**Introduction to Artificial Intelligence**

**Final project report**

**German Ruiz, Jinhao Chen, Georgy Loriya**

For our project we have chosen to create a pawn game with the board size of 3 by 5. The game rules are simple, the player wins only if their pawn reaches the end of the board, or captures all the opponent’s pawns. Pawns will be blocked if another pawn is directly in front of it. We have implemented the depth limited minimax algorithm. The algorithm will choose the move that is most beneficial to itself and most damaging to the opponent.

**Implementation:**

The program consists of board function that creates a 3x5 board. Then we have implemented a function that places all the pawns in the start position. Then we have a function that takes the input of the user and transforms it to an actual move on the board. Then we have a function that will determine if that move will lead to win or lose. On the other hand, we have implemented depth limited minimax algorithm for the PC to determine which move to make.

The board is represented as a 2D array and it is accessed and modified after each move. There is a single array for the location of the PC and the user’s pawns.

The game program was written in C++. The program uses a 3x5 board to check to see if a pawn can go all the way forward to the enemy’s end zone, or if it can capture all of the opponent’s pawns, or if all pawns of the opponent are blocked. The program was divided into separate functions to subdivide each step into sub-problems for simplicity.

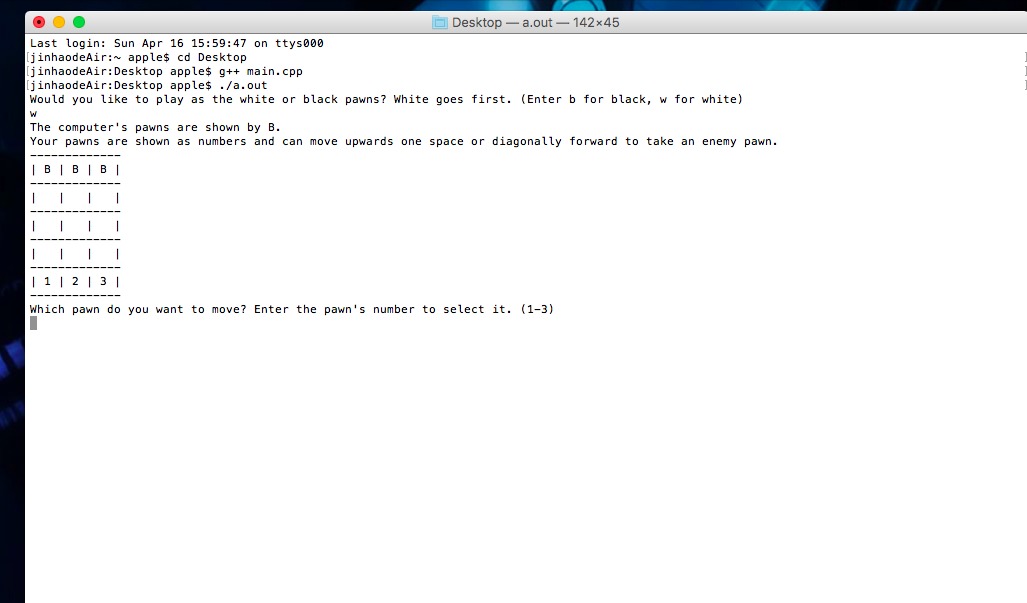
First, the program asks the user to specify the player as either black or white pawns. White pawns have the priority to play the first move. A 2D array was implement in the display function which was used to display the 3x5 board and initializes the computer to the first three elements of the array which is the first row. Whether player chooses the white or black, the function initializes the player to the last three elements of the array, which is the last (bottom) row. A choose function was implemented to ask the user which pawn she or he wants to move, and the function moves the pawn to the appropriate forward place. There is also a player’s turn function which helps to implement the right and left diagonal movements to capture the enemy pawn. In order to achieve those steps, three different Boolean functions were implemented to check the movements namely moving forward, left, and right. The left and right movement functions were only used to capture the enemy pawns. If these Boolean movement functions returned true, then the real move functions were called to make the actual movement on the board. The importance of these functions is to make sure that a player or computer cannot move the pawn to the outside of the scope of array. As mentioned earlier, the player’s turn function checks all the possible conditions for a pawn to be played. Therefore, if there is a capture movement, it asks the user to play which side he or she wants to capture. Also, if any pawn that is chosen by a player is blocked, then it asks player to choose another one to move.

After implementing all the movements and their helper functions, the results functions were implemented. Two different ‘win’ functions were used for both the computer and the player. Also, another two functions were used for the losing condition. Loss functions check to see if any of the enemy pawns reached the end point, all the pawns are captured, or all the pawns are blocked so there is nowhere to move. Win functions were also implemented using the same logic.

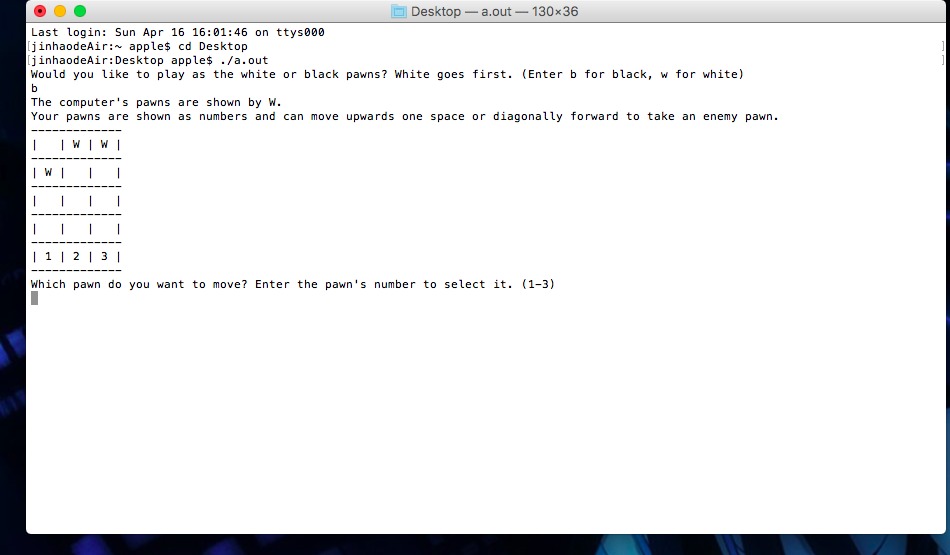
The most important thing in this project was to design the minimax function which is used for the computer’s best movements. This function searches all the possible movements to make for the computer. The function checks to see if the computer can move forward, left, or right, and it decides to choose the best one and store it as highest priority. The computer makes the move which leaves its opponent capable of doing the least damage.

**Result:**

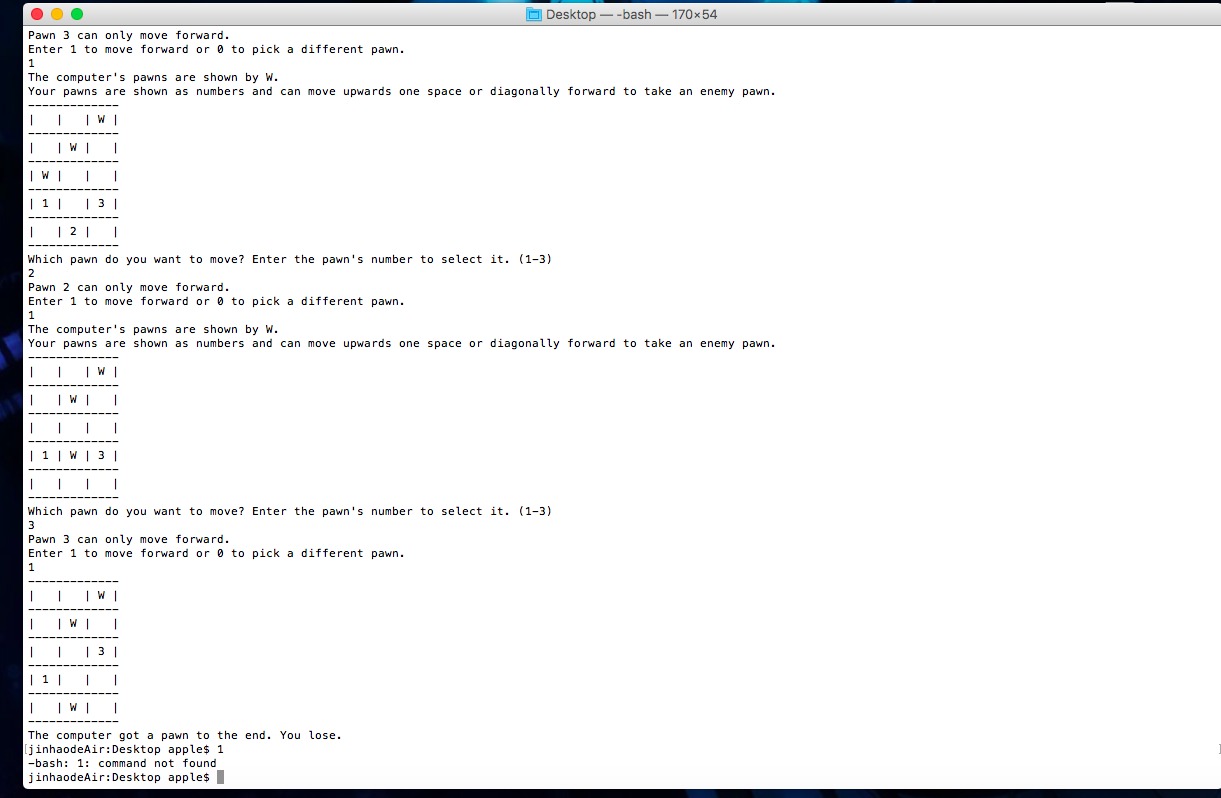
Here the player chooses White pawns to make first move.



The second image shows when the player chooses the Black pawn to move. And the computer will make the first move:



The third image is one test result that shows computer reach the end of the board. And computer wins!



Here are few more examples of the AI playing the game:

<https://youtu.be/F_nZoPONOcc>

This is the link to our game demo.

**Conclusion:**

From all the tests we made, it was difficult to defeat the AI. There were some obstacles to make the AI undefeatable such as complicated formulas to calculate the accurate heuristics. Since we have splitted the location of the pawns and the board it made our program more complicated. The algorithm determines the best position of the pawn and makes that move its priority. The minimax algorithm is extremely efficient in zero sum games and this example proves it.