- Modify the `Agent` class so that each agent will have its own path instance and will be able to follow it when in "`follow_path`" mode.
  - In the `__init__` method add the following (without the comments)

- In the `__init__` method add the following (without the comments)

```
self.path = Path()
self.randomise_path()  # <-- Doesn't exist yet but you'll create it
self.waypoint_threshold = 0.0  # <-- Work out a value for this as you test!
```

AI for games 〉 09 - Lab - Steering 2 〉 agent.py

```
        # keep a reference to the world object
        self.world = world
        self.mode = mode
        # where am i and where am i going? random start pos
        dir = radians(random() * 360)
        self.pos = Vector2D(randrange(world.cx), randrange(world.cy))
        self.vel = Vector2D()
        self.heading = Vector2D(sin(dir), cos(dir))
        self.side = self.heading.perp()
        self.scale = Vector2D(scale, scale)  # easy scaling of agent size
        self.force = Vector2D()  # current steering force
        self.accel = Vector2D()  # current acceleration due to force
        self.mass = mass

        self.path = Path()
        self.randomise_path()
        self.waypoint_threshold = 10.0

        # data for drawing this agent
        self.color = 'ORANGE'
        self.vehicle_shape = [
            Point2D(-1.0, 0.6),
            Point2D(1.0, 0.0),
            Point2D(-1.0, -0.6)
        ]
        self.wander_target = Vector2D(1, 0)
        self.wander_dist = 1.0 * scale
        self.wander_radius = 1.0 * scale
        self.wander_jitter = 10.0 * scale
        self.bRadius = scale
```

AI for games 〉 09 - Lab - Steering 2 〉 agent.py

```
        ]
        self.wander_target = Vector2D(1, 0)
        self.wander_dist = 1.0 * scale
        self.wander_radius = 1.0 * scale
        self.wander_jitter = 10.0 * scale
        self.bRadius = scale

        self.max_speed = 20.0 * scale
        self.max_force = 1000.0

        self.show_info = False

        # self.max_speed = 500.0 * (randrange(50, 100) / 100)
        # data for drawing this agent
        self.color = 'ORANGE'
        self.vehicle_shape = [
            Point2D(-1.0, 0.6),
            Point2D(1.0, 0.0),
            Point2D(-1.0, -0.6)
        ]

        # NEW WANDER INFO
        self.wander_target = Vector2D(1, 0)
        self.wander_dist = 1.0 * scale
        self.wander_radius = 1.0 * scale
        self.wander_jitter = 10.0 * scale
        self.bRadius = scale
        # Force and speed limiting code
        self.max_speed = 20.0 * scale
        self.max_force = 500.0
```

- Create the new `randomise_path()` so that it's ready to be called. In it, call the `path.create_random_path()` method using world-related parameters

```
cx = self.world.cx  # width
cy = self.world.cy  # height
margin = min(cx, cy) * (1/6)  # use this for padding in the next line ...
self.path.create_random_path(...)  # you have to figure out the parameters
```

```python
def randomise_path(self):
    cx = self.world.cx
    cy = self.world.cy
    margin = min(cx, cy) * 0.2
    self.path.create_random_path(8, margin, margin, cx, cy)
```

- Add a "`follow_path`" mode and modify the calculate method to use it by calling a (new) `follow_path` method.

- Add a `follow_path` method and code the following logical ideas:

```
# If heading to final point (is_finished?),
#     Return a slow down force vector (Arrive)
# Else
#   If within threshold distance of current way point, inc to next in path
#   Return a force vector to head to current point at full speed (Seek)
```

```python
def follow_path(self):

    if self.path.is_at_end_of_path():
        if self.pos.distance(self.path.current_pt()) <= self.waypoint_threshold:
            self.path.inc_current_pt()
        else:
            # arrive at current point
            return self.arrive(self.path.current_pt(), 'slow')
    else:
        # if within threashold distance of current point, inc_current_point
        if self.pos.distance(self.path.current_pt()) <= self.waypoint_threshold:
            self.path.inc_current_pt()
        # Else Seek current point
        else:
            return self.seek(self.path.current_pt())
    return Vector2D(0, 0)
```

- Modify the `render` method to call `path.render()` if currently in "`follow_path`" mode.


```python
def render(self, color=None):
    ''' Draw the triangle agent with color'''
    # draw the path if it exists and the mode is follow
    if self.mode == 'follow_path':
        self.path.render()
```

- Alter the `main.py` file so that you can reset the agent paths in response to a key press (say 'R'), by looping through all the agents and calling their `randomise_path()`.
- Test it! Adjust the waypoint threshold distance (which will depend on if you used a squared distance value or not), adjust force and max speed values, and try different arrive speeds.

```
def on_key_press(symbol, modifiers):
    if symbol == KEY.P:
        world.paused = not world.paused
    if symbol == KEY.R:
        for agent in world.agents:
            agent.randomise_path()
```

Tried Changing the threshold value but couldn't see much difference.

```
    self.path_looped = looped
    self.randomise_path(looped)
    self.waypoint_threshold = 20.0
```

Render:

```
def render(self, color=None):
    ''' Draw the triangle agent with color'''
    # draw the path if it exists and the mode is follow
    if self.mode == 'follow_path':
        self.path.render()
    # draw the ship
    egi.set_pen_color(name=self.color)
    pts = self.world.transform_points(self.vehicle_shape, self.pos,
                                      self.heading, self.side, self.scale)
    # draw it!
    egi.closed_shape(pts)

    # add some handy debug drawing info lines - force and velocity
    if self.show_info:
        s = 0.5  # <-- scaling factor
        # force
        egi.red_pen()
        egi.line_with_arrow(self.pos, self.pos + self.force * s, 5)
        # velocity
        egi.grey_pen()
        egi.line_with_arrow(self.pos, self.pos + self.vel * s, 5)
        # net (desired) change
        egi.white_pen()
        egi.line_with_arrow(self.pos + self.vel * s, self.pos + (self.force + self.vel) * s, 5)
        egi.line_with_arrow(self.pos, self.pos + (self.force + self.vel) * s, 5)
```

Transform points:

```python
def transform_points(self, points, pos, forward, side, scale):
    ''' Transform the given list of points, using the provided position,
        direction and scale, to object world space. '''
    # make a copy of original points (so we don't trash them)
    wld_pts = [pt.copy() for pt in points]
    # create a transformation matrix to perform the operations
    mat = Matrix33()
    # scale,
    mat.scale_update(scale.x, scale.y)
    # rotate
    mat.rotate_by_vectors_update(forward, side)
    # and translate
    mat.translate_update(pos.x, pos.y)
    # now transform all the points (vertices)
    mat.transform_vector2d_list(wld_pts)
    # done
    return wld_pts
```

Output: