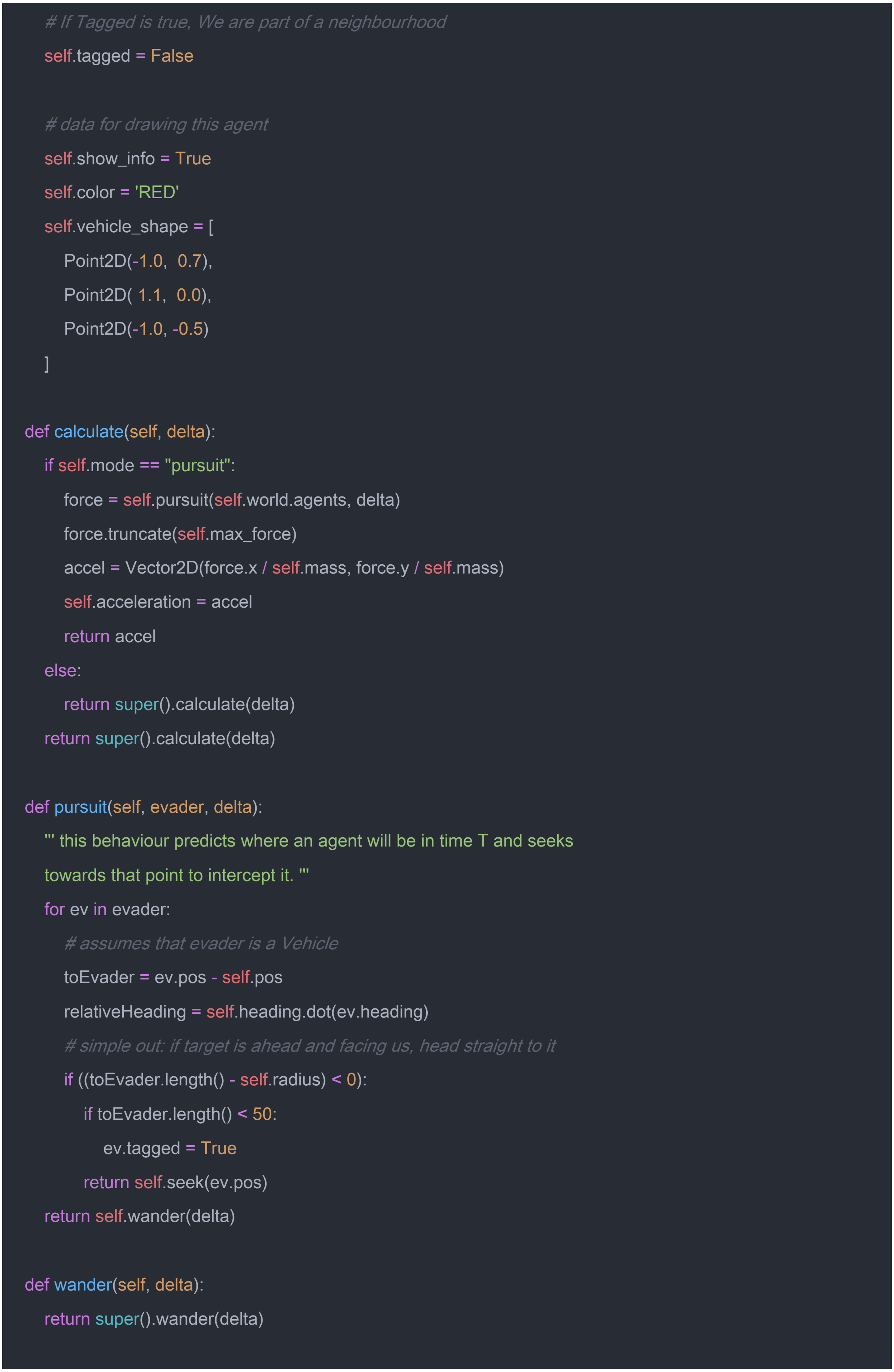
Hunter.py

Hunter will try and pursuit prey.

```
from agent import *
from path import Path
from vector2d import Vector2D
from vector2d import Point2D
from graphics import egi, KEY
from math import sin, cos, radians
from random import random, randrange, uniform

class Hunter(Agent):

    def __init__(self, world=None, scale=30.0, mass=1.0, mode='pursuit', looped = True):
        # keep a reference to the world object
        self.world = world
        self.mode = mode
        # where am i and where am i going? random
        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.acceleration = Vector2D()  # current steering force
        self.mass = mass
        # limits?
        self.max_speed = 20.0 * scale / 2
        self.max_force = 500.0

        # Wander Info
        self.wander_target = Vector2D (1,0)
        self.wander_dist = 1.0 * scale
        self.wander_radius = 1.0 * scale
        self.wander_jitter = 1.0 * scale
        self.bRadius = scale

        #Pursuit Info
        self.radius = 200
```

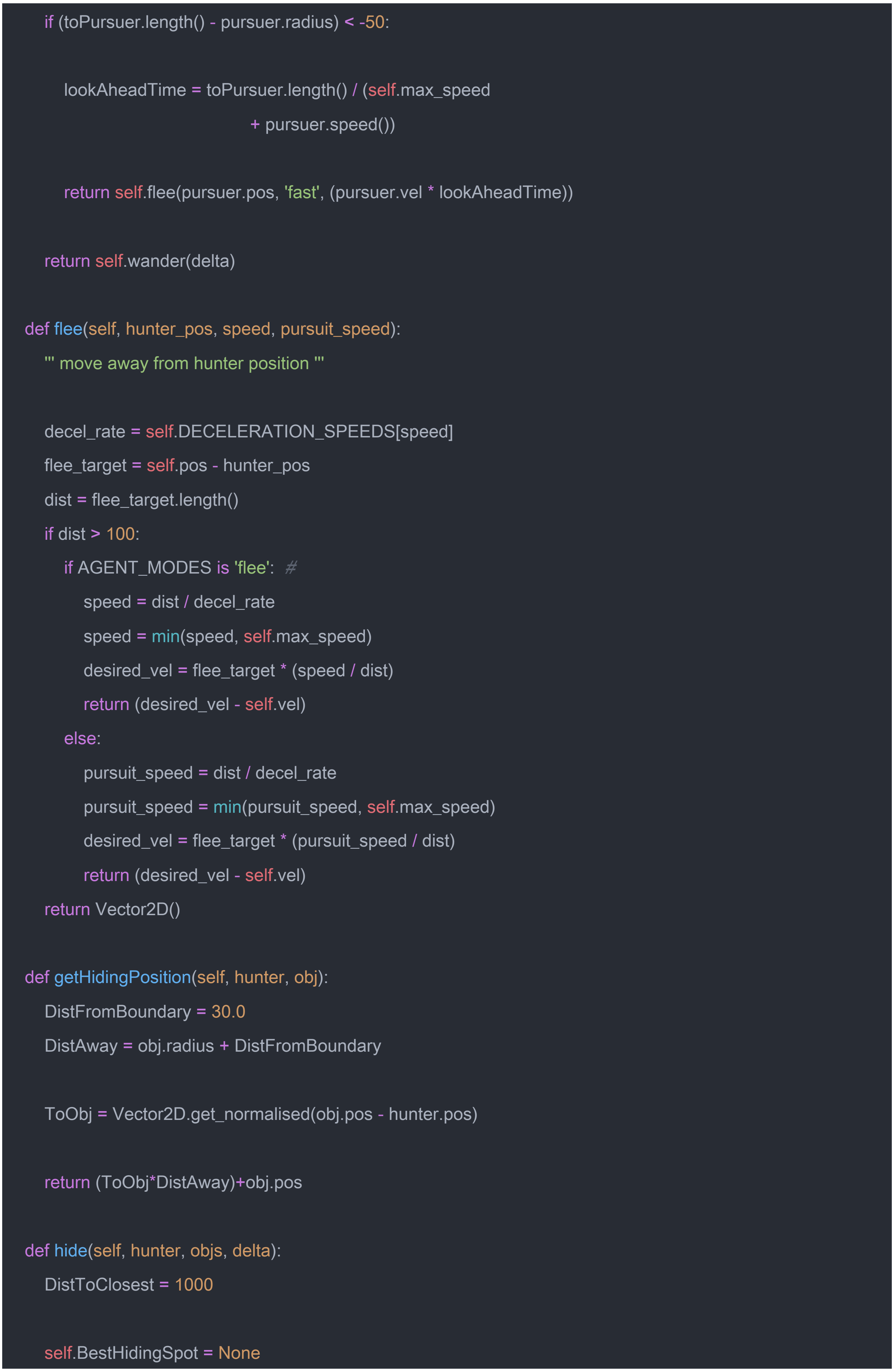


```
        # If Tagged is true, We are part of a neighbourhood
        self.tagged = False

        # data for drawing this agent
        self.show_info = True
        self.color = 'RED'
        self.vehicle_shape = [
            Point2D(-1.0,  0.7),
            Point2D( 1.1,  0.0),
            Point2D(-1.0, -0.5)
        ]

    def calculate(self, delta):
        if self.mode == "pursuit":
            force = self.pursuit(self.world.agents, delta)
            force.truncate(self.max_force)
            accel = Vector2D(force.x / self.mass, force.y / self.mass)
            self.acceleration = accel
            return accel
        else:
            return super().calculate(delta)
        return super().calculate(delta)

    def pursuit(self, evader, delta):
        ''' this behaviour predicts where an agent will be in time T and seeks
        towards that point to intercept it. '''
        for ev in evader:
            # assumes that evader is a Vehicle
            toEvader = ev.pos - self.pos
            relativeHeading = self.heading.dot(ev.heading)
            # simple out: if target is ahead and facing us, head straight to it
            if ((toEvader.length() - self.radius) < 0):
                if toEvader.length() < 50:
                    ev.tagged = True
                return self.seek(ev.pos)
        return self.wander(delta)

    def wander(self, delta):
        return super().wander(delta)
    def render(self, color = None):
        if self.show_info:
            s = 0.5
            egi.red_pen()
            egi.line_with_arrow(self.pos, self.pos + self.acceleration * s, 5)
        return super().render(color)

    def update(self, delta):
        return super().update(delta)
```

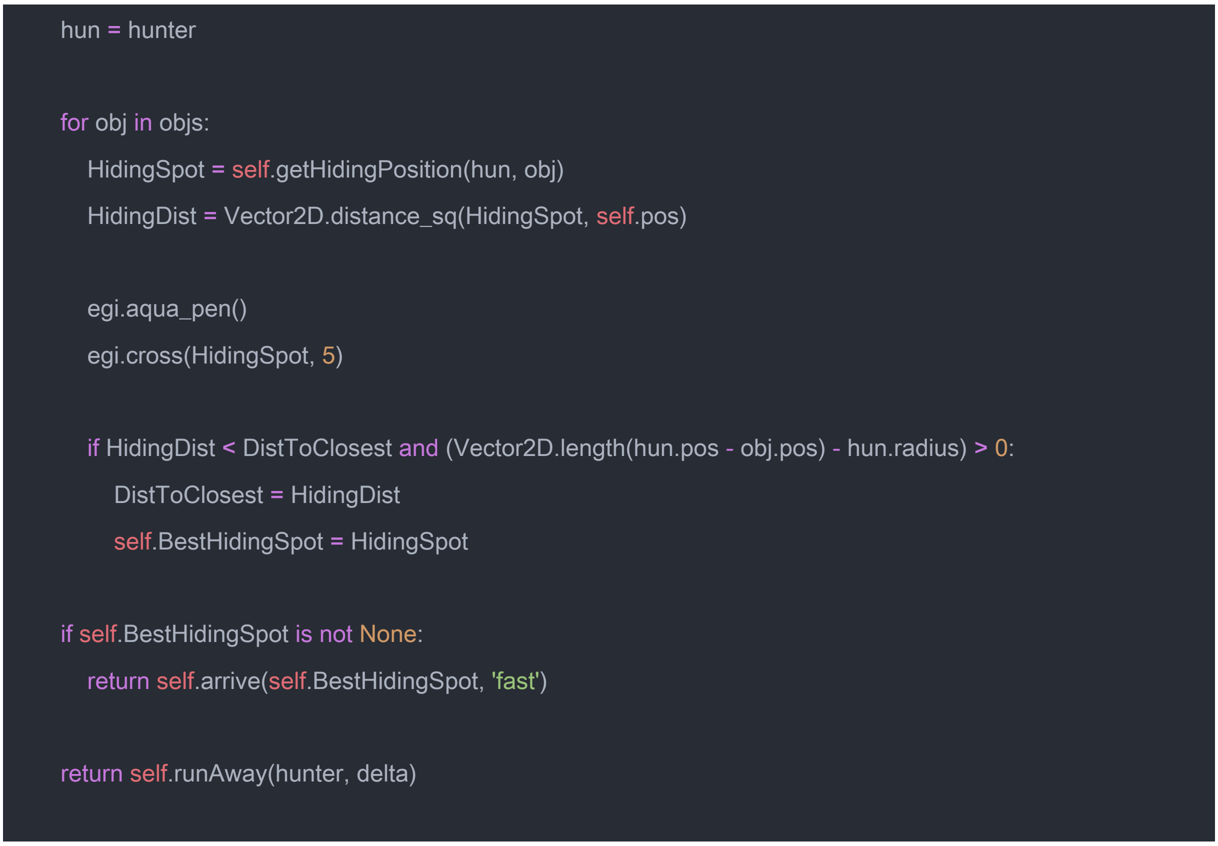
Prey.py

Prey will wander around and try to hide behind object when bing pursuited.

```
from agent import *
from path import Path
from vector2d import Vector2D
from vector2d import Point2D
from graphics import egi, KEY
from math import sin, cos, radians
from random import random, randrange, uniform


class Prey(Agent):

    def __init__(self, world=None, scale=30.0, mass=1.0, mode='hide', looped=True):

        self.world = world
        self.mode = mode

        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)
        self.acceleration = Vector2D()
        self.mass = mass

        self.max_speed = 20.0 * scale / 2
        self.max_force = 500.0

        self.path = Path()
        self.path_looped = looped
        self.randomise_path(looped)
        self.waypoint_threshold = 20

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


        self.BestHidingSpot = None

        self.color = 'GREEN'
        self.vehicle_shape = [
            Point2D(-1.0,  0.7),
            Point2D(1.1,  0.0),
            Point2D(-1.0, -0.5)
        ]

    def calculate(self, delta):
        if self.mode == 'flee':
            force = self.runAway(self.world.hunter, delta)
        elif self.mode == 'hide':
            force = self.hide(self.world.hunter, self.world.hideObjects, delta)
        else:
            force = super().calculate(delta)
        return force

    def runAway(self, pursuer, delta):
        toPursuer = pursuer.pos - self.pos
```



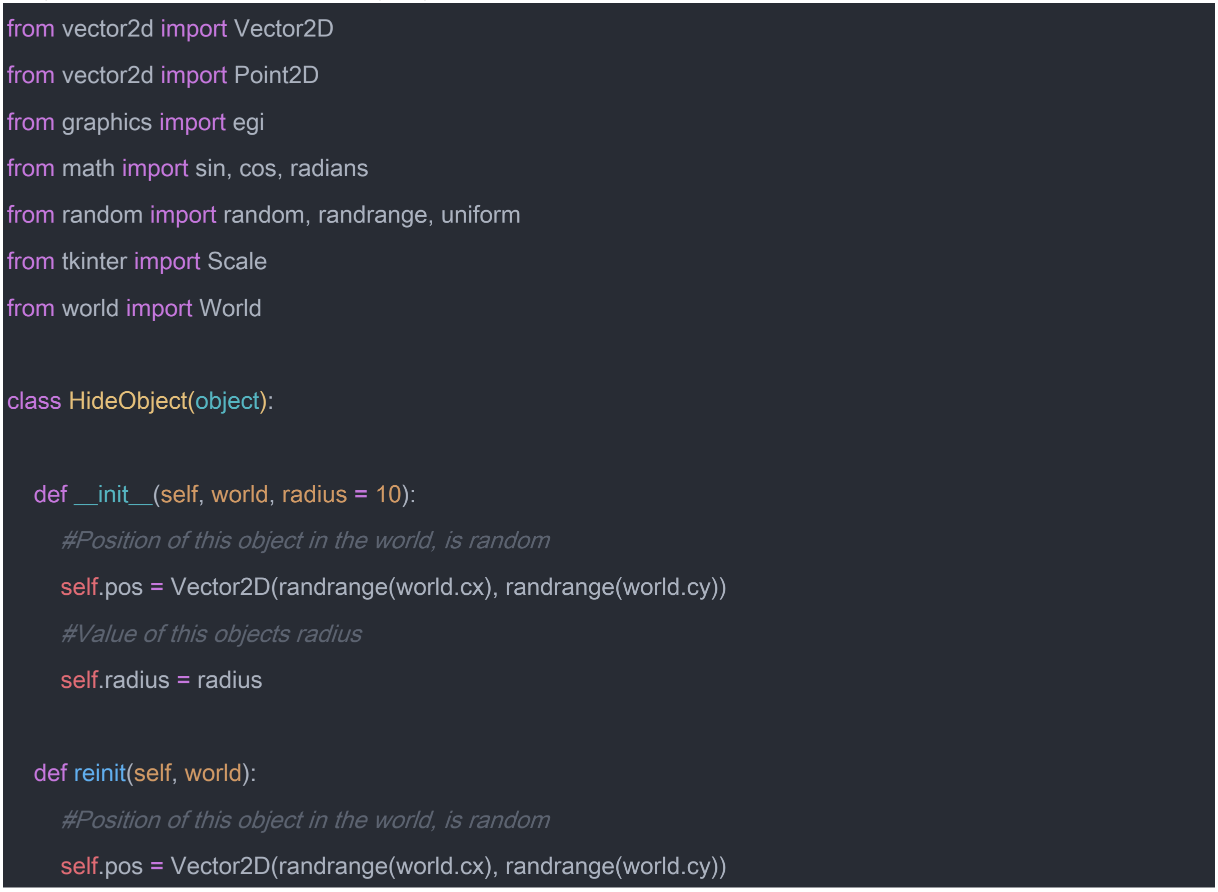
```
        if (toPursuer.length() - pursuer.radius) < -50:

            lookAheadTime = toPursuer.length() / (self.max_speed
                                                 + pursuer.speed())

            return self.flee(pursuer.pos, 'fast', (pursuer.vel * lookAheadTime))

        return self.wander(delta)

    def flee(self, hunter_pos, speed, pursuit_speed):
        ''' move away from hunter position '''

        decel_rate = self.DECELERATION_SPEEDS[speed]
        flee_target = self.pos - hunter_pos
        dist = flee_target.length()
        if dist > 100:
            if AGENT_MODES is 'flee':  #
                speed = dist / decel_rate
                speed = min(speed, self.max_speed)
                desired_vel = flee_target * (speed / dist)
                return (desired_vel - self.vel)
            else:
                pursuit_speed = dist / decel_rate
                pursuit_speed = min(pursuit_speed, self.max_speed)
                desired_vel = flee_target * (pursuit_speed / dist)
                return (desired_vel - self.vel)
        return Vector2D()

    def getHidingPosition(self, hunter, obj):
        DistFromBoundary = 30.0
        DistAway = obj.radius + DistFromBoundary

        ToObj = Vector2D.get_normalised(obj.pos - hunter.pos)

        return (ToObj*DistAway)+obj.pos

    def hide(self, hunter, objs, delta):
        DistToClosest = 1000

        self.BestHidingSpot = None
```



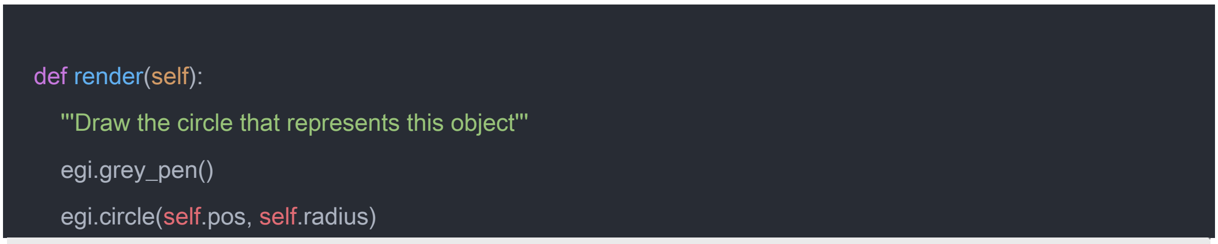
```
        hun = hunter

        for obj in objs:
            HidingSpot = self.getHidingPosition(hun, obj)
            HidingDist = Vector2D.distance_sq(HidingSpot, self.pos)

            egi.aqua_pen()
            egi.cross(HidingSpot, 5)

            if HidingDist < DistToClosest and (Vector2D.length(hun.pos - obj.pos) - hun.radius) > 0:
                DistToClosest = HidingDist
                self.BestHidingSpot = HidingSpot

        if self.BestHidingSpot is not None:
            return self.arrive(self.BestHidingSpot, 'fast')

        return self.runAway(hunter, delta)
```

hideObject.py

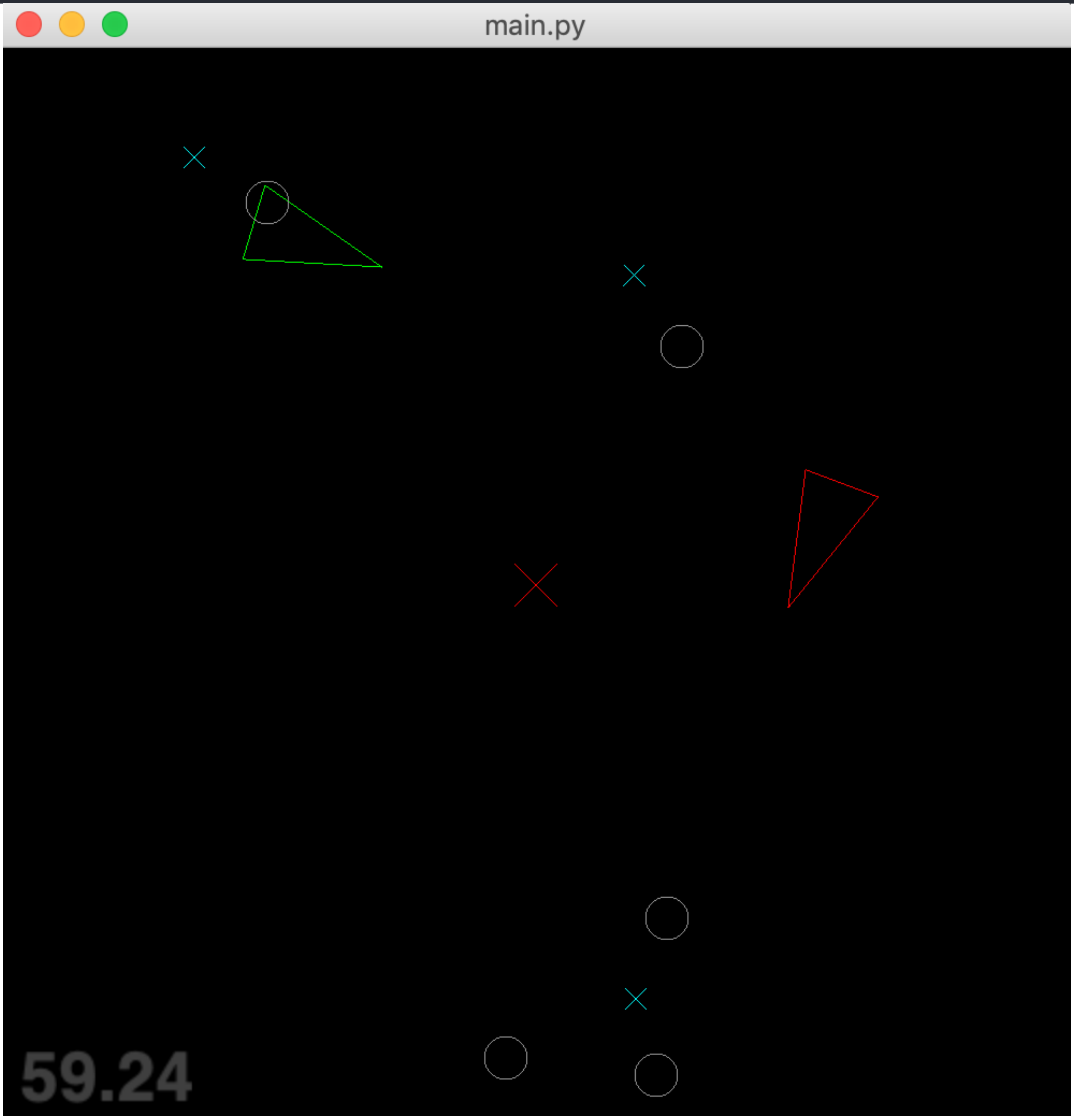
Object created as circles for prey to hide.



```
from vector2d import Vector2D
from vector2d import Point2D
from graphics import egi
from math import sin, cos, radians
from random import random, randrange, uniform
from tkinter import Scale
from world import World

class HideObject(object):

    def __init__(self, world, radius = 10):
        #Position of this object in the world, is random
        self.pos = Vector2D(randrange(world.cx), randrange(world.cy))
        #Value of this objects radius
        self.radius = radius

    def reinit(self, world):
        #Position of this object in the world, is random
        self.pos = Vector2D(randrange(world.cx), randrange(world.cy))
```



```
def render(self):
    '''Draw the circle that represents this object'''
    egi.grey_pen()
    egi.circle(self.pos, self.radius)
```



main.py
59.24