**Software Requirements Specification**

**for**

**<SuperTicTacToe>**

**Version 1.0 approved**

**Prepared by <Team PlasmaJet>**

**<Dr. Fu’s Software Engineering 4283/5283>**

**<2-11-1>**

**Table of Contents**

**Table of Contents**

**Revision History**

**1.** **Introduction**

1.1 Purpose and Scope

1.2 Definitions, Acronyms and Abbreviations

1.3 References

1.4 Overview

**2.** **Overall Description**

2.1 Product Functions

2.2 User Classes and Characteristics

2.3 Constraints

2.4 Assumptions and Dependencies

**3.** **Specific requirements**

3.1 External Interface Requirements

3.1.1 User Interfaces

3.1.2 Hardware Interfaces

3.1.3 Software Interfaces

3.1.4 Communications Interfaces

3.2 Functional requirements

3.2.1 Functional requirement 1

3.2.2 Functional requirement 2

3.3 Performance Requirements

3.4 Software System Attributes

3.4.1 Availability

3.4.2 Security

3.4.3 Maintainability

3.5 Design constraints

3.5.1 Standards Compliance

3.5.2 Hardware Limitations

**Appendix A: Issues List**

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
|  |  |  |  |
|  |  |  |  |

# **Introduction**

## **Purpose and Scope**

*This document specifies the design of Super Tic Tac Toe 1.0 ("the project"), Team 5 ("PlasmaJET")'s Spring 2015 CMSC 4283/5283 semester project. The System will efficiently perform all requirements found in the project description for Dr. Jicheng Fu's Spring 2015 CMSC 4283/5283 Software Engineering class. The basic of these is a system that creates connection to another team's system and perform within protocol a game of Tic Tac Toe with Achi tiebreaker.*

## **Definitions, Acronyms and Abbreviations**

*<Describe definitions of all the terms, acronyms and abbreviations used in the document. Special attention should be paid to the clarification of terms and concepts from the domain of application.>*

## **References**

* http://www.math.cornell.edu/~mec/2003-2004/graphtheory/tictactoe/howtoplayttt.html
* Dr. Jicheng Fu's Spring 2015 CMSC 4283/5283 Project Description, <https://github.com/loganhuskins/plasmaJET/blob/master/Project%2BDescription.docx>
* Super Tic Tac Toe protocol documentation (as yet unreleased)
* PlasmaJET Super Tic Tac Toe Use Case Diagrams (under development)
* PlasmaJET Super Tic Tac Toe Sequence Diagrams (under development)
* PlasmaJET Super Tic Tac Toe Class Diagrams
* Google Java Style Guide, <https://google-styleguide.googlecode.com/svn/trunk/javaguide.html>
* usability.gov User Interface Design Basics <http://www.usability.gov/what-and-why/user-interface-design.html>
* MinMax Algorithm for minimizing loss in tic-tac-toe <http://en.wikipedia.org/wiki/Minimax>

## **Overview**

The remainder of this document is organized as follows:

* Section 2 contains an overall description of the Super Tic Tac Toe system which includes product functions, user classes and characteristics, constraints, assumptions, and dependencies.
* Section 3 is comprised of the specific requirements of the Super Tic Tac Toe system including external interface requirements, functional requirements, performance requirements, software system attributes, and design constraints
* Appendix A consists of a list of the open requirements issues associated with the project that are unresolved.

# **Overall Description**

## **Product Functions**

This project provides a Super Tic Tac Toe client that enables an operator to identify another client ("opponent") to connect to and initiate an AI-controlled Super Tic Tac Toe game with. The client displays its current status to the operator as well as the state of the game as it progresses. Once connected, the client handles all protocol interactions with the opposing client until the completion of the round in accordance with Dr. Fu's project description and protocol documentation. The project will also include a player versus client mode in which a local user can play the game against an AI-controlled opponent.

## **User Classes and Characteristics**

The primary user class of the client is the student operator who is a member of Team PlasmaJET. Supervisory faculty members, students from other teams, and other interested parties may also occasionally use the client. By design, the user interactivity with the client is minimal and each of the mentioned individuals will use the client in the exact same way. For design purposes a single user class is identified that effectively meets the needs of all anticipated users.

## **Constraints**

Team PlasmaJET will design, develop, test, deploy, operate, and distribute the client in strict accordance with course policies, academic integrity guidelines, community decency standards, and applicable local and federal laws. During normal operation the client may traverse a game state space of upto 9! (362,880) boards in the ordinary 3x3 case and a few more in the tie-breaking Achi case. Modern personal and mobile computer hardware will adequately support the client's performance of all anticipated computations in a timely fashion. Specific timing requirements or more complex gameplay rules may later be introduced, and Team PlasmaJET will meet or exceed all requirements. The client will additionally require hardware and operating environment support to interoperate with opponents via the to-be-determined interconnection protocol. Team PlasmaJET will maintain design documents, source code, and validation protocols and procedures so as to enable responsiveness to changing requirements.

## **Assumptions and Dependencies**

It is assumed that all external components will operate according to their documented purposes and specifications. The Super Tic Tac Toe depends on a stable internet conection and it assumes that the opposing team's system will also comply with the chosen protocol decided by the class. It also assumes that an escalation path exists for any failures which inhibit Team PlasmaJET's ability to capitalize on their utility for the purpose of implementing the client.

# **Specific requirements**

## **External Interface Requirements**

### **User Interfaces**

User interfaces are required to define the opponent connection parameters and display the status and progress of the running game. Specific details of the connection parameters are determined by the chosen protocol. The user interface for connection parameters will enable the specification of connection parameter information at an appropriate level of detail to the underlying protocol (e.g. TCP/IP), for example, an input box labeled "Peer IP address: " and a "Connect" or "Exit" button. The client status will indicate to the user whether the client is negotiating a connection with an opponent, determining its next move, communicating its move to the opponent, waiting for the opponent to issue a move, receiving a move from the opponent, or negotiating a tie-breaking round. The game progress will dynamically update to reflect the current state of the board after each move is registered by the client. A graphical client will display the board graphically, with bitmaps or rendered graphics. A terminal client will display the board using standard printable ASCII characters arranged in visually descriptive ways. In the case of a graphical client, when the user is playing a local game versus an AI, the user will be able to click on individual quadrants to select their move. In the case of the graphical environment, the user will be asked to type their select quadrant (e.g. A1 for the top left coordinate). Screen shots will be included as the product reaches further stages of development.

### **Hardware Interfaces**

The client software runs in an operating environment that exposes hardware capabilities required by the client in environment standard ways. For example, a desktop or laptop PC running the Microsoft Windows operating system enables a user operating a keyboard, mouse, or touch-sensitive tablet to communicate with the client by passing input events to the program's message loop. The client can graphically feedback to the user with API calls that enable the construction and management of graphical display objects such as windows and dialog boxes that show text, buttons, and graphics. The client requires the operating environment afford the user a way to input opponent connection information (such as a keyboard with numbers and a period to input an IP address, or a virtual keyboard presented on a mobile phone enabling same), and a way for the program to communicate its status and present its game state back to the user, via a graphical display buffer or a textual console buffer.

### **Software Interfaces**

The Super Tic Tac Toe system will be a Java based desktop application that will connect to all competing systems through protocols proposed by the Software Engineering 4283/5283 protocol committee. The current proposal is for connections to be made through a known good server.

### **Communications Interfaces**

Standard games of User versus AI will be played through a browser server model where the user communicates with the application through a web browser running locally on their machine and the AI is running off of a server hosted at UCO, which will communicate over Hyper Text Transfer Protocol (HTTP). Games of AI versus AI will use a peer-to-peer protocol specified by the group leaders during their first meeting and the document will be updated to reflect any decisions made.

## **Functional requirements**

### **Create New Game**

#### ***Introduction***

This function allows the system to create a new game when prompted by the user and to begin to process moves in said game. This function will be necessary for all other functions in the system and the system’s use relies on proper execution of this function.

#### ***Inputs***

The system will accept input from the user in the form of the user selecting the creation of a new game.

#### ***Processing***

No formal processing is required in the creation of a new game.

The system will begin by selecting whether it or the opponent begins that game with the first move. It does so by seeing if the system has played this opposing system before, and if it has it will give the first move to the system who made the second move last game. If the systems have not previously interacted the system randomly assigns and makes note of the first move made. If the systems have interacted before, the user’s system will calculate the win-loss record versus this opponent

#### ***Outputs***

The system will output a field to enter the IP address of the system the user’s system would like to interact with.

### **Create Connection**

#### ***Introduction***

After prompting for a new game, the system prompts the user to enter the IP address of the system that it will be interacting with. By entering this, the user is both agreeing to connect to that IP address for a game of tic tac toe, but also agreeing to begin keeping a win-loss record and a record of which system made the first move in previous games. After creating the connection, the game will begin.

#### ***Inputs***

The input will be the IP address of the system that the user wishes to interact with.

#### ***Processing***

The system will attempt to establish a connection with the specified system after the IP address is entered. When this happens, the system will look at a log to see if it has interacted with this IP before. If it has, the system will display the win-loss record for the systems and choose who makes the first move based on previous interactions. Specifically, the system that moved first last time would move second this time. If the systems have not interacted before, the user’s system will begin a log of this IP to be referenced in the future.

#### ***Outputs***

Assuming a valid connection, the system will output that the connection was successful. Following this, the system will output the opposing system’s win-loss record versus the user’s system and who gets to make the first move. If the two systems have no history, the user’s system will inform the user that it is creating a log for the IP. After this is complete, the system will display the game board and begin the game.

### **Make Move**

#### ***Introduction***

For the game to progress, the system will be required to calculate and make moves. This action makes for the bulk of the system’s work and includes the act of calculating the move that minimizes the chance of loss against the opponent, sending the move over the network to the opposing client, and checking if that move won the game for the system.

#### ***Inputs***

No input from the user is required in this phase. During this phase a disconnect button will be presented. When the button is clicked, a stop game command is sent to the opposing client.

#### ***Processing***

To calculate the move, the system will use a variation on the Min-Max Algorithm with the intent of minimizing the possibility of loss. Due to the finite nature of tic-tac-toe, this will guarantee the impossibility of loss against the opposing system.

Before sending the move to the opponent over the network, the system will send a small packet to validate the connection. Following this, the system will encrypt the move using a public key system to inhibit tampering. The opposing client will then send a packet back specifying that the data arrived unhampered. If it does not, the opposing system will send a message saying that the data did not arrive and will send a message specifying what went wrong.

If the move puts the user’s system in a win state, the syste will update the win-loss total versus the opposing system.

#### ***Outputs***

The system will display a brief message stating the move was successfully sent to the opposing client to verify that the package arrived, assuming it did arrive. The system will them display the move on the user’s system screen. If a win state is achieved, the system will display a message as well as the new win-loss total against the opponent.

### **Receive Move**

#### ***Introduction***

During the course of the connection, the system will be expected to rcive moves form the opposing player’s system. These moves must be accepted over the network successfully, checked for purity and lack of tampering, validated, and displayed on the user’s system.

#### ***Inputs***

There will be no required inputs form the user during this step. However, a button will be displayed allowing the user to close the connection and terminate the game. The system will receive input over the network from an opponent’s system with their move selection.

#### ***Processing***

Using a public key system, the system will verify that the move sent over the network did come from the opponent’s system to ensure that there is no outside tampering. The system will be required to validate the move once it is received. To do this, it will check a list of possible moves given what spaces on the board are occupied and compare the opponent’s move to this list.

After the move is validated, the system will check if the move put the opponent in a win condition. If it has, it will display a message and update the win-loss record versus the opponent.

#### ***Outputs***

Once the move is validated, the system will display the move on the game board. If the move is either not validated because of a network issue, tampering, or because an illegal move was made, the system will display an appropriate error on the screen.

If a win state is achieved, the system will display a message as well as the updated win-loss total against the opponent.

## **Performance Requirements**

When deciding moves on the game board, both the user and the client will have a set time to make a decision before they are forced forfeit. While the client is required to meet the same time standards as the user, due to the small and finite number of moves the system is able to make, the system will be expected to make a decision much faster. It will generally be expected that the system makes any decisions in <1 second, or on a pace that is unnoticeable by the user while playing on a 3x3 board. When determining a winner or a draw, the system is expected to make an equally quick determination. The performance requirements in an AI versus AI game will depend largely on the networking between the two clients. In this situation, the system will be expected to make decisions and determine the winner as quickly as in a player versus AI from when the data reaches the system. It will send the moves out in an additional time-frame of about 1 second, or a time unnoticeable to the observers.

## **Software System Attributes**

*<Particular attention is paid to quality aspects. These requirements must be measurable and verifiable. They must be stated in objective terms.>*

### **Availability**

The desktop client will be available for player versus AI games as long as the hardware the system is running on is available. FOr any demonstration, the system will be running on one of plasmaJET's personal computers. Between the team members, we can promise 95% uptime. With respect to AI vs AI, we can promise 95% uptime for our application, but can make promises about uptime outside of our specific system due to uncontrollable variables.

### **Security**

The system will be able to check and prevent a user from performing illegal moves and adjust accordingly for player versus AI games. In AI versus AI games, any outgoing communications will be encrypted using an as-to-be determined public key system. In these games the system will also look for external systems attempting to cheat or alter any game elements.

### **Maintainability**

The system will have well produced documentation using Java's built-in documentation feature that will come alongside any version of the application. This documentation, along with designing alongside typical Object Oriented patterns and standards, will guarantee maintainability for any future work.

## **Design constraints**

### **Standards Compliance**

Several standards govern different aspects of the project. This design document itself and the design it details adhere to software engineering best practices as described by Dr. Jicheng Fu in course lectures, on Dr. Fu's slide decks and presentation notes, and in the course text, \_Object-Oriented Software Engineering using UML, Patterns, and Java (3rd Edition)\_. This project's use case diagrams adhere to the UML 2.0 standard. Work done in conjunction with the project is fully compliant with UCO academic integrity standards. Program code written for the project primarily adheres to Google Java Style guide as of March 21, 2014, favoring readability and consistency of style everywhere possible. Additional guidelines determined by team consensus regarding to documentation, testing, and program design patterns are also upheld, with minimal deviations from accepted conventions admitted where noted with an explanation as to why the best practice could not be followed and how the exception better supports the fulfillment of the component's role in meeting the design goal. The user interface and experience conforms to Usability.gov User Interface Design Basics.

### **Hardware Limitations**

The project is implemented in Java and will theoretically run anywhere a JVM is available. Centain hardware capabilities must be available to the JVM in order to meet specific design requirements. The user interface requires requires a hardware graphical display capable of rendering bitmap graphics as outlined in the interface design section of this document, as well as hardware input devices enabling the user to participate in the use cases outlined above, including choosing a game mode (AI-vs-AI or player-vs-AI), entering an opponent network address, indicating a move during gameplay, and acknowledging the result of the game at its conclusion. These user actions are facilitated by some combination of input devices such as a hardware keyboard, hardware mouse, virtual keyboard operated by hardware mouse, or hardware touchscreen as pointer enabling operation of a virtual keyboard -- it is these which will be successfully handled by the client.

In order for the client to successfully participate in AI-vs-AI mode, a TCP/IP network stack and supporting hardware must also be available to the JVM that is running the application code.

The computer tic-tac-toe algorithm is either CPU-bound or memory-bound and additional hardware architectural constraints must be met to ensure the game is capable of meeting the per-move time limit. Possible decision algorithm implementations include iterating through all possible game states each move, which is CPU-bound, or building a decision tree before the game starts and walking/pruning as the game progresses, which is memory-bound. Specific CPU and memory requirements satisfying these constraints will be determined during development.

**Appendix A: Issues List**

* The class has yet to decide on or communicate any decision on the network protocol that the systems will be using to connect to one another.
* A decision has yet to be made about what public-key protocol the system’s will use to communicate and validate moves.