**COP 4630**

**DR MARQUES**

**Summer 2022**

***Armando Arias Architect***

***Joshua Garden Developer***

***Flor Aguirre Reporter***

|  |  |
| --- | --- |
|  | REPORT PROJECT I  Game Implementation using the easyAi Framework |
|  |  |
|  |  |

**BUILDING GAMES WITH ARTIFICIAL INTELLIGENCE**

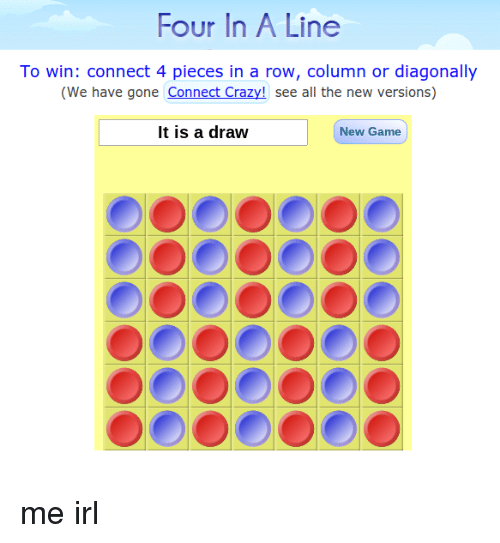
**Purpose:**

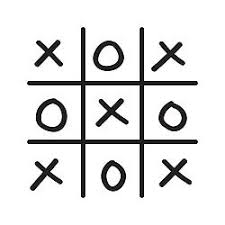
The purpose of this Assignment is to build games using search algorithms to effectively come up with strategies to win games, including Tic Tac Toe, Connector four and checkers and checkers with artificial intelligence.

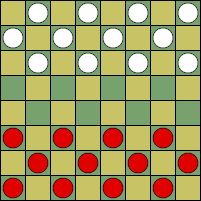
**Theory**:

In Artificial Intelligence, games are deterministic, using turns as two players or zero-sum, which describe if a participant will win or loss. A game with two players is a type of adversarial search such as min -max useful when something else affects the environment, where MAX moves first, and MIN is the response to that move.

Adversarial Search is characterized by opposition, it requires the anticipation and understanding of the actions of the opponent in pursuit of a goal, this is the type of search utilized in games like tic-tac-toe, connect Four and checkers.

**Connect Four** is a two-player game, **is an example of an adversarial, finite zero-sum game of perfect information**. This means that there are two players, against each other in a game where only one player can win. In this game the players take turns dropping the tokens into a vertical grid, the goal is to get for tokens in a line.

**Tic-Tac-Toe**: The idea is that by anticipating the moves of the opponent in response to moves by the algorithm, the algorithm can come up with the best possible next move.

**Checkers**: The objective of the game is to get as many pieces as possible from the opponent, the game can be won when the opponent is unable to make a move, because all the pieces were captured, or the opponent get the king.

**How to Build the Tree**:

* Start: From the root of the tree.
  + Node has several children (represented by the moves)
    - More children that represent the states after the opponent moves.
  + Result of the game (after executing various moves)
* End: When one of the players win.

**The sequence of tree processing using min-max search**

Diagram

Description automatically generated

**IMPLEMENTATION REPORT**

**Part I:**

1. Conda Installation and Configuration

Text

Description automatically generated

A screenshot of a computer

Description automatically generated

1. EasyAI Installation

Text

Description automatically generated

**Parte 2: Run Tic-Tac-Toe presented in easyAI documentation**

**Code:**

from easyAI import TwoPlayerGame

from easyAI.Player import Human\_Player

class TicTacToe( TwoPlayerGame ):

""" The board positions are numbered as follows:

7 8 9

4 5 6

1 2 3

"""

def \_\_init\_\_(self, players):

self.players = players

self.board = [0]\*9

self.current\_player = 1 # player 1 starts.

def possible\_moves(self):

return [i+1 for i,e in enumerate(self.board) if e==0]

def make\_move(self, move):

self.board[int(move)-1] = self.current\_player

def unmake\_move(self, move): # optional method (speeds up the AI)

self.board[int(move)-1] = 0

def lose(self):

""" Has the opponent "three in line ?" """

return any( [all([(self.board[c-1]== self.current\_player)

for c in line])

for line in [[1,2,3],[4,5,6],[7,8,9], # horiz.

[1,4,7],[2,5,8],[3,6,9], # vertical

[1,5,9],[3,5,7]]]) # diagonal

def is\_over(self):

return (self.possible\_moves() == []) or self.lose()

def show(self):

print ('\n'+'\n'.join([

' '.join([['.','O','X'][self.board[3\*j+i]]

for i in range(3)])

for j in range(3)]) )

def scoring(self):

return -100 if self.lose() else 0

if \_\_name\_\_ == "\_\_main\_\_":

from easyAI import AI\_Player, Negamax

ai\_algo = Negamax(6)

TicTacToe( [Human\_Player(),AI\_Player(ai\_algo)]).play()

Graphical user interface, application, Word

Description automatically generatedGraphical user interface, application, Word

Description automatically generated

**Four Connect Game:**

**Code**

#!/usr/bin/env python3

try:

import numpy as np

except ImportError:

print("Sorry, this example requires Numpy installed !")

raise

from easyAI import TwoPlayerGame

class ConnectFour(TwoPlayerGame):

"""

The game of Connect Four, as described here:

http://en.wikipedia.org/wiki/Connect\_Four

"""

def \_\_init\_\_(self, players, board = None):

self.players = players

self.board = board if (board != None) else (

np.array([[0 for i in range(7)] for j in range(6)]))

# self.nplayer = 1 # player 1 starts.

self.current\_player = 1

def possible\_moves(self):

return [i for i in range(7) if (self.board[:, i].min() == 0)]

def make\_move(self, column):

line = np.argmin(self.board[:, column] != 0)

self.board[line, column] = self.current\_player

def show(self):

print('\n' + '\n'.join(

['0 1 2 3 4 5 6', 13 \* '-'] +

[' '.join([['.', 'O', 'X'][self.board[5 - j][i]]

for i in range(7)]) for j in range(6)]))

def lose(self):

return find\_four(self.board, self.current\_player)

def is\_over(self):

return (self.board.min() > 0) or self.lose()

def scoring(self):

return -100 if self.lose() else 0

def find\_four(board, nplayer):

"""

Returns True iff the player has connected 4 (or more)

This is much faster if written in C or Cython

"""

for pos, direction in POS\_DIR:

streak = 0

while (0 <= pos[0] <= 5) and (0 <= pos[1] <= 6):

if board[pos[0], pos[1]] == nplayer:

streak += 1

if streak == 4:

return True

else:

streak = 0

pos = pos + direction

return False

POS\_DIR = np.array([[[i, 0], [0, 1]] for i in range(6)] +

[[[0, i], [1, 0]] for i in range(7)] +

[[[i, 0], [1, 1]] for i in range(1, 3)] +

[[[0, i], [1, 1]] for i in range(4)] +

[[[i, 6], [1, -1]] for i in range(1, 3)] +

[[[0, i], [1, -1]] for i in range(3, 7)])

if \_\_name\_\_ == '\_\_main\_\_':

# LET'S PLAY !

from easyAI import Human\_Player, AI\_Player, Negamax, SSS, DUAL

ai\_algo\_neg = Negamax(5)

ai\_algo\_sss = SSS(5)

game = ConnectFour([AI\_Player(ai\_algo\_neg), AI\_Player(ai\_algo\_sss)])

game.play()

if game.lose():

print("Player %d wins." % (game.current\_player))

else:

print("Looks like we have a draw.")

A screenshot of a computer

Description automatically generated with medium confidence

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |