI if available.

**[4.1]** AI players must be capable of handling a "move" command. The resulting response data will be interpreted and used as the AI player's next move.

**[5.0]** The system shall allow users to create accounts or play anonymously.

**[5.1]** User accounts shall be protected by passwords which can be changed by the user on request.

**[5.2]** The system shall store a user’s number of wins and losses at the end of each game played, and adjust their handicap accordingly.

**[6.0]** Each game must be stored and available for review immediately after completion.

**[6.1]** The user must be able to step backwards and forwards through each turn of a game.

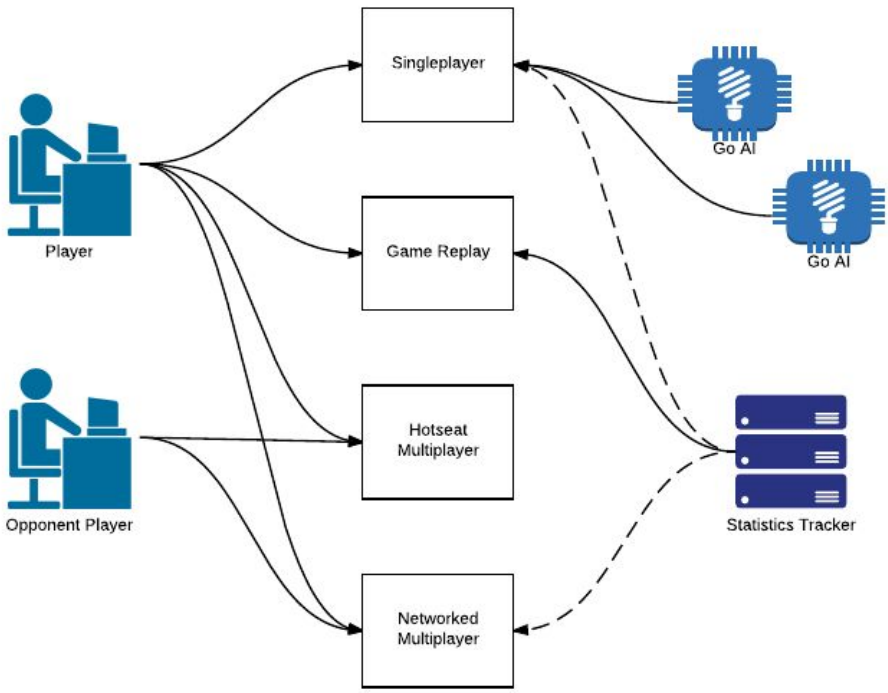
**[6.2]** The system must allow the user to save a text file containing a replay of the game in smart game format (SGF).

**[7.0]** Visual customization must be supported.

**[7.1]** The user shall be able to choose from a variety of predefined visual themes.

## 3.1 Use Cases

### *3.1.1 Use Case Diagram*




### *3.1.2 Use Case 1: Human vs. AI*

| | |
|---|---|
| **ID** | *Game-01* |
| **Description** | *An user plays against the provided AI.* |
| **Actors** | *Player, AI* |
| **Preconditions** | *A single user is playing.* |
| **Basic Steps** | *The game is initialized. Black plays, and then white and black alternate. The game ends when there are no valid moves or players pass consecutively. Post-game replay is displayed.* |

| | |
|---|---|
| **Alternate Steps** | |
| **Exceptions** | *Invalid Move: If an invalid move is played, the move is reverted and the user is prompted to choose another move.*<br>*Network Failure: If either the client or the server fails to respond to a request within a given period of time, an error message will be displayed to the user and the session will be terminated.* |
| **Postconditions** | *If the player is logged in, statistics are tracked.* |

### *3.1.3 Use Case 2: "Hot Seat" Play*

| | |
|---|---|
| **ID** | *Game-02* |
| **Description** | *Two users play against each other in a hot seat format.* |
| **Actors** | *Player, Player* |
| **Preconditions** | *Two users are playing. The users are logged in or have chosen to play anonymously.* |
| **Basic Steps** | *The game is initialized. Black plays, and then white and black alternate. The game ends when there are no valid moves or players pass consecutively. Post-game replay is displayed.* |
| **Alternate Steps** | |
| **Exceptions** | *Invalid Move: If an invalid move is played, the move is reverted and the user is prompted to choose another move.*<br>*Network Failure: If either the client or the server fails to respond to a request within a given period of time, an error message will be displayed to the user and the session will be terminated.* |
| **Postconditions** | |

### *3.1.4 Use Case 3: Networked Play*

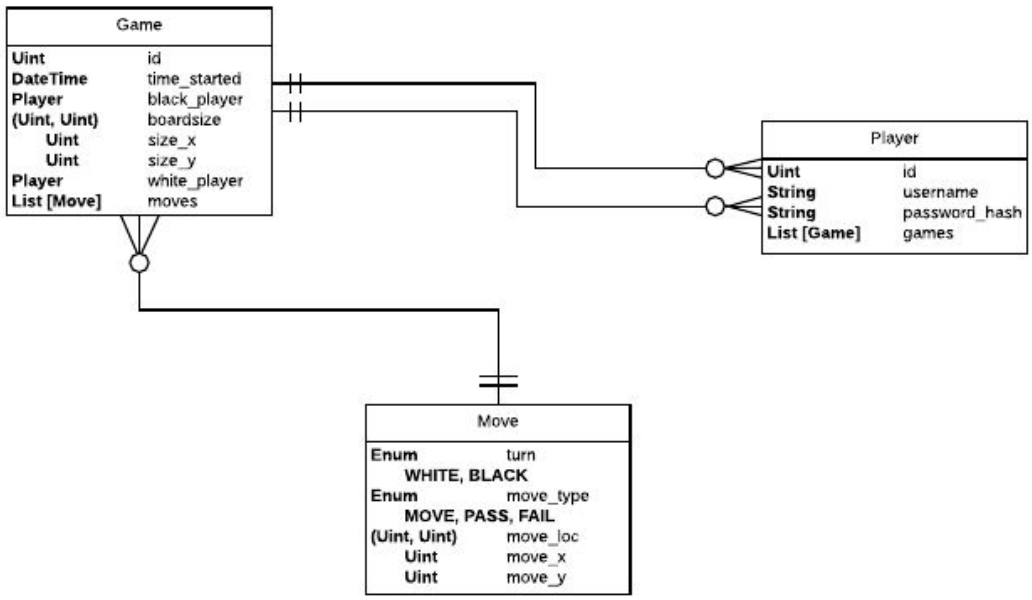
| | |
|---|---|
| **ID** | *Game-03* |
| **Description** | *Two users play against each other over the network.* |
| **Actors** | *Player, Player* |
| **Preconditions** | *Two users are playing. The users are logged in or have chosen to play anonymously.* |
| **Basic Steps** | *The game is initialized. Black plays, and then white and black alternate. The game ends when there are no valid moves or players pass consecutively. Post-game replay is displayed.* |
| **Alternate Steps** | |
| **Exceptions** | *Invalid Move: If an invalid move is played, the move is reverted and the user is prompted to choose another move.* |

| | |
|---|---|
| | *Network Failure: If either the client or the server fails to respond to a request within a given period of time, an error message will be displayed to the users and the session will be terminated.* |
| **Postconditions** | *If the players were logged in, the win/loss statistics are tracked.* |

### *3.1.5 Use Case 4: Review of Previous Games*

| | |
|---|---|
| **ID** | *Review-01* |
| **Description** | *A player reviews a saved game.* |
| **Actors** | *Player* |
| **Preconditions** | *The user has played a game, and the replay is saved.* |
| **Basic Steps** | *The user chooses the replay that they would like to view. The user is able to step forward and backward through each turn. The user can terminate the replay at any time.* |
| **Alternate Steps** | *The user may choose to download the file in Smart Game Format before terminating the session.* |
| **Exceptions** | *Network Failure: If either the client or the server fails to respond to a request within a given period of time, an error message will be displayed to the user and the session will be terminated.* |
| **Postconditions** | |

## 3.2 Entity Relationship Diagrams



## 3.3 Data Dictionary

### *3.3.1 Player*

A player object represents a single registered user. A player’s wins, losses and games are tracked, allowing the system to provide the appropriate handicap and opponents with a similar skill level. A player may be human or an AI (in which case some dummy Player object is used).

| Attribute | Type | Optional? | Notes |
|---|---|---|---|
| ID | Unsigned Integer | N | Unique identifier for each user profile. |
| Username | String | N | User’s name as displayed to other users and in game records. |
| Password Hash | String | N | Hash of the string required to access, make changes to or play games under a user profile. |
| Games | List[Game] | N | All game records associated with the user profile. |

### *3.3.2 Game*

Each game of Go should store a game object in the system. These give information on the moves made by each player and the players involved.

| Attribute | Type | Optional? | Notes |
|---|---|---|---|
| ID | Unsigned Integer | N | Unique identifier for each game. |
| Time | Date/Time | N | Date and time the game was initiated. |
| BoardSize | (Unsigned Int, Unsigned Int) | N | Number of squares in the game board. (width, height) |
| WhitePlayer | Player | N | Player using white armies. |
| BlackPlayer | Player | N | Player using black armies. |
| Moves | List[Move] | N | All moves played for the duration of the game. |

### *3.3.3 Move*

Several moves are played each game. Players or AIs can either move or pass during their turn.

| Attribute | Type | Optional? | Notes |
|---|---|---|---|
| Turn | Enum(WHITE, BLACK) | N | The color army being played for this move. |
| MoveType | Enum(MOVE, PASS, FAIL) | N | Whether the player moved the token, passed the turn or failed to respond. |
| MoveLoc | (Unsigned Int, Unsigned Int) | N | The location of the army played during the turn, or undefined if the turn was passed. This location |

| | | | |
|---|---|---|---|
| | | | should be a valid (x, y) coordinate on the game board. |

# 4 NON-FUNCTIONAL REQUIREMENTS

## 4.1 Technical Constraints

1. The system's user interface is required to be web-based. All user interaction will occur through a web browser.
2. The server must be written in Node.js, a server-side scripting framework in Javascript.
3. If the use of a database is required when implementing the system, MongoDB will be used.
4. This web application may allow users to play Go game on any device, instead of only on PC. This requirement is related to responsive web design (RWD).

## 4.2 Business Constraints

1. The system is required to interoperate with an external AI server. This server will respond to requests from the system with the AI player's next move.

# 5 PROJECT PLAN

## 5.1 Development Process

This project will follow the waterfall model for software development (as shown in the image below).

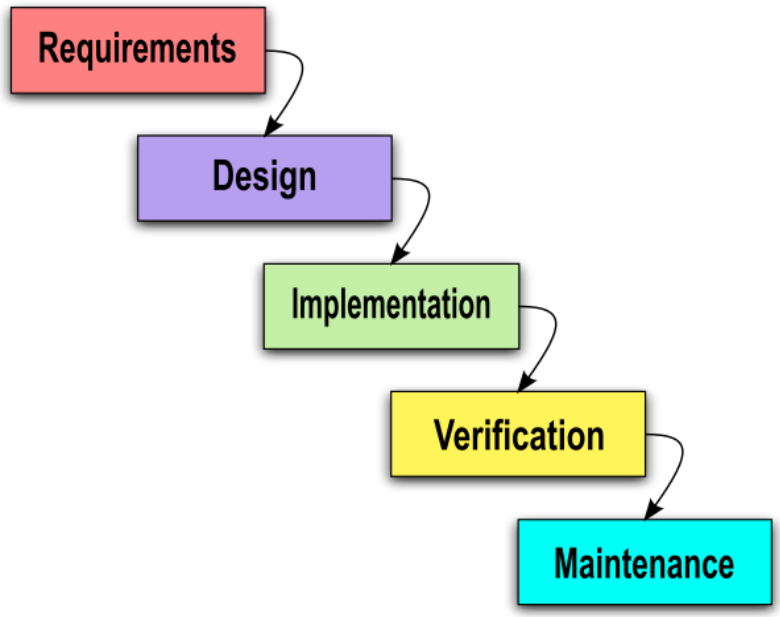



*Image courtesy: Peter Kemp/Paul Smith*

### *5.1.1 Requirements Stage*

If you are reading this document, the requirements stage is complete. Identification of requirements includes the brainstorming of desired functionality and identification of constraints on the project.

#### *5.1.2 Design Stage*

During the design stage, the method of implementation for the features envisioned in the requirements document is created. Documentation detailing the software’s internal and external interfaces and modules are created, and dependencies are identified. User interface designs are created through iterations of mock-ups.

#### *5.1.3 Implementation Stage*

Code for the application is developed during this phase. Unit testing is conducted during implementation to minimize bugs during the verification stage. API documentation is created for the code being written.

#### *5.1.4 Verification Stage*

Play-testing is conducted on the application. Bugs identified from tests are documented and eliminated. API documentation is looked over and edited.

#### *5.1.5 Maintenance Stage*

Bugs in the application are addressed as necessary. A final report is written to document progress made during the course of the project.

### 5.2 Project Team

| Name | Applicable Skills |
|---|---|
| Alex Laing | SSL/TLS and network security; OpenGL/WebGL/GLSL |
| Alex Rebalkin | Unix server management |
| Charlie Friend cdfriend@uvic.ca | Database design and management, server-side scripting, algorithm design. |
| Jia Xu | Web front-end development, also experienced in back-end development |
| Tyler Harnadek | Project management, organization. |

### 5.3 Timeline

Work on the project shall continue from May 12 to August 4, 2016. Estimated deployment date for the system is July 14, 2016.

| Phase # | Duration | Description |
|---|---|---|
| 1 | May 12-June 6, 2016 | Drafting, editing and completion of the System Design Report. Creation of the program skeleton and testing/deployment workflow (detailed in section 5.1). |
| 2 | June 7-June 26, 2016 | Drafting and minimal implementation of user interface. Initial implementation of basic system functionality. |
| 3 | June 27-July 13, 2016 | Assessment of changes proposed by design review report. Continued implementation, styling and debugging of the system. |
| 4 | July 14-July 21, 2016 | System should be feature-complete by the beginning of this phase. Final play-testing, debugging and unit testing. |
| 5 | July 21-August 4, 2016 | Ongoing maintenance of the system. Completion of final report on the project. |