**Gomoku**

**Part 1. Reflex agent:**

**Implementation:**

For the reflex agent, I start from a random location for both the red player and the blue player. And I use a 7\*7 integer array to record each state. The end state is when someone wins(5 in a row, column, or diagonal) or ties(the board is already totally filled up).

I check if I can get the chance to win the game(5 consecutive pieces) first, and then check if the opponent is going to win(4 consecutive pieces), and I then the opponent got 3 consecutive pieces without being blocked. At last, I are devoted to finding the location that is going to maximize our number of consecutive pieces. If the following piece can’t be determined by the 4 rules above, I assign the pieces randomly.

——————————————————————————————————————

**Part 2. Minimax agent and alpha-beta agent**

**1: Minimax agent**

**Implementation:**

For the minimax agent, I search the future state of the board in a depth of 3. I simply expand all of the possible states(the number of them is “(number\_of\_empty\_points\_in\_current\_state)\*( number\_of\_empty\_points\_in\_current\_state-1) \*( number\_of\_empty\_points\_in\_current\_state-2))”. The evaluation function I used is based on the rewards and penalty of each situation.

1. If the player is going to win with taking a certain step, I give the highest rewards for the step(5000).
2. If the player is going to have 4 consecutive pieces, I give the second highest rewards for the step(500).
3. If the player is going to have 3 consecutive pieces without being blocked, I give the third highest rewards for the step(100).
4. If the player is going to have 2 consecutive pieces without being blocked, I give the fourth highest rewards for the step(10).
5. If the opponent is going to win, I give the highest penalty for the step(-5000).
6. If the opponent is going to have 4 consecutive pieces, I give the second highest penalty for the step(-500).
7. If the opponent is going to have 3 consecutive pieces without being blocked, I give the third penalty for the step(-70).
8. If the opponent is going to 2 consecutive pieces without being blocked, I give the highest penalty for the step(-7).

With the rules of the evaluation function designed above, I are able to search

in depth 3 accordingly. For the third layer, I try to maximize the rewards for the player; for the second one, I try to minimize the rewards of the third layer; for the first one, I try to maximize the rewards of the second layer(minimax), with the rewards(penalty) calculated via the evaluation function.

**2: Alpha Beta agent**

**Implementation:**

For the alpha-beta agent, I search the future state of the board in a depth of 3. I don’t expand all of the possible states, if the minimum of the second layer is already decided, and the next found evaluation function result at the third layer is less than the current minimum of the second layer, I simply prune the rest of the branches(they give no help to the decision making). The evaluation function I used is based on the rewards and penalty of each situation.

1. If the player is going to win with taking a certain step, I give the highest rewards for the step(5000).
2. If the player is going to have 4 consecutive pieces, I give the second highest rewards for the step(500).
3. If the player is going to have 3 consecutive pieces without being blocked, I give the third highest rewards for the step(100).
4. If the player is going to have 2 consecutive pieces without being blocked, I give the fourth highest rewards for the step(10).
5. If the opponent is going to win, I give the highest penalty for the step(-5000).
6. If the opponent is going to have 4 consecutive pieces, I give the second highest penalty for the step(-500).
7. If the opponent is going to have 3 consecutive pieces without being blocked, I give the third penalty for the step(-70).
8. If the opponent is going to 2 consecutive pieces without being blocked, I give the highest penalty for the step(-7).

With the rules of the evaluation function designed above, I are able to search

in depth 3 accordingly, and with much less node expanded(due to the pruning). For the third layer, I try to maximize the rewards for the player; for the second one, I try to minimize the rewards of the third layer; for the first one, I try to maximize the rewards of the second layer(alpha-beta), with the rewards(penalty) calculated via the evaluation function.

**Part 3. Stochastic search:**

**Implementation:**

For the stochastic search, I generate a random number of the successive states, and calculate the evaluation function result of them with a certain depth. I then back track to the first following state with the highest corresponding evaluation function simulated.

This is similar to Monte-Carlo search, which I use the method of probability to estimate the best next move. It greatly reduces the node expanded and run time, while the performance is not degraded a lot.

The evaluation function I used is based on the rewards and penalty of each situation.

1. If the player is going to win with taking a certain step, I give the highest rewards for the step(5000).
2. If the player is going to have 4 consecutive pieces, I give the second highest rewards for the step(500).
3. If the player is going to have 3 consecutive pieces without being blocked, I give the third highest rewards for the step(100).
4. If the player is going to have 2 consecutive pieces without being blocked, I give the fourth highest rewards for the step(10).
5. If the opponent is going to win, I give the highest penalty for the step(-5000).
6. If the opponent is going to have 4 consecutive pieces, I give the second highest penalty for the step(-500).
7. If the opponent is going to have 3 consecutive pieces without being blocked, I give the third penalty for the step(-70).
8. If the opponent is going to 2 consecutive pieces without being blocked, I give the highest penalty for the step(-7).

I have limited and tested the number of simulation for each level of the

branches, and I choose N=5~20 to check the performance, I then find even if I only check 5 successive states, the results will be good enough.

**Part 4. Supervised learning(Extra credit):**

**Implementation:**

For the feature extraction part, I record the final state of the board. The feature is extracted with the following steps.

I check the state in each array element(blank, red piece, or blue piece), and if it is blank, then the corresponding 3\*(i\*column\_number+j)+1、3\*(i\*column\_number+j)+2、3\*(i\*column\_number+j)+3 features is going to be (1,0,0); if it is red piece, then the corresponding 3\*(i\*column\_number+j)+1、3\*(i\*column\_number+j)+2、3\*(i\*column\_number+j)+3 features is going to be (0,1,0); if it is blue piece, then the corresponding 3\*(i\*column\_number+j)+1、3\*(i\*column\_number+j)+2、3\*(i\*column\_number+j)+3 features is going to be (0,0,1), where “i” is the row index and “j” is the column index(both from 1~7)

So I can get a total of 147 features(7\*7\*3), and I collect the target(0 if lose or tied, and 1 if win). Hence, a total of 148 data are recorded in each round. This part is done in “2\_4\_user\_interface\_training” folder, I wrote a batch file to run the algorithm N=500 times and record N\*148 data(stored in training.csv file).

I then use a python file(numpy and pandas is used) to read the csv file, and implement a logistic regression and back propagation to calculate the weights corresponding to each features. I use the function f(x)= to map the inner product of the weights and features “x” to a value “f(x)” between 0 and 1, I can thus compare it with the target ground truth.

Finally, I implement an interface that enables the user to play gomoku with the computer agent(with the supervised learning model trained). The evaluation function of the computer agent is calculated as the inner products of the current board features and the calculated weights(obtained in the preivious step with python code). I can thus increase the chance of winning by the weight I have trained.