

Introduction and Overview

Draughts, often known as checkers in the United States, is a collection of two-player strategic board games that feature diagonal moves of uniform game pieces and required captures by jumping over opponent pieces. Alquerque gave rise to draughts.

The word "draughts" comes from the verb "to draw" or "to move," while "checkers" comes from the checkerboard on which the game is played.

English draughts, commonly known as American checkers, is the most popular variant, which is played on an 8-by-8 checkerboard.

There is a lot of websites you can play this game on it as it is a one to one game although you can play against the computer

This program conducts an alpha-beta tree search, using forward pruning. To insure the effectiveness of the alpha-beta technique, the Checkers Player does a shallow, breadth-first search to order alternatives according to plausibility. The overall tree search terminates whenever a node is at a maximum depth and the program judges it to be “dead” (i.e., there are no immediate jumps available). During a game it usually takes the Checkers Player less than a minute to perform.

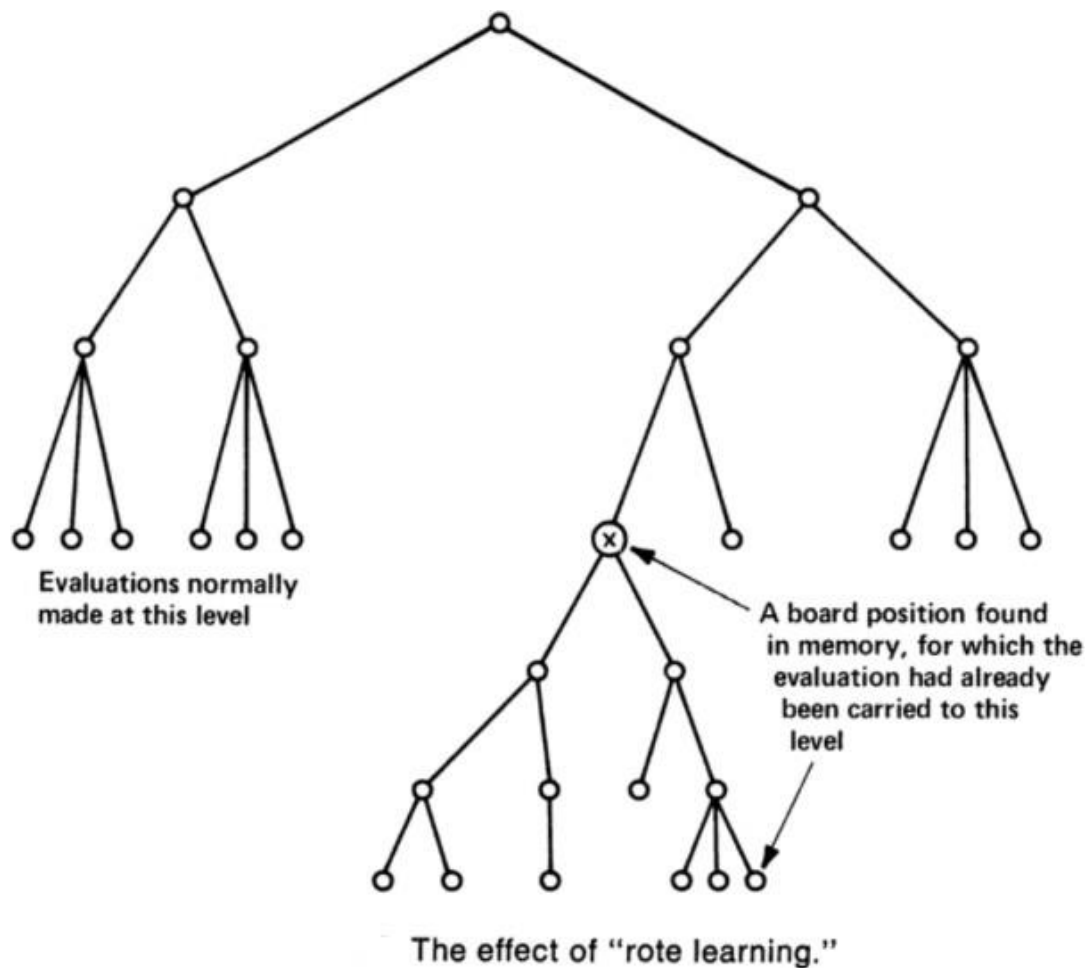
It will conduct a tree search and decide how it will move. The programme is unique in that it can create its own static-evaluation function to some extent.

- Learning

How these evaluation functions, whether polynomials or signature tables, are to be derived from the program's game-playing experience. The method for developing these functions is the program's "learning ability."

Work on the Checkers Player has primarily focused on two methods for accomplishing this: rote learning and learning by generalization. Rote learning can be accomplished by compiling a large database of board configurations and evaluations.

The creation of this file eliminates the need to recompute an evaluation each time such a configuration arises, which increases the program's efficiency (provided the search time through the file is kept low). Learning is influenced in the following ways: If the programme is successful,



- **Learning situations**

Alpha generalized its learning experience after each move and changed its coefficients accordingly, whereas Beta's polynomial evaluation function remained constant throughout any given game. The programme used against them was Alpha.

human opponents; the state of self-playing was effected: by playing

Alpha versus Beta, generally in a series of games, with the stipulation that if Alpha won a game, its polynomial would 'be used in the next game' by beta

while if Alpha lost too many games in a row

Alpha's polynomial would also suffer some large, random change if it lost too many games in a row. The goal of the change was to take the game in a different direction and (hopefully) allow the development of a completely new polynomial.

Alpha's polynomial was modified as follows: Alpha would compute the evaluation of the current board position as determined by its polynomial at each move. It would also compute a backed-up evaluation of the current board position, which would be determined by looking forward in the game.

Used Algorithms :

- **Alpha-beta**

Alpha-beta pruning is a modified version of the minimax algorithm. It is a technique for optimizing the minimax algorithm.

-In the minimax search algorithm, the number of game states to examine is proportional to the depth of the tree. We can't get rid of the exponent, but we can cut it in half. As a result, there is a technique known as pruning that allows us to compute the correct minimax decision without having to check each node of the game tree. Because this involves two threshold parameters, Alpha and Beta, for future expansion, it is referred to as alphabeta pruning. It is also known as the Alpha-Beta Algorithm.

It can be applied at any depth of a tree, and sometimes it not only prunes the tree leaves but also entire sub-tree.

Which has two parameters are :

1. Alpha: The best (highest-value) option we've found so far along the Maximizer path. The value of alpha at the start is $-\infty$.
2. Beta: The best (lowest-value) option we've found so far along Minimizer's path. Beta has a value of $+\infty$ at the start.

The Alpha-beta algorithm to a standard minimax algorithm returns the same move as the standard algorithm, but it removes all nodes that are not affecting the final decision but are slowing down the algorithm. As a result of pruning these nodes, the algorithm becomes faster.

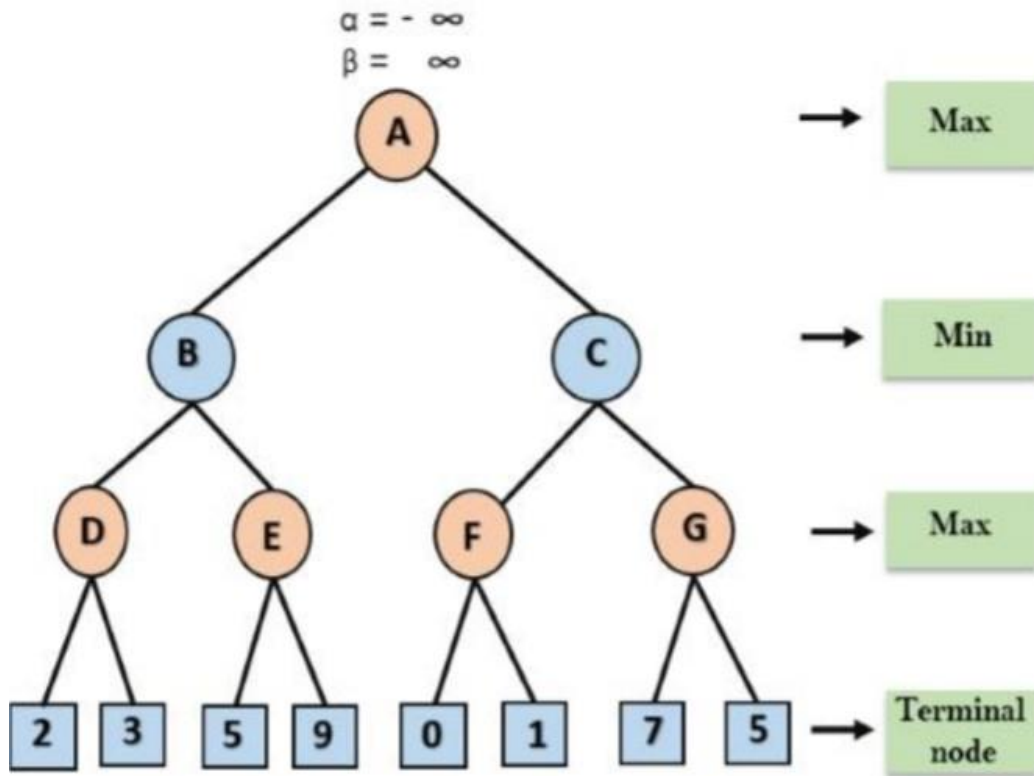
The main condition which required for alpha-beta pruning is: $\alpha \geq \beta$

- **Key points about alpha-beta**

- o The Max player will only change the alpha value.
- o The Min player will only update the beta value.
- o When backtracking the tree, node values will be passed to upper nodes instead of alpha and beta values.
- o Only the alpha and beta values will be passed to the child nodes.

- **How it works :**

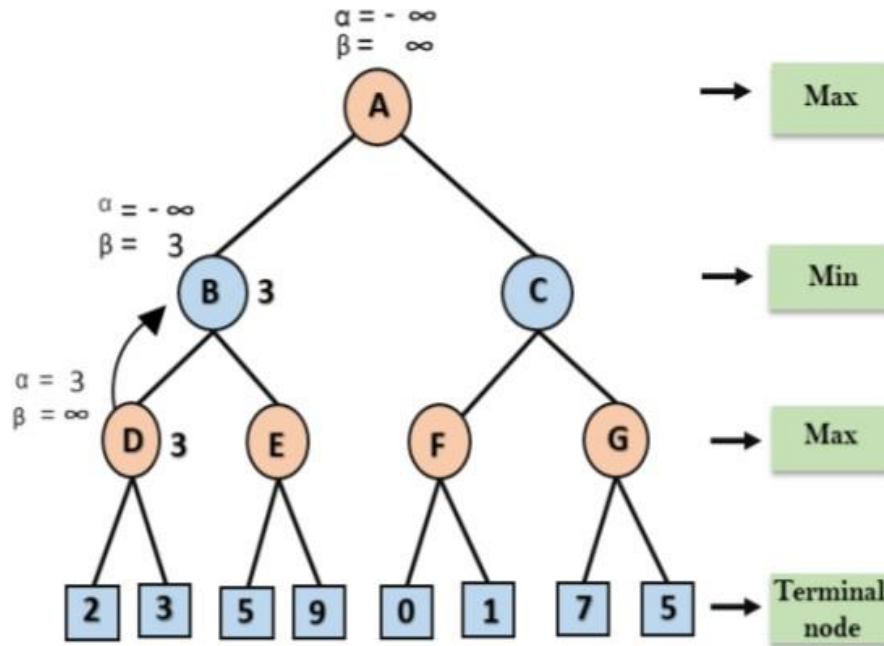
Step 1: In the first step, Max will begin by moving from node A where $\alpha = -\infty$ and $\beta = +\infty$, and these values of alpha and beta will be passed down to node B where again $\alpha = -\infty$ and $\beta = +\infty$, and Node B will pass the same value to its child D.



Step 2: At Node D, the value of will be computed as Max's turn. The value of is compared to 2, then 3, and the maximum $(2, 3) = 3$ will be the value of at node D, and the node value will also be 3.

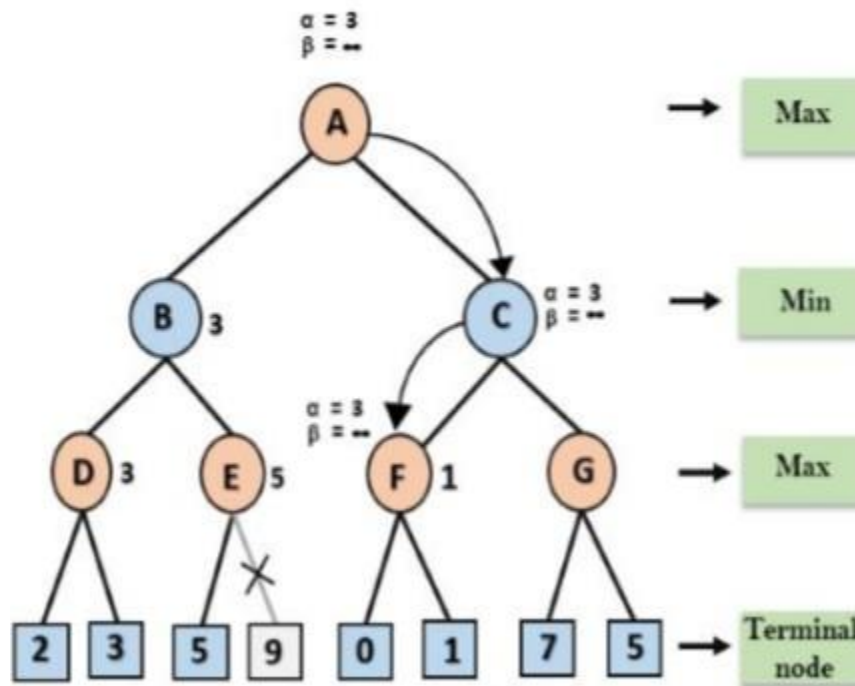
Step 3: The algorithm now backtrack to node B, where the value of β will change because this is a turn of Min

Now $\beta = +\infty$, will compare with the available subsequent node value $\min(\infty, 3) = 3$, so at node B now $\alpha = -\infty$, and $\beta = 3$.



In the following step, the algorithm traverses Node B's next successor, Node E, and the values of $\alpha = -\infty$, and $\beta = 3$ will also be passed.

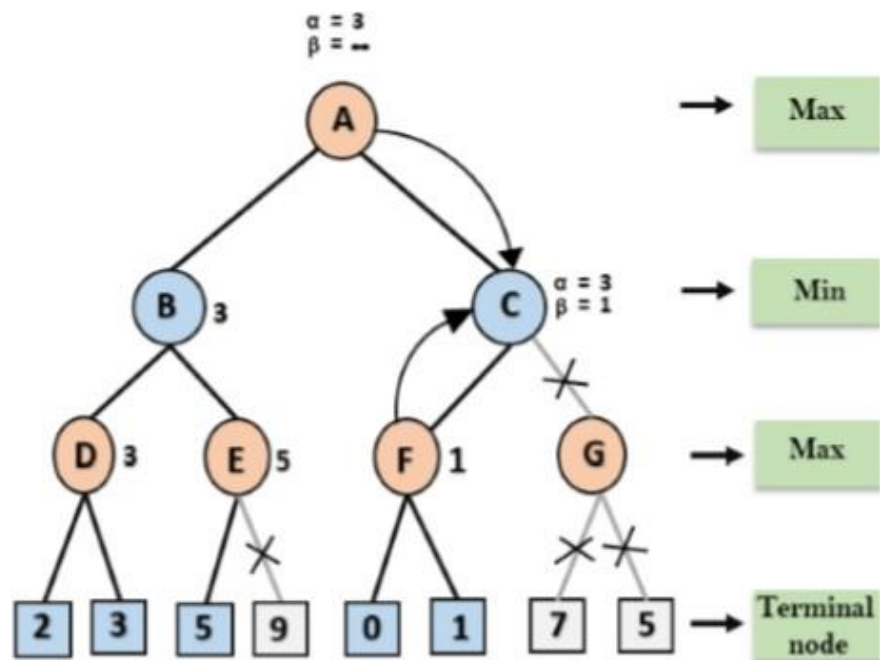
Step 4: Max will take its turn at node E, and the value of alpha will change. The current value of alpha will be compared with 5, so $\max(-\infty, 5) = 5$, which means that at node E $\alpha = 5$ and $\beta = 3$, where $\alpha \geq \beta$, the right successor of E will be pruned, and the algorithm will not traverse it, and the value at node E will be 5.



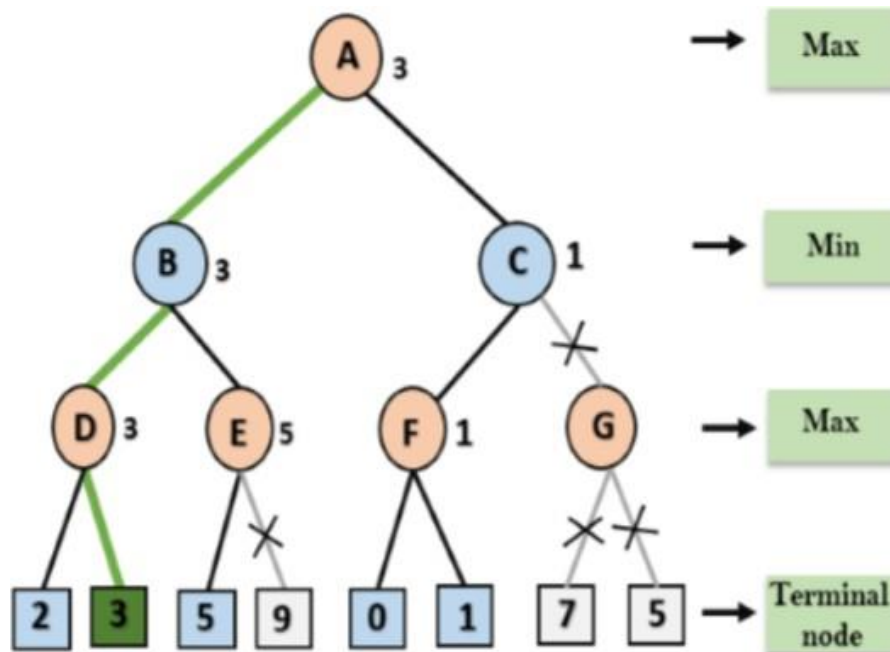
step 7: Node F returns the node value 1 to node C, at C $\alpha = 3$ and $\beta = +\infty$, here the value of beta is changed, and it compares with 1, $\min(\infty, 1) = 1$.

Now at C, $\alpha = 3$ and $\beta = 1$

it satisfies the condition $\alpha \geq \beta$, so the next child of (C,G) will be pruned, and the algorithm will not compute the entire sub-tree G.



Step 8: C now returns the value 1 to A; the best value for A here is $\max(3, 1) = 3$. The final game tree is shown below, with nodes that have been computed and nodes that have never been computed. As a result, the optimal value for the maximizer in this case is 3.



Results :

The effectiveness of alpha-beta algorithms is highly dependent on the order in which each node is examined. The order of moves is an important aspect of alpha-beta pruning.

It can be of two types:

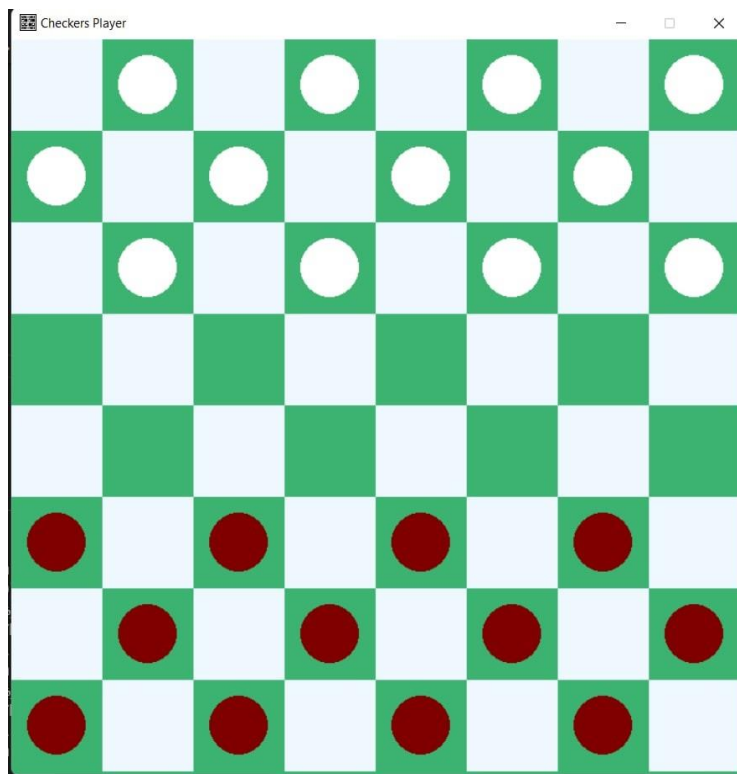
Worst ordering and Ideal ordering which are the following

1. In some cases, the alpha-beta pruning algorithm does not prune any of the tree's leaves and works exactly like the minimax algorithm. In this case, it also takes more time due to alpha-beta factors; such a pruning move is known as worst ordering.
The best move in this case occurs on the right side of the tree. The time complexity for such an order is $O(b^m)$.
2. When there is a lot of pruning in the tree and the best moves happen on the left side of the tree, this is the ideal ordering for alpha-beta pruning. We use DFS, so it searches the left side of the tree first and then goes deep twice as the minimax algorithm in the same amount of time. Complexity in ideal ordering is $O(b^{m/2})$.

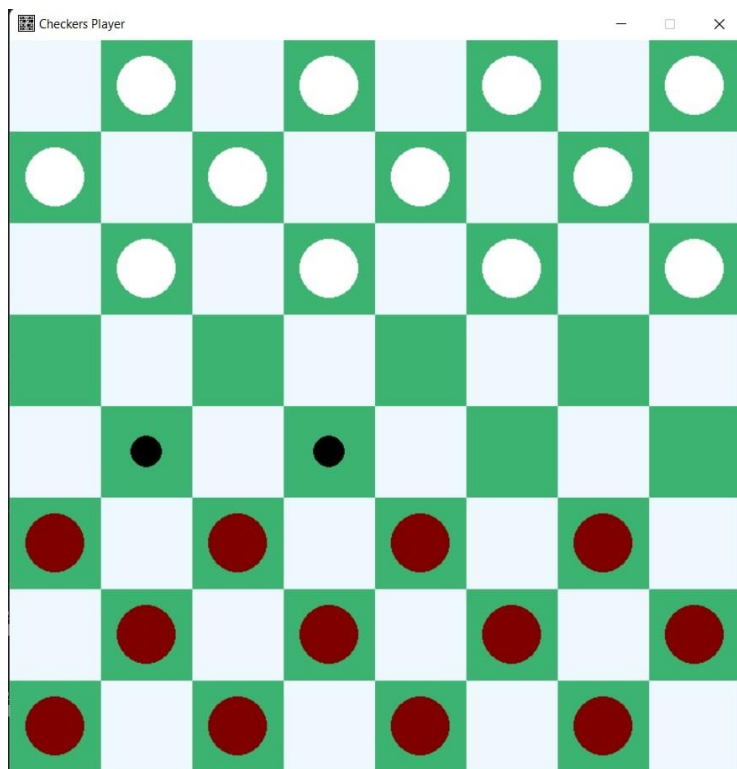
The following are the rules for finding good ordering:

- The best move will occur from the shallowest node.
- Sort the tree nodes so that the best nodes are checked first.
- When determining the best move, make use of your domain knowledge. For example, in chess, try this order: capture first, then threats, then forward moves, then backward moves.
- We can keep backtrack of the states because there is a chance that they will occur again.

-Input example of the program



While you are playing it shows for you all the possible move for each piece

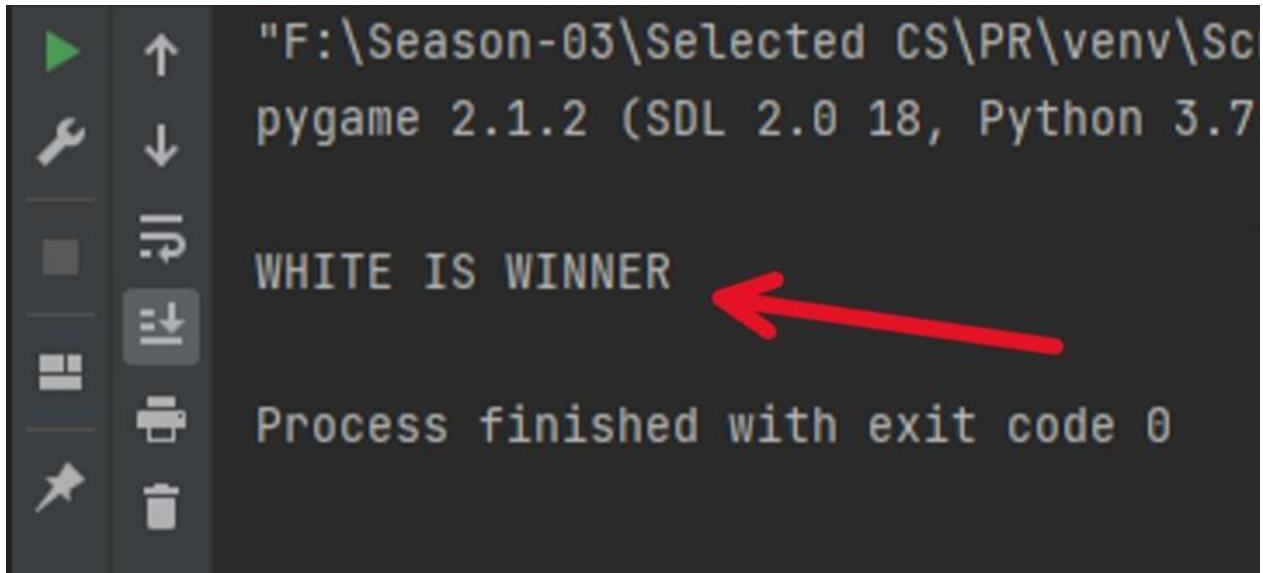


-Output example for the program

the depth is dependant how many solutions the program search in them (in scale)

So it may take a long time waiting for the program to make a move.

At the end of the game the program shows which player has won.



A screenshot of a terminal window with a dark background. On the left is a vertical toolbar with icons for running, navigating, and managing the terminal. The output text is as follows:

```
"F:\Season-03\Selected CS\PR\venv\Sc  
pygame 2.1.2 (SDL 2.0 18, Python 3.7  
  
WHITE IS WINNER  
  
Process finished with exit code 0
```

A red arrow points from the right side of the terminal towards the text "WHITE IS WINNER".