

Developing Tic-Tac-Toe with pyGame

Mr. Yergler
Dr. Ceder
Canterbury School

16 May 2002

Abstract

pyGame is a set of Python modules designed for writing games. pyGame serves as an abstraction layer for the SDL library, which means that games written with pyGame can be run on Windows, Linux, MacOS, and BeOS, just to name a few. This tutorial will guide you through writing a simple game in Python (tic-tac-toe) using the pyGame library. By the completion of the tutorial, you should have a grasp of the basic concepts of Python and pyGame.

1 Program Initialization

In developing tic-tac-toe, we're going to address the problem using logical steps, or blocks. The goal is to learn a single pyGame concept at each step. Our first step is to set up program and initialize the pyGame libraries. The following code sample demonstrates this:

```
# tic-tac-toe
# pyGame Tutorial
# Mr. Yergler
# 13 May 2002

import pygame
from pygame.locals import *

pygame.init()
ttt = pygame.display.set_mode((300,325))
pygame.display.set_caption = ('Tic-Tac-Toe')
```

The first four lines of code are comments, which Python ignores. Comments are used solely for informational purposes and to improve the “readability” of your code. In Python, any line beginning with a # is a comment.

The next two lines (import..., from pygame...) tell Python to load the pyGame libraries. This is a good time to note that Python is case sensitive, so pygame is not the same as PYGAME.

The final three lines call the pyGame initialization routines, create a window, and set its title bar caption. In this case, we're creating a 300x325 pixel window for our program.

Note that the empty parenthesis after `pygame.init()` are required. The double parenthesis around the window dimensions, `((300,325))`, are also required.

2 The Main Loop

In order to have a working Windows program we need to add some code to handle "events", like mouse clicks, key presses, etc. So let's do that now. PyGame processes events in a queue. That is, if you click the mouse, then press a key, the mouse click is processed before the keystroke.

To respond to events, we use a while loop. This basic loop is present below

```
# the initialization stuff goes here

#set up our loop control variable
running = 1

while (running==1):
    for event in pygame.event.get():
        if event.type is QUIT:
            running = 0
```

This loop basically says that until the user quits the game (generally by closing the window), continually check for events and update the display. In the example above, the only event we check for is quitting and the program doesn't do anything except stop, but now that the program skeleton works, we can add actual program code and behavior.

3 Drawing the board

Our next step is to draw the tic-tac-toe board itself. To do this, we're going to write a function. A function is a self-contained "package" of commands we can use from our main program. In Python functions begin with the `def` keyword. We're going to call our function `initBoard`. The initial function declaration is shown below:

```
def initBoard(ttt):
    # initialize the board and return it
    # as a variable
    # -----
    # ttt : a properly initialized display variable
```

```

    # return the board
    return background

```

The first line of code defines the function. Python uses indentation to organize code into logical blocks. Therefore, the entire function must be indented an equal amount. In its current form, this function simply returns the value of the variable `background` (which is undefined).

To actually initialize the board, we use the following code:

```

background = pygame.Surface (ttt.get_size())
background = background.convert()
background.fill ((250,250,250))

```

This code does three things. In the first two lines we define `background` as a “surface” variable. A pyGame surface is anything we can draw or place graphics on. The final line fills the background with white. The tuple `(250,250,250)` is the RGB representation for white. At this point our function would return a white background. Our next step is to draw the lines on the background. We do that with the following code:

```

# draw the grid lines
# vertical lines...
pygame.draw.line (background, (0,0,0), (100,0), (100,300), 2)
pygame.draw.line (background, (0,0,0), (200,0), (200,300), 2)

# horizontal lines...
pygame.draw.line (background, (0,0,0), (0,100), (300,100), 2)
pygame.draw.line (background, (0,0,0), (0,200), (300,200), 2)

```

The `pygame.draw.line` function call shown above takes 5 parameters. The first, `background`, tells pyGame what surface to do the drawing on. The second, `(0,0,0)`, is the RGB representation for the color (black in this case). The next two are the start and end points for the line. Finally, the width in pixels. The four lines of code above would draw two horizontal lines and two vertical lines.

Our finished `initBoard` function is shown below:

```

def initBoard(ttt):
    # initialize the board and return it
    # as a variable
    # -----
    # ttt : a properly initialized
    #      display variable

    background = pygame.Surface (ttt.get_size())
    background = background.convert()
    background.fill ((250,250,250))

```

```

# draw the grid lines
# vertical lines...
pygame.draw.line (background, (0,0,0), (100,0), (100,300), 2)
pygame.draw.line (background, (0,0,0), (200,0), (200,300), 2)

# horizontal lines...
pygame.draw.line (background, (0,0,0), (0,100), (300,100), 2)
pygame.draw.line (background, (0,0,0), (0,200), (300,200), 2)

# return the board
return background

```

Note that the last line of the function returns the background variable. Returning the variable allows us to use it elsewhere in the program. Now that we have our initialization function complete, we need to add code to call it and get the background variable. We'll add the following line above the main loop in the program:

```

# create the game board
board = initBoard (ttt)

```

4 Displaying the Board

At this point we have a function (`initBoard`) which draws the game board on a “surface” and returns it to the caller. However, if you ran this program, it wouldn't do anything for two reasons. First, we haven't actually called the function; we'll get to that. Second, `pyGame` does something called double-buffering with it's graphics. That is, when you make changes to a surface, they're not actually displayed until you explicitly put them on the screen. The reasoning behind this is that if you're doing complex animations, they'll appear smoother if the computer completes it's drawing operations before displaying the results. But that's not important right now. What is important is getting the results onto the screen. To accomplish this, we're going to write another function, `showBoard`. Our `showBoard` function takes two parameters: a variable representing the display, and a variable representing the surface we want drawn onto the screen. This function is very short and is presented below.

```

def showBoard (ttt, board):

    # (re)draw the game board (board) on the screen (ttt)
    ttt.blit (board, (0,0))
    pygame.display.flip()

```

Now that we have the code to display the board, we need to add it to the main event loop, so that the board will be (re)displayed after events are processed:

```

while (running==1):
    for event in pygame.event.get():
        if event.type is QUIT:
            running = 0
        # update the display
        showBoard (ttt, board)

```

5 Responding to Mouse Clicks

Tic-Tac-Toe isn't much of a game if the users can't click to make their moves. To respond to a mouse click, we're going to use a function that

- a. determines where the user clicked
- b. determines if that space has been used
- c. draws the X or O

To track what spaces have been used and whose turn it is we're going to declare two global variables. The first, XO, will track whose turn it currently is. The second, grid, will track which spaces have been used. We'll add their declarations, as shown below, before the pyGame initialization calls.

```

# X will go first...
XO = "X"

# declare an empty grid
grid = [ [ None, None, None ],
          [ None, None, None ],
          [ None, None, None ] ]

```

In declaring and initializing XO, we enclose the letter X in double quotes ("X") to indicate that it's a character, not another variable. When declaring the variable grid, we're using the None identifier. None is a special value in Python, which simply represents an empty variable space.

Next we need to add a handler to the main event loop. Modify the while loop as shown below:

```

while (running==1):
    for event in pygame.event.get():
        if event.type is QUIT:
            running = 0

    elif event.type is MOUSEBUTTONDOWN:
        clickBoard(board)

```

```
#update the display
showBoard (ttt, board)
```

As you can see, we're responding to the `MOUSEBUTTONDOWN` event (we don't distinguish between which button was clicked), and calling a function call `clickBoard`. We pass `clickBoard` a single parameter, the `pygame` surface we're drawing the board on.

`PyGame` provides a method to get the coordinates a user clicked at by using the following code:

```
(mouseX, mouseY) = pygame.mouse.get_pos()
```

This stores the `X` and `Y` coordinate of the mouse into two different variables: `mouseX` and `mouseY`. However, we want to deal with clicks in terms of which board space the user clicked in. We'll use a function to translate the mouse `X` and `Y` into the board's row and column. The `boardPos` function is shown below:

```
def boardPos (mouseX, mouseY):
    # determine the row the user clicked
    if (mouseY < 100):
        row = 0
    elif (mouseY < 200):
        row = 1
    else:
        row = 2

    # determine the column the user clicked
    if (mouseX < 100):
        row = 0
    elif (mouseX < 200):
        row = 1
    else:
        row = 2

    #return the row & column
    return (row, col)
```

Notice that in this case we're returning both the row and column in one step. Armed with the `boardPos` function and our knowledge of `pygame`, the `clickBoard` function is shown below.

```
def clickBoard (board):

    # determine where the user clicked on the board and draw the X or O
```

```

# tell Python that we want access to the global variables grid & X0
global grid, X0
(mouseX, mouseY) = pygame.mouse.get_pos()
(row, col) = boardPos (mouseX, mouseY)

# make sure this space isn't used
if ((grid[row][col] == "X") or (grid[row][col] == "O")):

    # this space is in use
    return

# draw an X or O
drawMove (board, row, col, X0)

# toggle X0 to the other player's move
if (X0 == "X"):
    X0 = "O"
else:
    X0 = "X"

```

Notice that when we check if the space is used, we used square brackets [] to access the contents of the array. We use two sets because this is a two-dimensional array. The drawMove function referenced here is covered below

6 Drawing X's & O's

Now that we can determine whose turn it is and where they clicked, we need to draw the X or O in the space. We're going to write a function called drawMove to handle this. drawMove takes four parameters. The first is a reference to the surface we want to draw on. The next two are the row and column we want to draw the move in. Note that the first row and column is numbered zero (0). Here is the drawMove function:

```

def drawMove (board, boardRow, boardCol, Piece):

    # draw an X or O (Piece) on the board at boardRow, boardCol

    # determine the center of the space
    # (this works because our spaces are 100 pixels wide and the first one
    # is numbered zero)
    centerX = boardCol * 100 + 50
    centerY = boardRow * 100 + 50

    # draw the piece
    if (Piece == "O"):

```

```

    # it's an 0; draw a circle
    pygame.draw.circle (board, (0,0,0), (centerX, centerY), 44, 2)
else:
    # it's an X
    pygame.draw.line (board, (0,0,0), (centerX - 22, centerY - 22), \
        (centerX + 22, centerY + 22), 2)
    pygame.draw.line (board, (0,0,0), (centerX + 22, centerY - 22), \
        (centerX - 22, centerY + 22), 2)

# mark the board space as used
grid[boardRow][boardCol] = Piece

```

First let's look at the `pygame.draw.circle` method. This method takes 5 parameters: the surface to draw on, the color to draw with (in RGB format), the center of the circle, the diameter, and the pen width. Note that the 44 pixel diameter was chosen rather arbitrarily as something that would fit in the square.

Something new that we haven't seen before is the backslash (`\`) at the end of the two `pygame.draw.line` lines. The backslash tells Python that the current line of code is continued onto the next line.

7 Determining The Winner

One of our final steps is to check if anyone has won the game. To accomplish this we're going to examine the `grid` variable that we've declared to hold the board status. It's relatively easy to check for a winner in Tic-Tac-Toe; simply see if any one row, any column or a diagonal has all the same value. For example, we can check for horizontal winners with the following code:

```

for row in range(0,3):

    if ((grid[row][0] == grid[row][1] == grid[row][2]) and \
        (grid[row][0] is not None)):
        # this row won
        winner = grid[row][0]
        pygame.draw.line(board, (250,0,0), (0, (row+1)*100 - 50), \
            (300, (row + 1) * 100 - 50), 2)
        break

```

The `range` function in the `for` loop (first line) tells python to repeat the indented block, starting with `row=0`, then `row=1`, and finally `row=2`. The last line, `break`, tells Python that once we've found a winner, there's no need to check the other

rows. We'll encapsulate the win-checking code into a function called `gameWon`. We're still going to pass in the board variable as the surface to draw the winning line on.

```
def gameWon(board):
    # determine if anyone has won the game
    # -----
    # board : the game board surface

    global grid, winner

    # check for winning rows
    for row in range (0, 3):
        if ((grid [row][0] == grid[row][1] == grid[row][2]) and \
            (grid [row][0] is not None)):
            # this row won
            winner = grid[row][0]
            pygame.draw.line (board, (250,0,0), (0, (row + 1)*100 - 50), \
                              (300, (row + 1)*100 - 50), 2)
            break

    # check for winning columns
    for col in range (0, 3):
        if (grid[0][col] == grid[1][col] == grid[2][col]) and \
            (grid[0][col] is not None):
            # this column won
            winner = grid[0][col]
            pygame.draw.line (board, (250,0,0), ((col + 1)* 100 - 50, 0), \
                              ((col + 1)* 100 - 50, 300), 2)
            break

    # check for diagonal winners
    if (grid[0][0] == grid[1][1] == grid[2][2]) and \
        (grid[0][0] is not None):
        # game won diagonally left to right
        winner = grid[0][0]
        pygame.draw.line (board, (250,0,0), (50, 50), (250, 250), 2)

    if (grid[0][2] == grid[1][1] == grid[2][0]) and \
        (grid[0][2] is not None):
        # game won diagonally right to left
        winner = grid[0][2]
        pygame.draw.line (board, (250,0,0), (250, 50), (50, 250), 2)
```

Note that we refer to the global variable `winner` in our function. This variable will store the winner (if any). Before we can use that variable, we need to

initialize it. In Python, initializing a variable consists of assigning a value to it. We're going to initialize `winner` by adding the following statement at the top of our program where we already initialized `grid` and `XO`.

```
winner = None
```

Finally, we want to check for a winner after every move, so we'll add a call to `gameWon` to our main loop as well.

```
while (running == 1):
    for event in pygame.event.get():
        if event.type is QUIT:
            running = 0
        elif event.type is MOUSEBUTTONDOWN:
            # the user clicked; place an X or O
            clickBoard(board)

        # check for a winner
        gameWon (board)

        # update the display
        showBoard (ttt, board)
```

8 Displaying the Game Status

At this point we have a playable game, but we can add a final piece of code to make it a little more "user-friendly". In this step we're going to add code that draws the game "status" below the board. The status is text such as "x's turn" or "O won!" We have two global variables, `XO` and `winner`, which hold the current turn and the winner (if any) respectively. We'll begin our function as follows:

```
def drawStatus (board):

    # draw the status (i.e., player turn, etc) at the bottom of the board
    # -----
    # board : the initialized game board surface

    # gain access to global variables
    global XO, winner
```

Now that we have declared our function and have access to the global variables, we need to figure out what status message we're going to draw. There are two possible types of messages: the "you won" message and the "someone's turn" message. We'll determine the appropriate message with an `if` statement, and store the message in the variable `message`.

```

# determine the status message
if (winner is None):
    message = X0 + "'s turn"
else:
    message = winner + " won!"

```

Now that we have the status message we need to display it on the game board. PyGame has three steps for displaying text on a surface. First, get the font you're working with. Second, render the text onto a new surface. Finally, copy (blit) that surface onto your destination surface. We'll accomplish that with the following code.

```

# render the status message
font = pygame.font.Font(None, 24)
text = font.render(message, 1, (0,0,0))

# copy the rendered message onto the board
board.fill ((250, 250, 250), (0, 300, 300, 25))
board.blit (text, (10, 300))

```

The first line gets the current font, specifying we want a height of 24 pixels. The second line calls the text render function. The render function takes three parameters: the text to render (message), whether we want the text anti-aliased (smoothed; 1 = true), and an RGB foreground color. We call board.fill next in order to clear any text we've previously displayed. Finally, we call the blit function which copies one surface to another.

The last thing we have to do is add a call to the drawStatus function. We want to update the status each time we draw the board, so we'll add the following line to the beginning of showBoard:

```
drawStatus(board)
```

Our finished showBoard looks like this:

```

def showBoard (ttt, board):
# (re)draw the game board (board) on the screen (ttt)

    drawStatus(board)
    ttt.blit (board, (0,0))
    pygame.display.flip()

```

9 Conclusion

We now have a working tic-tac-toe game written in Python with the pygame library. The source code is available on the F drive under Courses, pygame.

References

- [1] pyGame homepage: <http://www.pygame.org>
- [2] pyGame Documentation: <http://www.pygame.org/docs>
- [3] Python Language Reference: <http://www.python.org/doc/current/ref>
- [4] Python Standard Library Reference: <http://www.python.org/doc/lib>
- [5] The PyGame Code Repository: <http://www.pygame.org/pcr>
- [6] O'Reilly Network on PyGame: <http://www.onlamp.com/pub/a/python/2001/12/20/pygame.html>