Game Playing through Adversarial Search

Binaka Ponkiya

Bhanu Prakash Cherukuri

11/16/2020

I am submitting my own work I have not provided any help to any other student in this class and I did not receive any help from any source including those that are online, print or persons. The only exception is the interaction with my homework study partner within the permissible context. This submission is in full compliance with the academic dishonesty policy as elaborated by the University and the course syllabus.

Description of Program implementation:

In main function first we are setting the initial board position as given in the problem statement with 10 black stones in one corner and 10 white stones in one corner and it will print the board.

then **gameplay()** will take argument of initial board position and it will start game in alternate turns starting with first blacks turn here AI is playing with black stones and random agent is playing with white stones,.

Blackmoves() will get all the available moves and first check if it is having a move or not if it is having moves then it uses **evaluatemove()** which is implementing minimax algorithm with alpha-beta pruning technique. To implement minimax with alpha-beta pruning we come up with couple of evaluation functions which is discussed in the next section.

We used aspiration search from paper presented in IEEE by Jonathan Schaeffer (The History Heuristic and Alpha-Beta Search):

"*The cup search is usually started with a (-infinity, +infinity) search window. If some idea of the range in which the value of the search will fall is available, then tighter bounds can be placed on the initial window. Aspiration search involves using a smaller initial search window. If the value falls within the window, then the original window was adequate. Otherwise, one must re-search with a wider window*" (CUI-Jinling)

Here we used (-100,+100) as tighter bounds to use smaller initial search window.

Using this evaluation function, minimax and alpha-beta, blackmove() will get the move information with the position from where the player is making move and the direction where it is moving.

Here are using **struct moveinput** with variables(x,y,dir) where x and y is the block position and dir is the **direction** where it is moving i.e. (1,1) (0,1).

To create the board, we are using 4*4 array with two variables value and player where value is number of stones and player can be white or black.

We are also using some key functions as stated below…

getallmoves() will take board position and player as an argument and it will give list of all available moves for that position and player.

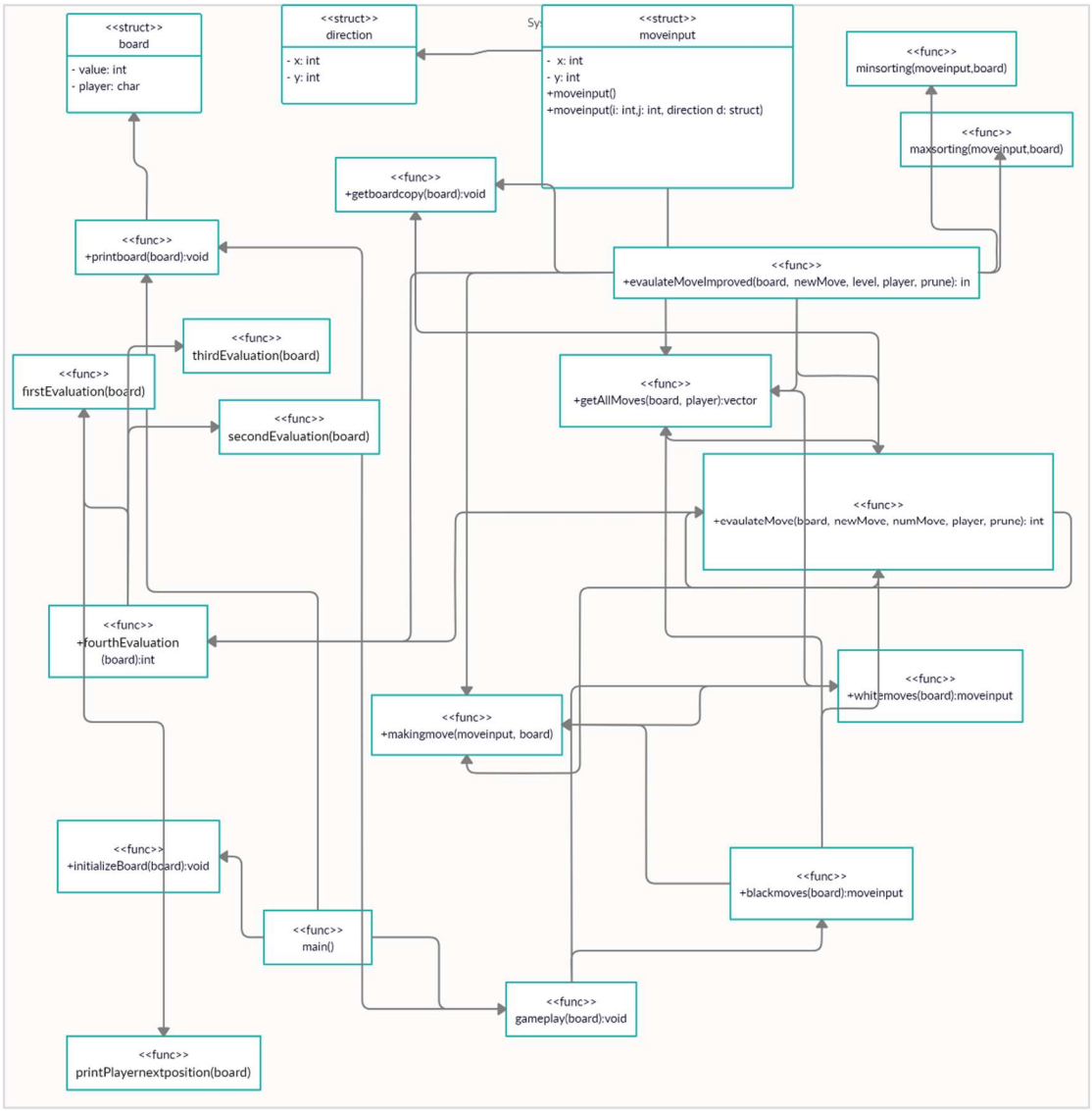
getboardcopy() will take original board and new empty board as an argument and copy it.
printboard() it will take board as an argument and print it.

minSorting() it will sort all the moves in ascending order according to its heuristic value

maxSorting() it will sort all the moves in descending order according to its heuristic value

UML diagram For the Conga Game

1..1



Enhancements in Practice:

We got the idea of sorting min nodes in ascending order and max nodes in descending order to enhance alpha-beta pruning from paper presented in IEEE by Jonathan Schaeffer(The History Heuristic and Alpha-Beta Search):

Though it is not every time improving efficiency of game as this game is not so complex but it helps sometimes.

We also went through literature regarding alpha-beta enhancements but we found that we should implement transposition table in-order to implement that enhancements which we are still trying to get unique values for each board position we are assigning 20 unique random numbers for each block which represents the 20 possibilities that block can have during game play. With this values we are representing the board position as unique integer value by performing XOR operation on random numbers of that particular board.

Evaluation functions:

Evaluation function will let us know the goodness of the next position so that we can move accordingly towards the winning position.

We tried playing game and we came up with below evaluation functions.

firstEvaluation():

First we noticed that spreading stones will lead to winning and we will have more possibilities to move so we came up with firstEvaluation().

It will count number of blocks occupied by the players and it will subtract number of blocks that has been held by white player from number of blocks that has been held by black player but when we used it the game was going into infinite loop sometimes.

secondEvaluation ():

Then we thought that if we choose position where opponent have less moves every time, we can lead to winning and also we will choose position where we have more moves, to implement this we came up with secondEvaluation().

It is checking that number of moves that player can play in next move, this function will subtract number of available moves for white from number of available moves for black.

thirdEvaluation ():

We also noticed that checking positions player can take in next move can help in deciding next move. According this we came up with thirdEvaluation ().

It will count the number of block positions player can take in next move. This function will subtract number of block white can occupy in next move from number of blocks black can occupy in next move.

fourthEvaluation ():

It is combined evaluation function of second and third evaluation functions.

fifthEvaluation ():

It is combined evaluation function of first, second and third evaluation functions.

Best Evaluation function:

As given in the [Table 1], We have done empirical assessment and evaluation of performance of evaluation functions through simulation.

Here we checked 10 game play outputs using depth 3,4 and 5. We are also checking parameters like

- moves - number of moves required to end the game,
- time – total time taken to end the game (microseconds),
- explored node – total number of node explored during the game
- pruning – number of node where search got ended.

It is clear from table that the first and second evaluation function are not much effective as it is taking more number of moves to end the game. Sometimes using second function game is ending in 18 moves but sometimes it goes into infinite loop.
Third heuristic is better, as average number of moves got reduced.
But when we combine these heuristics, we are getting much improved results as shown in the Table. Fourth and fifth heuristics are combination of above 3 heuristics and gives best results. But again, fifth heuristic sometimes goes into infinite loop and takes more moves to end the game compare to fourth heuristic.

Using depth 3 and 4 gives much faster results than using depth 5, given in the table(time is given in microseconds).

So we finalized that the **fouthEvaluation()** using **depth 4** is the best evaluation function in our case. Which gives best results like game ending in 10 moves and average moves to end the game is 38 according to [Table1].

Hardware Specifications:

All the experiments were performed on a PC with Intel® Core™ i7-10510U CPU 1.80 GHz, 2.30GHz 16GB memory, and Windows 10 64-bit operating system. Implementation, validation and testing are done on VisualStudio version 16.7.6

Outputs	Depth	Parameters	first(h1)	second(h2)	third(h3)	fourth(h2+h3)	fifth(h1+h2+h3)
1	3	moves	104	66	22	240	24
		time	7336	8588	1672	27702	2428
		explored node	234367	107956	13401	397628	32904
		pruning	4728	5152	1280	11954	1128
2	3	moves	256	48	78	10	20
		time	19152	6163	5698	536	1185
		explored node	541887	73633	96566	512	12161
		pruning	11669	3644	7189	333	947
3	4	moves	76	100	78	18	40
		time	4030	32984	5780	2298	6854
		explored node	586189	606494	727474	69612	173108
		pruning	93148	111527	82092	22454	60269
4	4	moves	228	100	76	38	76
		time	12279	32536	5234	8893	14912
		explored node	1685615	596656	865644	204822	273295
		pruning	311830	133110	67911	48947	74217
5	4	moves	infinite	36	80	46	86
		time		9854	5578	10324	31910
		explored node		182472	793125	251792	575580
		pruning		44289	74871	63349	107756
6	4	moves	290	68	32	50	54
		time	18414	23866	1998	9094	9063
		explored node	3298911	447399	278956	209860	242472
		pruning	663732	97082	28346	42203	73700
7	4	moves	infinite	62	74	38	16
		time		15925	5255	7094	2548
		explored node		362727	78543	171560	51870
		pruning		73742	75162	39860	9005
8	4	moves	116	18	78	78	52
		time	4870	3195	4318	16695	13512
		explored node	908250	69757	444933	407614	311170
		pruning	216869	22353	69508	87181	73106
9	5	moves	484	infinite	50	26	26
		time	248526		11544	68088	87072
		explored node	98676604		2975589	1866500	1878106
		pruning	1859786		141824	88777	89279
10	5	moves	386	infinite	290	48	infinite
		time	169317		125859	198284	
		explored node	68718687		35523454	4492960	
		pruning	1241715		1400756	92535	

[Table 1–assessment and evaluation of performance of evaluation functions though simulation]

Performance evaluation (Random Vs AI agent):

The outcome of 10 games played against the “Random agent” using the best evaluation function is given in the [Table 1], column 7 – fourth(h2+h3).

Time complexity: Given as the “explored node” which represents the number of node visited.
Response time: Given as the “time” which represents the total time taken to complete the game.
Amount of pruning: Given as the “pruning” which represents the number of nodes where search got ended and all the child nodes got pruned.

Output	Evaluation Function	Parameters	Alpha-Beta	Improved Alpha-Beta
1	Fourth	Moves	20	20
		Time	582	755
		Node Explored	12072	12072
		Prunning	947	947
2	Fourth	Moves	68	58
		Time	23776	5474
		Node Explored	447399	94827
		Prunning	970082	3162
3	Fourth	Moves	94	84
		Time	11754	5904
		Node Explored	152699	110666
		Prunning	6001	4722
4	Third	Moves	36	36
		Time	972	487
		Node Explored	23554	23554
		Prunning	1760	1760
5	Third	Moves	22	22
		Time	386	497
		Node Explored	13401	13401
		Prunning	1280	1280

[Table 2–assessment and evaluation of performance(AI agent Vs AI agent)]

Performance evaluation (AI agent(a-b) Vs AI agent(improved a-b)):

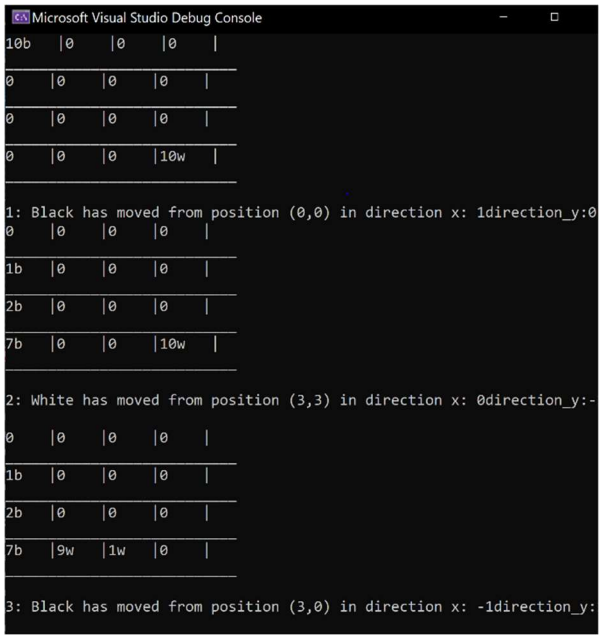
If we take both AI agent than game goes into infinite loop every time and never ends. Because both agents are smart, and they will not allow opponent to win. So, we thought of playing random agent Vs both AI agent and then comparing results.

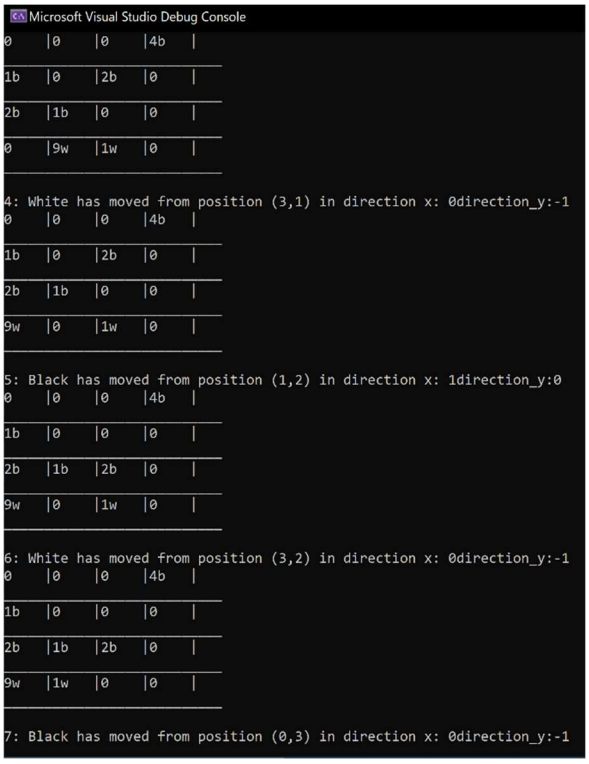
The outcome of 5 games played between the AI agent with alpha beta and AI agent using improved alpha bets using the best evaluation function and third evaluation function is given in the [Table 2].

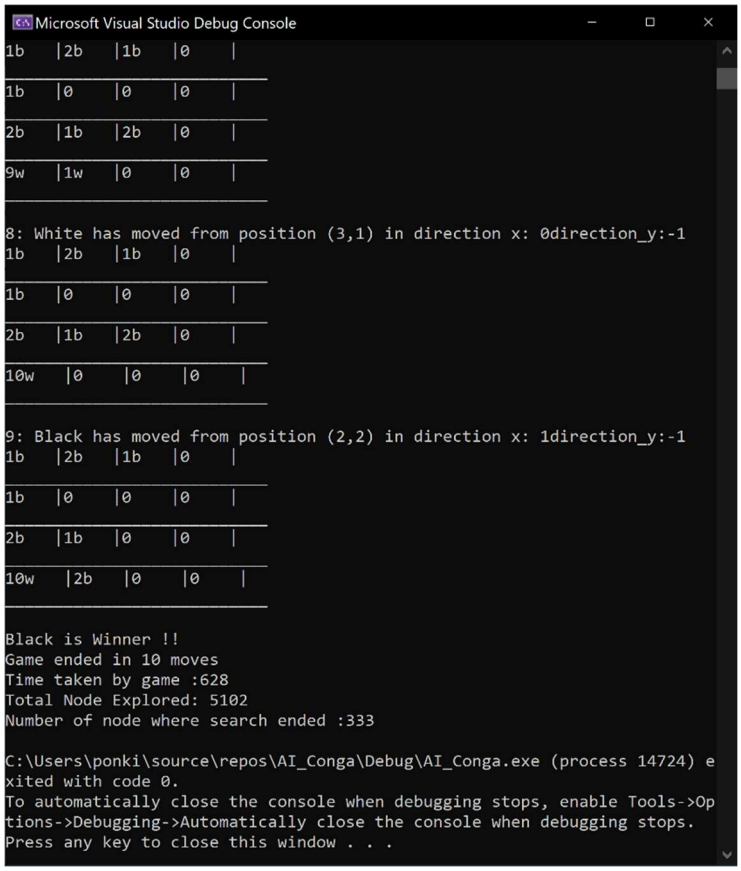
We can say that the improved alpha beta is not very efficient. Mostly it gives same result as the alpha beta. Sometimes it gets good results and sometimes it bad result. Check the shaded blocks in [Table 2], to check the differences we are getting.

Sample output of AI agent using the “best” evaluation function

Given below is the sample output of the game play. This is the best result we got with AI agent Vs Random Agent.
Here Black Player(AI agent) is taking move in such a way that it can block the white player.







Contribution :

We have brainstormed every functionality together.
Binaka taken care of implementation and testing – debugging.
Verification and validation are done by Bhanu.
We also prepared report together.

References

CUI-Jinling, ZHANG-Congpin. "Improved alpha-beta pruning of heuristic search in game-playing tree." *World Congress on Computer Science and Information Engineering* (2009).

https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/