

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:

A screenshot of a code editor window titled 'world.py'. The code defines a class 'HideObject' that inherits from 'object'. It has an 'init' method that sets 'world', 'pos', 'radius', and 'color'. It also has 'update', 'randomise_loc', 'render', and 'is_inside' methods. The 'update' method checks for prey and updates the 'agents' list. The 'render' method draws a circle and changes its color to red if there are agents inside. The 'is_inside' method calculates the distance to an agent and checks if it's within the radius. The code is line-numbered from 20 to 54. A 'Snipped' label is at the bottom.

```
20 class HideObject(object):
21     def __init__(self, world=None, position=None, radius=5, color='WHITE'):
22         self.world = world
23         self.pos = position if position is not None else self.randomise_loc()
24         self.radius = randrange(40,100)
25         self.color = color
26
27         self.agents = []
28
29     def update(self):
30         self.agents.clear()
31         for prey in self.world.get_preys():
32             if self.is_inside(prey):
33                 self.agents.append(prey)
34
35     def randomise_loc(self):
36         if self.world is not None:
37             x = random.randint(0, self.world.cx)
38             y = random.randint(0, self.world.cy)
39             return Vector2D(x, y)
40         else:
41             return Vector2D(0, 0)
42
43     def render(self):
44         egi.set_pen_color(name=self.color)
45         egi.circle(self.pos, self.radius)
46         if (len(self.agents) > 0):
47             self.color = 'RED'
48         else:
49             self.color = 'WHITE'
50
51     def is_inside(self, agent):
52         distance_to_agent = (self.pos - agent.pos).length()
53         return distance_to_agent <= self.radius
54
```

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

56 class World(object):
57     def __init__(self, cx, cy):
58         self.cx = cx
59         self.cy = cy
60         self.target = Vector2D(cx / 2, cy / 2)
61         self.hide_objects = [HideObject(self, Vector2D(200, 200), 50),
62                               HideObject(self, Vector2D(400, 400), 50)]
63
64         self.agents = []
65         self.paused = True
66         self.show_info = True
67
68     def update(self, delta):
69         if not self.paused and self.get_preys():
70             for agent in self.agents:
71                 agent.update(delta)
72             # Updates all all hiding positions, (checking for agents hiding at it)
73             for obj in self.hide_objects:
74                 obj.update()
75
76         for obj in self.hide_objects:
77             obj.update()
78
79     def render(self):
80         if self.target:
81             egi.red_pen()
82             egi.cross(self.target, 10)
83
84         if self.show_info:
85             infotext = ', '.join(set(agent.mode for agent in self.agents))
86             egi.white_pen()
87             egi.text_at_pos(0, 0, infotext)
88
89         for agent in self.agents:
90             agent.render()
91
92         for obj in self.hide_objects:
93             obj.render()
94
95     def wrap_around(self, pos):
96         ''' Treat world as a toroidal space. Updates parameter object pos '''
97         max_x, max_y = self.cx, self.cy
98         if pos.x > max_x:
99             pos.x = pos.x - max_x
100         elif pos.x < 0:
101             pos.x = max_x - pos.x
102         if pos.y > max_y:
103             pos.y = pos.y - max_y
104         elif pos.y < 0:
105             pos.y = max_y - pos.y
106
107     def transform_points(self, points, pos, forward, side, scale):
108         ''' Transform the given list of points, using the provided position,
109             direction and scale, to object world space. '''
110         # make a copy of original points (so we don't trash them)
111         wld_pts = [pt.copy() for pt in points]
112         # create a transformation matrix to perform the operations
113         mat = Matrix33()
114         # scale
115         mat.scale_update(scale.x, scale.y)
116         # rotate
117         mat.rotate_by_vectors_update(forward, side)
118         # and translate
119         mat.translate_update(pos.x, pos.y)
120         # now transform all the points (vertices)
121         mat.transform_vector2d_list(wld_pts)
122         # done
123         return wld_pts
124
125     def transform_point(self, point, pos, forward, side):
126         ''' Transform the given single point, using the provided position,
127             and direction (forward and side unit vectors), to object world space. '''
128         # make a copy of the original point (so we don't trash it)
129         wld_pt = point.copy()
130         # create a transformation matrix to perform the operations
131         mat = Matrix33()
132         # rotate
133         mat.rotate_by_vectors_update(forward, side)
134         # and translate
135         mat.translate_update(pos.x, pos.y)
136         # now transform the point (in place)
137         mat.transform_vector2d(wld_pt)
138         # done
139         return wld_pt
140
141     def get_hunters(self):
142         return [agent for agent in self.agents if agent.mode == "hunter"]
143
144     def get_preys(self):
145         return [agent for agent in self.agents if agent.mode == "prey"]
```

Snipped

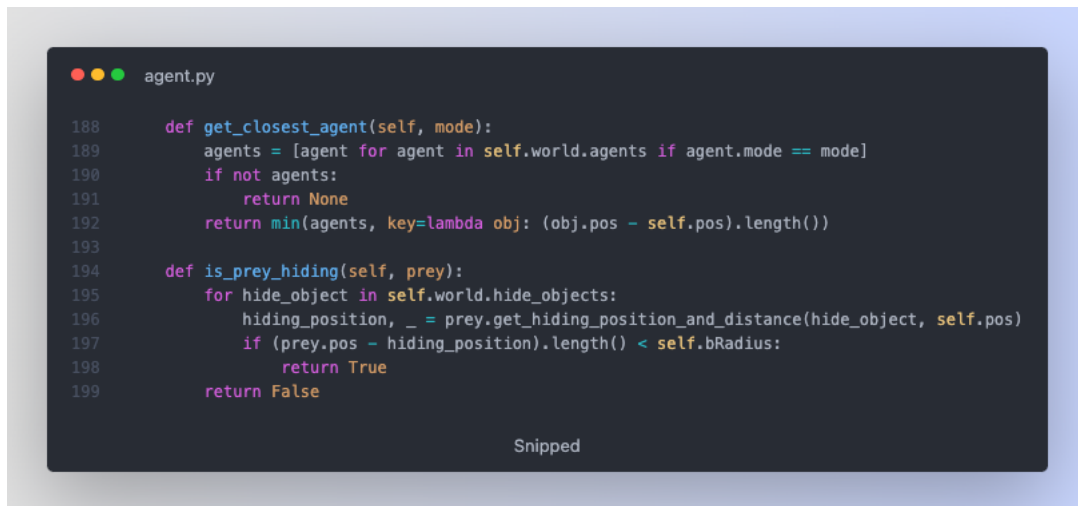
I have also updated the World class, I added functions get_hunters and get_preys to return a hunter or a prey defined word hunter/prey. And most likely the rest are the same.

```
agent.py

86 def calculate(self, delta):
87     # calculate the current steering force based on the agent's mode
88     mode = self.mode
89     if mode == 'prey':
90         # find the closest hunter agent
91         closest_hunter = self.get_closest_agent('hunter')
92
93         # Find the furthest hide object from the hunter
94         furthest_hide_obj = self.find_furthest_hide_object_from_agent(closest_hunter)
95         if furthest_hide_obj:
96             hiding_position, _ = self.get_hiding_position_and_distance(furthest_hide_obj, self.pos)
97         else:
98             hiding_position = None
99
100        if hiding_position and closest_hunter:
101            hiding_force = self.hide_behind_object(closest_hunter.pos)
102            fleeing_force = self.flee(closest_hunter.pos)
103            force = hiding_force + fleeing_force + self.seek(hiding_position)
104        elif hiding_position:
105            force = self.seek(hiding_position)
106        else:
107            force = Vector2D()
108    elif mode == 'hunter':
109        # find the closest prey agent
110        closest_pre = self.get_closest_agent('prey')
111        if closest_pre:
112            if self.is_pre_hiding(closest_pre):
113                # Calculate a force to avoid the hiding object
114                force = self.avoid_objects()
115            else:
116                force = self.seek(closest_pre.pos)
117        else:
118            force = Vector2D()
119    else:
120        force = Vector2D()
121
122    self.force = force
123    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 hiding object that furthest away from hunter that will be marked as identified hiding spot as required.



```
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_preying_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.