11 Building a GUI

Introduction

The title of this worksheet is a little ambitious — we are not going build a full Graphical User Interface (GUI) but just a main menu with some (hopefully cool) features. However, what we are going to do, is to develop everything ourselves (using turtle graphics) rather than using one of the many GUI libraries available in python.

You could ask *"Why?"* as in *"Why would we build our own menu system? Surely it is easier to use an existing GUI library?"*

The answer is: Yes, it would be easier, but we want to build our own because;

- We can.
- We will use this to learn how GUI respond to events (mouse click, keyboard press/release, etc.).
- We will be able to add our own special effects (more later) to our GUI.

Figure 1 – Completed main menu.

As usual we build this using a sequence of steps — testing our code along the way.

① **Creating the Menu Screen**

We have done this many time before. All we need do is import the `turtle` module and create a `Screen` where we place our drawing, and a `Turtle` or two to do the actual drawing. We do one thing that is new — we insert a background image (must be in `gif` format).

② **Creating Buttons that Work**

Here we need to do some fancy coding — both in drawing our buttons and figuring out how to respond to a button click.

③ **Building the Menu**

Finally, here you can design your menu screen anyway you want — don't be stuck with my boring, everything is in a gird layout.

④ **Updating our Earlier Programs so they Behave with Our Menu — TODO**

We will need to modify some of our earlier programs (Graphical Take Away, Monkey Wars, etc) so they work better when we run them from our menu.

This is a small task that we will leave until next week. Also we have yet to add the cool features!

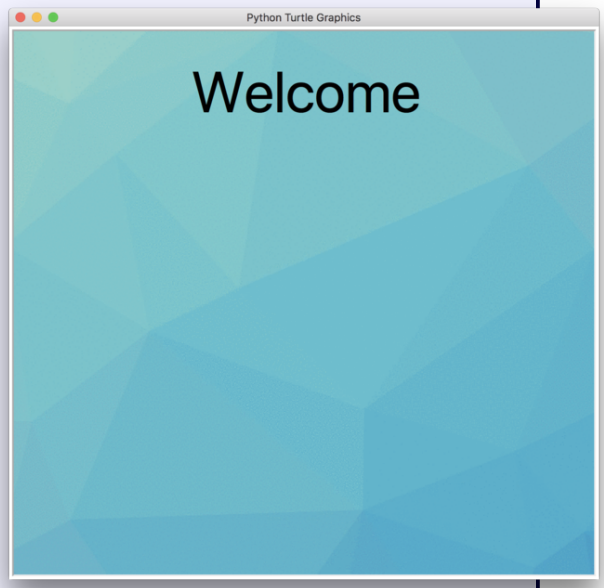
CoderDojo, Tramore, Waterford. (`kmurphy@wit.ie`)

1 Creating the Menu Screen

Create file called `Menu.py` and insert the following code.

Menu.py

```python
# 1 - Import needed modules


import turtle
import os, sys


# 2 - Create screen


background = "background_1.gif"

screen = turtle.Screen()
screen.addshape(background)
turtle.shape(background)

bob = turtle.Turtle()
bob.hideturtle()
bob.speed("fastest")


# 3 - Define helper functions


def jump(t, x, y):
    t.penup()
    t.goto(x,y)
    t.pendown()
    t.setheading(0)


# 4 - Build sceen


jump(bob,0,220)
bob.write("Welcome", align="center",
    font=("Comic Sans", 70, "normal"))

turtle.mainloop()
```



Our code will get long so we need to keep things organised. Here the code is divided in to four sections — import modules, create the empty screen and set background, define helper functions, and finally build the screen.
Currently this code just creates the window with the "Welcome" message. Next we will add buttons and mouse click events.

2 Creating Buttons that Work

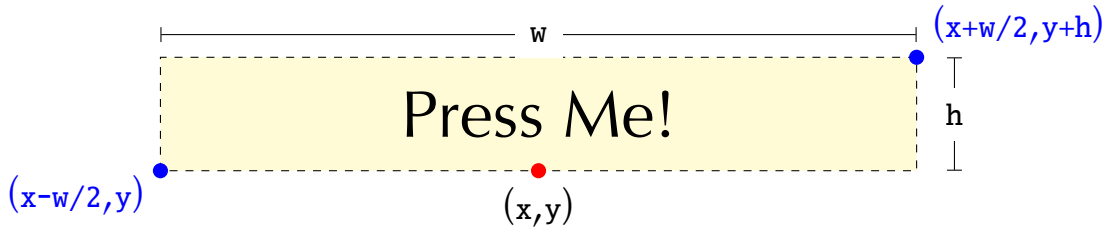


Figure 2 – Button position and dimensions.

2.1 Developing a function to build a Button

In order to create a button that we can click we need to decide on its:

- Position[1] on the screen, (`x,y`)

  We want the button text to be centred and draw the rectangle around the text so we will position a button based on the centre of the bottom edge (see Figure 3).

- Width, (`w`), and height, (`h`).

  We need to draw a box big enough so that it looks like the text is inside this. If we used a GUI library this would be done automatically but we will do this manually and just pick numbers for now.

- Button label, (`label`)

  This will be the text that the user sees and also the name of the button when we are responding to events.

- Font, (`font`)

  We might want the more important buttons to use a larger font so we need to be able to set this.

Since we are planning to create multiple buttons we should write a function that we can call for every button.

At the bottom of the "`Define helper functions`" section of your code insert the following function.

Menu.py

```
def drawButton(x=0 ,y=0, w=150,h=50, label="My Button",
        font=("Comic Sans", 40, "normal")):

    print ("Button '%s' Not done yet" % label)

    # draw label
    # draw border
    # save button information
```

[1] Remember how we navigate across the screen

- The centre, also called the **origin**, and is denoted by $(0,0)$.
- Every point on the screen is defined by two numbers, (x,y), where x = how far to the right of the origin and y = how far above the origin.

In the above code, we have given default values for every parameter — this will simplify our code later.

After the "Welcome" message code in the "`Build sceen`" section, try to create a few buttons using the following code:

Menu.py

```
drawButton(y=100, label="Press Me")
drawButton(y=0, label="No Press Me")
drawButton(y=-100, label="Forget them, PRESS ME")
```

Run your code

You should see nothing on the screen, but see three messages in Thonny saying

```
Button `Press Me` Not done yet
Button `No Press Me` Not done yet
Button `Forget them, PRESS ME` Not done yet
```

Now we are going to implement our button function …

2.1.1 Draw the button label

After the "`draw label`" comment insert code that jumps `bob` to correct position and writes the button label. Run this and you should see the labels.

Menu.py

```
    # draw label
    jump(bob, x,y)
    bob.write(label, align="center", font=font)
```

This looks good. Lets now add a border …

2.1.2 Draw the border

After the "`draw border`" comment insert code that draws a rectangle of width `w`, and height `h`

Menu.py

```
    # draw border
    for k in range(2):
        bob.forward(w)
        bob.left(90)
        bob.forward(h)
        bob.left(90)
```

I see two problems here. The width of the rectangles is too small — this is easy to fix. We just set the `w` parameter when calling `drawButton`. The second problem is that the rectangle is drawn starting from the bottom centre of the label — we fix this by moving `bob` to the bottom left corner (see Figure 3).

Insert code to move `bob` to bottom left corner, and change the widths of the button in your `drawButton` calls by setting parameter `w`. You should then get the following.

Menu.py

```
    # draw border
    jump(bob, x-w/2,y)
    for k in range(2):
        bob.forward(w)
        bob.left(90)
        bob.forward(h)
        bob.left(90)
```

Now the above works, but putting basic drawing of a rectangle inside our draw button is not a great idea. We will be drawing lots of rectangles so lets move that code out into a new helper function.

Before your `drawButton` function in `Define helper functions` section, insert the following function

Menu.py

```
def drawRectangle(x=0, y=0, w=150, h=50, color="black", fill=None):

    jump(bob, x,y)

    bob.color(color)
    if fill:
        bob.fillcolor(fill)
        bob.begin_fill()

    for k in range(2):
        bob.forward(w)
        bob.left(90)
        bob.forward(h)
        bob.left(90)

    if fill: bob.end_fill()
```

This function will draw a rectangle of width `w`, and height `h`, with bottom left corner at position (`x`,`y`). You can also specify the border colour and the fill colour.

You should think about extending this function by, for example,

- Adding parameter `pensize=2` which would set the thickness of the border.
- Adding parameter `shadow=False` which when set to `True` would draw a shadow.

  Drawing a shadow, is actual easy — we just draw a few rectangles a little to the right an a little below the position of the main rectangle BEFORE we draw the main rectangle. If you are exceedingly lazy (this is a good thing in a programmer!) you can draw the shadow rectangles by just calling the `drawRectangle` function from inside the `drawRectangle`

Now your `drawButton` can be simplified to

Menu.py

```
def drawButton(x=0 ,y=0, w=150,h=50, label="My Button",
        font=("Comic Sans", 40, "normal")):

    # draw label
    jump(bob, x,y)
    bob.write(label, align="center", font=font)

    # draw border
    drawRectangle(x-w/2,y, w,h, fill="yellow")

    # save button information
```

If you look, really really carefully at the output you will see that *we* have done something silly. To fix this, we need to draw the button label after drawing the border. Fix This.

Menu.py

```
def drawButton(x=0 ,y=0, w=150,h=50, label="My Button",
        font=("Comic Sans", 40, "normal")):

    # draw border
    drawRectangle(x-w/2,y, w,h, fill="yellow")

    # draw label
    jump(bob, x,y)
    bob.write(label, align="center", font=font)

    # save button information
```

2.1.3 Making the buttons work

OK, now we have things that looks like buttons, but they do not act like them — click on a button and see what happens — nothing.
To get the buttons to work we need to do two things:

- Listen for and respond to click events.

  This is easy — we will create our own `onclick` function to run whenever the user clicks on the screen.

- When a click occurs, decide whether it happened within a button region and which button.

  This is a little harder so we will talk about this in some detail.

Insert the following function at the bottom of the "`Define helper functions`" section of your code.

Menu.py

```
def onclick(x,y):

    print ("Mouse click at (%s,%s)" % (x,y))
```

and just before the `turtle.mainloop()` insert the lines

Menu.py

```
screen.listen()
turtle.mainloop()
```

Now run your code and make sure you see a message appearing in *Thonny* every time you click the left mouse button. You should see that you also get the position of the mouse in the screen when the click occurred. Next we need to use this position information to determine which button was "pressed".

Insert the code at the bottom of the "`Create screen`" section of your code.

Menu.py

```
21 from collections import namedtuple
22 Button = namedtuple('Button', 'x y w h label')
23 bob.buttons = []
```

- Lines 21 and 22, allow us to create an object (like the `Turtle` object or the `Screen` object) which stores information for a Button. Our `Button` object stores position (`x`,`y`), size `w` by `h`, and `label`.
- Line 23 defines an empty list, called `buttons`, that will store the information for all generated buttons.

So where do we get the information that we want to store in the `buttons` list?
We have this information at end of `drawButton` function so that is where we will build our `Button` object.

Modify the `drawButton` function so that it returns a `Button` object, as shown below.

Menu.py

```
def drawButton(x=0 ,y=0, w=150,h=50, label="My Button",
        font=("Comic Sans", 40, "normal"), fill="yellow"):

    # draw border
    drawRectangle(x-w/2,y, w,h, fill=fill)

    # draw label
    jump(bob, x,y)
    bob.write(label, align="center", font=font)

    # save button information
    bob.buttons.append( Button(x,y,w,h, label) )
```

Now bob has a list of all of the generated buttons which we can search whenever the user clicks on the screen. First we need to make sure we are happy with our geometry. The mouse click is recorded at the point (mx,my). The conditions that must be true if this point is inside a button rectangle are shown in the following figure.

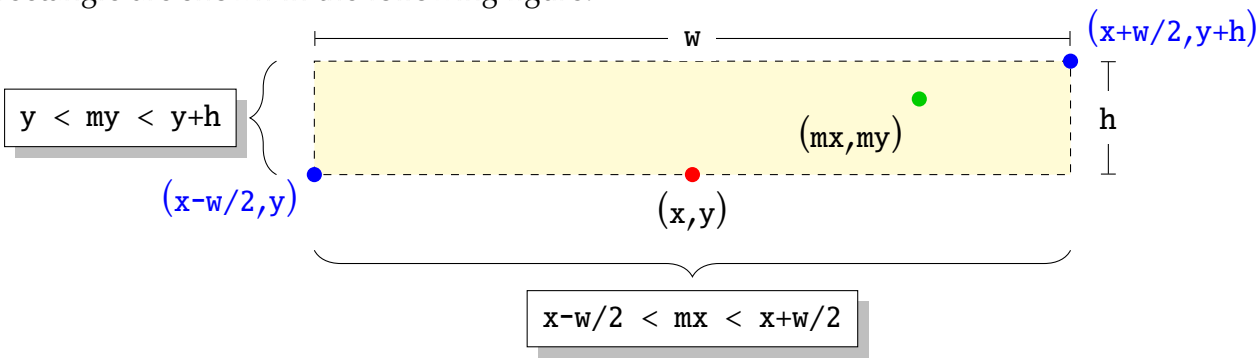


Figure 3 – Given a button and a point (mx,my), determine if point is inside button.

Modify your onclick function so that it check every button stored in bob.buttons using the conditions on (mx,my) shown in the above figure.

Run this code and verify that whenever you click on a button that button's label is printed.

Menu.py

```
def onclick(mx,my):

    for b in bob.buttons:
        if (b.x-b.w/2 <= mx <= b.x+b.w/2) and (b.y<=my<=b.y+b.h):

            print ("Button clicked =  %s " % b.label)
            break
```

So finally we want to run different code depending on which button has been pressed. To do this we check the label of the button.

Update the onclick function as shown below.

Menu.py

```
def onclick(mx,my):

    for b in bob.buttons:
        if (b.x-b.w/2 <= mx <= b.x+b.w/2) and (b.y<=my<=b.y+b.h):

            if b.label=="Press Me":
               print("Hey you pressed the first button")
            elif b.label=="No Press Me":
                print("I'm in the middle, like Golddielocks")
            elif b.label=="Forget them, PRESS ME":
                print("I'm button 3")
            else:
                print("Unknown button. label = %s" % b.label)

            break
```

2.2 It's Lego time — Putting your Programs Together

OK, we now have nearly everything to build our menu. To get the men
at the start of this worksheet I used the following code. You don't have

Menu.py

```
99
100 jump(bob,-350,160)
101 bob.write("Games ...", align="left",
102     font=("Comic Sans", 20, "normal"))
103
104 drawButton(y=100, w=350, label="Take Away Game")
105 drawButton(y=25, w=250, label="Monky Wars")
106
107 jump(bob,-350,-50)
108 bob.write("Turtle Graphics ...", align="left",
109     font=("Comic Sans", 20, "normal"))
110
111 drawButton(x=-200, y=-100, w=240, label="Squares")
112 drawButton(x=50, y=-100, w=240, label="Color Spiral")
113 drawButton(x=250, y=-100, w=150, label="Chess")
114 drawButton(x=-200, y=-175, w=240, label="Face")
115 drawButton(x=50, y=-175, w=240, label="Party")
116
117 drawButton(x=-200, y=-310,  w=300, label="Time to go home ...",
118         font=("Comic Sans", 30, "normal"),
```

Then in my `onclick` functions I have added **some** of the code needed to respond to the above button clicks.

Menu.py

```
72 def onclick(mx,my):
73
74     for b in bob.buttons:
75         if (b.x-b.w/2 <= mx <= b.x+b.w/2) and (b.y<=my<=b.y+b.h):
76
77             if b.label=="Take Away Game":
78                 os.system("python TakeAway_Graphical_Complete.py")
79
80             elif b.label=="Color Spiral":
81                 os.system("python Color_Spiral.py")
82
83             elif b.label=="Time to go home ...":
84                 print("Going home ... ")
85                 screen.bye()
86                 sys.exit(1)
87             else:
88                 print ("Unknown button clicked = %s" % b.label)
```

Notice in line 80 and in line 83 I run other python files. You will need to change this to match the name that you used for your files.