Assignment: A2

The Air-Track Framework

This assignment establishes the algorithmic framework (the Python classes) that will be used in the air-track and the air-table projects. This also introduces two basic calculation concepts:

- Time-based position calculations (our first dose of physics) and,
- The *meters* of the physics "World" and the *pixels* of the display screen, and converting back and forth between them.

(I'll have an air-track set up in the lab in case you've never had a chance to use one.)

Python language topics:

- Code blocks and indenting (a review).
- Functions: what they do and what they return.
- Classes:
    - Methods;
    - Properties;
    - Instantiation: creation and initialization of an instance (an object).
- Namespace and the "main" function.
- Floating-point and integer numbers:
    - Rounding of floating-point numbers.

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

Problem statement:

(First, be sure and start with a new Python file! You can use (copy and paste) stuff from the first exercise, but don't edit in your original copy of the previous exercise. That should be your pattern for each assignment: start a new file.)

Write a program to animate two rectangular objects (cars) in a one-dimensional space like an air-track.
Have the following controls:

1. The "esc" key to quit;
2. The number keys, "1" and "2", should be used to start each of two demos (without restarting the program). The two demos should differ in the initial position, velocity, and color of each of the cars.

Algorithmic description:

- Import content from modules.
- Define classes and functions.
    - GameWindow
        - Attributes: dimensions of the screen, and the display surface object.
        - Methods:
            - Initialize;

                - Set caption;
                - Erase screen.
        - Detroit
            - Attributes: car dimensions, position, speed, and rectangle object.
            - Methods:
                - Initialize;
                - Draw this car.
        - AirTrack
            - Attributes: the list of cars.
            - Methods:
                - Update car speed and position based on physics;
                - Make (instantiate) some cars based on demo mode.
        - Environment
            - Attributes: pixels-to-meters ratio, meters-to-pixels ratio.
            - Methods:
                - Pixels-to-meters conversion;
                - Meters-to-pixels conversion.

- Initialize the program
    - Initialize the first demo: build (instantiate) two cars, and initialize their position and velocity.
- The main game loop:
    - Erase the surface.
    - Establish the time step dt_s.
    - Check for user input: to quit, or to change demo mode.
    - Update the speed and x position of each car based on the time step for this frame.
        - position += velocity * dt_s
    - Draw each car at its new position
        - Convert from meters to pixels.
        - Then draw it.
    - Update the total time since starting (we don't actually use this yet but will later).
    - Make this update visible on the screen.

Python code: (see images on next few pages)

Here again, this code solution is provided (as an image) in the assignment statement. Some parts of the image have been obfuscated; you'll have to fill in the blanks to get this to run. Later in the course, code solutions will be provided as text files a day or two after the assignment is given.

```python
# Python
import sys, os
import pygame
import datetime

# PyGame Constants
from pygame.locals import *
from pygame.color import THECOLORS

#====================================================================
# Classes
#====================================================================

class GameWindow:
    def __init__(self, screen_tuple_px):
        self.width_px = screen_tuple_px[0]
        self.height_px = screen_tuple_px[1]

        # Create a reference to display's surface object. This object is a pygame "surface".
        # Screen dimensions in pixels (tuple)
        self.surface = [illegible]

        # Paint screen black.
        self.erase_and_update()

    def update_caption(self, title):
        pygame.dis[illegible]
        self.capti[illegible]

    def erase_and_update(self):
        # Useful for shifting between the various demos.
        self.surfac[illegible]
        pygame.disp[illegible]


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

        self.color = color

        self.height_px = hei[illegible]
        self.top_px    = gam[illegible]
        self.width_px  = wid[illegible]

        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 speed of the car. These are affected by the
        # physics calcs in the Track.
        self.center_m = env.m_from_px(left_px) + [illegible]
        self.speed_mps = speed_mps

```

```
55          # Create a rectangle object based on these dimensions
56          # Left: distance from the left edge of the screen in px.
57          # Top:  distance from the top  edge of the screen in px.
58          self.rect = pygame.Rect(left_px, self.top_px, self.width_px, self.height_px)
59
60      def draw_car(self):
61          # Update the pixel position of the car's rectangle object to match the value
62          # controlled by the physics calculations.
63          self.rect.centerx = env.px_from_m( self.[illegible])
64
65          # Draw the main rectangle.
66          pygame.draw.rect(game_window.surface, [illegible], self.rect)
67
68
69  class AirTrack:
70      def __init__(self):
71          # Initialize the list of cars.
72          self.cars = []
73
74      def update_SpeedandPosition(self, car, dt_s):
75          # Calculate the new physical car position
76          car.center_m = car.center_m + [illegible]
77
78      def make_some_cars(self, nmode):
79          # Update the caption at the top of the pygame window frame.
80          game_window.update_caption("Air Track (basic): Demo #" + str(nmode))
81
82          if (nmode == 1):
83              self.cars.append( Detroit(color=THECOLORS["red" ], [illegible]
84              self.cars.append( Detroit(color=THECOLORS["blue"], [illegible]
85
86          elif (nmode == 2):
87              self.cars.append( Detroit(color=THECOLORS["yellow" [illegible]
88              self.cars.append( Detroit(color=THECOLORS["green"], [illegible]
89
90
91  class Environment:
92      def __init__(self, length_px, length_m):
93          self.px_to_m = length_m/float(length_px)
94          self.m_to_px = (float(length_px)/length_m)
95
96      # Convert from meters to pixels
97      def px_from_m(self, dx_m):
98          return int(round(dx_m * self.m_to_px))
99
100     # Convert from pixels to meters
101     def m_from_px(self, dx_px):
102         return float(dx_px) * self.px_to_m
103
104     def get_local_user_input(self):
105
```

```
106         # Get all the events since the last call to get().
107         for event in pygame.event.get():
108             if (event.type == pygame.QUIT):
109                 return 'quit'
110             elif (event.type == pygame.KEYDOWN):
111                 if (event.key == K_[redacted]):
112                     return 'quit'
113                 elif (event.key==K_[redacted]
114                     return 1
115                 elif (event.key==K_[redacted]
116                     return 2
117                 else:
118                     return "Nothing set up for this key."
119 
120             elif (event.type == pygame.KEYUP):
121                 pass
122 
123             elif (event.type == pygame.MOUSEBUTTONDOWN):
124                 pass
125 
126             elif (event.type == pygame.MOUSEBUTTONUP):
127                 pass
128 
129 
130 #============================================================
131 # Main procedural functions.
132 #============================================================
133 
134 def main():
135 
136     # A few globals.
137     global env, game_window, air_track
138 
139     # Initiate pygame
140     pygame.init()
141 
142     # Tuple to define window dimensions
143     window_size_px = window_width_px, window_height_px = 950, 120
144 
145     # Instantiate an Environment object for converting back and forth from pixels and meters.
146     # The also creates the local client.
147     env = Environment(window_width_px, 1.5)
148 
149     # Instantiate the window.
150     game_window = GameWindow(window_size_px)
151 
152     # Instantiate an air track (this adds an empty car list to the track).
153     air_track = AirTrack()
```

```
154
155     # Make some cars (run demo #1).
156     air_track.[redacted]
157
158     # Instantiate clock to help control the framerate.
159     myclock = pygame.time.Clock()
160
161     # Control the framerate.
162     framerate_limit = 400
163
164     time_s = 0.0
165     user_done = [redacted]
166
167     while not user_done:
168
169         # Erase everything.
170         game_window.surface.fill([redacted]"black"])
171
172         # Get the delta t for one frame (this changes depending on system load).
173         dt_s = [redacted]
174
175         # Check for user initiated stop or demo change.
176         resetmode = env.[redacted]
177         if (resetmode in [0,1,2,3,4,5,6,7,8,9]):
178             print "reset mode =", resetmode
179
180             # This should remove all references to the cars and effectively deletes them.
181             air_track.cars = []
182
183             # Now just black everything out and update the screen.
184             game_window.era[redacted]
185
186             # Build new set of cars based on the reset mode.
187             air_track.make_some_cars( resetmode)
188
189         elif (resetmode == 'quit'):
190             user_done = [redacted]
191
192         elif (resetmode != None):
193             print resetmode
194
195         # Update speed and x position of each car based on the dt_s for this frame.
196         for car in air_track.cars:
197             air_track.update_[redacted]
198
199         # Draw the car at the new position.
200         for car in air_track.cars:
201             car.[redacted]
202
203         # Update the total time since starting.
204         time_s += [redacted]
205
206         # Make this update visible on the screen.
207         pygame.display.[redacted]
```

```
208
209 #=============================================================
210 # Run the main program.
211 #=============================================================
212
213 [redacted]
```