SWINBURNE UNIVERSITY OF TECHNOLOGY

COS30002

Tactical Steering (Hiding)

Task 10, Week 6
Spike: Spike_No10

Matthew Coulter
S102573957

23/04/2020

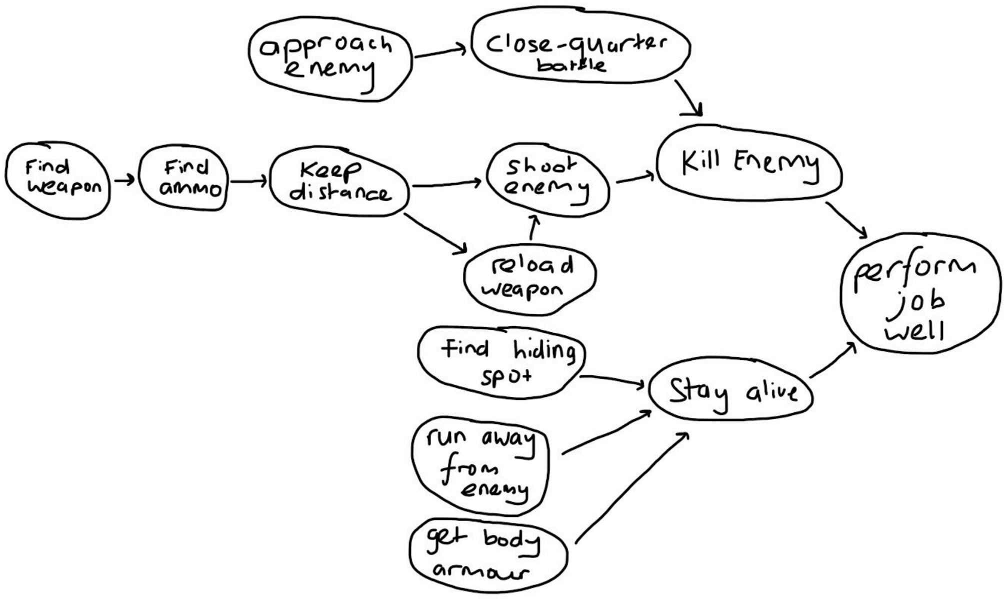
Tutor: Tien Pham

Table of Contents

Goals / Deliverables

The aim of this task is to combine goal oriented behaviour (GOB) with graph search to create a goal-oriented action planning (GOAP) system for agents to plan their way out of trouble. By doing this, our AI will become smarter in choosing how it should attack the enemies and escape dangerous situations.
Because these GOAP systems can become quite complicated, I have created a diagram that represents the

Outside of the code and this lab report, I have also created the following diagram:



Technologies, Tools, and Resources used:

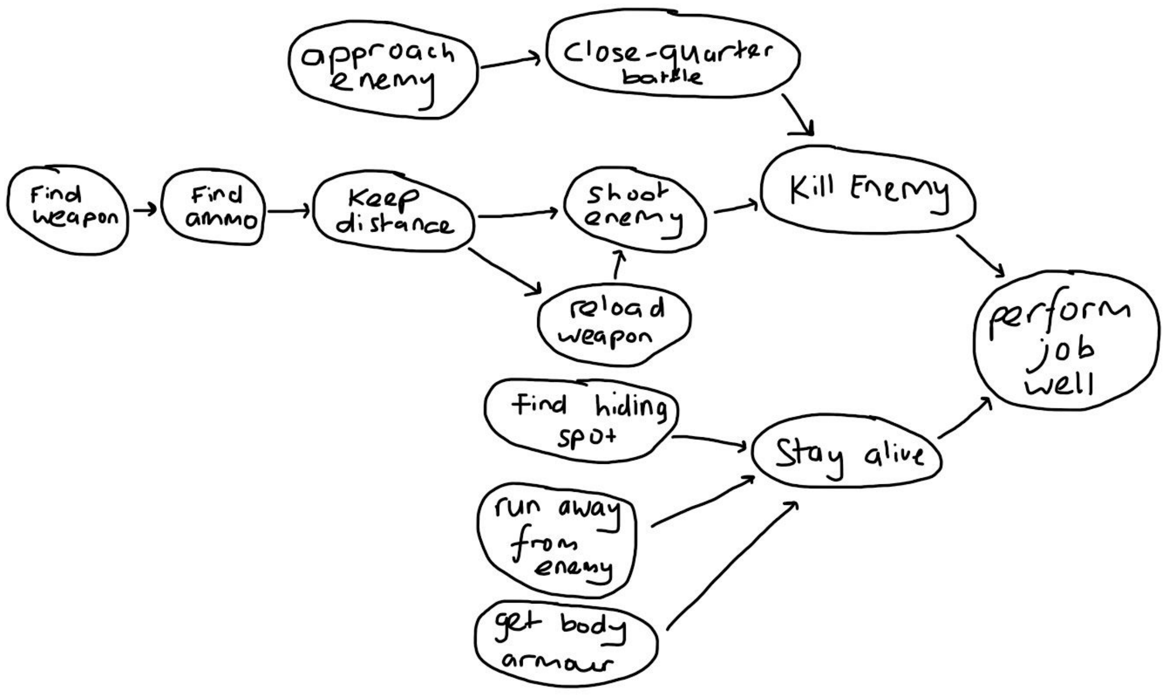
If you wish to reproduce my work, here is a list of all the required tools and resources that you will need:

- Visual Studio
- Python language pack
- This spike report
- Lab15 code
- Git bash or sourcetree (if you wish to access my code)
    - repo: ‘git clone https://mattcoulter7@bitbucket.org/mattcoulter7/cos30002-102573957.git’

Tasks undertaken:

Planning the GOAP

In our previous lab, we had 3 simple modes where each mode can be changed at any point in time. Those modes were shoot, reload and patrol. That system works great for a few modes however because we are implementing a GOAP method we need to add a few more modes and methods to make this method viable. Because these systems become more complicated as we had more features/considerations of the AI, I have created the following GOAP diagram which will outline the system we are going to implement:



Implementing the weapon pickup

Originally we had an agent spawn with a weapon. However, the weapon should always have an agent holding it as they will be spawning all around the room. Hence add a list into world for all of the weapons to be spawned:

```
self.weapon = []
```

When we spawn the weapon in, we will append it to that list and the agent will by default be None. However, when the agent flies over the weapon, we want them to pick it, hence add this function into the agent class:

```
    def pickup_object(self,object):
        # Picks up weapon from world if doesn't already have a weapon
        if getattr(self, object) is None:
            for obj in getattr(self.world, object):
                to_obj = obj.pos - self.pos
                dist = to_obj.length()
                if dist < self.scale.length() * 2:
                    obj.agent = self
                    setattr(self,object,obj)
```

This acts as a general pickup object method and we can enter the object name as a string into the parameter when we call it. Call this function from update with string 'weapon'

```
self.pickup_object('weapon')
```

We also need to have a way of spawning the weapon. We will assign the W key to this function and it will position the weapon randomly on the map.

If the mode is set to 'findweapon' it will call this function:

```
    def findweapon(self):
        closest_weapon = min(self.world.weapon, key = lambda e: (e.pos - 
self.pos).length())
        return self.arrive(closest_weapon.pos)
```

Add this function for now, we will change the calculate function later on.

Implementing the Body Armour

As this is just a simulation and the enemy does not actually attack, the body armour will simply display around the agent but does not serve any extra protection because they can't take damage.

The implementation for the body armour is very similar to that of the weapon. We will reuse the same pickup_object function but pass 'bodyarmour' as a string into it instead.

```
self.pickup_object('bodyarmour')
```

Hence we also need to have a list for the bodyarmour objects in the world.

```
self.bodyarmour = []
```

We will assign the 'B' key to spawn a random body armour object in the world.

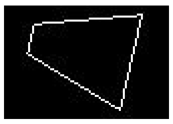
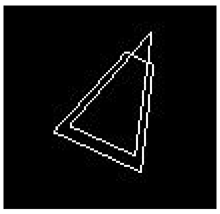
Agents will have a body armour variable that is by default None.

```
self.bodyarmour = None
```

Here is the code for the body armour:

```
from vector2d import Vector2D
from point2d import Point2D
from graphics import egi, KEY
from math import sin,cos,radians
from random import random,randrange

class BodyArmour(object):
    """Spawns randomly on the map and adds health to agent if picked up"""
    def __init__(self,world = None,agent = None,scale = 30.0):
        self.protection = 100
        self.world = world
        self.agent = agent
        self.color = 'WHITE'
        self.pos = Vector2D(randrange(0,world.cx),randrange(0,world.cy))
        dir = radians(random()*360)
        self.heading =  self.heading = Vector2D(sin(dir), cos(dir))
        self.side = self.heading.perp()
        self.scale = Vector2D(scale, scale)  # easy scaling of agent size

        self.shape = [
            Point2D(-1.2,  0.8),
            Point2D( 0.5,  0.25),
            Point2D( 0.5,  -0.25),
            Point2D(-1.2, -0.8)
        ]

    def update(self,delta):
        if self.agent:
            self.pos = self.agent.pos
            self.heading = self.agent.heading
            self.side = self.agent.side

    def render(self):
        '''Renders gun to screen'''
        egi.set_pen_color(name=self.color)
        pts = self.world.transform_points(self.shape, self.pos,
self.heading, self.side, self.scale)
        egi.closed_shape(pts)
```

This is what the body armour looks like:

If the mode is set to ‘findarmour’ it will call this function:

```
    def findarmour(self):
        closest_armour = min(self.world.bodyarmour, key = lambda e: (e.pos
- self.pos).length())
        return self.arrive(closest_armour.pos)
```

Add this function for now, we will change the calculate function later on.

Implementing the fighting system (without weapon)

Because the agent may not have a gun, we want them to be able to have the option to fight without a gun. However it is a higher risk as they will need to be at the enemy's position to attack them and the attack won't be as strong.

Add this code to the enemy to check if they are being attacked:

```
    def update(self,delta):
        if self.alive:
            if self.health <= 0:
                self.alive = False
        self.agent_fighting()

    def take_damage(self,damage):
        self.health -= damage

    def agent_fighting(self):
        ''' Takes damage if agent is in close vicinity fighting self '''
        for agent in self.world.agents:
            to_agent = self.pos - agent.pos
            dist = to_agent.length()
            if dist < 2 * agent.scale.length():
                self.take_damage(0.5)
```

Implementing combination of GOAP with GOB

Now we have all of these extra modes implemented, we will simply redesign our check_state() function in order to consider the best mode to be in for the current state. You can design this how you like, however I chose to do it by checking the prerequisites for each goal in our diagram. WHilst I did not state those states, they are rather self explanatory:

```
    def check_state(self):
        ''' Updates the state appropriately '''
        alive_enemies = list(filter(lambda e: e.alive == True,
self.world.enemies))
        dead_enemies = list(filter(lambda e: e.alive == False,
self.world.enemies))
        if alive_enemies:
            closest_alive = min(alive_enemies, key = lambda e: (e.pos -
self.pos).length())
            to_closest_alive = (closest_alive.pos - self.pos).length()
```

```
            if self.weapon:
                self.mode = 'shoot'
                if self.weapon.reloading:
                    if dead_enemies:
                        closest_dead = min(dead_enemies, key = lambda e:
(e.pos - self.pos).length())
                        to_closest_dead = (closest_dead.pos -
self.pos).length()
                        if to_closest_dead < to_closest_alive:
                            self.mode = 'hide'
                        else:
                            self.mode = 'fight'
            else:
                if self.world.bodyarmour and not self.bodyarmour:
                    self.mode = 'findarmour'
                elif self.world.weapon:
                    self.mode = 'findweapon'
                elif dead_enemies:
                    self.mode = 'hide'
                else:
                    self.mode = 'fight'
        else:
            self.mode = 'patrol'
```

Essentially, when the agent doesn't have a weapon but one exists, it will choose that option because it is safer then going and fighting the enemy. This is the same for body armour. It will first grab the body armour then attack the enemy. However, if there are no weapons and armour and the agent doesn't have any already it will have to look for a hiding spot and get backup or wait for a weapon to spawn. However if there are no hiding spots, it will try and fight the enemy. Once the enemy is taken down, it will return back to patrol state.

The last thing we need to do to implement these modesis update the calculate function to cater for our new modes. Here is the code for that:

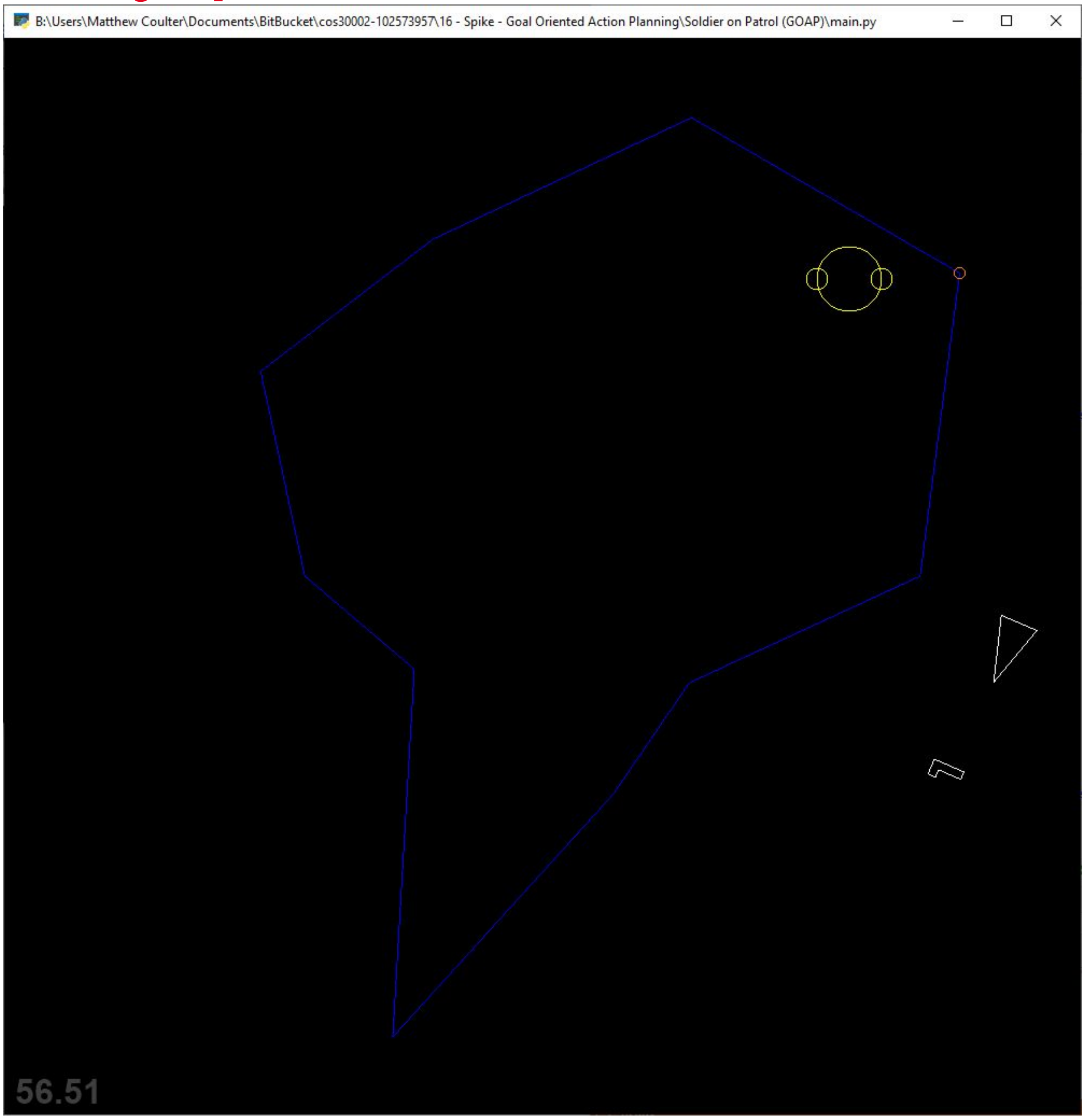
```
   def calculate(self,delta):
        # calculate the current steering force
        mode = self.mode
        force = Vector2D(0,0)
        if mode == 'patrol':
            force = self.follow_path()
            force += self.separate()
        elif mode == 'shoot':
            force = self.shoot()
        elif mode == 'hide':
            force = self.hide()
        elif mode == 'fight':
```

```python
            force = self.fight()
        elif mode == 'findweapon':
            force = self.findweapon()
        elif mode == 'findarmour':
            force = self.findarmour()
        else:
            force = Vector2D()
        self.force = force
        return force
```

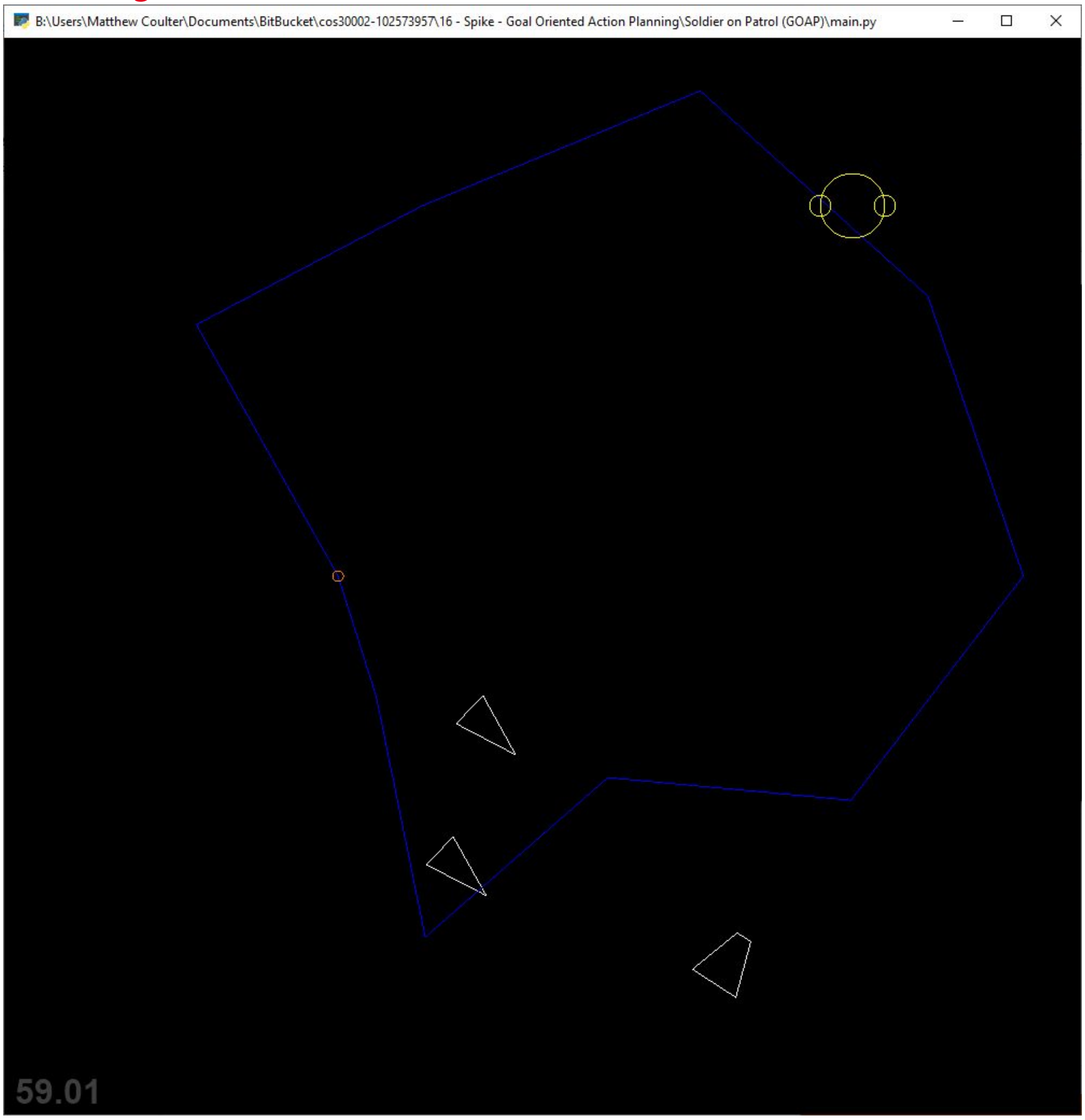
What we found out:

The agent definitely plays the safe game; when an enemy is nearby, it will try every single series of goals to help protect itself before attacking the enemy. Here are a few screenshots to convey this

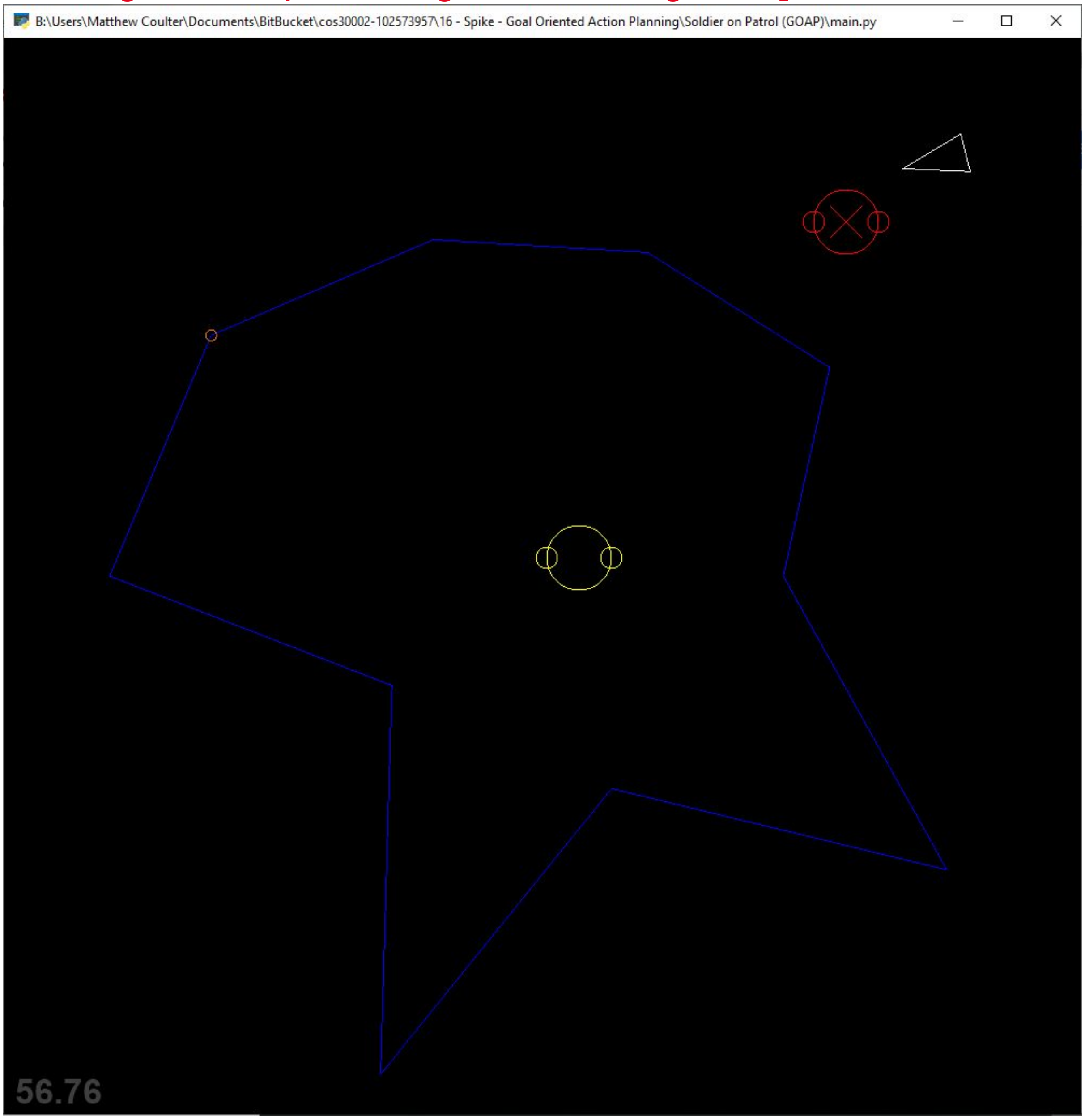
Getting weapon first:


B:\Users\Matthew Coulter\Documents\BitBucket\cos30002-102573957\16 - Spike - Goal Oriented Action Planning\Soldier on Patrol (GOAP)\main.py
56.51

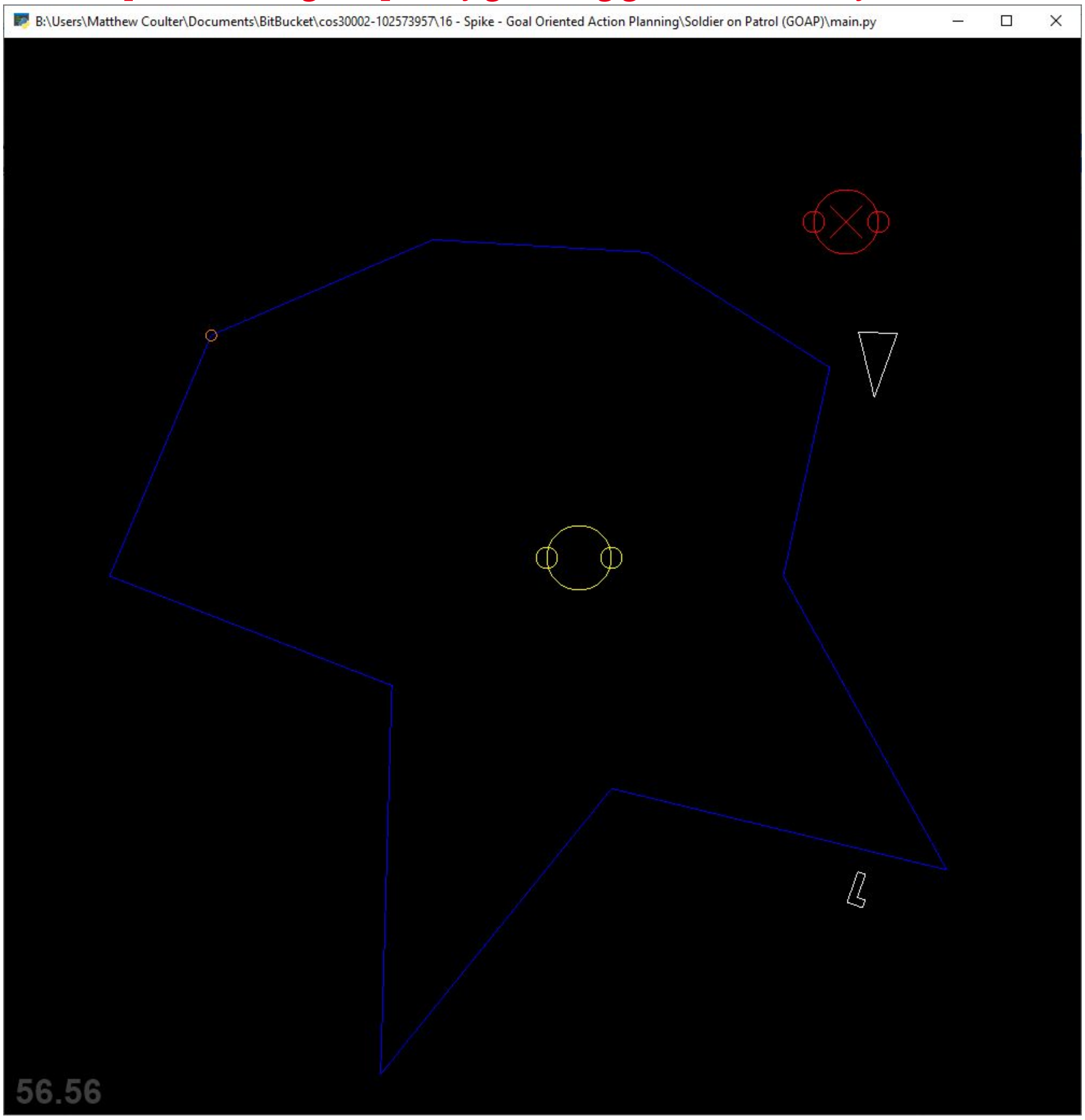
Getting armour first:


B:\Users\Matthew Coulter\Documents\BitBucket\cos30002-102573957\16 - Spike - Goal Oriented Action Planning\Soldier on Patrol (GOAP)\main.py
59.01

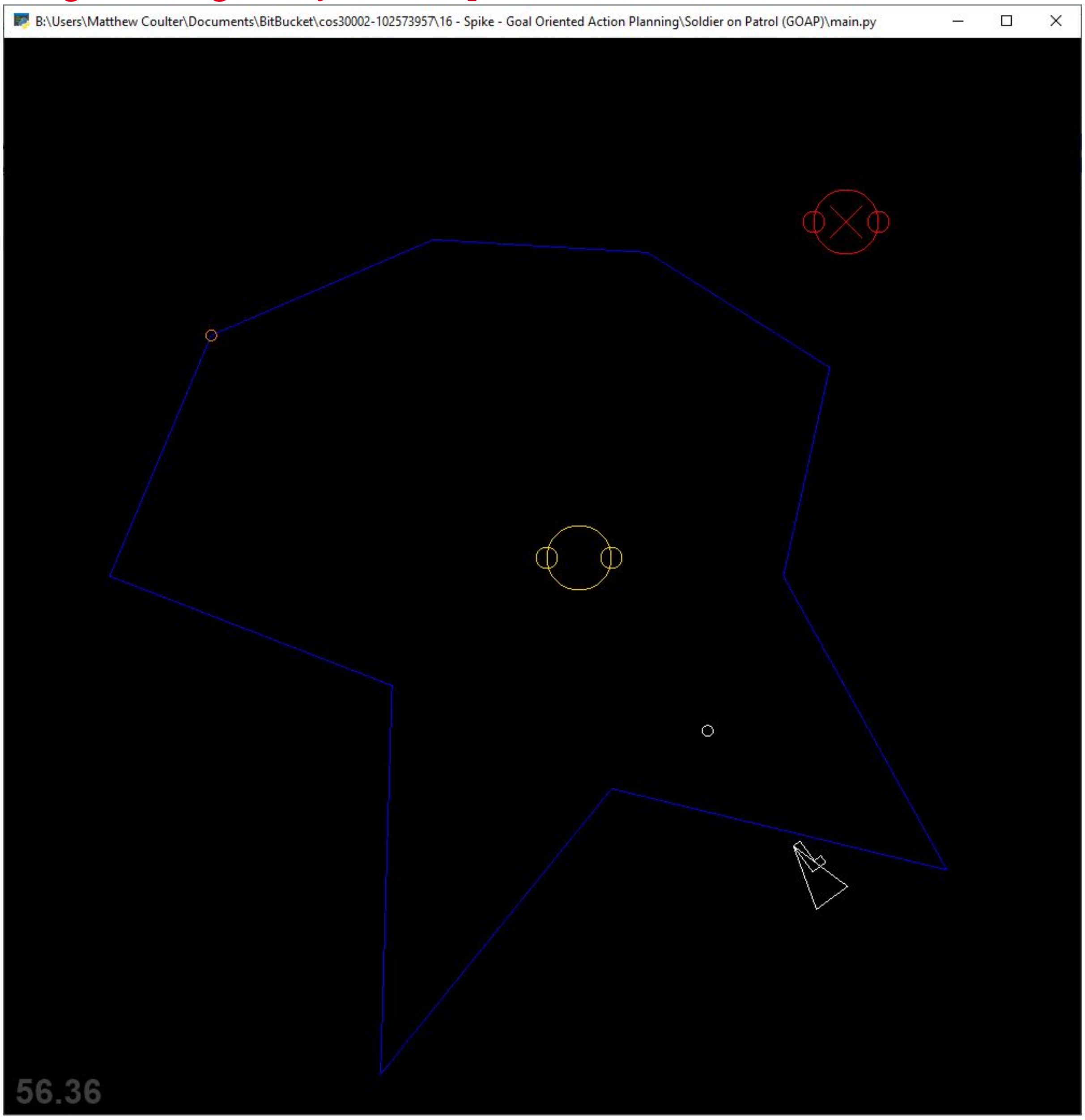
hiding behind object waiting for armour or gun to spawn:


B:\Users\Matthew Coulter\Documents\BitBucket\cos30002-102573957\16 - Spike - Goal Oriented Action Planning\Soldier on Patrol (GOAP)\main.py

Gun spawned an agent quickly grabbing gun to kill enemy:


B:\Users\Matthew Coulter\Documents\BitBucket\cos30002-102573957\16 - Spike - Goal Oriented Action Planning\Soldier on Patrol (GOAP)\main.py
56.56

Agent killing enemy with weapon:


B:\Users\Matthew Coulter\Documents\BitBucket\cos30002-102573957\16 - Spike - Goal Oriented Action Planning\Soldier on Patrol (GOAP)\main.py