Documentation:

The Program has 2 main Classes: PentagoBoard and Player. It simulates a Pentago game between 2 players: computer/player. It an implementation of the Mini-Max Algorithm based on a specific heuristic for evaluating each node.

Pay Attention: The program uses a "numpy library"!

```
python3 -m pip install numpy
```

Heuristic:

Occurrence x_i	Weight w_i
Token at margin	0
Token at the center	5
3 in a row/col/diagno	100
4 in a row/col/diagno	1,000
5 in a row/col/diagno	9,999

Occurrence y_j	Weight w_j
Token at margin	0
Token at the center	5
3 in a row/col/diagno	100
4 in a row/col/diagno	1,000
5 in a row/col/diagno	9,999

Xi – represents our tokens , Yj – represents the opponent's tokens.

: $h(x) =$ $\sum x\text{i} * w\text{i} - \sum \text{yj} * w\text{j}$

Tokens next to the center of the board get a higher rating than tokens touching the board margin - they have more possibilities to be next to other tokens.

3 tokens in a row/col/diagno get even a higher rating since they are getting us closer to the goal.

4 tokens in a row/col/diagno get much more higher rating since the distance of the goal is at least 1.

5 tokens in a row/col/diagno state are winning the game. Hence, it gets the highest value.

Omri Yenon

Functions used in the Program:

Main Function – Creates a Pentago Board according to given input, 2 players (can be human or computer) and simulates the game by a main while that apply a player move each iteration until someone win or there is a tie.

getComputerMove – The best valuable move is chosen according to minimax / minimax2 algorithm functions.

win – given a specific state, a board, the function returns true if there are 5 tokens in a row/column/diagno – means that we won the game.

lost – given a specific state, a board, the function returns true if there are 5 of the opponent's tokens in a row/column/diagno – means that we lost the game.

miniMax – Given a current state, a board with tokens placed on it, looks k steps ahead and finally return a move and a "backed up value". This is a recursive function deepening in the tree and returns ∞ , -∞, 0, or h(state) according to the board state and depth was reached so far. The function calls functions win, lost in order to stop the recursion in a case of winning/losing. In addition, it returns h(state) when the depth equals to maxDepth.

minimax2 – Given a current state, a board with tokens placed on it, looks 2 steps ahead and return a move and a "backed up value". This function is calling win function and h function to evaluate our and the opponent's nodes. This is in order to choose the best move for us.

Omri Yenon

Outputs:

For an empty board, where player A and player B both are computers:

```
Player A: type=computer, plays Black tokens
Player B: type=computer, plays White tokens

+-------+-------+
| . . . | . . . |
| . . . | . . . |
| . . . | . . . |
+-------+-------+
| . . . | . . . |
| . . . | . . . |
| . . . | . . . |
+-------+-------+
```

After 25 steps there is a tie:

```
+-------+-------+
| . b w | . b . |
| . b w | w w w |
| b b b | b w . |
+-------+-------+
| . b . | w . . |
| w b w | b w . |
| w w . | b b b |
+-------+-------+

Game ends in a tie (multiple winners).
```

For a given board where there is an advantage for the White player – B :

```
Player A: type=computer, plays Black tokens
Player B: type=computer, plays White tokens

+-------+-------+
| . . w | . b . |
| . b . | . . b |
| . . . | w w w |
+-------+-------+
| . . . | . w . |
| . . b | . . b |
| . . . | . . . |
+-------+-------+
```

After 6 steps, Player B (white color is winning):

```
Placing w in cell [1][3], and rotating Block 4 Left
+-------+-------+
| b . . | . b . |
| b b b | w . b |
| w . . | w w w |
+-------+-------+
| . . . | w b . |
| . . . | w . . |
| . b . | w . . |
+-------+-------+

B (White) wins
```

Omri Yenon