

3-Stone Documentation

Giancarlo Biasiucci, Svitlana Myranova, Kevin Armstrong Rwigamba



Last Revised: November 20, 2019

Dawson College

Project Status: In Progress (to be changed)

Svitlana Myranova

Giancarlo Biasiucci

Kevin Armstrong Rwigamba

3 Stone Project Documentation:

Date this document was created: 10/21/2019

Last modified: 10/21/2019

Section 1 – Basic-level Game Explanation and Functionality:

The 3 Stone computer game behaves in a much similar fashion to the 3 Stone board game, however with several added functionality geared towards a rule-enforced, online, and client-server approach.

Client-server functionality is intact such that the user must first connect to a server by running the appropriate server program, followed by instantiating the client by running the appropriate client program. A server IP is then requested for the user (client) to connect, which can either be the IP of a computer currently running the same server program or simply localhost to connect to one’s own machine (which as long as the server is running on one’s own computer is guaranteed to work and avoid any possible interference). Once a connection is established, the UI is displayed to the user and the game begins with the user making the first move.

The game takes place on an 11 x 11 board, and each player is allocated 15 stones per game. The client always uses white stones while the server always uses black stones, for both simplicity and making it easier to understand which stone belongs to which player. The center tile is unable to be interacted with, but the user is able to place a stone on any other tile for the first move. Following the traditional rules of the 3 Stone board game, one must place their forthcoming stone in the same row or column as the last stone played, which for functionality purposes is one of the variables stored within our back-end code to facilitate move validation. Points are earned when 3 of any one player’s stones are placed vertically, horizontally, or diagonally adjacent to one another (hence the namesake of the game). The game is over when both players run out of stones. The winner is whomever possesses more points. If, at the game’s end, both players possess an equal amount of points, the game ends in a tie.

Section 2 – Program/Code Structure and Division, More Specific Functionality:

The project encompassing our game’s code, both frontend and backend, is split into 3 partial, functional projects (which will be referred to as “portions” from here on out): client (which includes frontend code), server, and backend code.

Server: This portion sets up a remote server that can accept connection requests from clients, such as the one to be discussed. It starts the server when the portion is run, and closes the server when a game is finished (for efficiency purposes and to avoid any further, unwanted server interactions).

Client: This is where the UI comes into play. When the client portion is run, a window pops up asking for a server IP to request a connection with, which can either be the IP of another computer running a similar server portion, or localhost to connect to a server running on one’s own machine. Validation is performed, and if a server is both found AND running, a connection is established and the main game window appears. The client is given the first move and can place their first stone anywhere EXCEPT the center stone, which is barred for both players. The server then places their stone on a tile within the same row or column as the last stone played, which the client is then required to do, and this continues until both players run out of stones. If the client attempts to place a stone on the barred tile or one that is not within the same row or column as the last stone played, an error message will be displayed and the user must choose another tile to place a stone. The user can also exit the game at any time, which in turn closes the window and stops the server.

Backend: The necessary code required to properly run the game is contained within this portion, including code for the board, player, pieces, and overall game. An appropriate amount of variables are stored to maintain and pass game state as well as to validate actions and enforce necessary rules associated with the physical board game in such a way that is both automated and efficient enough to not get overlooked. The bulk of this backend code is placed within the Board class, as it is the focal class of the game and as such requires an adequate amount of helper and validation methods alike to ensure a fair, clean, and problem-free game.

Section 3 – Source Code Listings:

(Note that the following are links to Git branches and are only accessible by team members and our instructor, who are all maintainers of the repository for this project)

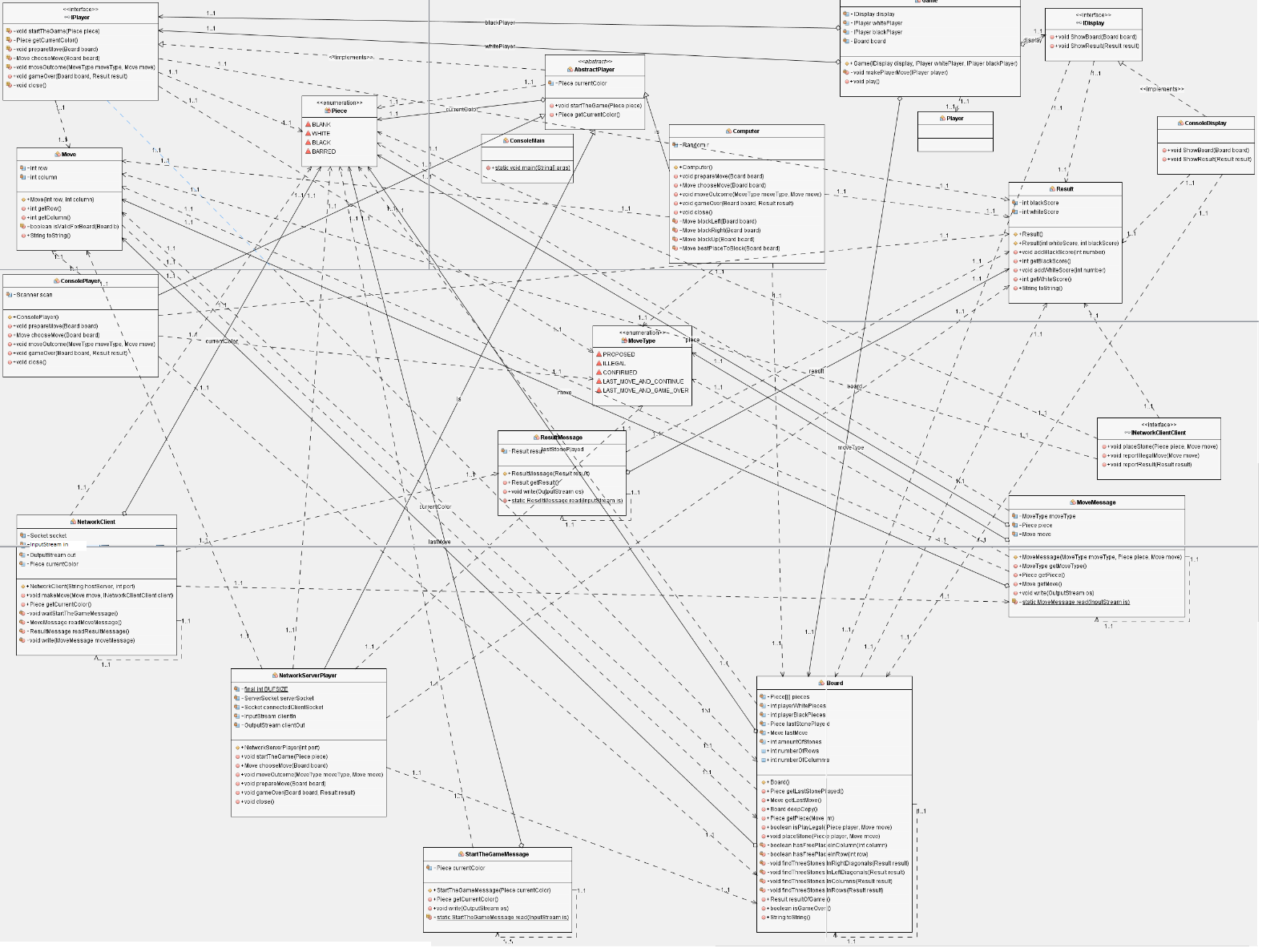
Client: <https://gitlab.com/Tall_Optimist_GC/datacommproject1/tree/staging/3Stone.client>

Server: <https://gitlab.com/Tall_Optimist_GC/datacommproject1/tree/staging/3Stone.server>

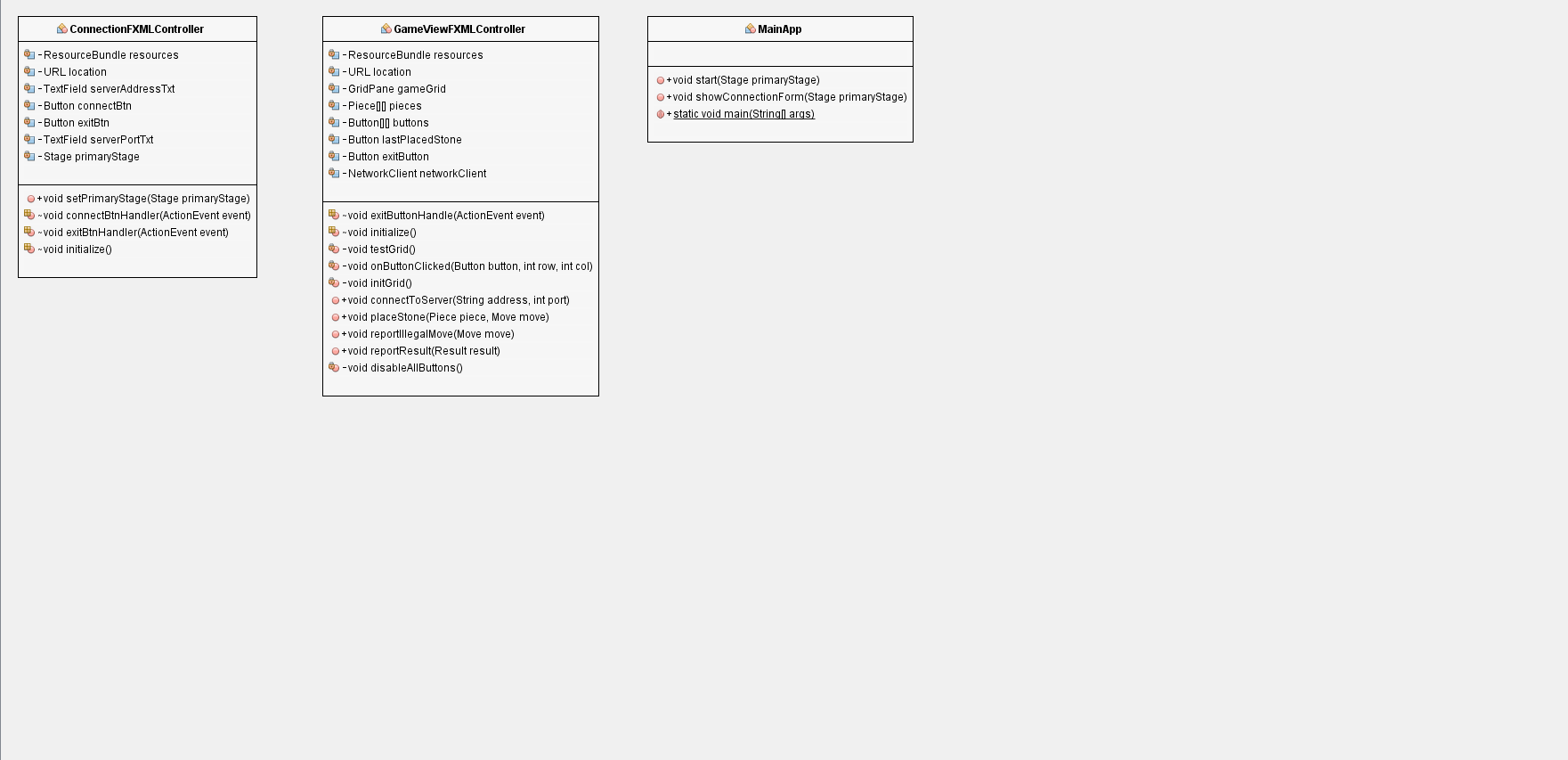
Back-end: <https://gitlab.com/Tall_Optimist_GC/datacommproject1/tree/staging/3Stone.common>

Section 4 : UML’s :

Back-end : (zoom in)



Client :



Server :

