Spike: Hunter Agent
Title: Hunter Agent

Author: Tran Duc Anh Dang | 103995439

Goals / deliverables:

Create a hunter-prey agent simulation for two or more agents, in which "prey" agents avoid "hunter" agents by concealing themselves **behind** objects in the environment. The simulation must:

- Include several "objects" that prey can hide behind (simple circles).
- Show a distinction between the "hunter" and "prey" agent appearance and abilities.
- Show an indicator ("x" or similar) to indicate suitable "hide" locations for prey to select from
- Prey agents must select a “good” location, and head to it, based on tactical evaluation.
- Do NOT hide “inside” objects – rather find a location outside (behind).

Technologies, Tools, and Resources used:

List of information needed by someone trying to reproduce this work

- Python 3+
- Built in Python libraries.
- IDE or Code Editor (Visual Studio Code)

Tasks undertaken:

- Install Python: Download and Install Python 3+ via https://www.python.org/downloads/
- Set up a code editor or IDE: Download and install a python compatible ide or code editor such as Visual Studio Code, PyCharm
- Open and familiarize with the code by reading through, paying attention to the comments that had been made.
- Run the code: Execute the code and observing the output.

What we found out:

world.py

```python
class HideObject(object):
    def __init__(self, world=None, position=None, radius=5, color='WHITE'):
        self.world = world
        self.pos = position if position is not None else self.randomise_loc()
        self.radius = randrange(40,100)
        self.color = color

        self.agents = []

    def update(self):
        self.agents.clear()
        for prey in self.world.get_preys():
            if self.is_inside(prey):
                self.agents.append(prey)

    def randomise_loc(self):
        if self.world is not None:
            x = random.randint(0, self.world.cx)
            y = random.randint(0, self.world.cy)
            return Vector2D(x, y)
        else:
            return Vector2D(0, 0)

    def render(self):
        egi.set_pen_color(name=self.color)
        egi.circle(self.pos, self.radius)
        if (len(self.agents) > 0):
            self.color = 'RED'
        else:
            self.color = 'WHITE'

    def is_inside(self, agent):
        distance_to_agent = (self.pos - agent.pos).length()
        return distance_to_agent <= self.radius

```

Snipped

In this task, I have added HideObject as required, this object will be in circle that will change color once "prey" is trying to hide away from the "hunter". This has been demonstrating in the code above in update function as well as render. Once it is detecting if prey is inside the circle, it will add the list, and when the length of that list is > 0, the circle will change its color to red, otherwise it going back to its default color.

world.py

```python
class World(object):
    def __init__(self, cx, cy):
        self.cx = cx
        self.cy = cy
        self.target = Vector2D(cx / 2, cy / 2)
        self.hide_objects = [HideObject(self, Vector2D(200, 200), 50),
                             HideObject(self, Vector2D(400, 400), 50)]

        self.agents = []
        self.paused = True
        self.show_info = True

    def update(self, delta):
        if not self.paused and self.get_preys():
            for agent in self.agents:
                agent.update(delta)
            # Updates all all hiding positions, (checking for agents hiding at it)
            for obj in self.hide_objects:
                obj.update()

        for obj in self.hide_objects:
            obj.update()

    def render(self):
        if self.target:
            egi.red_pen()
            egi.cross(self.target, 10)

        if self.show_info:
            infotext = ', '.join(set(agent.mode for agent in self.agents))
            egi.white_pen()
            egi.text_at_pos(0, 0, infotext)


        for agent in self.agents:
            agent.render()

        for obj in self.hide_objects:
            obj.render()

    def wrap_around(self, pos):
        ''' Treat world as a toroidal space. Updates parameter object pos '''
        max_x, max_y = self.cx, self.cy
        if pos.x > max_x:
            pos.x = pos.x - max_x
        elif pos.x < 0:
            pos.x = max_x - pos.x
        if pos.y > max_y:
            pos.y = pos.y - max_y
        elif pos.y < 0:
            pos.y = max_y - pos.y

    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

    def transform_point(self, point, pos, forward, side):
        ''' Transform the given single point, using the provided position,
        and direction (forward and side unit vectors), to object world space. '''
        # make a copy of the original point (so we don't trash it)
        wld_pt = point.copy()
        # create a transformation matrix to perform the operations
        mat = Matrix33()
        # rotate
        mat.rotate_by_vectors_update(forward, side)
        # and translate
        mat.translate_update(pos.x, pos.y)
        # now transform the point (in place)
        mat.transform_vector2d(wld_pt)
        # done
        return wld_pt

    def get_hunters(self):
        return [agent for agent in self.agents if agent.mode == "hunter"]

    def get_preys(self):
        return [agent for agent in self.agents if agent.mode == "prey"]
```

Snipped

I have also updated the World class, I added functions get_hunters and get_preys to return a hunter or a prey defined word hunter/prey. And most likely the rest are the same.

agent.py

```
85    def calculate(self, delta):
86            # calculate the current steering force based on the agent's mode
87            mode = self.mode
88            if mode == 'prey':
89                # find the closest hunter agent
90                closest_hunter = self.get_closest_agent('hunter')
91
92                for hide_object in self.world.hide_objects:
93                    hiding_position, _ = self.get_hiding_position_and_distance(hide_object, self.pos)
94                if hiding_position:
95                    force = self.seek(hiding_position)
96                if closest_hunter:
97                    hiding_force = self.hide_behind_object(closest_hunter.pos)
98                    fleeing_force = self.flee(closest_hunter.pos)
99                    force = hiding_force + fleeing_force + self.seek(self.world.target)
100               else:
101                   force = self.seek(self.world.target)
102           elif mode == 'hunter':
103               # find the closest prey agent
104               closest_prey = self.get_closest_agent('prey')
105               if closest_prey:
106                   if self.is_prey_hiding(closest_prey):
107                       force = self.wander(delta)
108                   else:
109                       force = self.seek(closest_prey.pos)
110               else:
111                   force = Vector2D()
112           else:
113               force = Vector2D()
114
115           self.force = force
116           return force
```

Snipped

In agent.py is where the most updated that I have. I have updated calculate function so I make hunter looking for prey to hunt and using wander as required. For the prey, it has 2 jobs, firstly, finding the hiding position, secondly detecting the closest hunter, if there is one it will increasing the hiding and fleeing force to seek for target X that will be marked as identified hiding spot as required.

agent.py

```
188       def get_closest_agent(self, mode):
189           agents = [agent for agent in self.world.agents if agent.mode == mode]
190           if not agents:
191               return None
192           return min(agents, key=lambda obj: (obj.pos - self.pos).length())
193
194       def is_prey_hiding(self, prey):
195           for hide_object in self.world.hide_objects:
196               hiding_position, _ = prey.get_hiding_position_and_distance(hide_object, self.pos)
197               if (prey.pos - hiding_position).length() < self.bRadius:
198                   return True
199           return False
```

Snipped

In this class, I also added 2 functions to find the closest agent (hunter or prey) and a function for hunter to check if the prey is hiding at that location.

Overall, I have completed what has been required to do in this task, there are a lot of things that needs to improve if I work on this task for the future.