Assignment: A5

Air-Track: Cursor Tethers

New physics calculation concepts:

- Spring and drag type forces applied to a selected car.

Python language topics:

- A dictionary.
- A dictionary of dictionaries.
- Concatenating strings.

Problem statement:

(Again, start with a new Python file.)

Add algorithmic content to the previous exercise to enable cursor-tether client interactions with the car objects.

Algorithmic description:

Create a Client class. (This class will eventually be used for multi-client games.) Use the client object to store mouse data (as Client attributes) as it is extracted from the Pygame event queue. In the Client methods that follow, determine the mouse state by referring to the Client attributes.

Add a "calc_tether_forces_on_cars" method to the Client class that does the following:

- If a car is not already selected, check to see if a car is under the cursor and then select it; if the car is selected but the mouse button is up, un-select it.
- If the car is selected and the mouse button is down, calculate the forces on the car, based on which of the three buttons is depressed, and the physics-world separation between the cursor and the car in the x-direction.

Add a second method to the Client class that draws the cursor tether (a line).

In the while loop, use the "calc_tether_forces_on_cars" method to calculate the forces applied by each client on the cars. (There is only one client at this point.) Do this BEFORE you update the velocity and position of the cars.

AFTER you draw the cars, draw the cursor tether for each client that has selected a car.

Python code: (see images on next few pages)

The following code (image) is not a complete solution to the problem. It shows changes (additional content) relative to assignment #4. There is no obfuscation this time, but you have to figure out where these pieces of code should go. The indent levels should be a clue to you. These are not necessarily in order, so of course the neighboring images are not necessarily a continuation from the image above.

```
class Client:
    def __init__(self, cursorString_color):
        self.cursor_location_px = (0,0)   # x_px, y_px
        self.mouse_button = 1             # 1, 2, or 3
        self.buttonIsStillDown = False

        self.cursorString_color = cursorString_color

        self.selected_car = None

        # Define the nature of the cursor strings, one for each mouse button.
        self.mouse_strings = {'string1':{'c_drag':   2.0, 'k_Npm':   60.0},
                              'string2':{'c_drag':   0.2, 'k_Npm':    2.0},
                              'string3':{'c_drag':  20.0, 'k_Npm': 1000.0}}

    def calc_tether_forces_on_cars(self):
        # Calculated the string forces on the selected car and add to the aggregate
        # that is stored in the car object.

        # Only check for a selected car if one isn't already selected. This keeps
        # the car from unselecting if cursor is dragged off the car!
        if (self.selected_car == None):
            if self.buttonIsStillDown:
                self.selected_car = air_track.checkForCarAtThisPosition(self.cursor_location_px)

        # If a car is selected
        else:
            if not self.buttonIsStillDown:
                # Unselect the car and bomb out of here.
                self.selected_car.selected = False
                self.selected_car = None
                return None

            # If button is down, calculate the forces on the car.
            else:
                # Use dx difference to calculate the hooks law force being applied by the tether line.
                # If you release the mouse button after a drag it will fling the car.
                # This tether force will diminish as the car gets closer to the mouse point.
                dx_m = env.m_from_px( self.cursor_location_px[0]) - self.selected_car.center_m

                stringName = "string" + str(self.mouse_button)
                self.selected_car.cursorString_spring_force_N  += dx_m * self.mouse_strings[stringName]['k_Npm']
                self.selected_car.cursorString_carDrag_force_N += (self.selected_car.v_mps *
                                                                   (-1) * self.mouse_strings[stringName]['c_drag'])

    def draw_cursor_string(self):
        car_center_xy_px = (env.px_from_m(self.selected_car.center_m), self.selected_car.center_y_px)
        pygame.draw.line(game_window.surface, self.cursorString_color, car_center_xy_px, self.cursor_location_px, 1)
```

```
    def checkForCarAtThisPosition(self, cursor_location_xy):
        x_px = cursor_location_xy[0]
        y_px = cursor_location_xy[1]
        x_m = env.m_from_px(x_px)
        for car in self.cars:
            if (((x_m > car.center_m - car.halfwidth_m) and (x_m < car.center_m + car.halfwidth_m)) and
                (y_px > game_window.height_px - car.height_px)):
                car.selected = True
                return car
        return None
```

```
        # 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
```

```
        # Reset the aggregate forces.
        car.cursorString_spring_force_N = 0
        car.cursorString_carDrag_force_N = 0
```

```
        # Calculate client related forces.
        for client_name in env.clients:
            env.clients[client_name].calc_tether_forces_on_cars()
```

```
        # Add a local (non-network) client to the client dictionary.
        self.clients = {'local':Client(THECOLORS["green"])}
```

```
            elif (event.type == pygame.MOUSEBUTTONDOWN):
                self.clients['local'].buttonIsStillDown = True

                (button1, button2, button3) = pygame.mouse.get_pressed()
                if button1:
                    self.clients['local'].mouse_button = 1
                elif button2:
                    self.clients['local'].mouse_button = 2
                elif button3:
                    self.clients['local'].mouse_button = 3
                else:
                    self.clients['local'].mouse_button = 0

            elif event.type == pygame.MOUSEBUTTONUP:
                self.clients['local'].buttonIsStillDown = False
                self.clients['local'].mouse_button = 0

        if self.clients['local'].buttonIsStillDown:
            # If it down, get the cursor position.
            self.clients['local'].cursor_location_px = (mouseX, mouseY) = pygame.mouse.get_pos()
```

```
        # Add up all the forces on the car.
        car_forces_N = (car.m_kg * self.g_mps2) + (car.cursorString_spring_force_N +
                                                   car.cursorString_carDrag_force_N )
```

```
        # Draw cursor strings.
        for client_name in env.clients:
            if (env.clients[client_name].selected_car != None):
                env.clients[client_name].draw_cursor_string()
```

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