Spike: Task 16

Title: Goal-Oriented Action Planning

Author: Sam Huffer, 101633177

Goals / Deliverables

- Basic deliverable: a simulation for agent control demonstrating the effectiveness of goal-oriented action planning in considering long-term outcomes (e.g. side-effects, time delays) of actions, and can plan and act intelligently.

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 15: Soldier on Patrol into the Task 16: Goal-Oriented Action Planning folder, stripping the spike report down to what was needed for this task.
- Next, I consulted my lecture notes and the task instructions document to plan out what I wanted to do for this task, as how students implemented goal-oriented action planning was left up to us. The planned goals I put together for this task were:
    - The agent will have to contend with hunger, which will be increased with each step it takes, and each shot it fires. When its hunger reaches a threshold, it must consider whether to continue its current actions, or return to the food station and eat if it would not compromise its current attack against the target. When its hunger reaches a high enough level that if it continued its current actions it would not have the stamina to return to a food station to reduce its hunger (based on the maximum possible distance between itself and the food station), it must return to the food station and eat.
    - The soldier has two weapons at a time, with a limited number of magazines. When patrolling, they must consider if their ammo remaining, between the two weapons, will theoretically be sufficient to reduce the target's HP to 0. If it is not, the agent must return to an ammunition station to get new weapons. When attacking, if the soldier runs out of ammo, they must return to the ammo station to exchange their weapons. The choice of weapons should be random, and all weapons, including the ones just handed back, will be available.
    - When aiming its shot, the agent will consider whether staying with its current weapon or its secondary weapon will be more conducive or sufficient in reducing the target's HP to 0, in terms of shot impact, speed, rounds per minute. For example, if the target has less than full health, the soldier shouldn't use two rifle rounds to take out the target if another weapon would also reduce the target's health to 0 in one shot.
- First, I took the code outlining what each weapon was, as well as the code for pooling projectiles, and created a weapon class that could hold the appropriate variables and be passed from world to soldier and back.
- Created points for ammo station and food station.

- Added logic for considering when to swap to the next weapon, and to go to the ammo station if attacking and out of ammo, or patrolling and ammo would be insufficient to kill the target when encountered.
- Added logic for handling explosive weapons, for not firing one if too close to the target, and for avoiding existing explosive projectiles so as not to get caught in the blast radius.
- Added logic for checking if the soldier will starve if they don't go and get food now, and for sending them to the food station to satiate its hunger.
- Looking at canvas instructions and instructions pdf again after the lecture and having done all this, not sure it meets the requirements for this task, so submitting a draft to get feedback and moving on to the next task in the meantime.

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

```
class Weapon (object):
    def __init__(self, world=None, owner=None, name='', cooldown=1, effective_range=100, damage=25, reload_time=1,
        self.world = world
        self.owner = owner
        self.name = name
        self.cooldown = cooldown
        self.effective_range = effective_range
        self.speed = speed
        self.damage = damage
        self.damage_factor = damage_factor
        self.reload_time = reload_time
        self.magazine_size = magazine_size
        self.rounds_left_in_magazine = 0
        self.magazines_left = magazines
        self.accuracy_modifier = accuracy_modifier
        self.stamina_drain = stamina_drain
        self.explosive = explosive
        self.explosion_radius = explosion_radius

        self.projectile_pool = []
        i = 0

        while i < 200:
            self.projectile_pool.append(Projectile(world=self.world, weapon=self))
            i += 1
```

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

    if not self.hungry() and self.choose_weapon():
        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'

        if self.weapons[0].rounds_left_in_magazine == 0 and (datetime.now() - self.last_shot).total_seconds() <= self.weapons[0].reload_time:
            self.combat_mode = 'Reloading'
        else:
            if self.weapons[0].rounds_left_in_magazine == 0:
                self.weapons[0].rounds_left_in_magazine += self.weapons[0].magazine_size
                self.weapons[0].magazines_left -= 1

            if (datetime.now() - self.last_shot).total_seconds() <= self.weapons[0].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'

        if self.movement_mode == 'Attack' and self.combat_mode == 'Aiming':
            if len(self.weapons[0].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.weapons[0].rounds_left_in_magazine -= 1
                    self.shoot(target)

    elif self.movement_mode == 'Exchange Weapons' and self.distance(self.world.ammo_station) < self.world.station_size:
        self.world.change_weapons(self)
        self.movement_mode = 'Patrol'
    elif self.movement_mode == 'Get Food' and self.distance(self.world.food_station) < self.world.station_size:
        self.hunger = 0
        self.movement_mode = 'Patrol'


    self.move(delta)
```

```
def calculate_shooter(self, delta):
    if self.movement_mode == 'Exchange Weapons':
        return self.arrive(self.world.ammo_station, 'slow')
    elif self.movement_mode == 'Get Food':
        return self.arrive(self.world.food_station, 'slow')
    elif 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)

    return Vector2D(0,0)
```

```
def hungry(self):
    if self.hunger >= 50:
        self.world.destroy_agent(self)

    if self.last_hunger_ping == None:
        self.last_hunger_ping = datetime.now()

    if (datetime.now() - self.last_hunger_ping).total_seconds() >= 1:
        self.last_hunger_ping = datetime.now()
        self.hunger += 1

    if self.movement_mode == 'Get Food':
        return True
    elif self.movement_mode == 'Patrol':
        max_distance = (Vector2D(0,0) - Vector2D(self.world.cx, self.world.cy)).length()
        max_time = max_distance / self.max_speed

        if (self.hunger + max_time) * 1.1 >= 50:
            self.movement_mode = 'Get Food'
            return True

    elif self.movement_mode == 'Attack':
        current_distance = self.distance(self.world.food_station)
        current_time = current_distance / self.max_speed

        if (self.hunger + current_time) * 1.1 >= 50:
            self.movement_mode = 'Get Food'
            return True

    return False
```

```
def choose_weapon(self):
    if self.movement_mode == 'Exchange Weapons':
        return False

    if len(self.weapons) <= 1:
        self.movement_mode = 'Exchange Weapons'
        return False

    weapon_0 = self.weapons[0]
    weapon_1 = self.weapons[1]

    weapon_0_ammo = weapon_0.rounds_left_in_magazine + weapon_0.magazine_size * weapon_0.magazines_left
    weapon_1_ammo = weapon_1.rounds_left_in_magazine + weapon_1.magazine_size * weapon_1.magazines_left

    weapon_0_avg_dmg = weapon_0.damage * weapon_0.damage_factor
    weapon_1_avg_dmg = weapon_1.damage * weapon_1.damage_factor

    if self.world.target is not None:
        target_health = self.world.target.health

    # check if out of ammo or if patrolling and have insufficient ammo to kill the target
    if weapon_0_ammo <= 0 and weapon_1_ammo <= 0:
        self.movement_mode = 'Exchange Weapons'
        return False
    elif self.movement_mode == 'Patrol':
        if self.world.target is not None:
            if weapon_0_avg_dmg * weapon_0_ammo + weapon_1_avg_dmg * weapon_1_ammo < target_health:
                self.movement_mode = 'Exchange Weapons'
                return False
        elif weapon_0_avg_dmg * weapon_0_ammo + weapon_1_avg_dmg * weapon_1_ammo < self.start_health:
            self.movement_mode = 'Exchange Weapons'
            return False

    # check if only current weapon is out of ammo
    if weapon_0_ammo <= 0 and weapon_1_ammo > 0:
        self.next_weapon()
    # check if both weapons have ammo, if both weapons' probable damage dealt (accounting for explosive splash damage and multiple s
    # would be sufficient to kill the target, and the next weapon deals less damage
    if self.world.target is not None and weapon_0_ammo > 0 < weapon_1_ammo and weapon_0_avg_dmg > weapon_1_avg_dmg > target_health:
        self.next_weapon()

    return True
```

In-Simulation Screenshots

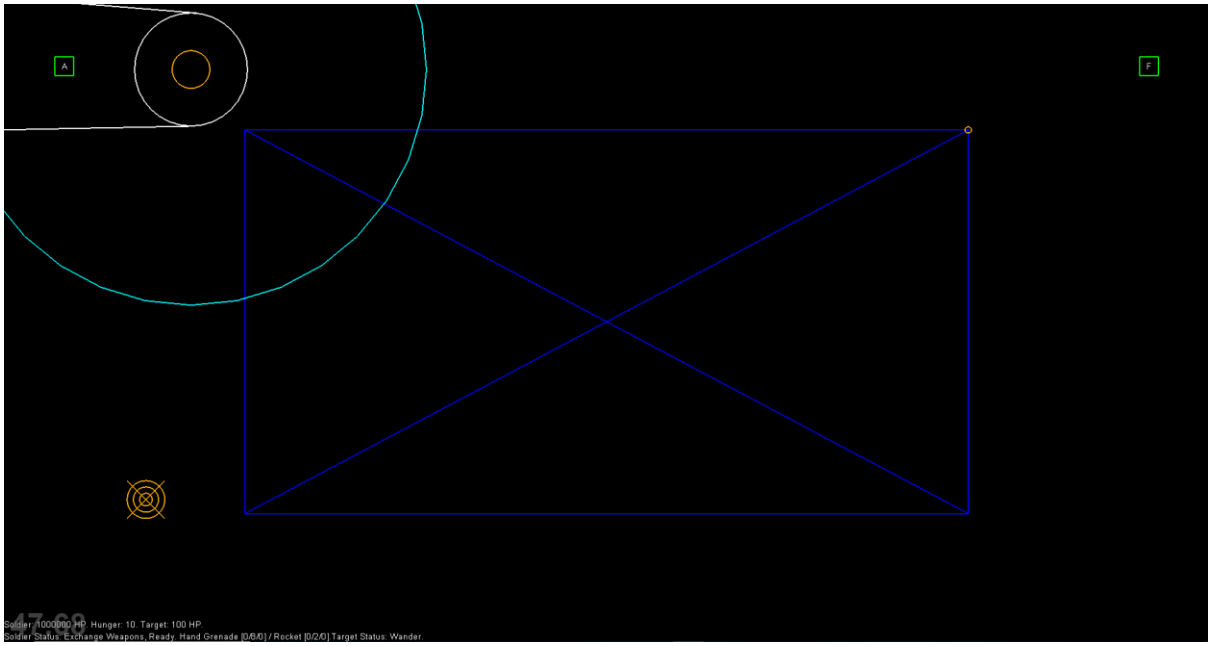

Figure 2: soldier agent is out of ammo and needs to swap its weapons.

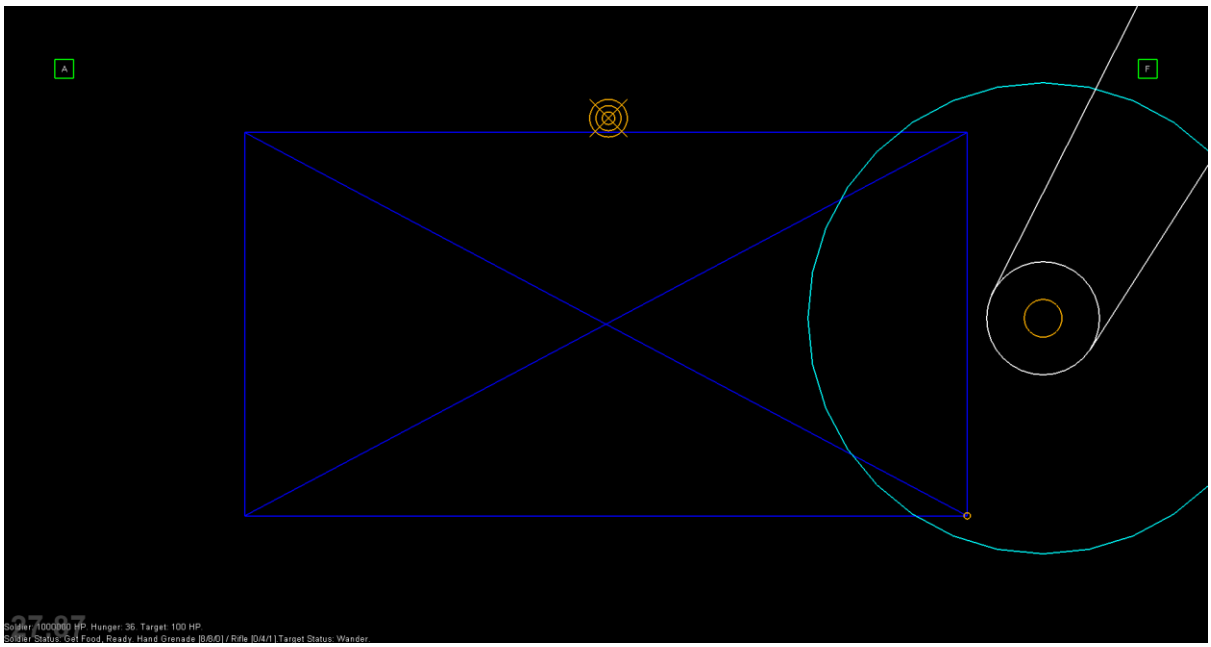

Figure 1: soldier agent is hungry and needs to eat.

What I Found Out

TBD