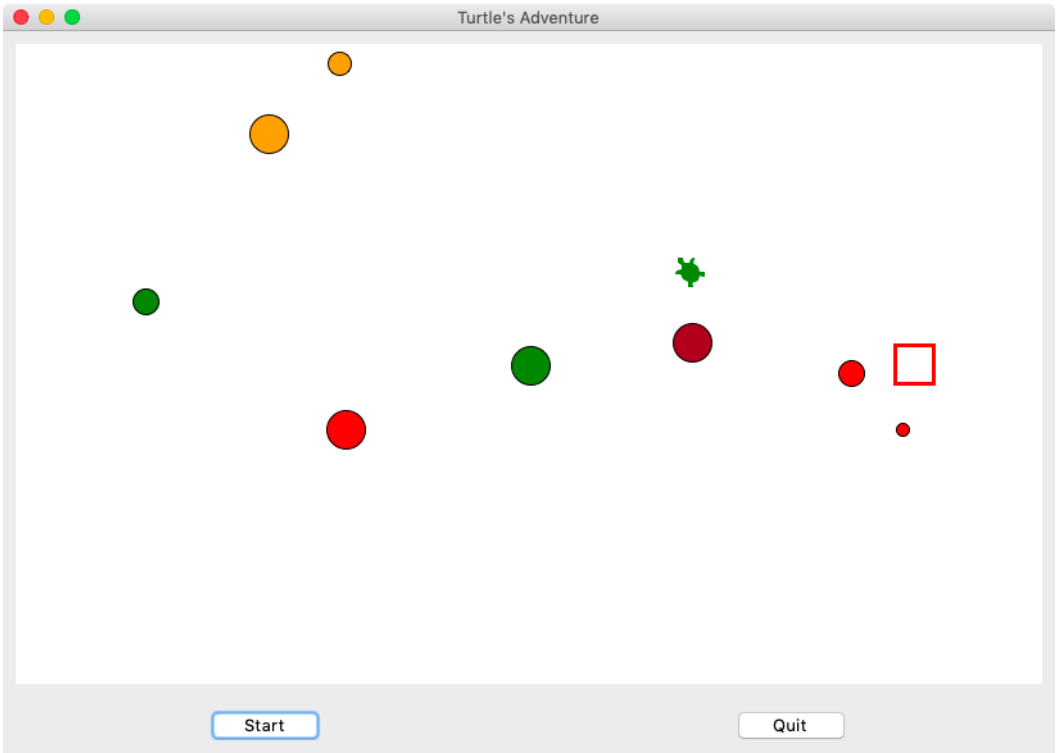
Lab 6: Turtle's Adventure

01219116/117 Computer Programming II

In this lab assignment, we will create a simple game that utilizes Python's turtle graphics and animation, as shown in the picture below.



The game consists of a turtle character that the player can control with a mouse. The player clicks on the canvas to set a waypoint for the turtle to walk to. The goal of the game is to bring the turtle back to his home, depicted by the red square. However, several enemies, presented by circles of various colors, are lurking around in the area. Some may stay still, while others may move in certain patterns. If the turtle gets hit by one of them, he is dead and the player loses the game. If the turtle arrives at the red square, the player wins. The game ends once the player loses or wins.

1. Preparing Game Window and Canvas

Let us define the `TurtleAdventure` class inheriting from the `Frame` class. Use the following code as your starting point. The initial code already creates a canvas inside the application's frame, along with a Start button for starting and pausing the game, and a Quit button to quit the game. At this point, the Start button is bound to a method that does nothing. The application window is made unresizable by calling `root.resizable(False, False)`.

```
import tkinter as tk
import tkinter.ttk as ttk

CANVAS_WIDTH = 800
CANVAS_HEIGHT = 500

class TurtleAdventure(ttk.Frame):

    def __init__(self, parent):
        super().__init__(parent)
        parent.rowconfigure(0, weight=1)
        parent.columnconfigure(0, weight=1)
        self.grid(row=0, column=0, sticky="news")
        self.rowconfigure(0, weight=1)
        self.columnconfigure(0, weight=1)
        self.columnconfigure(1, weight=1)
        self.create_widgets()

    def create_widgets(self):
        self.canvas = tk.Canvas(self, borderwidth=0,
            width=CANVAS_WIDTH, height=CANVAS_HEIGHT, highlightthickness=0)
        self.canvas.grid(row=0, column=0, columnspan=2,
            sticky="news", padx=10, pady=10)
        self.btn_start_top = ttk.Button(self, text="Start",
            command=self.toggle_animation)
        self.btn_start_top.grid(row=1, column=0, pady=10)
        ttk.Button(self, text="Quit", command=root.destroy).grid(
            row=1, column=1, pady=10)

    def toggle_animation(self):
        pass

if __name__ == "__main__":
    root = tk.Tk()
    root.title("Turtle's Adventure")

    # do not allow window resizing
    root.resizable(False, False)
    app = TurtleAdventure(root)
    root.mainloop()
```

The initial application window should look like this.

2. Introducing the Turtle

We will define a class for the main turtle character by subclassing the `RawTurtle` class from the Python's `turtle` module. However, the coordinate system used by the turtle graphics defines the origin at the center of the canvas, and the y-axis value increases toward the top of the window. We use the turtle screen's `setworldcoordinates()` method to adjust the coordinate system so that the origin is located at the top-left corner of the canvas, and the y-axis value increases towards the bottom of the screen. This is the coordinate system we are more familiar with. At the end of the constructor, we move the turtle to the middle and left side of the screen.

```
from turtle import RawTurtle

class Turtle(RawTurtle):

    def __init__(self, canvas):
        super().__init__(canvas)
        self.screen.setworldcoordinates(0, CANVAS_HEIGHT-1, CANVAS_WIDTH-1, 0)
        self.shape("turtle")
        self.color("green")
        self.penup()
        self.goto(10, CANVAS_HEIGHT/2)
```

We need to create an instance of this Turtle class inside our application class's constructor.

```
class TurtleAdventure(ttk.Frame):

    def __init__(self, parent):
        :
        self.turtle = Turtle(self.canvas)
```

Try launching the application. A green turtle should show up and immediately moves toward the middle-left position of the screen.

3. Animating the Turtle

Now we want to be able to click anywhere on the canvas to set a waypoint for the turtle to move to at a specified speed. However, we don't want the turtle to move to a waypoint in a single stretch because other objects (which we will add later) may also be moving at the same time. To do so, we define the `animate()` method that gradually moves the turtle towards the waypoint in accordance with the defined turtle's speed. The `set_waypoint()` method is also defined for convenience. By calling `animate()` repeatedly, the turtle will be moving towards the waypoint, if there is one defined.

As we perform movement animation ourselves, we need to disable the built-in animation feature provided by the turtle module. Calling the screen's `tracer(False)` will disable the turtle's update animation entirely, so we need to call the screen's `update()` method ourselves after each turtle's movement. The code implementing this part is shown below.

```
class Turtle(RawTurtle):

    def __init__(self, canvas, speed=3):
        :
        self.speed = speed
        self.waypoint = None
        self.screen.tracer(False) # disable turtle's built-in animation

    def animate(self):
        if self.waypoint is not None:
            x,y = self.waypoint
            if self.distance(x,y) >= self.speed:
                self.setheading(self.towards(x,y))
                self.forward(self.speed)
        self.screen.update()

    def set_waypoint(self, x, y):
        self.waypoint = (x,y)
```

Let us now allow the player to define a waypoint using a mouse click by binding **<Button-1>** event to a function that calls the turtle's `set_waypoint()` method. Note that when such a click event is triggered, detailed event information is passed to the event handler as an argument. Hence, we can extract the (x,y) coordinates of the click location and use them to set the turtle's waypoint.

Back to our main `TurtleAdventure` class. To toggle the animation, we implement the `toggle_animation()` method (currently bound to the Start button) to update the button's text and call the `animate()` method to start the animation. The `animate()` method calls the turtle's `animate()` method and schedules a timer to call itself repeatedly every 33 milliseconds, which effectively triggers the animation approximately 30 times per second. The code for this part is shown below.

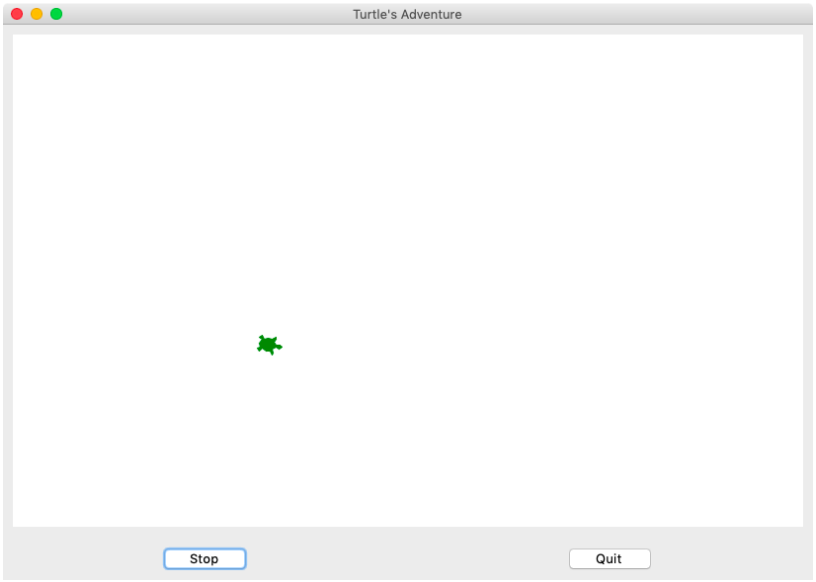
```
class TurtleAdventure(ttk.Frame):

    def __init__(self, parent):
        :
        self.canvas.bind("<Button-1>", lambda e: self.turtle.set_waypoint(e.x, e.y))
        self.is_animating = False

    def toggle_animation(self):
        self.is_animating = not self.is_animating
        if self.is_animating:
            self.btn_start_top.config(text="Stop")
            self.animate()
        else:
            self.btn_start_top.config(text="Start")

    def animate(self):
        self.turtle.animate()
        if self.is_animating: # schedule the next update
            self.after(33, self.animate)
```

Launch the application. Click the Start button to start the animation. Then click anywhere on the canvas to see the turtle moving towards the click point.



4. Home Sweet Home

Let us make a home for the turtle using a square canvas object. Instead of just creating a rectangle item directly inside our `TurtleAdventure` class, we will define a new `Home` class for modularity and ease of future development.

```
class Home:
    def __init__(self, canvas, size, pos):
        self.canvas = canvas
        self.size = size
        self.pos = pos
        x,y = pos
        self._id = canvas.create_rectangle(
                x-size/2, y-size/2, x+size/2, y+size/2,
                outline="red", width=3)

    def contains_turtle(self, turtle):
        x,y = self.pos
        size = self.size
        x1,x2 = x-size/2,x+size/2
        y1,y2 = y-size/2,y+size/2
        tx,ty = turtle.pos()
        return x1 <= tx <= x2 and y1 <= ty <= y2
```

Our `Home` class defines the `contains_turtle()` which checks whether the specified turtle is inside of the perimeter. We then create one instance of Home inside our `TurtleAdventure` class and place it on the right side of the screen.

The next part is to check whether the turtle has arrived home by calling the `contains_turtle()` method above every time an animation update just took place. If the turtle is already inside, a pop-up message box is displayed and the application is terminated. The pop-up message box is provided by Tkinter's `messagebox` module, which must be imported at the beginning of the program.

```
from tkinter import messagebox
:
class TurtleAdventure(ttk.Frame):

    def __init__(self, parent):
        :
        self.home = Home(self.canvas, size=30, pos=(CANVAS_WIDTH-100,CANVAS_HEIGHT/2))

    def animate(self):
        :
        if self.home.contains_turtle(self.turtle):
            messagebox.showinfo(title="Turtle Game",
                message="Turtle is home!  You win.")
            root.destroy()
            return
```

Try out our application. Click Start and click inside the red box to set a waypoint. Watch the turtle moving to it. Once the turtle is inside the box, a message will appear, as shown.

5. Watch Out! Enemies!

It's time to add some enemies to the game. All enemies are represented by circles. Let's start with the `BasicEnemy` class below to implement basic, non-moving enemies. The `hits_turtle()` method checks whether the turtle's center pixel is inside the enemy's circle, which means the turtle is hit by the enemy. (A better collision-detection method could be used, but we just keep the code simple for now.) The location of each enemy is kept in its `pos` attribute. However, changing this attribute won't affect its appearance on the screen until the `render()` method is called. The empty `animate()` method is meant to be overridden by derived classes but it has to be defined here so that we can call `animate()` for all enemy-related classes without getting an error.

```
class BasicEnemy:

    def __init__(self, canvas, size=30, pos=(CANVAS_WIDTH/2,CANVAS_HEIGHT/2)):
        self.canvas = canvas
        self.size = size
        self._id = canvas.create_oval(0, 0, size, size)
        self.pos = pos
        self.set_color("yellow")
        self.render()

    def set_color(self, color):
        self.canvas.itemconfigure(self._id, fill=color)

    def render(self):
        x,y = self.pos
        self.canvas.moveto(self._id, x-self.size/2, y-self.size/2)

    def animate(self):
        pass

    def hits_turtle(self, turtle):
        x,y = self.pos
        return turtle.distance(x,y) < self.size/2
```

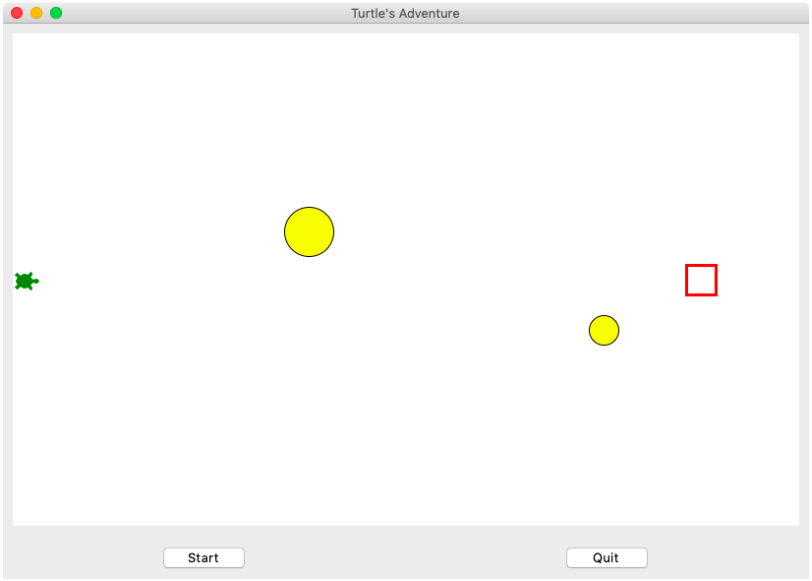
Let's add a few basic enemies into the game by modifying the `TurtleAdventure`'s constructor. We will also revise the main `animate()` method to animate all enemies and check whether one of them has hit the turtle. If this happens, a pop-up message will be displayed and the application is terminated.

```
class TurtleAdventure(ttk.Frame):

    def __init__(self, parent):
        :
        self.enemies = [
            BasicEnemy(self.canvas, size=50, pos=(300,200)),
            BasicEnemy(self.canvas, size=30, pos=(600,300)),
        ]

    def animate(self):
        :
        for enemy in self.enemies:
            enemy.animate()
            if enemy.hits_turtle(self.turtle):
                messagebox.showinfo(title="Turtle Game",
                    message="Turtle is dead!  You lose.")
                root.destroy()
                return
```

Launch the application to see two enemies appear as yellow circles.



Start the game and try moving the turtle towards one of the enemies. Once the turtle collides with the enemy, a message box should appear.

6. Moving Enemies

Playing with static enemies is boring. Let's define another class that allows enemies to move. We subclass the `BasicEnemy` and call it `MovingEnemy`. The moving script is defined in the `script()` method, which gets called to create a generator/coroutine to be called repeatedly by the `animate()` method. The random module is used to generate random movement. The `speed` argument can be provided to the constructor to control how far the enemy can move in each step. The code also makes sure the enemy will not fall off the screen. Enemies of this class are drawn in orange to differentiate them from yellow basic enemies.

If the coroutine script has stopped for any reason, the `animate()` method will simply set the internal variable `_coro` to `None` so that it will no longer attempt to run the script during the next animation trigger.

```
import random
:

class MovingEnemy(BasicEnemy):

    def __init__(self, canvas, size=30, speed=2):
        super().__init__(canvas, size=size)
        self.speed = speed
        self.set_color("orange")
        self._coro = self.script()

    def move(self, x, y):
        self.pos = (x,y)

    def animate(self):
        if self._coro is None:
            return
        try:
            next(self._coro)
            self.render()
        except StopIteration:
            self._coro = None
```

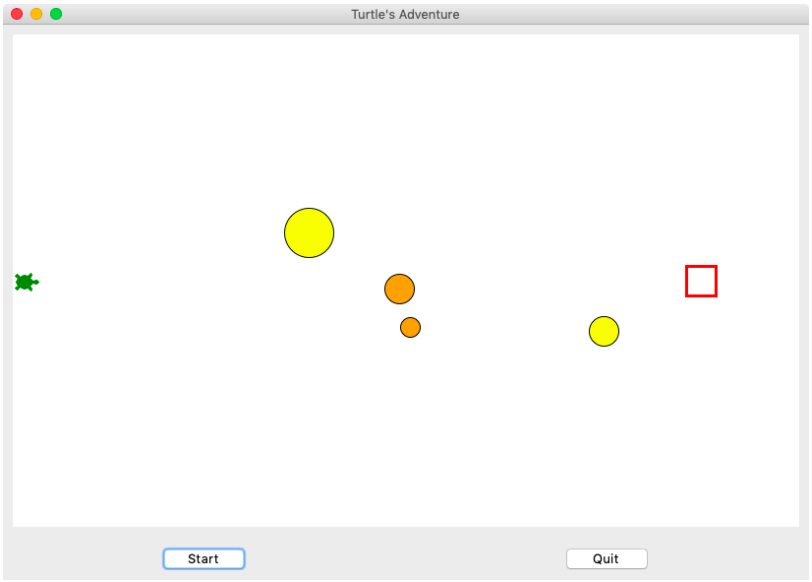
```
    def script(self):
        while True:
            x,y = self.pos
            newx = x + random.randint(-self.speed, self.speed)
            newy = y + random.randint(-self.speed, self.speed)
            newx = min(max(newx, 0), CANVAS_WIDTH)
            newy = min(max(newy, 0), CANVAS_HEIGHT)
            self.move(newx, newy)
            yield
```

Let's add our new type of enemies.

```
class TurtleAdventure(ttk.Frame):

    def __init__(self, parent):
        :
        self.enemies = [
            BasicEnemy(self.canvas, size=50, pos=(300,200)),
            BasicEnemy(self.canvas, size=30, pos=(600,300)),
            MovingEnemy(self.canvas, size=30),
            MovingEnemy(self.canvas, size=20, speed=5),
        ]
```

Our game should have two more enemies in orange randomly walking around.



7. Fencing Enemies

Our next enemy class is called `FencingEnemy`. Enemies of this class will be moving in rectangular patterns. The constructor takes a `fence` argument in the form of 4-tuple `(tlx,tly,brx,bry)`, representing the top-left corner and the bottom-right corner of the fencing area. Complete the `script()` method below by using the `speed` and `fence` attributes to control the movement of this enemy type.

Hint: this is very similar to one of the examples discussed in the lecture class.

```
class FencingEnemy(MovingEnemy):

    def __init__(self, canvas, size=30, speed=2, fence=(50,50,100,100)):
        super().__init__(canvas, size=size, speed=speed)
        self.fence = fence
        self.set_color("red")

    def script(self):
        x1,y1,x2,y2 = self.fence
        newx = x1
        newy = y1
        while True:
            # Add your code so that the enemy moves in a rectangular pattern
            # whose top-left corner is (x1,y1) and bottom-right corner is (x2,y2).
            # The enemy must also move at the speed defined by the 'speed' attribute.
            if newy == y1 and newx < x2:
                newx += self.speed
            elif newx == x2 and newy < y2:
                newy += self.speed
            elif newy == y2 and newx > x1:
                newx += -self.speed
            elif newx == x1 and newy > y1:
                newy += - self.speed
            newx = min(max(newx, 0), CANVAS_WIDTH)
            newy = min(max(newy, 0), CANVAS_HEIGHT)
            self.move(newx, newy)
            yield
```

Add some enemies of this type to the game by modifying the definition of `enemies` attribute of the main class.

```
class TurtleAdventure(ttk.Frame):

    def __init__(self, parent):
        :
        self.enemies = [
            :
            FencingEnemy(self.canvas, size=20, speed=5, fence=(200,100,500,300)),
            # this guy should be guarding the home
            FencingEnemy(self.canvas, size=20,
                fence=(CANVAS_WIDTH-150, CANVAS_HEIGHT//2-50, CANVAS_WIDTH-50, CANVAS_HEIGHT//2+50)),
        ]
```

8. Creating Your Own Enemy Type

Design your own kind of enemies whose behavior must be different from all the enemy types defined earlier. Explain your design with a complete class implementation in the box below. Don't forget to pick a new color for the new enemies.

```
class TurtleAdventure(ttk.Frame):
    def __init__(self, parent):
        .
        .
        self.enemies = [
            BasicEnemy(self.canvas, size=50, pos=(300,200)),
            BasicEnemy(self.canvas, size=30, pos=(600,300)),
            MovingEnemy(self.canvas, size=30),
            MovingEnemy(self.canvas, size=20, speed=5),
            FencingEnemy(self.canvas, size=20, speed=5, fence=(200,100,500,300)),
            # this guy should be guarding the home
            FencingEnemy(self.canvas, size=20,
                fence=(CANVAS_WIDTH-150, CANVAS_HEIGHT//2-50, CANVAS_WIDTH-50,
CANVAS_HEIGHT//2+50)),
            BounceEnemy(self.canvas, size=25, speed = 10, x_pos=200),
            BounceEnemy(self.canvas, size=25, speed = 10, x_pos=400),
            BounceEnemy(self.canvas, size=25, speed = 10, x_pos=600)
        ]
```

```
class BounceEnemy(MovingEnemy):
    def __init__(self, canvas, size=10, speed=2, x_pos=200, start_move='down'):
        super().__init__(canvas, size=size, speed=speed)
        self.x_pos = x_pos
        self.moving = start_move
        self.set_color("maroon")
    def script(self):
        x = self.x_pos
        newy = 0
        while True:
            if self.moving == 'down':
                newy += self.speed
                if newy >= CANVAS_HEIGHT:
                    self.moving = 'up'
            if self.moving == 'up':
                newy += -self.speed
                if newy <= 0:
                    self.moving = 'down'
            newx = min(max(x, 0), CANVAS_WIDTH)
            newy = min(max(newy, 0), CANVAS_HEIGHT)
            self.move(newx, newy)
            yield
```

Submitting Your Work

- Name your application code `turtle-adventure.py` and submit it along with this labsheet.
- Don't forget to click **Turn In** to complete your submission.