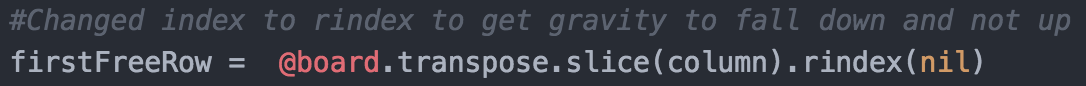
Ruby Assignment

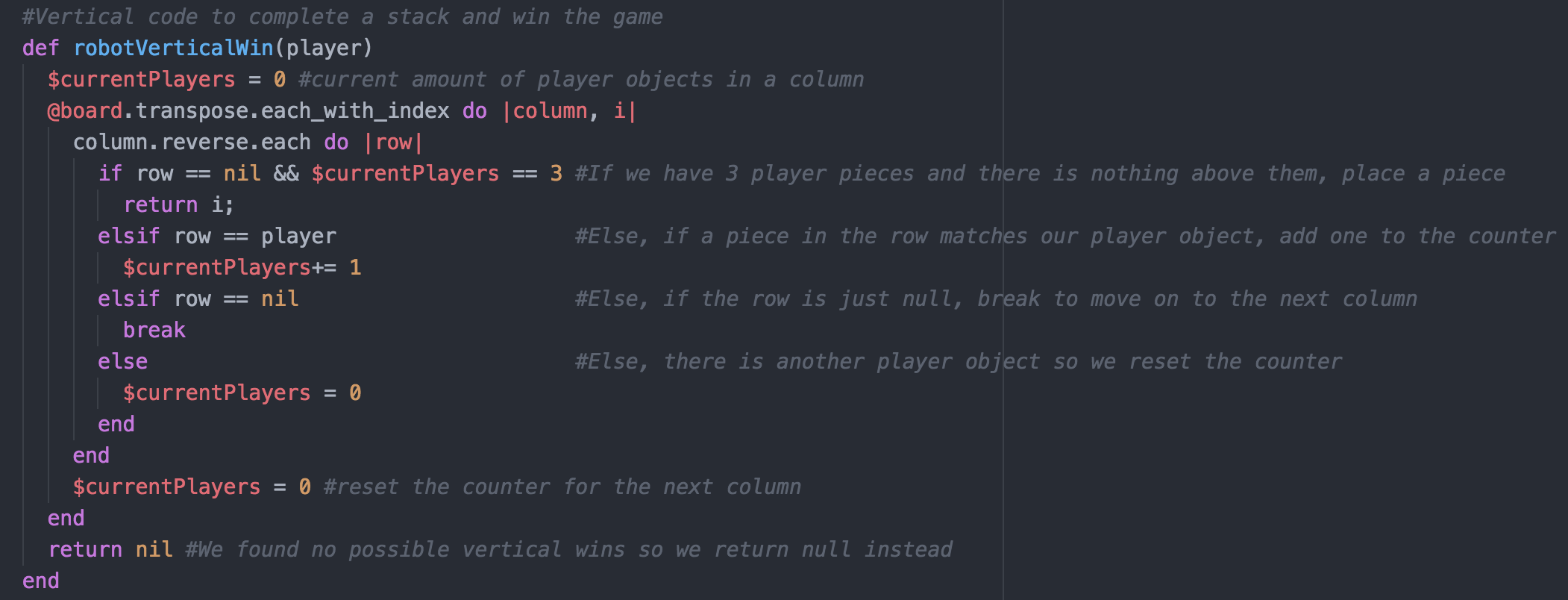
1. **Reverse Gravity so disk fall down:**

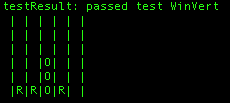




I change the index in “@board.transpose.slice(column).index(nil)” to “rindex” which reverses the index to start at the bottom of the array rather than the top. This allows the program to place pieces at the bottom of the board.

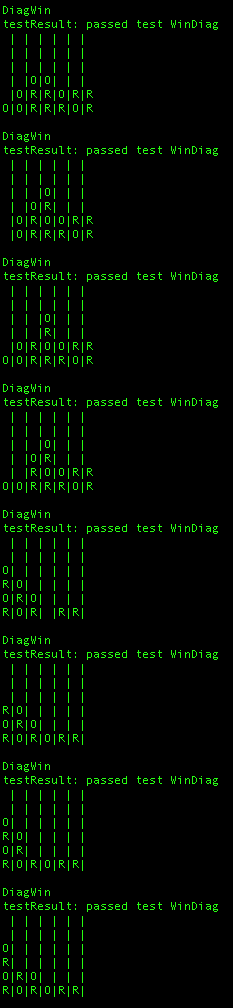
1. **Make the Robot move to achieve vertical wins**





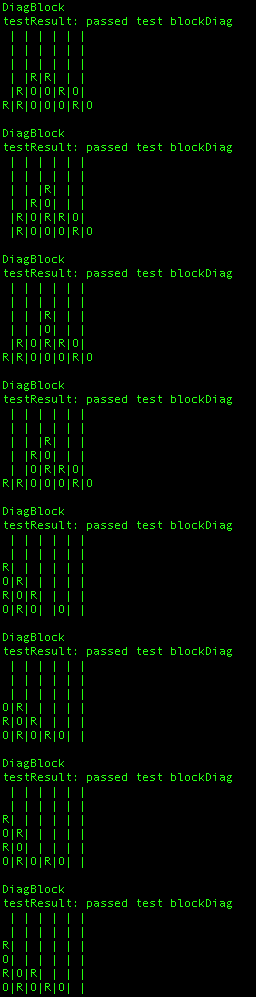
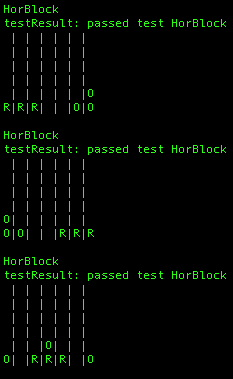
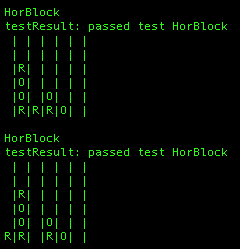
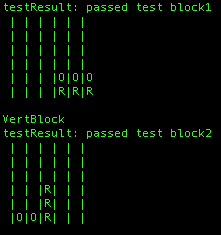
I created a method to allow the robot to achieve a vertical win. So, I first transposed the board array to move column by column rather than row by row. Then I have a counter to count how many times I encounter the robots piece. If I have 3 pieces in a row and the space above it is null, then I return that index so a piece is placed there. If there is another piece above the robots 3rd piece or it is null, I reset the counter and move on to the next column. The counter is also reset if I encounter the other players piece.

1. **Robot moves to achieve diagonal wins**



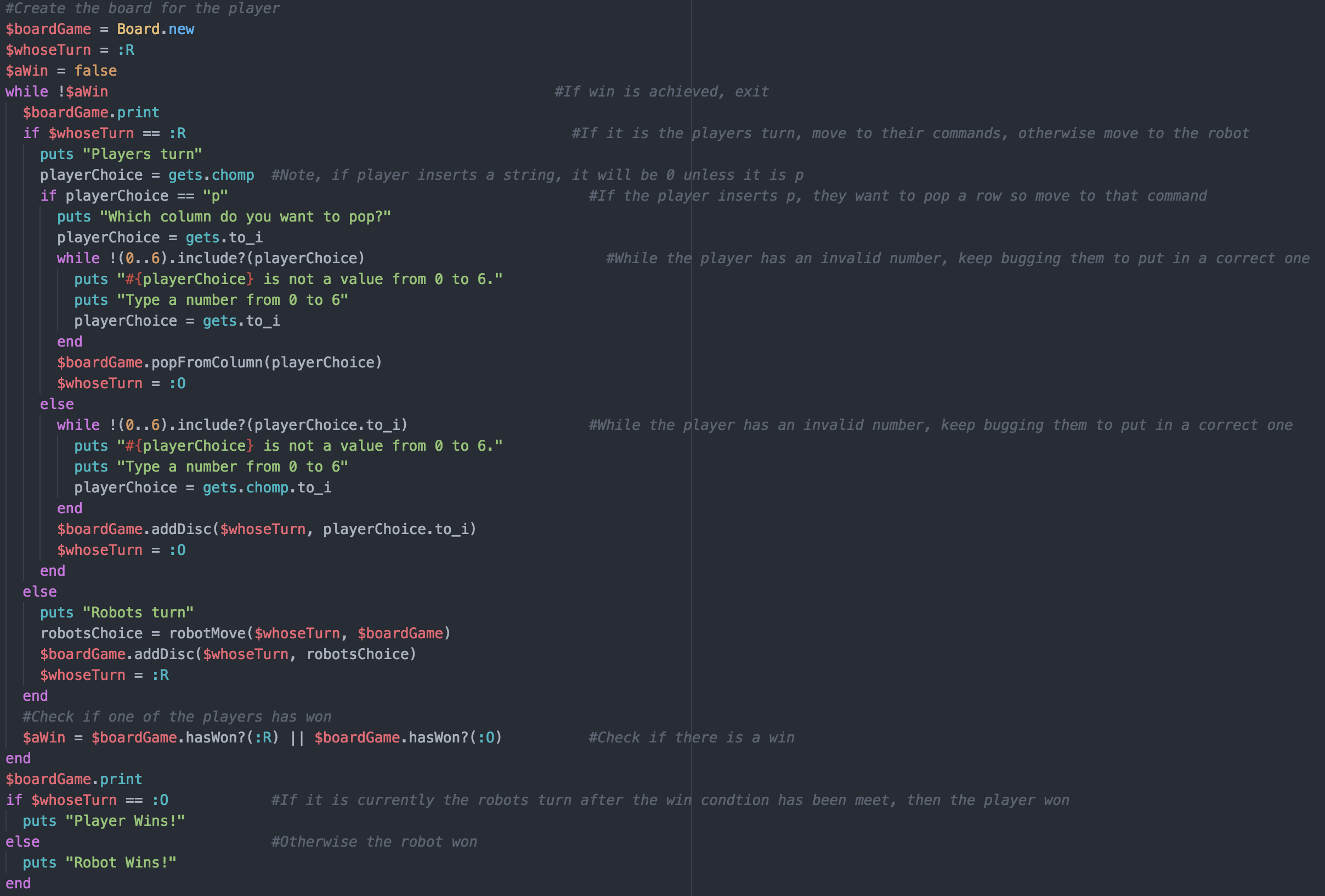
Here is the method that achieve diagonal wins. The method is very long because I do 4 checks. The lower left diagonal, upper left diagonal, lower right diagonal, and the upper right diagonal.

1. **Robot must block when the player can win the next turn**



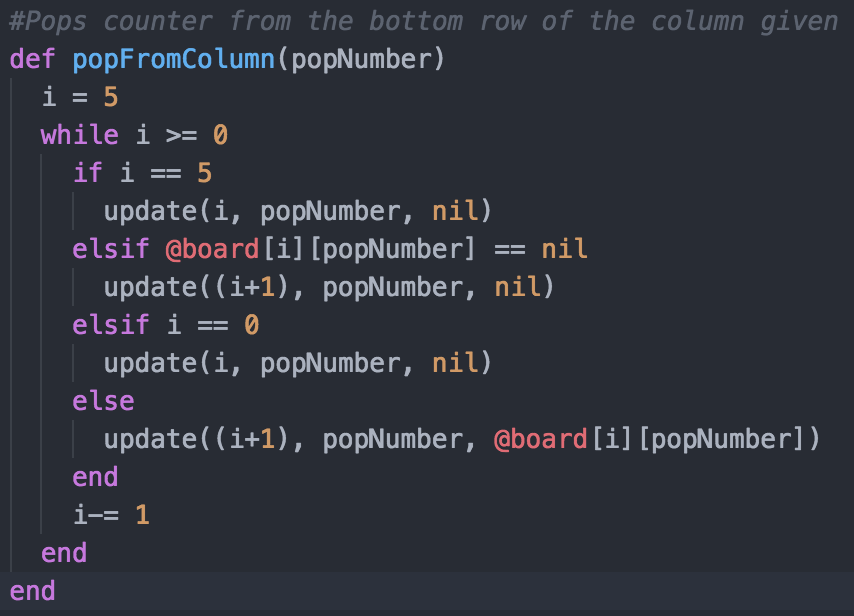
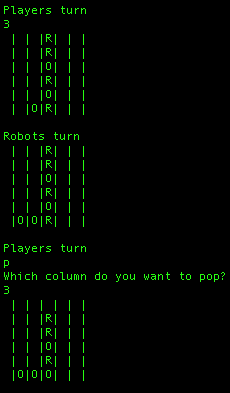
The robot block code is the same as the win code, but instead of looking for the robots piece we look for the other players piece. This allows us to block the other player and prevent them from winning. (Code available towards the end of the assignment).

**h) Add game loop so a player can play with the robot**



This is my function to allow a player to play with the robot. It is a while loop that exits only if one of the players has won. The player types in what column they want to drop the piece in. If it is within the range 0-6, it is placed in the board. If not, they are bugged by the system to put in a valid number to continue the game. If the player types in a string, their choice is defaulted to 0 unless the string is p. The players also have the option of popping a row instead of placing a piece. This is started by typing in p for your choice, then you are prompted to type in which column you want to pop. Players are bugged if it isn’t valid. Once they finish their turn the robot goes next. Once a win has been achieved, we check who won. If it is the robots turn after the win has been achieved, the player won, the robot won otherwise.

**j) Add the option to pop a column, shifting all the pieces down.**



This is the code to pop from a column. Since we are passed what column we want to pop, we use a while loop to go bottom up in a column. We make the bottom null, and shift all the pieces down if there is any.

**Code:**

# Ruby Assignment Code Skeleton

# Nigel Ward, University of Texas at El Paso

# April 2015

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

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

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, columns)

if firstPlayer == :R

players = [:R, :O].cycle

else

players = [:O, :R].cycle

end

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

end

def addDisc(player, column)

if column >= 7 || column < 0

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

return false

end

#Changed index to rindex to get gravity to fall down and not up

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

if firstFreeRow == nil

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

return false

end

update(firstFreeRow, column, player)

return true

end

def update(row, col, player)

@board[row][col] = player

end

def print

puts @board.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

#Pops counter from the bottom row of the column given

def popFromColumn(popNumber)

i = 5

while i >= 0

if i == 5

update(i, popNumber, nil)

elsif @board[i][popNumber] == nil

update((i+1), popNumber, nil)

elsif i == 0

update(i, popNumber, nil)

else

update((i+1), popNumber, @board[i][popNumber])

end

i-= 1

end

end

#---------------------------Robots Moves Code---------------------------------------

#Vertical code to complete a stack and win the game

def robotVerticalWin(player)

$currentPlayers = 0 #current amount of player objects in a column

@board.transpose.each\_with\_index do |column, i|

column.reverse.each do |row|

if row == nil && $currentPlayers == 3 #If we have 3 player pieces and there is nothing above them, place a piece

return i;

elsif row == player #Else, if a piece in the row matches our player object, add one to the counter

$currentPlayers+= 1

elsif row == nil #Else, if the row is just null, break to move on to the next column

break

else #Else, there is another player object so we reset the counter

$currentPlayers = 0

end

end

$currentPlayers = 0 #reset the counter for the next column

end

return nil #We found no possible vertical wins so we return null instead

end

#Vertical code to stop the other player from completing the stack

def robotVerticalBlock(player)

$otherPlayers = 0 #current amount of the other players objects in a column

@board.transpose.each\_with\_index do |column, i|

column.reverse.each do |row|

if row == nil && $otherPlayers == 3 #If the other player has 3 player pieces and there is nothing above them, place a piece

return i;

elsif row != player && row != nil #Else, if a piece in the row matches the other players object, add one to the counter

$otherPlayers+= 1

elsif row == nil #Else, if the row is just null, break to move on to the next column

break

else #Else, our player object is in this position so we reset the counter

$otherPlayers = 0

end

end

$otherPlayers = 0 #reset the counter for the next column

end

return nil #We found no possible vertical blocks so we return null instead

end

#Horizontal code to complete a stack and win the game

def robotHorizontalWin(player)

$currentPlayers = 0 #current amount of player objects in a diagonal

@board.reverse.each do |row|

row.each\_with\_index do |column, i|

if column == player #We found a piece that is the same to the robots

if (i+3) <= 6 #Check if the horizontal move is legal as to not go out of bounds when checking

j = i

while j <= (i+3) #Using our iterator j, move 3 places to the left to see if there is a potential win

if row[j] == player #Increase our counter if we find a piece that is ours

$currentPlayers+= 1

end

j+= 1

end

if $currentPlayers == 3 #If our counter is 3, the robot could win

j = i

while j <= (i+3) #Use our iterator j again to move 3 spaces to the right

if row[j] == nil #if we find an empty space, then return the index

return j

end

j+= 1

end

end

if $currentPlayers == 3 && row[i-1] == nil && row[i+3] != nil && row[i+3] != player && (i-1) >= 0 #if there a piece blocking us from a win to the right, check 1 space behind, if it is empty, return that index

return (i-1)

end

else

$currentPlayers = 0

end

if (i+2) == 6 && row[i-1] == nil && row[i+2] == player && row[i+1] == player #if there are 3 pieces all the way to the right of the board, check behind the first piece, if it empty return that index

return (i-1)

end

end

$currentPlayers = 0

end

end

return nil

end

#Horizontal code to block a stack and prevent a win

def robotHorizontalBlock(player)

$otherPlayers = 0 #current amount of player objects in a diagonal

@board.reverse.each do |row|

row.each\_with\_index do |column, i|

if column != player && column != nil #We found a piece that is the other player

if (i+3) <= 6 #check if the horizontal move is legal as to not go out of bounds when checking

j = i

while j <= (i+3) #Using our iterator j, move 3 places to the left to see if there is a potential block

if row[j] != player && row[j] != nil #Increase our counter if we find a piece that belongs to the other player

$otherPlayers+= 1

end

j+= 1

end

if $otherPlayers == 3 #If our counter is 3, the robot could block

j = i

while j <= (i+3) #Use our iterator j again to move 3 spaces to the right

if row[j] == nil #if we find an empty space, then return the index

return j

end

j+= 1

end

end

if $otherPlayers == 3 && row[i-1] == nil && row[i+3] == player && (i-1) >= 0 #if there a piece blocking the other player from a win to the right, check 1 space behind, if it is empty, return that index

return (i-1)

end

else

$otherPlayers = 0

end

if (i+2) == 6 && row[i-1] == nil && (row[i+2] != player && row[i+2] != nil) && (row[i+1] != player && row[i+1] != nil) #if there are 3 pieces all the way to the right of the board, check behind the first piece, if it empty return that index

return (i-1)

end

end

$otherPlayers = 0

end

end

return nil

end

#Diagonal code to complete the stack and win the game

def robotDiagonalWin(player)

$currentPlayers = 0 #current amount of player objects in a diagonal

@board.each\_with\_index do |row, i|

row.each\_with\_index do |column, j|

if column == player #If we find a a piece that is the same as the robots piece

##########################################Down left diagonal win check##########################################

if (i+3) <= 5 && (j-3) >= 0 #We check if the diagonal is a valid move in this position

#puts "#{i} #{j} Down left" ################Debugging#############################

k = 0

while k <= 3 #We have k to iterate through the diagonal

if @board[i+k][j-k] == player #If the diagonial is equal to our piece, add one to our currentPlayers counter

$currentPlayers+= 1

end

k+= 1

end

if $currentPlayers == 3 #If we have 3 pieces in a diagonal, then we must find the nil area

k = 0

while k <= 3 #We have k to iterate through the diagonal

if @board[i+k][j-k] == nil #If we find the nil area, then we return the column index were the piece will go

return (j-k)

end

k+= 1

end

else

$currentPlayers = 0 #Reset and changes to our currentPlayers counter

end

end

##########################################Up left diagonal win check##########################################

if (i-3) >= 0 && (j-3) >= 0 #We check if the diagonal is a valid move in this position

#puts "#{i} #{j} Up left" ################Debugging#############################

k = 0

while k <= 3 #We have k to iterate through the diagonal

if @board[i-k][j-k] == player #If the diagonial is equal to our piece, add one to our currentPlayers counter

$currentPlayers+= 1

end

k+= 1

end

if $currentPlayers == 3 #If we have 3 pieces in a diagonal, then we must find the nil area

k = 0

while k <= 3 #We have k to iterate through the diagonal

if @board[i-k][j-k] == nil #If we find the nil area, then we return the column index were the piece will go

return (j-k)

end

k+= 1

end

else

$currentPlayers = 0 #Reset and changes to our currentPlayers counter

end

end

##########################################Down right diagonal win check##########################################

if (i+3) <= 5 && (j+3) <= 6 #We check if the diagonal is a valid move in this position

#puts "#{i} #{j} Down right" ################Debugging#############################

k = 0

while k <= 3 #We have k to iterate through the diagonal

if @board[i+k][j+k] == player #If the diagonial is equal to our piece, add one to our currentPlayers counter

$currentPlayers+= 1

end

k+= 1

end

if $currentPlayers == 3 #If we have 3 pieces in a diagonal, then we must find the nil area

k = 0

while k <= 3 #We have k to iterate through the diagonal

if @board[i+k][j+k] == nil #If we find the nil area, then we return the column index were the piece will go

return (j+k)

end

k+= 1

end

else

$currentPlayers = 0 #Reset and changes to our currentPlayers counter

end

end

##########################################Up right diagonal win check##########################################

if (i-3) >= 0 && (j+3) <= 6 #We check if the diagonal is a valid move in this position

#puts "#{i} #{j} Up right" ################Debugging#############################

k = 0

while k <= 3 #We have k to iterate through the diagonal

if @board[i-k][j+k] == player #If the diagonial is equal to our piece, add one to our currentPlayers counter

$currentPlayers+= 1

end

k+= 1

end

if $currentPlayers == 3 #If we have 3 pieces in a diagonal, then we must find the nil area

k = 0

while k <= 3 #We have k to iterate through the diagonal

if @board[i-k][j+k] == nil #If we find the nil area, then we return the column index were the piece will go

return (j+k)

end

k+= 1

end

else

$currentPlayers = 0 #Reset and changes to our currentPlayers counter

end

end

else

$currentPlayers = 0

end

end

$currentPlayers = 0

end

return nil

end

#Diagonal code to block a stack and continue the game

def robotDiagonalBlock(player)

$otherPlayers = 0 #current amount of player objects in a diagonal

@board.each\_with\_index do |row, i|

row.each\_with\_index do |column, j|

if column != player && column != nil

##########################################Down left diagonal block check##########################################

if (i+3) <= 5 && (j-3) >= 0 #We check if the diagonal is a valid move in this position

#puts "#{i} #{j} Down left" ################Debugging#############################

k = 0

while k <= 3 #We have k to iterate through the diagonal

if @board[i+k][j-k] != player && @board[i+k][j-k] != nil #If the diagonial is equal to the other players piece and not nil, add one to our otherPlayers counter

$otherPlayers+= 1

end

k+= 1

end

if $otherPlayers == 3 #If we have 3 pieces in a diagonal, then we must find the nil area to stop the win

k = 0

while k <= 3 #We have k to iterate through the diagonal

if @board[i+k][j-k] == nil #If we find the nil area, then we return the column index were the piece will go

return (j-k)

end

k+= 1

end

else

$otherPlayers = 0 #Reset and changes to our currentPlayers counter

end

end

##########################################Up left diagonal block check##########################################

if (i-3) >= 0 && (j-3) >= 0 #We check if the diagonal is a valid move in this position

#puts "#{i} #{j} Up left" ################Debugging#############################

k = 0

while k <= 3 #We have k to iterate through the diagonal

if @board[i-k][j-k] != player && @board[i-k][j-k] != nil #If the diagonial is equal to the other players piece and not nil, add one to our otherPlayers counter

$otherPlayers+= 1

end

k+= 1

end

if $otherPlayers == 3 #If we have 3 pieces in a diagonal, then we must find the nil area to stop the win

k = 0

while k <= 3 #We have k to iterate through the diagonal

if @board[i-k][j-k] == nil #If we find the nil area, then we return the column index were the piece will go

return (j-k)

end

k+= 1

end

else

$otherPlayers = 0 #Reset and changes to our currentPlayers counter

end

end

##########################################Down right diagonal block check##########################################

if (i+3) <= 5 && (j+3) <= 6 #We check if the diagonal is a valid move in this position

#puts "#{i} #{j} Down right" ################Debugging#############################

k = 0

while k <= 3 #We have k to iterate through the diagonal

if @board[i+k][j+k] != player && @board[i+k][j+k] != nil #If the diagonial is equal to the other players piece and not nil, add one to our otherPlayers counter

$otherPlayers+= 1

end

k+= 1

end

if $otherPlayers == 3 #If we have 3 pieces in a diagonal, then we must find the nil area to stop the win

k = 0

while k <= 3 #We have k to iterate through the diagonal

if @board[i+k][j+k] == nil #If we find the nil area, then we return the column index were the piece will go

return (j+k)

end

k+= 1

end

else

$otherPlayers = 0 #Reset and changes to our currentPlayers counter

end

end

##########################################Up right diagonal block check##########################################

if (i-3) >= 0 && (j+3) <= 6 #We check if the diagonal is a valid move in this position

#puts "#{i} #{j} Up right" ################Debugging#############################

k = 0

while k <= 3 #We have k to iterate through the diagonal

if @board[i-k][j+k] != player && @board[i-k][j+k] != nil #If the diagonial is equal to the other players piece and not nil, add one to our otherPlayers counter

$otherPlayers+= 1

end

k+= 1

end

if $otherPlayers == 3 #If we have 3 pieces in a diagonal, then we must find the nil area to stop the win

k = 0

while k <= 3 #We have k to iterate through the diagonal

if @board[i-k][j+k] == nil #If we find the nil area, then we return the column index were the piece will go

return (j+k)

end

k+= 1

end

else

$otherPlayers = 0 #Reset and changes to our currentPlayers counter

end

end

else

$otherPlayers = 0

end

end

$otherPlayers = 0

end

return nil

end

=begin

def robotHorizontalWin(player)

$currentPlayers = 0 #current amount of player objects in a diagonal

@board.reverse.each do |row|

row.each\_with\_index do |column, i|

if column == player

if (i+3) <= 6

j = i

while j <= (i+3)

if row[j] == player

$currentPlayers+= 1

end

j+= 1

end

if $currentPlayers == 3

j = i

while j <= (i+3)

if row[j] == nil

return j

end

j+= 1

end

end

if $currentPlayers == 3 && row[i+3] != nil && row[i+3] != player && (i-1) >= 0

return (i-1)

end

else

$currentPlayers = 0

end

if (i+2) == 6 && row[i+2] == player && row[i+1] == player

return (i-1)

end

end

$currentPlayers = 0

end

end

return nil

end

=end

end # Board

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

def robotMove(player, board) # stub

#Verticle win

placePlayer = board.robotVerticalWin(player)

if placePlayer != nil

#puts "VertWin"

return placePlayer

end

#Horizontal win

placePlayer = board.robotHorizontalWin(player)

if placePlayer != nil

#puts "HorWin"

return placePlayer

end

#Diagonal win

placePlayer = board.robotDiagonalWin(player)

if placePlayer != nil

#puts "DiagWin"

return placePlayer

end

#Vertical block

placePlayer = board.robotVerticalBlock(player)

if placePlayer != nil

#puts "VertBlock"

return placePlayer

end

#Horizontal block

placePlayer = board.robotHorizontalBlock(player)

if placePlayer != nil

#puts "HorBlock"

return placePlayer

end

#Diagonal Block

placePlayer = board.robotDiagonalBlock(player)

if placePlayer != nil

#puts "DiagBlock"

return placePlayer

end

#default case: randomly set a piece down

return rand(7)

#return 7

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

=begin

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)

testResult(:block1, robotMove(:R, testboard1),[3], 'robot should block horiz')

testboard1.print

testboard2 = Board.new #Passed

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

testResult(:block2, robotMove(:O, testboard2), [3], 'robot should block vert')

testboard2.print

testboard2 = Board.new #Passed

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

testResult(:WinVert, robotMove(:O, testboard2), [3], 'robot should complete stack')

testboard2.print

testboard3 = Board.new

testboard3.addDiscs(:O, [1,1,2,2,3,3])

testResult(:avoid, robotMove(:O, testboard3), [0,4], 'robot should avoid giving win')

testboard3.print

#=end

testboard4 = Board.new

testboard4.addDiscs(:O, [2,2,3,3,4,4])

testResult(:Horwin, robotMove(:O, testboard4), [1,5], 'robot should avoid giving win')

testboard4.print

#Horizontal Tests

testboard5 = Board.new #Passed

testboard5.addDiscs(:O, [4,4,5,5,6,6])

testResult(:HorWin, robotMove(:O, testboard5), [3], 'robot should put piece to win')

testboard5.print

testboard6 = Board.new #Passed

testboard6.addDiscs(:O, [0,0,1,1,2,2])

testResult(:HorWin, robotMove(:O, testboard6), [3], 'robot should put piece to win')

testboard6.print

testboard7 = Board.new #Passed

testboard7.addDiscs(:R, [0,6,1,5,2,6])

testResult(:HorBlock, robotMove(:O, testboard7), [3], 'robot should put piece to block')

testboard7.print

testboard8 = Board.new #Passed

testboard8.addDiscs(:R, [6,0,5,1,4,0])

testResult(:HorBlock, robotMove(:O, testboard8), [3], 'robot should put piece to block')

testboard8.print

testboard9 = Board.new #Passed

testboard9.addDiscs(:R, [2,0,3,3,4,6])

testResult(:HorBlock, robotMove(:O, testboard9), [1,5], 'robot should put piece to block')

testboard9.print

testboard9 = Board.new #Passed

testboard9.addDiscs(:O, [1, 1, 2, 1, 3, 3, 1, 4])

testResult(:HorWin, robotMove(:O, testboard9), [0], 'robot should put piece to block')

testboard9.print

testboard9 = Board.new #Passed

testboard9.addDiscs(:O, [1, 1, 0, 1, 3, 3, 1, 4])

testResult(:HorWin, robotMove(:O, testboard9), [2], 'robot should put piece to block')

testboard9.print

testboard9 = Board.new #Passed

testboard9.addDiscs(:R, [1, 1, 2, 1, 3, 3, 1, 4])

testResult(:HorBlock, robotMove(:O, testboard9), [0], 'robot should put piece to block')

testboard9.print

testboard9 = Board.new #Passed

testboard9.addDiscs(:R, [1, 1, 0, 1, 3, 3, 1, 4])

testResult(:HorBlock, robotMove(:O, testboard9), [2], 'robot should put piece to block')

testboard9.print

#=begin

#Diagonal Tests

testboard2 = Board.new #Passed

testboard2.addDiscs(:R, [3, 1, 4, 5, 2, 1, 6, 0, 3, 4, 5, 3, 2, 2, 6 ]);

testResult(:WinDiag, robotMove(:O, testboard2), [3], 'robot should complete diag')

testboard2.print

testboard10 = Board.new #Passed

testboard10.addDiscs(:R, [3, 1, 4, 5, 2, 1, 6, 3, 3, 4, 5, 3, 2, 2, 6 ]);

testResult(:WinDiag, robotMove(:O, testboard10), [0], 'robot should complete diag')

testboard10.print

testboard11 = Board.new #Passed

testboard11.addDiscs(:R, [3, 1, 4, 5, 2, 1, 6, 3, 3, 4, 5, 3, 2, 0, 6 ]);

testResult(:WinDiag, robotMove(:O, testboard11), [2], 'robot should complete diag')

testboard11.print

testboard12 = Board.new #Passed

testboard12.addDiscs(:R, [3, 1, 4, 5, 2, 0, 6, 3, 3, 4, 5, 3, 2, 2, 6 ]);

testResult(:WinDiag, robotMove(:O, testboard12), [1], 'robot should complete diag')

testboard12.print

testboard13 = Board.new #Passed

testboard13.addDiscs(:R, [0, 1, 1, 0, 0, 0, 2, 1, 4, 2, 5 ]);

testResult(:WinDiag, robotMove(:O, testboard13), [3], 'robot should complete diag')

testboard13.print

testboard13 = Board.new #Passed

testboard13.addDiscs(:R, [0, 1, 1, 0, 0, 3, 2, 1, 4, 2, 5 ]);

testResult(:WinDiag, robotMove(:O, testboard13), [0], 'robot should complete diag')

testboard13.print

testboard13 = Board.new #Passed

testboard13.addDiscs(:R, [0, 1, 1, 0, 0, 3, 2, 1, 4, 0, 5 ]);

testResult(:WinDiag, robotMove(:O, testboard13), [2], 'robot should complete diag')

testboard13.print

testboard13 = Board.new #Passed

testboard13.addDiscs(:R, [0, 1, 1, 0, 0, 3, 2, 2, 4, 0, 5 ]);

testResult(:WinDiag, robotMove(:O, testboard13), [1], 'robot should complete diag')

testboard13.print

#Diagonal Block Tests

testboard2 = Board.new #Passed

testboard2.addDiscs(:O, [3, 1, 4, 5, 2, 1, 6, 0, 3, 4, 5, 3, 2, 2, ]);

testResult(:blockDiag, robotMove(:O, testboard2), [3], 'robot should complete diag')

testboard2.print

testboard10 = Board.new #Passed

testboard10.addDiscs(:O, [3, 1, 4, 5, 2, 1, 6, 3, 3, 4, 5, 3, 2, 2 ]);

testResult(:blockDiag, robotMove(:O, testboard10), [0], 'robot should complete diag')

testboard10.print

testboard11 = Board.new #Passed

testboard11.addDiscs(:O, [3, 1, 4, 5, 2, 1, 6, 3, 3, 4, 5, 3, 2, 0 ]);

testResult(:blockDiag, robotMove(:O, testboard11), [2], 'robot should complete diag')

testboard11.print

testboard12 = Board.new #Passed

testboard12.addDiscs(:O, [3, 1, 4, 5, 2, 0, 6, 3, 3, 4, 5, 3, 2, 2 ]);

testResult(:blockDiag, robotMove(:O, testboard12), [1], 'robot should complete diag')

testboard12.print

testboard13 = Board.new #Passed

testboard13.addDiscs(:O, [0, 1, 1, 0, 0, 0, 2, 1, 4, 2 ]);

testResult(:blockDiag, robotMove(:O, testboard13), [3], 'robot should complete diag')

testboard13.print

testboard13 = Board.new #Passed

testboard13.addDiscs(:O, [0, 1, 1, 0, 0, 3, 2, 1, 4, 2 ]);

testResult(:blockDiag, robotMove(:O, testboard13), [0], 'robot should complete diag')

testboard13.print

testboard13 = Board.new #Passed

testboard13.addDiscs(:O, [0, 1, 1, 0, 0, 3, 2, 1, 4, 0 ]);

testResult(:blockDiag, robotMove(:O, testboard13), [2], 'robot should complete diag')

testboard13.print

testboard13 = Board.new #Passed

testboard13.addDiscs(:O, [0, 1, 1, 0, 0, 3, 2, 2, 4, 0 ]);

testResult(:blockDiag, robotMove(:O, testboard13), [1], 'robot should complete diag')

testboard13.print

=end

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

#Game with a human player

#=begin

#Create the board for the player

$boardGame = Board.new

$whoseTurn = :R

$aWin = false

while !$aWin #If win is achieved, exit

$boardGame.print

if $whoseTurn == :R #If it is the players turn, move to their commands, otherwise move to the robot

puts "Players turn"

playerChoice = gets.chomp #Note, if player inserts a string, it will be 0 unless it is p

if playerChoice == "p" #If the player inserts p, they want to pop a row so move to that command

puts "Which column do you want to pop?"

playerChoice = gets.to\_i

while !(0..6).include?(playerChoice) #While the player has an invalid number, keep bugging them to put in a correct one

puts "#{playerChoice} is not a value from 0 to 6."

puts "Type a number from 0 to 6"

playerChoice = gets.to\_i

end

$boardGame.popFromColumn(playerChoice)

$whoseTurn = :O

else

while !(0..6).include?(playerChoice.to\_i) #While the player has an invalid number, keep bugging them to put in a correct one

puts "#{playerChoice} is not a value from 0 to 6."

puts "Type a number from 0 to 6"

playerChoice = gets.chomp.to\_i

end

$boardGame.addDisc($whoseTurn, playerChoice.to\_i)

$whoseTurn = :O

end

else

puts "Robots turn"

robotsChoice = robotMove($whoseTurn, $boardGame)

$boardGame.addDisc($whoseTurn, robotsChoice)

$whoseTurn = :R

end

#Check if one of the players has won

$aWin = $boardGame.hasWon?(:R) || $boardGame.hasWon?(:O) #Check if there is a win

end

$boardGame.print

if $whoseTurn == :O #If it is currently the robots turn after the win condtion has been meet, then the player won

puts "Player Wins!"

else #Otherwise the robot won

puts "Robot Wins!"

end

#=end