Université de Haute-Alsace
Université franco-azerbaïdjanaise

Master DSAI
2020-2021

Project-Mode applied Programming in Python

Practical Work : Making a Pong

Goals

Learn to build a Grahical User Interface in Python
Manage mouse and click events
Create a Pong

Instructions

Use Python at 100%, Python comes with lot of tools to make your life easy, use them.

Think "object", use object programming at your advantage.

Code neatly with well-chosen variables and functions/methods names. Add useful comments to your code in order to be able to understand it in some days.

Try to respect coding styleguides. I advise you to follow python styleguide PEP8 [1] or Google Python Styleguide [2].

Simple is beautiful. Do not try to code complicated, keep it simple, it will be more efficient and less error-prone.

Think before you code, take some time to draw/write your idea on a sheet. The tinking time before you code will save you a lot of debugging time after.

Read the documentation links that I give you. When using a new library, a coder spends lot of time on its documentation to understand how it works. There is an expression dedicated to this : RTFM which means 'Read The F... Manual'.

Read acarefully my instructions, they are here to help you (or try to ;-)).

---

1 The TkInter package

The TkInter package `https://docs.python.org/fr/3/library/tkinter.html` allows to easily create graphical user interface in Python. It allows you to manage a graphics window, buttons, as well as other classic objects (check boxes, list of options, etc.), it has a zone for "drawing" and allows you to retrieve keyboard events. or mouse.

1. PEP8 : `https://www.python.org/dev/peps/pep-0008/`
2. Google Python Styleguide : `https://google.github.io/styleguide/pyguide.html`

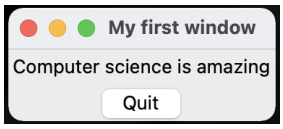
1.1 Create our first graphical window

To create your first window, you can use the following code.

```
import tkinter as tk

window = tk.Tk()
window.title('My first window')
text1 = tk.Label(window, text="Computer science is amazing", fg='black')
text1.pack()
button1 = tk.Button(window, text='Quit', command=window.destroy)
button1.pack()
window.mainloop()
```

which must give you the following window when executed :



Analyze what is going on

```
import tkinter as tk
```

This line allows us to import `Tkinter` and call it using `tk`

```
window = tk.Tk()
```

This instruction creates a graphics window and attaches it to the variable `window`

```
window.title('My first window')
```

This instruction defines a title for our window, note that now we can call **methods** related to the window object we created. Remind that a **methods** are **functions** internal to an object.

```
text1 = tk.Label(window, text="Computer science is amazing", fg='black')
```

`Label` allows you to display text in a **widget** added to your window and has several options [3].

```
text1.pack()
```

Adapt the size of your **widget**, without options this will reduce it to the minimum necessary for its content.

```
button1 = tk.Button(window, text='Quit', command=window.destroy)
button1.pack()
```

These instructions add and pack a button widget which also has several options [4]. Here, in addition to the text to display, we specify the function to be executed when we click on this button. In this case, we destroy the window.

```
window.mainloop()
```

`mainloop` launches the window display and listens to the various events, for example for this window by clicking on the button.

3. https://docs.python.org/3/library/tkinter.ttk.html#label-options
4. https://www.tutorialspoint.com/python/tk_button.htm

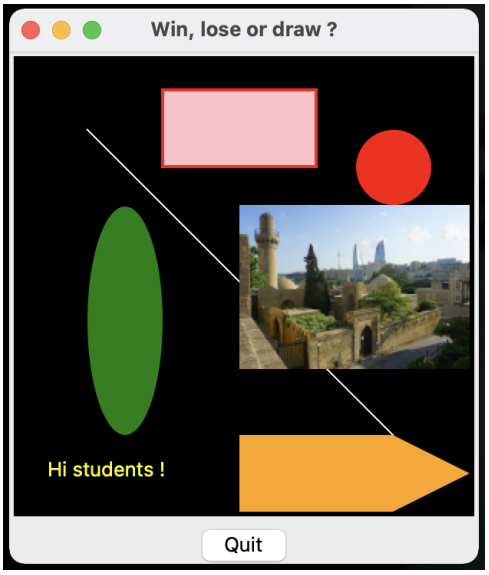
1.2 Draw a little

To allow you to draw shapes and different things, `TkInter` has the `canvas` [5] widget.
Go back to the sample code below.

```
import tkinter as tk

window = tk.Tk()
window.title("Win, lose or draw ?")
canvas1 = tk.Canvas(window, bg="black", width=300, height=300)
# Drawing instructions
line1 = canvas1.create_line(50, 50, 250, 250, fill='white')
oval1 = canvas1.create_oval(50, 100, 100, 250, fill='green')
oval2 = canvas1.create_oval(225, 50, 275, 100, fill='red')
polygon1 = canvas1.create_polygon(150, 250, 250, 250, 300, 275, 250, 300, 150, 300, fill='orange')
filename = tk.PhotoImage(file="img/oldtown.png")
image1 = canvas1.create_image(300, 100, anchor=tk.NE, image=filename)
rectangle1 = canvas1.create_rectangle(100, 25, 200, 75, fill='pink', outline='red', width='2')
text1 = canvas1.create_text(25, 280, anchor=tk.SW, font="Calibri", text="Hi students !", fill='yellow')
canvas1.pack()
button1 = tk.Button(window, text='Quit', command=window.destroy)
button1.pack()
window.mainloop()
```

Run it, you should get the following window :



Normally, the different examples should give you the option of making displays of almost everything now.

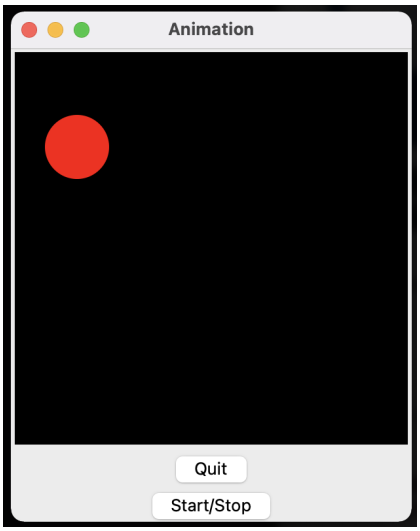
2 Animation

To create an animation, we will have to modify the content of our canvas at regular intervals. For this we will rely on the function `After` [6] which is a method of your graphics window which is called with two arguments, the first the waiting time in milliseconds and the second the function to be executed after this waiting time.

5. https://www.tutorialspoint.com/python/tk_canvas.htm
6. https://www.geeksforgeeks.org/python-after-method-in-tkinter/

2.1 A moving disk

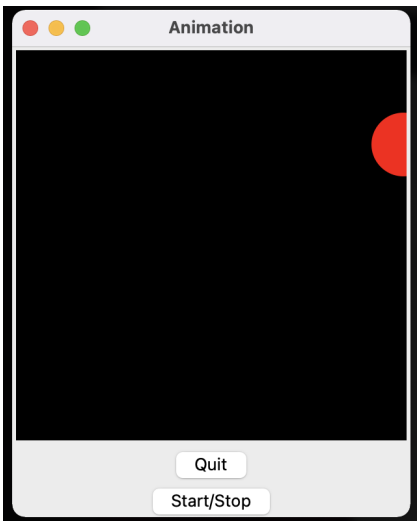
Make an application that looks like the following capture.



Clicking on button `Start` must call a function moving the red disk horizontally, thanks to the function `move` [7]. To allow the function you called after the click to be recalled every XX milliseconds, you will need to use the `after` introduced before.

2.2 Keeping the disk in the window

Great, we have an animation that works but our disk tends to come out of the window like in the screenshot below.



What is needed is to change the direction of movement when the disk touches the edge. We should therefore know the position of our object, which is possible thanks to the `coords` [8] method.

Modify the code from the previous exercise so that the red circle goes back in the opposite direction when it touches the edge of the canvas.

7. https://www.geeksforgeeks.org/python-tkinter-moving-objects-using-canvas-move-method/
8. http://tkinter.fdex.eu/doc/caw.html#Canvas.coords

2.3 Now let's do a real bouncing ball program

Take the code from the previous exercise but add the following features :

— Transform the "Start" button into a "Start/Stop" button and make sure that it also allows you to stop the animation

— Your object can now also move vertically, implement random [9] initialization of horizontal and vertical speeds.

— Now handle collisions with top and bottom edges as well

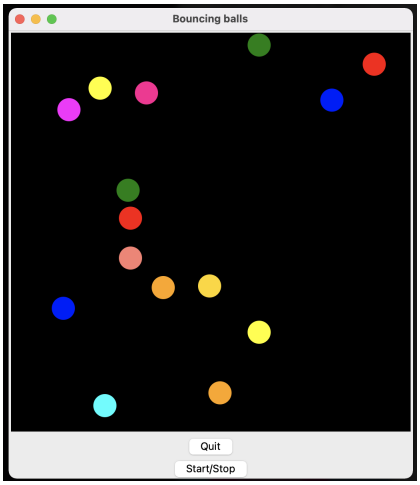
2.4 And with multiple objects ?

— Modify the code to have X balls of different color, think of an elegant and generic way to manage them to be able to manage any number of balls (hint : put them in a list for example).

— For now, we manage the displacement with a horizontal and vertical speed parameter, this is not optimal, modify the code so that each object has an absolute speed and a direction (hint : create a class **Ball**)

— Finally manage the collisions between the different balls.

It should look like the screenshot below.



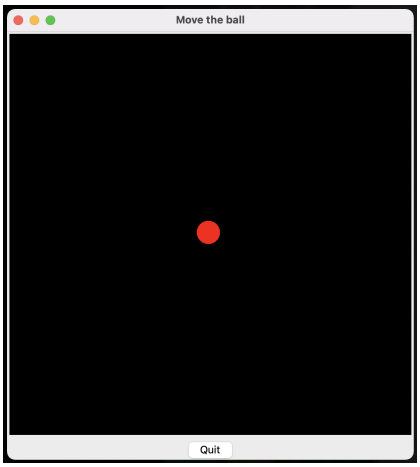
3 Retrieve the keys the user presses

To make our application a little more interactive, we are now going to retrieve the keys the user presses, in this case the ones that interest us are the arrows, and we will then use them to move an object in our window.

3.1 Let's create a window with a ball in the center

Write the python code to create the window below :

9. https://docs.python.org/3/library/random.html



Reuse what you learned in the last lab (but without dumbly copy/pasting!).
Think about genericity, having global parameters for the size of the window and the size of your ball, so you can easily change them if needed.

3.2 It likes to move

`tkInter` allows you to do `binding` (links) between a key and a function so that the press of a key by a user triggers a function. So this uses a mechanism called `callback` like the one we used last time with the `after` function. You can learn more about it here : `https://effbot.org/tkinterbook/tkinter-events-and-bi htm`. The function that you should use to link the pressed key to the function it should trigger is the `bind` function. You can of course find other info on the Internet.

⚠ The special keys you will need are `<Up>`, `<Down>`, `<Left>` and `<Right>`

3.3 It is alive but ...

Now your ball is moving but ... it comes off the screen too ...

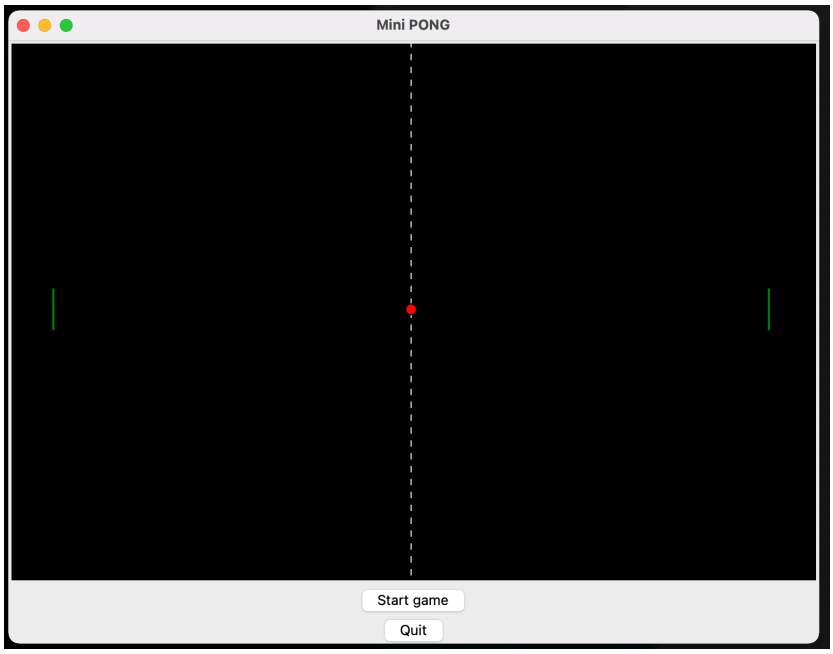
Verify that the ball is still in the drawing area and corrects the coordinates of the object to put it back at the edge of the area if it is ever outside.

4 So, now we do a mini-pong?

We now know how to control a ball but it's not very fun, we will now try to create a little game with a moving ball and elements controlled by players ... does that remind you of anything? Yes, we are going to recreate one of the first games in history, PONG.

4.1 Graphics

Recreate the graphics window below by drawing its elements.



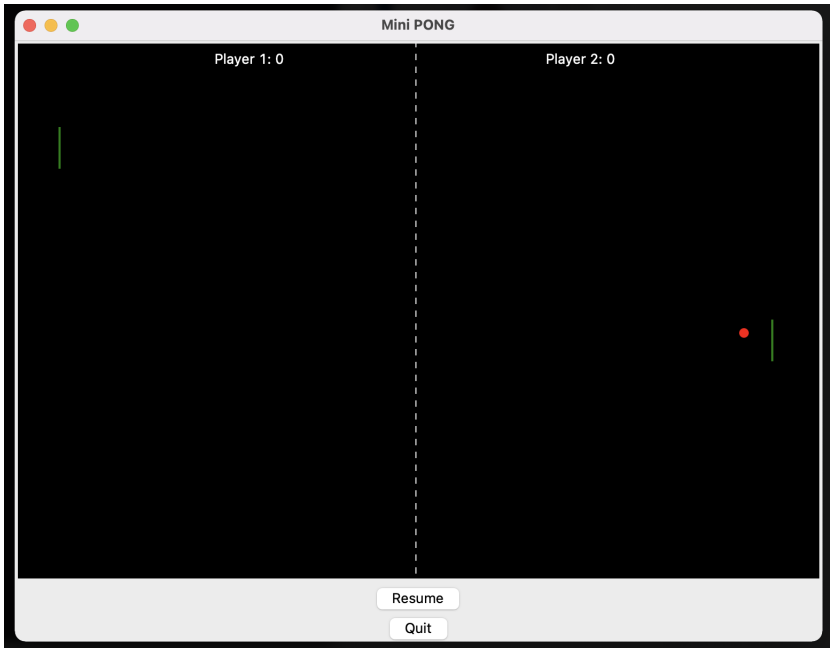
⚠To draw the dotted line in the middle of the field, the option `dash` of `create_line` can be useful.

4.2 Paddle's animation

Now add to your code the possibility of moving the paddles, with the keys `A` and `q` for player 1 (left) and with the up and down arrows for player 2 (right).

4.3 And the ball?

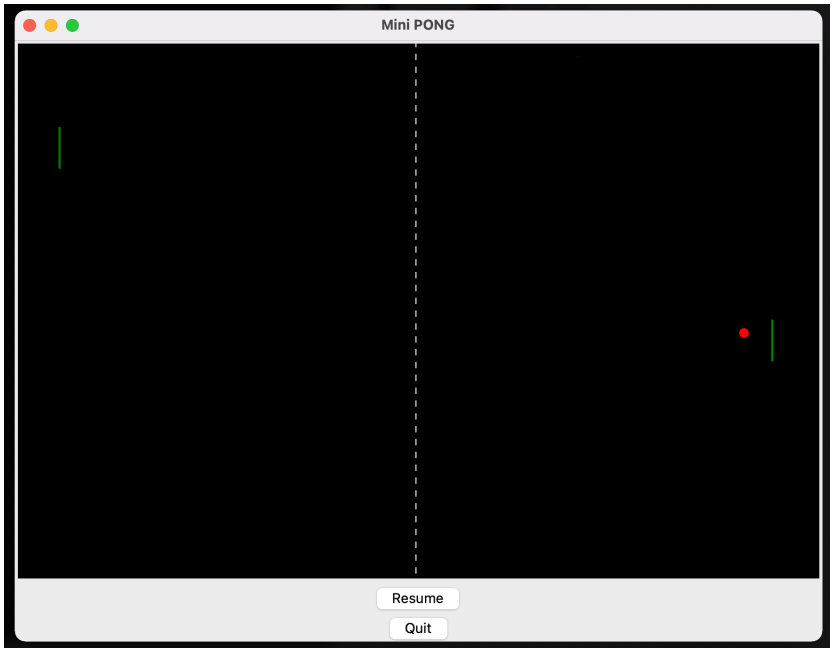
We must now manage the movements of the ball :
— Make sure that the ball can randomly go to the 4 angles (45, 135, 225 and 315) from the center
— Manage collisions with the floor and the ceiling
— If the ball comes out to the right or to the left, start it again from the center
— Manage collisions with paddles that cause it to start again in the other direction
You will then get something similar to what we have below.



⚠Paddles collisions are not trivial to handle, think about it.

4.4 Do we keep scores?

Now we are going to add score management.
— Add score display as in the screenshot below
— Update the score every time a player scores a point.
⚠To update the score the `Itemconfig` of `tkInter` will be very useful to you!!



4.5 Nice game but a little bit boring, no?

We are now going to add a little challenge, in fact as the ball is relatively slow, the game is simple. To make matters worse, make sure that each time a player returns the ball, the speed increases a little (by 10% for example).

4.6 Drop shot, forehand and striking backhand!

Let's spice up the game a little more, adding the possibility of special moves according to the movements of the rackets and the reflexes of the players.

We can consider the following three cases :
— The player has not moved the racket recently, we are in the classic case with a very small acceleration of the ball
— The player moved his racket less than X seconds ago (for example $x = 0.1$) :
  — The movement of the racquet went in the same direction as that of the ball, we accelerate the ball a little more, it's a forehand!
  — The movement of the racket did not go in the same direction as that of the ball, we have a random choice :
    — Drop shot : the ball continues in the same direction but is slowed down (no luck)
    — Backhand : the ball changes direction and is also strongly accelerated (well done!)

Implement this feature! Properly balance the acceleration percentages, times since last movement, and drop shot/backhand probabilities to get a fun and tactical PONG!

5 Bonus

You already finished? Congratulations, now you can try to make a Snake [10] game.

10. https://en.wikipedia.org/wiki/Snake_(video_game_genre)