CS 451: Software Engineering Term Project Design Document

Group Members

Phillip Ryan
Daniel Fitzick
Troy Santry
Karishma Changlani

Revision History

Name	Date	Reason for Change	Revision
Phillip Ryan Daniel Fitzick Troy Santry Karishma Changlani	07/18/16	Initial template	0.0
Phillip Ryan Daniel Fitzick Troy Santry Karishma Changlani	07/20/16	First Draft	1.0
Phillip Ryan Daniel Fitzick Troy Santry Karishma Changlani	07/24/16	Second Draft	1.1

Table of Contents

1. Introduction

1.1. Purpose of Document

This document is to describe the implementation of the Program as described in the CS451_Requirements document. The Program is designed to allow two players to play a game of checkers remotely from their computers.

1.2. Scope of Document

This document describes the implementation details of the game. It will contain various components between two namespaces: Client and Server. This requires to individuals to run the application at the same time. Either of which have the capability to be the Client or the Server. This document will describe the process of in each of the cases and how it affects the other namespace.

1.3. Definitions, Acronyms, Abbreviations

1.3.1 Game State

Board: Refers to the entire board of the game as a collection of pieces and squares.

Square: It's a position on the board that can store a piece of the Checker's Board.

Turn This refers to each move a player makes and how long he can make a move or required to.

Piece: Each checker's piece that is playable by the player or the enemy player.

Move: It's a decision made by the Player to move the piece to a certain position.

King piece: It's a special type of Piece that can move forward and backward.

1.3.2 Outside of Game Board:

Client: The player who is connection to the Game

Host: The player who is creating the game on a server.

GUI: Graphical User interface used by the players to interact with the application.

2. System Overview

2.1. Description of Software

The program is intended to be used by pairs of players. These players will play a game of checkers. The program will enforce the official checkers rules. Players will use their computers' standard input devices (mouse and keyboard) to select moves.

2.2. Technologies Used

The program is targeted for use on a computer using the Microsoft Windows operating system. The program is also dependant on Java Runtime 8. The two computers will connect directly to each other, without the need for an external server. These computers will communicate over the local network. The development environment is Eclipse. The version control service used is BitBucket.

3. System Architecture

3.1. Network Design Components

This system allows for a user to find a list of of discoverable available games, via UDP. And the ability to connect and communicate with one another via TCP. The main components of this architecture include the Discovery Protocol, Server Connection and Client Connection.

3.1.1. Discovery Protocol

The Discovery protocol defines a system where a UDP client acts as the user who wishes to connect to an existing game. UDP server acts as a user who creates a new game for other users to join

3.1.1.1. UDP Server

3.1.1.1.1. Creates a socket on a pre-defined port. Server then waits for a message from a client on that port, if there are any clients available waiting to play a game then each message received from a client will prompt for a response message saying that this a game is available

3.1.1.2. UDP Client

3.1.1.2.1. Creates a socket with a defined address and port number. Once the socket is created a hello message is sent to the server and once a response is received each incoming message is populated in a list of available connections.

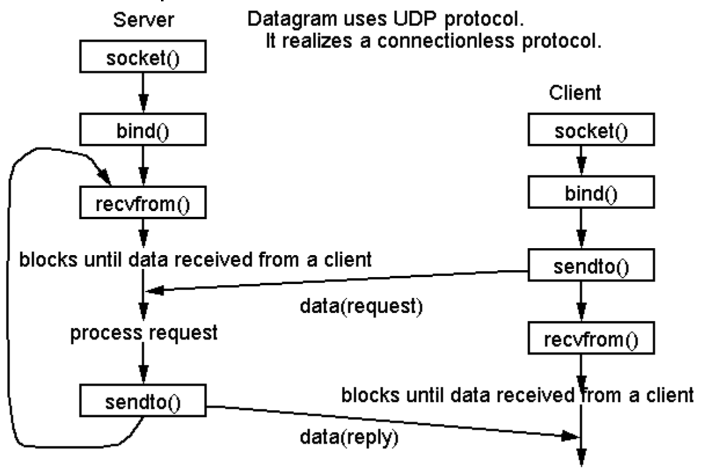
3.1.2. Server Connection

3.1.2.1. Once a UDP connection is established a TCP connection is created in order to allow for game data to be sent across this socket. Upon creation of a game a TCP socket is created based on the information shared between the UDP connection. The TCP socket will go into a listening state. When contacted by the client, the server creates a new TCP socket for server process to communicate with the client.

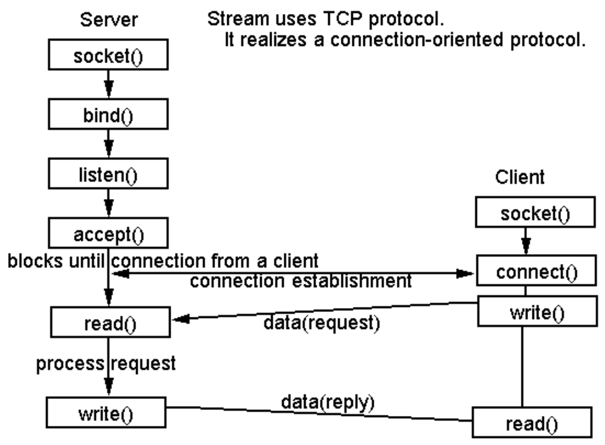
3.1.3. Client Connection

3.1.3.1. Once a UDP connection is established a TCP connection is created in order to allow for game data to be sent across this socket. The client connection will attempt to contact a server by creating a client-local TCP socket, specifying an IP address and port. When the client creates a socket the client TCP establishes a connection to the server TCP protocol.

3.1.4. UDP protocol


Server
Datagram uses UDP protocol.
It realizes a connectionless protocol.
socket()
bind()
recvfrom()
blocks until data received from a client
process request
sendto()
Client
socket()
bind()
sendto()
data(request)
recvfrom()
blocks until data received from a client
data(reply)

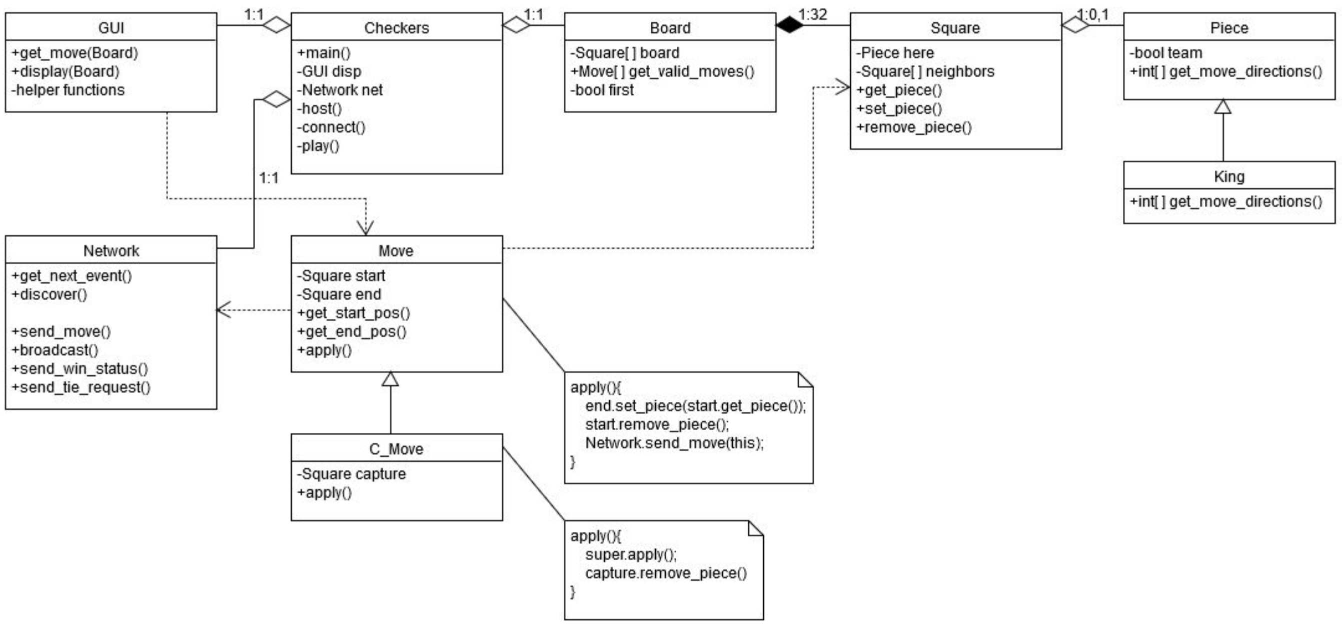
3.1.5. TCP protocol



3.2. Design Rationale

4. Component Design

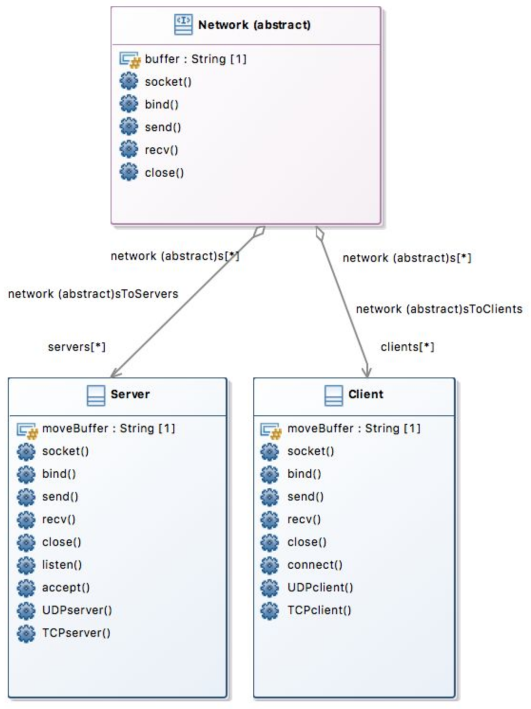
4.1. Overview



```
apply(){
    end.set_piece(start.get_piece());
    start.remove_piece();
    Network.send_move(this);
}
```

```
apply(){
    super.apply();
    capture.remove_piece()
}
```

4.2. Network Component



4.3. Game Component

4.3.1. Checkers Class

The class brings everything together and passes information between classes. It contains the main class and only does one think which is instantiating a Checkers object. All object instantiation and application setup processes are run from the Checkers constructor.

4.3.1.1. Attributes

Name	**Type**	**Description**
board	Board	The game board. This will contain information about all the spaces
net	Network	This object will be used for any communications with the other player and for initially finding a player to play with.
gui	GUI	Contains all the information about how to

		display the information to the user.
checkers	Checkers	An instance of the Checkers class.

4.3.1.2. Methods

public static void **main**(String[] args)	
Input:	Void
Output:	Void
Description:	Kicks off the program by instantiating a 'checkers' object.

public **Checkers**()	
Input:	Void
Output:	Void
Description:	Instantiates the necessary objects in order to play the game. This includes making instances of Board, GUI, Network, etc.

void **host**()	
Input:	Void
Output:	Void
Description:	Sets up the game that another player can join.

void **connect**()	
Input:	Void
Output:	Void

Description:	Sets up the application so that it can join another player's game.

void **play**()	
Input:	Void
Output:	Void
Description:	Starts a new game. Updates the screen through GUI obj. Makes sure board is set up for new game, etc.

4.3.2. Board Class

The class stores the location of all the piece and positions on the Board and calculates the moves that are valid.

4.3.2.1. Attributes

Name	**Type**	**Description**
first	bool	If the team about to go is first or not
positions	Square[]	List of all the squares on the board

4.3.2.2. Methods

get_valid_moves()	
Input:	Void
Output:	Move[]
Description:	Returns all the valid moves possible for the current team

4.3.3. Square Class

The class stores the location of an object of Piece Class and helps contain between various points on the Board near the location in order to help calculate each move in the Board Class.

4.3.3.1. Attributes

Name	**Type**	**Description**
here	Piece	The piece contained by the Square
neighbours	Square[]	List of the four neighbouring squares

4.3.3.2. Methods

get_piece()	
Input:	Void
Output:	Piece
Description:	Returns the piece in the current square

set_piece(Piece here)	
Input:	Piece
Output:	Void
Description:	Sets the piece in the current position to the Input

remove_piece()	
Input:	Void
Output:	Void
Description:	Sets the here attribute to NULL

4.3.4. Move Class

This class holds all the information required to specify a single move. Used to organize the information to display legal moves to the user and execute them.

4.3.4.1. Attributes

Name	**Type**	**Description**
start	Square	The game square containing the piece to be moved.
end	Square	This game square the piece will be moved to.

4.3.4.2. Methods

public int[ ] **get_start_pos**()	
Input:	Void
Output:	Integer array of length 2
Description:	Returns the position of the piece to be moved.

public int[ ] **get_end_pos**()	
Input:	Void
Output:	Integer array of length 2
Description:	Returns the position of the square the piece will be moved to.

public void **apply**()	
Input:	Void
Output:	Void
Description:	Makes all required changes to execute a

	move.

4.3.5. C_Move class

This class represents a move that is also capturing an enemy piece. It extends the Move class.

4.3.5.1. Attributes

Name	**Type**	**Description**
capture	Square	The game square containing the piece to be captured.

4.3.5.2. Functions

public void **apply**()	
Input:	Void
Output:	Void
Description:	Makes all required changes to execute a move. (This is an overloaded version that also removes the captured piece)

4.3.6. Board Class

The class stores the location of all the piece and positions on the Board and calculates the moves that are valid.

4.3.6.1. Attributes

Name	**Type**	**Description**
first	bool	If the team about to go is first or not
positions	Square[]	List of all the squares on the board

4.3.6.2. Methods

get_valid_moves()

Input:	Void
Output:	Move[]
Description:	Returns all the valid moves possible for the current team

4.3.7. Piece Class

The class stores the location of an object of Piece Class and helps contain between various points on the Board near the location in order to help calculate each move in the Board Class.

4.3.7.1. Attributes

Name	**Type**	**Description**
team	bool	Specifies which team the piece is on.

4.3.7.2. Methods

Int[] **get_move_directions**()	
Input:	void
Output:	int[]
Description:	This method returns an array specifying the current possible directions it can move in.

4.3.8. King Class

4.3.8.1. Methods

Int[] **get_move_directions**()	
Input:	void
Output:	int[]
Description:	This method returns an array specifying the current possible directions it can move in.

4.4. User Interface Component

The User Interface component is in charge of updating the screen as the user switches between and interacts with each screen. It is responsible for displaying the game board, any buttons, labels, etc, and allows for user input through mouse clicks and triggers the correct actions based on the input.

4.5. GUI Class

4.5.1. Attributes

Name	**Type**	**Description**
frame	Jframe	This is the window that we will add out GUI objects to.
pPanel	JPanel	Holds all the GUI components for the Player Selection Screen
gPanel	JPanel	Holds all the GUI components for the Game Screen
nPanel	JPanel	Holds all the GUI components for the Notation Screen
pLstPlayers	JList	Contains a list of other players found on on our LAN
pBtnPlay	JButton	After selecting a player, clicking this button will start a game with them.
pBtnRefresh	JButton	Refresh the found players list.
pTxtHostAddress	JTextField	Users can input an ip address for another players computer.
pBtnConnect	JButton	After entering an ip address in the host address textbox, clicking this button will attempt to connect to the other player and begin a game.
gSpaces	JLabel[]	Holds all GUI representations for the spaces on the board.

gBtnResign	JButton	Allows player to resign.
gBtnDraw	JButton	Allows player to offer a draw.
gBtnNotation	JButton	Allows player to switch to the Notations Screen.
nLstNotationDisplay	JList	Shows the notation for the current game.
nBtnBack	JButton	Allows player to go back to the Game Screen.

4.5.2. Methods

Move **get_Move**(Board)	
Input:	The board instance that we are using for our game.
Output:	A list of Move objects.
Description:	Returns the move the user just made.

void **display**(Board)	
Input:	The board instance that we are using for our game.
Output:	Void
Description:	This function will update the screen with any changes that need to be made.

void **setupScreens**()	
Input:	Void
Output:	Void
Description:	Create all GUI components and adds them to each screen.

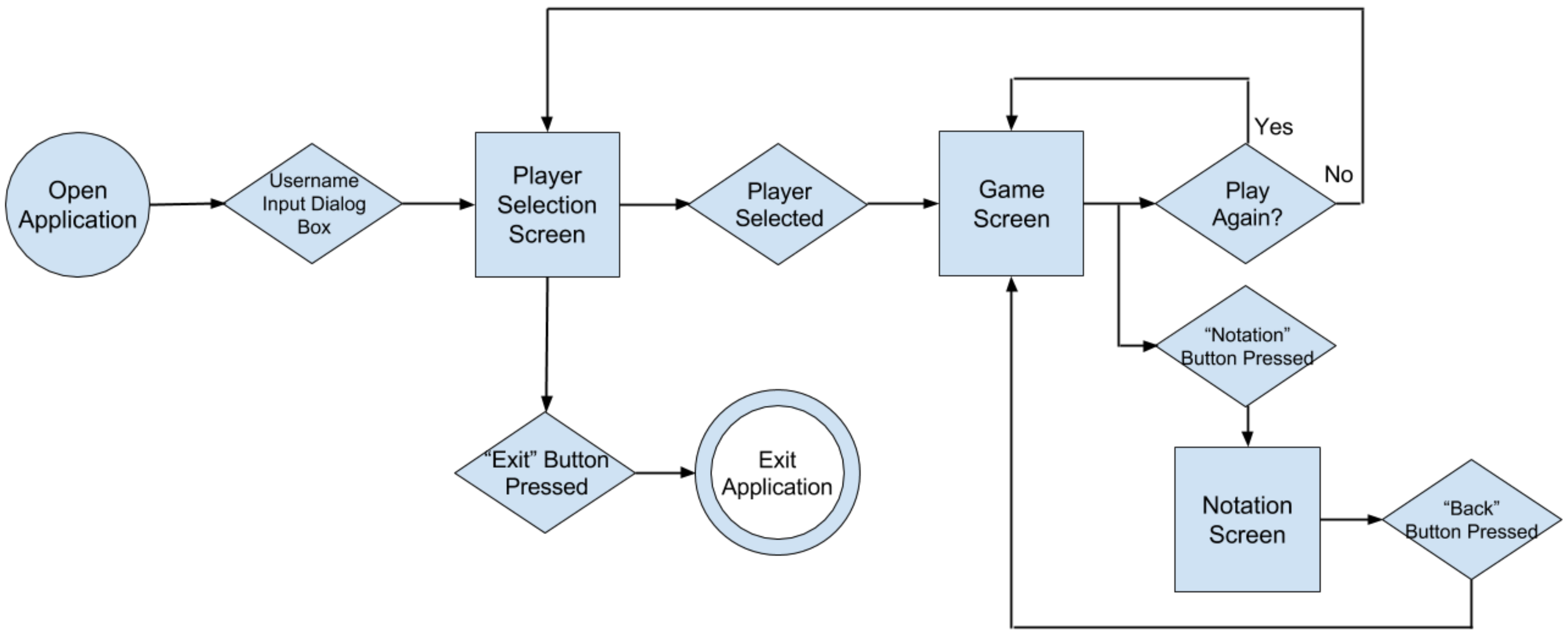
void **setScreen**()	
Input:	Void
Output:	Void
Description:	Create all GUI components and adds them to each screen.

5. User Interface Design

5.1. Overview of User Interface

Our checkers application has three different views that the user will be able to switch between at the appropriate time. We will refer to each view in this document as ’Screens’. The first screen the user will be greeted with, after he chooses a username, will be the ‘Player Selection Screen’. After a user is selected from this page, the screen will switch to the ‘Game Screen’. The Game Screen is where the checkers game is actually played. After a game is completed the user has the option to play again with the same player or not. If the user chooses to not play again the application will switch back to the Player Selection Screen, else it will remain on the same screen and start a new game. The third and last screen will be the ‘Notation Screen’, and this will include a list of all the moves that were made during the current game. You can get to this screen through a ‘Notation’ button on the Game Screen. There is a back button on this screen to get back to the Game Screen. A detailed illustration of the Screen flow is shown below.

5.2. Application Screen Flow



6. References

http://cs.uccs.edu/~cs522/pp/pp.html