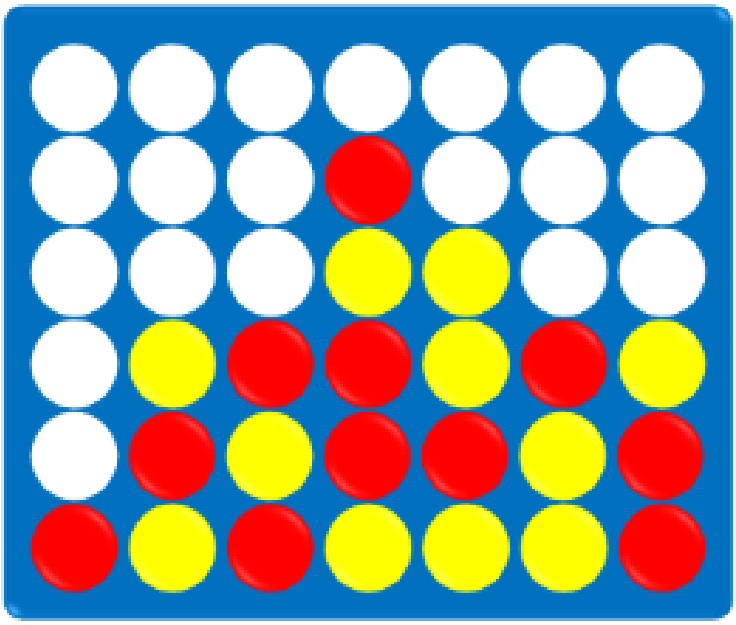
AI

Connect4 Project



Project done by:

Mohamed Mohab Elbattawy	**ID:7206**
Ahmed Hany Ismail	**ID:7248**
Youssef Sameh Fayek	**ID:6911**
Mostafa Mohamed Elkassas	**ID:6992**

Project overview:

A python written project, it was required to create connect 4 game where the players are human VS computer agent.

Connect-Four is two-player game in which players alternately place pieces on a vertical board 7 columns across and 6 rows high in our project its human VS computer as mentioned.

Each player uses pieces of a particular color (green and red), and the object is to obtain maximum number of four pieces in a horizontal, vertical, or diagonal line.

Because the board is vertical, pieces inserted in a given column always drop to the lowest unoccupied row of that column. As soon as a column contains 6 pieces, it is full and no other piece can be placed in the column.

Problem Approach

The main complexity was that a full game tree is very huge O(10^35) so it is impossible to traverse .So we had to design a suitable heuristic function that evaluates the state of a game and returns a number indicating whether the computer is near to win or lose.

In our project it was done using the following algorithms:

- **Minimax without alpha-beta pruning**
- **Minimax with alpha-beta pruning**

Functions used In the main code:

- get1_h()
- get_heuristic()
- get_neighbours()
- minimax()
- alphabeta()
- get_neighbourstree()
- minimaxtree()
- alphabetatree()
- gettree()

Functions used in GUI of the tree:

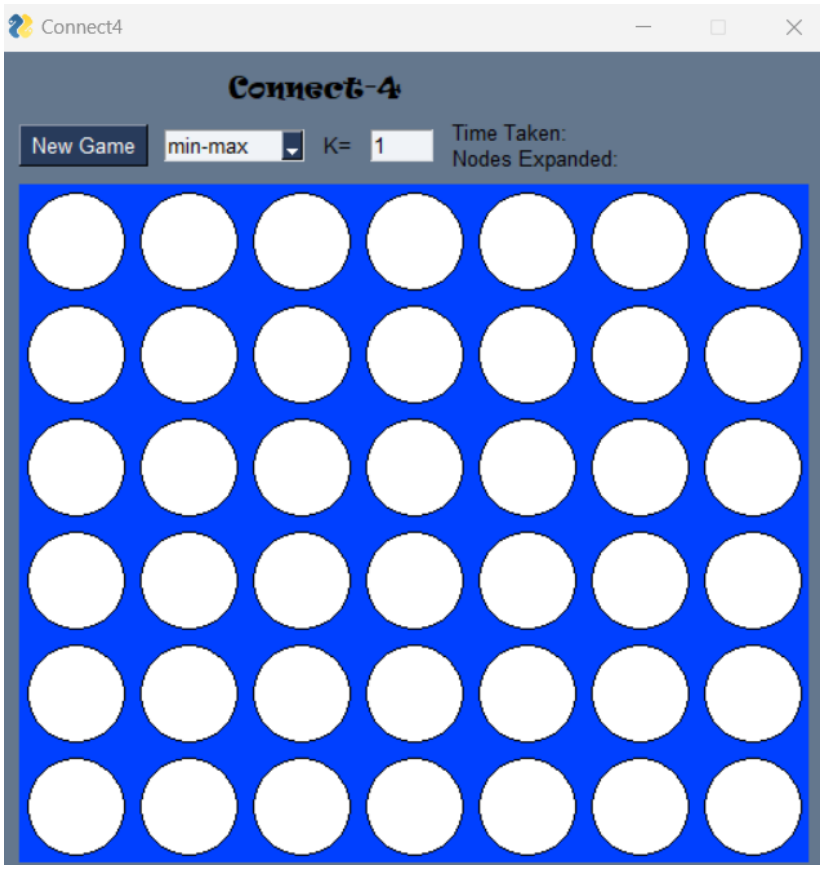
Our GUI was implemented using PySimpleGui

- showtree()
- drawtree()

User interface

The game starts with an empty board where the user plays first.

The user can choose the algorithm he wishes to use from a drop down menu



As well the user can choose K at different levels.

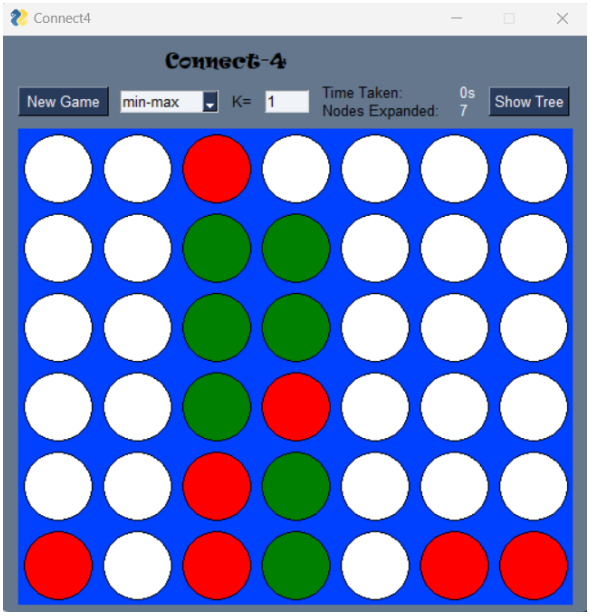
Also the program shows the following:

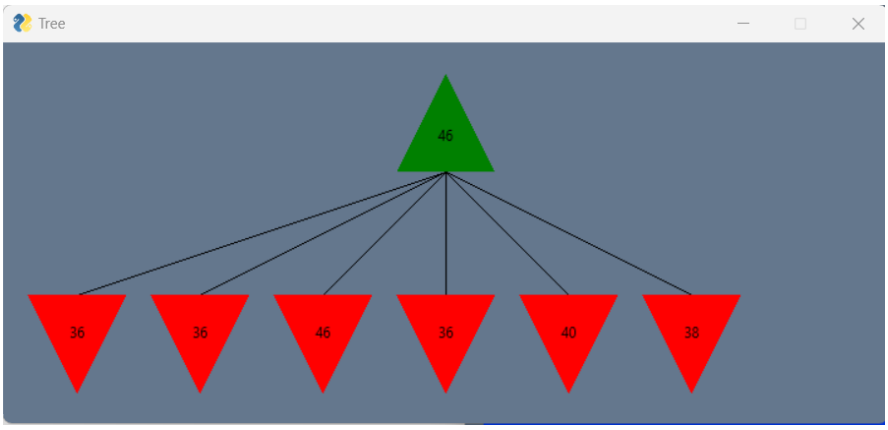
- Nodes expanded
- Time taken
- Score
- Minimax Tree

Which will be different according to each algorithm used to play the game.

Sample runs and their corresponding minimax trees:

1)k=1 without alphabeta pruning at an instance:

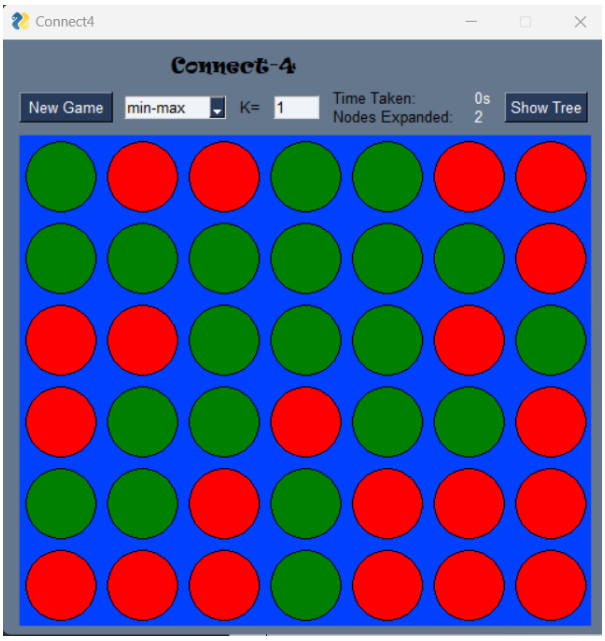


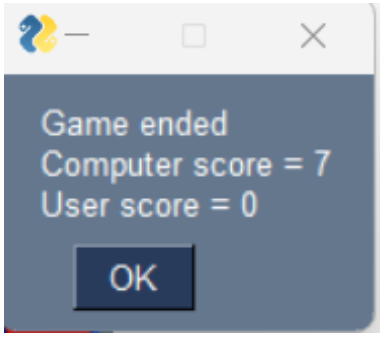


Nodes expanded:7

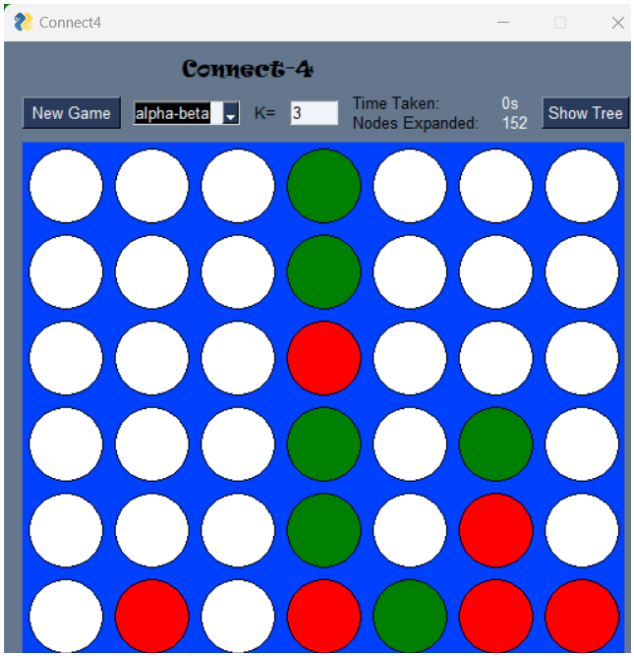
Time taken:0s

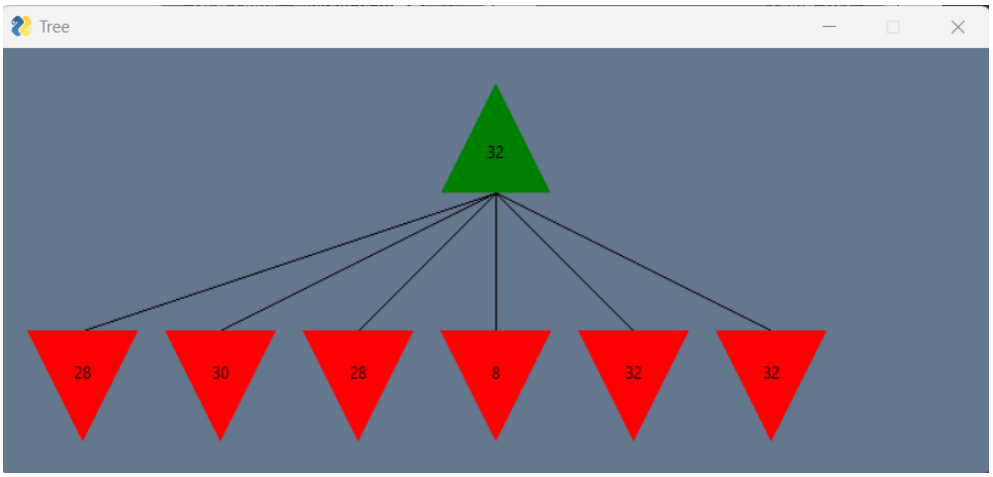
Final score:





2)k=3 with alphabeta pruning at an instance:

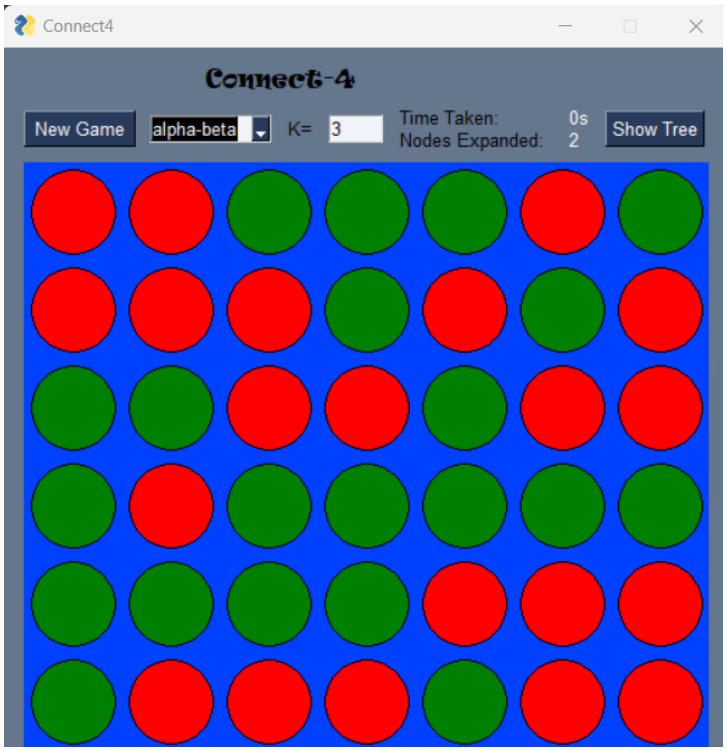


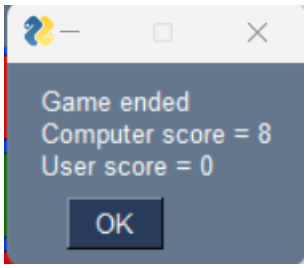


Nodes expanded:152

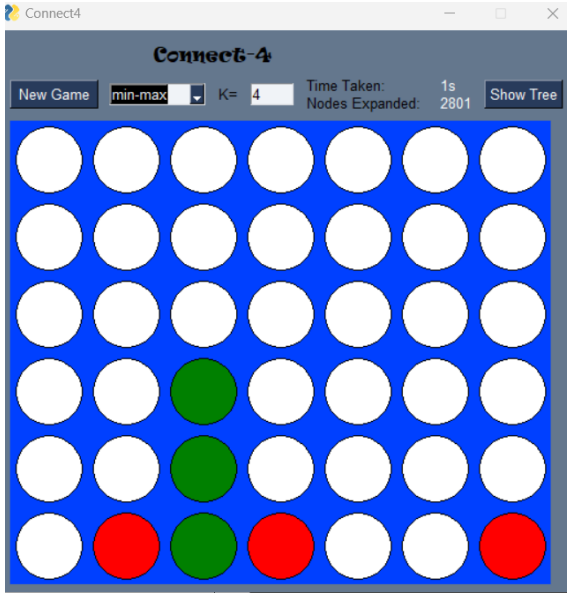
Time taken:0s

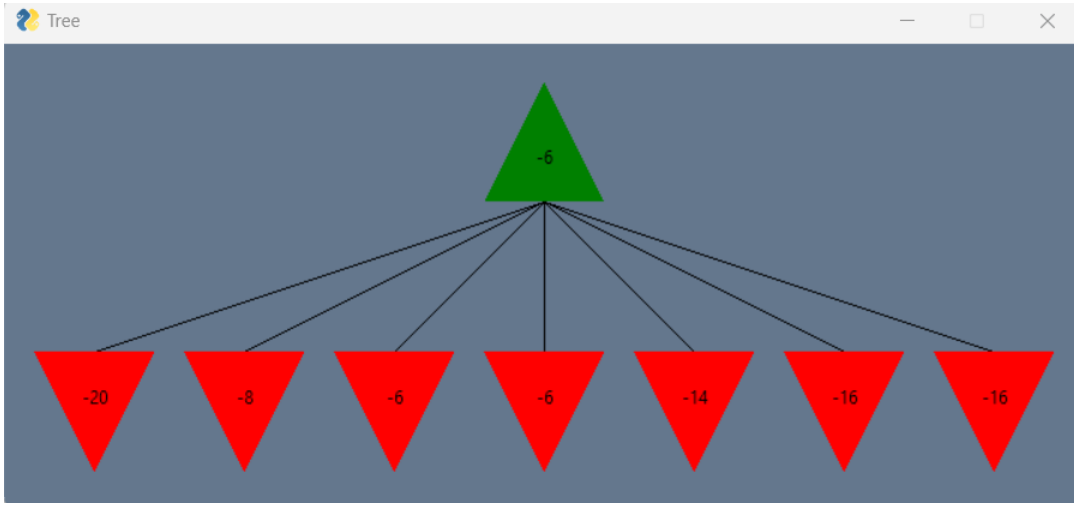
Final score:





3)k=4 without alpha-beta pruning at an instance:

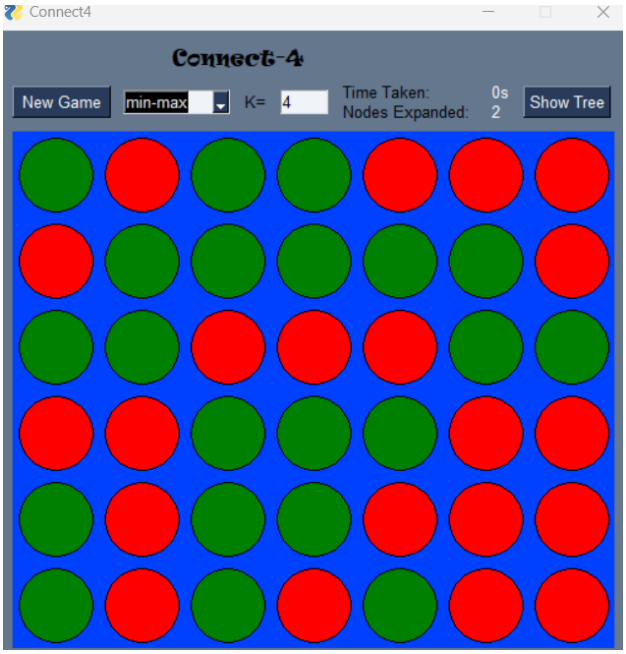


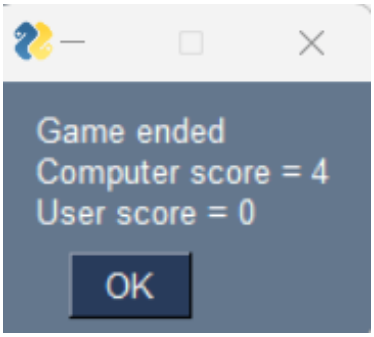


Nodes expanded:2801

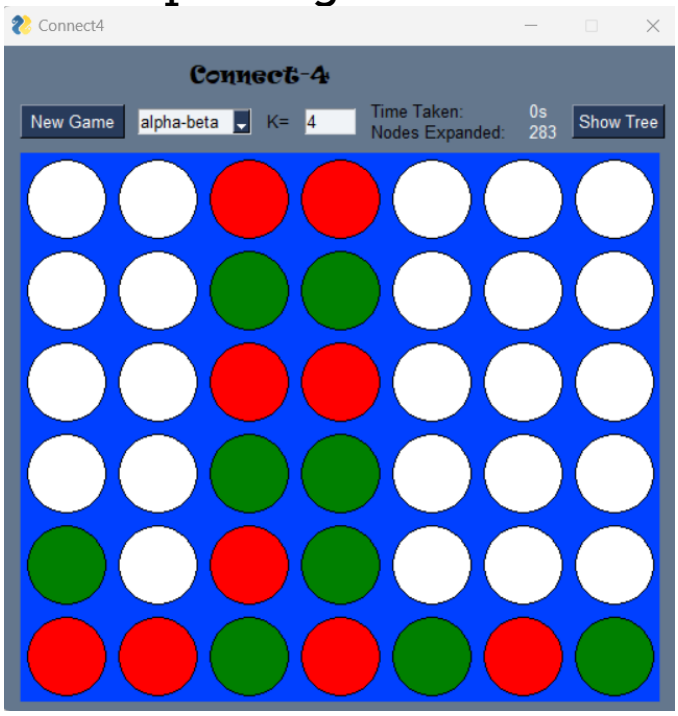
Time taken:1s

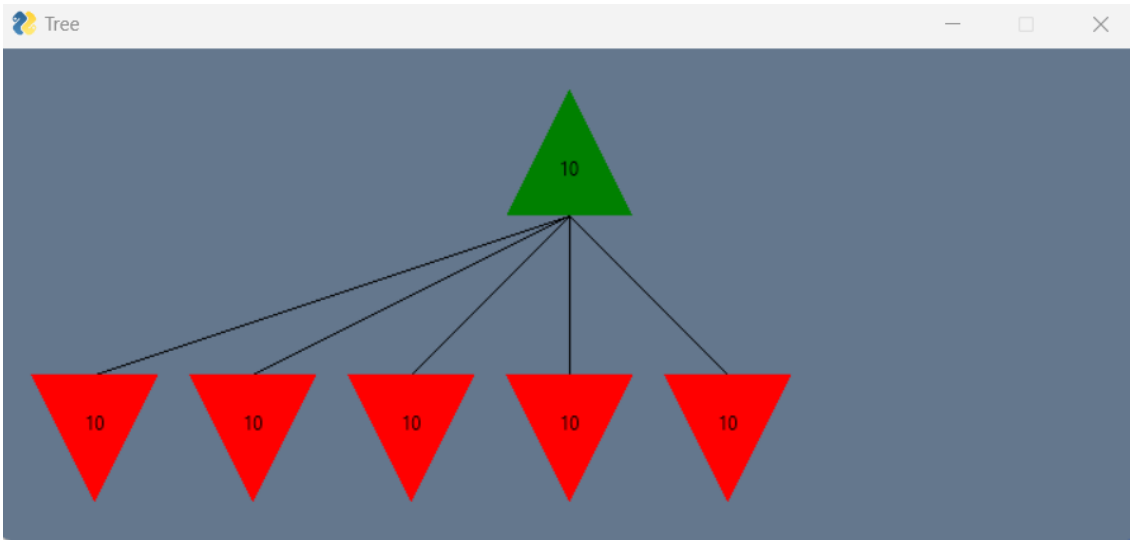
Final score:





4)k=4 with alpha-beta pruning at an instance:

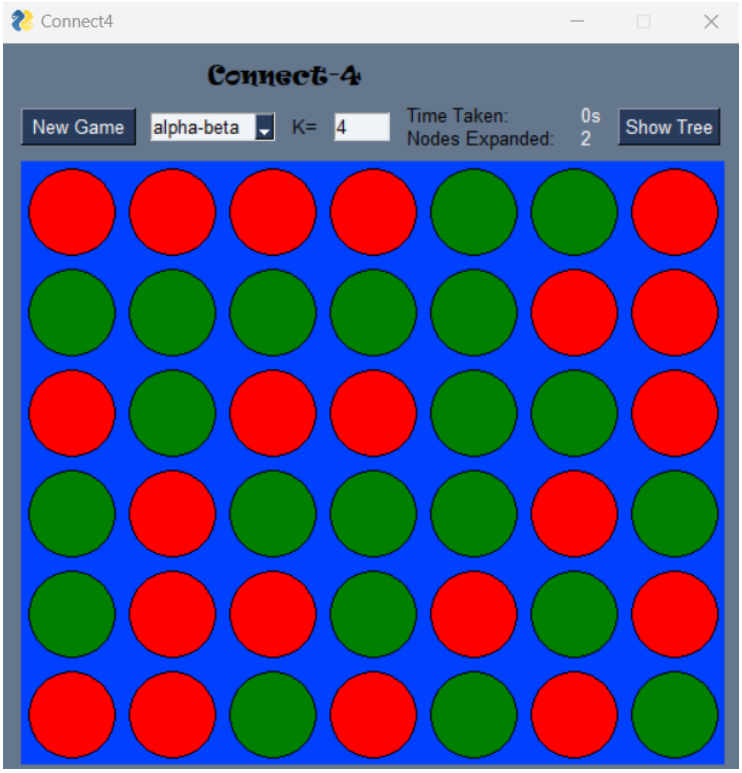


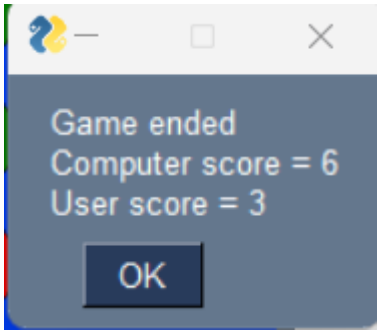


Nodes expanded:283

Time taken:0s

Final score:



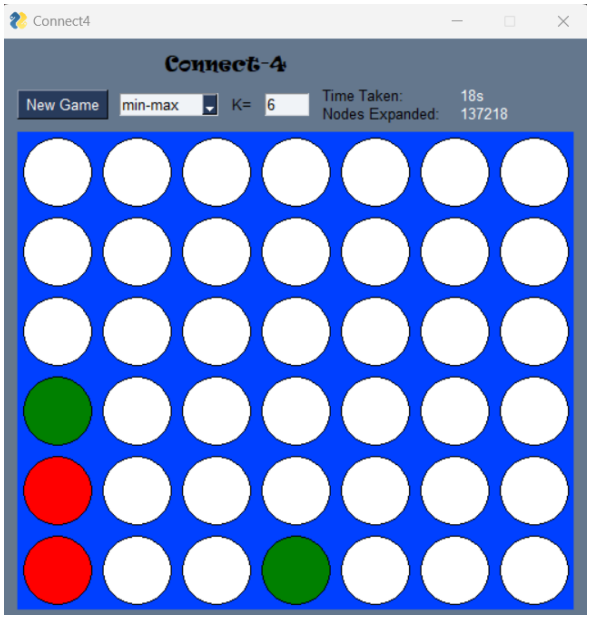


Comparison between the 2 algorithms in terms of the following:

- Time taken
- Nodes expanded at different K values

1)at k=6

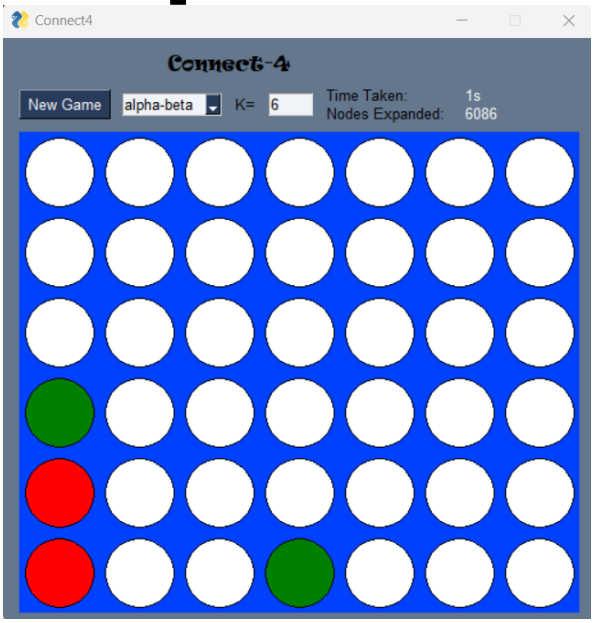
Min-max



Time taken=19s

Nodes expanded=137218

Alpha-beta

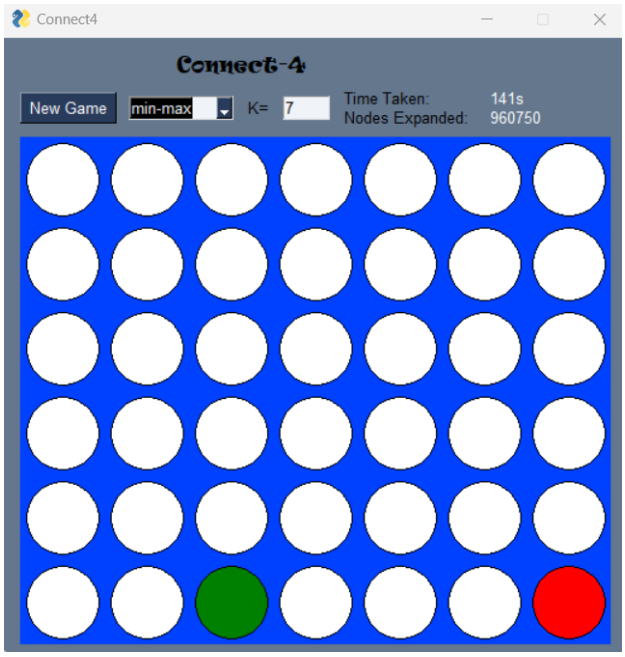


Time taken=1s

Nodes expanded=6086

2)at k=7

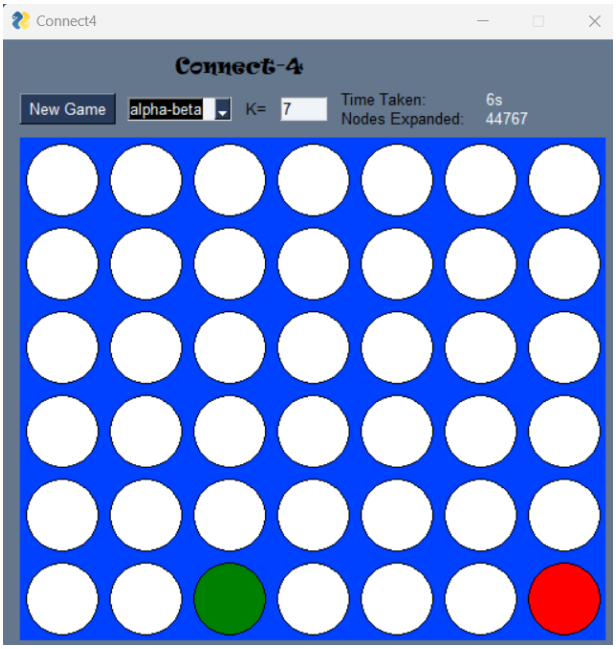
Min-max



Time taken=141s

Nodes expanded=960750

Alpha-beta



Time taken=6s

Nodes expanded=44767

Notes:

After comparing between the two algorithms we can find the following:

- Time in alpha-beta is better, specially at high k levels.
- Nodes expaned in alpha-beta are much less than that of min-max.

Data Structures used:

- Bitarray
- deque