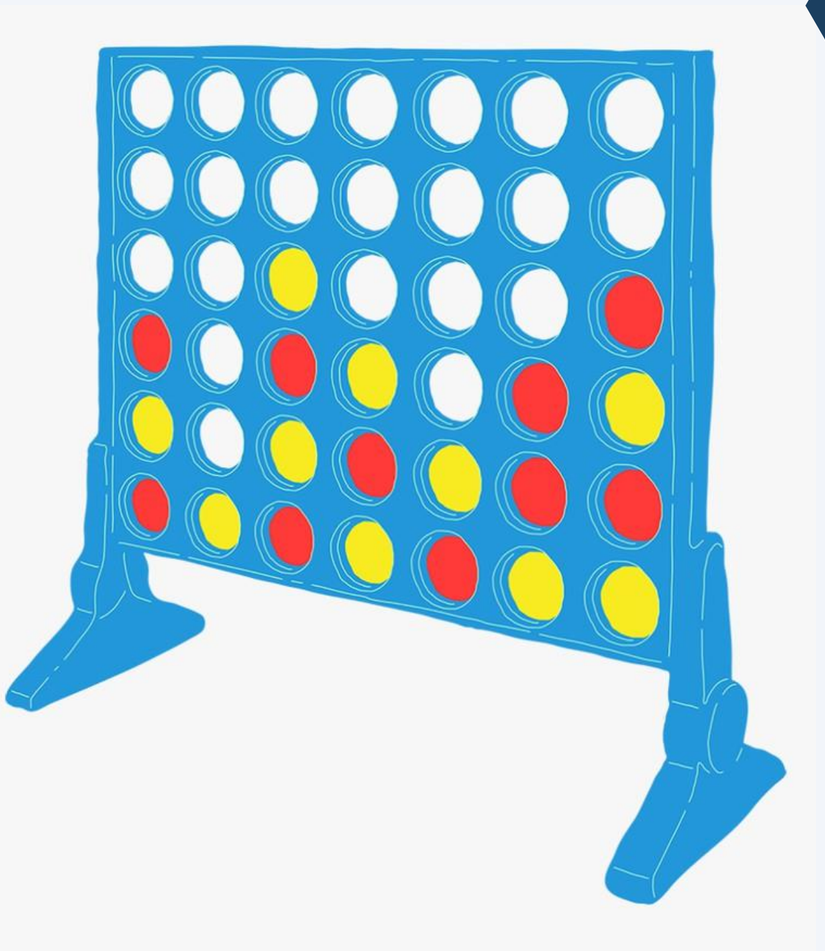
CONNECT 4

Piya Bhalla	102103233
Nimisha Gujral	102203894
Alisha Sood	102103638



Submitted to:

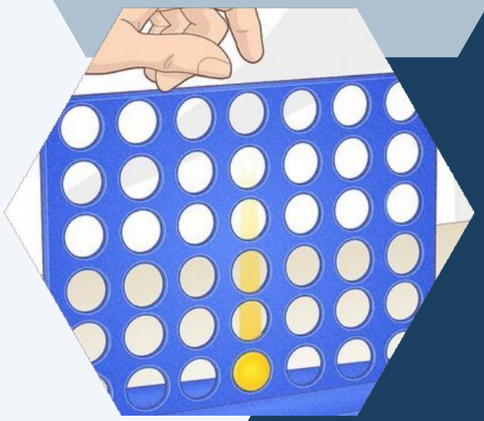
Dr. Simran Setia

About the Project

Connect Four is a two-player strategy game played on a vertical board consisting of a grid of seven columns and six rows. The game is typically played with counters of two colors, and the players take turns dropping their discs into the columns, one at a time. The objective of the game is to be the first player to connect four of their discs vertically, horizontally, or diagonally. In the following project of AI, the player would be able to play this game with the computer.

Rules for the game:

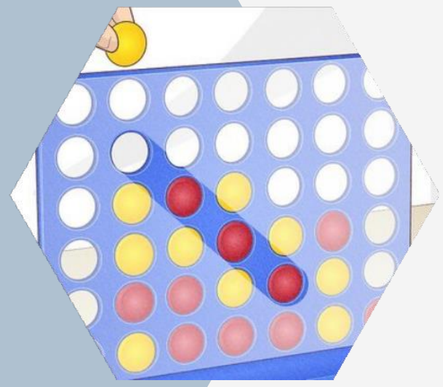
The game is played by two players who take turns dropping colored disks into the columns of the board from the top.



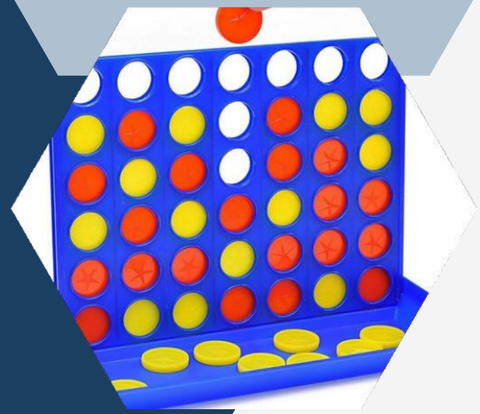
The disks must be dropped into an empty slot at the top of the column, and they will fall to the bottom of the column or onto another disk that is already in the column.

Players can only drop one disk per turn, and they must alternate turns throughout the game.

Rules continued:



Players must be careful to watch for potential winning moves by their opponent and try to block them.



If all of the columns are filled and no player has connected four disks in a row, the game is considered a tie.

The game ends when one player successfully connects four of their own colored disks in a row, either horizontally, vertically, or diagonally and that player is the winner.

Methodology

Objective:
The objective of Connect 4 is to be the first player to form a horizontal, vertical, or diagonal line of four of one's own discs.

Equipment:
A grid with 7 columns and 6 rows.
21 red discs and 21 yellow discs (or any two distinct colors).

Setup:
Place the empty grid between the two players.
Decide which player will use which color discs.
Determine which player goes first (can be done by coin toss, mutual agreement, etc.).

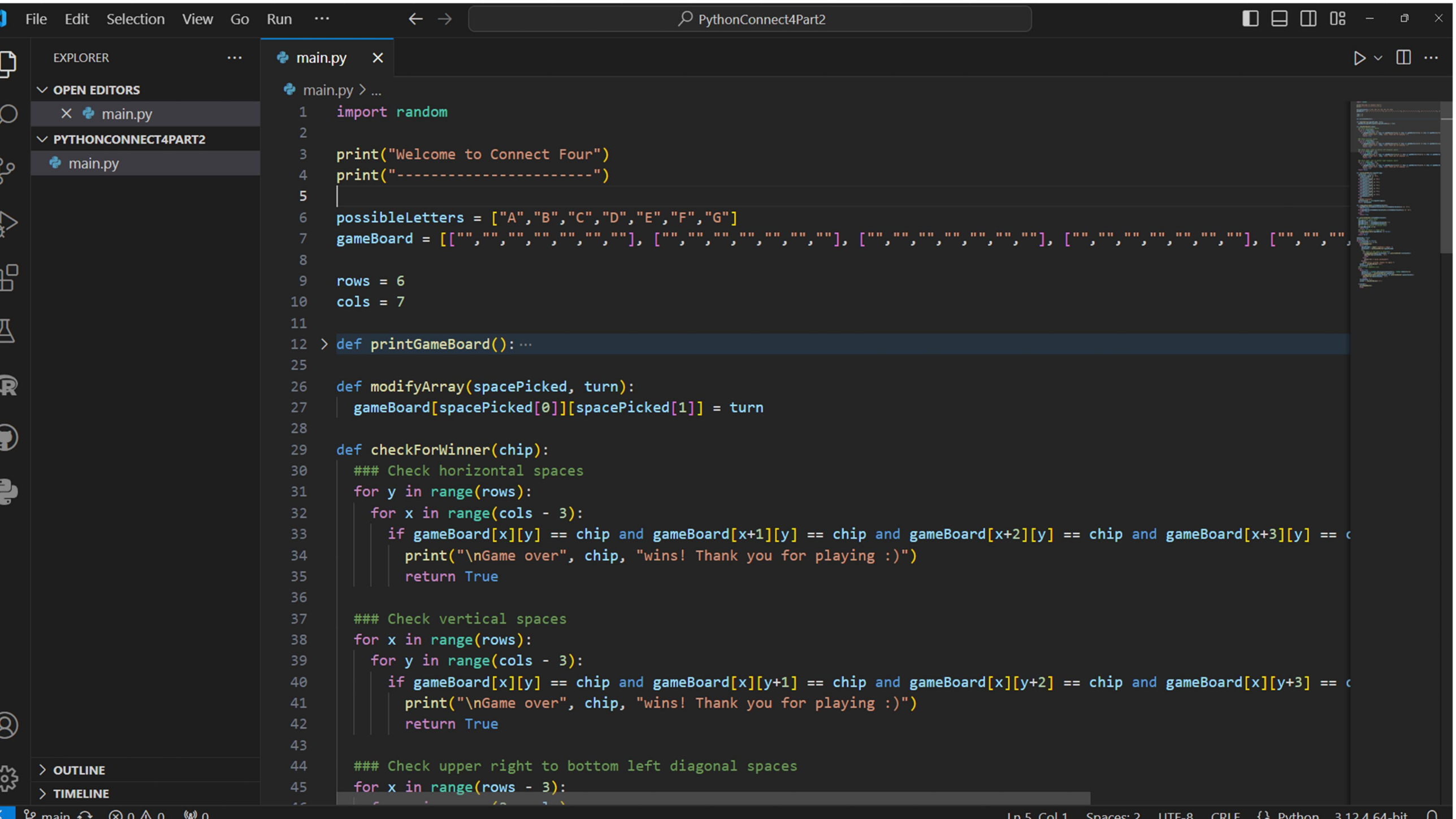
Minimax algorithm

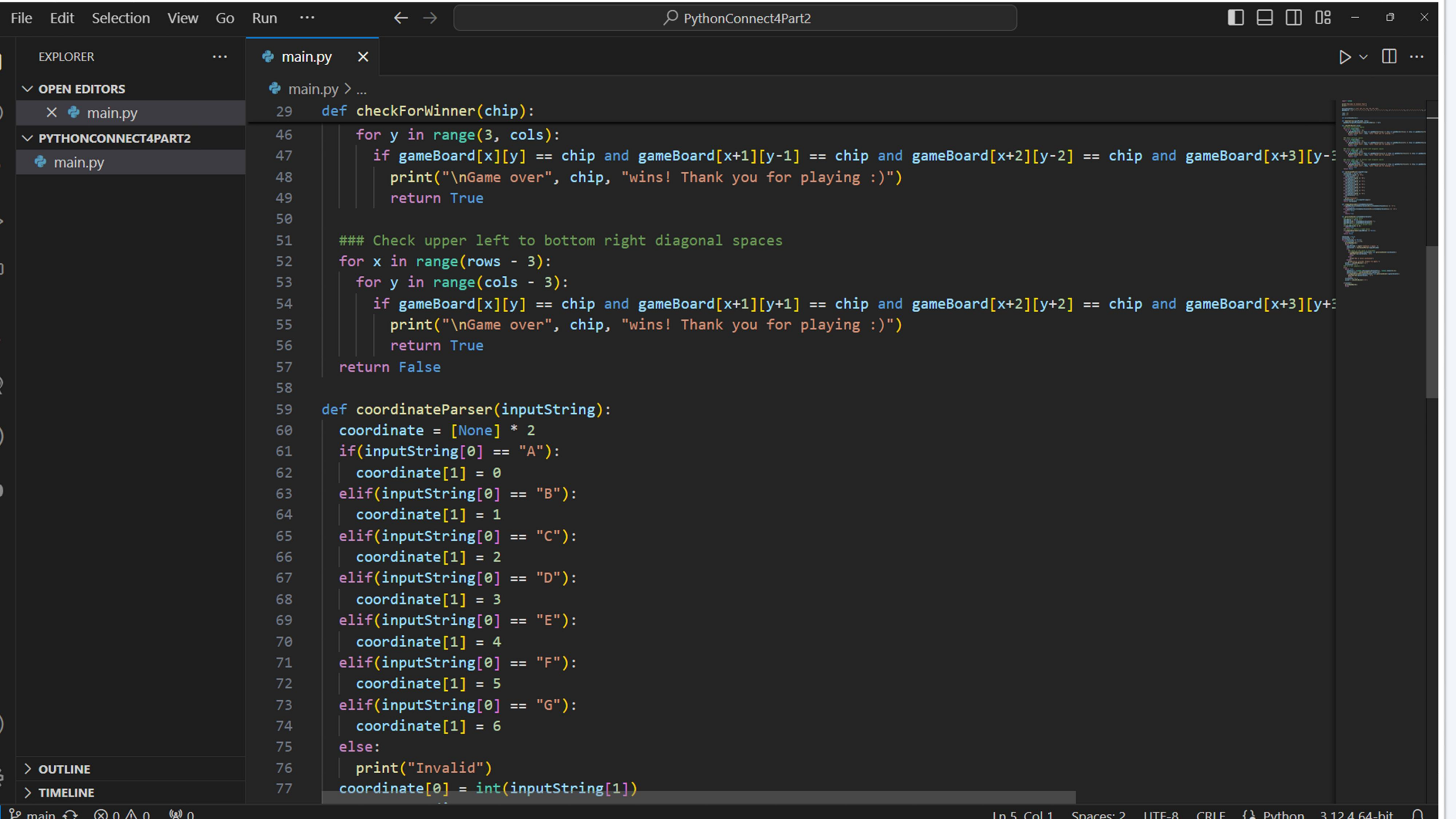
The minimax algorithm is a decision-making algorithm commonly used in game theory, specifically in two-player games with zero-sum outcomes. The algorithm is designed to determine the best move for a player, assuming that the opponent is also playing optimally.

The algorithm works by building a tree of possible game states and then evaluating each node of the tree by assigning it a value based on how favorable it is for the player. The values of the nodes are then propagated up the tree to the root node, where the player can choose the move that leads to the highest value.

Alpha-beta Pruning

Alpha-beta pruning is an optimization technique used in the minimax algorithm for game playing. It is used to reduce the number of nodes that are evaluated by the minimax algorithm by pruning the search tree in a way that does not affect the final result. The idea behind alpha-beta pruning is to minimize the number of nodes that need to be evaluated in the minimax tree by cutting off branches that will not affect the final decision.


File Edit Selection View Go Run ···
PythonConnect4Part2
EXPLORER
OPEN EDITORS
main.py
PYTHONCONNECT4PART2
main.py
main.py
main.py > ...
import random
print("Welcome to Connect Four")
print("----------------------")
possibleLetters = ["A","B","C","D","E","F","G"]
gameBoard = [["","","","","","",""], ["","","","","","",""], ["","","","","","",""], ["","","","","","",""], ["","","
rows = 6
cols = 7
def printGameBoard(): ···
def modifyArray(spacePicked, turn):
gameBoard[spacePicked[0]][spacePicked[1]] = turn
def checkForWinner(chip):
Check horizontal spaces
for y in range(rows):
for x in range(cols - 3):
if gameBoard[x][y] == chip and gameBoard[x+1][y] == chip and gameBoard[x+2][y] == chip and gameBoard[x+3][y] == c
print("\nGame over", chip, "wins! Thank you for playing :)")
return True
Check vertical spaces
for x in range(rows):
for y in range(cols - 3):
if gameBoard[x][y] == chip and gameBoard[x][y+1] == chip and gameBoard[x][y+2] == chip and gameBoard[x][y+3] == c
print("\nGame over", chip, "wins! Thank you for playing :)")
return True
Check upper right to bottom left diagonal spaces
for x in range(rows - 3):
OUTLINE
TIMELINE
main
Ln 5, Col 1
Spaces: 2
UTF-8
CRLF
Python
3.12.4 64-bit


File Edit Selection View Go Run ···
PythonConnect4Part2
EXPLORER
OPEN EDITORS
main.py
PYTHONCONNECT4PART2
main.py
OUTLINE
TIMELINE
main.py
main.py > ...
29 def checkForWinner(chip):
46 for y in range(3, cols):
47 if gameBoard[x][y] == chip and gameBoard[x+1][y-1] == chip and gameBoard[x+2][y-2] == chip and gameBoard[x+3][y-
48 print("\nGame over", chip, "wins! Thank you for playing :)")
49 return True
50
51 ### Check upper left to bottom right diagonal spaces
52 for x in range(rows - 3):
53 for y in range(cols - 3):
54 if gameBoard[x][y] == chip and gameBoard[x+1][y+1] == chip and gameBoard[x+2][y+2] == chip and gameBoard[x+3][y+
55 print("\nGame over", chip, "wins! Thank you for playing :)")
56 return True
57 return False
58
59 def coordinateParser(inputString):
60 coordinate = [None] * 2
61 if(inputString[0] == "A"):
62 coordinate[1] = 0
63 elif(inputString[0] == "B"):
64 coordinate[1] = 1
65 elif(inputString[0] == "C"):
66 coordinate[1] = 2
67 elif(inputString[0] == "D"):
68 coordinate[1] = 3
69 elif(inputString[0] == "E"):
70 coordinate[1] = 4
71 elif(inputString[0] == "F"):
72 coordinate[1] = 5
73 elif(inputString[0] == "G"):
74 coordinate[1] = 6
75 else:
76 print("Invalid")
77 coordinate[0] = int(inputString[1])
main
Ln 5, Col 1
Spaces: 2
UTF-8
CRLF
Python
3.12.4 64-bit

File Edit Selection View Go Run ··· PythonConnect4Part2

EXPLORER

OPEN EDITORS
main.py

PYTHONCONNECT4PART2
main.py

OUTLINE

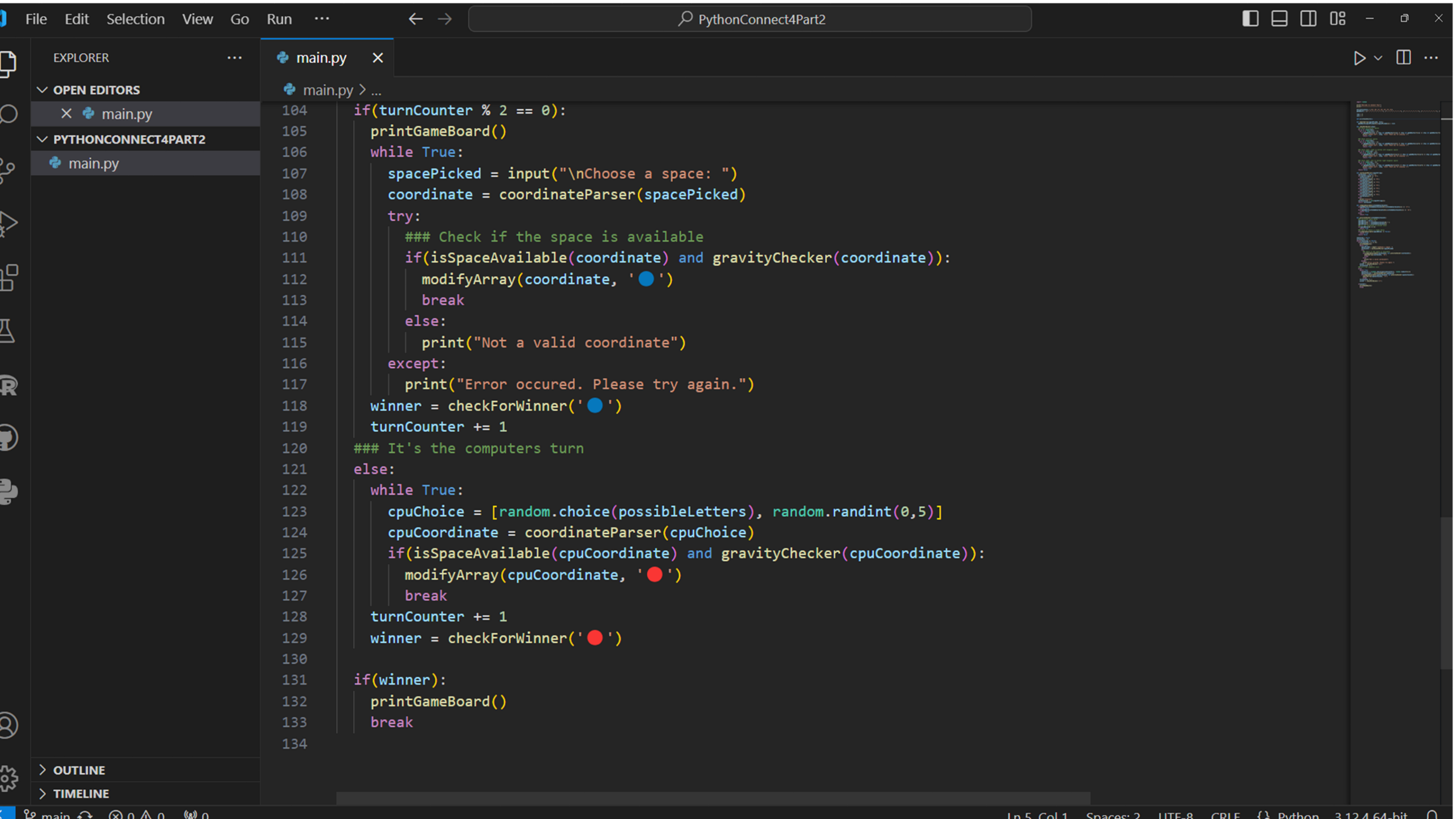
TIMELINE

main.py

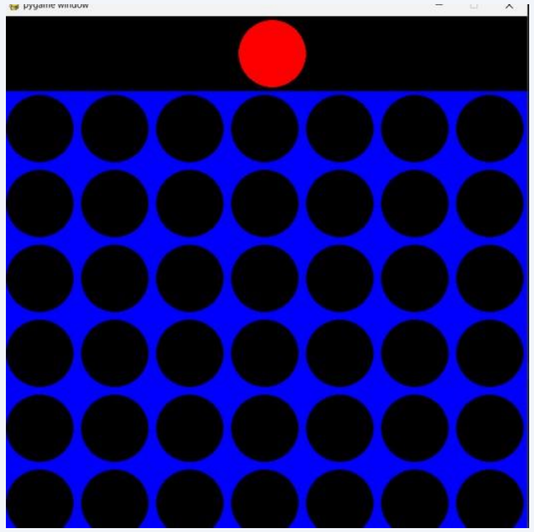
main.py > ...

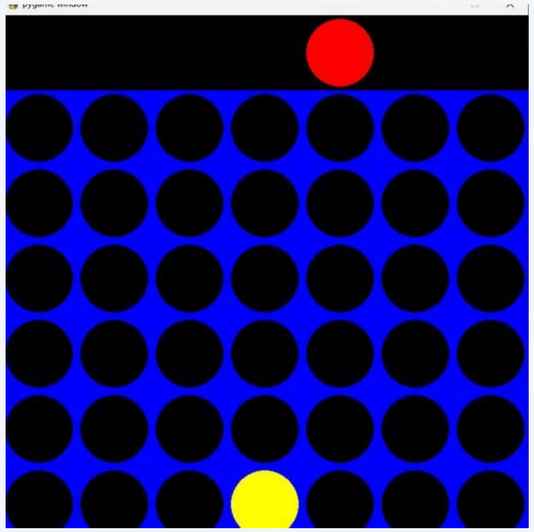
```
59  def coordinateParser(inputString):
78    return coordinate
79
80  def isSpaceAvailable(intendedCoordinate):
81    if(gameBoard[intendedCoordinate[0]][intendedCoordinate[1]] == '🔴'):
82      return False
83    elif(gameBoard[intendedCoordinate[0]][intendedCoordinate[1]] == '🔵'):
84      return False
85    else:
86      return True
87
88  def gravityChecker(intendedCoordinate):
89    ### Calculate space below
90    spaceBelow = [None] * 2
91    spaceBelow[0] = intendedCoordinate[0] + 1
92    spaceBelow[1] = intendedCoordinate[1]
93    ### Is the coordinate at ground level
94    if(spaceBelow[0] == 6):
95      return True
96    ### Check if there's a token below
97    if(isSpaceAvailable(spaceBelow) == False):
98      return True
99    return False
100
101 leaveLoop = False
102 turnCounter = 0
103 while(leaveLoop == False):
104   if(turnCounter % 2 == 0):
105     printGameBoard()
106     while True:
107       spacePicked = input("\nChoose a space: ")
108       coordinate = coordinateParser(spacePicked)
109       try:
110         ### Check if the space is available
```

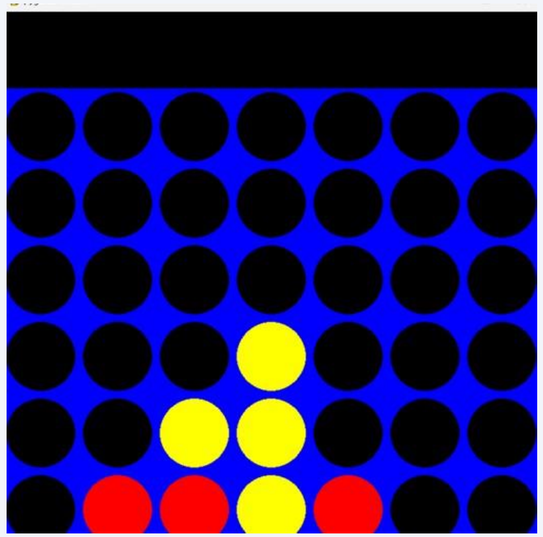
main ⊗ 0 ⚠ 0 0 Ln 5, Col 1 Spaces: 2 UTF-8 CRLF Python 3.12.4 64-bit

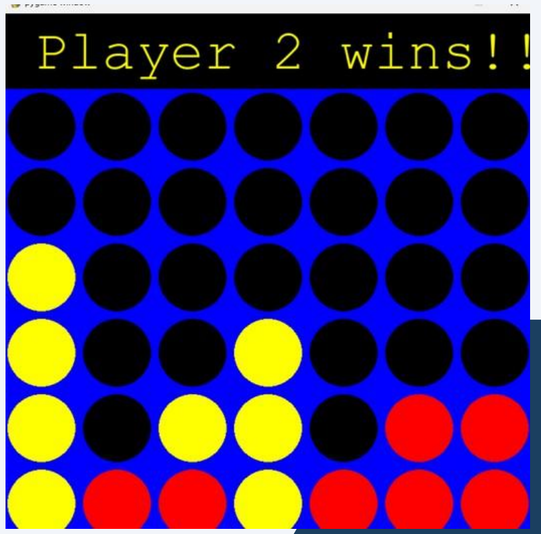

File Edit Selection View Go Run ···
PythonConnect4Part2
EXPLORER
OPEN EDITORS
main.py
PYTHONCONNECT4PART2
main.py
OUTLINE
TIMELINE
main.py
main.py > ...
104 if(turnCounter % 2 == 0):
105 printGameBoard()
106 while True:
107 spacePicked = input("\nChoose a space: ")
108 coordinate = coordinateParser(spacePicked)
109 try:
110 ### Check if the space is available
111 if(isSpaceAvailable(coordinate) and gravityChecker(coordinate)):
112 modifyArray(coordinate, '🔵')
113 break
114 else:
115 print("Not a valid coordinate")
116 except:
117 print("Error occured. Please try again.")
118 winner = checkForWinner('🔵')
119 turnCounter += 1
120 ### It's the computers turn
121 else:
122 while True:
123 cpuChoice = [random.choice(possibleLetters), random.randint(0,5)]
124 cpuCoordinate = coordinateParser(cpuChoice)
125 if(isSpaceAvailable(cpuCoordinate) and gravityChecker(cpuCoordinate)):
126 modifyArray(cpuCoordinate, '🔴')
127 break
128 turnCounter += 1
129 winner = checkForWinner('🔴')
130
131 if(winner):
132 printGameBoard()
133 break
134
main
Ln 5, Col 1
Spaces: 2
UTF-8
CRLF
Python
3.12.4 64-bit

Pictures of the game working









Conclusion

The minimax algorithm is a powerful decision-making algorithm used in game theory and decision theory. It evaluates all possible moves from the current state of the game, assuming that both players play optimally. The alpha-beta pruning technique can be used to reduce the number of nodes that need to be evaluated in the minimax algorithm, resulting in more efficient computation. In the context of a game like Connect Four, the minimax algorithm can be used to determine the best move for the AI player by evaluating all possible moves from the current state of the game and using a heuristic evaluation function to estimate the value of the board state. The depth of the search determines how many moves ahead the algorithm looks in the game tree, and increasing the depth can improve the accuracy of the evaluation but also increase computational costs.

THANK YOU