**Breakthrough**

Python 3 was used to implement this game.

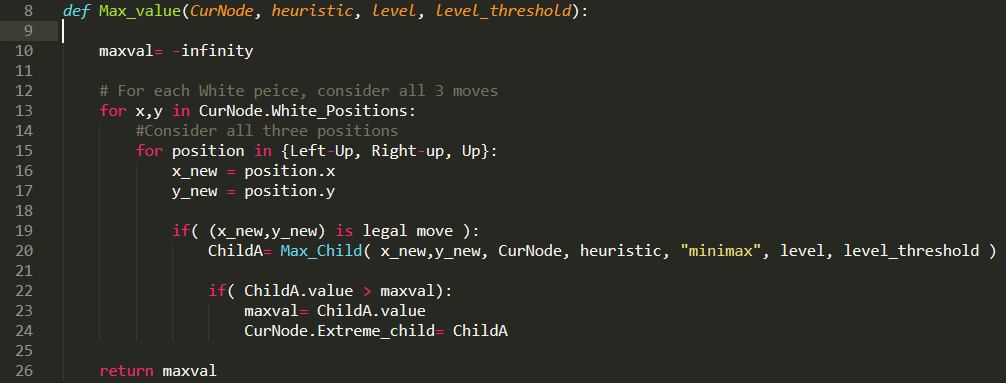
A **configuration of the board** is a node in a search tree. Each node is implemented as a class “State.”

**State** class contains members:

|  |  |  |
| --- | --- | --- |
| **Member name:** | **Type:** | **Purpose:** |
| Value | int | Stores Min or Max value of the State |
| White\_positions | List of (int,int) | 2-D coordinate of each white pieces |
| Black\_positions | List of (int,int) | 2-D coordinate of each black pieces |
| Subtree\_size | Int | Number of nodes expanded, by the State |
| Extreme\_child | Pointer to a State | Stores pointer to next optimal State |

In our implementation, **White player is Max-player** and **black player is Min-player.**

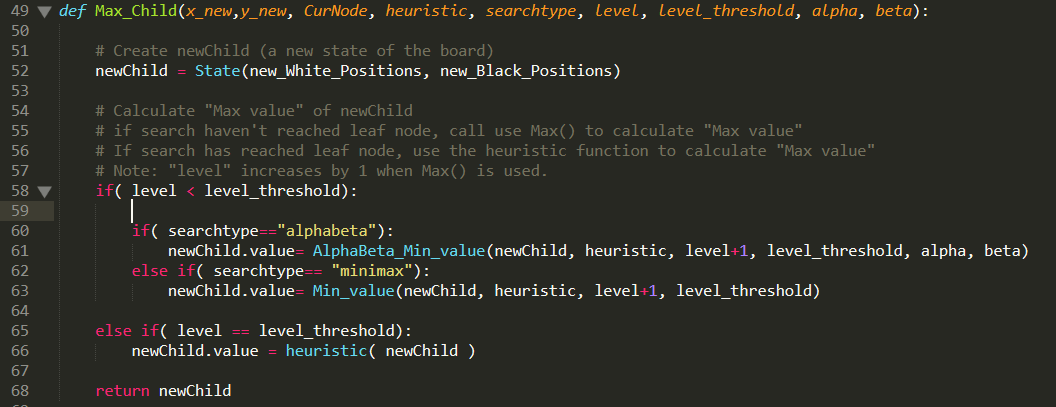
**Minimax search:**



**Max\_value(CurNode)** calculates max value of CurNode. The heuristic used in Max is defined in parameter. Search depth is defined by “level\_threshold”. The parameter “level” keeps track of level.

Flow:

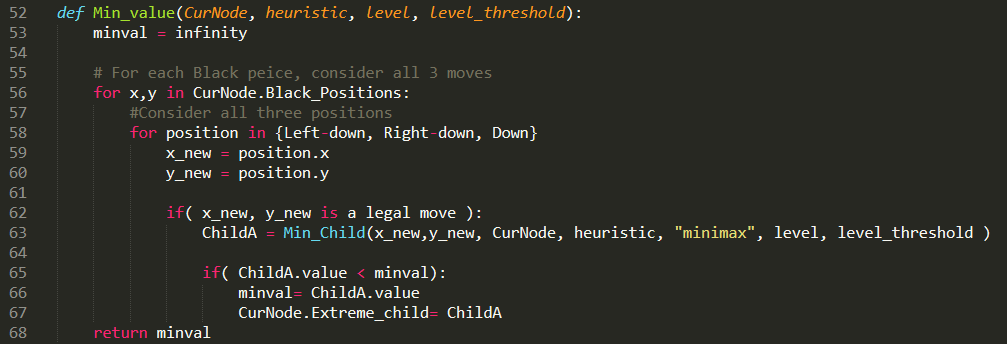
* For each white piece, each 3 moves are considered: Left-up, Right-up, and up (line 12-15). (The maximum possible number of moves is 16\*3=48)
* If the considered move is legal, a child node is created (line 19 and 20.) The child node is created using helper function Max\_Child(). (see below for pseudocode of Max\_Child)
* Each time a child node is created, the child node’s value is checked whether if it’s the new maximum. The maximum valued child is stored in CurNode.Extreme\_child (line 22 – 24.)



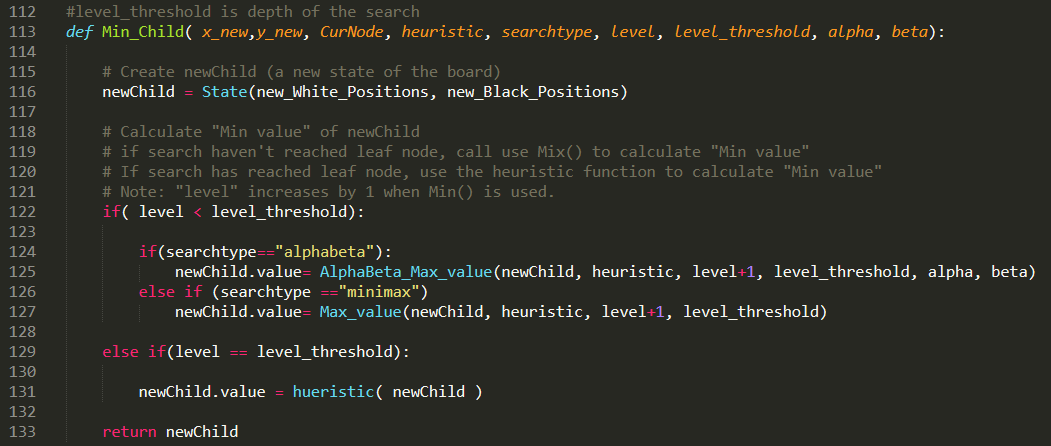
Max\_Child() creates a child state and calculates its min value.

Flow:

* A node representing new state of the board, newChild, is created. In this state, the positions of white and black pieces are updated (line 52.) (details are trivial and omitted in pseudocode.)
* The newChild’s min value is calculated by recursively calling Min() or using heuristic (if the node is a leaf) (line58-66.) When Min() is used, the level increases by 1. The newChild uses parent’s alpha and beta value for minimum calculation, in Alpha-Beta search.
* The newChild is returned (line 68.)

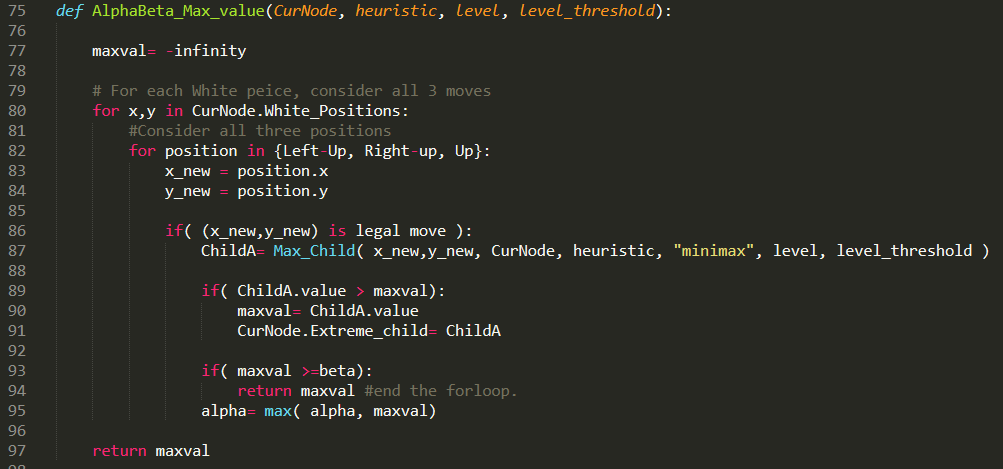


Min\_value() follows similar flow as Max\_value().



Min\_Child() follows similar flow as Max\_Child()

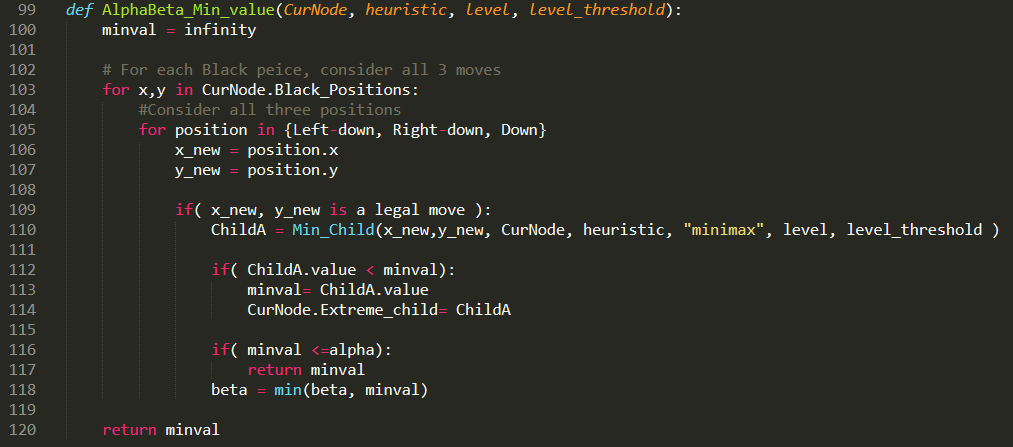
**Alpha-Beta Search**



Psuedocode for AlphaBeta\_MaxValue() is identical to Max\_value(), except 3 additional lines at 93-95.

Flow:

* The search tree is stops creating branches (search is pruned) if maxval exceeds beta (beta is the highest maxval can be.) (line 93-94)
* Alpha value is maximized if higher child value is found (line 95.)



Follows similar flow as AlphaBeta\_Max\_value()

**Running a game**



In a game, white player finds optimal move using Max and black player find optimal move using Min.

Flow:

* A game start with all black pieces at row 1 and 2, and all white pieces at row 7 and 8 (line 123) and white player starts (line 124.)
* If the game has not ended, players alternate taking moves (line 126.)
* The player color is identified then search type is identified.
* In alpha-beta search, depth of search is 4 (level\_threshold is 4) (line 130)
* After the search, state of the game is updated with optimal move (line 135) and the player is switched (line 137.)
* After game ends, the game status is printed.

**Heuristic Functions:**

Remark: White player chooses maximum heuristic value, while black player chooses minimum heuristic value.

In beginning of a game, black player’s pieces start at row 1 and 2, and white player’s pieces start at row 7 and 8.

**Offensive 2:**

Where:

Offensive2 strategy has two goals: minimize (# of black pieces) and place white pieces close to row 1, 2, or 3.

“*White offensive value”* measures how good white position is. When white pieces are at row 1, 2, or 3, white player has good chance of winning. Therefore, “weight” have high value in row 1,2, and 3.

**Defensive 2:**

1,1,1,1,1,1,5,10)

Defensive2 strategy has two goals: minimize (Black Threat value) and maximize (White defense value.)

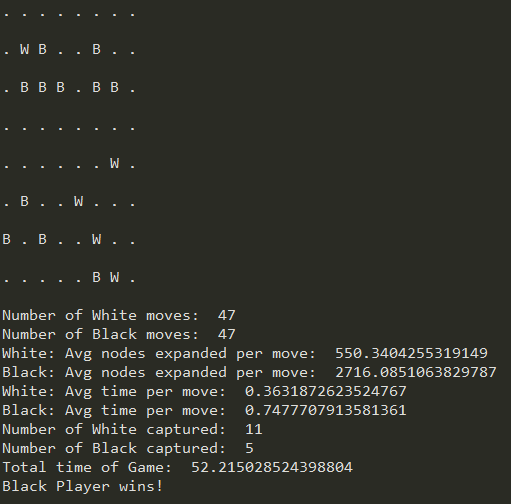
BTweight has highest values at row 7 and 8 because those are very threatening position to white player.

WDweight has highest values at row 7 or 8 because white pieces placed at row 7 or 8 forms good defense.

**Result**

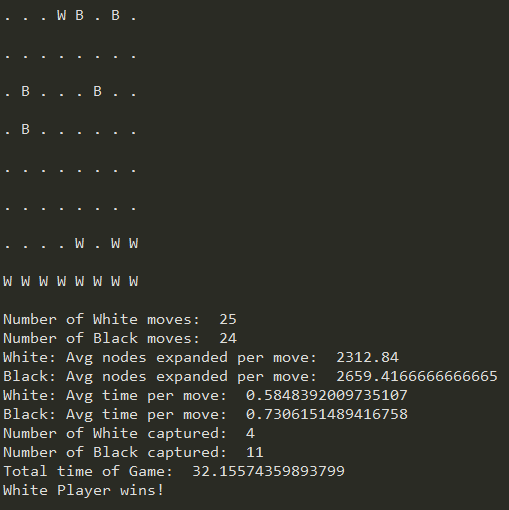
1.White Minimax (Offensive Heuristic 1) vs Black Alpha-beta (Offensive Heuristic 1)

Shown below:

****

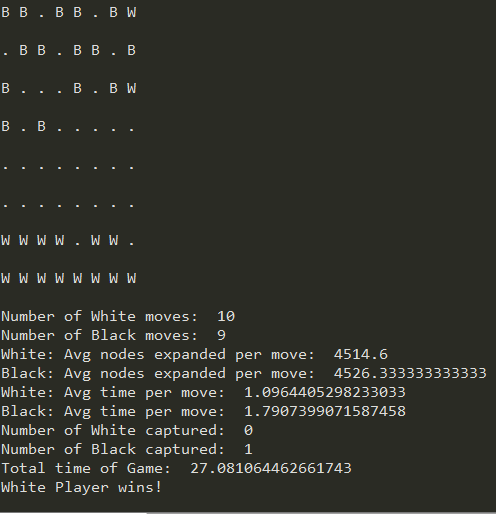
2.White Alpha-beta (Offensive Heuristic 2) vs Black Alpha-beta (Defensive Heuristic 1)

Shown below:



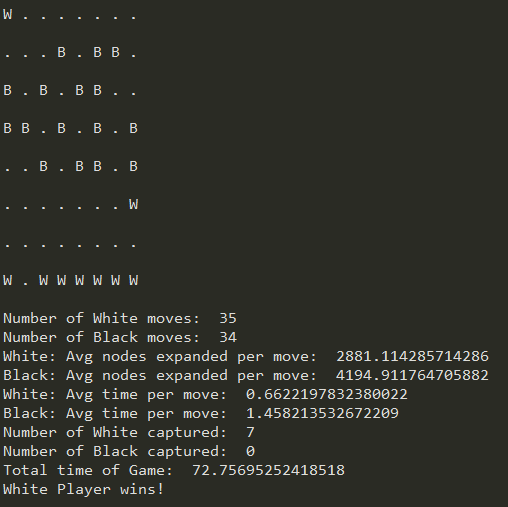
3. White Alpha-beta (Defensive Heuristic 2) vs Black Alpha-beta (Offensive Heuristic 1)

Shown below:



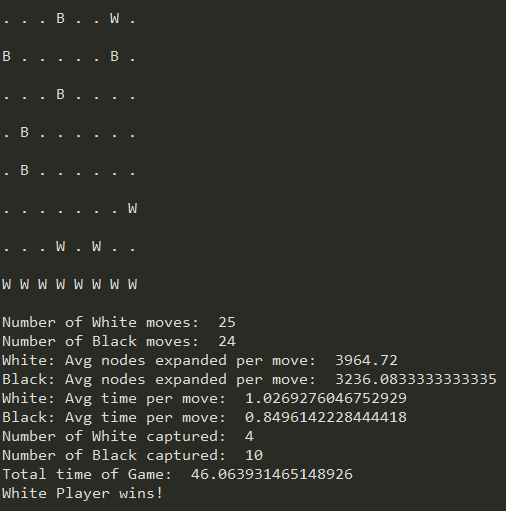
4.White Alpha-beta (Offensive Heuristic 2) vs Black Alpha-beta (Offensive Heuristic 1)

Shown below:



5.White Alpha-beta (Defensive Heuristic 2) vs Black Alpha-beta (Defensive Heuristic 1)

Shown below:



6.White Alpha-beta (Offensive Heuristic 2) vs Black Alpha-beta (Defensive Heuristic 2)

Shown below:

