Step 1: The game was improved using a simple track that the agent could follow. Track used can be randomised by pressing R.

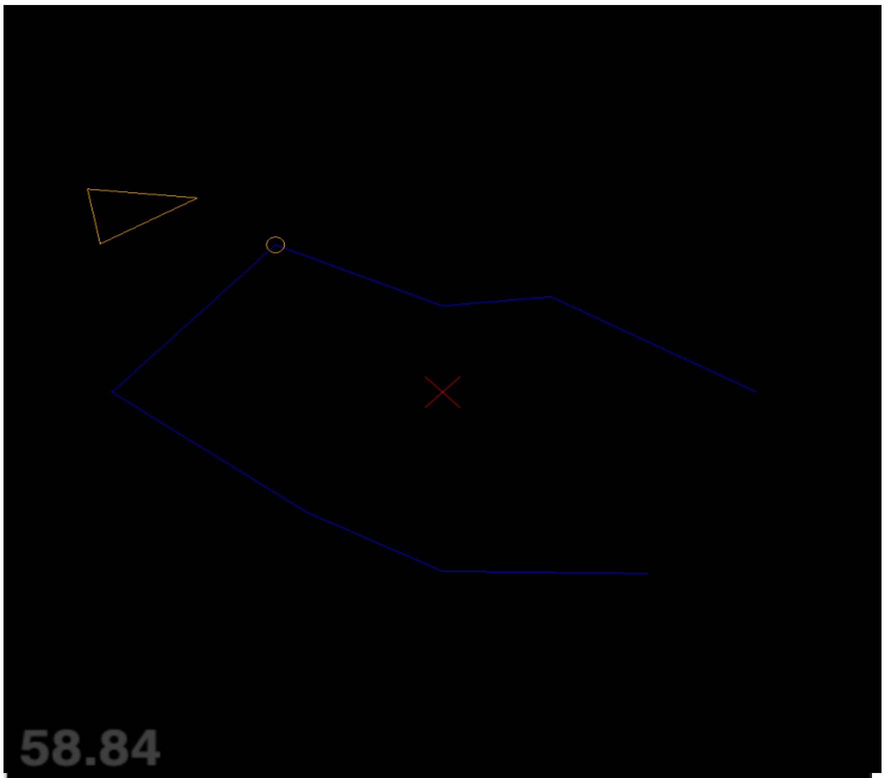
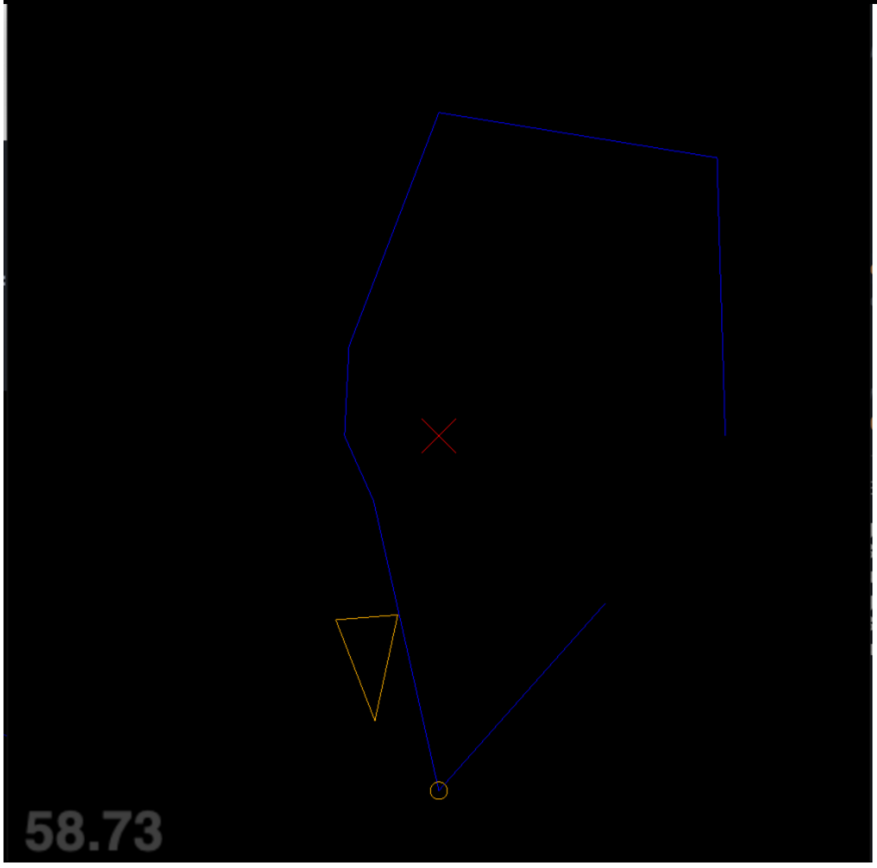
Agent.py

```
def follow_path(self):
    self.path.render()
    if self.path.is_finished():
      return self.arrive(self.path.current_pt(), 'slow')
    else:
      if (self.path.current_pt() - self.pos).length() < 50:
        self.path.inc_current_pt()
      else:
        return self.seek(self.path.current_pt())
    return Vector2D()
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
    if(color == None):
      egi.set_pen_color(name=self.color)
    else:
      egi.set_pen_color(name=color)
    pts = self.world.transform_points(self.vehicle_shape, self.pos,
                     self.heading, self.side, self.scale)
    # draw it!
    egi.closed_shape(pts)
```

Main.py

```
## LAB 09 STEP 1: Reset all paths to new random ones
  elif symbol == KEY.R:
    for agent in world.agents:
      agent.randomise_path()
```


58.84

58.73

Step 2: The ship is now equipped with wandering mode. Adding shapes to the ship by pressing I.

Agent.py

```python
if self.mode == 'wander':
        ## ...
        wnd_pos = Vector2D(self.wander_dist, 0)
        wld_pos = self.world.transform_point(
            wnd_pos, self.pos, self.heading, self.side)

        egi.green_pen()
        egi.circle(wld_pos, self.wander_radius)

        egi.red_pen()
        wnd_pos = (self.wander_target + Vector2D(self.wander_dist, 0))
        wld_pos = self.world.transform_point(
            wnd_pos, self.pos, self.heading, self.side)
        egi.circle(wld_pos, 3)

def update(self, delta):
    ''' update vehicle position and orientation '''
    # calculate and set self.force to be applied
    ## force = self.calculate()
    force = self.calculate(delta)  # <-- delta needed for wander
    ## limit force? <-- for wander
    # ...
    force.truncate(self.max_force)
    # determin the new accelteration
    self.accel = force / self.mass  # not needed if mass = 1.0
    # new velocity
    self.vel += self.accel * delta
    # check for limits of new velocity
    self.vel.truncate(self.max_speed)
    # update position
    self.pos += self.vel * delta
    # update heading is non-zero velocity (moving)
    if self.vel.length_sq() > 0.00000001:
        self.heading = self.vel.get_normalised()
        self.side = self.heading.perp()
    # treat world as continuous space - wrap new position if needed
    self.world.wrap_around(self.pos)
```