Project 1 – Part 1

Richard L. McCormick

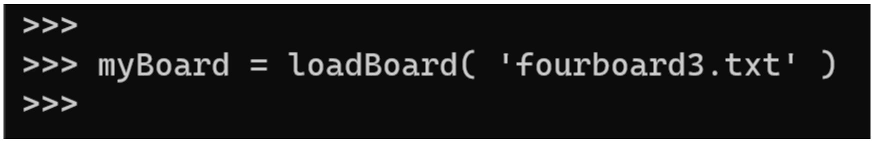
Northern Arizona University

CS470: Artificial Intelligence

Prof. Dr. Lan Zhang

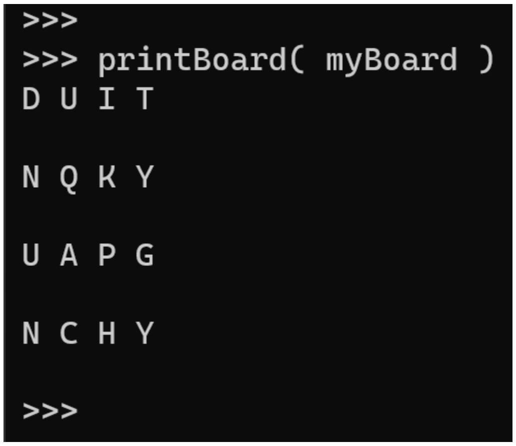
February 2nd, 2024

1. Save the file attached here, and feed it to your loadBoard function to get this board loaded into your solver.



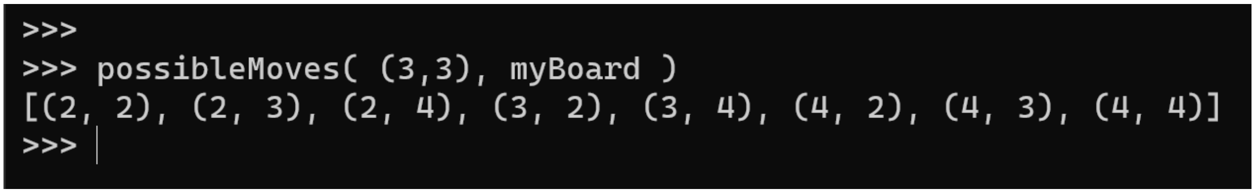
```
>>>
>>> myBoard = loadBoard( 'fourboard3.txt' )
>>>
```

2. Run your printBoard function to show us the loaded board.



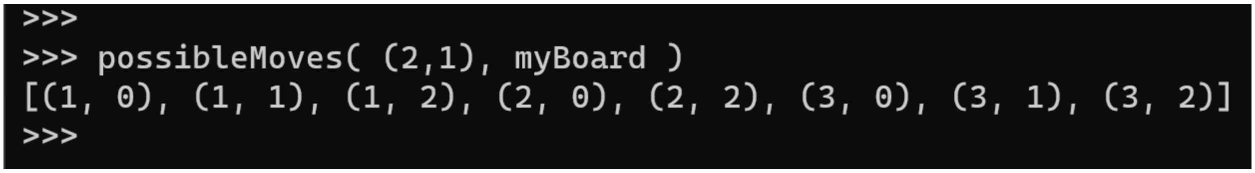
```
>>>
>>> printBoard( myBoard )
D U I T

N Q K Y

U A P G

N C H Y

>>>
```

3. Run your possibleMoves function on the board for position (3,3) #remember, all indexing starts at (0,0) in the top left corner!



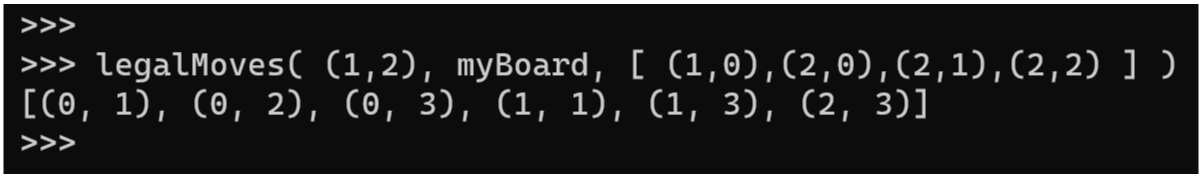
```
>>>
>>> possibleMoves( (3,3), myBoard )
[(2, 2), (2, 3), (2, 4), (3, 2), (3, 4), (4, 2), (4, 3), (4, 4)]
>>>
```

4. Run your possibleMoves function on the board for position (2,1)



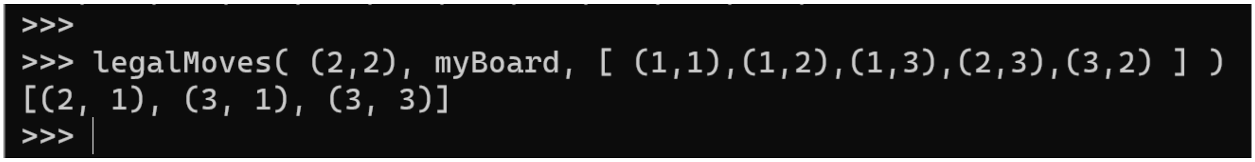
```
>>>
>>> possibleMoves( (2,1), myBoard )
[(1, 0), (1, 1), (1, 2), (2, 0), (2, 2), (3, 0), (3, 1), (3, 2)]
>>>
```

5. Run your legalMoves function on the board for position (1,2), assuming you have a past path of ((1,0),(2,0),(2,1),(2,2))



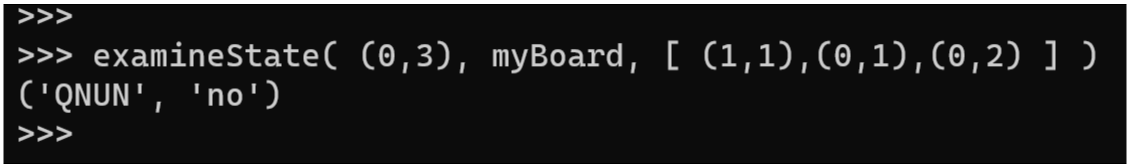
```
>>>
>>> legalMoves( (1,2), myBoard, [ (1,0),(2,0),(2,1),(2,2) ] )
[(0, 1), (0, 2), (0, 3), (1, 1), (1, 3), (2, 3)]
>>>
```

6. Run your legalMoves function on the board for position (2,2), assuming a path of ((1,1),(1,2),(1,3),(2,3),(3,2))



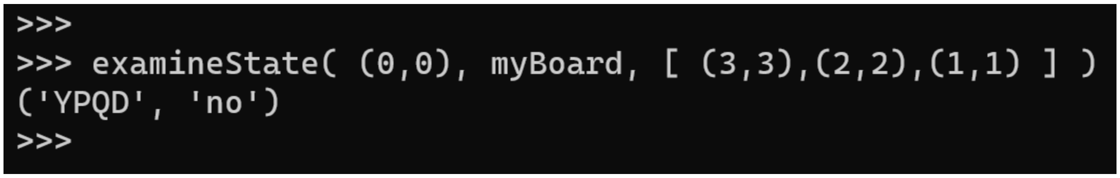
```
>>>
>>> legalMoves( (2,2), myBoard, [ (1,1),(1,2),(1,3),(2,3),(3,2) ] )
[(2, 1), (3, 1), (3, 3)]
>>>
```

7. Run examineState on the board at (0,3), with a past path of ((1,1), (0,1),(0,2))



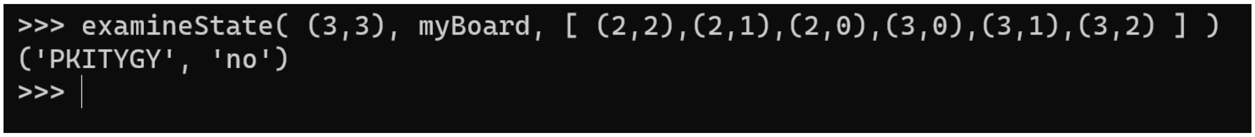
```
>>>
>>> examineState( (0,3), myBoard, [ (1,1),(0,1),(0,2) ] )
('QNUN', 'no')
>>>
```

8. Run examineState on the board at (0,0), with a past path of ((3,3), (2,2), (1,1))



```
>>>
>>> examineState( (0,0), myBoard, [ (3,3),(2,2),(1,1) ] )
('YPQD', 'no')
>>>
```

9. Run examineState on the board at (3,3), with a past path of ((2,2),(2,1),(2,0),(3,0),(3,1),(3,2))



```
>>> examineState( (3,3), myBoard, [ (2,2),(2,1),(2,0),(3,0),(3,1),(3,2) ] )
('PKITYGY', 'no')
>>>
```

```
__author__ = "RLM443"

import math
"""
CS 470 - Artificial Intelligence
Project One - Part One
Richard McCormick
Northern Arizona University
"""

def loadBoard( filename ):
    """
    Function:       loadBoard
    Description:    Loads a Boggle board from a text file.
    Params:         filename    Name of file to open
    Return:         2D Game Board Array
    """
    file = open( filename, 'r' )    # Open the file
    text = file.read().split()      # Read the file, split into letters
    file.close()                    # Close the file

    # Get the size of the board by taking the root of the number of letters
    size = int( math.sqrt( len( text ) ) )

    # Create a 2D matrix of the board
    finalBoard = [[0 for i in range(size)] for j in range(size)]

    i=0                                 # Establish a counter
    for y in range(size):               # Iterate Y-Axis
        for x in range(size):           # Iterate X-Axis
            finalBoard[x][y] = text[i]  # Fill the board slot with letter
            i = i + 1                   # Increment counter

    return( finalBoard )                # Return completed board

def printBoard( board ):
    """
    Function:       printBoard
    Description:    Prints a given Boggle board to screen.
    Params:         board       Board to print out.
    Return:         None
    """
    N = len( board )                        # Get board size (N)
    for y in range(N):                      # Traverse Y axis
        for x in range(N):                  # Traverse X axis
            print( board[x][y], end=" " )   # Print current letter
        print("\n")                         # At end of row, print new line

def possibleMoves( cords, board ):
    """
    Function:       possibleMoves
    Description:    Returns a list of possible moves given a set of
                    coordinates and a game board.
    Params:         cords           Co-ordinates to check.
                    board           Board to check.
    Return:         Array of coordinates
    """
    x = cords[0]        # Get X coordinate
    y = cords[1]        # Get Y coordinate
    possibleMoves = []  # Initialize empty list of possible moves
    N = len( board )    # Get size of board

    for i in range( x-1, x+2 ):
    # Search the X axis first, within 1 space of the current X Coord
        if( i >= 0 and i <= N ):
        # If coord is negative or outside board, it is not valid.
            x_temp = i
            # Everything else is valid, so add to list.
            for j in range( y-1, y+2 ):
            # Repeat loop for the Y axis (within 1 space either way).
                if( j >= 0 and j <= N ):
                # Coord must be non-negative and inside board.
                    y_temp = j
                    # Take the valid Y coord.
                    possibleMoves.append( ( x_temp, y_temp ) )
                    # Add both coords to list.

    possibleMoves.remove( cords )   # Self is not a valid move, remove.
    return( possibleMoves )         # Return list.

def legalMoves( cords, board, history ):
    """
    Function:       legalMoves
    Description:    Gets all legal moves for a current position and board.
    Params:         cords       Coordinates to check.
                    board       Board to check.
                    history     History of the board / current path.
    Return:         Array of coordinates
    """
    # Get all possible moves for current position.
    legalmoves = possibleMoves( cords, board )

    # Iterate over past moves.
    for move in history:
        # Check if move is in legal moves list.
        if move in legalmoves:
            # If move has been made, it is not legal. Remove.
            legalmoves.remove( move )

    # Return list of legal moves.
    return( legalmoves )

def examineState( cords, board, history ):
    """
    Function:       examineState
    Description:    Examines the current state of the board, and checks to see
```

```
                if a word in the dictionary has been reached.
Params:         cords       Coordinates to check.
                board       Board to check.
                history     History of the board / current path.
Return:         Tuple ( Current Word, In Dictionary(y/N) )
"""
myHist = history                            # Local history copy
newcord = tuple( [ cords[0], cords[1] ] )   # Get current cord as tuple
myHist.append( newcord )                    # Append current cord to history

word = ""       # Establish var for current 'word'
inDict = "no"   # Establish var for if current word is in dictionary

fileobj=open( "twl06.txt" )         # Open dictionary
lines=[]                            # Convert words in dict to list
for line in fileobj:                # Iterate over lines in dict...
    lines.append( line.strip() )    # Strip whitespace and add to word list

for coord in history:                       # For each word in hist...
    word = word + board[coord[0]][coord[1]] # Append the letter...
if word.lower() in lines:                   # Convert finished word to lower
    inDict = "yes"                          # If word in dict, set to "yes"

return( word, inDict )              # Return word and inDict bool
```