



Contents

- [1 Introduction](#)
- [2 Features](#)
- [3 Usage](#)
- [4 Source code](#)
- [5 Screenshots](#)
- [6 Related Links](#)

Introduction

This article demonstrates generation of graphics for S60 devices. The simplest mobile application development tool, PYS60, is used to develop this GraphicsGenerator.

Basically, the code snippet given in this article generates beautiful patterns (slide strokes for now) of the desired colour and latency (a parameter to interpret the pattern).

Features

- Generates slide stroke patterns.
- Supports all colours.
- Support for random colours.
- Supports 30 different patterns - all slide strokes.
- Graphics generated can be saved.

Usage

The following modules are used when developing this application:

- appuifw module
- os module
- graphics module
- e32 module
- key_codes module

How_to_generate_graphics_in_Python

- random module
- math module
- time module

The following functions are used to create GraphicsGenerator:

- save(): save the screenshots of Graphicsgenerator
- draw_screen(): draw the screen on canvas
- Start(): create a different pattern on canvas

These all functions are defined in source code in detail.

Source code

The code for the GraphicsGenerator is below.

```
#
# Graphics Generator
#
'''
1.00 2009-07-05 Initial release
'''

VERSION = "1.00"

# importing the necessary modules
import e32,random
import time
import appuifw, os
import graphics, time
import math

# define color code
BLACK=(0,0,0)
RED=(255,0,0)
GREEN=(0,255,0)
BLUE=(0,0,255)
WHITE=(255,255,255)
running=1

#initializing pattern
pattern=30

# Define the exit function
app_lock = e32.Ao_lock()
def quit():
global running
    running=0
    appuifw.app.quit()
appuifw.app.exit_key_handler = quit

# Defining the save function for Screenshots
```

Usage

How_to_generate_graphics_in_Python

```
def save():
    dirpath=GraphicsGenerator"
try:
os.mkdir(dirpath)
except: pass

# Sleep for 1 second
time.sleep(1)

# This function converts the current view to the new images
image=graphics.snapshot()

#current time is return by this function
time.strftime("%H:%M:%S")

# This is the path and name for storing the screenshots
save_dirpath+"\\screenshot.jpg")
appuifw.Saved to "+ unicode(dirpath))

#Define a function that will be called when the canvas needs to be redrawn
def handle_redraw(rect):
    canvas.blit(img)

#define the draw function
def draw_screen():

    #Set the screen to full
    appuifw.app.screen = 'full'

    #Global variable for application UI
    global canvas

    #Create an instance of Canvas and set it as the application's body

    canvas = appuifw.Canvas(redraw_callback=handle_redraw)
    appuifw.app.body = canvas
    appuifw.app.exit_key_handler = quit

    #Global variable for application UI
    global img
    img = graphics.Image.new(canvas.size)

    #Clear the image
    img.clear(BLACK)
    handle_redraw(None)

def menu_about():
    ''' Callback for menu item About '''

    appuifw.Graphics Generator version 1.0"+"\\n Developed by Nirpsis")

#defining start function

def start():
    global running, pattern
    running=
    UserWantRandom=

# Define x & y coordinate
120 x=
160 y=
1ace1X=-
```

How_to_generate_graphics_in_Python

```
lacyY=-
    app.mainloop(Choose the colour of the pattern")
L=[u"Red", u"Blue", u"Green", u"White", u"Random",u"Manual Colour"]

#Selectionlist allows the user to select the list items
    i = app.selection_list(L, search_field=1)
if (i==0): COLOR=RED
if (i==1): COLOR=BLUE
if (i==2): COLOR=GREEN
if (i==3): COLOR=WHITE
if (i==4): UserWantRandom=1
if (i==5):
    query(a="Enter Red concentration 0-255", "number")
if a>255 or a<0:
    255
    a=
    query(b="Enter Green concentration 0-255", "number")
if b>255 or b<0:
    255
    b=
    query(c="Enter Blue concentration 0-255", "number")
if c>255 or c<0:
    c=255
    (a,b,c) COLOR=
else:
    () quit

    pattern=app.prompt("Enter patter 1-30", "number")
if pattern>30 or pattern<1:
    30 pattern=

# Calling the draw_screen function
    draw_screen
    running=

# Main loop
while running:
if x<0:
if acelX==1:
                                acelX=-1
else:
                                acelX=1
if y<0:
if acely==1:
                                acely=-1
else:
                                acely=1
if x>240:
if acelX==1:
                                acelX=-1
else:
                                acelX=1
if y>320:
if acely==1:
                                acely=-1
else:
                                acely=1
if(y <= 0):
    0; y =
if(y >= 320):
    320; y =
if(x <= 0):
```

How_to_generate_graphics_in_Python

```
0;                x =
if(x >= 240):
    240;           x =

    #defining the position of the points

    * acelX      x = x
    * acelY      y=y
                x=x+pattern
                y=y+pattern
if UserWantRandom==1:

    #below script draws random points
#defining the colours of points
random.randint( 0, 255)
random.randint( 0, 255)
random.randint( 0, 255)
    (a,b,c)      COLOR=

    # Drawing points
    point((x,y),color=COLOR,width=15)

# Sleep for 0.01
    ao_sleep(0.01)
    ao_yield()

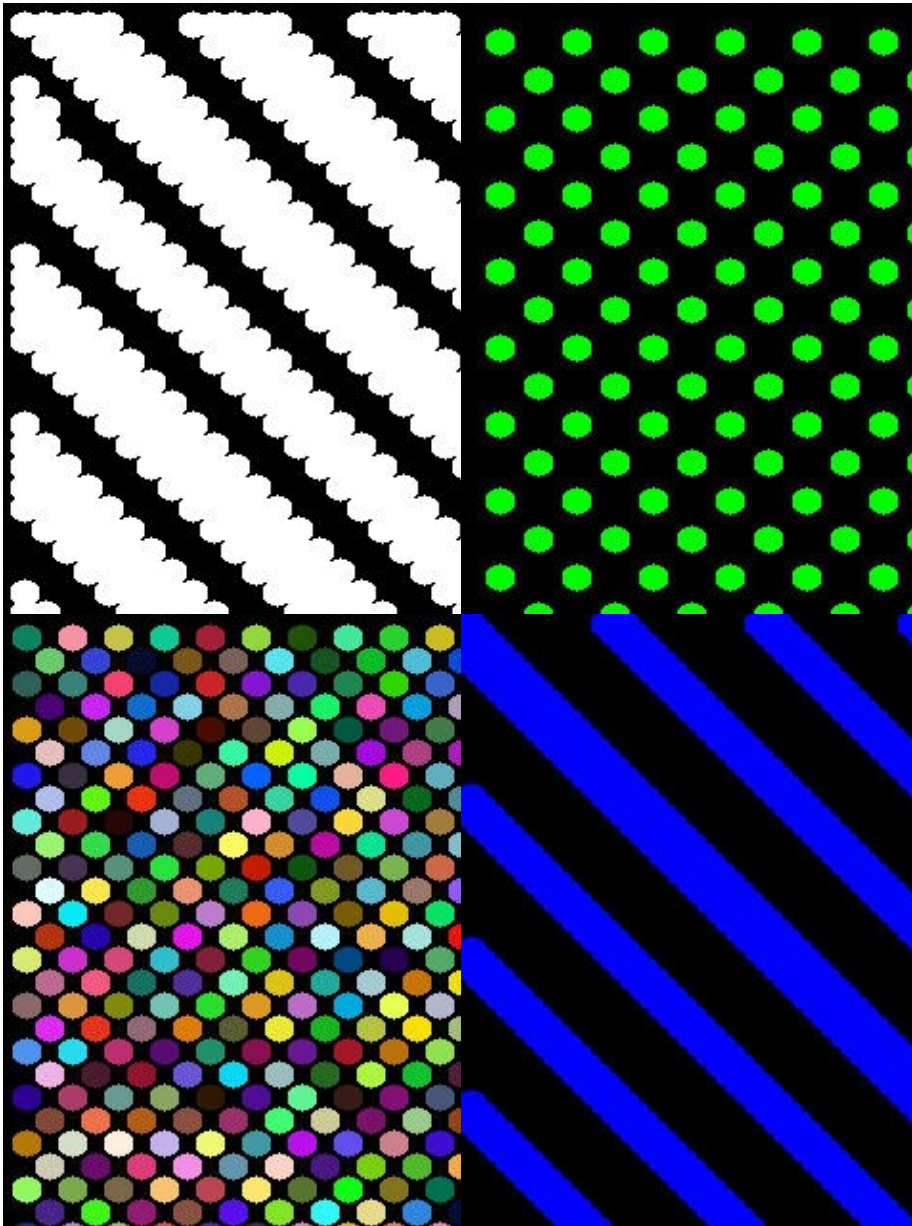
# Main menu
    app.menuapp.add_menu_item("Save Graphics", save), (u"Start again", start), (u"About", menu_about), (u"
    (None), handle_redraw

start()
app_lock = e32.Ao_lock()

#Wait for the user to request the exit
app_lock.wait()
```

Screenshots

Below are few of the graphics generated by GraphicsGenerator application, demonstrating the different pattern generation on canvas.



Related Links

- <http://nirpsis.blogspot.com/>
- <http://sites.google.com/site/nirpsis/graphicsgenerator>