Architecture: My program consists of 1 class, Board.py, which Evaluation.py file and Gobang.py file utilize to interpret the state of the board. Board.py defines the state of the board. It consists of the board, which is a 2D array initially implemented with 0's, which gets filled by 1 for the light player move or a 2 for the black player move. It has a size, which stores the specified command line value for -n or the default 11. It has a won Boolean, which is set to True if my code detects a victory (arbitrary who won). It has a player, which represents the current player and a my_player which represents the color my AI represents. It has max_cells which is how many spaces there are in the board and a cells_occupied which is how many spaces are taken (arbitrary who took it). These are used to evaluate whether the board has been completely filled resulting in a tie with nobody having won. Board.py has several helper functions which help me determine invalid moves or get the current status of a property of the board from Evaluation or Gobang, but the main highlight is the move function, which copies the board and updates the properties of the new board, but also verifies if the new board is a winning board. Is_win is a function that loops through all possible directions in the form of coordinates, and calls a recursive function called check_connection, which keeps iterating and adding 1 representing how many pieces are connected on the board, until it reaches out of scope or hits a color that is the opposite, in which case it returns, and should get called again in the other direction by the for loop in is_win. This is to ensure that both sides are checked for horizontal, vertical or diagonal connections. Evaluation is through which I control the board, without having to touch the board at all in the Gobang.py. Many of the functions there I added from suggestion of the book for the class, for example Actions or Result. I decided to use the iterative deepening method, to call alpha_beta_search which will use max_value and min_value back and forth, pruning all the while. The iterative deepening is based off of the global restricted time I set (28 seconds) and it will stop once the code sees it surpassed the 28 seconds, to which it will unwind and return its best evaluation. Gobang.py just has the code to read the specifications for the input, and uses Evaluation to influence the Board.

Search: I decided to use the iterative deepening method, to call alpha_beta_search which will use max_value and min_value back and forth, pruning all the while. The iterative deepening is based on the global restricted time I set (28 seconds) in combination with the restricted depth which keeps getting incremented when the recursion unwinds and returns until the time restriction is met, to which it will unwind for the last time and return its best evaluation. I read that the iterative deepening process can be utilized to better enhance the efficiency of the program if it was used to order according to the best move before recursing through again. So every time it unwound it would return the selected coordinate and its corresponding value (the maximum points) as a pair of coordinate and the value ((x,y),value) only to recurse through the new depth again. Every time I would sort the list according to the value so the action decided both returns the maximum value, but the code iteration will also go directly towards the best move and continue to deepen immediately. My evaluation function basically keeps count of the connected pieces by looking at the moves for the light and dark that I kept count of in an array, and assigning weights according to 4^n for every additional connected piece. If it sees a 4, it adds an additional 4^n to the summation because it should prioritize connections of 4 for both the opponent or myself.

Challenges: I found it difficult to figure out how to structure the code so it works together to represent a game. I had trouble defining the state, which was the board and initially it seemed a little abstract how I would incorporate the evaluation function with the alpha beta pruning. I resolved this through a lot of reading and re-reading of the book, looking through lecture slides then looking at lectures online

defining the concepts and their use. After figuring out the structure it became simple enough to code what I designed, however when I reached the evaluation function (the heuristic) I reached another struggle because I was trying to figure out how to define the weights to exemplify what is a better board than another. I resolved this by looking into what a good evaluation function does for games, which defines weights according to attack and defensive techniques.

Weaknesses: I am very sure my heuristic could be designed much better. I see how it neglects many cases and doesn't assign overlap weights for locations that could be connected in a few ways. I could probably also make the search much more efficient thereby leaving room to further examine potential paths and verify that they are indeed the best action to make. I feel the main problem though is that because the code is written in python it cannot go down the tree very deep on large input sizes, and so it doesn't have time to choose a good move and ignores the requirement to block.