Connect 4 Artificial Intelligence Agent

Evaluation Function

To design our evaluation function we searched the Internet for some suggestions on ways to play connect 4. We read a couple of articles and watched a couple of videos and came out with a list of things that are good or bad for you as a player. Among these things, we found that playing the middle row is good, having pairs of chips with spaces on one or both sides is good, and having three chips with spaces on both sides guarantees your win, using what is called a 7 configuration is also good, and looking various moves ahead. You should try to avoid these from your opponent and, you should block a win from your opponent and try not to set a win for your opponent. With this in mind, we set to make auxiliary functions to identify these things in the current state of the board being evaluated (which would be several moves in the future). Inside these auxiliary functions, we will set some evaluation points depending on what we find for the specified player. On the evaluation function, we multiply the evaluation points returned from the auxiliary functions by a weight assigned to each one based on what the auxiliary function looks like for each of the two players. What is good for one player is bad for the other. We decided to put more weight on the bad things for this player so the agent could block the opponent from making these moves.

The evaluation function is thus, in terms of the auxiliary functions that find the specified conditions:

Value = 1 * countInColumn(3, player)
- 1 * countInColumn(3, opponent)
+ 3 * consecutive2(player)
+ 10 * consecutive3(opponent)
- 3 * consecutive2(player)
- 10 * consecutive3(opponent)
+ 20 * seven(player)
- 50 * seven(opponent)
+ 1000 * gameOver(player)
- 1000 * gameOver(opponent)

The countInClumn function just counts how many chips the specified player has in the specified column and gives a point for each chip. Column 3 is the middle column. The consecutive2 and consecutive3 functions look for consecutive chips in all horizontal, vertical, and diagonal positions and give a point for each pair with a space before or after the pair adding another point if spaces on both sides. The seven looks for the 7 positions normal, reflected and inverted giving a point for each one found. There is another configuration like the 7 that we noticed but is not programmed as it was not in the reference material. GameOver looks to see if the player has a winning move on the board giving a point if it finds one. One possible function not included that we realized looking at the system play is to look for three chips with a space in the middle because that is a possible winner. We think this would improve the evaluation function. We use the same evaluation function for the minimax and alpha-beta algorithms. Because of this, we made the alpha-beta class inherit from the minimax class which inherits from the connect4 class.

Here is an example of a board being evaluated for player 2:

```
[[0 0 0 0 0 0 0]
 [0 0 2 0 0 0 0]
 [0 0 2 0 0 0 0]
 [0 1 2 1 0 0 0]
 [0 1 2 1 0 2 0]
 [0 2 1 1 2 1 0]]
```

Value = 1 * countInColumn(3, player)
- 1 * countInColumn(3, opponent)
+ 3 * consecutive2(player)
+ 10 * consecutive3(opponent)
- 5 * consecutive2(player)
- 10 * consecutive3(opponent)
+ 20 * seven(player)
- 50 * seven(opponent)
+ 1000 * gameOver(player)
- 1000 * gameOver(player)

Value = 1 * 0
- 1 * 3
+ 3 * 0
+ 10 * 0
- 5 * 1
- 10 * 1
+ 20 * 0
- 50 * 0
+ 1000 * 1
- 1000 * 0

Value = 982

Coding the Agent

With the code provided, we had the *minimaxAI* and *alphabetaAI* classes to implement. We implemented the *minimaxAi* class derived from the *connect4* class as it was. Then we changed the *alphabetaAi* to inherit from our *minimaxAI* class as we wanted to use the same code for the evaluation function without having to copy it over again. We used a shallow depth of 2 for execution in 0.5 secs or less as required with great results, so we did not implemented reordering in the alpha-beta.

Benchmarking the Evaluation Function

alphabetaAI vs stupidAI

alphabetaAI wins both as player 1 and player 2

alphabetaAI vs randomAI

Alphabeta wins out of 5 games

Wins	Seed				
Player	0	1	2	3	4
As 1	5	5	5	5	5
As 2	5	5	5	5	5

Alphabeta wins 100 % of the time

alphabetaAI vs montecarloAI

Alphabeta wins out of 10 games

Wins	Seed				
Player	0	1	2	3	4
As 1	7	10	10	3	10
As 2	9	10	1	1	10

Wins	Seed				
Player	5	6	7	8	9
As 1	0	10	10	10	10
As 2	6	10	1	8	10

Total Wins as player 1 = 80/100 = 80%
Total Wins as player 2 = 66/100 = 66%

Total Wins = 146/200 = 73%

Discussion

The desired results were to beat the stupidAi and the randomAi and it does it. Also, it was expected to be competitive with monteCarloAI and we think we achieved that by beating the monteCarloAI overall on 73% of the games, with an outstanding 80% win rate when it starts the game.