Spike: 13
Title: Tactical Steering (Hide!)

Author: Luke Valentino, 103024456

Goals / deliverables:
The goal of this spike is to use tactical analysis of an environment to give an agent the ability to find and seek good hiding locations.

Technologies, Tools, and Resources used:
List of information needed by someone trying to reproduce this work
- Swinburne Lecture Materials
- Swinburne Maths and Physics PDF
- Docs.python.org

Tasks undertaken:
To create a hunter-prey simulation the first step copy over old code as a base. Then,
- Create an object class that renders a circle.

```
class Object(object):
    def __init__(self, position, radius):
        self.position = position
        self.radius = radius

    def render(self):
        # Code to draw the object as a circle
        egi.circle(self.position, self.radius)
```

- Now that you have an object class, we need to render it within the world.
    - The first step is to create a list of objects in our world class.
    - Then we can update the world render method to iterate over that objects list, and call each objects render method.

```
for obj in self.objects:
    obj.render()
```

    - Now we can instantiate an object by appending it to the worlds list as so.

```
world.objects.append(Object(Vector2D(150,150), 20))
```

- We can now move on to creating our hunter and prey agents.
    - The first step is to ensure that our world has hunter and prey attributes

```
        self.hunter = None
        self.prey = None
```

- From there we can create our hunter and prey subclasses.

```
- class Hunter(Agent):
-     def __init__(self, world=None, scale=30.0, mass=1.0):
-         super(Hunter, self).__init__(world, scale, mass, mode='wander')
-         self.color = 'RED'
```

```
class Prey(Agent):
    def __init__(self, world=None, scale=30.0, mass=1.0):
        super(Prey, self).__init__(world, scale, mass, mode='hide')
        self.color = 'BLUE'
```

- Then we instantiate these classes in our world.

```
hunter = Hunter(world)
world.agents.append(hunter)
world.hunter = hunter  # Set the hunter in the world for prey to access

# Add a prey
prey = Prey(world)
world.agents.append(prey)
```

- We need to determine our hiding spots for the prey to path to. To do that we create a GetHidingPosition method in world.

```
-     def GetHidingPosition(self, hunter_pos, obj_pos, obj_radius):
-         # set the distance between the object and the hiding point
-         DistFromBoundary = 30.0
-         DistAway = obj_radius + DistFromBoundary
-
-         ToObj = (obj_pos - hunter_pos).normalise()
-
-         hiding_position = (ToObj * DistAway) + obj_pos
-         return hiding_position
-
```

- To see if our hiding spots are working we can adjust the world render method to draw lines from our hunter to hiding spot.

```
-         if self.hunter:
-             hunter_pos = self.hunter.pos
-             for obj in self.objects:
-                 hiding_spot = self.GetHidingPosition(hunter_pos,
obj.position, obj.radius)
-
-                 obj.render()
-
-                 # draw line from hunter to object
-                 egi.blue_pen()
-                 egi.line(hunter_pos.x, hunter_pos.y, obj.position.x,
obj.position.y)
-
```

```
-                   # draw line from object to hiding spot
-                   egi.line(obj.position.x, obj.position.y, hiding_spot.x,
   hiding_spot.y)
-
-                   # mark the hiding spot with an X
-                   egi.cross(hiding_spot, 5)
```

- The last thing to do is to make sure our prey object can move to the hiding spots. To do this, we create an evade method in agent.py:

```
-       def evade(self, pursuer):
-           toPursuer = pursuer.pos - self.pos
-           lookAheadTime = toPursuer.length() / (self.max_speed +
   pursuer.speed())
-           futurePosition = pursuer.pos + pursuer.vel * lookAheadTime
-           return self.flee(futurePosition)
-
```

- Then we implement a hide method in the prey subclass.

```
-       def Hide(self, hunter, objs):
-           DistToClosest = float('inf')
-           BestHidingSpot = None
-           # check for possible hiding spots
-           for obj in objs:
-               HidingSpot = self.world.GetHidingPosition(hunter.pos,
   obj.position, obj.radius)
-               HidingDist = (HidingSpot - self.pos).lengthSq()
-               if HidingDist < DistToClosest:
-                   DistToClosest = HidingDist
-                   BestHidingSpot = HidingSpot
-           # if we have a best hiding spot, use it
-           if BestHidingSpot:
-               return self.arrive(BestHidingSpot, 'fast')  # Use arrive behavior
   to move to the hiding spot
-           # default - run away!
-           return self.evade(hunter)
```

-
- Lastly we need to be able to switch the prey to its hide mode, to do that we add evade as an agent mode and make a calculate method in the prey subclass to determine the current mode.

```
    def calculate(self, delta):
        mode = self.mode
        if mode == 'hide':
            force = self.Hide(self.world.hunter, self.world.objects)
        else:
            force = super(Prey, self).calculate(delta)
```

```
        self.force = force
        return force
```

What we found out:
The goal of this spike was to allow an agent to find and use hiding spot using tactical analysis. This was implemented successfully.

Key Outcomes:
Hiding Spot Calculations:
- Implemented GetHidingPosition method to determine the best hiding spot for the prey.

Evade and Hide Functionalities:
- Implemented the evade and hide methods to allows the prey to find its way to hiding spots or evade the hunter.

Behaviour Switching:
- Implemented functionality to switch the prey between hiding and evading modes based on the situation.

This spike relates to the following ULO's:

1. Discuss and implement software development techniques to support the creation of AI.
2. Understand and utilize a variety of graph and path planning techniques.

3. Create realistic movement for agents using steering force models.

5. Combine AI techniques to create more advanced game AI.