from spike import PrimeHub, LightMatrix, Button, StatusLight, ForceSensor, MotionSensor, Speaker, ColorSensor, App, DistanceSensor, Motor, MotorPair

from spike.control import wait\_for\_seconds, wait\_until, Timer

hub = PrimeHub()

hub.light\_matrix.show\_image('HAPPY')

wait\_for\_seconds(2)

motor\_up = Motor('F')

motor\_down = Motor('B')

head\_colorsensor = ColorSensor('D')

motor\_up.set\_default\_speed(100)

motor\_down.set\_default\_speed(10)

player = 0

robot = 1

class TikTakToeWithRobot:

degree\_for\_move\_one\_line = 30

degree\_for\_move\_one\_row1 = 250

degree\_for\_move\_one\_row2 = 225

whose\_move = ''

robot\_chip\_color = 'violet'

player\_chip\_color = 'black'

tic\_tac\_toe\_matrix = [['', '', ''], ['', '', ''], ['', '', '']]

number\_of\_moves = 0

def play(self):

hub.light\_matrix.off()

while True:

hub.right\_button.wait\_until\_pressed()

new\_tic\_tac\_toe\_matrix = self.get\_new\_tic\_tac\_toe\_matrix()

self.check\_tic\_tac\_toe\_matrix\_for\_player\_move(new\_tic\_tac\_toe\_matrix)

self.show\_light\_matrix()

print(self.tic\_tac\_toe\_matrix)

win = evaluate(self.tic\_tac\_toe\_matrix)

if win == 10:

hub.light\_matrix.write('I WIN!!!')

return

elif win == -10:

hub.light\_matrix.write('You WIN!!!')

return

elif not isMovesLeft(self.tic\_tac\_toe\_matrix):

hub.light\_matrix.write('Draw')

return

line\_move, column\_move = self.get\_move()

self.show\_move(line\_move, column\_move)

self.show\_light\_matrix()

win = evaluate(self.tic\_tac\_toe\_matrix)

if win == 10:

hub.light\_matrix.write('I WIN!!!')

return

elif win == -10:

hub.light\_matrix.write('You WIN!!!')

return

elif not isMovesLeft(self.tic\_tac\_toe\_matrix):

hub.light\_matrix.write('Draw')

return

def show\_light\_matrix(self):

hub.light\_matrix.off()

for line in range(3):

for column in range(3):

if self.tic\_tac\_toe\_matrix[line][column] == player:

hub.light\_matrix.set\_pixel(line, column, 80)

elif self.tic\_tac\_toe\_matrix[line][column] == robot:

hub.light\_matrix.set\_pixel(line, column, 100)

def get\_new\_tic\_tac\_toe\_matrix(self):

new\_tic\_tac\_toe\_matrix = [['', '', ''], ['', '', ''], ['', '', '']]

self.number\_of\_moves = 0

for line in range(3):

for column in range(3):

color = head\_colorsensor.get\_color()

print(color)

if color == "black":

new\_tic\_tac\_toe\_matrix[line][column] = player

hub.light\_matrix.set\_pixel(line, column, 80)

self.number\_of\_moves +=1

elif color == "violet":

new\_tic\_tac\_toe\_matrix[line][column] = robot

hub.light\_matrix.set\_pixel(line, column, 100)

self.number\_of\_moves +=1

else:

hub.light\_matrix.set\_pixel(line, column, 0)

if column == 0:

motor\_up.run\_for\_degrees(self.degree\_for\_move\_one\_row1 \* -1)

elif column == 1:

motor\_up.run\_for\_degrees(self.degree\_for\_move\_one\_row2 \* -1)

motor\_up.run\_for\_degrees(self.degree\_for\_move\_one\_row1 + self.degree\_for\_move\_one\_row2)

if line != 2:

motor\_down.run\_for\_degrees(self.degree\_for\_move\_one\_line \* -1)

motor\_down.run\_to\_position(340)

return new\_tic\_tac\_toe\_matrix

def check\_tic\_tac\_toe\_matrix\_for\_player\_move(self, new\_tic\_tac\_toe\_matrix):  
 ####################

# Recommended place to write code

####################

self.tic\_tac\_toe\_matrix = new\_tic\_tac\_toe\_matrix

return

def get\_move(self):

if self.number\_of\_moves == 0:

return 0,0

elif self.number\_of\_moves == 1:

if self.tic\_tac\_toe\_matrix[0][0] == '':

return 0,0

elif self.tic\_tac\_toe\_matrix[2][0] == '':

return 2,0

elif self.number\_of\_moves == 2:

if self.tic\_tac\_toe\_matrix[1][1] == player:

return 2,2

if self.tic\_tac\_toe\_matrix[0][1] == player or self.tic\_tac\_toe\_matrix[1][0] == player or self.tic\_tac\_toe\_matrix[1][2] == player or self.tic\_tac\_toe\_matrix[2][1] == player:

return 1,1

else:

if self.tic\_tac\_toe\_matrix[2][2] == '':

return 2,2

elif self.tic\_tac\_toe\_matrix[2][0] == '':

return 2,0

else :

return 0, 2

return findBestMove(self.tic\_tac\_toe\_matrix)

def show\_move(self, line, column):

####################

# Recommended place to write code

while not hub.left\_button.was\_pressed():

for i in range(5, 10):

hub.light\_matrix.set\_pixel(line, column, 10 \* i)

wait\_for\_seconds(0.25)

for i in range(0, 5):

hub.light\_matrix.set\_pixel(line, column, 100 - 10 \* i)

wait\_for\_seconds(0.25)

# Recommended place to write code

####################

self.tic\_tac\_toe\_matrix[line][column] = robot

hub.light\_matrix.set\_pixel(line, column, 100)

return

# This function returns true if there are moves

# remaining on the board. It returns false if

# there are no moves left to play.

def isMovesLeft(board):

for i in range(3):

for j in range(3):

if (board[i][j] == ''):

return True

return False

# This is the evaluation function as discussed

# in the previous article ( http://goo.gl/sJgv68 )

def evaluate(b):

# Checking for Rows for 1 or 0 victory.

for row in range(3):

if (b[row][0] == b[row][1] and b[row][1] == b[row][2]):

if (b[row][0] == robot):

return 10

elif (b[row][0] == player):

return -10

# Checking for Columns for 1 or 0 victory.

for col in range(3):

if (b[0][col] == b[1][col] and b[1][col] == b[2][col]):

if (b[0][col] == robot):

return 10

elif (b[0][col] == player):

return -10

# Checking for Diagonals for 1 or 0 victory.

if (b[0][0] == b[1][1] and b[1][1] == b[2][2]):

if (b[0][0] == robot):

return 10

elif (b[0][0] == player):

return -10

if (b[0][2] == b[1][1] and b[1][1] == b[2][0]):

if (b[0][2] == robot):

return 10

elif (b[0][2] == player):

return -10

# Else if none of them have won then return 0

return 0

# This is the minimax function. It considers all

# the possible ways the game can go and returns

# the value of the board

def minimax(board, depth, isMax):

score = evaluate(board)

# If Maximizer has won the game return his/her

# evaluated score

if (score == 10):

return score

# If Minimizer has won the game return his/her

# evaluated score

if (score == -10):

return score

# If there are no more moves and no winner then

# it is a tie

if (isMovesLeft(board) == False):

return 0

# If this maximizer's move

if (isMax):

best = -1000

# Traverse all cells

for i in range(3):

for j in range(3):

# Check if cell is empty

if (board[i][j] == ''):

# Make the move

board[i][j] = robot

# Call minimax recursively and choose

# the maximum value

best = max(best, minimax(board,

depth + 1,

not isMax))

# Undo the move

board[i][j] = ''

return best

# If this minimizer's move

else:

best = 1000

# Traverse all cells

for i in range(3):

for j in range(3):

# Check if cell is empty

if (board[i][j] == ''):

# Make the move

board[i][j] = player

# Call minimax recursively and choose

# the minimum value

best = min(best, minimax(board, depth + 1, not isMax))

# Undo the move

board[i][j] = ''

return best

# This will return the best possible move for the player

def findBestMove(board):

bestVal = -1000

bestMove = (-1, -1)

# Traverse all cells, evaluate minimax function for

# all empty cells. And return the cell with optimal

# value.

for i in range(3):

for j in range(3):

# Check if cell is empty

if (board[i][j] == ''):

# Make the move

board[i][j] = robot

# compute evaluation function for this

# move.

moveVal = minimax(board, 0, False)

# Undo the move

board[i][j] = ''

# If the value of the current move is

# more than the best value, then update

# best/

print("The value of the Move is :", moveVal, "for", i, j)

if (moveVal > bestVal):

bestMove = (i, j)

bestVal = moveVal

print("The value of the best Move is :", bestVal)

print()

return bestMove

game = TikTakToeWithRobot()

game.play()