KingMe

Software Design Document

|  |  |
| --- | --- |
| **Group Members** | Chris Deslongchamp, Vincent Finn, William Brown, Matthew Quaschnick |
| **Faculty Advisor** | Dr. Filippos Vokolos, Ph. D. |
| **Project Stakeholder** | Dr. Filippos Vokolos, Ph. D. |

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason for Change** | **Version** |
| Vincent Finn | 8/7/2018 | First Draft – Sections Outlined and assigned | 0.9 |
| Vincent Finn | 8/10/2018 | Section 4 filled in | 0.9.1 |
| William Brown, Chris Deslongchamp, Matthew Quaschnick | 8/11/2018 | Sections 1,2,3, and 5 filled in | 1.0 |
| William Brown, Chris Deslongchamp, Matthew Quaschnick, Vincent Finn | 8/12/2018 | Reviewed. Comments addressed and changes made based on comments. | 1.1 |

Table of Contents

[Revision History 2](#_Toc521871832)

[1. Introduction 3](#_Toc521871833)

[1.1 Purpose of Document 3](#_Toc521871834)

[1.2 Scope of Document 3](#_Toc521871835)

[2. System Overview 3](#_Toc521871836)

[2.1 Description of Software 3](#_Toc521871837)

[2. 2 Technologies Used 3](#_Toc521871838)

[3. System Architecture 4](#_Toc521871839)

[3.1 Architectural Design Components 4](#_Toc521871840)

[3.2 Design Rationale 5](#_Toc521871841)

[4. Component Design 6](#_Toc521871842)

[4.1 Overview 6](#_Toc521871843)

[4.2 Server Program 6](#_Toc521871844)

[4.2.1 Server.cs 6](#_Toc521871845)

[4.2.4 ServerCheckersGame.cs 11](#_Toc521871846)

[4.3 Client Program 16](#_Toc521871847)

[4.3.1 Client.cs 16](#_Toc521871848)

[4.3.4 CheckersGameForm.cs 20](#_Toc521871849)

[4.4 Universal / Shared Classes 23](#_Toc521871850)

[4.4.1 Enums 23](#_Toc521871851)

[4.4.2 GameBoard 24](#_Toc521871852)

[4.4.5 PlayerMove 27](#_Toc521871853)

[4.4.8 CKPoint 29](#_Toc521871854)

# Introduction

## Purpose of Document

This document is to describe the implementation of King Me software as described in the King Me Requirement document. King Me is an online two-player checkers game using professional checkers tournament rules.

## Scope of Document

This document describes the implantation details of the King Me software. The software consists of several systems that will be split into two main namespaces, host and client. The host and client share many of the same functions but differ in how the initial connection is made. Code in the host is intended to host the game and the connection between the players. The client code is intended to join the game. Both the client and host are able to make moves and perform tasks in the game in the same way.

# System Overview

## 2.1 Description of Software

King Me is designed to be an online two-player checkers game utilizing professional tournament rules. Players will take turns making moves and jumps. The game will end once player has captured all the other player’s tokens or forces one player to make the same three moves resulting in a win, both players make the same three moves resulting in a tie, or one player surrenders.

## 2. 2 Technologies Used

King Me will use two PC (laptops or desktops) that utilize the Windows 10 operating system. Operating systems like Mac and Linux may be supported if time permits. One player will use their machine to host the game while the other uses their machine to join the game. Once both players have connected, they will make their moves on their own machine until the game ends.

# System Architecture

## 3.1 Architectural Design Components

**Server Application** – The user interacts with the Server application to allow the player to start the process of hosting a game. The player will be able to see their host name, so they can supply that to the player who will join.

**Server Class** – A class that handles all the socket communications and connections for hosting a game. It also has access to the ServerCheckersGame Class.

**ServerCheckersGame Class** – A class to keep track of all games currently in progress. When a player chooses to host a game (become the server) their game is added to this class and kept track of incase players disconnect.

**Client Application** – The user interacts with the Client application to allow the player to join a game that needs an opponent (host is the only player in the game). The player will be able to search for a specific host or choose a random opponent.

**Client Class** – A class that handles all the socket communications and connections for joining a game.

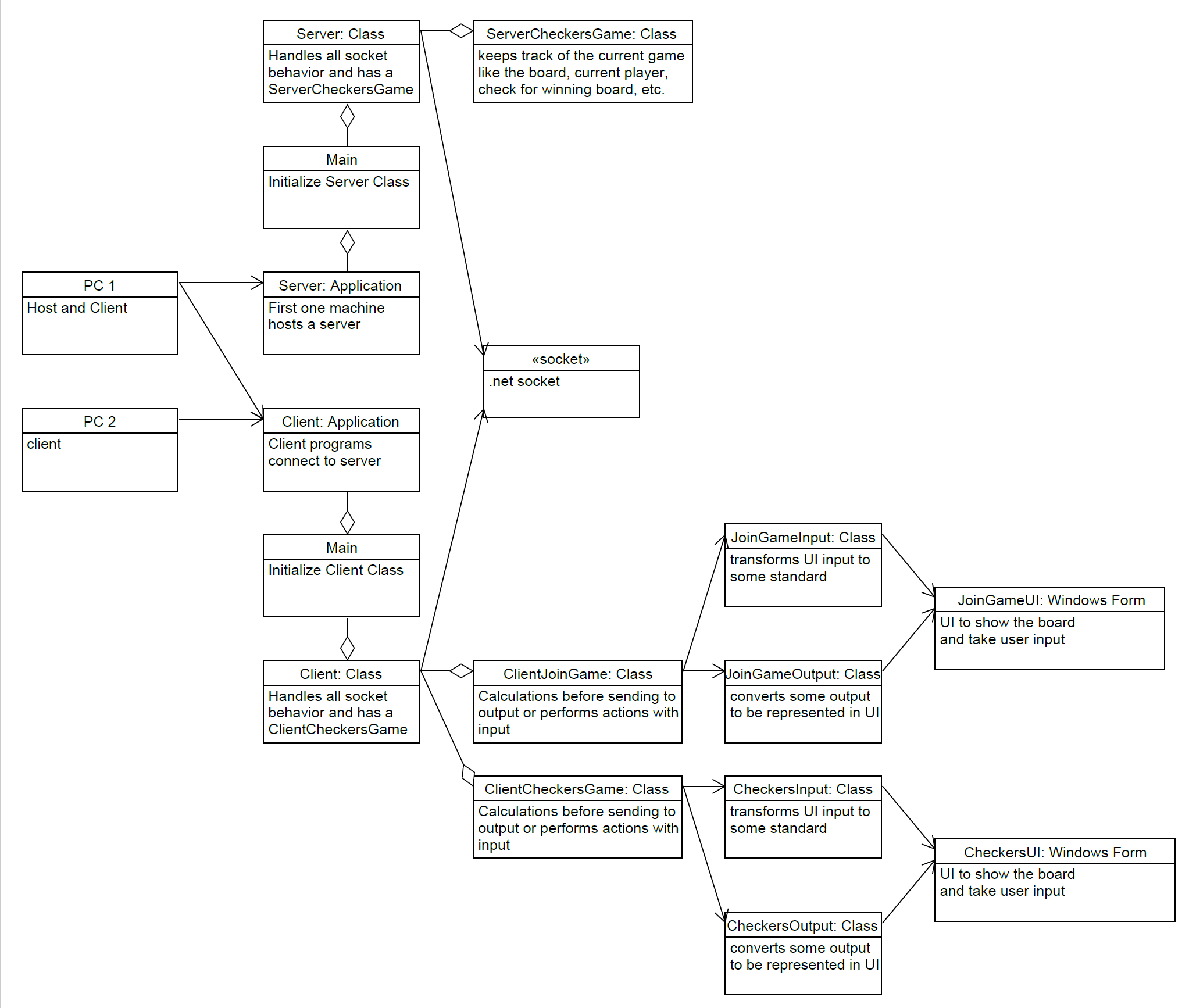
**CheckersGame Class** – A class that handles calculations such as move verification, jump verification, as well as board arrangement, timer, and pause functions. This class also checks if an end game state has been reached and calls on the GameOver class if one has been reached.

**Checkers Input Class** – A class that receives the opponents move/action, deciphers the input, and displays the board changes or action requested. Once the message is received it sends a verification to the opponent.

**Checkers Output Class** – A class that encodes the players move/action and sends it to the opponent. The class awaits a verification that the message has been received. If one is not received, it sends the message again until it is received.

**CheckersUI Windows Form** – Displays the checker board UI for the player.

**GameOver Class** – A class that checks the current board state if the game is over by either a win, tie, or surrender.



## 3.2 Design Rationale

**Why socket connections?**

The C# language has a Peer-to-Peer library that utilizes socket connections to transfer data between two machines. We can utilize the library to help make our data transfers and connections easy to set-up.

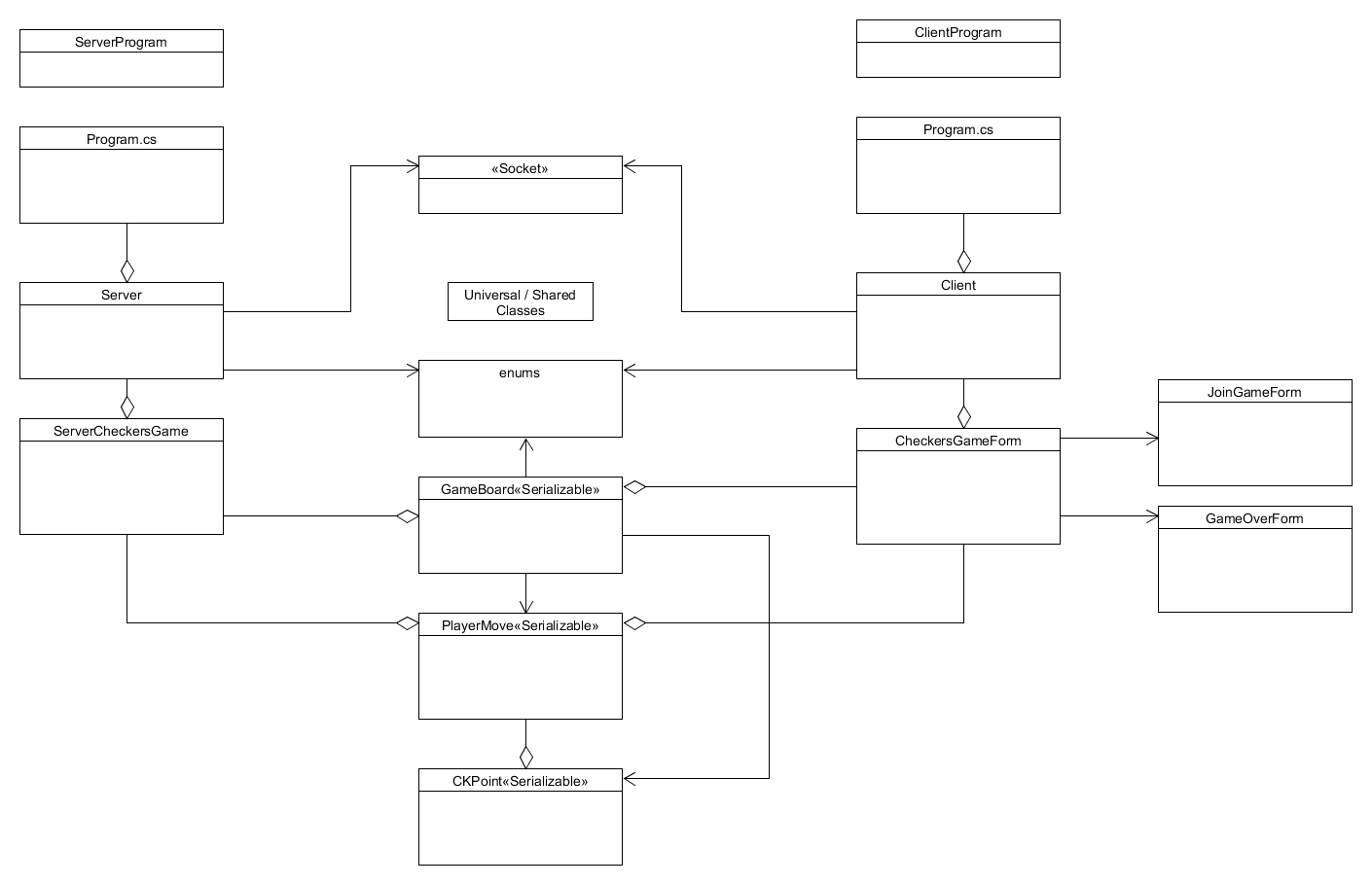
**Why encode moves and actions on the player’s machine and then decipher them on the opponent’s machine?**

We felt that encoding our messages will allow us to send shorter packages across our connections. This will also allow us to check what is being sent to the opponent and verify that the information is correct by having and established syntax for the messages.

# 4. Component Design

## 4.1 Overview

In this section, more details on each component are given. For each component, UML and a brief description are given.

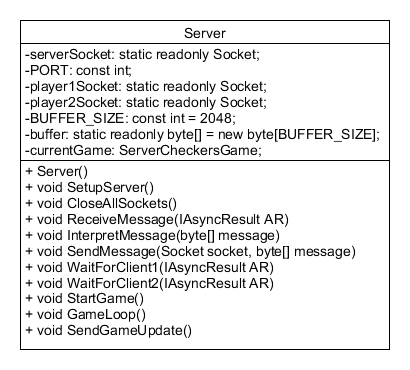


## 4.2 Server Program

The Server Program would be run on its own, wait for 2 players to connect, then start a loop to keep sending game updates to clients until the game is over.

## 4.2.1 Server.cs

The Server Class is responsible for handling all of the networking interactions using the ServerCheckersGame class to make decisions on what to do. It can receive a message from the client, either expected or unexpected, and needs to send a reply to an individual client with a message with instructions each client will interpret.



**4.2.2 Server Attributes**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| serverSocket | Static readonly Socket | Used as the server side socket that needs to opened so that clients can connect. |
| PORT | Const int | Port that server and clients connect on |
| player1Socket | Static readonly Socket | Used as the socket for the first client who joins. |
| Player2Socket | Static readonly Socket | Used as the socket for the second client who joins. |
| BUFFER\_SIZE | Const int = 2048 | Size of the buffer |
| buffer | Static readonly byte[] = new byte[BUFFER\_SIZE] | Byte array that is the size of BUFFER\_SIZE |
| currentGame | ServerCheckersGae | Used to keep track of the current state of the game, and do things like applyMove |
| player1Ready | Bool | Used to see if Player1 is ready to receive a message |
| player2Ready | Bool | Used to see if player2 is ready to receive a message |

**4.2.3 Server Methods**

|  |  |
| --- | --- |
| Server() |  |
| Input: | Void |
| Output: | Void |
| Description: | Constructor for the server class |

|  |  |
| --- | --- |
| Void SetupServer() |  |
| Input: | Void |
| Output: | Void |
| Description: | Sets up the server socket and opens it. Then calls WaitForClient1() |

|  |  |
| --- | --- |
| Void CloseAllSockets() |  |
| Input: | Void |
| Output: | Void |
| Description: | Simply Closes The Client Sockets, then the Server Socket gracefully. |

|  |  |
| --- | --- |
| Void ReceiveMessage(IAsyncResult AR) |  |
| Input: | Asynchrounous result needed to wait for a client message on a separate thread |
| Output: | Void |
| Description: | Used to Wait for messages from clients with Socketname.BeginReceive(buffer, 0, BUFFER\_SIZE, SocketFlags.None, ReceiveMessage, current); restarts itself after handling the message it received.The client needs to send an update letting the server know that it is ready to receive messages. |

|  |  |
| --- | --- |
| Void InterpretMessage(byte[] message) |  |
| Input: | A byte[] received from the client |
| Output: | Void |
| Description: | Reads the first byte and converts to an int and uses a switch statement and MessageIdentifierrs enum to decide what to do. Example if first byte is 2, then it would  Take the rest of the byte[], deserialize it and cast it as a PlayerMove, an try to apply it to the gameboard. |

|  |  |
| --- | --- |
| Void SendMessage(Socket socket, byte[] message) |  |
| Input: | The socket we are sending a message to, and the message in a byte[] |
| Output: | Void |
| Description: | Sends the given message to a given socket. It should only be able to send a message if that client is ready to receive. |

|  |  |
| --- | --- |
| Void WaitForClient1(IAsyncResult AR) |  |
| Input: | Asynchrounous result needed to wait for a client message on a separate thread |
| Output: | Void |
| Description: | Waits for the first Client to connect. When they do, it sets player1Socket to the current socket, waits to receive the first ready message from that client, then starts WaitForClient2 |

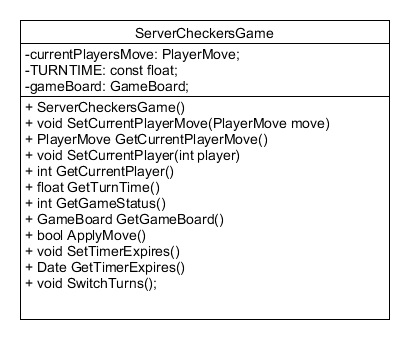
|  |  |
| --- | --- |
| Void WaitForClient2(IAsyncResult AR) |  |
| Input: | Asynchrounous result needed to wait for a client message on a separate thread |
| Output: | Void |
| Description: | Waits for the second Client to connect. When they do, it sets player2Socket to the current socket, waits to receive the first ready message from that client, then sends it a “StartingGame” message, and “Player2”, then it makes sure that client1 is ready, and sends it the same message with “Player1”, then calls StartGame() |

|  |  |
| --- | --- |
| Void GameLoop() |  |
| Input: | Void |
| Output: | Void |
| Description: | Handles most of decision making of the game. Continues to sendGameUpdates to the clients an receiving moves or pause requests until currentGame.GetGameStatus is anything than InProgress. Make sure to wait until both players are receiving to start the next turn. |

|  |  |
| --- | --- |
| SendGameUpdate() |  |
| Input: | Void |
| Output: | Void |
| Description: | Sends a GameUpdate message to both clients with a serialized Gameboard object |

## 4.2.4 ServerCheckersGame.cs

The ServerCheckersGame class contains the gameboard and keeps track of the current state of the game, and uses the gameboard for things like applying moves and checking for a winner or tie.



**4.2.5 ServerCheckersGame Attributes**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| currentPlayersMove | PlayerMove | Holds the Move the current player sent so that try to be applied to the board |
| TURNTIME | Const float | The amount of seconds a player should have to submit a move |
| gameBoard | GameBoard | The current gameboard object |

**4.2.6 ServerCheckersGame Methods**

|  |  |
| --- | --- |
| ServerCheckersGame() |  |
| Input: | Void |
| Output: | Void |
| Description: | Constructor for the ServerCheckersGame class |

|  |  |
| --- | --- |
| Void SetCurrentPlayerMove(PlayerMove move) |  |
| Input: | The playermove we want to set |
| Output: | Void |
| Description: | Set currentPlayersMove |

|  |  |
| --- | --- |
| PlayerMove GetCurrentPlayerMove() |  |
| Input: | Void |
| Output: | currentPlayersMove |
| Description: | Get currentPlayersMove |

|  |  |
| --- | --- |
| Void SetCurrentPlayer(int player) |  |
| Input: | 1 or 2 for player id |
| Output: | Void |
| Description: | Set currentPlayer in GameBaord |

|  |  |
| --- | --- |
| Int GetCurrentPlayer() |  |
| Input: | Void |
| Output: | 1 or 2 for player id |
| Description: | Get CurrentPlayer out of gameBoard |

|  |  |
| --- | --- |
| Float GetTurnTime() |  |
| Input: | Void |
| Output: | TURNTIME |
| Description: | ` Get TURNTIME |

|  |  |
| --- | --- |
| GameStatus GetGameStatus() |  |
| Input: |  |
| Output: | GameStatus from gameboard |
| Description: | Get GameStatus from gameboard |

|  |  |
| --- | --- |
| GameBoard GetGameBoard() |  |
| Input: |  |
| Output: | gameBoard |
| Description: | Get gameBoard |

|  |  |
| --- | --- |
| Bool ApplyMove() |  |
| Input: | Void |
| Output: | True if applyMove succeeded, false if the move was invalid |
| Description: | Try to apply the currentPlayersMove to update the gameboard. If the move is invalid, return false, otherwise return true. |

|  |  |
| --- | --- |
| Void SetTimerExpires() |  |
| Input: | Void |
| Output: | Void |
| Description: | Set the date of when the timer will expire |

|  |  |
| --- | --- |
| Date GetTimerExpires() |  |
| Input: | Void |
| Output: | timerExpires from gameboard |
| Description: | Get the Date the timerexpires from gameboard |

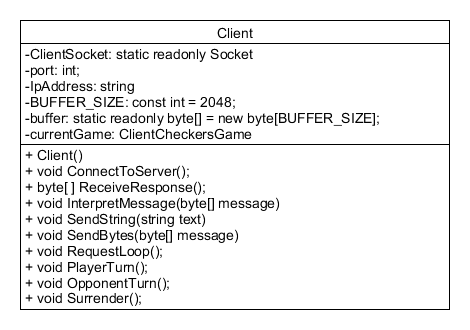
|  |  |
| --- | --- |
| Void SwitchTurns() |  |
| Input: | Void |
| Output: |  |
| Description: | Change the CurrentPlayer in Gameboard to the next player, start new timer |

## 4.3 Client Program

The Client Program directs which UI form is currently needed, connects to the server using the IP address and port given by the user, and keeps waiting until it is the clients turn and allows players to submit moves which get sent to the server.

## 4.3.1 Client.cs

The Client class handles most of the networking, and uses a currentGame object to help update the gameboard and get player moves when the client is told by the server that it is their turn.



**4.3.2 Client Attributes**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| ClientSocket | Static readonly Socket | Holds the current socket connection for this client |
| bort | Int | The Port value used to connect to a server |
| ipAddress | String | The ipAdress used to connect to a server |
| BUFFER\_SIZE | Const int = 2048 | The size of the buffer used to get messages |
| buffer | Static readonly byte[] = new byte[BUFFER\_SIZE] | Used to hold the data received from the server |
| currentGame | ClientCheckersGame | Used to do stuff regarding the checkers game |

**4.3.3 Client Methods**

|  |  |
| --- | --- |
| Client() |  |
| Input: | Void |
| Output: | Void |
| Description: | Constructor for the Client class |

|  |  |
| --- | --- |
| Void ConnectToServer() |  |
| Input: | Void |
| Output: | Void |
| Description: | Use the JoinGameForm to get the ipAddress and port, and try to connect to the server when the user clicks join game. |

|  |  |
| --- | --- |
| byte[] ReceiveResponse() |  |
| Input: | Void |
| Output: | Return the message as a byte[] |
| Description: | Wait for a response from the server and return it as a byte[] |

|  |  |
| --- | --- |
| Void InterpretMessage(byte[] message) |  |
| Input: | A byte[] received from the client |
| Output: | Void |
| Description: | Reads the first byte and converts to an int and uses a switch statement and MessageIdentifierrs enum to decide what to do. Example if first byte is 2, then it would  Take the rest of the byte[], deserialize it and cast it as a GameBoard, and Update the current gameboard, and if it is this clients turn wait for them to submit a move, or for the server to send a message that their time is up. |

|  |  |
| --- | --- |
| SendString(string text) |  |
| Input: | Text to send |
| Output: | Void |
| Description: | Convert the text to a byte[] and send it to the server |

|  |  |
| --- | --- |
| Void SendBytes(byte[] message) |  |
| Input: | A byte[] of something that was probably already serialized |
| Output: | Void |
| Description: | Send the given byte[] |

|  |  |
| --- | --- |
| Void RequestLoop() |  |
| Input: | Void |
| Output: | Void |
| Description: | Continously call ReceiveResponse() and do something based on the response received, until the gamestatus is anything other than InProgress |

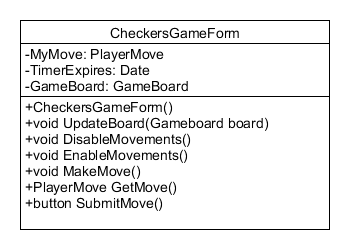
|  |  |
| --- | --- |
| Void PlayerTurn(); |  |
| Input: | Void |
| Output: | Void |
| Description: | It is currently this clients turn, wait for them to submit a move, then send it. Should be canceled if the server sends a game update or pause request. |

|  |  |
| --- | --- |
| Void OpponentTurn() |  |
| Input: | Void |
| Output: | Void |
| Description: | Disable the ability for the client to move pieces until it is their turn, wait for a gameupdate |

|  |  |
| --- | --- |
| Void Surrender() |  |
| Input: | Void |
| Output: | Void |
| Description: | Send a message to the server saying that you giveup |

## 4.3.4 CheckersGameForm.cs

The ClientcheckersGame class holds the gameboard object and other things needed to keep track of the current state of the checkers game on the client’s end



**4.3.5 CheckersGameForm Attributes**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| MyMove | PlayerMove | Holds the Move the current player needs to send so that try to be applied to the board |
| TimerExpires | Date | Time when the player will forfeit his turn |
| gameBoard | GameBoard | The current gameboard object |

**4.3.6 CheckersGameForm Methods**

|  |  |
| --- | --- |
| CheckersGameForm() |  |
| Input: | Void |
| Output: | Void |
| Description: | Constructor for the ClientCheckersGame class |

|  |  |
| --- | --- |
| Void UpdateBoard(GameBoard board) |  |
| Input: | Given a gameboard |
| Output: | Void |
| Description: | Set the gameBoard = board |

|  |  |
| --- | --- |
| void DisableMovements() |  |
| Input: | Void |
| Output: | Void |
| Description: | Stop the player from being able to submit a move |

|  |  |
| --- | --- |
| void EnableMovements() |  |
| Input: | Void |
| Output: | Void |
| Description: | Allow the player to submit moves again |

|  |  |
| --- | --- |
| void MakeMove() |  |
| Input: | Void |
| Output: | Void |
| Description: | Wait for the player to use the UI to make a move. Needs to be canceled if the server sends an update and it is no longer the players turn |

|  |  |
| --- | --- |
| PlayerMove GetMove() |  |
| Input: | Void |
| Output: | MyMove |
| Description: | Get the MyMove variable |

|  |  |
| --- | --- |
| button SubmitMove() |  |
| Input: | Void |
| Output: | MyMove |
| Description: | The user pushed the submit move button |

## 4.4 Universal / Shared Classes

The Client Program directs which UI form is currently needed, connects to the server using the IP address and port given by the user, and keeps waiting until it is the clients turn and allows players to submit moves which get sent to the server.

## 4.4.1 Enums

Three Enums used by Both Server and Client

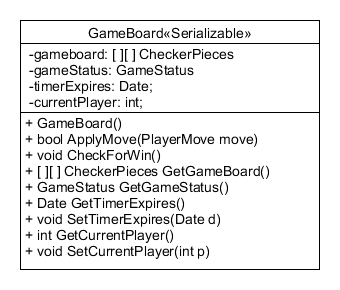
enum MessageIdentifiers { WaitingForOpponent, StartingGame, GameUpdate, RetryGameUpdate, GameOver, PauseRequest, Pausegame}

enum GameStatus {InProgress, Player1Wins, Player2Wins, Draw}

enum CheckersPieces {Red,RedKing, Black,BlackKing}

## 4.4.2 GameBoard

The GameBoard Object will be what we use to represent the checkers gameboard. It will have an array of arrays of CheckersPeices values which correspond to the rows and columns of a checkers board. It has methods to apply moves and check for a game ending status. It is also serializable, so that it can be transferred across the network.



**4.4.3 GameBoard Attributes**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| gameBoard | [ ] [ ] CheckerPieces | The data structure for the checkers board |
| gameStatus | GameStatus | Value of the current game status |
| timerExpires | Date | Date that the timer will expire |
| currentPlayer | Int | 1 is player 1, 2 is player 2 |

**4.4.4 GameBoard Methods**

|  |  |
| --- | --- |
| GameBoard() |  |
| Input: | Void |
| Output: | Void |
| Description: | Constructor for the GameBoard class |

|  |  |
| --- | --- |
| Bool ApplyMove(PlayerMove move) |  |
| Input: | A player move trying to be applied |
| Output: | True if the move was applied, false if it couldn’t be applied |
| Description: | Try to apply a move to the gameboard, and return if it was a success or not. |

|  |  |
| --- | --- |
| GameStatus CheckForWin() |  |
| Input: | Void |
| Output: | gameStatus |
| Description: | Analyze the board and return the status. |

|  |  |
| --- | --- |
| [ ] [ ] CheckerPeices GetGameBoard() |  |
| Input: | Void |
| Output: | The data structure for the board |
| Description: | Return the data structure for the board |

|  |  |
| --- | --- |
| GameStatus GetGameStatus() |  |
| Input: | Void |
| Output: | gameStatus |
| Description: | Get gameStatus |

|  |  |
| --- | --- |
| Date GetTimerExpires() |  |
| Input: | Void |
| Output: | timerExpires |
| Description: | Get timerExpires |

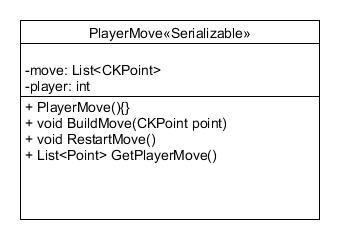
|  |  |
| --- | --- |
| Void SetTimerExpires(Date d) |  |
| Input: | Date d |
| Output: | Void |
| Description: | Set the timerExpires with the given date |

|  |  |
| --- | --- |
| Int GetCurrentPlayer() |  |
| Input: | Void |
| Output: | currentPlayer |
| Description: | Get currentPlayer |

|  |  |
| --- | --- |
| Void SetCurrentPlayer(int p) |  |
| Input: | Player Id |
| ` | Void |
| Description: | Set currentPlayer with the given int |

## 4.4.5 PlayerMove

The PlayerMove is the object for building a player move, from given CKPoints. It is serializable so the clients can send their moves to the server



**4.4.6 PlayerMove Attributes**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| Move | List<CKPoints> | The data structure for the playerMove |
| Player | Int | Value of the current player making the move |

**4.4.7 PlayerMove Methods**

|  |  |
| --- | --- |
| PlayerMove() |  |
| Input: | Void |
| Output: | Void |
| Description: | Constructor for the PlayerMove class |

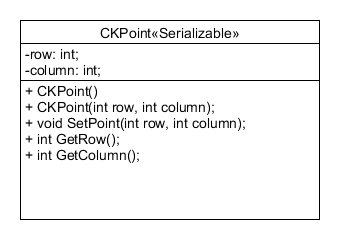
|  |  |
| --- | --- |
| Void BuildMove(CKPoint point) |  |
| Input: | A point |
| Output: | Void |
| Description: | Appends the point to move |

|  |  |
| --- | --- |
| Void RestartMove() |  |
| Input: |  |
| Output: | Void |
| Description: | Clears the list to start a fresh move |

|  |  |
| --- | --- |
| List<Point> GetPlayerMove() |  |
| Input: | Void |
| Output: | Move |
| Description: | Gets move |

## 4.4.8 CKPoint

A simple class to act as a point to easily access a part of the gameBoard.



**4.4.9 CKPoint Attributes**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| row | Int | The row index |
| column | Int | The column index |

**4.4.10 CKPoint Methods**

|  |  |
| --- | --- |
| CKPoint() |  |
| Input: | Void |
| Output: | Void |
| Description: | Constructor for the CKPoint class |

|  |  |
| --- | --- |
| CKPoint(int r, int c) |  |
| Input: | Row and column |
| Output: | Void |
| Description: | Constructor for the GameBoard class that immediately initializes the point |

|  |  |
| --- | --- |
| Void SetPoint(int r, int c) |  |
| Input: | Row and column |
| Output: | Void |
| Description: | Set point with given r and c |

|  |  |
| --- | --- |
| Int GetRow() |  |
| Input: | Void |
| Output: | Row |
| Description: | Get row |

|  |  |
| --- | --- |
| Int GetColumn() |  |
| Input: | Void |
| Output: | Column |
| Description: | Get column |

# 5. Human Interface Design

## 5.1 UI Overview

In this section, more details on UI design are given. Each subsection contains details, and some contain a diagram as well.

## 5.2 Screen Objects and Actions

**Server**

The intended input method for use on the server is a mouse. There are not many inputs on the server. After the game has been started via the server the rest of the game is played through the client.

**Client**

The intended input method for use on the client is a mouse. The mouse can be used to access various menus as well as move the checkers pieces.

## 5.3 Server Menu Flow

The server’s screen flow is very simple as it only consists of one screen with multiple buttons. A button to restart the game early, a button to end the game early and a button to disconnect a player.

## 5.4 Client Menu Flow

Below is a listing of the client’s screen flow for convince:

