Assignment: A14

Air-Table: The Puck Gun

New concepts:

- Adding a new force to the physics engine.
- Impulse force = F_avg * dt = dv * mass.
- Launching bullet pucks from a moving puck.

Python language topics:

- Deleting elements of a list based on timestamp data.
- Another example of class inheritance.

Problem statement:

(Again, start with a new Python file.)

Add content to the A13 exercise to create a puck-firing gun. Control the firing with the "i" key. Control the rotation of the gun tube with the "j" and "l" tube. Fire the gun repeatedly (ten bullets per second) while holding down the "i" key. Bullets should be deleted if they are more than 3 seconds old. Model the recoil associated with firing each bullet using an impulse calculation in the physics engine.

Algorithmic description:

- Create a Gun class based on the RotatingTube class.
- Use the physics-world clock to timestamp the firing of the gun and the creation of a bullet.
    - Use a time-delay value of 0.1 seconds to determine when it is time to fire another bullet (if the "i" key is still down).
    - Use an age-limit value of 3 seconds to determine when it is time to delete an existing bullet.
        - Use a copy of the puck list to drive the "for" loop while deleting from the original puck list.
- Modify the physics calculations to include an impulse-force vector based on the velocity and mass of the fired bullet.

Python code: (see images on next few pages)
The following code is not a complete solution to the problem. It shows additional content relative to the A13 assignment. There is some obfuscation and some highlighting of new code lines. The images below should contain all the additional code you will need. The indent levels should be a clue to you.

```
class Puck:
    def __init__(self, pos_2d_m, radius_m, density_kgpm2, puck_color = THECOLORS["grey"]):
        self.radius_m = radius_m
        self.radius_px = int(round(env.px_from_m(self.radius_m * env.viewZoom)))

        self.density_kgpm2 = density_kgpm2    # mass per unit area
        self.mass_kg = self.density_kgpm2 * math.pi * self.radius_m ** 2
        self.pos_2d_m = pos_2d_m
        self.vel_2d_mps = Vec2D(0.0,0.0)

        self.SprDamp_force_2d_N = Vec2D(0.0,0.0)
        self.jet_force_2d_N = Vec2D(0.0,0.0)
        self.cursorString_spring_force_2d_N = Vec2D(0.0,0.0)
        self.cursorString_puckDrag_force_2d_N = Vec2D(0.0,0.0)

        self.impulse_2d_Ns = Vec2D(0.0,0.0)

        self.selected = False

        self.color = puck_color

        self.client_name = None
        self.jet = None
        self.gun = None
        self.rawtube = None
        self.hit = False

        # Bullet data...
        self.bullet = False
        self.birth_time_s = env.time_s
        self.age_limit_s = 3.0
```

```
class Gun( RotatingTube):
    def __init__(self, puck):
        RotatingTube.__init__(self, puck)

        # Degrees of rotation per rendering cycle.
        self.rotation_rate = 1.5

        #self.color = THECOLORS["yellow"]
        self.color = env.clients[self.puck.client_name].cursor_color

        self.rotate_everything( 45)

        self.bullet_speed_mps = 5.0
        self.fire_time_s = env.time_s
        self.firing_delay_s = 0.1

        self.testing_gun = False

    def client_rotation_control(self, client_name):
        if (env.clients[client_name].key_j == "D"):
            self.rotate_everything( +self.rotation_rate)
        if (env.clients[client_name].key_l == "D"):
            self.rotate_everything( -self.rotation_rate)

    def control_firing(self, client_name):
        if ((env.clients[client_name].key_i == "D") or self.testing_gun):
            if ((env.time_s - self.fire_time_s) > [redacted]):
                self.[redacted]()
                # Timestamp the firing event.
                self.fire_time_s = [redacted]

    def fire_gun(self):
        bullet_radius_m = 0.05
        # Set the initial position of the bullet so that it clears (doesn't collide with) the host puck.
        initial_position_2d_m = (self.puck.pos_2d_m +
                                 (self.direction_2d_m * (1.1 * self.puck.radius_m + 1.1 * bullet_radius_m)) )
        temp_bullet = Puck(initial_position_2d_m,  bullet_radius_m, 0.3)

        temp_bullet.vel_2d_mps = (self.puck.vel_2d_mps + ([redacted]))
        temp_bullet.bullet = True
        temp_bullet.color = env.clients[self.puck.client_name].cursor_color
        temp_bullet.client_name = self.puck.client_name

        air_table.pucks.append( [redacted])

        # Recoil impulse from firing the gun (opposite the direction of the bullet).
        self.puck.impulse_2d_Ns = temp_bullet.vel_2d_mps * temp_bullet.mass_kg * [redacted]

    def draw(self):
        # Draw the gun tube.
        line_thickness = 3
        pygame.draw.polygon(game_window.surface, self.color,
                self.convert_from_world_to_screen(self.tube_vertices_2d_m, self.puck.pos_2d_m), line_thickness)
```

```
    def update_PuckSpeedAndPosition(self, puck, dt_s):
        # Net resulting force on the puck.
        puck_forces_2d_N = (self.g_2d_mps2 * puck.mass_kg) + (puck.SprDamp_force_2d_N +
                                                              puck.jet_force_2d_N +
                                                              puck.cursorString_spring_force_2d_N +
                                                              puck.cursorString_puckDrag_force_2d_N +
                                                              puck.impulse_2d_Ns/dt_s)

        # Acceleration from Newton's law.
        acc_2d_mps2 = puck_forces_2d_N / puck.mass_kg

        # Acceleration changes the velocity:  dv = a * dt
        # Velocity at the end of the timestep.
        puck.vel_2d_mps = puck.vel_2d_mps + (acc_2d_mps2 * dt_s)

        # Calculate the new physical puck position using the average velocity.
        # Velocity changes the position:  dx = v * dt
        puck.pos_2d_m = puck.pos_2d_m + (puck.vel_2d_mps * dt_s)

        # Now reset the aggregate forces.
        puck.SprDamp_force_2d_N = Vec2D(0.0,0.0)
        puck.cursorString_spring_force_2d_N = Vec2D(0.0,0.0)
        puck.cursorString_puckDrag_force_2d_N = Vec2D(0.0,0.0)
        puck.impulse_2d_Ns = Vec2D(0.0,0.0)
```

```
            for controlled_puck in air_table.controlled_pucks:
                # Rotate based on keyboard of the controlling client.
                controlled_puck.jet.client_rotation_control( controlled_puck.client_name)
                controlled_puck.gun.client_rotation_control( controlled_puck.client_name)

                # Turn gun on/off
                controlled_puck.gun.control_firing( controlled_puck.client_name)

                # Turn shield on/off
                #controlled_puck.gun.control_shield( controlled_puck.client_name)
                pass
```

```
            # Clean out old bullets.
            puck_list_copy = air_table.pucks[:]
            for thisPuck in puck_list_copy:
                if (thisPuck.bullet) and ((env.time_s - [redacted].birth_time_s) > thisPuck.[redacted]):
                    air_table.pucks.remove([redacted])
            del puck_list_copy
```

```
            for eachpuck in air_table.pucks:
                eachpuck.draw()
                if (eachpuck.jet != None) or (eachpuck.rawtube != None):
                    if ((env.clients[eachpuck.client_name].Qcount < qCount_limit) or (eachpuck.client_name == 'local')):
                        eachpuck.jet.draw()
                        eachpuck.gun.draw()
```