Spike: Task 15

Title: Soldier on Patrol

Author: Sam Huffer, 101633177

Goals / Deliverables

- Basic deliverables:
    - A "soldier on patrol" simulation where agents have 2+ high-level FSM modes, and low-level FSM behaviour, including at least:
        - High-level "patrol" and "attack" states
        - The "patrol" mode must control low-level states so the agent will visit way-points along a path.
        - The "attack" mode must control low-level fighting states (shoot, reload, etc.)
    - Enemy agents need to be able to be added.
    - With the soldier attacking enemies, a basic health / attack model is needed.

Technologies, Tools, and Resources Used

- SublimeText (for editing, executing and testing the code)
- Learning materials on Canvas (for instructions and sample code)

Tasks Undertaken

- I started by copying the project from Task 14: Agent Marksmanship folder to Task 15: Soldier on Patrol folder. I stripped the spike report down to what was needed for this task, and tweaked the base code to separate out weapon and agent types from agents' modes and sub modes, to allow more flexibility with states during this task.
- I set up a pre-set patrol path for the soldier, and changed its and target's starting positions to be the first point in the path and the centre of the simulation space respectively.
- I updated the soldier agent to expand its field of view and remove gaps in said field of view, and re-wrote the decision logic for how each agent moves to fit the current scenario and to make states clearer in a way that can be displayed by the simulation's UI.
- I separated the code for deciding where to shoot and actually firing the shot into separate methods so that the soldier could display more appropriate states, and then modified projectiles to deal damage to agents. This also involved implementing damaging projectiles, the death of agents when they reached 0 HP, and respawning them according to keyboard input.
- Lastly, I tweaked the soldier's update method to include not just a cooldown between shots, but a magazine size, a rounds-left-in-magazine count, and a reload time when the magazine runs out.

Instructions for Operating the Code

- A: toggle the display of agents' collision avoidance range.
- I: toggle the display of agents' force, velocity and net desired change in position.
- P: pause or un-pause the game.
- S: re-spawn a dead soldier.
- T: re-spawn a dead target.
- W: scroll through soldier weapons.
- Escape: exit the simulation.

Code Snippets

```
def next_weapon(self):
    max_index = 4
    self.weapon_index += 1

    if self.weapon_index > max_index:
        self.weapon_index = 0

    self.last_shot = datetime.now()
    self.rounds_left_in_magazine = 0

    # Numbers borrowed from weapons in Halo CE where possible; rate of fire modified for
    # how fast I'd probably shoot with them
    if self.weapon_index == 0:
        self.weapon = 'Rifle'
        self.cooldown = 1.5 # max rpm of 0.5 sec / round
        self.effective_range = 10 * 2300 # effective range 2300 m
        self.damage = 50
        self.reload_time = 2.6
        self.magazine_size = 4
    elif self.weapon_index == 1:
        self.weapon = 'Rocket'
        self.cooldown = 1.5 # max rpm of 0.6 sec / round
        self.effective_range = 5 * 160 # estimated effective range 160 m
        self.damage = 6        # explosive; does damage over time
        self.reload_time = 3
        self.magazine_size = 2
    elif self.weapon_index == 2:
        self.weapon = 'Hand Gun'
        self.cooldown = 0.286 # max rpm
        self.effective_range = 5 * 122.7 # effective range 122.7 m
        self.damage = 20
        self.reload_time = 1.8
        self.magazine_size = 12
    elif self.weapon_index == 3:
        self.weapon = 'Hand Grenade'
        self.cooldown = 2     # estimated max rpm of 2 sec / round
        self.effective_range = 5 * 75 # estimated effective range 75 m
        self.damage = 4        # explosive; does damage over time
        self.reload_time = 2
        self.magazine_size = 8
    elif self.weapon_index == 4:
        self.weapon = 'Shotgun'
        self.cooldown = 1    # max rpm of 1 sec / round
        self.effective_range = 30 * 5 # estimated effective range 5 m
        self.damage = 20      # multiple pellets; damage is spread out amongst them
        self.reload_time = 6
        self.magazine_size = 12
```

Figure 1: the method for switching over to the next weapon, and changing all the weapon's values to match the selected weapon.

```python
def update_shooter(self, delta):
    self.see_target = False
    self.update_fov(self.world.obstacles, self.world.agents)

    if self.see_target or (self.world.target is not None and self.distance(self.world.target.pos) < self.hunt_dist + self.world.target.radius):
        self.movement_mode = 'Attack'

    else:
        self.movement_mode = 'Patrol'

    # change to work with reload
    if self.rounds_left_in_magazine == 0 and (datetime.now() - self.last_shot).total_seconds() <= self.reload_time:
        self.combat_mode = 'Reloading'
    else:
        if self.rounds_left_in_magazine == 0:
            self.rounds_left_in_magazine = self.magazine_size

        if (datetime.now() - self.last_shot).total_seconds() <= self.cooldown:
            self.combat_mode = 'Weapon is Loading Next Round'
        elif self.movement_mode == 'Attack':
            self.combat_mode = 'Aiming'
        elif self.movement_mode == 'Patrol':
            self.combat_mode = 'Ready'

    self.move(delta)

    if self.movement_mode == 'Attack' and self.combat_mode == 'Aiming':
        if len(self.projectile_pool) == 0:
            self.combat_mode = 'No Projectiles Pooled'
        else:
            target = self.aim_shot(self.world.target)

            if target is not None:
                self.combat_mode = 'Shooting'
                self.last_shot = datetime.now()
                self.rounds_left_in_magazine -= 1
                self.shoot(target)

def update_target(self, delta):
    if self.world.shooter is not None and (self.distance(self.world.shooter.pos) < self.avoidance_range or self.world.shooter.movement_mode == 'Attack'):
        self.movement_mode = 'Escape'
    else:
        self.movement_mode = 'Wander'

    #if self.movement_mode is not 'Stationary' or self.vel.length() > 0:
    self.move(delta)
```

Figure 2: the custom update methods for the two agent types. Both feature clearly named movement and combat modes that are readily displayable on-screen.

```python
def calculate_target(self, delta):
    if self.movement_mode == 'Wander' or self.world.shooter == None:
        return self.wander(delta)
    elif self.movement_mode == 'Escape':
        return self.avoid(self.world.shooter.pos)
    # elif self.movement_mode == 'Stationary':
    #     if self.vel.length() > 0:
    #         return -self.vel

    return Vector2D(0,0)

def calculate_shooter(self, delta):
    if self.movement_mode == 'Patrol':
        return self.follow_path()
    elif self.movement_mode == 'Attack':
        if self.world.obstacles_enabled:
            return self.hunt(self.world.target, delta)
        else:
            return self.seek(self.world.target.pos)
    # elif self.movement_mode == 'Stationary':
    #     if self.vel.length() > 0:
    #         return -self.vel

    return Vector2D(0,0)
```

Figure 3: the methods for calculating the agent's movement based on agent type and current movement mode.

```
if self.showinfo:
    infotext = ', '.join(set(agent.agent_type for agent in self.agents))
    egi.white_pen()

    if self.shooter is not None:
        health_status = 'Soldier: ' + str(self.shooter.health) + ' HP. '
        agent_status = 'Soldier Status: ' + self.shooter.movement_mode + ', ' + self.shooter.combat_mode + '. Soldier Weapon: ' +
    else:
        health_status = 'Soldier: 0 HP. '
        agent_status = 'Soldier Status: Dead. Soldier Weapon: N/A. '

    if self.target is not None:
        health_status = health_status + 'Target: ' + str(self.target.health) + ' HP.'
        agent_status = agent_status + 'Target Status: ' + self.target.movement_mode + '.'
    else:
        health_status = health_status + 'Target: 0 HP.'
        agent_status = agent_status + 'Target Status: Dead.'

    egi.text_at_pos(0, 20, health_status)
    egi.text_at_pos(0, 0, agent_status)
```

Figure 4: code from world.render() regarding the displaying of agent information on the screen. The cropped line of code, in full, is " agent_status = 'Soldier Status: ' + self.shooter.movement_mode + ', ' + self.shooter.combat_mode + '. Soldier Weapon: ' + self.shooter.weapon + ', ' + str(self.shooter.rounds_left_in_magazine) + '/' + str(self.shooter.magazine_size) + '. '"

```
def set_agents(self, max_x, max_y):
    if self.shooter == None:
        self.shooter = Agent(world=self, agent_type='shooter', weapon='Rifle')
        self.agents.append(self.shooter)
        self.shooter.path = Path(num_pts=9, looped=True)
        self.shooter.update_hunt_dist()
    if self.target == None:
        self.target = Agent(world=self, agent_type='target')
        self.agents.append(self.target)

    self.shooter.pos = Vector2D(max_x * 0.2, max_y * 0.2)
    self.shooter.heading = Vector2D(0,1)
    self.shooter.side = self.shooter.heading.perp()
    self.shooter.path.recreate_preset_path(maxx=self.cx, maxy=self.cy)

    self.target.pos = Vector2D(max_x /2, max_y / 2)
    self.target.heading = (self.shooter.pos - self.target.pos).get_normalised()
    self.target.side = self.target.heading.perp()
    self.target.current_pt = Vector2D(self.target.pos.x, max_y * 0.25)
    self.target.next_pt = Vector2D(self.target.pos.x, max_y * 0.75)
```

Figure 5: the method in world for setting up the agents at the start of the simulation or if the screen is resized.

```
elif symbol == KEY.S:
    if world.shooter == None:
        world.shooter = Agent(world=world, agent_type='shooter', weapon='Rifle')
        world.agents.append(world.shooter)
        world.shooter.path = Path(num_pts=9, looped=True)
        world.shooter.heading = Vector2D(0,1)
        world.shooter.side = world.shooter.heading.perp()
        world.shooter.path.recreate_preset_path(maxx=world.cx, maxy=world.cy)
        world.shooter.update_hunt_dist()
# respawn dead target
elif symbol == KEY.T:
    if world.target == None:
        world.target = Agent(world=world, agent_type='target')
        world.agents.append(world.target)
        world.target.heading = Vector2D(0,1)
        world.target.side = world.target.heading.perp()
# scroll through shooter weapons
elif symbol == KEY.W:
    world.shooter.next_weapon()
```

Figure 6: inputs from main.on_key_press() for respawning agents and scrolling through the soldier's weapons.

In-Simulation Screenshots

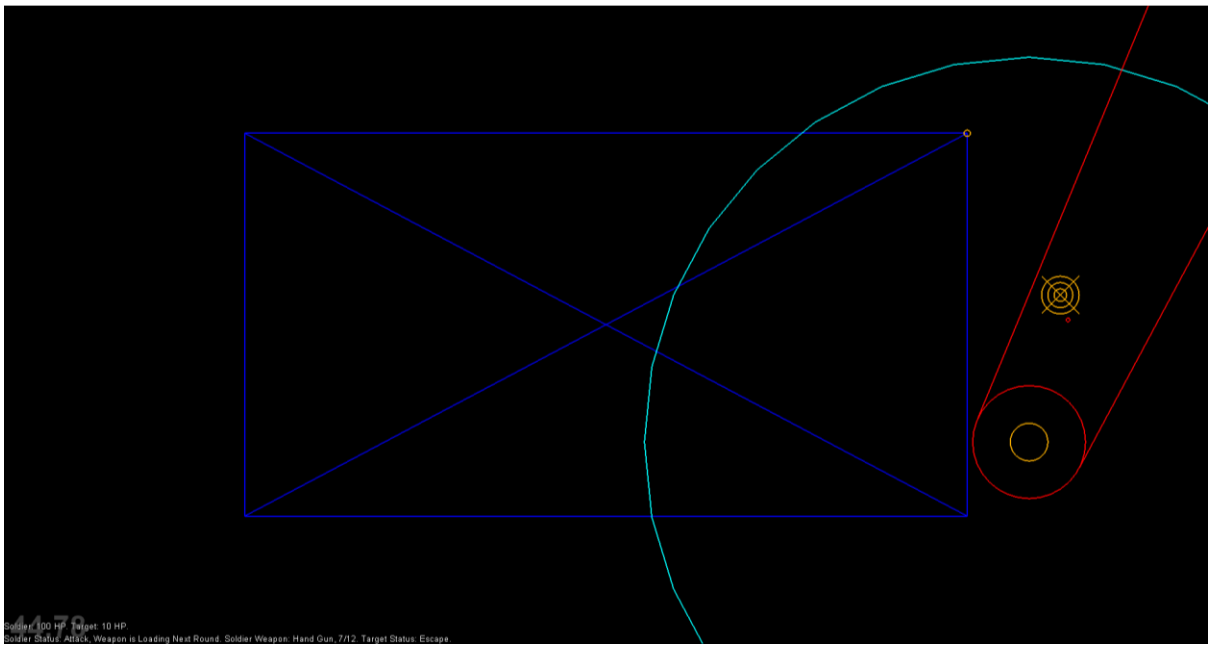

Figure 7: The soldier is attacking the target, shooting it with its hand gun.

What I Found Out

- The soldier's field of view needs to be large enough to be meaningful and realistic, not being so narrow that it can't spot targets right in front of itself. It also needs to be able to detect intrusion anywhere within its boundaries, and not have gaps where an intruder is clearly within its field of view, but it still can't see it.
- When referencing the attributes of a single, specific agent (e.g. agent's self.world.shooter or self.world.target), one needs to accommodate cases where self.world.shooter or self.world.target has been destroyed and set to None, thereby rendering any attributes required of those agents unavailable to be accessed.
- Cooldown times between shots and reload times need to be separate variables, and be handled differently and separately.