Ruby report

This assignment allowed me to understand more about this programming language (pl) and about the concept of object-oriented programming since this pl is fully object oriented. The installation of this pl was easy since I just had to run an executable and some commands to be able to run it on vscode. I struggled the most trying to understand the syntax of the language. It took me around 10 hours to complete the assignment. I was able to flip the board, also implemented functions to check for three in a row to block the opponent and also to win the match without the user selecting the column to put the disk, Also included a function to pop a disk from the board and shift the disks..

The function print adding the .reverse at the beginning allowed me to flip the map so the disks can go to the bottom.

#-------------------------------------------------------------------------

  # Function print returns directly a

  #------------------------------------------------------------------------

  def print

    puts @board.reverse.map {|row| row.map { |e| e || " "}.join("|")}.join("\n")

    puts "\n"

  end

  #------------------------------------------------------------------------

The function hasTree? like the function has won call four methods to check if there are tree disks in a row.

  #-------------------------------------------------------------------------

# Veryfies if has tree in a row for vertical, horizontal,and diagonals

  #------------------------------------------------------------------------

  def hasTree? (player)

    return verticalTree?(player)| horizontalTree?(player) |

           diagonalUpTree?(player)| diagonalDownTree?(player)

  end

The function verticalTree? Works with verticalInRow? To check if there are three disks in a row.

  #-------------------------------------------------------------------------

# Veryfies if there are three vertical disks in a row

  #------------------------------------------------------------------------

  def verticalTree? (player)

    r = 0

    while r <= 3

      c = 0

      while c <= 6

        temp = verticalInRow?(player, r, c) #Check for tree in a row.

        if temp[0] == true

          return temp

          break

        end

        c+=1

      end

      r+=1

    end

    return [false, -10,-10]

  end

  def verticalInRow?(player, r, c)

    if (@board[r][c] == player) && (@board[r+1][c] == player) && (@board[r+2][c] == player)

      temp = [true, r-1, r+3]

      return temp

    else

      temp = [false, -10,-10]

      return temp

    end

  end

The function horizontalTree? With horizontalInRow? Check if there are tree horizontal disks from the same player.

    #-------------------------------------------------------------------------

# Veryfies if there are three horizontal disks in a row

  #------------------------------------------------------------------------

  def horizontalTree? (player)

    r = 0

    temp = 0

    while r <= 5

      c = 0

      while c <= 4

        temp = horizontalInRow?(player, r, c) #Check for tree in a row.

        if temp[0] == true

          return temp

        end

        c+=1

      end

      r+=1

    end

    return temp

  end

  def horizontalInRow?(player, r, c)

    if (@board[r][c] == player) && (@board[r][c+1] == player) && (@board[r][c+2] == player)

      temp = [true, c-1, c+3]

      return temp

    else

      temp = [false, -10, -10]

      return temp

    end

  end

The functions diagonalUpTree? And diagonalUpDown? Works with their respective methods to check if there are tree disk diagonally.

      #-------------------------------------------------------------------------

# Veryfies if there are three diagonal either up or down

  #------------------------------------------------------------------------

  def diagonalUpTree? (player)

    r = 0

    while r <= 4

      c = 0

      while c <= 3

        temp = diagonalUpInRow?(player, r, c)

        if temp[0] == true

          return temp

          break

        end

        c+=1

      end

      r+=1

    end

    return [false, -10,-10]

  end

  def diagonalUpInRow?(player, r, c)

    if (@board[r][c] == player) && (@board[r+1][c+1] == player) && (@board[r+2][c+2] == player) #Check for tree in a row.

      temp = [true, c-1, c+3]

      return temp

    else

      temp = [false, -10,-10]

      return temp

    end

  end

  def diagonalDownTree? (player)

    r = 0

    while r <= 4

      c = 0

      while c <= 3

        temp = diagonalDownInRow?(player, r, c)

        if temp[0] == true

          return temp

          break

        end

        c+=1

      end

      r+=1

    end

    return [false, -10, -10]

  end

  def diagonalDownInRow?(player, r, c)

    if (@board[r][c] == player) && (@board[r-1][c+1] == player) && (@board[r-2][c+2] == player)

      temp = [true, c-1, c+3]

      return [true, c-1, c+3]

    else

      temp = [false, -10, -10]

      return temp

    end

  end

The function popDisk works by popping the last element of the selected column shift the rest one down.

#--------------------------------------------------------------------------

# Pops the disc at the bottom of a row.

  #------------------------------------------------------------------------

  def popDisk(c)

    (0..5).all?{|r| @board[r][c] = @board[r+1][c]} #iterates the board rows

  end

#--------------------------------------------------------------------------

Text

Description automatically generated Text

Description automatically generated

The function robotMove allow us to make the robot decide where to put the disk when there are pieces on the board. It is allowed to block the opponent and is able to put pieces to win a match.

#Robot movement

def robotMove(player, board)

  temp1 = board.hasTree?(:O)

  temp2 = board.hasTree?(player)

  #Blocks the moves of the player

  if (temp1.length == 5) && (temp2.length == 7)

    if (temp1[2]==true) && (temp1[2]==true)

      if temp2[4] >= 0

        board.addDisc(:R, temp2[4])

        return temp2[4]

      else

        board.addDisc(:R, temp2[6])

        return temp2[6]

      end

    end

  end

  if (temp1.length == 5)

    if temp1[2]==true

      if temp1[3] >= 0

        board.addDisc(:R, temp1[3])

        return temp1[3]

      else

        board.addDisc(:R, temp1[4])

        return temp1[4]

      end

    end

  end

  if (temp1.length == 7)

    if temp1[2]==true

      if temp1[4] >= 0

        board.addDisc(:R, temp1[4])

        return temp1[4]

      else

        board.addDisc(:R, temp1[6])

        return temp1[6]

      end

    end

  end

  #Adds to win

  if (temp2.length == 5)

    if temp2[0]==true

      if temp2[1] >= 0

        board.addDisc(:R, temp2[1])

        return temp2[1]

      else

        board.addDisc(:R, temp2[2])

        return temp2[2]

      end

    end

    if temp2[2]==true

      if temp2[3] >= 0

        board.addDisc(:R, temp2[3])

        return temp2[3]

      else

        board.addDisc(:R, temp2[4])

        return temp2[4]

      end

    end

  end

  if (temp2.length == 7)

    if temp2[2]==true

      if temp2[4] >= 0

        board.addDisc(:R, temp2[4])

        return temp2[4]

      else

        board.addDisc(:R, temp2[6])

        return temp2[6]

      end

    end

  else

    temp = rand(0..6)

    puts "Robot entered a disk in column: #{temp}!"

    board.addDisc(player, temp)

    return temp

  end

end

Test case: Horizontal Win

Graphical user interface, text

Description automatically generatedText

Description automatically generated

Test case: Vertical win

Text

Description automatically generated A picture containing text

Description automatically generated

Test case: Diagonal win

Text

Description automatically generated Text

Description automatically generated

Test case: Block Horizontal (Note: blocks left side but not right side.)

Text

Description automatically generated Text

Description automatically generated

---Code---

# Ruby Assignment Code Skeleton

# Nigel Ward, University of Texas at El Paso

# April 2015, April 2019

# borrowing liberally from Gregory Brown's tic-tac-toe game

#------------------------------------------------------------------

# Edited by Emiliano de la Cruz

#------------------------------------------------------------------

class Board

def initialize

@board = [[nil,nil,nil,nil,nil,nil,nil],

[nil,nil,nil,nil,nil,nil,nil],

[nil,nil,nil,nil,nil,nil,nil],

[nil,nil,nil,nil,nil,nil,nil],

[nil,nil,nil,nil,nil,nil,nil],

[nil,nil,nil,nil,nil,nil,nil] ]

end

# process a sequence of moves, each just a column number

def addDiscs(firstPlayer, moves)

if firstPlayer == :R

players = [:R, :O].cycle

else

players = [:O, :R].cycle

end

moves.each {|c| addDisc(players.next, c)}

end

def addDisc(player, column)

if column >= 7 || column < 0

puts " addDisc(#{player},#{column}): out of bounds; move forfeit"

end

firstFreeRow = @board.transpose.slice(column).index(nil)

if firstFreeRow == nil

puts " addDisc(#{player},#{column}): column full already; move forfeit"

end

update(firstFreeRow, column, player)

end

def update(row, col, player)

@board[row][col] = player

end

#-------------------------------------------------------------------------

# Function print returns directly a

#------------------------------------------------------------------------

def print

puts @board.reverse.map {|row| row.map { |e| e || " "}.join("|")}.join("\n")

puts "\n"

end

#------------------------------------------------------------------------

def hasWon? (player)

return verticalWin?(player)| horizontalWin?(player) |

diagonalUpWin?(player)| diagonalDownWin?(player)

end

def verticalWin? (player)

(0..6).any? {|c| (0..2).any? {|r| fourFromTowards?(player, r, c, 1, 0)}}

end

def horizontalWin? (player)

(0..3).any? {|c| (0..5).any? {|r| fourFromTowards?(player, r, c, 0, 1)}}

end

def diagonalUpWin? (player)

(0..3).any? {|c| (0..2).any? {|r| fourFromTowards?(player, r, c, 1, 1)}}

end

def diagonalDownWin? (player)

(0..3).any? {|c| (3..5).any? {|r| fourFromTowards?(player, r, c, -1, 1)}}

end

def fourFromTowards?(player, r, c, dx, dy)

return (0..3).all?{|step| @board[r+step\*dx][c+step\*dy] == player}

end

#-------------------------------------------------------------------------

# Veryfies if has tree in a row for vertical, horizontal,and diagonals

#------------------------------------------------------------------------

def hasTree? (player)

return verticalTree?(player)| horizontalTree?(player) |

diagonalUpTree?(player)| diagonalDownTree?(player)

end

#-------------------------------------------------------------------------

# Veryfies if there are three vertical disks in a row

#------------------------------------------------------------------------

def verticalTree? (player)

r = 0

while r <= 3

c = 0

while c <= 6

temp = verticalInRow?(player, r, c)

if temp[0] == true

return temp

break

end

c+=1

end

r+=1

end

return [false, -10,-10]

end

def verticalInRow?(player, r, c)

if (@board[r][c] == player) && (@board[r+1][c] == player) && (@board[r+2][c] == player)

temp = [true, r-1, r+3]

return temp

else

temp = [false, -10,-10]

return temp

end

end

#-------------------------------------------------------------------------

# Veryfies if there are three horizontal disks in a row

#------------------------------------------------------------------------

def horizontalTree? (player)

r = 0

temp = 0

while r <= 5

c = 0

while c <= 4

temp = horizontalInRow?(player, r, c)

if temp[0] == true

return temp

end

c+=1

end

r+=1

end

return temp

end

def horizontalInRow?(player, r, c)

if (@board[r][c] == player) && (@board[r][c+1] == player) && (@board[r][c+2] == player)

temp = [true, c-1, c+3]

return temp

else

temp = [false, -10, -10]

return temp

end

end

#-------------------------------------------------------------------------

# Veryfies if there are three diagonal either up or down

#------------------------------------------------------------------------

def diagonalUpTree? (player)

r = 0

while r <= 4

c = 0

while c <= 3

temp = diagonalUpInRow?(player, r, c)

if temp[0] == true

return temp

break

end

c+=1

end

r+=1

end

return [false, -10,-10]

end

def diagonalUpInRow?(player, r, c)

if (@board[r][c] == player) && (@board[r+1][c+1] == player) && (@board[r+2][c+2] == player)

temp = [true, c-1, c+3]

return temp

else

temp = [false, -10,-10]

return temp

end

end

def diagonalDownTree? (player)

r = 0

while r <= 4

c = 0

while c <= 3

temp = diagonalDownInRow?(player, r, c)

if temp[0] == true

return temp

break

end

c+=1

end

r+=1

end

return [false, -10, -10]

end

def diagonalDownInRow?(player, r, c)

if (@board[r][c] == player) && (@board[r-1][c+1] == player) && (@board[r-2][c+2] == player)

temp = [true, c-1, c+3]

return [true, c-1, c+3]

else

temp = [false, -10, -10]

return temp

end

end

#--------------------------------------------------------------------------

# Pops the disc at the bottom of a row.

#------------------------------------------------------------------------

def popDisk(c)

(0..5).all?{|r| @board[r][c] = @board[r+1][c]}

end

#--------------------------------------------------------------------------

end # Board

#------------------------------------------------------------------

#Robot movement

def robotMove(player, board)

temp1 = board.hasTree?(:O)

temp2 = board.hasTree?(player)

#Blocks the moves of the player

if (temp1.length == 5) && (temp2.length == 7)

if (temp1[2]==true) && (temp1[2]==true)

if temp2[4] >= 0

board.addDisc(:R, temp2[4])

return temp2[4]

else

board.addDisc(:R, temp2[6])

return temp2[6]

end

end

end

if (temp1.length == 5)

if temp1[2]==true

if temp1[3] >= 0

board.addDisc(:R, temp1[3])

return temp1[3]

else

board.addDisc(:R, temp1[4])

return temp1[4]

end

end

end

if (temp1.length == 7)

if temp1[2]==true

if temp1[4] >= 0

board.addDisc(:R, temp1[4])

return temp1[4]

else

board.addDisc(:R, temp1[6])

return temp1[6]

end

end

end

#Adds to win

if (temp2.length == 5)

if temp2[0]==true

if temp2[1] >= 0

board.addDisc(:R, temp2[1])

return temp2[1]

else

board.addDisc(:R, temp2[2])

return temp2[2]

end

end

if temp2[2]==true

if temp2[3] >= 0

board.addDisc(:R, temp2[3])

return temp2[3]

else

board.addDisc(:R, temp2[4])

return temp2[4]

end

end

end

if (temp2.length == 7)

if temp2[2]==true

if temp2[4] >= 0

board.addDisc(:R, temp2[4])

return temp2[4]

else

board.addDisc(:R, temp2[6])

return temp2[6]

end

end

else

temp = rand(0..6)

puts "Robot entered a disk in column: #{temp}!"

board.addDisc(player, temp)

return temp

end

end

#------------------------------------------------------------------

def testResult(testID, move, targets, intent)

if targets.member?(move)

puts("testResult: passed test #{testID}")

else

puts("testResult: failed test #{testID}: \n moved to #{move}, which wasn't one of #{targets}; \n failed: #{intent}")

end

end

#------------------------------------------------------------------

# test some robot-player behaviors

testboard1 = Board.new

testboard1.addDisc(:R,4)

testboard1.addDisc(:O,4)

testboard1.addDisc(:R,5)

testboard1.addDisc(:O,5)

testboard1.addDisc(:R,6)

testboard1.addDisc(:O,6)

#testboard1.print

testResult(:hwin, robotMove(:R, testboard1),[3], 'robot should take horizontal win')

testboard1.print

puts "#-----------------------------------"

testboard2 = Board.new

testboard2.addDiscs(:R, [3, 1, 3, 2, 3, 4]);

#testboard2.print

testResult(:vwin, robotMove(:R, testboard2), [3], 'robot should take vertical win')

testboard2.print

puts "#-----------------------------------"

testboard3 = Board.new

testboard3.addDiscs(:O, [3, 1, 4, 5, 2, 1, 6, 0, 3, 4, 5, 3, 2, 2, 6 ]);

#testboard3.print

testResult(:dwin, robotMove(:R, testboard3), [3], 'robot should take diagonal win')

testboard3.print

puts "#-----------------------------------"

testboard4 = Board.new

testboard4.addDiscs(:O, [1,1,2,2,3])

#testboard4.print

testResult(:preventHoriz, robotMove(:R, testboard4), [4], 'robot should avoid giving win')

testboard4.print

puts "#-----------------------------------"

testboardpop = Board.new

testboardpop.addDiscs(:O, [1,1,2,2,3,1,5,3,6])

puts "Before pop disc"

testboardpop.print

#Pop elements

testboardpop.popDisk(1)

testboardpop.popDisk(5)

testboardpop.popDisk(2)

puts "After pop disc"

testboardpop.print