Spike: Spike_5
Title: Soldier on Patrol

Author: Le Bao Duy Nguyen 102449993

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:

- Visual Studio Code
- Python 3.0+

Tasks undertaken:

- Download and install Visual Studio Code
- Download and install Python 3.0 & above
- Download and install Python extension within Visual Studio Code
- Read the codes and guidelines. Researching Canvas materials as well as Google, YouTube, etc.

The hunter was extended to two states, attacking and patrolling, each with their own transitional data and behaviours.

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

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.

Patrol function transitions between Idle and walking states for when its heading towards its next waypoint and upon reaching its target waypoint.

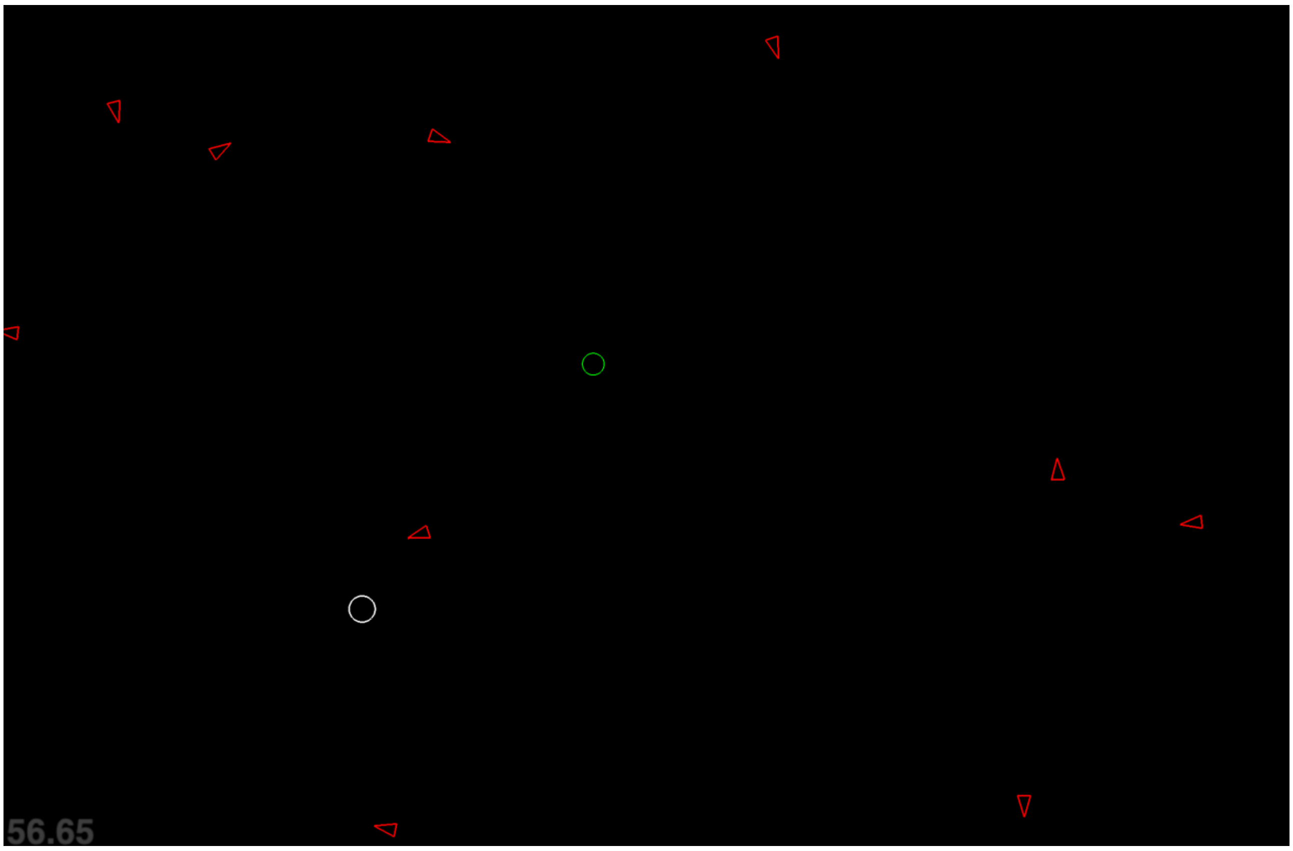
Attack function is responsible for firing on prey within its view radius.

```
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'


    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 of 200. The perform function just makes a function call to the respective function based upon the agent's 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:



Layering FSM behaviour allowed us to group related behaviours in order to make implementation easier to work with without maintaining large state machines with similar transition events.