Spike: 16
Title: Soldier on Patrol

Author: Ben Holmes, 103024841

Goals / deliverables:
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)
(a) High level "patrol" and "attack" modes
(b) 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.
(c) 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, and Resources used:

- Visual Studio Code
- Python 3.12.2
- Pyglet

Tasks undertaken:

- Copied code from spike 15
- Adjusted the patrol path code I developed in task 15 to use an agent specific set of 4 points roughly in top left, top right, bottom left and bottom right.

```
#patrol
    self.patrol_points = []
    left_x = randrange(100,300)
    right_x = randrange(600,700)
    top_y = randrange(600,700)
    bottom_y = randrange(100,300)
    self.patrol_points.append(Vector2D(left_x,top_y))# random point in top left corner
    self.patrol_points.append(Vector2D(right_x,top_y))# random point in top right corner
    self.patrol_points.append(Vector2D(left_x,bottom_y))# random point in bottom left corner
    self.patrol_points.append(Vector2D(right_x,bottom_y))# random point in bottom right corner
    self.patrol_couter = 0
```

- Adjusted patrol mode code to change mode to attack if a target agent exists

```
def patrol(self):
    '''patrols between a patrol point list in order, looping'''
    # patroling
    target = self.patrol_points[self.patrol_couter]
    target_vel = self.seek(target)
    if self.pos.distance(target) < 10:
        self.patrol_couter += 1
        if self.patrol_couter >= len(self.patrol_points):
            self.patrol_couter = 0
    # attacking decision
    if self.world.target_agent != None:
        self.mode = 'attack'
    return target_vel
```

- Created an attack mode that shoots if there are enough bullets (with a delay between shots) and triggers a reload function if not. It also seeks to the target_agent position. If the target agent does not exist then it switches back to patrol mode

```
def attack(self,target):
    if self.world.target_agent != None:
        if self.bullets > 0:
            # shooting (shot delay on outside of shoot function to not cause clarity issues in shoot function)
            if self.shot_delay_timer == 0:
                self.shoot(target,self.world.bullet_mode)
                self.bullets -= 1
                self.shot_delay_timer = self.shot_delay_max
            else:
                self.shot_delay_timer -= 1
        # reloading
        else:
            self.reload()

        return self.seek(target.pos)
    else:
        self.mode = 'patrol'
        return Vector2D()
```

- Created the reload function and the time/state variables for the bullets, reloading and shot delay

```
def reload(self):
    if self.reload_timer > 0:
            self.reload_timer -= 1
    else:
        self.reload_timer = self.reload_max
        self.bullets = self.bullets_max
```

```
# soldier
    # how many bullets at full and bullet number counter
    self.bullets_max = 5
    self.bullets = self.bullets_max

    # how many frames/updates reloading takes and reload time counter
    self.reload_max = 30
    self.reload_timer = self.reload_max

    # how many frames/updates between shots and delay time counter
    self.shot_delay_max = 10
    self.shot_delay_timer = self.shot_delay_max
```

- Bullet class in bullet.py is unchanged from task 15
- Adjusted how the bullet registered hits and how that interacted with the target agent, mainly changing to use target_agent = none and target agent != none

```
def update(self, delta):
    if not self.paused:
        for agent in self.agents:
            agent.update(delta)
        if self.target_agent != None:
            self.target_agent.update(delta)
        self.hunter.update(delta)
        active_bullets = [] # this is to replace list of current bullets after list is checked
        for bullet in self.bullets:
            bullet.update(delta)
            if self.target_agent != None:
                if bullet.check_hit():
                    self.target_agent = None
            if bullet.check_lifetime():
                active_bullets.append(bullet)
        self.bullets = active_bullets
```

- Adjusted the spacebar to create a new target_agent if one didn't exist in wander mode for a bit more interactiveness

```
elif symbol == pyglet.window.key.SPACE:
    if self.target_agent == None:
        self.target_agent = Agent(self,mode='wander')
```

What we found out:

As I had already implemented a patrol mode in task 15, it was simple to adjust it to be at the agent level instead of the world level. I had also created the shoot function to be very modular, so it was simple to just move that to within the attack function and use the attack mode instead. The only slight

difficulty was getting the removal of the target agent when hit to work because of how the bullet hit check works.

Because of task 15 various mechanics like the bullet speeds and inaccuracies are still implemented in this one, having been tested and encountering no issues

Testing buttons:

Spacebar creates a target agent if one exists

Bullet modes numpad:
1: rifle (fast accurate)
2: rocket (fast inaccurate)
3: pistol (slow accurate)
4: grenade (slow inaccurate)