**Product Specifications**

Table of Contents:

[Modification History 3](#_Toc405483247)

[Major Milestones 4](#_Toc405483248)

[Abstract 5](#_Toc405483249)

[Document References 6](#_Toc405483250)

[Glossary 7](#_Toc405483251)

[Class Diagram 8](#_Toc405483252)

[Use Case Sequence Diagram 12](#_Toc405483253)

[Use case 1 – Start new game 12](#_Toc405483254)

[Use case 2 – Pick difficulty 13](#_Toc405483255)

[Use case 3 – Login 14](#_Toc405483256)

[Use case 4 – Register 15](#_Toc405483257)

[Use case 5 – Play as guest 16](#_Toc405483258)

[Use case 6 – Complete game 17](#_Toc405483259)

[Use case 7 – Reset 18](#_Toc405483260)

[Use case 8 – Who is first to act? 19](#_Toc405483261)

[Use case 9a – Place a stone (empty cell; no win) 20](#_Toc405483262)

[Use case 9b – Place a stone (empty cell; win) 21](#_Toc405483263)

[Use case 9c – Place a stone (occupied cell) 22](#_Toc405483264)

[Use case 10 – Display user history 23](#_Toc405483265)

[Use case 11 – Update player history 24](#_Toc405483266)

[User Interface 25](#_Toc405483267)

[Others 27](#_Toc405483268)

# Modification History

9/16/2014: Initial draft.

9/22/2014: Added UI mockup

10/14/2014: Added UML Classes

11/2/2014: Added Use Case Sequence Diagrams

11/3/2014: Refined Use Case Sequence Diagrams

# Major Milestones

|  |  |
| --- | --- |
| **Date** | **Milestone** |
| 9/16/2014 | Complete preliminary specifications |
| 10/7/2014 | Refine specifications |
| 10/21/2014 | Draft use case sequence diagrams |
| 11/3/2014 | Complete use case sequence diagrams |
|  |  |
|  |  |
|  |  |

# Abstract

Design a multi-platform tic-tac-toe game played on a 5x5 grid that requires 4 stones in-a-row to win. Allow players to register or play as a guest. Keep track of registered player history (i.e.: wins vs losses). Game should allow player to select a difficulty when playing against the computer.

# Document References

|  |  |
| --- | --- |
| **Document** | **Location** |
| Requirements.doc | Google docs |
| SoftwareManagementPlan.doc | Google docs |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Glossary

|  |  |  |  |
| --- | --- | --- | --- |
| **Term** | | **Definition** | |
| Blue | Player 1 |
| Red | Player 2/AI |
| Grid | The playing board |
| Stone | Player Piece |
| Guest | Unregistered Player |
| Player | Registered Player |
| Leaderboard | List of top N players |
|  | |  | |

|  |  |
| --- | --- |
| **Acronym** | **Meaning** |
| PVP | Player-vs-Player |
| AI | Artificial Intelligence (the computer) |
| TTT | Tic-Tac-Toe |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Class Diagram

|  |
| --- |
| Player Class |
| - username : string  - objectID : string  - session : string  - wins : int  - losses : int  - draws : int |
| + getWins() : int  + setWins(value : int) : void  + getLosses() : int  + setLosses(value : int) : void  + getDraws() : int  + setDraws(value : int) : void  + getUsername() : string  + setUsername(username : string) : void  + loginPlayer (username : string) : void  + registerPlayer(username : string) : void  + updatePlayer(win : int, losses : int, draws : int) : void  + logout() : void  + deletePlayer() : void |

|  |
| --- |
| AI Class |
| - difficulty : int |
| + makeMove() : int  +getDifficulty(): int  +setDifficulty(value : int): void  - randomMove() : int  - mediumAi() : int  - smartAI() : int |

|  |
| --- |
| Grid Class |
| + \_grid : QList< QList <int> >  + \_lastMove : int  + \_gridLength : int  + \_countToWin : int  - \_lastMoveX : int  - \_lastMoveO : int  - \_moveCount : int  - \_firstPlayer : int |
| + getGrid() : QList< QList <int> >  + getGridLength() : int  + setGridLength(length : int) : void  + getLastMove() : int  + setLastMove(player : int) : void  + getCountToWin() : int  + setCountToWin(count : int) : void  + getMoveCount() : int  + setMoveCount(moveCount : int) : void  + getFirstPlayer() : int  + setFirstPlayer(player : int) : int  + placePiece(index : int, player : int) : void  + checkWin(player : int) : bool  + indexToRow(index : int) : int  + indexToColumn(index : int) : int  + coordinateToIndex(row : int, column : int) : int  + valueFromIndex(index : int) : int  + isFilled() : bool  + boardReset() : void  + lastMoveX() : int  + setLastMoveX(lastMoveX : int) : void  + lastMove0() : int  + setLastMove0(lastMove0 : int) : void |

|  |
| --- |
| Parse Helper Class |
| - netManager : QNetworkAccessManager  - request : QNetworkRequest |
| - setContentLength(size : int) : void  - setSession(session : string) : void  + login(username : string) : void  + registerPlayer(username : string) : void  + updatePlayer(id : string, session : string, wins : int, losses : int, draws : int) : void  + deletePlayer(id : string, session : string) |

|  |
| --- |
| Main Class |
| - turn : int  - blue : color  - red : color  - isVisible : bool  - board : grid  + players : QList < Player\* > |
| + createOpponent(difficulty : int) : void  + lastMoveCheck() : void  + tileMouseArea.onClicked() : void  + tileMouseArea.onEntered() : void  + tileMouseArea.onExited() : void |

|  |
| --- |
| Line Class |
| - \_startingPos : int  - \_value : int  - \_length : int  - \_pieces : int  - \_dir : Direction |
| + startingPos() : int  + value() : int  + length() : int  + pieces() : int  + direction() : Direction  + initialize(pos : int, d : Direction, v : int, l : int, pc : int) : void  + addPiece(piece : int) : void  + isFull() : bool  + directionToString() : string  + magnitude() : int  + operator =(&line : Line) : void  + operator ==(&line : Line) : bool  + operator >(&line : Line) : bool  + operator <(&line : Line) : bool |

|  |
| --- |
| Main Menu Class |
| + divisor : int |
| + backButton() : void  + historyButton() : void  + newGameButton() : void  + restartButton() : void  + settingsButton() : void  + logButton() : void |

|  |
| --- |
| Settings View Class |
| - background : image |
| + setBGImage(bg : image) : void  + getBGImage() : image |

# Use Case Sequence Diagram

## Use case 1 – Start new game

### General Description

When the user starts the application, the start page is displayed.

NOTE: The start page presents the user the option to login to play as an existing player, register to create a new player, or play as guest.

### Sequence Diagram



## Use case 2 – Pick difficulty

### General Description

After the user selects single-player mode, he is prompted to select the computer’s difficulty setting.

### Sequence Diagram



## Use case 3 – Login

### General Description

The user provides his username and clicks the “Login” button to play as a registered player.

### Sequence Diagram



## Use case 4 – Register

### General Description

The user provides his username and clicks the “Register” button. The application then validates the username, and if it is a valid username, it saves it to the database and automatically logs-in the user.

### Sequence Diagram



## Use case 5 – Play as guest

### General Description

The user clicks the “Play as Guest” button.

### Sequence Diagram



## Use case 6 – Complete game

### General Description

When the user makes a finishing move, the application updates his win/loss/draw record, clears the grid, and prompts for rematch/new-game.

### Sequence Diagram



## Use case 7 – Reset

### General Description

The user clicks the menu button (≡) then clicks “Reset.” The application then clears the grid and prompts the user “who is first to act?”

### Sequence Diagram



## Use case 8 – Who is first to act?

### General Description

When the user starts a new game, resets a current game, or chooses to rematch a completed game, he is prompted to select which player will be first to act. After making his selection, the grid will be displayed and the game will begin.

### Sequence Diagram



## Use case 9a – Place a stone (empty cell; no win)

### General Description

When the user clicks an empty cell on the grid, the application places a stone with the color that corresponds to the player whose turn it is. If this is not a winning-move, the application switches players, and waits for the next move.

### Sequence Diagram



## Use case 9b – Place a stone (empty cell; win)

### General Description

When the user clicks an empty cell on the grid, the application places a stone with the color that corresponds to the player whose turn it is. If this is a winning-move, the application ends the game.

### Sequence Diagram



## Use case 9c – Place a stone (occupied cell)

### General Description

When the user clicks an occupied cell on the grid, the application does nothing and waits for the next move.

### Sequence Diagram



## Use case 10 – Display user history

### General Description

The user clicks the menu button (≡) then clicks “History.” The application closes the menu and displays the Player History view.

### Sequence Diagram



## Use case 11 – Update player history

### General Description

When a round is over, the player’s win/loss/draw history is updated.

### Sequence Diagram



# User Interface

Responsive 5x5 grid.

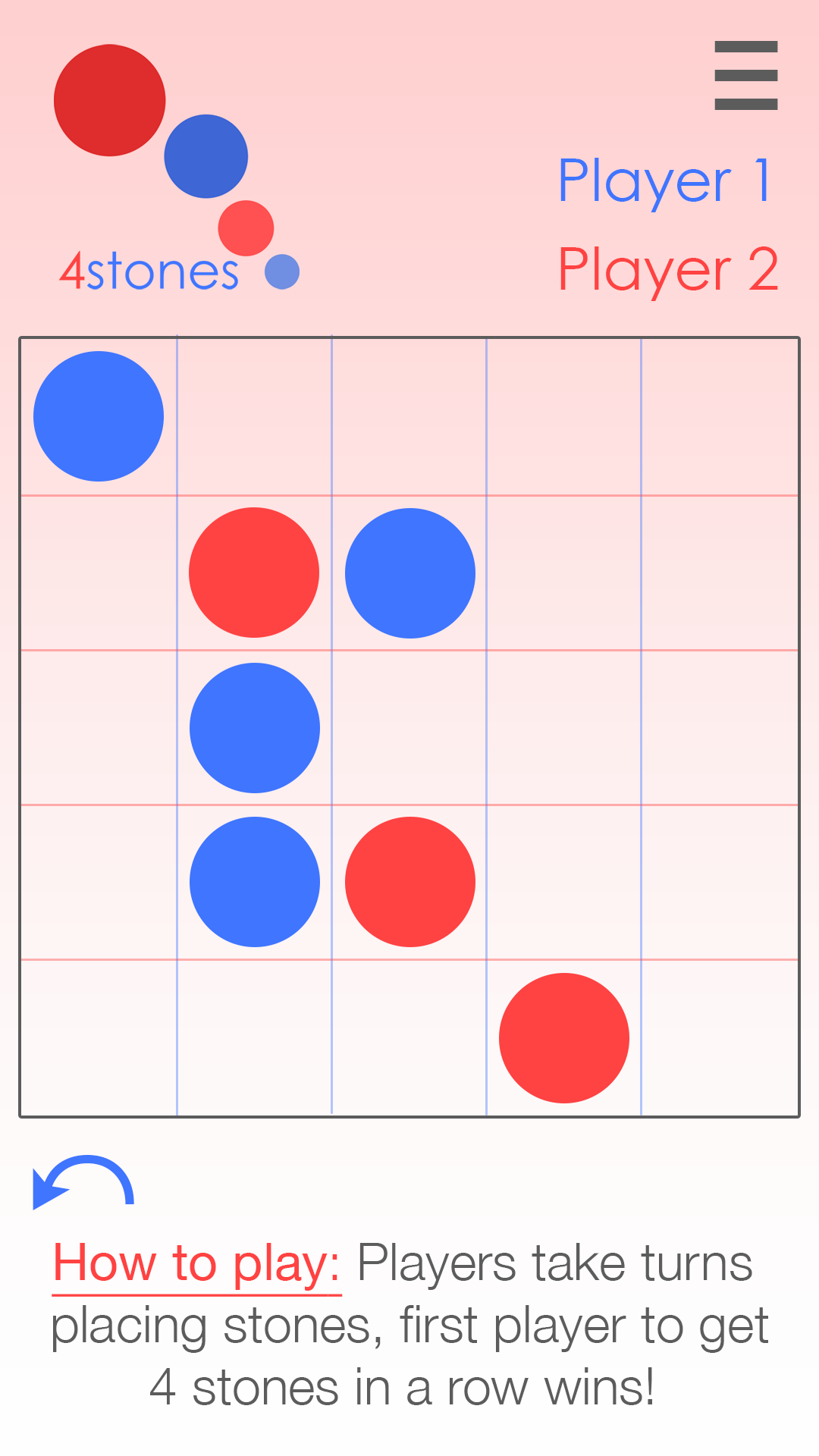
Stones are blue and grid.

Main Menu:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tic-Tac-Toe | | | | |
|  |  |  |  |  |
|  | Play with: | | |  |
|  | A friend |  | The Computer |  |
|  |  |  |  |  |

Main Menu: Play the computer

|  |  |  |
| --- | --- | --- |
| Tic-Tac-Toe | | |
|  |  |  |
|  | Choose a difficulty. |  |
|  | Easy |  |
|  |  |  |
|  | Medium |  |
|  |  |  |
|  | Hard |  |
|  |  |  |



# Others

