Goals:

Create a “soldier on patrol” simulation where an agent has two or more high-level FSM modes of behaviour and low-level FSM behaviour. The model must show (minimum)

- High level "patrol" and "attack" modes
- The "patrol" mode must use a FSM to control low-level states so that the agent will visit (seek/arrive?) a number of patrol-path way points.
- The "attack" mode must use a FSM to control low-level fighting states. (Think “shooting”, “reloading” – the actual states and transition rules are up to you.)

Technologies, Tools, Resources used:

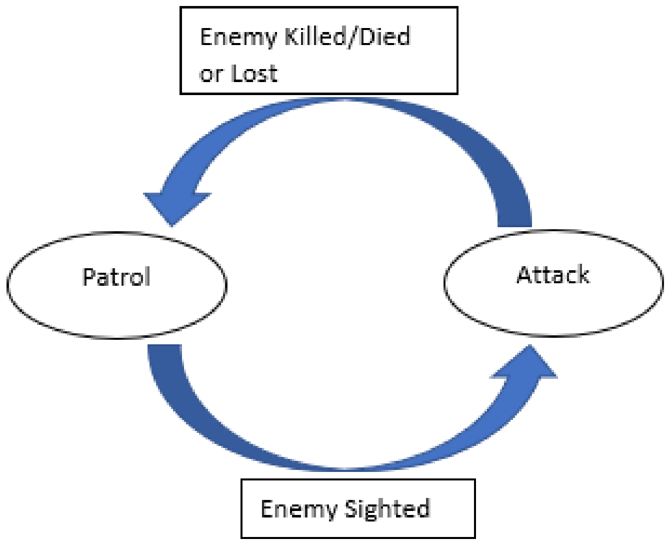
Technologies, Tools, and Resources used:

- Python IDE(PyCharm) with python 3 installed
- Piglet Documentation http://pyglet.readthedocs.io/en/pyglet-1-3-maintenance/
- Python 3 Documentation http://docs.python.org/
- The code from previous lab task
- Help from peers.
- YouTube Videos (Reference at the bottom)

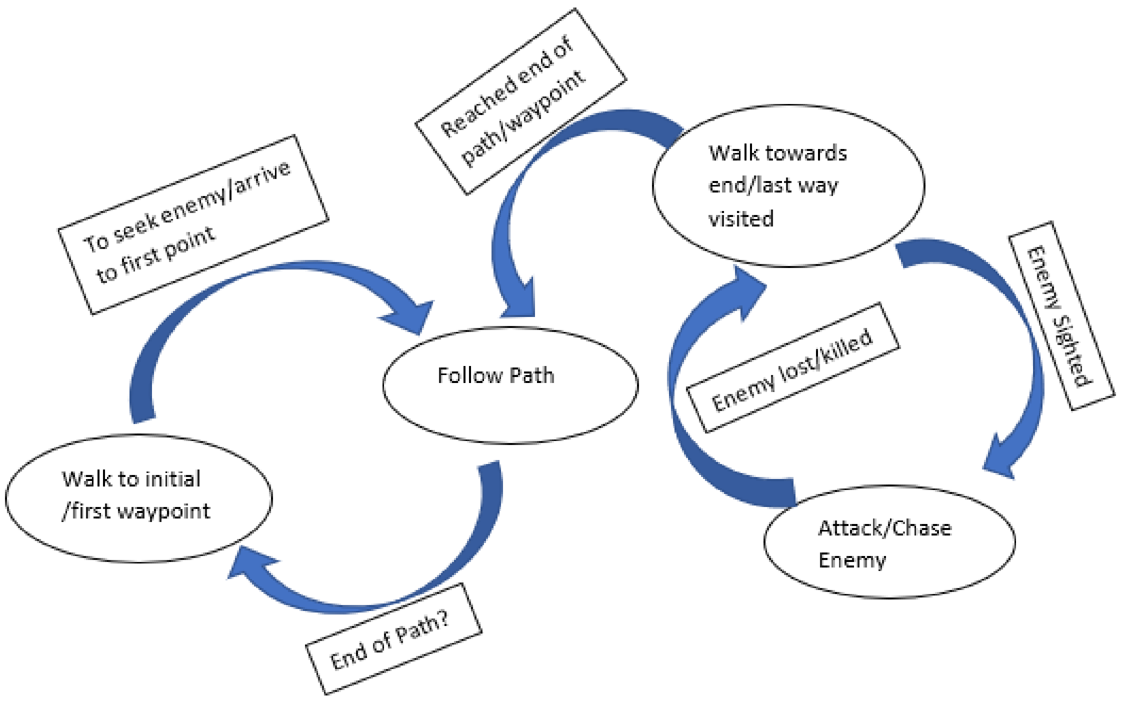
Tasks Undertaken:

1. Creating FSM Diagrams based on the code implementation and outcomes

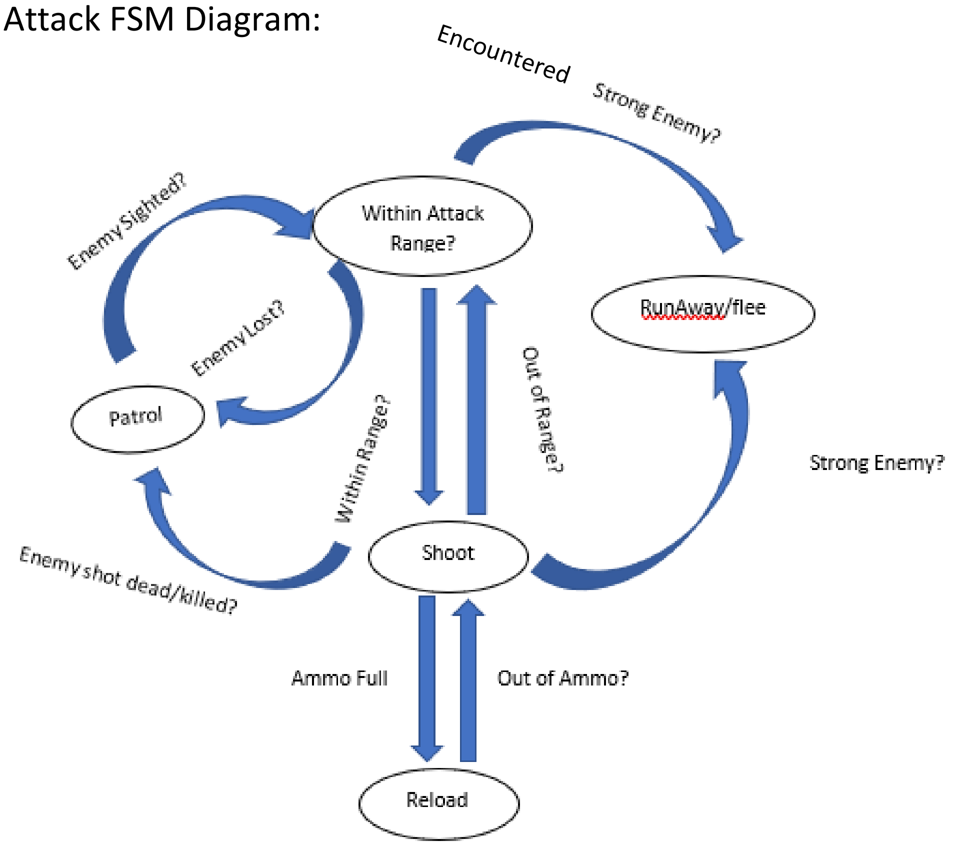
High Level FSM Diagram:



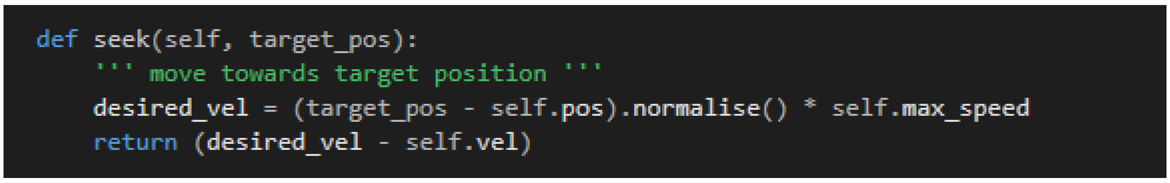
Patrol FSM Diagram:


To seek enemy/arrive to first point
Reached end of path/waypoint
Walk towards end/last way visited
Follow Path
Enemy lost/killed
Enemy Sighted
Walk to initial /first waypoint
Attack/Chase Enemy
End of Path?

Attack FSM Diagram:

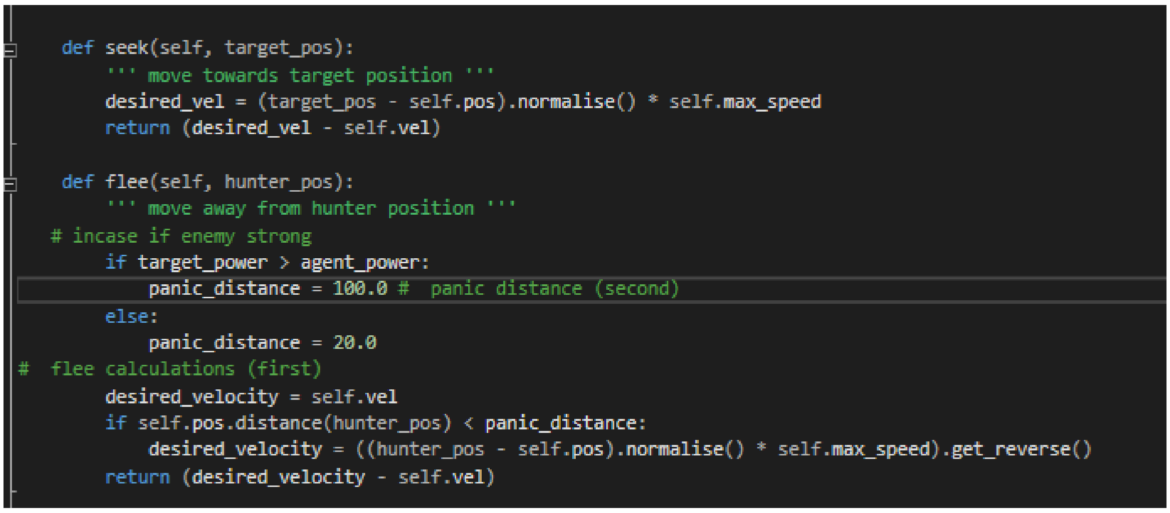


2. Some Code Snips for methods with comments to make it easier to reimplement it:
   - Seek Target based on the current position of agent and the target and close in to target at velocity.



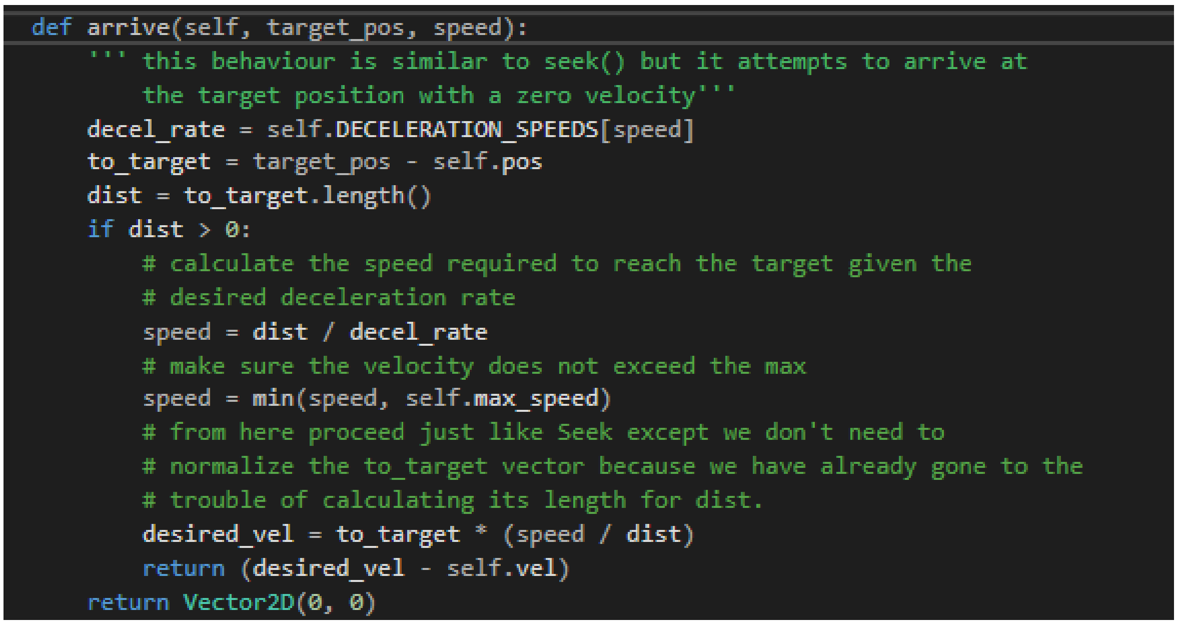
```python
def seek(self, target_pos):
    ''' move towards target position '''
    desired_vel = (target_pos - self.pos).normalise() * self.max_speed
    return (desired_vel - self.vel)
```

   - Flee from target if he is stronger than the agent and flee and used panic distance to calculate the velocity in order to flee.



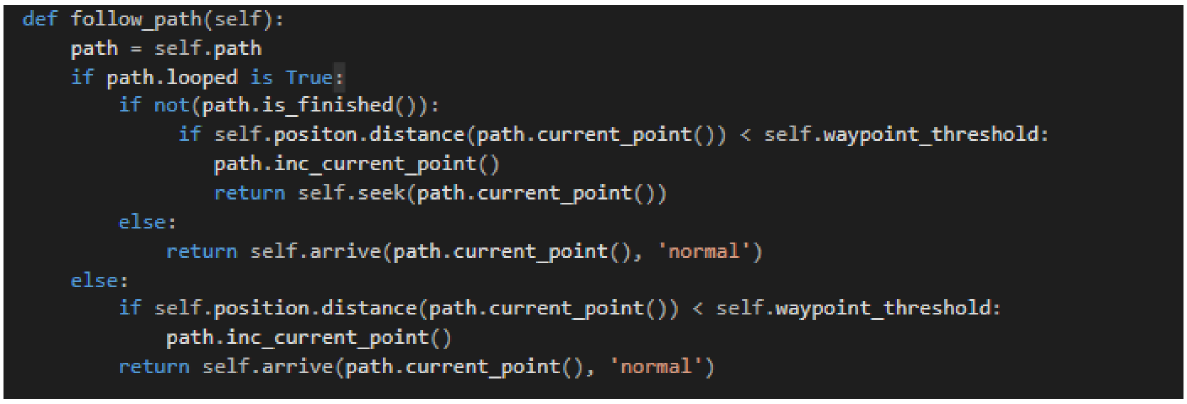
```python
    def seek(self, target_pos):
        ''' move towards target position '''
        desired_vel = (target_pos - self.pos).normalise() * self.max_speed
        return (desired_vel - self.vel)

    def flee(self, hunter_pos):
        ''' move away from hunter position '''
   # incase if enemy strong
        if target_power > agent_power:
            panic_distance = 100.0 #  panic distance (second)
        else:
            panic_distance = 20.0
#  flee calculations (first)
        desired_velocity = self.vel
        if self.pos.distance(hunter_pos) < panic_distance:
            desired_velocity = ((hunter_pos - self.pos).normalise() * self.max_speed).get_reverse()
        return (desired_velocity - self.vel)
```

   - Arrive at position of target since we’re not fleeing from the target the velocity is kept lower than the max.



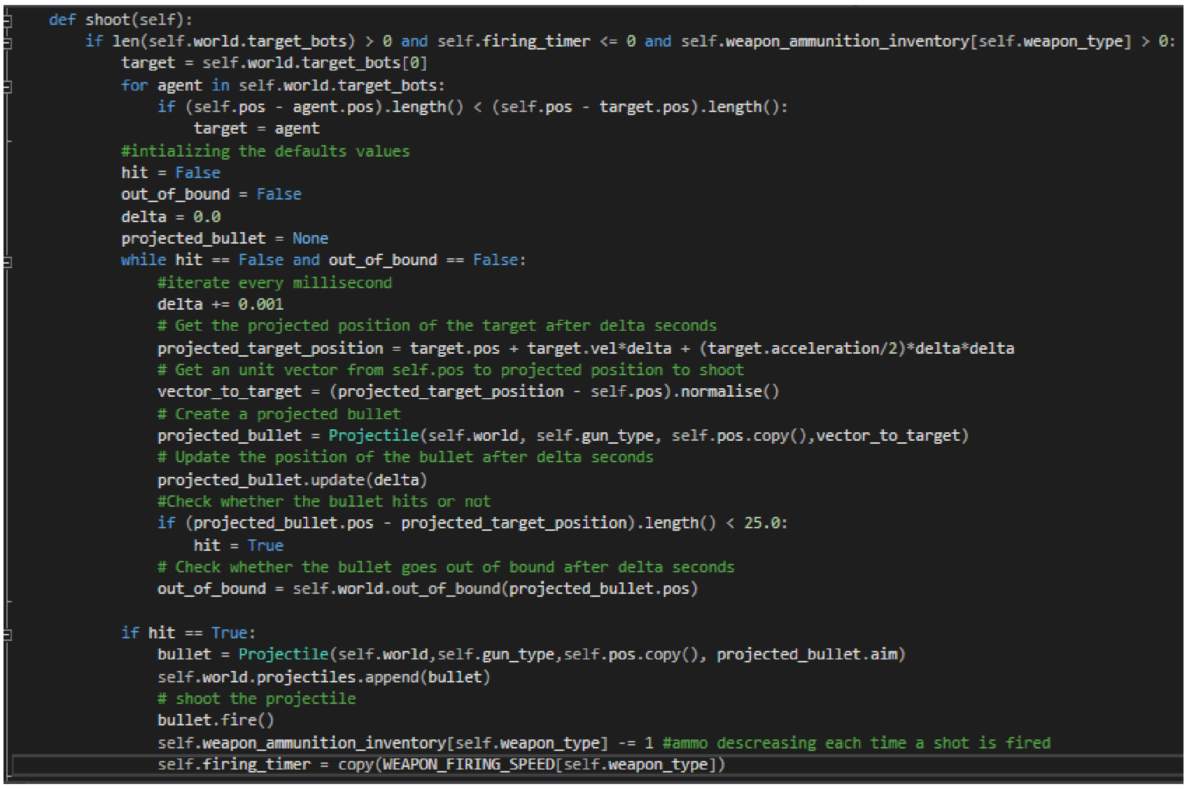
```python
def arrive(self, target_pos, speed):
    ''' this behaviour is similar to seek() but it attempts to arrive at
        the target position with a zero velocity'''
    decel_rate = self.DECELERATION_SPEEDS[speed]
    to_target = target_pos - self.pos
    dist = to_target.length()
    if dist > 0:
        # calculate the speed required to reach the target given the
        # desired deceleration rate
        speed = dist / decel_rate
        # make sure the velocity does not exceed the max
        speed = min(speed, self.max_speed)
        # from here proceed just like Seek except we don't need to
        # normalize the to_target vector because we have already gone to the
        # trouble of calculating its length for dist.
        desired_vel = to_target * (speed / dist)
        return (desired_vel - self.vel)
    return Vector2D(0, 0)
```

- Follow path to way points from initial to end of waypoint.



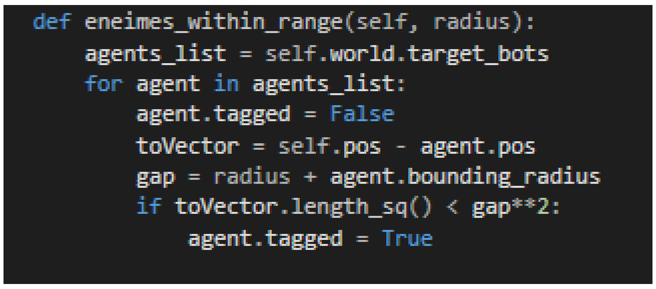
```python
def follow_path(self):
    path = self.path
    if path.looped is True:
        if not(path.is_finished()):
             if self.positon.distance(path.current_point()) < self.waypoint_threshold:
                path.inc_current_point()
                return self.seek(path.current_point())
        else:
            return self.arrive(path.current_point(), 'normal')
    else:
        if self.position.distance(path.current_point()) < self.waypoint_threshold:
            path.inc_current_point()
        return self.arrive(path.current_point(), 'normal')
```

-Shoot target based on the distance calculate it’s the shot fired speed and the projectile it should follow to hit the target what weapon was selected to fire the shot and if the actual shot has been fired if yes decrease the ammo count from the weapon ammunition inventory reminder it’s based on the weapon type itself so each weapon has its own ammunition inventory.



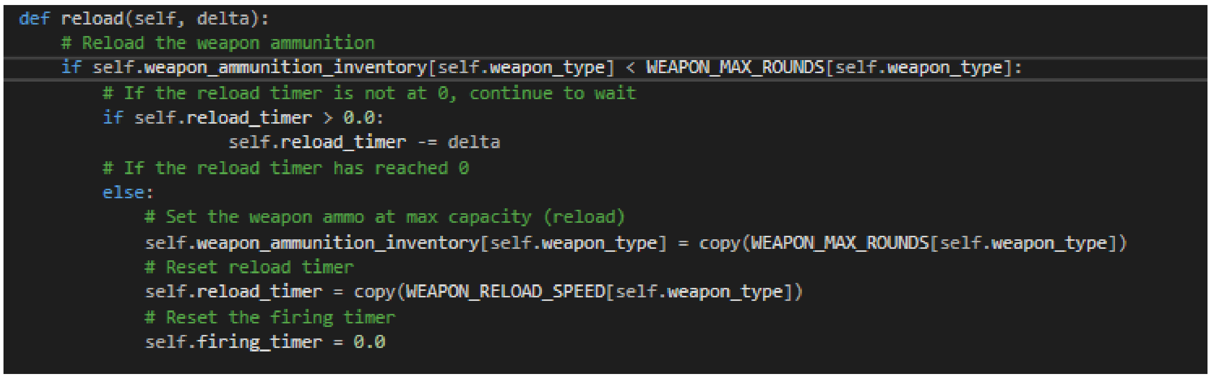
```python
def shoot(self):
    if len(self.world.target_bots) > 0 and self.firing_timer <= 0 and self.weapon_ammunition_inventory[self.weapon_type] > 0:
        target = self.world.target_bots[0]
        for agent in self.world.target_bots:
            if (self.pos - agent.pos).length() < (self.pos - target.pos).length():
                target = agent
        #intializing the defaults values
        hit = False
        out_of_bound = False
        delta = 0.0
        projected_bullet = None
        while hit == False and out_of_bound == False:
            #iterate every millisecond
            delta += 0.001
            # Get the projected position of the target after delta seconds
            projected_target_position = target.pos + target.vel*delta + (target.acceleration/2)*delta*delta
            # Get an unit vector from self.pos to projected position to shoot
            vector_to_target = (projected_target_position - self.pos).normalise()
            # Create a projected bullet
            projected_bullet = Projectile(self.world, self.gun_type, self.pos.copy(),vector_to_target)
            # Update the position of the bullet after delta seconds
            projected_bullet.update(delta)
            #Check whether the bullet hits or not
            if (projected_bullet.pos - projected_target_position).length() < 25.0:
                hit = True
            # Check whether the bullet goes out of bound after delta seconds
            out_of_bound = self.world.out_of_bound(projected_bullet.pos)

        if hit == True:
            bullet = Projectile(self.world,self.gun_type,self.pos.copy(), projected_bullet.aim)
            self.world.projectiles.append(bullet)
            # shoot the projectile
            bullet.fire()
            self.weapon_ammunition_inventory[self.weapon_type] -= 1 #ammo descreasing each time a shot is fired
            self.firing_timer = copy(WEAPON_FIRING_SPEED[self.weapon_type])
```

-enemies within range based on the distance from current position the boundary radius:



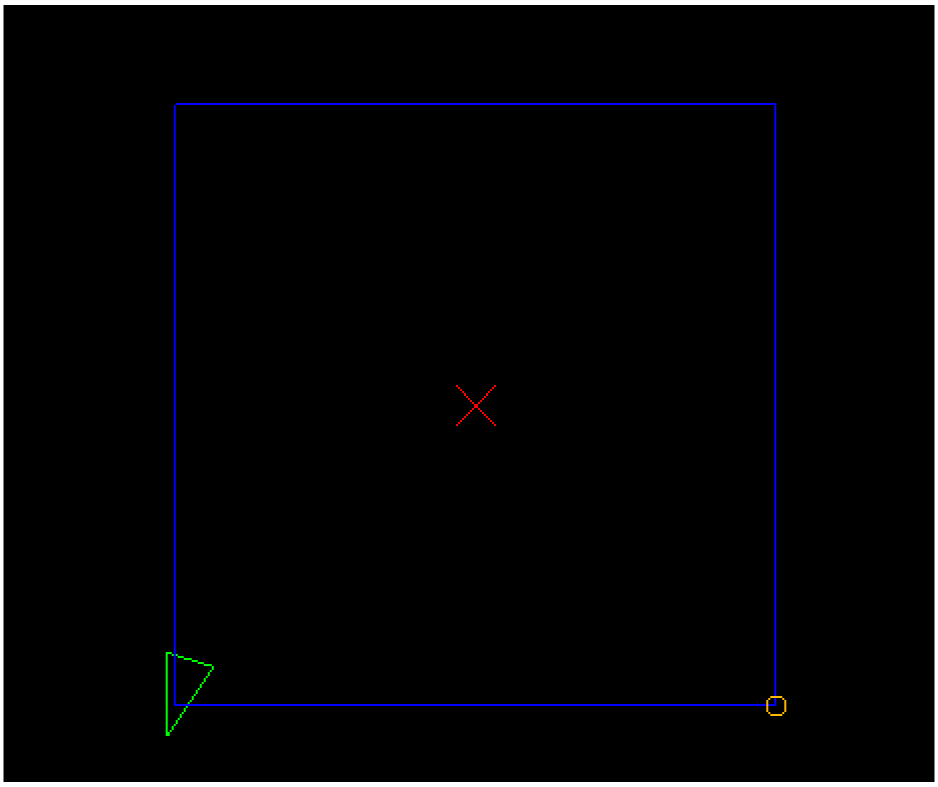
```python
def eneimes_within_range(self, radius):
    agents_list = self.world.target_bots
    for agent in agents_list:
        agent.tagged = False
        toVector = self.pos - agent.pos
        gap = radius + agent.bounding_radius
        if toVector.length_sq() < gap**2:
            agent.tagged = True
```

-reload ammunition if out of rounds and based on the weapon type accesss it’s amunition while setting relad speed and time.

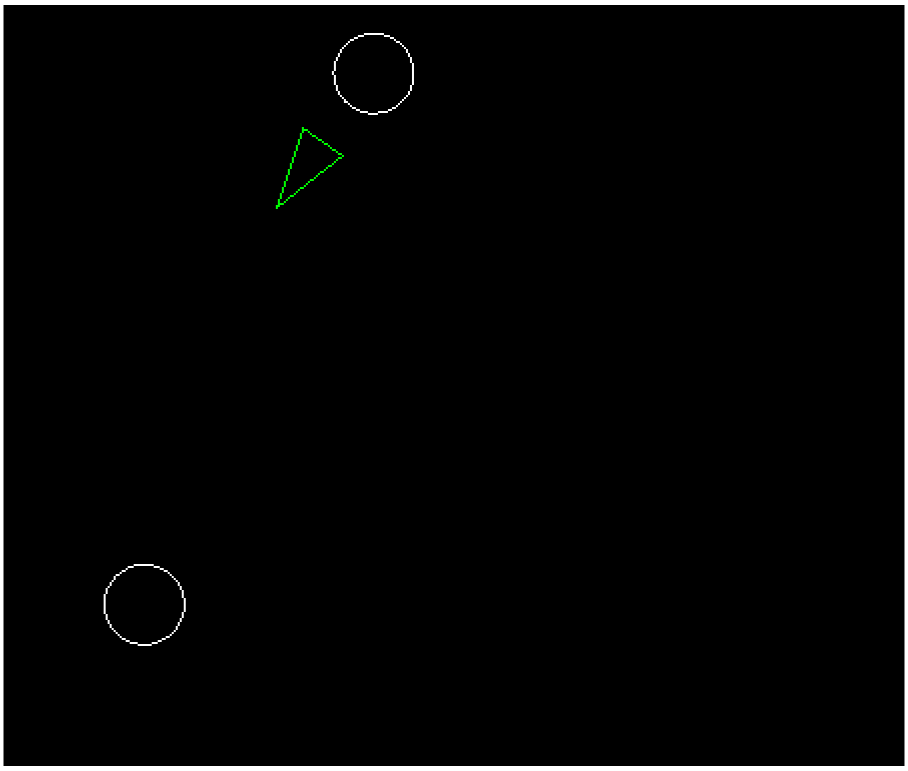


```
def reload(self, delta):
    # Reload the weapon ammunition
    if self.weapon_ammunition_inventory[self.weapon_type] < WEAPON_MAX_ROUNDS[self.weapon_type]:
        # If the reload timer is not at 0, continue to wait
        if self.reload_timer > 0.0:
                self.reload_timer -= delta
        # If the reload timer has reached 0
        else:
            # Set the weapon ammo at max capacity (reload)
            self.weapon_ammunition_inventory[self.weapon_type] = copy(WEAPON_MAX_ROUNDS[self.weapon_type])
            # Reset reload timer
            self.reload_timer = copy(WEAPON_RELOAD_SPEED[self.weapon_type])
            # Reset the firing timer
            self.firing_timer = 0.0
```

3. Output
    a. Patrolling



b. Enemy encountered:



What we Found Out:

The path of the Target is always randomized close to the Agent within the radius and once deployed it uses the AI to follow the agent's path and if too many targets are deployed the agent tries to flee from the target because of the difference in level of their strength and resets the panic distance so it can calculate the speed at which it should flee from that radius of herd and takes out the weakest targets that tries to close in using the shoot method while losing ammunition for every shot fired out of the weapon type. Once all targets eliminated or removed from the scene the agents goes back to patrolling.

The problem was that sometimes it would crash if too many targets were deployed within the radius of the target, if the target closed on the agent with must faster speed than agent or sometime the agent won't shift over to different weapon if he is out of ammo for one weapon type.

These can be revised or fixed..