Lucas Lopilato
3288867

CS165A MP2 Report

Architecture

For MP2, I tried to keep as much encapsulation as possible. The main point of entry is the file Gobang.cpp. This file is responsible for argument parsing and initializing the Player Class. The player then initializes the Board and handles the entire course of playing the game.

Player handles the AI and Player handling for the entire game. The function play switches between AI and human players, outputs moves, and manipulates the board. It is in this class that the minimax function is implemented.

The Board class handles all of the logistics of placing pieces, scoring moves, determining boundaries etc. The physical Board consists of a 2d array of an enum Color which is either EMPTY LIGHT or DARK. For scoring, the algorithm builds strings from directions (class) which are then parsed for certain patterns. A pattern consists of 'X' which means friendly, 'Y' means out of bounds or enemy, and 'E' means empty. From there the Player's scoring dictionary will evaluate and score based on which patterns are detected.

The Direction Class is a simple helper to allow a vector to easily travel in one direction. For example to search a specific diagonal, one would simply call next with the direction LL, and the vector will move down and to the left on the matrix and return that position.

The Move Class is a simple wrapper of a Position(std::pair) and a color since Moves are easily bundled together as such. The Board class accepts Moves in order to update the board.

Search

For search, I implemented one level of depth of search in minimax. Due to the high complexity of the search, I had trouble being able to extend the search to deeper levels. As it stands, the program will search all spaces for the best possible combination of patterns and will move appropriately. Since I had trouble with search optimization, I have the algorithm watch the clock to try and prevent timeouts.

To score, I had a for loop through the 2d array searching in 4 directions. The search would begin at the upper left corner, read the first piece in the opposite direction(to check if it is open or taken), then move in the original direction and build the rest of the string. With this approach I found I had much more information about my surroundings and could make better decisions.

From there, I have each string matched with a pattern to score. Whatever was found results in the final score, allowing for many situations to be handled.

Challenges

My biggest challenge was search optimization. No matter what I tried, I found it very difficult to have my program run efficiently. The cost would come as a direct tradeoff of proper search, and a best-guess. As more moves are made, however, the time of search reduces significantly, and easily allows

the full search to be run.

Weaknesses

My weaknesses are early game moves. Since my program may guess rather than get a complete search, it may miss certain moves that are necessary to play a perfect game. Since I had a hard time with larger arrays, I also had to implement more clock checks which take away from the ability to finish more searches.

Overall, I really liked this project, I am just a little disappointed in the level of depth I achieved. I believe I will go back and work on this project after finals. Thank you for a wonderful quarter!