ECE 122: Introduction to Programming for ECE- Spring 2021

Project 3: Drawing and Animations with Tkinter (OOP and Graphics examples)

Due Date: **See website, class policy and moodle for submission**
This is an individual project (discussions are encouraged but no sharing of code)

Description

The goal of this project is to practice some OOP using class, instance variables and methods. You will also operate Graphics.

Your project zip file must include five files:

1. `Mapping_for_Tkinter.py`: module containing the class `Mapping_for_Tkinter` (which was part of the bonus pre-submission and which will provided after your first bonus submission).
2. `Racket.py` application file that includes the class `Racket` and its main function for testing the animation.
3. `Ball.py`: application file that includes the class `Ball` and its main function for testing the animation.
4. `game1.py`: application main file that makes use of the class `Racket` and `Ball`.
5. `game2.py`: another application main file that makes use of the class `Racket` and `Ball`.

The project is designed to be incremental, you can then debug, test and run your code after each new task/option is implemented. Do not forget to comment your code. Make sure you obtain the **exact same output** for the **exact same input** for the application examples. Your program will also be tested with different inputs by the graders.

Submission/Grading Proposal

You will regroup all your files into one zip file at the time of submission. This project will be graded out of 100 points:

1. Your program should implement all basic functionality/Tasks and run correctly (100 points).
2. Overall programming style: program should have proper identification, and comments. (-5 points if not).

Task1- Racket (simple move)- [25pts]

When running `Racket.py`, the main function of the program is executed. A square window of dimension LxL should appear with a black rectangle (that we will be calling a racket) of x-length `Lx=L/10` and y-width `Ly=L/60` (in the middle and at the bottom of the window) as shown in Figure 1.

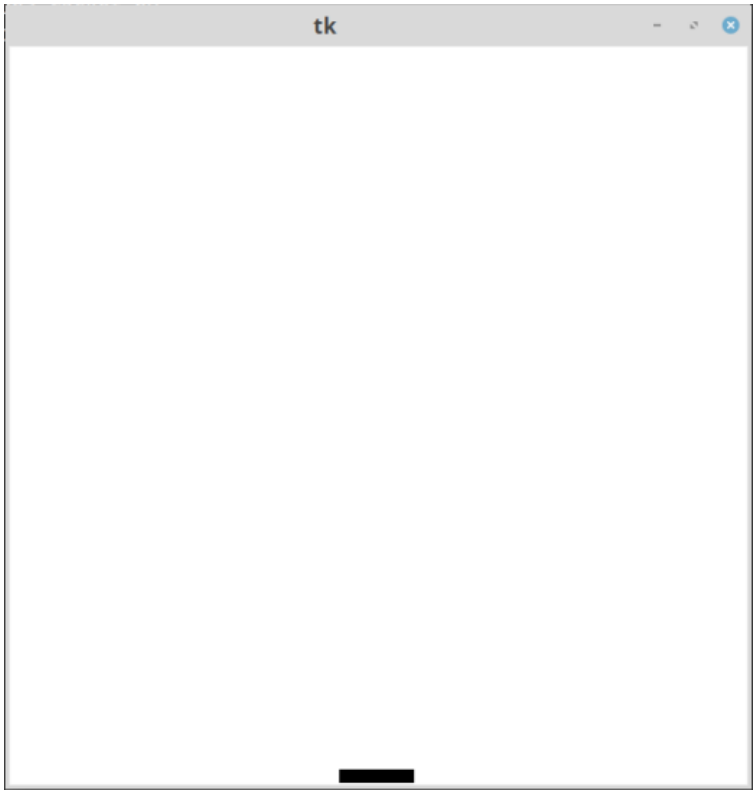

Figure 1: `Racket.py`

Requirement

- The Tkinter window is using the `Mapping_for_Tkinter` which must be defined as follows: `mapping=Mapping_for_Tkinter(-L/2,L/2,-L/2,L/2,L)`
  It means that using the mapping the middle of the window is located at x=0,y=0.
- Initially, the racket will be located at x=0 and y=ymin+Ly/2.
- As soon as your move your mouse cursor on top of the window, you will be able to use either the left click mouse or the right click mouse. Each click is associated with an action. Using a left click, the racket shifts by `Lx/2` pixels to the left. You can keep clicking left until the edge of the racket reaches `xmin`. Using a right click, the racket shifts by `Lx/2` pixels to the right. You can keep clicking right until the edge of the racket reaches `xmax`. Example video is provided in `racket.mp4` (but you do not see the mouse clicks in the video).

How to proceed?

- The file `Racket.py` must contain the class `Racket` and a main function.

- The constructor should accept four arguments: the mapping, the canvas, the length of the racket `Lx` and its width `Ly`. The four arguments are used to initialize the instance attributes (all public). Three more public instance attribute must be considered: (i) the position x of the center of the racket which is initialized to 0, (ii) the position y of the center of the racket which is initialized to ymin+Ly/2, (iii) a rectangle object that you must create and initialize from the canvas object (using the function `create_rectangle`).

- You should implement a method `shift_left` that will be used to move your object rectangle left by `Lx/2` pixels (only if the move is permitted...make sure you do not reach the boundary of the window).

- You should implement a method `shift_right` that will be used to move your object rectangle right by `Lx/2` pixels (only if the move is permitted....make sure you do not reach the boundary of the window).

- You must implement a function `main()` that will: (i) create the mapping, (ii) create the window/root and canvas, (iii) instantiate your object racket, (iv) bind the action of clicking to the mouse with the corresponding action. For example, to bind the left click with the action of shifting left use the following command (if your object name is myracket):
  `canvas.bind("<Button-1>",lambda e:myracket.shift_left())`
  Similarly you can bind the action of clicking right using `"<Button-3>"`. We note that `lambda` is a Tkinter keyword used to create a link between a Tkinter event `e` and a callback function.

- at this point everything should work fine if you end your main function with the method `.mainloop()`

Task2- Ball (bouncing)- [25pts]

It is time to do a bit of (simple) physics. Let us execute the application `Ball.py`. You should be getting (with default values):

```
Enter window size in pixels (press Enter for default 600):
Enter velocity and theta (press Enter for default: 500 pixel/s and 30 degree):
Total time: 8.279999999999868s
```

The last statement is printed after 10 rebounds of the ball with the sides of the window, the animation of the bouncing ball will then stop. Two examples video of the bouncing balls are included in this project: `ball1.mp4` that uses the default values, and `ball2.mp4` which is associated with this input/output:

```
Enter window size in pixels (press Enter for default 600):
Enter velocity and theta (press Enter for default: 500 pixel/s and 30 degree):1000 90
Total time: 5.609999999999925s
```

How to proceed?

1. The file `Ball.py` must contain the class `Ball` and a main function.

2. The constructor should accept six arguments: the mapping, the canvas, the initial x0 and y0-position of the center of the ball which is (0,0) to start with, the velocity and the angle of the direction of the motion (the 0 angle is traditional horizontal right axis from the center of the ball, $\pi/2$ or $90 degree$ is the top vertical axis from the center of the ball, etc.). These arguments are used to initialize the public instance attributes. You will consider three additional instance attributes, the current x and y position of the ball (initialized at x0,y0 to start with) and a circle object that you must initialize (using the function `create_oval`). The ball itself is a filled blue circle of radius `R=L/120`, where `L` is the width of the canvas. Actually, it would make sense to add the radius as instance attribute.

3. You should implement a method `update_xy` that will update the x-y coordinates of the ball after a certain time t has passed. As a result, the method needs the time t as argument. Starting from x0,y0 at time t=0, the new position of the ball can be obtained using the following simple formula (**Attention:** θ is in radian):

$$x = x_0 + v\cos(\theta)t$$

$$y = y_0 + v\sin(\theta)t$$

   To move the ball, you may want to use the method `coords` of the canvas object. Before moving the ball, however, several conditions that are defining the event, must be taken care of. We call a single event, the ball trajectory/animation until it reaches one side (one boundary). A rebound will happen when the side of the ball (i.e. center of the ball +/- `R` pixels horizontal or vertical) reaches one side. Here are some more info:

   - If the center ball reaches either the left (xmin+R) or right (xmax-R) sides, its angle will change to $\pi - \theta$
   - If the center ball reaches either the bottom (ymin+R) or top (ymax-R) sides, its angle will change to $-\theta$
   - As soon as a rebound occurred you will re-initialize x0,y0 at the positions x and y (the method will then be ready to start a new event– the time t will also be set again to 0 in the main program).

4. Finally, the method `update_xy` must have a return `int` value that should be equal to 0 if no boundary window has been reached (different than 0 otherwise). It is useful for the rest of the project to identify the boundary, you could return 1 for the bottom boundary, 2 for the top, and 3 and 4 for the left and right, respectively.

5. Most of the function `main` is provided for you! You will be using the mapping and canvas defined in Task1. As you can see, at each `while` loop iteration, both the time between events and the total time are incremented by 0.01s. The code is keeping track of the number of rebounds and it is setting the event time to 0 again as soon as a rebound occurs. After 10 rebounds, the loop stops.

Task3- Game1- [25pts]

This program must use the class `Racket` and `Ball`, to create a simple game. In this game, the ball will make rebound and you will move your racket in order to make sure you can get a rebound at the bottom (like playing squash). You loose if you miss the ball... An example video is included in this project: `game1.mp4`. This was the corresponding output:

```
Game over! Total time: 21.350000000000538s
```

Requirement

- You will use the same window/canvas defined in Task-1 and Task-2, the game starts by having the center of the ball located at x=0 and y=ymin+Ly+R (middle top of the racket which has a thickness of Ly). The ball must have an initial velocity of 200, and an angle of 53 degree.
- the time step (real and simulation time) should be 0.01s
- At the bottom, the ball is making a rebound when its bottom touches the top of the racket.
- When the ball is making a rebound at the top, its velocity increases by 25%
- When the ball is making a rebound at the top, its new angle becomes random (think of a surface roughness), the angle can be chosen uniformly at random between -170 and -10 degree (both included).

How to proceed?

- As in Task-1 and Task-2, create the window, create the canvas, create the racket and bind it to the mouse clicks, create the ball. As in Task-2 starts the simulation loop (very similar loop).
- Once the value of the `update_xy` method is returned, you need to add extra conditions in the main code that may change velocity, angle, or check if the x ball coordinate is within the racket range (it is game over otherwise).
- You should have noted that the bottom rebound is not at ymin+R anymore, but ymin+R+Ly. You can easily modified the method `update_xy` to include an optional argument that can account for this bottom shift. Attention: your new method should still work fine with Task 2.
- Use the function `uniform` from the random module to define the new angle after the the top rebound. No seed needed here, since we do not want reproducibility (this is a game!). Hint: `random.uniform(1,9)` returns a float between $[1, 9]$.

Task4- Game2- [25pts]

This program is also using the class `Racket` and `Ball`, to create another simple game. In this game, there will be two rackets that you will control. As soon as the ball leaves racket 1 at the bottom, you will get control of the racket 2 at the top, and vice-versa. An example video is included in this project: `game2.mp4`. This was the corresponding output:

```
Game Over for racket 2!
```

How to proceed?

- Create the window, create the canvas (same as Task-3), create two rackets, create the ball. At the start of the game, the second racket on top should be located ax x=0, y=ymin-Ly/2. In order to be achieve this, you could add to the racket constructor, an optional argument (or arguments) that pinpoints the racket position (Your code should still work fine with Task-2 and Task-3).
- At the start of the simulation, the ball is located at the top of the first racket (like Task3), its velocity will be constant throughout the entire game v=300 pixels/s, you will set the initial angle to 45 degree.
- The binding between mouse clicks and rackets need to be redefined at each time you are taking turn. When the ball is moving up, it is the turn of the top racket 2 to be activated. When it is moving down, that will be the turn of the bottom racket 1.
- As soon as one racket is activated, its color switches to red. You may want to add a couple of methods to the class `racket`, one that can be used to activate the racket (turn it red), and another one used to deactivate (turn it black again). Hint: you can use the `itemconfig` method of the canvas object.
- Similarly to Task3 where the bottom rebound may happen at ymin+R+Ly from the center of the ball (if it hits the racket), the top rebound would happen at ymax-R-Ly. Again, you could modified the method `update_xy` to include another optional argument that can account for this top shift. Your new method should still work fine with Task-2 and Task-3.
- Finally, as for Task-3, the new random angle after the the top rebound is chosen at random between [-170,-10], while the angle after the bottom rebound is chosen between [10,170]. No seed needed here, which makes the game more unpredictable.

Task5- 10pts [bonus- no help from TA]

Using the class Racket and Ball, design your own Game 3. It must be fun to play, colorful with some nice features. Example: fancier Game 2 like including obstacles, or a pinball game, or a space invader? When you run your program, you should also display a short description of your game's rule. You are allowed to add optional arguments to constructors, create new methods, etc. but everything should be backward compatible with previous tasks.