Python Coursework Script (Code Review)

Kartik: The basic idea with the code is to create an extremely small version of a 2d game engine. This means that we have all of the game entities as objects and these objects are rendered on to a screen at a specific interval. The interval determines the FPS, or frames per second that the game runs at.

Helper functions:

Kartik: These two functions are just there to make pixel tuple generation easy, one of them generates a completely random color pixel while the other is used to generate a pixel of the given color

Helper classes:

Firas: As Kartik mentioned before all the game entities are objects so these classes are used to create those objects. The Window class is used to create our GUI (using Python's inbuilt GUI library - Tkinter). It has a lot of code to get the fluid grid working but it's not really important to the project. The Screen class is used to abstract away opc.py's methods so that we have more readable and less error-prone code. And finally the Point class which is just used to create a very simple 2d Point. It has an 'x' and a 'y' and then we overwrite pythons `__hash__` and `__eq__` functions so that we can check if two points are the same just by comparing them directly. So if two points have the same x and y values they would be `==` to each other in comparisons.

The `Snake` class:

Kartik: The snake class holds a bulk of the logic of the game. It has a body (which is an array of Points), a direction (which is used to move it) and a grow boolean which is used to determine whether the Snake should be growing at the current turn.

It's `show()` function is used to add it to the screen and all it does is loop through the Snake's body and add each Point to the Screen.

The `eat()` function is used when the Snake's head collides with a piece of food. All it does is remove the food from the screen and set the Snake's grow boolean to True

Then we have `set_direction()` which is used to set the Snake's direction. The code for this is pretty long because we have to make sure that the snake can't move backwards. So for example if the Snake's direction was 'right' and `set_direction()` was called with 'left' it would ignore it.

Finally we have the `move()` function which moves the Snake on the screen. So first of all we create a copy of our Snake, we have to use deepcopy because our Snake's body is an array of objects and Python would make a shallow copy by default. Then we take the copies head and move it based on whatever direction the Snake is going. These are just some if statements to make sure that when we reach the end of the screen we come back on the other side. Then if the Snake is not supposed to grow we remove its last element using `pop()` on its body. Then add the new head to the front of our Snake and then reset the grow boolean.

The `game_over_screen()` function:

Kartik: The `game_over_screen()` function is used to render out the game over animation. All we do is read the hardcoded game over screen from a text file. As you can see all this text file contains is a hard coded game over screen that I generated previously using some image processing. This gives us the "Game Over" letters but then in the function we change the sides to be random pixel every frame so it looks more interesting.

The `update()` function:

Firas: As you can see now we declare instances of all the classes that we created previously. We also declare an Apple variable which is just an instance of our Point class but you will see later how it is used. Also the frame rate has been set to 10 FPS.

The entire game is taking place within `update()` and its our game loop. So what we do first is set the game over boolean to False. Then we reset the screen array (the array that holds all the pixels to black) this hasn't been rendered to the display yet so the user won't see that. Then we check if the game is actually over by comparing the length of the Snake's body to the length of the Snake's body as a set. The `set()` function in Python just returns the array you provide it with no duplicates. This works because if we collide with ourselves in game we'll have 2 Points in the Snake's body that have the same 'x' and 'y' position. Which would lead to the set version of Snake's body being smaller than the actual body.

So if the game over boolean is true we just render out a death animation where all the pixels are random. And then call `update()` recursively at the decided frame rate. The `window.root.after` is a way to tell the Tkinter window to call the provided function at an interval.

Then if the game isn't over we use the Snake's `show()` function to add it to the screen array. And then we also add the Apple to the screen as well, just using it's 'x' and 'y' position as index values to change the screen array.

We then quickly perform a check to see if the Snake has eaten the Apple, this is done by simply checking if the Snake's head is at the same point as the Apple. Since we overwrote our Point classes comparison function we can just use `==` to check if they are at the same location. If they are then we call the Snake's eat function and pass the Apple to it, we also have to generate a new random spot for our Apple, we accomplish this by using `randint()` to get a new location and then we check if the new location is in the Snake's body. if it is we generate another one and we keep doing this until we get a location that isn't in our Snake's body.

Once all of this is done all that's left to do is render out our screen array onto to the display (the list comprehension there is just to flatten the 2d array into 1d because opc.py is expecting a 1d array). Then we call the Snake's move function and finally call `update()` recursively.

Since we haven't actually called our `update()` function yet that's what we do in the end just as an initial call and then we call our Tkinter windows `mainloop()` method so that the window is shown to the user.