Assignment: A4

Air-Track: Car-Car Collisions

New physics calculation concepts:

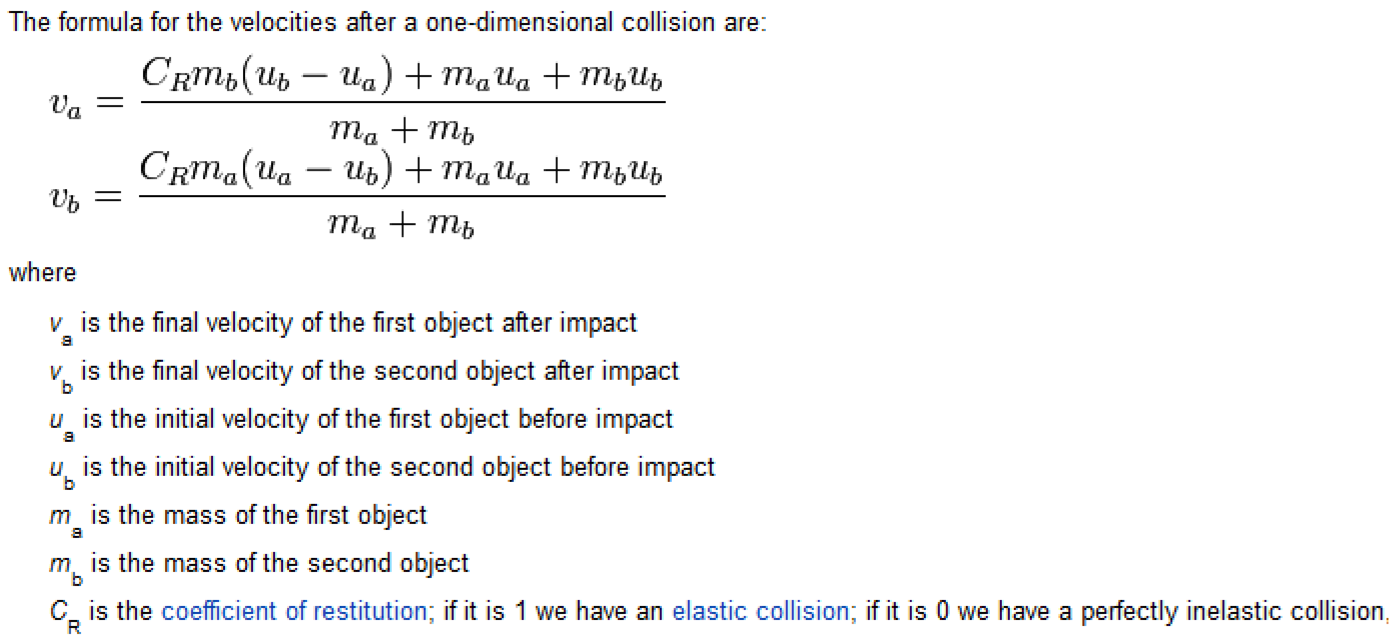
- Car-Car Collisions:
    - Detection of car-car collisions.
    - Penetration (overlap) correction.
    - Velocity changes caused by car collisions.

The following links give good background theory for the car-car collision physics. However, all that we need in this assignment is given in the two formulas below.

http://en.wikipedia.org/wiki/Elastic_collision#One-dimensional_Newtonian

http://en.wikipedia.org/wiki/Inelastic_collision

(Note: the formulas below are from the inelastic collision link above)



The formula for the velocities after a one-dimensional collision are:

$$v_a = \frac{C_R m_b(u_b - u_a) + m_a u_a + m_b u_b}{m_a + m_b}$$

$$v_b = \frac{C_R m_a(u_a - u_b) + m_a u_a + m_b u_b}{m_a + m_b}$$

where

v_a is the final velocity of the first object after impact

v_b is the final velocity of the second object after impact

u_a is the initial velocity of the first object before impact

u_b is the initial velocity of the second object before impact

m_a is the mass of the first object

m_b is the mass of the second object

C_R is the coefficient of restitution; if it is 1 we have an elastic collision; if it is 0 we have a perfectly inelastic collision.

Notice that in the special case when CR=1 and m_a=m_b, then the equations above reduce to v_a=u_b and v_b=u_a; that is, the two cars simply exchange velocities after the collision.

Python language topics:

- Enumerated for loops.
- List slices.

Note: Here again are the references links:
http://docs.python.org/2/tutorial/index.html
http://learnpythonthehardway.org/book/
http://www.pygame.org/docs/index.html

http://programarcadegames.com/ http://inventwithpython.com/appendixa.html

Problem statement:

(Again, start with a new Python file.)

Add algorithmic content to the previous exercise to simulate car-car collisions. Have at least three demo keys 1, 2, and 3... At least one of these should have gravity turned on. Add an attribute to the AirTrack class that keeps a running total of all the collisions.

Demonstrate that your penetration-correction code works by running demos with and without the corrections. First, convert the two variables (fix_wall_stickiness, fix_car_stickiness) that control this to AirTrack class attributes so that you can access them outside of the AirTrack class. Remember, you have given the air_track object global scope. Toggle these two variables (between True and False) through use of the "s" key.

Make a similar toggle for the color_transfer attribute. Toggle this with the "c" key.

Print the collision count and the two attributes out every frame in the while loop.

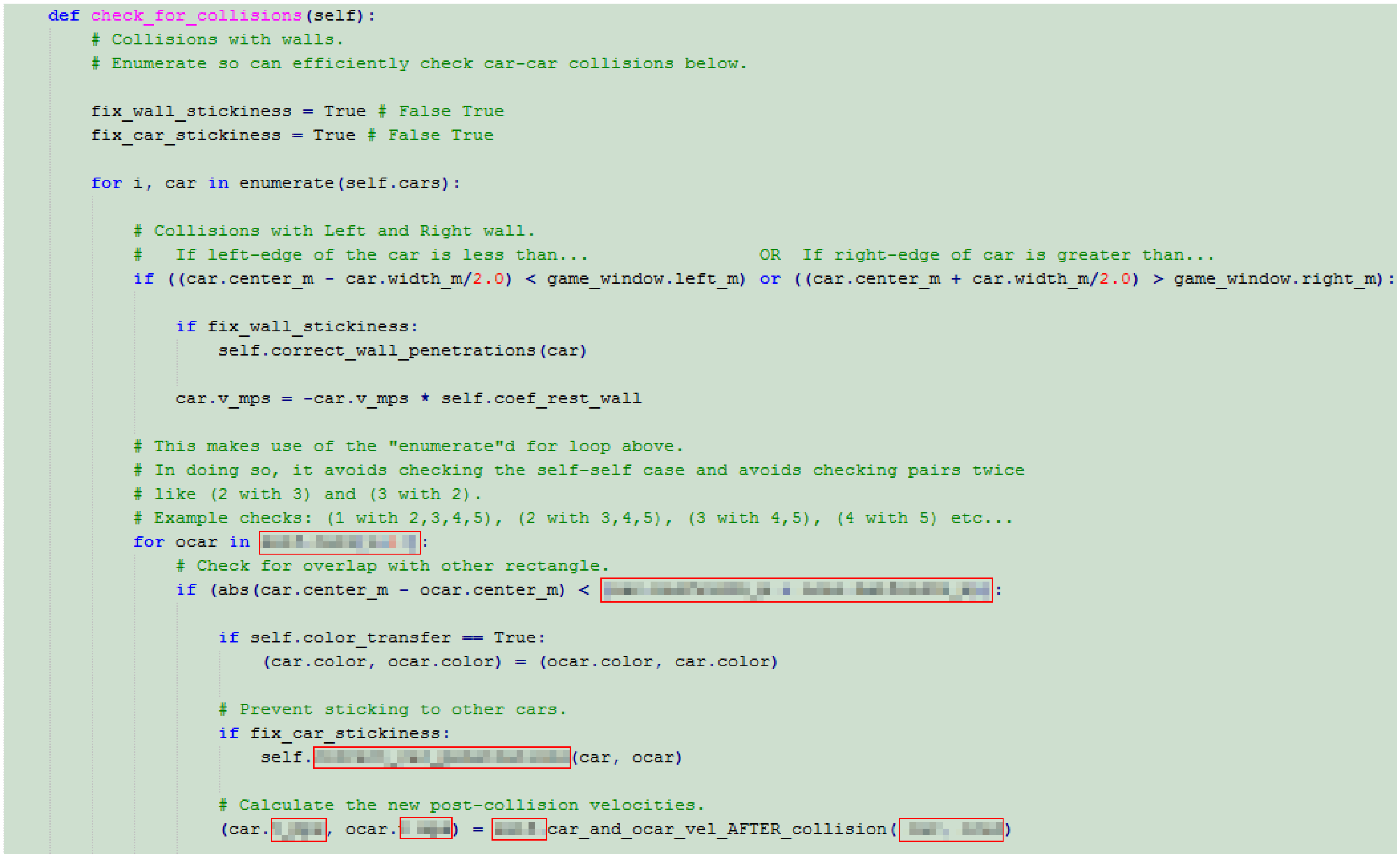
Algorithmic description:

Collisions: This is an upgrade to the existing collision related code in the air_track class.

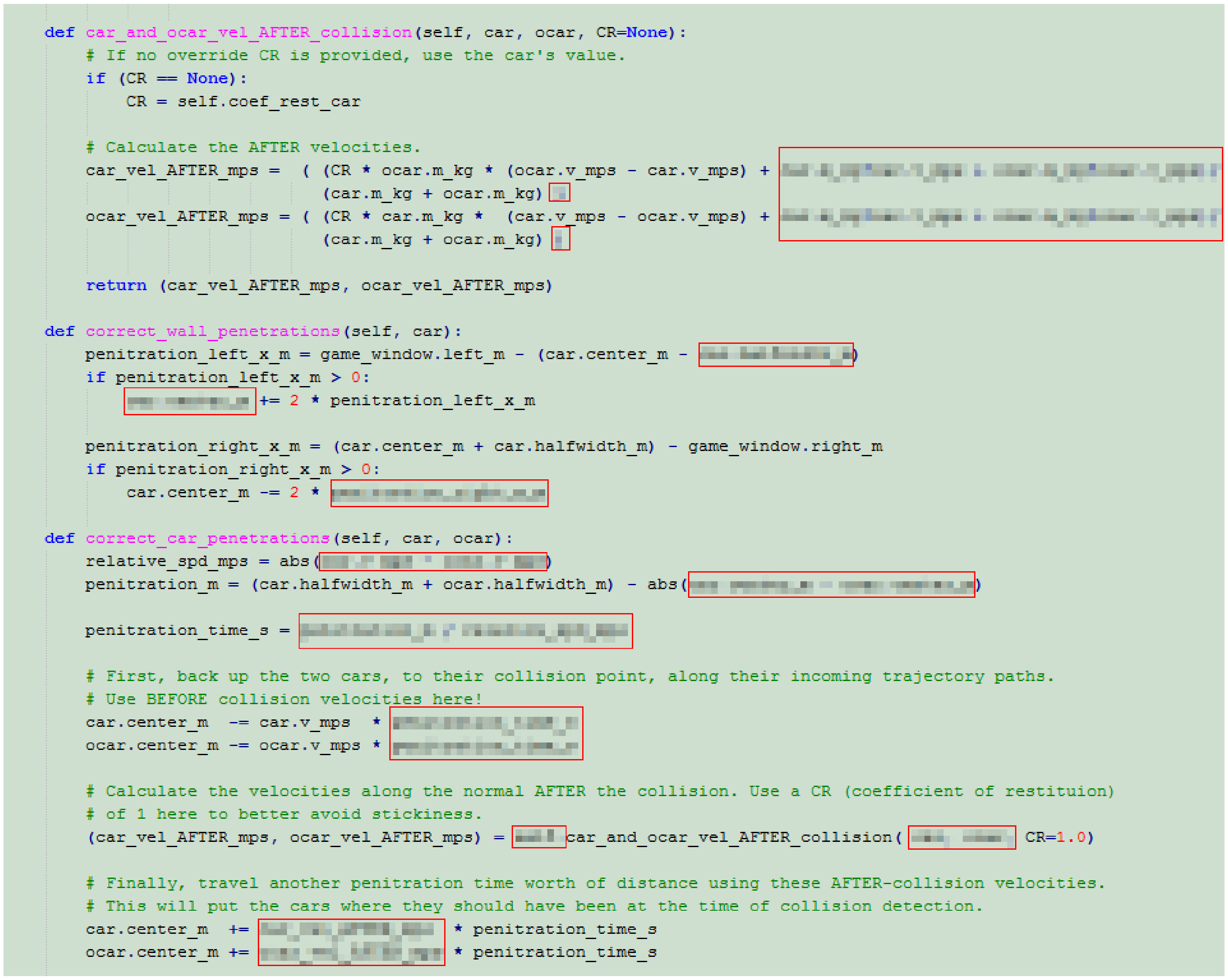
- Loop over each car in the car list (use enumeration in this for-loop). Call this car in the list "car":
  - Check for car-wall collisions with the left wall by comparing the position of the left edge of the car with the position of the left wall (left edge of the Pygame window). Similarly, check the 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
  - Have another for-loop over the remaining cars in the car list (use a Python slice). Call this car "ocar" in the for-loop (for <u>o</u>ther <u>car</u>). This sub loop will allow us to identify unique car-car pairs without repeating any. It will also avoid needlessly checking a car for collisions with itself (impossible).
    - Check for car-car collisions by comparing the position of the car and the ocar (other car): If the separation between the cars is less than the sum of the two half-widths, they have collided and are overlapping (penetrated).
      - Correct for car penetration: again, as with the wall collisions, back the position up to where the cars would be if there was no penetration. Since each car travels at a different velocity before and after the collision, you must do this in two steps.
        - First calculate the penetration time. This is the time needed for the two cars to reach the position they are at now (at collision detection) from the point where they would be just touching. This is the penetration distance divided by the relative velocity between the two cars.
          t_pen = x_pen / abs(car.v_mps – ocar.v_mps)
        - Next, back up the cars one penetration-time amount of travel at the incoming velocities (their velocities BEFORE the collision).

        ○ Next, move the cars one penetration-time amount of travel at the outgoing velocities (their velocities AFTER the collision). Use the equations above to calculate the velocities after the collision. But, use a CR=1 here to best avoid stickiness problems.
- Assign the corresponding post-collision velocities to the cars.
    ○ Again, use the equations above to calculate the post-collision velocities, but here use the actual CR value.

Python code: (see images on next few pages)
The following code (image) is not a complete solution to the problem. It mainly shows changes relative to assignment #3. The task in the problem statement that relates to the fix_wall_stickiness and fix_car_stickiness variables is not shown in the version of the code captured in the images below.



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

        fix_wall_stickiness = True # False True
        fix_car_stickiness = True # False True

        for i, car in enumerate(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 ((car.center_m + car.width_m/2.0) > game_window.right_m):

                if fix_wall_stickiness:
                    self.correct_wall_penetrations(car)

                car.v_mps = -car.v_mps * self.coef_rest_wall

            # This makes use of the "enumerate"d for loop above.
            # In doing so, it avoids checking the self-self case and avoids checking pairs twice
            # like (2 with 3) and (3 with 2).
            # Example checks: (1 with 2,3,4,5), (2 with 3,4,5), (3 with 4,5), (4 with 5) etc...
            for ocar in [redacted]:
                # Check for overlap with other rectangle.
                if (abs(car.center_m - ocar.center_m) < [redacted]:

                    if self.color_transfer == True:
                        (car.color, ocar.color) = (ocar.color, car.color)

                    # Prevent sticking to other cars.
                    if fix_car_stickiness:
                        self.[redacted](car, ocar)

                    # Calculate the new post-collision velocities.
                    (car.[redacted], ocar.[redacted]) = [redacted]car_and_ocar_vel_AFTER_collision([redacted])
```



```
    def car_and_ocar_vel_AFTER_collision(self, car, ocar, CR=None):
        # If no override CR is provided, use the car's value.
        if (CR == None):
            CR = self.coef_rest_car

        # Calculate the AFTER velocities.
        car_vel_AFTER_mps =  ( (CR * ocar.m_kg * (ocar.v_mps - car.v_mps) + [redacted]
                               (car.m_kg + ocar.m_kg) [redacted]
        ocar_vel_AFTER_mps = ( (CR * car.m_kg *  (car.v_mps - ocar.v_mps) + [redacted]
                               (car.m_kg + ocar.m_kg) [redacted]

        return (car_vel_AFTER_mps, ocar_vel_AFTER_mps)

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

        penitration_right_x_m = (car.center_m + car.halfwidth_m) - game_window.right_m
        if penitration_right_x_m > 0:
            car.center_m -= 2 * [redacted]

    def correct_car_penetrations(self, car, ocar):
        relative_spd_mps = abs([redacted])
        penitration_m = (car.halfwidth_m + ocar.halfwidth_m) - abs([redacted])

        penitration_time_s = [redacted]

        # First, back up the two cars, to their collision point, along their incoming trajectory paths.
        # Use BEFORE collision velocities here!
        car.center_m  -= car.v_mps  * [redacted]
        ocar.center_m -= ocar.v_mps * [redacted]

        # Calculate the velocities along the normal AFTER the collision. Use a CR (coefficient of restituion)
        # of 1 here to better avoid stickiness.
        (car_vel_AFTER_mps, ocar_vel_AFTER_mps) = [redacted]car_and_ocar_vel_AFTER_collision( [redacted] CR=1.0)

        # Finally, travel another penitration time worth of distance using these AFTER-collision velocities.
        # This will put the cars where they should have been at the time of collision detection.
        car.center_m  += [redacted] * penitration_time_s
        ocar.center_m += [redacted] * penitration_time_s
```