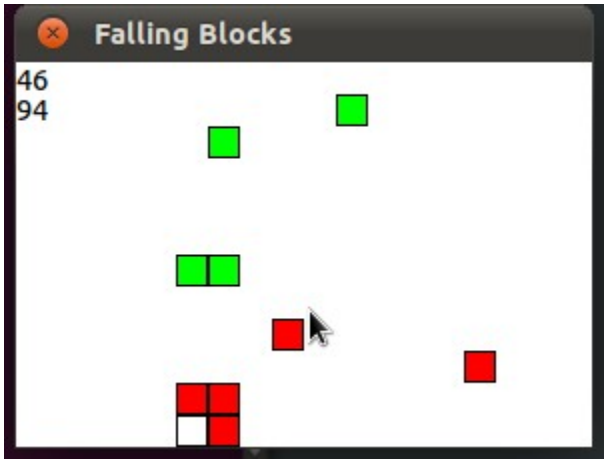
Game development with wxPython



Index

Introduction

First of all, if you are reading this, I assume you have prior knowledge of Python and wxPython. Another assumption is that you are using Python 2.6.x or Python 2.7.x and wxPython 2.8.x. Neither prior nor posterior versions of wxPython are recommended for now. Notice, though, that this was written in 2011 and in case wxPython 2.9 is already stable, using it might cause no problems.

Since Python and wxPython are both cross-platform, this is, they work in more than one operating systems, the guide is aimed for any operating systems.

I once tried pyGame and I was not very comfortable with it. I had been using wxPython for Graphical User Interfaces for a long time and I wanted to be able to develop games with it. After some research, I discovered the Graphics Device Widget, and in order to make my first game I had to search for information on several websites. I decided to make a tutorial to help beginners get comfortable with wxPython in order to develop games.

I must also note that many Python programmers will recommend pyGame because it was made especially for games. I don't disagree, but I think you should try both and then decide which one to use.

Regarding coding styles, I will always be using PEP-8, here are some of the key guidelines:

- Number of columns per line should be higher than 80;
- 4 spaces for indentation (never use TABs);
- All classes and functions should have a docstring below explaining them;
- Variables and Functions names are of type “my_variable” and classes are of type “MyClass”

There are other guidelines that you can find on Python's website:
http://www.python.org/dev/peps/pep-0008/

Why wxPython?
Most of the times, when one wants to develop a game using Python, he chooses to use pyGame. It is the most used library for game development in Python because it was made from the start to be used in games and that is only purpose.

wxPython has as main purpose providing programmers a simple but powerful way of developing Graphical User Interfaces (from now on GUIs). The vast majority of games are made in GUIs because terminals don't have graphical capabilities.

However, wxPython has support for many widgets, one of them, the Graphics Device (explained more detailedly later) allows the programmer to simply generate colored 2D Graphics. We will be using this in order to make games.

The Graphics Device Widget
wxPython has many built-in widgets such as buttons, text controls, menus, and so on, but in order to develop games that use graphics, we will focus on the Graphics Device Widget.

The Graphics Device allows us to develop simple 2D graphics and we can use it for animations, simulations or games. The Graphics Device has several components, we will be using wxPaintDC:

http://www.wxpython.org/docs/api/wx.PaintDC-class.html

Creating a wxPaintDC widget is very simple, here's how it's done (self is a wxPanel here):

```
self.dc = wx.PaintDC(self) #Creates the wxPaintDC
self.dc.Clear() #Clears (to white) the wxPaintDC
```

Now, as I already said, self is a wxPanel here, so here is a class that inherits wxPanel, adds a wxPaintDC Component to it and draws a line.

```
class Panel(wx.Panel):
    """Class with the main panel and its functions"""
    def __init__(self, parent):
        """Defines bindings and sets the panel up"""
        super(Panel, self).__init__(parent)  #super the panel
        self.Bind(wx.EVT_PAINT, self.OnPaint)

    def OnPaint(self, event):
        """Is called with EVT_PAINT, draws a line"""
        self.dc = wx.PaintDC(self)  #Creates the wxPaintDC
        self.dc.Clear()  #Clears (to white) the wxPaintDC
        self.dc.DrawLine(5, 5, 100, 5)  #Draws a line from (5,5) to (100,5)
```

Now, this code by itself doesn't do anything, we need to put that panel in a frame and display the frame. But first I'll explain this to you.

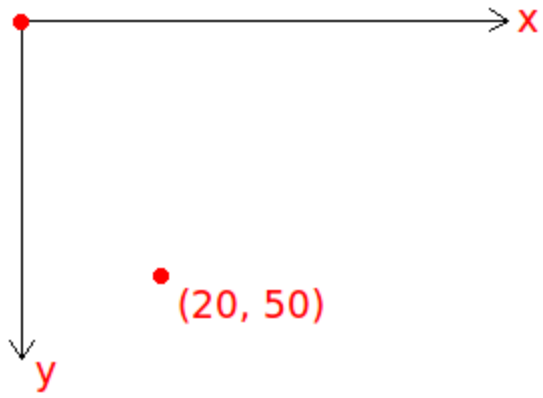
We created a class that inherits wx.Panel, and on the __init__ we set it up by supering it to it's parent (which will be a wxFrame). Then we bind the event wx.EVT_PAINT , which is called when the frame is opened to the function OnPaint.

So whatever is inside OnPaint will be done after the frame is displayed, meaning the line we drew on it will immediately appear on the screen.

self.dc.DrawLine(x1, y1, x2, y2)

wxPaintDC.DrawLine() is a function that draws a line from the coordinates (x1, y1) to the coordinates (x2, y2).

Everything inside the wxPaintDC is done with coordinates:

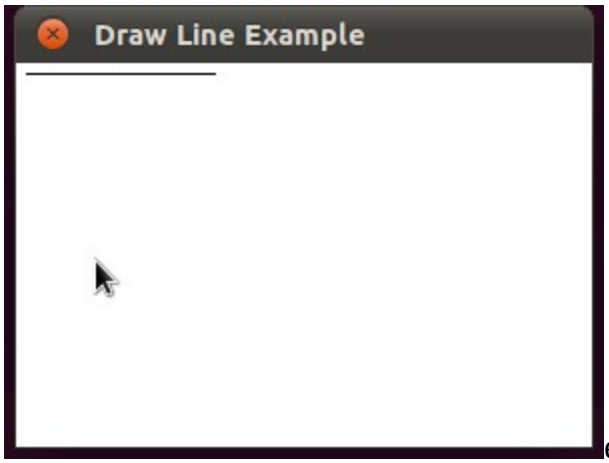


In this quick sketch I made it's easy to understand how coordinates work on wxPaintDC.

Now in the following sample I'll add the panel to a frame (the code is in the file DrawLine.py in code samples folder).

```
import wx

class MainFrame(wx.Frame):
    """Class with the GUI and GUI functions"""
    def __init__(self, parent,id):
        """Displays the frame, creates the GUI"""
        wx.Frame.__init__(self, parent,id, "Draw Line Example",
                          size=(288,192), style=wx.CAPTION | wx.CLOSE_BOX)
        sizer= wx.BoxSizer(wx.VERTICAL)
        self.SetSizer(sizer)
        panel= Panel(self)
        sizer.Add(panel, 1, wx.EXPAND)
        self.Layout()

class Panel(wx.Panel):
    """Class with the main panel and its functions"""
    def __init__(self, parent):
        """Defines bindings and sets the panel up"""
        super(Panel, self).__init__(parent)  #super the panel
        self.Bind(wx.EVT_PAINT, self.OnPaint)

    def OnPaint(self, event):
        """Is called with EVT_PAINT, draws a line"""
        self.dc = wx.PaintDC(self)  #Creates the wxPaintDC
        self.dc.Clear()  #Clears (to white) the wxPaintDC
        self.dc.DrawLine(5, 5, 100, 5)  #Draws a line from (5,5) to (100,5)

if __name__=='__main__':
    """Displays the GUI and starts the GUI Loop"""
    app = wx.PySimpleApp()
    frame = MainFrame(parent=None, id=-1)
    frame.Show()
    app.MainLoop()
```

Here's the result of the above code on Ubuntu 11.04:



The line from coordinates (5, 5) to (100, 5) is easily noticeable and the code is quite simple.

You can also notice that the frame cannot be maximized, minimized nor resized because I set it to be like that in the code:

```
style=wx.CAPTION | wx.CLOSE_BOX
```

Timers and Games

In most games we have to keep redrawing the screen so that is always updated. The faster it redraws itself, the better, but this also consumes more CPU. So if you think about most modern games you don't even notice that the frame is actually being redrawn, unless if you have high latency (online games) or a slow computer. However, in perfect conditions it has to look natural, just like in real life.

In order to do this we need to have a function that is called many times per second and that draws everything on the screen. wxPython provides us an object that is very useful for this, wxTimer.

Since I think it's easier to learn from code than from text here's an example of a line that will go left until it hits the border and when it hits the border goes left.

The following code might be harder to understand than the previous one so look over it line by line and read the comments. The file is called “MovingLine.py”.

```
import wx

class MainFrame(wx.Frame):
    """Class with the GUI and GUI functions"""
    def __init__(self, parent,id):
        """Displays the frame, creates the GUI"""
        wx.Frame.__init__(self, parent,id, "Draw Line Example",
                          size=(300,200), style=wx.CAPTION | wx.CLOSE_BOX)
        sizer= wx.BoxSizer(wx.VERTICAL)
        self.SetSizer(sizer)
        panel= Panel(self)
        sizer.Add(panel, 1, wx.EXPAND)
        self.Layout()

class Panel(wx.Panel):
    """Class with the main panel and its functions"""
    line_x = 1 #X position of line
    going_left = False
    going_right = True #The line goes to the right first

    def __init__(self, parent):
        """Defines bindings and sets the panel up"""
        super(Panel, self).__init__(parent)  #super the panel
        self.Bind(wx.EVT_PAINT, self.OnPaint)

        #Create the timer
        self.timer = wx.Timer(self)
        self.Bind(wx.EVT_TIMER, self.Update, self.timer)
        self.timer.Start(50) #Updates Every 50 milliseconds

    def Update(self, event):
        """Update function, always being called"""
        self.dc.Clear() #Clear dc
        self.dc.DrawLine(self.line_x, 0, self.line_x, 200) #Draw line
```

```
        #Check if it reached the parent's size width
        #Or if it hit the left border
        if self.line_x == self.GetParent().GetSize()[0] or self.line_x==0:
            #Reverse directions
            self.going_left = not self.going_left
            self.going_right = not self.going_right
        if self.going_left:
            self.line_x -= 1 #Go to the left
        else:
            self.line_x += 1 #Go to the right

    def OnPaint(self, event):
        """Is called with EVT_PAINT, draws a line"""
        self.dc = wx.PaintDC(self)  #Creates the wxPaintDC
        self.dc.Clear()  #Clears (to white) the wxPaintDC

if __name__=='__main__':
    """Displays the GUI and starts the GUI Loop"""
    app = wx.PySimpleApp()
    frame = MainFrame(parent=None, id=-1)
    frame.Show()
    app.MainLoop()
```

I recommend that you use the file “MovingLine.py” instead of copying from this file.

The line will move right until it hits the border and when it does it moves left until it hits the border). There is a timer associated with the function Update that makes it possible to redraw the frame every 50 milliseconds to make it look natural (you can change this number).

“Falling Blocks” - An example

My first wxPython game was “Falling Blocks”, a very simple game where you use the arrow keys to avoid being hit from blocks falling from the sky. It features highscores and colours. The code is not very well-commented but I think it is fairly understandable. From now on, feel free to explore the wxPaintDC and wxPython in order to develop more games or if you didn't like wxPython, feel free to use pyGame or another language for game development.

```
import wx
import random
import time

class MainFrame(wx.Frame):
    """Class with the GUI and GUI functions"""
    def __init__(self, parent,id):
        """Displays the frame, creates the GUI"""
        wx.Frame.__init__(self, parent,id, "Falling Blocks",
size=(288,192),
                          style=wx.CAPTION | wx.CLOSE_BOX)
        sizer= wx.BoxSizer(wx.VERTICAL)
        self.SetSizer(sizer)
        panel= Panel(self)
        sizer.Add(panel, 1, wx.EXPAND)
        self.Layout()

class Panel(wx.Panel):
    """Class with the main panel and its functions"""

    def initVariables(self):
        """Starts the variables at the title screen"""
        self.x = 128       #Player starts in the middle
        self.score = 0
        self.blocks = []

        #Following variables tell us at what screen the user is
        self.onTitleScreen = True    #Game starts at title screen
        self.onGame = False
        self.onGameOver = False

        #Get Highscore
        try:
            f = open("data", "r")
            self.highScore = int(f.read()[2:], 2)
            f.close()
        except IOError: #In case file doesn't exist
            f = open("data", "w")
            f.write("0b00")
            f.close()
            self.highScore = 0

    #Initialize Constants
    y = 176
```

```
    #The game works on a grid, so there is a limited number of positions
    possibleLocations = [0, 16, 32, 48, 64, 80, 96, 112, 128, 144, 160,
176,
                         192, 208, 224, 240, 256, 272]

    def __init__(self, parent):
        super(Panel, self).__init__(parent)
        self.Bind(wx.EVT_KEY_DOWN, self.OnKeyDown)
        self.Bind(wx.EVT_PAINT, self.OnPaint)

        self.initVariables()

        #Set the timer
        self.timer = wx.Timer(self)
        self.Bind(wx.EVT_TIMER, self.Update, self.timer)
        self.timer.Start(50)

    def Update(self, event):
        if self.onGame:
            if random.randint(0,3) == 1:
                #Add random blocks in random location
                self.blocks.append([random.choice(self.possibleLocations),
0])
            if self.score>20: #Keep the beginning of the game fairly easy.
                if random.randint(0,3) == 1:
                    self.blocks.append([self.x, 0]) #Add new blocks above
user
            for i in self.blocks:
                if i[0] == self.x  and (i[1] == 160 or i[1] == 176):
                    self.onGame = False
                    self.onGameOver = True
                    break
                i[1] = i[1] + 16 #Move blocks down

                #Remove blocks below border from the list to keep game
fast.
                if i[1] == 192:
                    self.blocks.pop(self.blocks.index(i))

            #Update Score and HighScore
            self.score+=1
            if self.score>self.highScore:
                self.highScore = self.score
        if self.onTitleScreen:
            #To keep the text clean
            self.dc.Clear()

        self.Paint()

    def OnKeyDown(self, event):
        """Handles key presses"""
        keyCode = event.GetKeyCode()
        if self.onGame:
            #Go to the right
```

```
            if keyCode == wx.WXK_RIGHT and self.x != 272:
                self.x = self.x + 16
            #Go to the left
            elif keyCode == wx.WXK_LEFT and self.x != 0:
                self.x = self.x - 16
        elif self.onTitleScreen:
            #Start the game
            self.onTitleScreen = False
            self.onGame = True
            time.sleep(0.5)
        else:
            #If on gameOver screen wait a bit and go to title screen again
            time.sleep(2)
            self.initVariables()
        if keyCode == wx.WXK_ESCAPE:  #Close the game
            self.Destroy()
        self.Paint()  #Repaint
        event.Skip()

    def Paint(self):
        """Redraws the game"""
        if self.onGame:
            self.dc.Clear()
            self.dc.SetFont(wx.NORMAL_FONT)
            self.dc.DrawText(str(self.score), 0, 0)
            self.dc.DrawText(str(self.highScore), 0, 15)

            self.dc.SetBrush(wx.WHITE_BRUSH)
            self.dc.DrawRectangle(self.x, self.y, 16, 16)

            for i in self.blocks:
                if i[1] >= 128:
                    self.dc.SetBrush(wx.RED_BRUSH)
                    self.dc.DrawRectangle(i[0], i[1], 16, 16)
                else:
                    self.dc.SetBrush(wx.GREEN_BRUSH)
                    self.dc.DrawRectangle(i[0], i[1], 16, 16)
        elif self.onTitleScreen:
            self.dc.SetFont(wx.Font(14, wx.SWISS, wx.NORMAL, wx.BOLD))
            self.dc.DrawText("Falling Blocks", 65, 30)
        else:
            self.dc.Clear()
            self.dc.SetFont(wx.Font(14, wx.SWISS, wx.NORMAL, wx.BOLD))
            self.dc.DrawText("GAME OVER", 2, 2)

            self.dc.SetFont(wx.NORMAL_FONT)
            self.dc.DrawText("%s %d" % ("Score: ", self.score), 2, 25)
            self.dc.DrawText("%s %d" % ("Highscore: ", self.highScore), 2,
40)

            #Save the highscore
            f = open("data", "w")
            f.write(bin(self.highScore))
            f.close()
```

```
    def OnPaint(self, event):
        """First called with EVT_PAINT"""
        self.dc = wx.PaintDC(self)
        self.dc.Clear()
        self.Paint()

if __name__=='__main__':
    """Displays the GUI and starts the GUI Loop"""
    app = wx.PySimpleApp()
    frame = MainFrame(parent=None, id=-1)
    frame.Show()
    app.MainLoop()
```

Conclusion

Now that you have learned the basics of developing games with wxPython, feel free to start coding more complex things. Check the wxPython.org's website for more information on all the widgets and reread this if you forgot about the basics.

Contact

If you have any comments to make on the tutorial, any ideas, or you find any bug you can contact me on IRC:

ephan on irc.freenode.net