Assignment: A3

Air-Track: Gravity and Wall Collisions

New physics calculation concepts:

- Wall collisions:
  - Detection of wall collisions.
  - Velocity changes caused by wall collisions.
- Gravity:
  - Forces (like gravity) cause a rate of change in the velocity (acceleration). The average velocity over the time step is then used to calculate the change in position.

    The algorithm outlined below for calculating position as affected by forces and acceleration is based on Euler's method:
    http://en.wikipedia.org/wiki/Euler_method

    Euler's method is the simplest of the Runge-Kutta methods:
    http://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_method

  - Mass. This is a calculated property based on the density and dimensions of the car. Mass is needed to calculate the force of gravity on an object.

Python language topics:

- If tests
- For loops
- Lists

(Note: the links to Python tutorials are in A01_game_loop_and_events.pdf.)

Problem statement:

(Again, start with a new Python file!)

Add algorithmic content to the previous exercise to simulate wall collisions and the force of gravity. Again, have a feature to demonstrate your code. Have at least **three** demo keys 1, 2, and 3... At least one of these should have gravity turned on.

Algorithmic description:

Gravity: (this should be added to the update_SpeedandPosition method of the air_track)

- Characterize gravity as the component along the tilted track. (Note: a level track has no component of gravity along the track.) This component will be some small fraction of normal gravity because tilted air tracks usually have a relatively small tilt angle. Something like one 20th of normal g is a realistic component value.
    - g_mps2 = 9.8/20.0
- Have this gravity value be an attribute of the air_track object.
- Use Newton's law (a = F/m) to calculate the acceleration and associated change in velocity during the time step. Then calculate the change in position based on the average velocity during the time step:
    - total_force_on_car_N = m_kg * g_mps2 + 0.0 + 0.0 + 0.0
    - a_mps2 = total_force_on_car_N / m_kg
    - v_end_mps = v_mps + (a_mps2 * dt)
    - v_avg_mps = (v_mps + v_end_mps)/2
    - x_m = x_m + (v_avg_mps * dt)
    - v_mps = v_end_mps

Collisions: (this should be a new method of the air_track. Call this method from the main game loop so that it gets executed in each frame).

- Loop over each car in the car list.
    - Check for car-wall collisions with the left wall by comparing the position of the left edge of the car with the position of the left wall (left edge of the Pygame window). Similarly, check the right wall.
        - Correct for wall penetration (overlap): move the car to the position it would be if had bounced at the surface and not penetrated. That is, back the car out a distance twice the amount of the penetration.
        - If there is a car-wall collision, reverse the value of the velocity:
            - v_mps = -1 * v_mps * CR

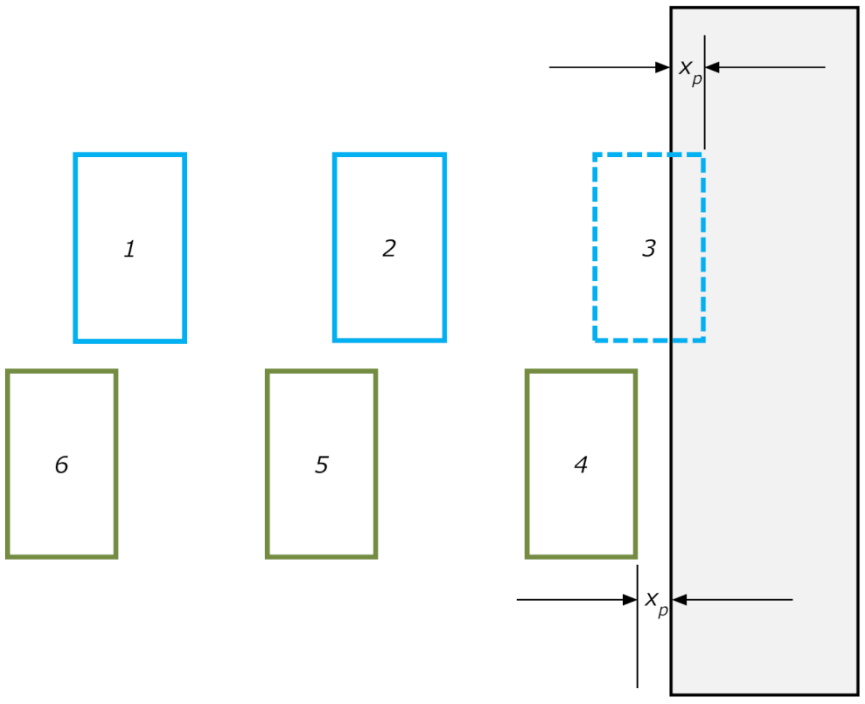

Figure 1. Car-wall collision. The numbers mark the sequence. The blue rectangles illustrate the car approaching the wall; green rectangles illustrate the car rebounding off the wall. The blue and green rectangles are shown vertically separated for clarity. The dotted-line blue rectangle #3 shows the detected collision (an overlap with the wall); this rectangle is not drawn in the game loop. Rectangle #4 shows the corrected position of the car; this rectangle is drawn. The corrected position is where the car would be if it had bounced at the surface instead of penetrating the wall.

Python code: (see images on next few pages)

The following code (image) is not a complete solution to the problem. It shows changes relative to assignment #2.

```
class AirTrack:
    def __init__(self):
        # Initialize the list of cars.
        self.cars = []
        self.carCount = 0

        # Coefficients of restitution.
        self.coef_rest_base = 0.90  # 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.gbase_mps2 = 9.8/20.0 # one 20th of g.
        self.g_mps2 = self.gbase_mps2

        self.color_transfer = False

    def update_SpeedandPosition(self, car, dt_s):

        # Add up all the forces on the car.
        car_forces_N = (car.m_kg * self.g_mps2) + 0.0 + 0.0

        # Calculate the acceleration based on the forces and Newton's law.
        car_acc_mps2 =

        # Calculate the velocity at the end of this time step.
        v_end_mps =

        # Calculate the average velocity during this timestep.
        v_avg_mps =

        # Use the average velocity to calculate the new position of the car.
        # Physics note: v_avg*t is equivalent to (v*t + (1/2)*acc*t^2)
        car.center_m =

        # Assign the final velocity to the car.
        car.v_mps =

    def check_for_collisions(self):
        # Collisions with walls.
        # Enumerate so can efficiently check car-car collisions below.

        fix_wall_stickiness = True # False True

        for car in self.cars:

            # Collisions with Left and Right wall.
            #   If left-edge of the car is less than...                OR  If right-edge of car is greater than...
            if ((car.center_m - car.width_m/2.0) < game_window.left_m) or :

                if fix_wall_stickiness:
                    self.correct_wall_penetrations(car)

                car.v_mps =
```

```
    def correct_wall_penetrations(self, car):
        penitration_left_x_m = game_window.left_m - ( - car.halfwidth_m)
        if penitration_left_x_m > 0:
            car.center_m += 2 *

        penitration_right_x_m = (car.center_m + car.halfwidth_m) -
        if > 0:
            car.center_m -= 2 * penitration_right_x_m
```

```
class Detroit:
    def __init__(self, color=THECOLORS["white"], left_px=10, width_px=26, height_px=98, v_mps=1):

        self.color = color

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

        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

        self.density_kgpm2 = 600.0
        self.m_kg = self.height_m * self.width_m * [redacted]

        # Increment the car count.
        air_track.carCount += 1
        # Name this car based on this air_track attribute.
        self.name = air_track.carCount

        # 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)
```

A couple updates to the make_some_cars method.

```
    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))

        if (nmode == 1):
            air_track.g_mps2 = 0
            air_track.carCount = 0
            self.cars.append( Detroit(color=THECOLORS["red" ], left_px = 240, width_px=26, v_mps=  0.2))
            self.cars.append( Detroit(color=THECOLORS["blue"], left_px = 340, width_px=26, v_mps= -0.2))

        elif (nmode == 2):
            air_track.g_mps2 = air_track.gbase_mps2
            air_track.carCount = 0
            self.cars.append( Detroit(color=THECOLORS["yellow" ], left_px = 240, width_px=26, v_mps= -0.1))
            self.cars.append( Detroit(color=THECOLORS["green"],   left_px = 440, width_px=50, v_mps= -0.2))

        elif (nmode == 3):
            air_track.carCount = 0
            air_track.g_mps2 = 0
            self.cars.append( Detroit(color=THECOLORS["yellow" ], left_px = 240, width_px=26, v_mps= -0.1))
            self.cars.append( Detroit(color=THECOLORS["green"],   left_px = 440, width_px=50, v_mps= -0.2))
```

An update to a portion of the main() function.

```
        # Update velocity and x position of each car based on the dt_s for this frame.
        for car in air_track.cars:
            air_track.update_SpeedandPosition(car, dt_s)

        # Check for collisions and apply collision physics to determine resulting
        # velocities.
        air_track.check_for_collisions()
```