ball.py

Import the class turtle

Create a class called Ball (notice the big B)

- Inside the class
  - Create the **`__init__`** method that takes 6 attributes
    `x`
    `y`
    `dx`
    `dy`
    `r`
    `color`
    Don't forget the self in the beginning

  - Inside the **`__init__`** method
    1. Call the **`__init__`** method of the Turtle Class
    2. Make sure the turtle's pen is not down
    3. Use the `x` and `y` to set the position of the ball
    4. Save the three attributes (`dx`, `dy`, `r`) into self
    5. Set the shape of the ball to be circle
    6. Set the size of the shape to be the r/10 (why divided by 10)
    7. Set the `color` of the ball to the `color` passed in the init

  - Create a method called **`move`** that takes 2 inputs
    1. `screen_width`
    2. `screen_height`
    3. Don't forget the self in the beginning
  - Inside the **`move`** method
    1. Get the x coordinates of the ball and save it inside a variable called `current_x`
    2. Calculate the new `x` coordination (hint current x + the change in x) and save it inside a variable called `new_x`
    3. Do the last 2 steps for y coordination
    4. Get the right side of the ball (hint new x coordination + the radius) and save it inside a variable called `right_side_ball`
    5. Do the last step for the 3 other sides of the ball
    6. Let the ball go to the new x and y coordinates
    7. Check if one of the ball sides will be out of the screen borders, then bounce it back (hint change only the direction in `dx` or/and `dy`)

agario.py

Getting Started

1. First, you will need to import certain python modules that are needed over the course of the lab. Import the following classes:

- `turtle`
- `time`
- `random`
- `Ball` (from `ball.py`)

2. Call `turtle.tracer` with the first argument set to 0 to speed up the drawing of the computer graphics

3. Hide the turtle using `hideturtle` method of the `turtle` class.

4. Add the following global variables to your code:

- boolean RUNNING set to `True`
- float SLEEP set to 0.0077
- SCREEN_WIDTH set to `turtle.getcanvas().winfo_width()/2`
- SCREEN_HEIGHT set to `turtle.getcanvas().winfo_height()/2`

We are now ready to start working on the lab!

Part 0: Creating the Balls

In this section, we will create all the balls that we will be using in the game. First, we will create a ball called `MY_BALL` which will be the ball that we control. Then, we will create randomly create other balls in the game.

1. **Create an new object from the `Ball` class and call it `MY_BALL`.** This is your ball.

2. **Initialize the following variables:**
   - Set `NUMBER_OF_BALLS` to 5.
   - Set `MINIMUM_BALL_RADIUS` to 10.
   - Set `MAXIMUM_BALL_RADIUS` to 100.
   - Set `MINIMUM_BALL_DX` to -5.
   - Set `MAXIMUM_BALL_DX` to 5.

- Set `MINIMUM_BALL_DY` to -5.
- Set `MAXIMUM_BALL_DY` to 5.

3. **Create an empty list and call it `BALLS`.**

4. **Create a `for` loop to iterate through the integers in the range of `NUMBER_OF_BALLS`**
    a. Initialize new variables for the parameters of this new `Ball`. Use `random.randint()` which takes in two numbers as parameters and returns a random integer between those two numbers.
        i. Set `x` to a random integer between `-SCREEN_WIDTH + MAXIMUM_BALL_RADIUS` and `SCREEN_WIDTH - MAXIMUM_BALL_RADIUS`.
        ii. Set `y` to a random integer between `-SCREEN_HEIGHT + MAXIMUM_BALL_RADIUS` and `SCREEN_HEIGHT - MAXIMUM_BALL_RADIUS`.
        iii. Set `dx` to a random integer between `MINIMUM_BALL_DX` and `MAXIMUM_BALL_DX`. Make sure it's not set to `0`, or else the ball won't move!
        iv. Set `dy` to a random integer between `MINIMUM_BALL_DY` and `MAXIMUM_BALL_DY`. Make sure it's not set to `0`, or else the ball won't move!
        v. Set `radius` to a random integer between `MINIMUM_BALL_RADIUS` and `MAXIMUM_BALL_RADIUS`.
        vi. Set `color` to `(random.random(), random.random(), random.random())`.
    b. Create a new `Ball` using the above variables.
    c. Append the new `Ball` to the list `BALLS`.

Part 1: Move All Balls

You will now write code for the function `move_all_balls`. The purpose of this function is to ensure that all balls move. Here is what you will need to do:

1. **Use a `for` loop to iterate through all the balls in the list `BALLS`.** Recall that the syntax for a for-loop is as follows: `for [variable] in [listname]`

2. **For each ball, call its `move` function** which takes two arguments: width and height of the screen.

Part 2: Check for ball collisions

In this part, you will write code for the function `collide` which takes two `Ball` objects as arguments and returns `True` if there is a collision detected between the two balls or `False` if there was no collision. Follow the approach outlined below:

1. **Create a function `collide`** that takes two `Ball` objects as arguments: `ball_a` and `ball_b`

2. **Check if `ball_a` and `ball_b` are the same**. If yes, return `False`

3. **Calculate the distance between the centers of the balls**

4. **Determine whether a collision happened between two balls**:
    a. If the distance between the centers + small number of pixels (10) is less than or equal to the sum of the radiuses of the balls, return `True`
    b. Otherwise, return `False`

Part 3: Check collisions for all balls

Now that you wrote a function to check for a collision between two balls, it is time to write a function called `check_all_balls_collision` to check for a collision between ANY two balls. If such a collision is found, the bigger ball will become bigger and the smaller ball will be moved to a different location with different characteristics.

1. **Create a function called `check_all_balls_collision`** that takes in no arguments

2. **Create a nested for-loop** in which both loops iterate through `BALLS`. Use `ball_a` as the variable name for the first loop and `ball_b` for the second loop

3. If there is a collision between the two balls, **save the radius of each ball in a separate variable**

Before you determine which ball is smaller and gets relocated after the collision, let's define the new characteristics that the relocated ball will have.

4. **Generate random values for the following characteristics.** Make sure to generate values within the proper bounds:

- X coordinate
- Y-coordinate
- X-axis speed *****
- Y-axis speed *****
- Radius
- Color (Use the rgb representation)

***** Note that it is possible to generate 0 as a random number since the minimum axis speeds are negative. Therefore, make sure that these values are not equal to 0. (Hint: Use a while loop)

5. **Determine which ball is smaller.**

- For the smaller ball, set its characteristics to the new characteristics which you just defined.

- For the bigger ball, increase the radius by 1. Don't forget to update the shapesize for both balls.

Part 4: Check collision with my ball

Now that we have a way to detect collisions between any two balls, you will write another function `check_myball_collision` that will detect whether a collision happened between `MY_BALL` and another ball.

1. **Create a function called `check_myball_collision`** that takes in no arguments

2. **Write a for-loop** to iterate through all the balls

3. For each ball, check whether `MY_BALL` collides with it. If yes, **save the radius of each ball in a separate variable**

4. If `MY_BALL` collides with a larger ball, this method returns `False`. Otherwise, follow the same steps to update the characteristics of the smaller ball as in Part 3. Don't forget to update the radius of `MY_BALL`

5. **Finally, return `True`** at the end outside of the for-loop

Part 5: Movearound

Next, we will write a function to handle the mouse moving on the screen. Everytime the mouse moves, an "event" happens. This function is an "event handler" for a mouse moving "event".

1. **Create a function called `movearound`** that takes an argument `event` (Note that `event` corresponds to the mouse moving on the screen)

2. **Move your ball** to the following coordinates:
   - X coordinate: event.x - screen's width
   - Y coordinate: screen's height - event.y

Part 6: The While Loop

We are nearly done! The last thing you have to do is write a while loop that will run as long as the game is running and `MY_BALL` didn't lose. Make sure that the global variable `RUNNING` is defined and set to `True`. Here is the pseudocode:

1. While RUNNING is True:

    2. **Check if the player changed the turtle window size** by making sure the screen width and height are equal to the canvas width and height divided by 2.

    3. If either of the comparisons fail, set both the screen width and screen height equal to canvas width / 2 and canvas height / 2 respectively

    4. Otherwise, **move all the balls** in the game and then check for any collisions between all balls

    5. **Use the `move` method** to move `MY_BALL`. Update the value of `RUNNING` based on whether `MY_BALL` collided with any other balls

    6. **Update the screen** to reflect the changed view of the game

    7. Pause the game using the `sleep` method provided by the `time` module