Assignment: A7

Air-Track: Hollow Cars and Demos

New physics calculation concepts:

- Variation in mass between cars is represented by hollowing out the lighter cars. Keeping the width of each car identical results in interesting collision patterns.

Python language topics:

- Nothing much new here.

Problem statement:

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

- Alter the Detroit (car) class to support hollow cars. Instead of specifying a pixel width, input the car mass. Specify hollow rendering with a T/F car attribute named "hollow."
- Use the inflate method of Pygame's rectangle object to draw the smaller rectangle that will represent the hole. Keep the width of the hole constant and adjust the hole area only through its height.
- Show off your stuff by making a set of demos (The following serves mainly as an explanation of the 10 new demos in this assignment's code. Skip some or all of these and make your own if that's more fun for you. Try to make at least 10 distinct demos.):
    - Demo **#1 (**#1 on the keyboard; above the "Q"): Closely-spaced set of cars falling under gravity with slightly inelastic collisions. Have all the cars the same color except for one. When these "settle" at one end of the track illustrate the collisions by turning the color-transfer control on and off.
    - Demo **#2 and #3**: Collisions between a pair of closely-spaced cars falling where the lighter one is higher on the track; slightly inelastic. The more the difference in mass, the stronger the lighter car rebounds when the pair hits the end of the track. Have one case where the masses are slightly different in mass (8 to 6) and one case where the cars are strongly different in mass (8 to 1).
    - Demo **#4**: Completely inelastic collision between a set of converging (all headed toward the center of the track) cars where the total system momentum is zero. The cars should all stick together in the middle of the track and the glob of cars should have zero speed (to prove the system initially had zero momentum).
    - Demo **#5**: Closely-spaced set of stationary cars. One moving car with a different color. Elastic collisions.
    - Demo **#6**: Same as #5 but with some gravity.
    - Demo **#7**: Same as #5 but with a linear variation in the mass of each car in the set.
    - Demo **#8**: Elastic collisions between two cars. The heavier car is initially stationary and 3 times the mass of the lighter car. Color transfer is set to be off. The resulting pattern shows a complete transfer of energy into and out of the heavier car. Another interesting feature is that when both cars are in motion, they have the same speed.
    - Demo **#9**: This is like #5 but a bit more dramatic and illustrates momentum moving through the stationary set in two directions. Make two stationary sets (12 cars in each set) of closely-spaced cars, all the same color, with a significant gap between the two sets. Gravity is off; totally elastic collisions. Then have two cars, each a different color, moving toward the sets from the ends of the track, and two cars moving from the central gap out toward the ends of the track. Have color transfer on.

- Demo **#0**: Similar to #5, but with closer spacing in the set and three moving cars. Elastic collisions. Initially no color transfer. This is kind of like a Newton's cradle with three moving balls.
- Demo **#1 (kp)** (#1 on the keypad; underneath #4): This is the former demo #1 from the previous assignment. This illustrates car rendering without the "hollow" technique. Two stationary cars.
- Demo **#2 (kp)**: Two non-hollow cars with contrasting masses.
- Demo **#3 (kp)**: Two non-hollow cars with even more contrasting masses.
- More demos: It's your turn. Come up with something new for each of the remaining number keys on the keypad.

Algorithmic description:

- Implement a new attribute "hollow" for the Detroit (car) class to support shifting gears from hollow to non-hollow cars. Do one or the other in each demo, not both types.
- As each car is added, keep track of the mass value of the heaviest car. Make this an attribute of the air_track object.
- Then update, if needed, the characteristics of the hole on each car relative to the most massive car. The most massive cars(s) will be solid (no hole).
- During the game loop draw both a regular rectangle and then a smaller rectangle (on top) to represent the hole. The hole's color should be the same as the background color (probably black).
    - Use the inflate method (with negative dimensions) of the rectangle object to make a smaller copy of the main rectangle.

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 #6. 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. There may be small pieces of code missing; it's up to you to fill in those blanks.

```python
class Detroit:
    def __init__(self, color=THECOLORS["white"], left_px=10, width_px=26, height_px=98, hollow=True, m_kg=None, v_mps=1):

        self.color = color
        self.hollow = hollow

        self.height_px = height_px
        self.top_px    = game_window.height_px - self.height_px
        self.width_px  = width_px

        # Use y midpoint for drawing the cursor line.
        self.center_y_px = int(round( float(game_window.height_px - self.height_px) + float(self.height_px)/2.0) )
        # For use with cursor-tethers selection.
        self.selected = False

        self.width_m = env.m_from_px(width_px)
        self.halfwidth_m = self.width_m/2.0

        self.height_m = env.m_from_px(width_px)

        # Initialize the position and velocity of the car. These are affected by the
        # physics calcs in the Track.
        self.center_m = env.m_from_px(left_px) + self.halfwidth_m
        self.v_mps = v_mps

        # Aggregate type forces acting on car.
        self.cursorString_spring_force_N = 0
        self.cursorString_carDrag_force_N = 0

        # Calculate the mass of the car.
        if self.hollow:
            self.m_kg = m_kg
        else:
            self.density_kgpm2 = 600.0
            self.m_kg = self.height_m * self.width_m * self.density_kgpm2

        # Increment the car count. The air_track object has global scope.
        air_track.carCount += 1
        # Name this car based on this air_track attribute.
        self.name = air_track.carCount

        # Update the list of masses and recalculate the maximum.
        air_track.mass_list.append(self.m_kg)
        air_track.max_m_kg = max(air_track.mass_list)

        # Create a rectangle object based on these dimensions
        # Left: distance from the left edge of the screen in px.
        # Top:  distance from the top  edge of the screen in px.
        self.rect = pygame.Rect(left_px, self.top_px, self.width_px, self.height_px)
        # Calculate the hole characteristics (shrink values).
        if self.hollow:
            if (self.m_kg == air_track.max_m_kg):
                # If you're the top dog (heaviest), everyone else will have to re-calculate their shrink
                # values accordingly.
                for eachcar in air_track.cars:
                    eachcar.calc_hole_shrink()
            else:
                # Oh well, not the top dog. Then I'm the only one that needs to calculate
                # shrink values.
                self.calc_hole_shrink()

    def calc_hole_shrink(self):
        # Calculate a special density in kg per pixel area. Use this only for the
        # hole calculation. Notice the reference to the air_track's max_m_kg which is
        # the mass of the heaviest car.

        self.density_kgppx2 = float(air_track.max_m_kg)/float(self.width_px * self.height_px)

        # Keep the hole width consistent for all the cars.
        hole_width_pxi = self.width_px - 2

        # Calculate the hole height based on the difference in mass it represents.
        hole_height_pxf = (air_track.max_m_kg - self.m_kg)/(self.density_kgppx2 * hole_width_pxi)
        hole_height_pxi = int(round(hole_height_pxf))

        # These shrink values will be used (relative to the main car rectangle) when time comes
        # to draw the rectangle that represents the hole.
        self.shrink_x_px = self.width_px - hole_width_pxi
        self.shrink_y_px = self.height_px - hole_height_pxi

    def draw_car(self):
        # Update the pixel position of the car's rectangle object to match the value
        # controlled by the physics calculations.
        self.rect.centerx = env.px_from_m( self.center_m)

        # Draw the main rectangle.
        pygame.draw.rect(game_window.surface, self.color, self.rect)

        if self.hollow and (self.m_kg <> air_track.max_m_kg):

            # Draw a subrectangle (a hole) to illustrate the mass of this car relative
            # to the mass of the most massive car. The closer we are to the most massive car
            # the more shrinking of the hole. So heavier cars look more
            # solid.

            # Make a hole by shrinking the main rectangle.
            hole_rect = self.rect.inflate(-self.shrink_x_px, -self.shrink_y_px)  # x,y

            # Draw the hole.
            pygame.draw.rect(game_window.surface, THECOLORS["black"], hole_rect)
    def make_some_cars(self, nmode):
        # Update the caption at the top of the pygame window frame.
        game_window.update_caption("Air Track (hollow cars): Demo #" + str(nmode))

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

        if nmode == '1p':
            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, hollow=False, v_mps=  0.0))
            self.cars.append( Detroit(color=THECOLORS["orange"],  left_px = 340, width_px=30, hollow=False, v_mps= -0.0))

        elif nmode == '2p':
            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, hollow=False, v_mps= -0.1))
            self.cars.append( Detroit(color=THECOLORS["orange"],  left_px = 440, width_px=60, hollow=False, v_mps= -0.2))

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

            self.cars.append( Detroit(color=THECOLORS["yellow" ], left_px = 240, width_px=20, hollow=False, v_mps= -0.1))
            self.cars.append( Detroit(color=THECOLORS["orange"],  left_px = 440, width_px=80, hollow=False, v_mps= -0.2))

        elif nmode == 0:
            gui_form['gravity_factor'].value = 0
            gui_form['colorTransfer'].value = False

            self.coef_rest_car  = 1.00
            self.coef_rest_wall = 1.00
            x_steps = Next_x( 100, 29)
            cars_v_mps = 0.0 #.15 #m/s
            cars_m_kg =  0.3 #kg
            color_list = ["yellow","red","green","blue","pink"]
            k_color = 0
            for j in range(9):
                if (k_color > (len(color_list) - 1)):
                    k_color = 0
                else:
                    k_color += 0
                self.cars.append( Detroit(color=THECOLORS[color_list[k_color]], left_px=x_steps.step(),
                                  v_mps=cars_v_mps, m_kg=cars_m_kg))

            cars_v_mps = -0.3 #.15 #m/s
            x_steps = Next_x( 750, 45)
            k_color = 0
            for j in range(3):
                if (k_color > (len(color_list) - 1)):
                    k_color = 2
                else:
                    k_color += 1
                self.cars.append( Detroit(color=THECOLORS[color_list[k_color]], left_px=x_steps.step(),
                                  v_mps=cars_v_mps, m_kg=cars_m_kg))

        elif nmode == 1:
            gui_form['gravity_factor'].value = 2
            gui_form['colorTransfer'].value = True

            self.coef_rest_car  = 0.95
            self.coef_rest_wall = 0.95

            color_list = ["yellow","red","green","blue","pink"]
            x_steps = Next_x( 050, 30)
            cars_v_mps = 0.0 #.15 #m/s
            cars_m_kg =  0.3 #kg
            k_color = 0
            for j in range(4):
                if (k_color > (len(color_list) - 1)):
                    k_color = 0
                self.cars.append( Detroit(color=THECOLORS[color_list[k_color]], left_px=x_steps.step(),
                                  v_mps=cars_v_mps, m_kg=cars_m_kg))
                k_color += 0
            k_color += 1
            self.cars.append( Detroit(color=THECOLORS[color_list[k_color]], left_px=x_steps.step(),
                              v_mps=cars_v_mps, m_kg=cars_m_kg))

        elif nmode == 2:
            gui_form['gravity_factor'].value = 3
            gui_form['colorTransfer'].value = False

            x_steps = Next_x( 450, 30)
            cars_v_mps = 0.0 #.15 #m/s
            cars_m_kg =  0.3 #kg

            self.cars.append( Detroit(color=THECOLORS["white"], left_px=x_steps.step(), v_mps=cars_v_mps, m_kg=cars_m_kg*6))
            self.cars.append( Detroit(color=THECOLORS["white"], left_px=x_steps.step(), v_mps=cars_v_mps, m_kg=cars_m_kg*8))

        elif nmode == 3:
            gui_form['gravity_factor'].value = 3
            gui_form['colorTransfer'].value = False

            x_steps = Next_x( 450, 30)
            cars_v_mps = 0.0 #.15 #m/s
            cars_m_kg =  0.3 #kg

            self.cars.append( Detroit(color=THECOLORS["white"], left_px=x_steps.step(), v_mps=cars_v_mps, m_kg=cars_m_kg*1))
            self.cars.append( Detroit(color=THECOLORS["white"], left_px=x_steps.step(), v_mps=cars_v_mps, m_kg=cars_m_kg*8))

        elif nmode == 4:
            gui_form['colorTransfer'].value = False

            self.coef_rest_car  = 0.0
            self.coef_rest_wall = 1.0

            gui_form['gravity_factor'].value = 0

            cars_v_mps = 0.1 #.15 #m/s
            cars_m_kg =  0.3 #kg

            self.cars.append( Detroit(color=THECOLORS["white"], left_px= 30, v_mps=+7.0*cars_v_mps, m_kg=2*cars_m_kg))
            self.cars.append( Detroit(color=THECOLORS["white"], left_px=500, v_mps=+2.0*cars_v_mps, m_kg=1*cars_m_kg))
            self.cars.append( Detroit(color=THECOLORS["white"], left_px=600, v_mps=-2.0*cars_v_mps, m_kg=2*cars_m_kg))
            self.cars.append( Detroit(color=THECOLORS["white"], left_px=700, v_mps=-1.0*cars_v_mps, m_kg=1*cars_m_kg))
            self.cars.append( Detroit(color=THECOLORS["white"], left_px=900, v_mps=-5.5*cars_v_mps, m_kg=2*cars_m_kg))

        elif nmode == 5:
            self.coef_rest_car  = 1
            self.coef_rest_wall = 1

            gui_form['gravity_factor'].value = 0
            gui_form['colorTransfer'].value = True

            x_steps = Next_x(450, 35)
            cars_v_mps = 0.0 #.15 #m/s
            cars_m_kg =  0.3 #kg

            for j in range(10):
                self.cars.append( Detroit(color=THECOLORS["yellow"],
                                             left_px=x_steps.step(),
                                             v_mps=cars_v_mps,
                                             m_kg=cars_m_kg))
            self.cars.append( Detroit(color=THECOLORS["red"], left_px=x_steps.step(), v_mps=0.5, m_kg=cars_m_kg))

        elif nmode == 6:
            self.coef_rest_car  = 1
            self.coef_rest_wall = 1

            gui_form['gravity_factor'].value = -1
            gui_form['colorTransfer'].value = True

            x_steps = Next_x(450, 35)
            cars_v_mps = 0.1
            cars_m_kg =  0.3 #kg

            for j in range(10):
                self.cars.append( Detroit(color=THECOLORS["yellow"],
                                             left_px=x_steps.step(),
                                             v_mps=cars_v_mps,
                                             m_kg=cars_m_kg))
            self.cars.append( Detroit(color=THECOLORS["red"], left_px=x_steps.step(), v_mps=0.5, m_kg=cars_m_kg))

        elif nmode == 7:
            self.coef_rest_car  = 1 #0.99
            self.coef_rest_wall = 1 #0.99
            gui_form['gravity_factor'].value = 0

            gui_form['colorTransfer'].value = True

            x_steps = Next_x(450, 35)
            cars_v_mps = 0.0 #.15 #m/s
            cars_m_kg =  0.3 #kg

            for j in range(10):
                cars_m_kg += .1
                self.cars.append( Detroit(color=THECOLORS["yellow"],
                                             left_px=x_steps.step(),
                                             v_mps=cars_v_mps,
                                             m_kg=cars_m_kg))
            cars_m_kg += .1
            self.cars.append( Detroit(color=THECOLORS["red"], left_px=x_steps.step(), v_mps=0.5, m_kg=cars_m_kg))

        elif nmode == 8:
            gui_form['colorTransfer'].value = False

            self.coef_rest_car  = 1.000
            self.coef_rest_wall = 1.000

            gui_form['gravity_factor'].value = 0

            cars_v_mps = 0.0 #.15 #m/s
            cars_m_kg =  0.3 #kg

            # This does interesting COMPLETE energy transfers between the cars when the first car is
            # 3,4,5,6... times the mass of the other car, and the lighter car is initially stationary.
            # If the heavy car is initially stationary, then only 3x works.
            self.cars.append( Detroit(color=THECOLORS["yellow"], left_px=200, v_mps=0, m_kg=3*cars_m_kg))
            self.cars.append( Detroit(color=THECOLORS["red"],    left_px=500, v_mps=1, m_kg=1*cars_m_kg))

        elif nmode == 9:
            #self.gui_menu = True #False
            gui_form['gravity_factor'].value = 0
            gui_form['colorTransfer'].value = True

            self.coef_rest_car  = 1.00
            self.coef_rest_wall = 1.00

            cars_v_mps = 0.0 #.15 #m/s
            cars_m_kg =  0.3 #kg
            color_list = ["yellow","red","green","blue","pink","grey"]

            self.cars.append( Detroit(color=THECOLORS[color_list[1]], left_px= 20, v_mps= 0.05, m_kg=cars_m_kg))
            x_steps = Next_x( 50, 29)
            for j in range(12):
                self.cars.append( Detroit(color=THECOLORS[color_list[0]], left_px=x_steps.step(), v_mps=cars_v_mps, m_kg=cars_m_kg))
            self.cars.append( Detroit(color=THECOLORS[color_list[2]], left_px= 430, v_mps=-0.05, m_kg=cars_m_kg))
```

```
            self.cars.append( Detroit(color=THECOLORS[color_list[3]], left_px= 460, v_mps= 0.20, m_kg=cars_m_kg))
            x_steps = Next_x( 500, 29)
            for j in range(12):
                self.cars.append( Detroit(color=THECOLORS[color_list[0]], left_px=x_steps.step(), v_mps=cars_v_mps, m_kg=cars_m_kg))
            self.cars.append( Detroit(color=THECOLORS[color_list[5]], left_px=890, v_mps=-0.05, m_kg=cars_m_kg))

        else:
            print "nothing set up for this key"


class Next_x:
    # Initialize the positions of the cars.
    def __init__(self, x_start, x_increment):
        self.x = x_start
        self.dx = x_increment
    def step(self):
        self.x += self.dx
        return self.x
```