Project 1

Keanu Francis, Name 1, Name 2

Florida Atlantic University

Abstract

In this this project we applied the first primate for of which comprises of an intelligent search algorithm. This intelligent search will play three games, the first two being Tic-Tac-Toe and Connect-4. These two games are already completed and is used to demonstrate that Easy-Ai

The final program will checker is incomplete and we are responsible for deploying the functions make\_move(), lose(), is\_over() and scoring().

Project 1

To run these programs, we will require a valid version of python installed on the system. Since I have already used python in the past, I have already had python and I operate on python project using the Visual studio code.

# Setup

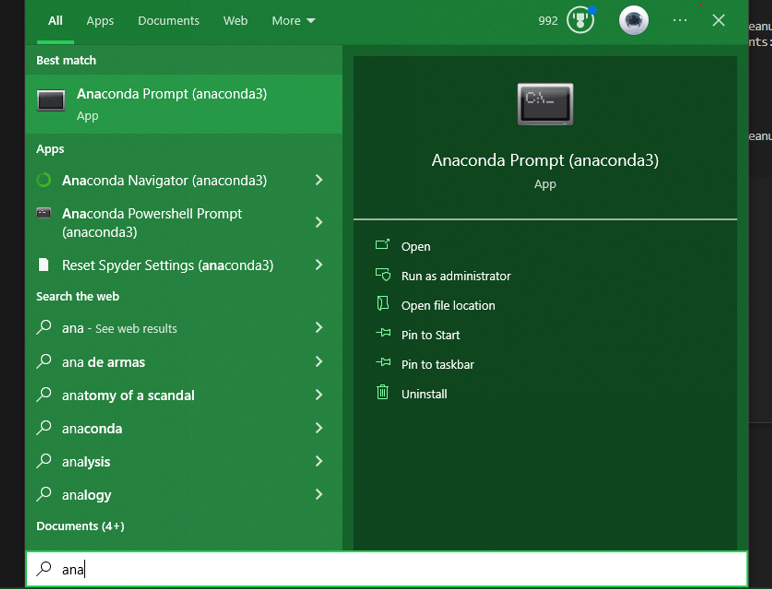
## Installing Python

Using the terminal, I can determine the current install of python which is my previous install of Python 3.9.

Text

Description automatically generated

## Using Conda



Once it is installed from the search bar, I can find the anaconda terminal and install the create and activate the python 3.8 virtual environment.

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

We can see that the Python 3.8 environment has been installed and successfully activated. However from now on I will be using the terminal inside of visual studio code.

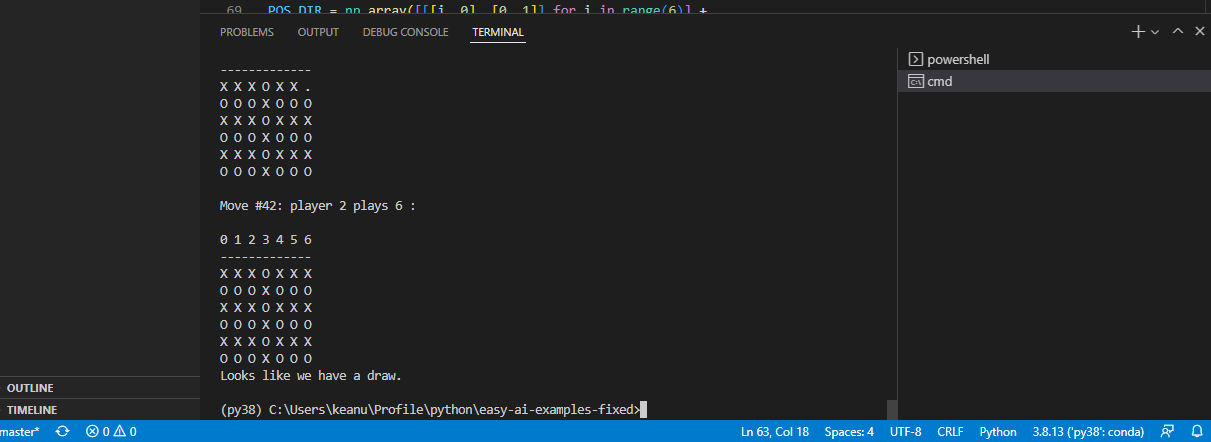
## EasyAI Installation

From inside of the visual studio code terminal, we will activate the py38 environment and install the easy ai package using the command python -m pip install EasyAIText

Description automatically generated

The next step is to run the Tic-Tac-Toe and Connect-4 games.

## Running Tic-Tac-Toe and Connect-4

From inside of the visual studio code terminal, we will activate the py38 environment and install the easy ai package using the command **python -m pip install EasyAI.** After copying the GitHub repo of the examples, we can run the programs from inside of the command prompt we will first start the connect-4.

We can see that this game runs until there is a draw.

A screenshot of a computer

Description automatically generated Next, we run the Tic-Tac-Toe in the same way but in this case, it requires user input this uses the numbers 1 – 9 to determine which space on the board the user will use. We can see that player 1 is the winner in this case.

## Implementing the game of checkers.

In order to run the game of checkers we need to implement 4 functions these are make\_move() , lose(), is\_over() and scoring().

## Implementing make\_move().

The first we will implement is the make move. From the instruction inside of the sample code we can see that the input will be a NumPy array containing the move that is required.

Text

Description automatically generated.

To make this move the board and the current locations of each piece will have to be updated after some deliberations we can see the algorithm from the pseudocode.

Function make\_move(self, pos):

Input: self-reference to the object of a class and pos[]s an 8x8 NumPy array

Output: None

TempW <- [Empty]

TempB <- [Empty]

For j = 0 to 7

For i = 0 to 7

If pos[j,i] = ‘B’

Append tempB with (j,i)

Else If pos[j,i] = ‘W’

Append tempW with (j,i)

self.board <- pos

player 1 pos <= tempW

player 2 pos <= tempB

However, there was a function in the program called that did the same thing as my pseudocode.

A screenshot of a computer

Description automatically generated with medium confidence get\_pice\_pos\_from\_table(self, table\_pos) does essentially the same thing as my pseudocode . It can be implemented by simply assigning the calling the object self.players[self.current\_players-1] and assigning it the return value of the function after passing it the pos variable.

Text

Description automatically generated

## Implementing lose()

For lose() to work the function needs to determine if the current player has lost. This means that the opposing player would have to have pieces in the current players territory as represented by self.white\_territory or the self.black\_territory. There are two tuples in which we need to determine if there is any intersection, the opposing player position, and the current players territory. We can do this by performing the list(set(A)&set(B)) this function returns a new list containing values in which both lists share if this list is not empty then we can assume the player lost. For example, if A = ((1,2),(6.5),(4.9),) and B=((2,2),(6.5)) the command would return (6,5).

Text

Description automatically generated

We can see that depending on which player is current it checks the opposing players position and sees if there are any intersections.

## Implementing is\_over(self)

The game is currently over if one of the players have encountered a loss. This loss is determined from the is over function. So, if we return the value from the lose() function we can determine if the game is over or if there are no moves.

A screenshot of a computer

Description automatically generated with medium confidence

## Implementing is\_over(self)

Finally, there is scoring this works by returning -100 if there is a lose and a 0 if there is a win this is done simple done by checking the value of self.lose() if it is true then the player lost return -100 else return 0.

Shape

Description automatically generated with medium confidence

## Running checker\_questions.py

Finally, after activating the py38 from the terminal, we can run the program

Text

Description automatically generated

We can see that at the end of the execution white enters black territory, achieves victory, and ends the game.

The source code is available at. <https://github.com/Kfrancis2018/Project-1-Checker-AI.git>

Finally, we can run the test checker to see the validation of out code.