Assignment: A6

Air-track: GUI controls

New physics calculation concepts:

- Changing the gravitational force during runtime.

Python language topics:

- Use of the GUI features of the pgu module.

Problem statement:

(Again, start from a copy of your previous assignment file.)

Review the code for the "gui10.py" example in the pgu module's example folder. By imitating this example, add a GUI interface for the A5 assignment. The GUI interface should include a slider control for gravity and checkboxes for color transfer and collision stickiness. Then also add a button control for the motion freeze. Almost all of the pgu examples have sample code for buttons. Use "F2" key to toggle the GUI on and off.

Note: The GUI controls should not interfere with the cars. So when you interact with a control this should not inadvertently trigger a car selection. This distinction is enforced in the checkForCarAtThisPosition function that was defined in previous assignments.

Algorithmic description:

- Organize GUI controls in a class.
- Instantiate controls and supporting GUI objects.
- Feed the GUI events from Pygame while in the game loop.
- Read the state of the GUI controls and update corresponding object attributes. Do this after all user input has been collected and before the physic calculations change the velocity and position of the cars.
- Paint the GUI on the screen.

Python code: (see images on next few pages)

The following code is not a complete solution to the problem. It shows changes (additional content) relative to assignment #5. There is no obfuscation, but you have to figure out where these pieces of code should go. The indent levels should be a clue to you. These are in order, but of course the neighboring images are not necessarily a continuation from the image above.

```
# gui from Phil's PyGame Utilities
from pgu import gui
```

```
class TrackGuiControls( gui.Table):
    def __init__(self, **params):
        gui.Table.__init__(self, **params)

        text_color = THECOLORS["yellow"]  #(0, 0, 255)

        # Make a table row in the gui (like a row in HTML).
        self.tr()

        # Color transfer.
        self.td( gui.Label(" Color Transfer (c): ", color=text_color), align=1)
        self.td( gui.Switch(value=False, name='colorTransfer'))

        # Stickiness.
        self.td( gui.Label("     Fix stickiness (s): ", color=text_color), align=1)
        self.td( gui.Switch(value=True, name='fix_Stickiness'))

        # Gravity.
        self.td( gui.Label("     Gravity (g): ", color=text_color))
        self.td( gui.HSlider(0,-3,3, size=20, width=100, height=16, name='gravity_factor'))

        # Freeze the cars.
        self.td( gui.Label("     Freeze (f): ", color=text_color))
        # Form element (freeze_button).
        freeze_button = gui.Button("v=0")
        # Note: must invoke the method to be called WITHOUT parentheses.
        freeze_button.connect( gui.CLICK, self.stop_cars)
        self.td( freeze_button)

        # Just a help tip for starting a new demo.
        self.td( gui.Label("         New demo (1-3).", color=THECOLORS["green"]))

    # The method that's called by the button must be defined here in this class.
    def stop_cars(self):
        air_track.stop_the_cars()

    # Set air_track attributes based on the values returned from the gui.
    def queryIt(self):
        # Color transfer.
        air_track.color_transfer = gui_form['colorTransfer'].value

        # Stickiness
        air_track.fix_wall_stickiness = gui_form['fix_Stickiness'].value
        air_track.fix_car_stickiness = air_track.fix_wall_stickiness

        # Gravity
        air_track.g_mps2 = air_track.gbase_mps2 * (gui_form['gravity_factor'].value/2.0)
```

```
class AirTrack:
    def __init__(self):
        self.clean()
        self.gui_menu = True

    def clean(self):
        # Initialize the list of cars.
        self.cars = []
        self.carCount = 0

        # Coefficients of restitution.
        self.coef_rest_base = 0.95  # Useful for reseting things.
        self.coef_rest_car = self.coef_rest_base
        self.coef_rest_wall = self.coef_rest_base

        # Component of gravity along the length of the track.
        self.g_toggle = False
        self.gbase_mps2 = 9.8/40.0 # one 40th of g.

        self.color_transfer = False

        self.collision_count = 0

        self.fix_wall_stickiness = True
        self.fix_car_stickiness = True
```

```
    def stop_the_cars(self):
        for car in self.cars:
            car.v_mps = 0

    def make_some_cars(self, nmode):
        # Update the caption at the top of the pygame window frame.
        game_window.update_caption("Air Track (basic): Demo #" + str(nmode))

        # Scrub off the old cars and reset some stuff.
        air_track.clean()

        if (nmode == 1):
            gui_form['gravity_factor'].value = 0.0
            gui_form['colorTransfer'].value = False

            self.cars.append( Detroit(color=THECOLORS["yellow" ], left_px = 240, width_px=20, v_mps=  0.0))
            self.cars.append( Detroit(color=THECOLORS["orange"],  left_px = 340, width_px=30, v_mps=  0.0))

        elif (nmode == 2):
            gui_form['gravity_factor'].value = 2.0
            gui_form['colorTransfer'].value = False

            self.cars.append( Detroit(color=THECOLORS["yellow" ], left_px = 240, width_px=20, v_mps= -0.1))
            self.cars.append( Detroit(color=THECOLORS["orange"],  left_px = 440, width_px=60, v_mps= -0.2))

        elif (nmode == 3):
            gui_form['gravity_factor'].value = -1.0
            gui_form['colorTransfer'].value = True

            self.cars.append( Detroit(color=THECOLORS["yellow" ], left_px = 440, width_px=20, v_mps= -0.1))
            self.cars.append( Detroit(color=THECOLORS["orange"],  left_px = 540, width_px=80, v_mps= -0.2))
```

```
            elif (event.key==K_s):
                gui_form['fix_Stickiness'].value = not gui_form['fix_Stickiness'].value

            elif (event.key==K_c):
                gui_form['colorTransfer'].value = not gui_form['colorTransfer'].value

            elif (event.key==K_f):
                air_track.stop_the_cars()

            elif (event.key==K_g):
                air_track.g_toggle = not air_track.g_toggle
                if air_track.g_toggle:
                    gui_form['gravity_factor'].value = -2
                else:
                    gui_form['gravity_factor'].value = 0

            elif (event.key==K_F2):
                # Turn the gui menu on/off
                air_track.gui_menu = not air_track.gui_menu
```

```
        # In all cases, pass the "event" to the Gui application.
        gui_application.event( event)
```

```
        # Set object attributes based on the values returned from a query of the Gui.
        # Important to do this AFTER all initialization and car building is over, AFTER
        # any user input from the pygame event queue, and BEFORE the updates to the speed
        # and position.
        env.gui_controls.queryIt()
```

```
# A few globals.
global env, game_window, air_track, gui_form, gui_application
```

```
# Initialize gui...
gui_form = gui.Form()
env.gui_controls = TrackGuiControls()
```

```
        # Paint the gui. (F2 toggles gui on/off)
        if air_track.gui_menu:
            gui_application.paint()
```