Assignment: A15 (help)

Air-Table: Final features

The following two images highlight the changes that need to be made to the “Spring” class to support the pinned-spring feature.

```
class Spring:
    def __init__(self, p1, p2, length_m=3.0, strength_Npm=0.5, spring_color=THECOLORS["yellow"], width_m=0.025, drag_c=0.0):

        # Optionally this spring can have one end pinned to a vector point. Do this by passing in p2 as a vector.
        if (p2.__class__.__name__ == 'Vec2D'):
            # Creat a point puck at the pinning location.
            # The location of this point puck will never change because
            # it is not in the pucks list that is processed by the
            # physics engine.
            p2 = Puck( p2, 1.0, 1.0)
            p2.vel_2d_mps = Vec2D(0.0,0.0)
            length_m = 0.0

        self.p1 = p1
        self.p2 = p2
        self.p1p2_separation_2d_m = Vec2D(0,0)
        self.p1p2_separation_m = 0
        self.p1p2_normalized_2d = Vec2D(0,0)

        self.length_m = length_m
        self.strength_Npm = strength_Npm
        self.damper_Ns2pm2 = 0.5 #5.0 #0.05 #0.15
        self.unstretched_width_m = width_m #0.05

        self.drag_c = drag_c

        self.spring_vertices_2d_m = []
        self.spring_vertices_2d_px = []

        self.spring_color = spring_color
        self.draw_as_line = False
```

```python
def calc_spring_forces_on_pucks(self):
    self.p1p2_separation_2d_m = self.p1.pos_2d_m - self.p2.pos_2d_m

    self.p1p2_separation_m =  self.p1p2_separation_2d_m.length()

    # The pinned case needs to be able to handle the zero length spring. The
    # separation distance will be zero when the pinned spring is at rest.
    # This will cause a divide by zero error if not handled here.
    if ((self.p1p2_separation_m == 0.0) and (self.length_m == 0.0)):
        spring_force_on_1_2d_N = Vec2D(0.0,0.0)
    else:
        self.p1p2_normalized_2d = self.p1p2_separation_2d_m / self.p1p2_separation_m

        # Spring force:  acts along the separation vector and is proportional to the separation distance.
        spring_force_on_1_2d_N = self.p1p2_normalized_2d * (self.length_m - self.p1p2_separation_m) * self.strength_Npm

    # Damper force: acts along the separation vector and is proportional to the relative speed.
    v_relative_2d_mps = self.p1.vel_2d_mps - self.p2.vel_2d_mps
    v_relative_alongNormal_2d_mps = v_relative_2d_mps.projection_onto(self.p1p2_separation_2d_m)
    damper_force_on_1_N = v_relative_alongNormal_2d_mps * self.damper_Ns2pm2

    # Net force by both spring and damper
    SprDamp_force_2d_N = spring_force_on_1_2d_N - damper_force_on_1_N

    # This force acts in opposite directions for each of the two pucks. Notice the "+=" here, this
    # is an aggregate across all the springs. This aggregate MUST be reset (zeroed) after the movements are
    # calculated. So by the time you've looped through all the springs, you get the NET force, one each ball,
    # applied of all individual springs.
    self.p1.SprDamp_force_2d_N += SprDamp_force_2d_N * (+1)
    self.p2.SprDamp_force_2d_N += SprDamp_force_2d_N * (-1)

    # Add in some drag forces if a non-zero drag coef is specified. These are based on the
    # velocity of the pucks (not relative speed as is the case above for damper forces).
    self.p1.SprDamp_force_2d_N += self.p1.vel_2d_mps * (-1) * self.drag_c
    self.p2.SprDamp_force_2d_N += self.p2.vel_2d_mps * (-1) * self.drag_c
```