Project Part 3: Progress Report 1

Project

Team Name: GBAR

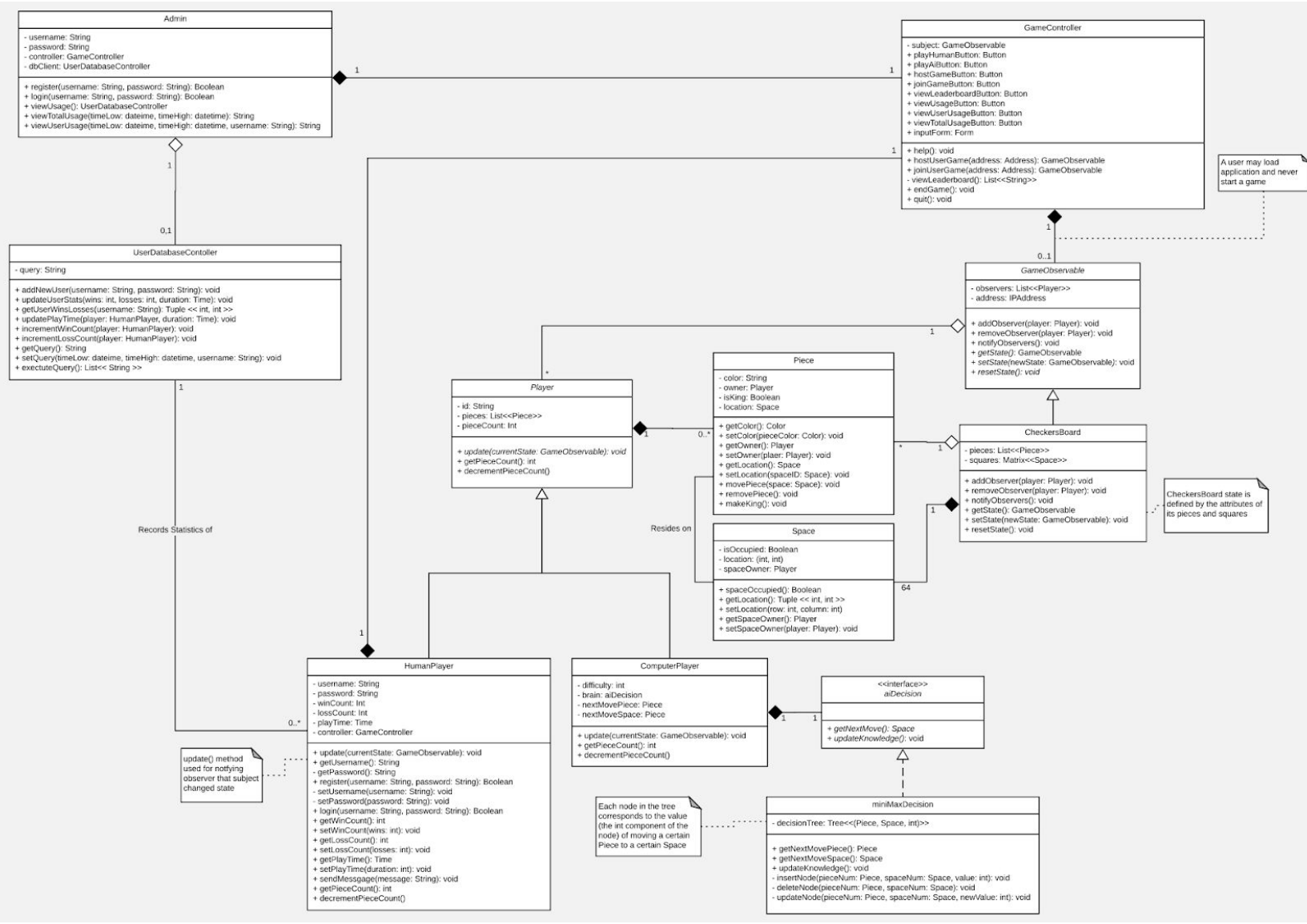
Team Number: 14

Team Members: Blake Galbavy, Autin Metz, Galen Pogoncheff, Richard Poulson

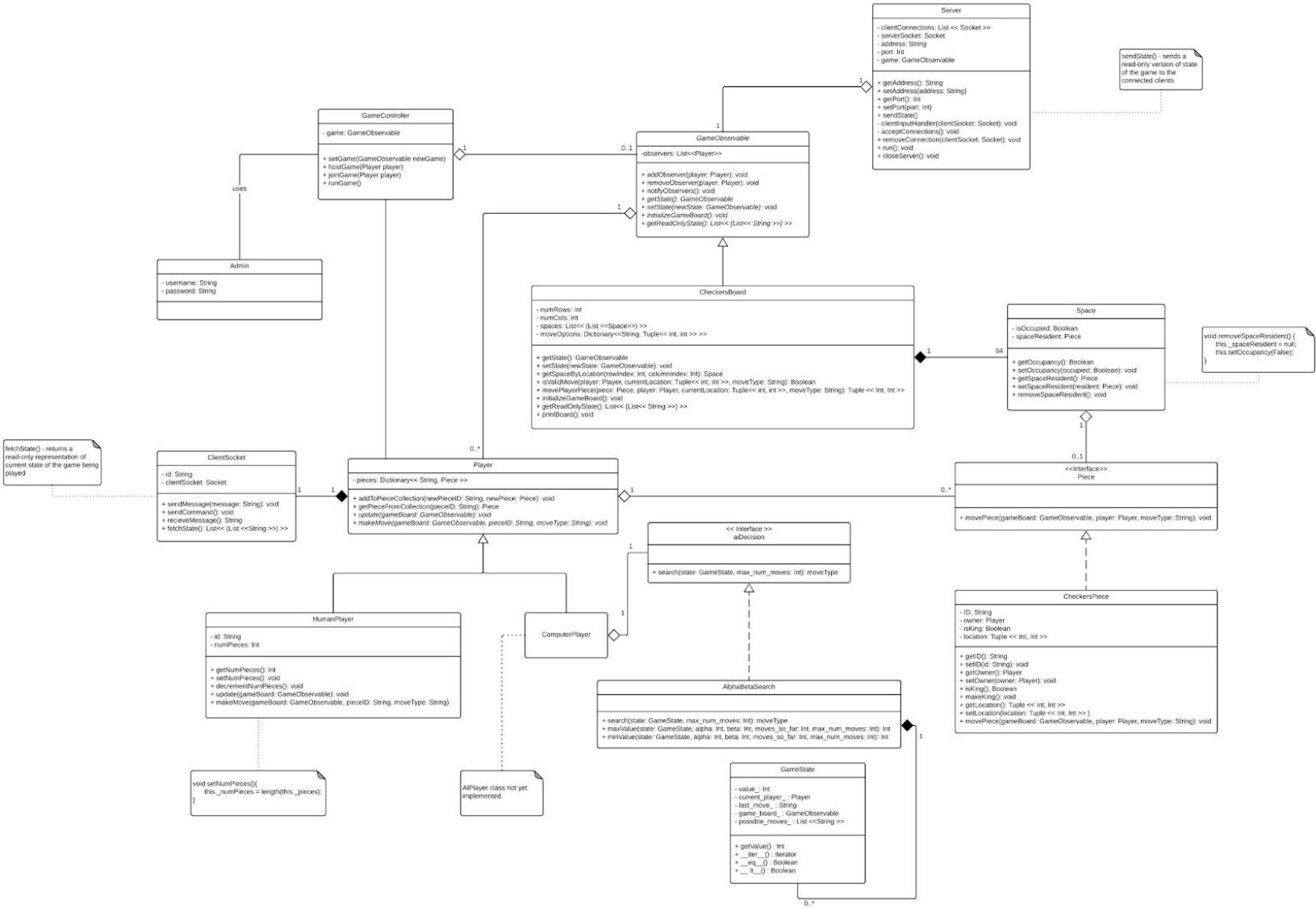
Vision: To create a Checkers game system that can be played both locally or remotely with a friend.

Project Description: **'**Checkers' is an application that allows users to play the classic Checkers board game. Users of the application will have the ability to play checkers against other users on the network, as well as against an automated computer player (driven by artificial intelligence algorithms). Within the game, users can view and interact with a game board whose state reflects the moves of the players in real time. Users also have access to an in-game chat room with their opponent. User statistics, such as number of wins, number of losses, and total play time, will be recorded in a database. Standard users can access win/loss statistics through a leaderboard, and administrators can view user play time as they wish.

Previous Class Diagram



Complete Class Diagram



Summary

Over the past two weeks, we have successfully implemented the basic gameplay functionality of the checkers game, developed the classes required to host a game and send and receive information through network sockets, and we have made progress towards implementing the artificial intelligence gameplay algorithms and incorporating a graphical user interface. With the work that has been completed so far, a game of checkers can be played through a command line interface on a single local machine. In order to accomplish this, the essential gameplay features of all of the components of the Checkers game have been implemented, however, not refined. During these last two weeks, we have also spent time revising the design of our system to better comply with the object oriented principles and design patterns that we have been studying.

Breakdown

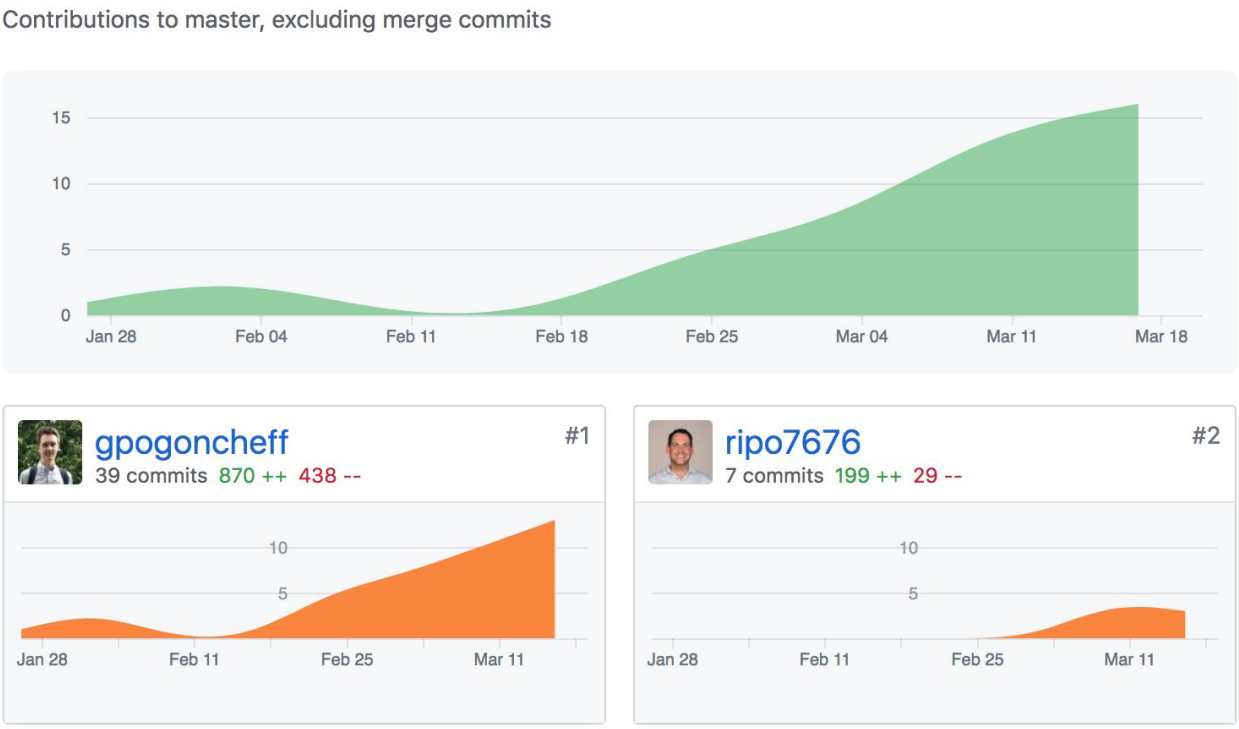
Blake Galbavy: Blake has been working on developing the User and Admin classes.

Austin Metz: Austin has been working on development of the graphical user interface and its integration with the system via the Model View Controller architectural pattern.

Galen Pogoncheff: Galen has implemented the classes of the system required for playing a Checkers game along with some of the methods used in the GameController class that modify the Game model. This includes the implementation of the abstract GameObservable class, the CheckerBoard class, the Space class, the Piece interface, the CheckersPiece class, the abstract Player class, and the HumanPlayer class. Galen has also implemented the classes that will be used to enable the network gameplay functionality. This includes development of the Server class and the ClientSocket class.

Richard Poulson: Richard has been working on the strategy algorithm used by the artificial intelligence computer player to decide its next move.

GitHub Graph:



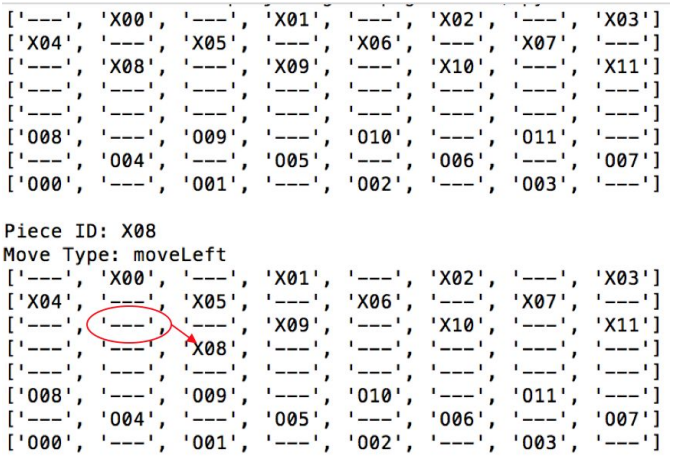
Estimated Remaining Effort:

The bulk of the system that is critical to the gameplay of the checkers game has been completed in this iteration. Moving forward the major areas that need development attention are as follows. We will need to connect the game system to a database to enable the recording of user statistics and login functionality. Another area of effort is the integration of the game system with the network socket components to enable a user to play a game against another user over a network connection. These two separate components have been implemented in this iteration, but the convergence of these components must be done. To allow a user to play a single-player game of checkers locally, we must also finish the Artificial Intelligence player class (ComputerPlayer) and its associated decision algorithms. The final area of effort is the graphical interface used to interact with the game model.

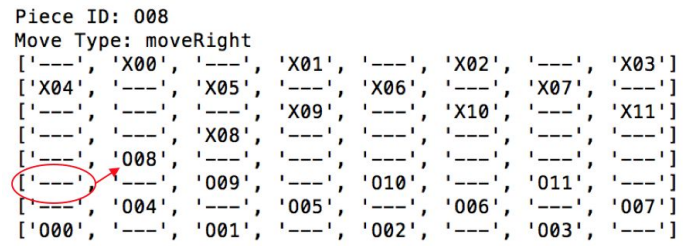
As we progress in the development of our system, we have been updating our system design. One driving factor of our design changes are to better enforce single responsibility of the classes of our system, support the open-closed principle, and reduce coupling between system classes. To accomplish this, we have better encapsulated the relevant attributes and methods of each class to make the class more cohesive, introduced sources of composition to make sure that each class if responsible for tasks relevant to it, and we altered some of the class methods and attributes to reduce dependence on the implementation of other classes.

Provided below of screenshots of our system in use:

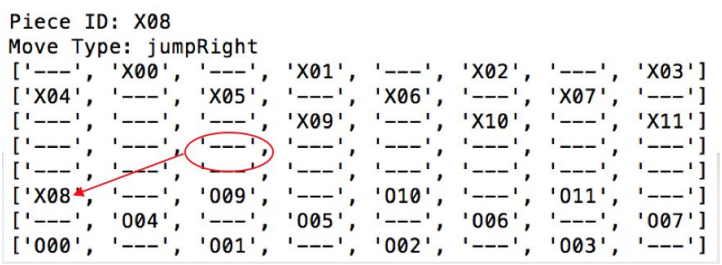
Player 1 moving piece ‘X08’ left diagonally (from perspective of Player 1):



Player 2 moving piece ‘O08’ right diagonally (from perspective of Player 2):



Player 1 jumping piece ‘X08’ right diagonally (from perspective of Player 1) over ‘O08’, therefore capturing opponent piece ‘O08’:



Screenshot of Client-Server Communication:

```
recieved connection from 127.0.0.1:53673
recieved connection from 127.0.0.1:53674
Message 1: This is a message from player 1
Client 1 recieved:  This is a message from player 1
Client 2 recieved:  This is a message from player 1
Message 2: This is a message from player 2!
Client 1 recieved:  This is a message from player 2!
Client 2 recieved:  This is a message from player 2!
```

Next Iteration:

Next iteration we plan to have the game system integrated with the network socket classes to enable gameplay with another user over the network. We also plan to have basic functionality of the artificial intelligence player implemented and plan to connect the game system to a database and begin implementation of a database proxy. During this first iteration, we have also made note of parts of our system that may benefit from refactoring to design patterns (e.g., using the strategy pattern to delegate movement calculations to piece movement algorithms, and possibly introducing the flyweight pattern to manage the game pieces).