Project Documentation
Austin Wellman
Kingdon Barrett

EXECUTIVE SUMMARY

The purpose of this projet was to create an online checkers game to gain experience utilizing the TCP and UDP protocols in java, and the operations of a client-server relationship. The end result was a non-operational incomplete program. This was mostly due to poor planning. Due to the issues that occurred, functionality, documentation, UDP implementation, and a few other specifications are either missing or incomplete.

REQUREMENTS

<not completed due to time running out>

SPECIFICATION

Project specifications govern the actual operating structure and internal workings of the project code. The project specifications have been summarized here.

* Server should support multiple simultaneous game sessions, but only one game is allowed per player.
* Servers should listen on a single port for both incoming TCP and UDP connections.
* Clients should be able to choose to connect using either TCP or UDP.
* TCP communication should be in plaintext.
* UDP communication should be encoded in a binary format.
* Players in the same game may choose to communicate with the server using different protocols.
* Players are identified by a String name, which contains no spaces.
* Games are created on-demand by a client requesting a connection.
* New games sit in a wait state until both players have joined.
* Client prints connection status messages as described in the project documentation to standard output, until the game has begun.
* Client prints error messages (i.e. "Invalid Move") to standard output.
* Client must maintain a data model of the Board, and it should be output on request (or visible at all times).
* ??? After the 15 seconds waiting time, a player generates next move and sends it to the server along with the checkers positions.
* Upon winning or losing, the player must be notified on standard output.
* At the end of the game, the player terminates. Only one game per execution.
* Project documentation will be submitted as `pr1_atw_kpb.pdf`.
* Project source will be submitted as `pr1_atw_kpb.jar`.

FEASIBILITY STUDY

=== MVC Project Architecture vs. Layered Monolithic Design ===
In designing our Checkers product, we had to consider a choice of design philosophy. Some areas of code were a perfect logical fit for a Model-View-Controller architecture. For example, the Player and client interface has a few main requirements, broken down simply:

* Track the game pieces in play, and their position. This is obviously the client data model, represented with a Board object.
* Display the data contained in the model in a format that the human player can easily understand, and allow them to interact with it. The GUI, or view, should display all of the pieces on the board, and it should be clear what options are available to the player. In a checkers game, it is probably an acceptable onus to place on the user that he should understand the rules of the game; thus a simple view is sufficient. The interface should also provide a mechanism to notify the player when their turn has come, and accept moves from the player for processing.
* Processing of player commands is minimalistic; the player "click" events translate directly into a logical move, which can be encoded and transmitted to the server. We leave it up to the server to decide whether a move is valid, and to respond with a set of updates for the local Board. Further, it is the job of the server to notify each Player when their turn has begun and ended.

With such a minimalistic client, the heavy lifting is left mostly for the server. This is beneficial from a security point-of-view, as well as for extensibility. If we are to put a great deal of time and effort into our client to make it attractive and easy to use, ideally it will extend simply to other games like Chess or Go. To implement such a client would require only minor updates to Board, and new game piece images.

The server has a similarly minimalistic architecture, but with a little extra responsibility. A game server must know the rules of the game, it must accept and validate player moves, and it must be capable of manipulating the pieces on the game board. The server architecture breaks down much the same:

* Keep track of multiple games and game pieces. This is a simple Map of Board objects, keyed by game name. Another data structure is necessary to remember which player is currently "up." Optionally, also track the ladder history of players, wins, and losses. The server model has grown slightly more complex, but it is still clearly a simple model.
* Accept connections from a number of clients, and do matchmaking. Once the connections are established, be prepared to send game status updates and receive moves from each player. Validate the received moves, as there is absolutely no guarantee that a client will not cheat. Keep the game board up to date.
* Output status information to standard output for logging. This view only displays a very simple subset of the information available, for instance wins, losses, ties; this is all specified in greater detail in the project requirements.

That's well and good for the client and server, but what about communication between them? We were hoping to find that there would be a large amount of code that could be shared.

=== Dumb Client with Server Validation vs. Client/Server with a shared codebase ===
A major issue to be dealt with was just how much the Player program should do. We initially thought about giving each player class an amount of checkers logic to allow it to determine if it's moves were valid and to allow multi-jumps. Upon thinking about it more, we determined that it would be easiest for the clients to act as terminals, verifying each move with the server to determine what to do next.

=== GUI Builder vs. Plain Text (telnet-style) client interface ===
The interface in which the player would interact with the game was the most obscure part of the documentation for the project that was rarely mentioned. We debated
FIXME: Talk about the benefits of each approach. In short, GUI is nice because it doesn't suck, and telnet-style would probably have the advantage in that it would be a minimal drain on our coding efforts. Board.toString() and a couple of text input methods later, and the whole of the interface would be done.

IMPLEMENTATION
The classes that compose the meat and bones of the project are described here for reference. Further documentation is included within the project code, in the form of comments.

=== Server.java ===
The Server class is used to initiate the game management service. Its basic function is to act as a listener for the initial TCPand UDP connections. Beyond that, it manages the status of the various players (are they playing, waiting, or being waited for?). To sum it up, it acts as the matchmaking service, with the games running within the ServerGame processes used in it.

=== ServerGame.java ===
The ServerGame class where the actual management of the games takes place. It is created initially with one player. After that, another player will join, and it will run as a thread and manage what the game does depending on the data it receives from clients.

=== Checkers.java ===
Checkers class handles validation of moves and manipulation of the board and is an integral part of ServerGame. The complete rules of the game are implemented in the Checkers class, and this is the exclusive controller for the game. Only the server has need for a Checkers object (one per game), while the client simply maintains a board for display and interaction purposes. The client accepts move requests and passes them along to the server for validation and execution. In turn, the server passes messages to the client to indicate game status updates and atomic changes to the board.

=== Board.java ===
Board is a simple 8x8 `char[][]` wrapped in an object. Each Checkers object maintains a Board, and each FIXME!!!!

=== BaseConnection.java ===
BaseConnection acts as a template for TCP and UDP connections to allow for seamless interaction with the game server regardless of the protocols the players are using. It allows for the send, receive, and disconnect commands to be uniform.

=== TCPConnection.java ===
TCPConnection extends BaseConnection. It uses the TCP protocol to send strings over the internet and is used in many classes in our program, both server and client.

=== UDPConnection.java ===
Much like TCPConnection, this is also an extension of BaseConnection. It uses UDP for the same functions.

=== TCPInit.java ===
The TCPInit class used on the Server and is responsible for the always-on TCP port that listens for game requests. It implements Runnable and works like a thread to work parallel to the other processes. It listens for client requests, and if they are valid, initializes a game through the server.

=== UDPInit.java ===
The UDPInit class serves the same function as the TCPInit class, but is for UDP requests instead of TCP.

=== Player.java ===
Player is the executable for the player client. It serves to very fiy the proper connection string, initiate a connection with the server (which is handled by TCPInit or UDPInit depending on the protocol), and wait for the other player, join a game started by another player, or abort if there are any issues. It creates a new PlayerGame for each Player which handles the work after the connection is made.

=== PlayerGame.Java ==
PlayerGame is the main class of the Player. It is in charge of launching the GUI and manipulating the GUI depending on the commands sent to the server, and allowing the player to manipulate the GUI and forward those through a BaseConnection to the server.

=== ClientGUI.java ===
ClientGUI is the player interface to the game. It contains the GUI code along with functions to manipulate the GUI based on server commands, and allow the user to interact with the server. Instead of a board object, it contains a two-dimensional array of buttons that each act as a place on the board.

PRODUCT TESTING
<unable to fully perform and document due to lack of time>

DEVELOPMENT PROCESS
For the most part, the development process we followed was fairly casual. This and poor organization is what led to the final product being non-functional. We both toyed around with how we wanted to do it before we began coding. It began with Austin creating the GUI to get an idea of what the Players would be using and what would be the best way to represent the TCP streams. Things such as LinkedLists were considered, but we eventually settled on plain strings due to their simplicity to use with TCP/UDP connections. A short while after Austin designed the GUI to organize his thoughts and began designing the format of the internet string transmissions, Kingdon set up a project page and subversion repository to coordinate our efforts.

Around this time is where there were some issues with another possible team member who was unable to decide weather or not he was going to be in our group. Upon realizing the code he had done was the same things Austin had done, it was mutually decided that he would not be joining our group.

We had a general idea of how things were going to work, but hadn't ironed out the fine details. Austin began working on the Server class and Kingdon began working on the Checkers logic. So that players could play with eachother using different protocols, it was decided that it would be easiest to represent the TCP and UDP connections as a single type of connection, a BaseConnection. Threading was beginning to become necessary, and different classes were created for active games, protocol-specific initialization tools, and other things that needed to be executed in parallel. Separate TCPinit and UDPinit classes were then created to deal with initialization between players and the server that would run parallel to any ServerGame's that were being played by connected players.

It was discovered that the GUI would not run on the CS machines due to using NetBeans specific features (the FreeLayout to be specific). It was an issue for a while until we changed the layouts used in the GUI to GridBag to conform to the versions on the CS machines.

At the end of the development cycle, I (Austin) have put a lot of time into this project with minimal contributions by the other team member in comparison. I also understand this flaw could have been prevented if I started earlier on as I was told I should.