CREATING A METEOR SHOWER GAME

For this PA, you'll work in teams (see Team Assignments handout). You aren't permitted to receive help from any other team or from anyone else except a CptS 111 TA, the tutoring center, or me. It's also okay for you to use the `pygame` docs available at `https://www.pygame.org/docs/`, and you're free to look for background images if you want to add a background, i.e., you don't want to use the default standard color. *Don't copy any code!* **All team members have to fill out and sign a form affirming that they have followed the rules.**

Note that this is a team effort, and you'll need to put your heads together to complete the PA in time, so be sure to get started early, and decide who will code the different functions. The `main()` function is provided, but you may make changes to it. You'll write six user functions called by `main()`, and you'll need to understand `main()` in order to write them. You'll need to have the first four functions working to some degree as soon as possible. After these are working, you can spend time on the remaining two functions, and then you can go back and refine your entire program. Note that you'll have to play the game a number of times to make sure you detect all possible collisions.

You can send email to your team through Canvas using your team's name. Email addresses are also included in the Team Assignments handout without the `@wsu.edu` which you'll need to add. Don't wait for everyone to reply before starting!

Note: Only use coding concepts we covered in class this semester.

As with previous programming assignments, 25% of your grade will be based on readability, use of comments, and so on (see PA #4 or #5 or the grading rubric for details). *Importantly*, your final grade on the PA will be determined using a weighting factor based on your contribution to the PA as assessed by your teammates. Each team member will assign up to $100 \times (n-1)$ points, where n is the number of members on a team, to their teammates. For example, for a team of four, team members might be assigned points as follows:

Team Member 1: 100, 100 (weight = 1)

Team Member 2: 10, 10, 10 (weight = 0.1)

Team Member 3: 150, 150 (weight = 1.5)

Team Member 4: 90, 80 (weight = 0.85)

In this example, Team Member 2 seems to have been a slacker (they didn't even submit their point assignments which is why they have three scores, but everyone else has two), and Team Member 3 was recognized for doing a lot of the work. If the PA grade is 90%, then final scores for each team member will be as follows:

Team Member 1: 90%

Team Member 2: 0 (because they didn't submit points)

Team Member 3: 125% (nominally 135%)

Team Member 4: 76.5%

The final scores are determined by multiplying the grade (90%) times the weight. However, the maximum score for the PA is 125%. Weights here were determined by adding the points and dividing by 200 or 300, depending on the number of scores.

IMPORTANT: A 5% penalty will be applied to a student's score if they don't submit the team evaluations by 11:59 pm, on Saturday, 4.29.23.

Header information. Your program must include the following information in the header.

- Names of team members
- Date
- Programming Assignment #7
- Name of program
- Brief description and/or purpose of the program; include sources

Submission information. Submit a zipped file called **`<first_initial+lastname>_pa7.zip`** in Canvas. **Only one person per team may submit the program.** Decide between yourselves who that person is, and be sure everyone's name is in the program header. Include in the zipped file everyone's signed and dated Affirmation Statement (another handout).

Program Requirements

Your strategy for completing this PA should be as follows:

- **Determine how `main()` works; this is very important! You'll need to adapt one of the lines of code in `main()` to use in one of the functions.**
- Code basic implementations of the six user functions until your entire program works, i.e., get it all to run first!
- Decide how you want to modify your functions and perhaps the `main()` function to improve your program
- Modify one function at a time, checking each time to make sure you didn't break something, i.e., that your program still runs

Before you can start writing your program, you'll need to download the `pygame` module which isn't part of the standard Python library. You'll do this using the `pip` command. Unfortunately, which version of `pip` you need to use depends on your operating system and the version of Python you're using. First try using `pip`. If it doesn't work, try `pip3`.

> **Windows:** Search for `cmd` or `PowerShell`. You can also use the Terminal app if you have it. Start this program, and after it has started issue the following command:
>
> ```
> pip install pygame
> ```
>
> Hopefully, you won't be prompted to reinstall Python again. If you are, try using `pip install Python`.

macOS: In a Finder window, go to the Applications folder and scroll down to the Utilities folder. In the Utilities folder, launch the Terminal app. At the prompt, issue the following command:

```
pip install pygame
```

After you've downloaded the **pygame** module, the first line of your code should be:

```
import pygame as pyg
```

You also need to import the **random** module which is in the standard Python library. You may use whichever import statement you like.

You should understand the `main()` function thoroughly before starting to code the six functions described below.

- **`set_speed()`:** A non-void function with one parameter, an integer score, which returns the speed. The speed should be based on the score: the higher the score, the faster the speed should be. However, don't set the speed to the score! Initially the score is zero, and if the speed is zero, the meteors won't fall. Also, the speed is the number of pixels that each meteor "falls" at each update (updates occur about 30 times per second!), so if you set the speed to the score later, the meteor will move too fast for you to see it.
- **`draw_meteors()`:** A void function with four parameters: the nested list of meteor positions (initially empty), the size of the meteors, the game screen, and the color of the meteors. A meteor should be drawn at each meteor position using the `draw.rect()` function in `pygame`.
- **`drop_meteors()`:** A void function with three parameters: the nested list of meteor positions, the size of the meteors, and the width of the game screen. The meteors should be placed at the top of the screen (y position fixed at zero) at random positions along the width of the screen (x positions random, but meteors can't overlap each other). Each time a meteor is placed, its location should be appended to the list of meteor positions. Finally, meteors should fall at random times. You should consider using the `random()` and `randint()` or `random()` and `randrange()` functions in the `random` module when coding this function. Note that this function doesn't actually drop the meteors, but it positions meteors along the top of the game screen so they can fall when their positions are updated by `update_meteor_position()`. However, to code this function correctly, you have to think of them falling at random times and at random positions along the x-axis. Otherwise, you'll end up with a sheet of meteors falling at all times. Remember that the game screen is updated about 30 times per second which is fast!
- **`update_meteor_position()`:** A non-void function with four parameters: the nested list of meteor positions, the height of the game screen, the score, and the speed of the meteors; it returns the integer score. For each meteor still in play, its y position should be increased by the meteor speed (pixels moved). Each meteor hitting the ground should be removed from the nested list and the user's score increased by one. Note that a meteor is either still in play or it has hit the ground.
- **`detect_collision()`:** A non-void function with four parameters: the position (a list) of the player, the position (a list) of the meteor, the size of the player, and the size of the meteor.

This function is called by `collision_check()` and returns `True` if the meteor has hit the player or `False` if it hasn't. Think carefully about how to check to determine whether or not the meteor has hit the player. Drawing some sketches will help. **Note that the positions of the meteor and the player are measured from their top left corners (0,0). Note also that this function considers one meteor at a time; it only takes one meteor hitting the player for the player to die.**

- **`collision_check()`:** A non-void function with four parameters: the nested list of meteor positions, the position (a list) of the player, the size of the player, and the size of a meteor. This function calls `detect_collision()` *for each meteor*, and if (and as soon as) a collision is detected, it returns `True` to `main()`. Otherwise, it returns `False` and the game continues.

No output examples are provided because the output is a game. You'll know how your program is working (or not working) based on your game screen.
