Spike: Spike_8
Title: Tactical Steering

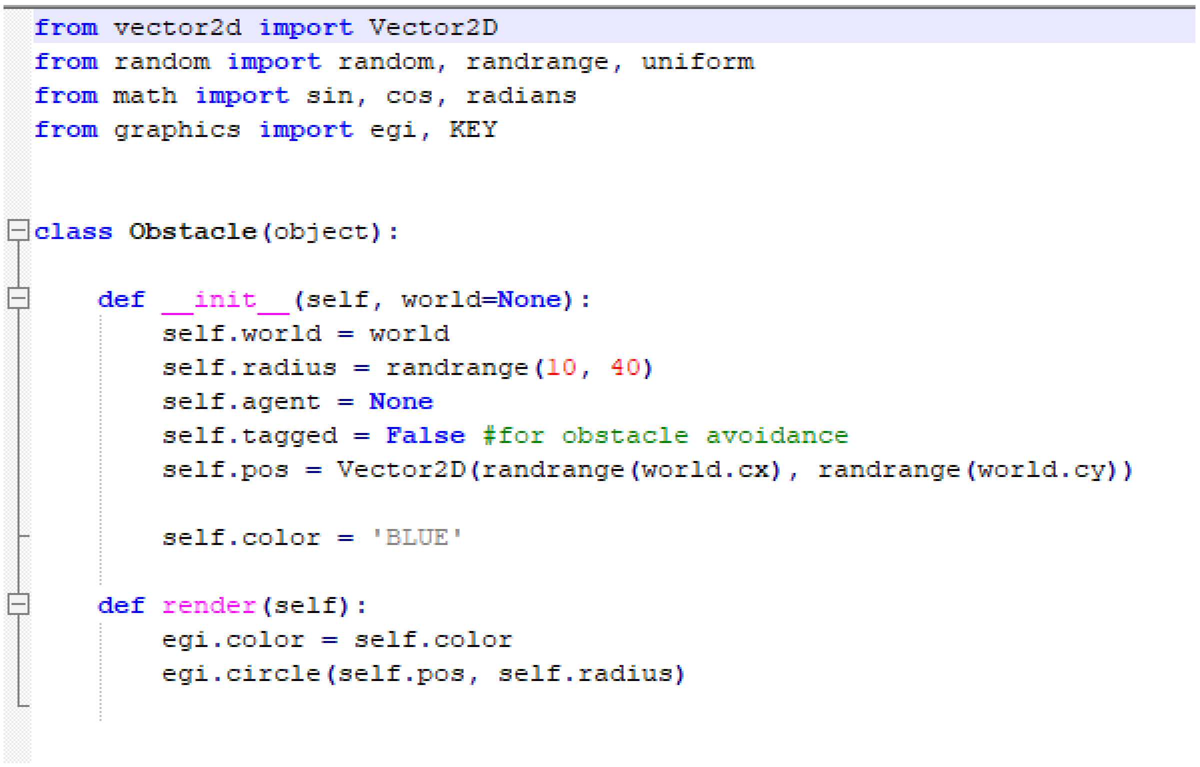
Author: Daniel Newman, 6449921

Goals / deliverables:
The goal of this spike is to extend new behaviour into our autonomous boids to facilitate tactical analysis to support the goal of hiding from a predator

Repo Link:

Technologies, Tools, and Resources used:
- Sublime Text 3 or equivalent Python supported IDE
- Python 3.6.4

Tasks undertaken:
We need to generate some obstacles for our boids to hide behind, so we created a class to comprise the objects we need to generate. This is our Obstacle class.



```python
from vector2d import Vector2D
from random import random, randrange, uniform
from math import sin, cos, radians
from graphics import egi, KEY


class Obstacle(object):

    def __init__(self, world=None):
        self.world = world
        self.radius = randrange(10, 40)
        self.agent = None
        self.tagged = False #for obstacle avoidance
        self.pos = Vector2D(randrange(world.cx), randrange(world.cy))

        self.color = 'BLUE'

    def render(self):
        egi.color = self.color
        egi.circle(self.pos, self.radius)
```

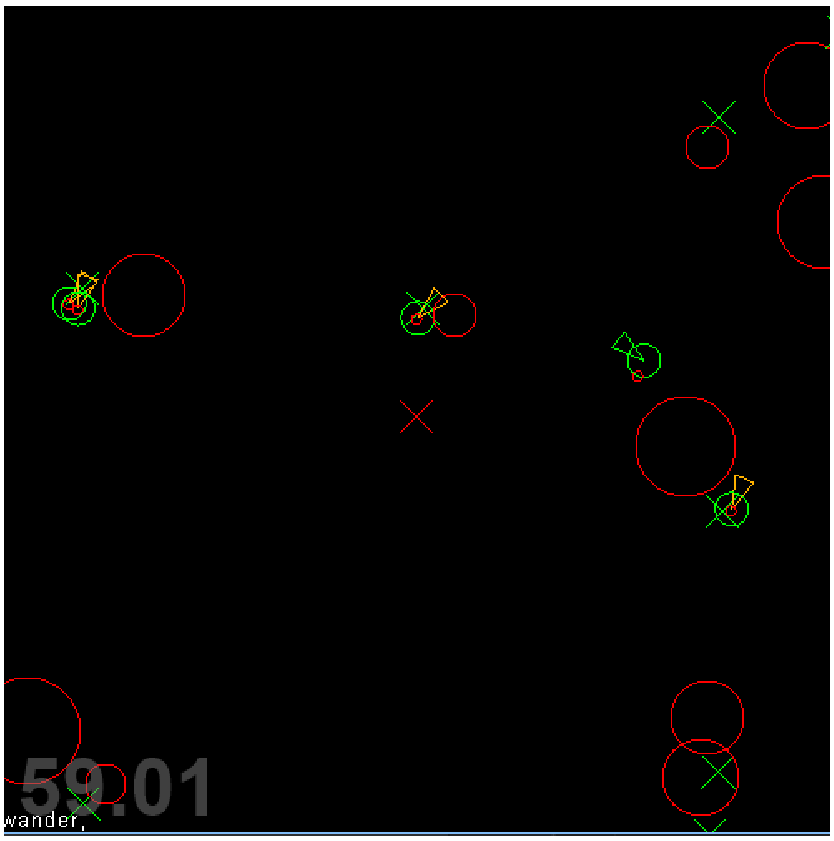
The obstacle class is a very simple class, it doesn't account for overlapping spheres but for the purposes of this spike it is sufficient.

Now the world class must contain an array of obstacle objects for our agents to take advantage of. The world class also holds the agent that will function as our predator.

This brings us to the agents hide function. This function is responsible for the behaviour of identifying the best hiding spot from our list of obstacles and moving towards that spot.

```
def hide(self, hunter_pos, delta):
    if self.world.hunter is self:
        return self.wander(delta)

    best_hiding_spot = None
    best_hiding_dist = 99999999999

    for obs in self.world.obstacles:
        boundary_dist = obs.radius + (obs.radius/2)
        hiding_spot = obs.pos + Vector2D.get_normalised(obs.pos - hunter_pos) * boundary_dist
        hiding_dist = Vector2D.distance(self.pos, hiding_spot)**2

        if hiding_dist < best_hiding_dist:
            best_hiding_spot = hiding_spot
            best_hiding_dist = hiding_dist
        egi.green_pen()
        egi.cross(hiding_spot,10)

    return self.arrive(best_hiding_spot, "fast")
```

In order to achieve this behaviour, we loop through all of our obstacles and determine which obstacle is the shortest distance away from the agent and place a positional marker that puts the obstacle between the agent and the predator. This code gives us this behaviour, the predator is displayed in green.



What we found out:
We have been able to extend our autonomous behaviour to model a simplified prey/predator environment. We did this by using a simplified analysis of the environment around us, determining which obstacle would be the best for each agent to move towards to keep the predator away from the prey.