# Flippy (an Othello or Reversi clone)

# By Al Sweigart al@inventwithpython.com

# http://inventwithpython.com/pygame

# Released under a "Simplified BSD" license

# Based on the "reversi.py" code that originally appeared in "Invent

# Your Own Computer Games with Python", chapter 15:

# http://inventwithpython.com/chapter15.html

import random, sys, pygame, time, copy

from pygame.locals import \*

FPS = 10 # frames per second to update the screen

WINDOWWIDTH = 640 # width of the program's window, in pixels

WINDOWHEIGHT = 480 # height in pixels

SPACESIZE = 50 # width & height of each space on the board, in pixels

BOARDWIDTH = 8 # how many columns of spaces on the game board

BOARDHEIGHT = 8 # how many rows of spaces on the game board

WHITE\_TILE = 'WHITE\_TILE' # an arbitrary but unique value

BLACK\_TILE = 'BLACK\_TILE' # an arbitrary but unique value

EMPTY\_SPACE = 'EMPTY\_SPACE' # an arbitrary but unique value

HINT\_TILE = 'HINT\_TILE' # an arbitrary but unique value

ANIMATIONSPEED = 25 # integer from 1 to 100, higher is faster animation

# Amount of space on the left & right side (XMARGIN) or above and below

# (YMARGIN) the game board, in pixels.

XMARGIN = int((WINDOWWIDTH - (BOARDWIDTH \* SPACESIZE)) / 2)

YMARGIN = int((WINDOWHEIGHT - (BOARDHEIGHT \* SPACESIZE)) / 2)

# R G B

WHITE = (255, 255, 255)

BLACK = ( 0, 0, 0)

GREEN = ( 0, 155, 0)

BRIGHTBLUE = ( 0, 50, 255)

BROWN = (174, 94, 0)

TEXTBGCOLOR1 = BRIGHTBLUE

TEXTBGCOLOR2 = GREEN

GRIDLINECOLOR = BLACK

TEXTCOLOR = WHITE

HINTCOLOR = BROWN

def main():

global MAINCLOCK, DISPLAYSURF, FONT, BIGFONT, BGIMAGE

pygame.init()

MAINCLOCK = pygame.time.Clock()

DISPLAYSURF = pygame.display.set\_mode((WINDOWWIDTH, WINDOWHEIGHT))

pygame.display.set\_caption('Flippy')

FONT = pygame.font.Font('freesansbold.ttf', 16)

BIGFONT = pygame.font.Font('freesansbold.ttf', 32)

# Set up the background image.

boardImage = pygame.image.load('flippyboard.png')

# Use smoothscale() to stretch the board image to fit the entire board:

boardImage = pygame.transform.smoothscale(boardImage, (BOARDWIDTH \* SPACESIZE, BOARDHEIGHT \* SPACESIZE))

boardImageRect = boardImage.get\_rect()

boardImageRect.topleft = (XMARGIN, YMARGIN)

BGIMAGE = pygame.image.load('flippybackground.png')

# Use smoothscale() to stretch the background image to fit the entire window:

BGIMAGE = pygame.transform.smoothscale(BGIMAGE, (WINDOWWIDTH, WINDOWHEIGHT))

BGIMAGE.blit(boardImage, boardImageRect)

# Run the main game.

while True:

if runGame() == False:

break

def runGame():

# Plays a single game of reversi each time this function is called.

# Reset the board and game.

mainBoard = getNewBoard()

resetBoard(mainBoard)

showHints = False

turn = random.choice(['computer', 'player'])

# Draw the starting board and ask the player what color they want.

drawBoard(mainBoard)

playerTile, computerTile = enterPlayerTile()

# Make the Surface and Rect objects for the "New Game" and "Hints" buttons

newGameSurf = FONT.render('New Game', True, TEXTCOLOR, TEXTBGCOLOR2)

newGameRect = newGameSurf.get\_rect()

newGameRect.topright = (WINDOWWIDTH - 8, 10)

hintsSurf = FONT.render('Hints', True, TEXTCOLOR, TEXTBGCOLOR2)

hintsRect = hintsSurf.get\_rect()

hintsRect.topright = (WINDOWWIDTH - 8, 40)

while True: # main game loop

# Keep looping for player and computer's turns.

if turn == 'player':

# Player's turn:

if getValidMoves(mainBoard, playerTile) == []:

# If it's the player's turn but they

# can't move, then end the game.

break

movexy = None

while movexy == None:

# Keep looping until the player clicks on a valid space.

# Determine which board data structure to use for display.

if showHints:

boardToDraw = getBoardWithValidMoves(mainBoard, playerTile)

else:

boardToDraw = mainBoard

checkForQuit()

for event in pygame.event.get(): # event handling loop

if event.type == MOUSEBUTTONUP:

# Handle mouse click events

mousex, mousey = event.pos

if newGameRect.collidepoint( (mousex, mousey) ):

# Start a new game

return True

elif hintsRect.collidepoint( (mousex, mousey) ):

# Toggle hints mode

showHints = not showHints

# movexy is set to a two-item tuple XY coordinate, or None value

movexy = getSpaceClicked(mousex, mousey)

if movexy != None and not isValidMove(mainBoard, playerTile, movexy[0], movexy[1]):

movexy = None

# Draw the game board.

drawBoard(boardToDraw)

drawInfo(boardToDraw, playerTile, computerTile, turn)

# Draw the "New Game" and "Hints" buttons.

DISPLAYSURF.blit(newGameSurf, newGameRect)

DISPLAYSURF.blit(hintsSurf, hintsRect)

MAINCLOCK.tick(FPS)

pygame.display.update()

# Make the move and end the turn.

makeMove(mainBoard, playerTile, movexy[0], movexy[1], True)

if getValidMoves(mainBoard, computerTile) != []:

# Only set for the computer's turn if it can make a move.

turn = 'computer'

else:

# Computer's turn:

if getValidMoves(mainBoard, computerTile) == []:

# If it was set to be the computer's turn but

# they can't move, then end the game.

break

# Draw the board.

drawBoard(mainBoard)

drawInfo(mainBoard, playerTile, computerTile, turn)

# Draw the "New Game" and "Hints" buttons.

DISPLAYSURF.blit(newGameSurf, newGameRect)

DISPLAYSURF.blit(hintsSurf, hintsRect)

# Make it look like the computer is thinking by pausing a bit.

pauseUntil = time.time() + random.randint(5, 15) \* 0.1

while time.time() < pauseUntil:

pygame.display.update()

# Make the move and end the turn.

x, y = getComputerMove(mainBoard, computerTile)

makeMove(mainBoard, computerTile, x, y, True)

if getValidMoves(mainBoard, playerTile) != []:

# Only set for the player's turn if they can make a move.

turn = 'player'

# Display the final score.

drawBoard(mainBoard)

scores = getScoreOfBoard(mainBoard)

# Determine the text of the message to display.

if scores[playerTile] > scores[computerTile]:

text = 'You beat the computer by %s points! Congratulations!' % \

(scores[playerTile] - scores[computerTile])

elif scores[playerTile] < scores[computerTile]:

text = 'You lost. The computer beat you by %s points.' % \

(scores[computerTile] - scores[playerTile])

else:

text = 'The game was a tie!'

textSurf = FONT.render(text, True, TEXTCOLOR, TEXTBGCOLOR1)

textRect = textSurf.get\_rect()

textRect.center = (int(WINDOWWIDTH / 2), int(WINDOWHEIGHT / 2))

DISPLAYSURF.blit(textSurf, textRect)

# Display the "Play again?" text with Yes and No buttons.

text2Surf = BIGFONT.render('Play again?', True, TEXTCOLOR, TEXTBGCOLOR1)

text2Rect = text2Surf.get\_rect()

text2Rect.center = (int(WINDOWWIDTH / 2), int(WINDOWHEIGHT / 2) + 50)

# Make "Yes" button.

yesSurf = BIGFONT.render('Yes', True, TEXTCOLOR, TEXTBGCOLOR1)

yesRect = yesSurf.get\_rect()

yesRect.center = (int(WINDOWWIDTH / 2) - 60, int(WINDOWHEIGHT / 2) + 90)

# Make "No" button.

noSurf = BIGFONT.render('No', True, TEXTCOLOR, TEXTBGCOLOR1)

noRect = noSurf.get\_rect()

noRect.center = (int(WINDOWWIDTH / 2) + 60, int(WINDOWHEIGHT / 2) + 90)

while True:

# Process events until the user clicks on Yes or No.

checkForQuit()

for event in pygame.event.get(): # event handling loop

if event.type == MOUSEBUTTONUP:

mousex, mousey = event.pos

if yesRect.collidepoint( (mousex, mousey) ):

return True

elif noRect.collidepoint( (mousex, mousey) ):

return False

DISPLAYSURF.blit(textSurf, textRect)

DISPLAYSURF.blit(text2Surf, text2Rect)

DISPLAYSURF.blit(yesSurf, yesRect)

DISPLAYSURF.blit(noSurf, noRect)

pygame.display.update()

MAINCLOCK.tick(FPS)

def translateBoardToPixelCoord(x, y):

return XMARGIN + x \* SPACESIZE + int(SPACESIZE / 2), YMARGIN + y \* SPACESIZE + int(SPACESIZE / 2)

def animateTileChange(tilesToFlip, tileColor, additionalTile):

# Draw the additional tile that was just laid down. (Otherwise we'd

# have to completely redraw the board & the board info.)

if tileColor == WHITE\_TILE:

additionalTileColor = WHITE

else:

additionalTileColor = BLACK

additionalTileX, additionalTileY = translateBoardToPixelCoord(additionalTile[0], additionalTile[1])

pygame.draw.circle(DISPLAYSURF, additionalTileColor, (additionalTileX, additionalTileY), int(SPACESIZE / 2) - 4)

pygame.display.update()

for rgbValues in range(0, 255, int(ANIMATIONSPEED \* 2.55)):

if rgbValues > 255:

rgbValues = 255

elif rgbValues < 0:

rgbValues = 0

if tileColor == WHITE\_TILE:

color = tuple([rgbValues] \* 3) # rgbValues goes from 0 to 255

elif tileColor == BLACK\_TILE:

color = tuple([255 - rgbValues] \* 3) # rgbValues goes from 255 to 0

for x, y in tilesToFlip:

centerx, centery = translateBoardToPixelCoord(x, y)

pygame.draw.circle(DISPLAYSURF, color, (centerx, centery), int(SPACESIZE / 2) - 4)

pygame.display.update()

MAINCLOCK.tick(FPS)

checkForQuit()

def drawBoard(board):

# Draw background of board.

DISPLAYSURF.blit(BGIMAGE, BGIMAGE.get\_rect())

# Draw grid lines of the board.

for x in range(BOARDWIDTH + 1):

# Draw the horizontal lines.

startx = (x \* SPACESIZE) + XMARGIN

starty = YMARGIN

endx = (x \* SPACESIZE) + XMARGIN

endy = YMARGIN + (BOARDHEIGHT \* SPACESIZE)

pygame.draw.line(DISPLAYSURF, GRIDLINECOLOR, (startx, starty), (endx, endy))

for y in range(BOARDHEIGHT + 1):

# Draw the vertical lines.

startx = XMARGIN

starty = (y \* SPACESIZE) + YMARGIN

endx = XMARGIN + (BOARDWIDTH \* SPACESIZE)

endy = (y \* SPACESIZE) + YMARGIN

pygame.draw.line(DISPLAYSURF, GRIDLINECOLOR, (startx, starty), (endx, endy))

# Draw the black & white tiles or hint spots.

for x in range(BOARDWIDTH):

for y in range(BOARDHEIGHT):

centerx, centery = translateBoardToPixelCoord(x, y)

if board[x][y] == WHITE\_TILE or board[x][y] == BLACK\_TILE:

if board[x][y] == WHITE\_TILE:

tileColor = WHITE

else:

tileColor = BLACK

pygame.draw.circle(DISPLAYSURF, tileColor, (centerx, centery), int(SPACESIZE / 2) - 4)

if board[x][y] == HINT\_TILE:

pygame.draw.rect(DISPLAYSURF, HINTCOLOR, (centerx - 4, centery - 4, 8, 8))

def getSpaceClicked(mousex, mousey):

# Return a tuple of two integers of the board space coordinates where

# the mouse was clicked. (Or returns None not in any space.)

for x in range(BOARDWIDTH):

for y in range(BOARDHEIGHT):

if mousex > x \* SPACESIZE + XMARGIN and \

mousex < (x + 1) \* SPACESIZE + XMARGIN and \

mousey > y \* SPACESIZE + YMARGIN and \

mousey < (y + 1) \* SPACESIZE + YMARGIN:

return (x, y)

return None

def drawInfo(board, playerTile, computerTile, turn):

# Draws scores and whose turn it is at the bottom of the screen.

scores = getScoreOfBoard(board)

scoreSurf = FONT.render("Player Score: %s Computer Score: %s %s's Turn" % (str(scores[playerTile]), str(scores[computerTile]), turn.title()), True, TEXTCOLOR)

scoreRect = scoreSurf.get\_rect()

scoreRect.bottomleft = (10, WINDOWHEIGHT - 5)

DISPLAYSURF.blit(scoreSurf, scoreRect)

def resetBoard(board):

# Blanks out the board it is passed, and sets up starting tiles.

for x in range(BOARDWIDTH):

for y in range(BOARDHEIGHT):

board[x][y] = EMPTY\_SPACE

# Add starting pieces to the center

board[3][3] = WHITE\_TILE

board[3][4] = BLACK\_TILE

board[4][3] = BLACK\_TILE

board[4][4] = WHITE\_TILE

def getNewBoard():

# Creates a brand new, empty board data structure.

board = []

for i in range(BOARDWIDTH):

board.append([EMPTY\_SPACE] \* BOARDHEIGHT)

return board

def isValidMove(board, tile, xstart, ystart):

# Returns False if the player's move is invalid. If it is a valid

# move, returns a list of spaces of the captured pieces.

if board[xstart][ystart] != EMPTY\_SPACE or not isOnBoard(xstart, ystart):

return False

board[xstart][ystart] = tile # temporarily set the tile on the board.

if tile == WHITE\_TILE:

otherTile = BLACK\_TILE

else:

otherTile = WHITE\_TILE

tilesToFlip = []

# check each of the eight directions:

for xdirection, ydirection in [[0, 1], [1, 1], [1, 0], [1, -1], [0, -1], [-1, -1], [-1, 0], [-1, 1]]:

x, y = xstart, ystart

x += xdirection

y += ydirection

if isOnBoard(x, y) and board[x][y] == otherTile:

# The piece belongs to the other player next to our piece.

x += xdirection

y += ydirection

if not isOnBoard(x, y):

continue

while board[x][y] == otherTile:

x += xdirection

y += ydirection

if not isOnBoard(x, y):

break # break out of while loop, continue in for loop

if not isOnBoard(x, y):

continue

if board[x][y] == tile:

# There are pieces to flip over. Go in the reverse

# direction until we reach the original space, noting all

# the tiles along the way.

while True:

x -= xdirection

y -= ydirection

if x == xstart and y == ystart:

break

tilesToFlip.append([x, y])

board[xstart][ystart] = EMPTY\_SPACE # make space empty

if len(tilesToFlip) == 0: # If no tiles flipped, this move is invalid

return False

return tilesToFlip

def isOnBoard(x, y):

# Returns True if the coordinates are located on the board.

return x >= 0 and x < BOARDWIDTH and y >= 0 and y < BOARDHEIGHT

def getBoardWithValidMoves(board, tile):

# Returns a new board with hint markings.

dupeBoard = copy.deepcopy(board)

for x, y in getValidMoves(dupeBoard, tile):

dupeBoard[x][y] = HINT\_TILE

return dupeBoard

def getValidMoves(board, tile):

# Returns a list of (x,y) tuples of all valid moves.

validMoves = []

for x in range(BOARDWIDTH):

for y in range(BOARDHEIGHT):

if isValidMove(board, tile, x, y) != False:

validMoves.append((x, y))

return validMoves

def getScoreOfBoard(board):

# Determine the score by counting the tiles.

xscore = 0

oscore = 0

for x in range(BOARDWIDTH):

for y in range(BOARDHEIGHT):

if board[x][y] == WHITE\_TILE:

xscore += 1

if board[x][y] == BLACK\_TILE:

oscore += 1

return {WHITE\_TILE:xscore, BLACK\_TILE:oscore}

def enterPlayerTile():

# Draws the text and handles the mouse click events for letting

# the player choose which color they want to be. Returns

# [WHITE\_TILE, BLACK\_TILE] if the player chooses to be White,

# [BLACK\_TILE, WHITE\_TILE] if Black.

# Create the text.

textSurf = FONT.render('Do you want to be white or black?', True, TEXTCOLOR, TEXTBGCOLOR1)

textRect = textSurf.get\_rect()

textRect.center = (int(WINDOWWIDTH / 2), int(WINDOWHEIGHT / 2))

xSurf = BIGFONT.render('White', True, TEXTCOLOR, TEXTBGCOLOR1)

xRect = xSurf.get\_rect()

xRect.center = (int(WINDOWWIDTH / 2) - 60, int(WINDOWHEIGHT / 2) + 40)

oSurf = BIGFONT.render('Black', True, TEXTCOLOR, TEXTBGCOLOR1)

oRect = oSurf.get\_rect()

oRect.center = (int(WINDOWWIDTH / 2) + 60, int(WINDOWHEIGHT / 2) + 40)

while True:

# Keep looping until the player has clicked on a color.

checkForQuit()

for event in pygame.event.get(): # event handling loop

if event.type == MOUSEBUTTONUP:

mousex, mousey = event.pos

if xRect.collidepoint( (mousex, mousey) ):

return [WHITE\_TILE, BLACK\_TILE]

elif oRect.collidepoint( (mousex, mousey) ):

return [BLACK\_TILE, WHITE\_TILE]

# Draw the screen.

DISPLAYSURF.blit(textSurf, textRect)

DISPLAYSURF.blit(xSurf, xRect)

DISPLAYSURF.blit(oSurf, oRect)

pygame.display.update()

MAINCLOCK.tick(FPS)

def makeMove(board, tile, xstart, ystart, realMove=False):

# Place the tile on the board at xstart, ystart, and flip tiles

# Returns False if this is an invalid move, True if it is valid.

tilesToFlip = isValidMove(board, tile, xstart, ystart)

if tilesToFlip == False:

return False

board[xstart][ystart] = tile

if realMove:

animateTileChange(tilesToFlip, tile, (xstart, ystart))

for x, y in tilesToFlip:

board[x][y] = tile

return True

def isOnCorner(x, y):

# Returns True if the position is in one of the four corners.

return (x == 0 and y == 0) or \

(x == BOARDWIDTH and y == 0) or \

(x == 0 and y == BOARDHEIGHT) or \

(x == BOARDWIDTH and y == BOARDHEIGHT)

def getComputerMove(board, computerTile):

# Given a board and the computer's tile, determine where to

# move and return that move as a [x, y] list.

possibleMoves = getValidMoves(board, computerTile)

# randomize the order of the possible moves

random.shuffle(possibleMoves)

# always go for a corner if available.

for x, y in possibleMoves:

if isOnCorner(x, y):

return [x, y]

# Go through all possible moves and remember the best scoring move

bestScore = -1

for x, y in possibleMoves:

dupeBoard = copy.deepcopy(board)

makeMove(dupeBoard, computerTile, x, y)

score = getScoreOfBoard(dupeBoard)[computerTile]

if score > bestScore:

bestMove = [x, y]

bestScore = score

return bestMove

def checkForQuit():

for event in pygame.event.get((QUIT, KEYUP)): # event handling loop

if event.type == QUIT or (event.type == KEYUP and event.key == K\_ESCAPE):

pygame.quit()

sys.exit()

if \_\_name\_\_ == '\_\_main\_\_':

main()

**function** minimax(node, depth, maximizingPlayer) **is**

**if** depth = 0 **or** node is a terminal node **then**

**return** the heuristic value of node

**if** maximizingPlayer **then**

value := −∞

**for each** child of node **do**

value := max(value, minimax(child, depth − 1, FALSE))

**return** value

**else** *(\* minimizing player \*)*

value := +∞

**for each** child of node **do**

value := min(value, minimax(child, depth − 1, TRUE))

**return** value