Architecture:

Representation of the board:

I use a struct named "state" which represents the board in each state. The struct contains a pointer to a two-dimensional char array which represents the board.

Private variables:

Private variables contains currentState(of board), AIPiece ('b' or 'w'), HumanPiece, depth(of minimax), etc.

Constructor:

Constructor is used to initialize these variables through the input taken from command line **arguments.**

Classes used to for the heruistics:

The functions **evalFunction** is used to calculate score for each score. It takes in state, depth, and bool for isAITurn as an argument. It uses the functions "horizontal", "vertical", and "diagonal" to manipulate strings vertically, horizontally and diagonally. And with the help of countPattern, it count the number of pattern that cause threat, and give them score accordingly.

Classes used for implementing minimax:

AIOptimalMove and minmax are used to implement recursive minimax algorithm. AIOpitimalMove applies minimax to all the available moves for maximizer and compares the value for each state and saves the state which gives the maximum value.

The minmax is a recursive function that takes the current state, depth, isAITurn, alpha and beta as an argument and tries to find the maximizer move by doing depth first search.

Utilities:

The function startGame starts the game. GameEnds function is to check whether in the current state game ends. isTie function is to check the current state is tie.

Search:

I used minimax algorithm to find the best move. So, lets say s is the current state. I apply minimax to all the available moves of current state until depth = 2 or state reaches terminal (game ends or tie) to get the maximum evaluated state for all those moves. And then opponent plays. And I repeat minimax algorithm again.

I have also applied alpha-beta pruning in my algorithm. Initially values of alpha and delta are -infinity and infinity. When I do DFS, I track the values of alpha and beta, and whenever the alpha >= beta, I stop searching that branch.

For my evaluation function, I look for specific pattern of straight four “XbbbbX” (100000 points), the four “bbbbX” or “Xbbbb” (50000) points, the three, and broken three. These are the real threats while someone plays the move. Using the evaluation points, my AI immediately block the threats when opponent tries to play like this.

Weakness:

I could have just searched adjacent moves instead of all available moves. It would have been much more optimized. All I use lot of space of string manipulation for my evaluation function, instead I should have just kept track of evaluated values for each state while searching.