Spike: Spike_14
Title: Soldier on Patrol

Author: Daniel Newman, 6449921

Goals / deliverables:
Using a hierarchical finite state machine (HSM), seperate out high level behaviours with low level implementation to demonstrate layering fsm on top of each other to create new behaviours

Technologies, Tools, and Resources used:
List of information needed by someone trying to reproduce this work

- Python 3.6.4
- Python supported IDE

Tasks undertaken:
In order to create the HSM, we used the code from our Hunter/Prey scenario as our baseline. The hunter was then extended to include two new states, attacking and patrolling, each with their own transitional data and behaviours.

```
#States
self.mode_attack = 'Firing'
self.mode_patrol = 'Walking'
```

We then extended our update loop to incorporate the different states.

These included the creation of two new functions, patrol & attack. Each would handle the low-level behaviour required to perform their function, whether that is hunting down new prey or patrolling its set waypoints.

The patrol code transitions between Idle and walking states for when its heading towards its next waypoint and upon reaching its target waypoint. The state doesn't handle transition data between patrol and attack, that is done in the hunters update loop.

```
def patrol(self):
    if(self.mode_patrol == 'Idle' and self.last_idle > 1):
        self.mode_patrol = 'Walking'

    if self.mode_patrol == 'Walking':

        direction = (self.waypoints[self.current_waypoint] - self.pos).normalise()
        self.pos += direction * self.speed

        if self.pos.distance_sq(self.waypoints[self.current_waypoint]) < 25:
            self.next_waypoint()
            self.last_idle = time.time()
            self.mode_patrol = 'Idle'
```

The attack code is responsible for firing on prey within its view radius. The firing code is essentially the same as the Hunter/Prey scenario.

```
def attack(self, target):
    if self.mode_attack == 'Reloading' and time.time() - self.last_fire > GUN_COOLDOWNS[self.gun.mode]:
        self.mode_attack = 'Firing'
    if self.mode_attack == 'Firing':
        self.gun.fire(target)
        self.last_fire = time.time()
        self.mode_attack = 'Reloading'
```

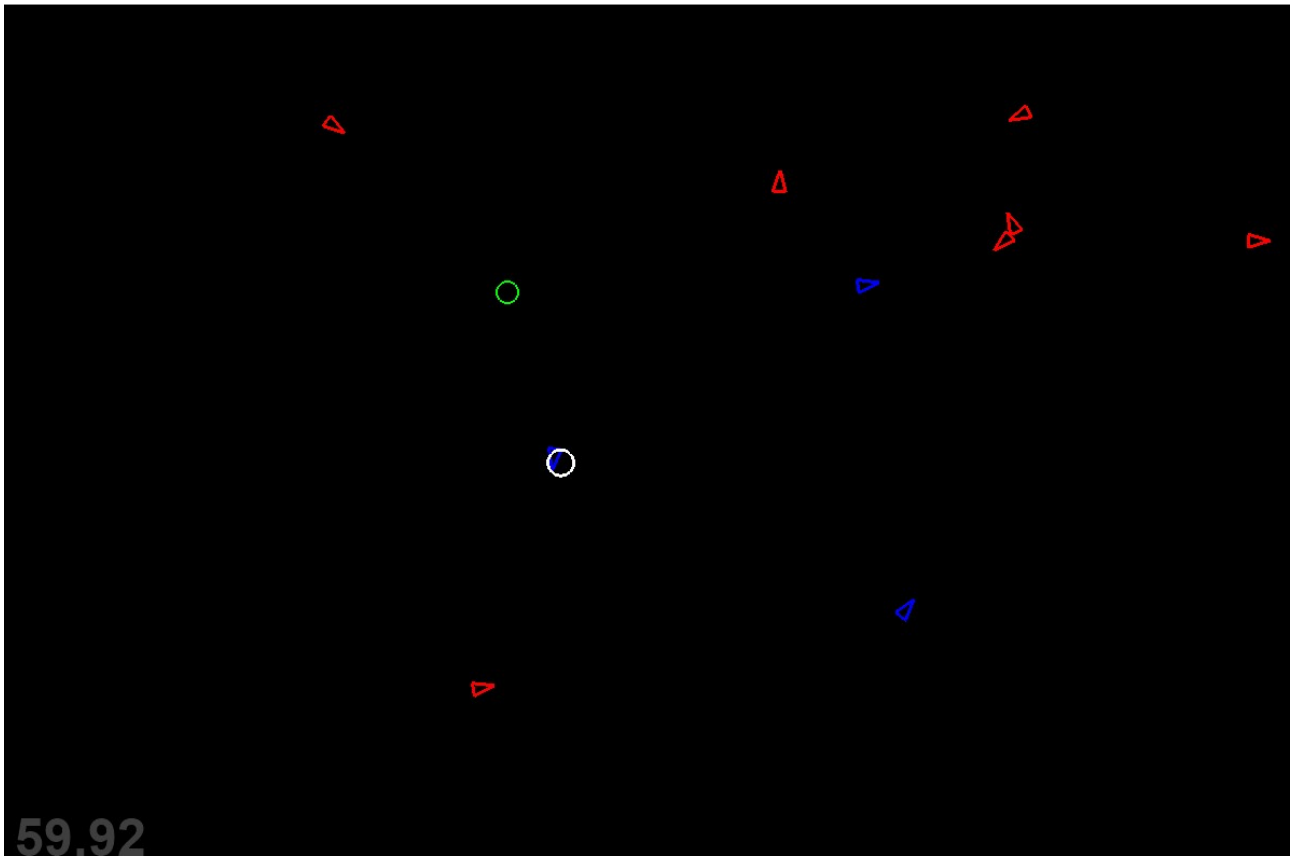
The update loop determines which behaviour to transition between based upon a check for any targets within range. The perform function just makes a function call to the respective function based upon the agents mode.

```
def update(self, delta):
    target = self.find_prey(200)
    self.gun.update_firing_pos(self.pos)
    if self.mode == 'Patrol' and target is not None:
        self.mode = 'Attack'
    elif self.mode == 'Attack' and target is None:
        self.mode = 'Patrol'

    self.perform(self.mode, target)
```

What we found out:

Describe the outcomes, and how they relate to the spike topic + graphs/screenshots/outputs as needed



We uncovered that layering FSM behaviour allowed us to group together related behaviours in a way that made implementation easier to work with without having to resort to large state machines with similar transition events. This allowed us to create more dynamic behaviour for our bot without increasing code complexity.