```
 1 import numpy as np
 2 import random
 3 import math
 4
 5 TOTAL_ROWS = 6
 6 TOTAL_COLUMNS = 7
 7 PLAYER = 0
 8 PLAYER2 = 1
 9 AI = 1
10
11 def initializeBoard(): # numpy configs are in the
   assignment package
12     board = np.zeros((TOTAL_ROWS, TOTAL_COLUMNS), int) #
   numpy is used only to initalize board
13     return board
14
15
16 # to check if last row of given column is not full
17 def validPosition(board, col):
18     return board[TOTAL_ROWS - 1][col] == 0
19
20
21 # give the next valid position for the turn in a column
22 def nextValidPosition(board, col):
23     for r in range(TOTAL_ROWS):
24         if board[r][col] == 0:
25             return r
26
27
28 # function to initalize the turn
29 def nextTurn(board, row, col, value):
30     board[row][col] = value
31
32
33 # print the current state of the board
34 def printState(board):
35     a = []
36     for r in reversed(range(TOTAL_ROWS)):
37         for c in reversed(range(TOTAL_COLUMNS)):
38             a.append(board[r][c])
39         a.reverse()
40         print(a)
41         a.clear()
42
43
44 # to check the winning move acc to rules, it takes the
```

```
44 value of player's move
45 def isWinningMove(board, value):
46     # to check if horizontal connect 4 is complete
47     for c in range(TOTAL_COLUMNS - 3):
48         for r in range(TOTAL_ROWS):
49             if board[r][c] == value and board[r][c + 1] ==
   value and board[r][c + 2] == value and board[r][
50                 c + 3] == value:
51                 return True
52
53     # to check if vertical connect 4 is complete
54     for c in range(TOTAL_COLUMNS - 3):
55         for r in range(TOTAL_ROWS - 3):
56             if board[r][c] == value and board[r + 1][c] ==
   value and board[r + 2][c] == value and board[r + 3][
57                 c] == value:
58                 return True
59
60         # to check if downward diagonal connect 4 is
   complete
61     for c in range(TOTAL_COLUMNS - 3):
62         for r in range(3, TOTAL_ROWS):
63             if board[r][c] == value and board[r - 1][c + 1
   ] == value and board[r - 2][c + 2] == value and board[r - 3
   ][
64                 c + 3] == value:
65                 return True
66
67     # to check if upward diagonal connect 4 is complete
68     for c in range(TOTAL_COLUMNS - 3):
69         for r in range(TOTAL_ROWS - 3):
70             if board[r][c] == value and board[r + 1][c + 1
   ] == value and board[r + 2][c + 2] == value and board[r + 3
   ][
71                 c + 3] == value:
72                 return True
73
74
75 # AI's turn or action is passed into utility function
76 def utilityFunction(board, action):
77     hScore = 0
78     # heuristic score for action at center column
79     centerColumn = [int(i) for i in list(board[:,
   TOTAL_COLUMNS//2])]
80     centerCount = centerColumn.count(action)
81     hScore += centerCount * 3
```

```
 82
 83      # heurisitc score for next horizontal move
 84      for r in range(TOTAL_ROWS):
 85          rowArray = [int(i) for i in list(board[r,:])]  #
    every column for specific row
 86          for c in range(TOTAL_COLUMNS - 3):
 87              toCheck = rowArray[c:c + 4] # checking each
    row starting from c to next 4
 88              hScore += staticEvaluation(toCheck, action)
 89
 90      # heurisitc score for next vertical move
 91      for c in range(TOTAL_COLUMNS):
 92          colArray = [int(i) for i in list(board[:,c])]  #
    every row for specific column
 93          for r in range(TOTAL_ROWS - 3):
 94              toCheck = colArray[r:r + 4] # checking each
    column starting from r to next 4
 95              hScore += staticEvaluation(toCheck, action)
 96
 97      # heurisitc score for next downward diagaonal move
 98      for r in range(TOTAL_ROWS - 3):
 99          for c in range(TOTAL_COLUMNS - 3):
100              toCheck = [board[r + 3 - i][c + i] for i in
    range(4)] # checking each diagonal starting from r to next
     4
101              hScore += staticEvaluation(toCheck, action)
102
103      # heurisitc score for next upward diagonal move
104      for r in range(TOTAL_ROWS - 3):
105          for c in range(TOTAL_COLUMNS - 3):
106              toCheck = [board[r + i][c + i] for i in range(
    4)]
107              hScore += staticEvaluation(toCheck, action)
108
109      return hScore
110
111
112 # helps to calculate heurisitic score by taking how many
    empty or filled(toCheck) action
113 def staticEvaluation(toCheck, action):
114      score = 0
115      opponent = 1
116      if action == 1:
117          opponent = 2
118
119      if toCheck.count(opponent) == 3 and toCheck.count(0
```

```
119 ) == 1:
120         score -= 4 # if there is an opportunity for opp to
     make 3 then assign -4
121 
122     elif toCheck.count(action) == 3 and toCheck.count(0
    ) == 1:
123         score += 5 # if there is an opportunity to make 3
    then assign 5
124 
125     elif toCheck.count(action) == 2 and toCheck.count(0
    ) == 2:
126         score += 2  # if there is an opportunity to make 2
     then assign 2
127 
128     elif toCheck.count(action) == 4:
129         score += 100 # if there is an opportunity to make
    4 then assign 100
130 
131     if toCheck.count(opponent) == 2 and toCheck.count(0
    ) == 1:
132         score -= 2 # if there is an opportunity for opp to
     make 3 then assign -2
133 
134     return score
135 
136 
137 def validMoves(board):
138     moves = []
139     for col in range(TOTAL_COLUMNS):
140         if validPosition(board, col):
141             moves.append(col)
142     return moves
143 
144 
145 def terminalTest(board):
146     return len(validMoves(board)) == 0 or isWinningMove(
    board, 1) or isWinningMove(board, 2)
147 
148 
149 def minimax(board, depth, maximizer):
150     successor = validMoves(board)
151     lastNode = terminalTest(board)
152     if lastNode:
153         if isWinningMove(board, 2):
154             return (None, math.inf)
155         elif isWinningMove(board, 1):
```

```
156             return (None, -math.inf)
157         else:
158             return (None, 0)
159     elif depth == 0:
160         return (None, utilityFunction(board, 2))
161 
162     if maximizer:
163         v = -math.inf
164         column = random.choice(successor)
165         for col in successor:
166             row = nextValidPosition(board, col)
167             # a copy is made so the maximizer can test the
    outcome while doing recursively
168             b = board.copy()
169             nextTurn(b, row, col, 2)
170             updateStatEval = minimax(b, depth - 1, False)[
    1]
171             if updateStatEval > v:
172                 v = updateStatEval
173                 column = col
174         return column, v
175     # minimizer
176     else:
177         v = math.inf
178         column = random.choice(successor)
179         for col in successor:
180             row = nextValidPosition(board, col)
181             # a copy is made so the minimizer can test the
    outcome
182             b = board.copy()
183             nextTurn(b, row, col, 1)
184             updateStatEval = minimax(b, depth - 1, True)[1
    ]
185             if updateStatEval < v:
186                 v = updateStatEval
187                 column = col
188         return column, v
189 
190 
191 a = int(input("Press 1 for HUMAN or Press 2 for AI: "))
192 print("")
193 board = initializeBoard()
194 printState(board)
195 gameOver = False
196 if a == 1:
197     turn = random.randint(PLAYER, PLAYER2)  # to start
```

```
197 with random turn
198     print("Begin Game!!!!!!")
199 
200     while (gameOver == False):
201 
202         if turn == PLAYER:
203             col = int(input("First Player's turn (enter
    between 0-6): "))
204             if validPosition(board, col):
205                 row = nextValidPosition(board, col)
206                 nextTurn(board, row, col, 1)
207 
208                 if isWinningMove(board, 1):
209                     print("Player 1 Wins!")
210                     gameOver = True
211 
212                 printState(board)
213                 turn += 1
214                 turn = turn % 2  # to keep the value
    between 0 & 1
215 
216         if turn == PLAYER2 and gameOver == False:
217             col = int(input("Second Player's turn (enter
    between 0-6): "))
218             if validPosition(board, col):
219                 row = nextValidPosition(board, col)
220                 nextTurn(board, row, col, 2)
221 
222                 if isWinningMove(board, 2):
223                     print("Player 2 Wins!")
224                     gameOver = True
225 
226                 printState(board)
227                 turn += 1
228                 turn = turn % 2  # to keep the value
    between 0 & 1
229     print("Game Over!!!!!")
230 
231 if a == 2:
232     turn = random.randint(PLAYER, AI)  # to start with
    random turn
233     print("Begin Game with AI")
234     d = int(input("Enter Depth for AI (4 = less than a sec
    , 5 = 5 sec, 6 = 40 sec, 7 = 3 min+):  "))
235 
236     while (gameOver == False):
```

```
237
238         if turn == PLAYER:
239             col = int(input("First Player's turn (enter
    between 0-6): "))
240             if validPosition(board, col):
241                 row = nextValidPosition(board, col)
242                 nextTurn(board, row, col, 1)
243
244                 if isWinningMove(board, 1):
245                     print("Player 1 Wins!")
246                     gameOver = True
247
248                 printState(board)
249                 turn += 1
250                 turn = turn % 2  # to keep the value
    between 0 & 1
251
252         elif turn == AI and gameOver == False:
253
254             col, minimaxScore = minimax(board, d, True)
255             print("AI's Turn")
256             if validPosition(board, col):
257                 row = nextValidPosition(board, col)
258                 nextTurn(board, row, col, 2)
259
260                 if isWinningMove(board, 2):
261                     print("AI Wins!")
262                     gameOver = True
263
264                 printState(board)
265                 turn += 1
266                 turn = turn % 2  # to keep the value
    between 0 & 1
267     print("Game Over!!!!!")
268
269 else:
270     print("Wrong Choice! Run again.")
271
```