SWINBURNE UNIVERSITY
OF TECHNOLOGY

COS30002

Spike Extensions

Task 19, Week 11
Spike: Spike_No19

Matthew Coulter
S102573957

23/04/2020

Tutor: Tien Pham

Table of Contents

Tasks

Lab 3

Machine learning Tic Tac Toe bot

As an extension for my second bot, I decided to invest time into learning about machine learning. I had a had a text file that would write one line for the outcome of every game into it. One game would look like this:

```
258674301tie
```

because the board was a grid of 0-8, it was easy to track a sequence of move. Once the game finished, it would append that game to the file. When the AI is playing, it would search through all of the games in the file, filter it to the ones it won, then choose random next move out of all of the moves that won to be the next move.

I ran this overnight and ended up having ~15000 unique games that the bot would read from, however, this wasn't enough because there are so many more possible games that can be played. Hence, the bot still lost so most simple bots!

Spike 10

Considering amount of coverage in a hiding spot

When calculating which hiding spot to choose, it would consider the size of the hiding spot as well as this distance to it. The following ratio was used to achieve this:

```
(h.pos - prey.pos).length()/h.size
```

Make the hunter actually "chase" the prey it can see, and "eat" the prey when it contacts them.

For a more realistic simulation, i had the pray be eaten when it was contacted through the following:

```
# Eats prey
if (self.mode == 'prey' and self.intersect_hunter()):
      self.world.agents.remove(self)
```

This in turn allowed me to assess the effectiveness of each variable combinations as we could measure survival times:

Record 1	Survival Time
Closest Hiding Spot	26.86s
MyNewBot	38.52s

Record 2	Survival Time
Closest Hiding Spot	69.15s
MyNewBot	85.36s

Record 3	Survival Time
Closest Hiding Spot	40.64s
MyNewBot	32.83s

Record 4	Survival Time
Closest Hiding Spot	3.08s
MyNewBot	41.31s

Record 5	Survival Time
Closest Hiding Spot	60.92s
MyNewBot	88.41s

Results:
Closest Hiding Spot (ave) = 40.13s
MyNewBot (ave) = 57.29s

Spike 14

Shooting moving targets

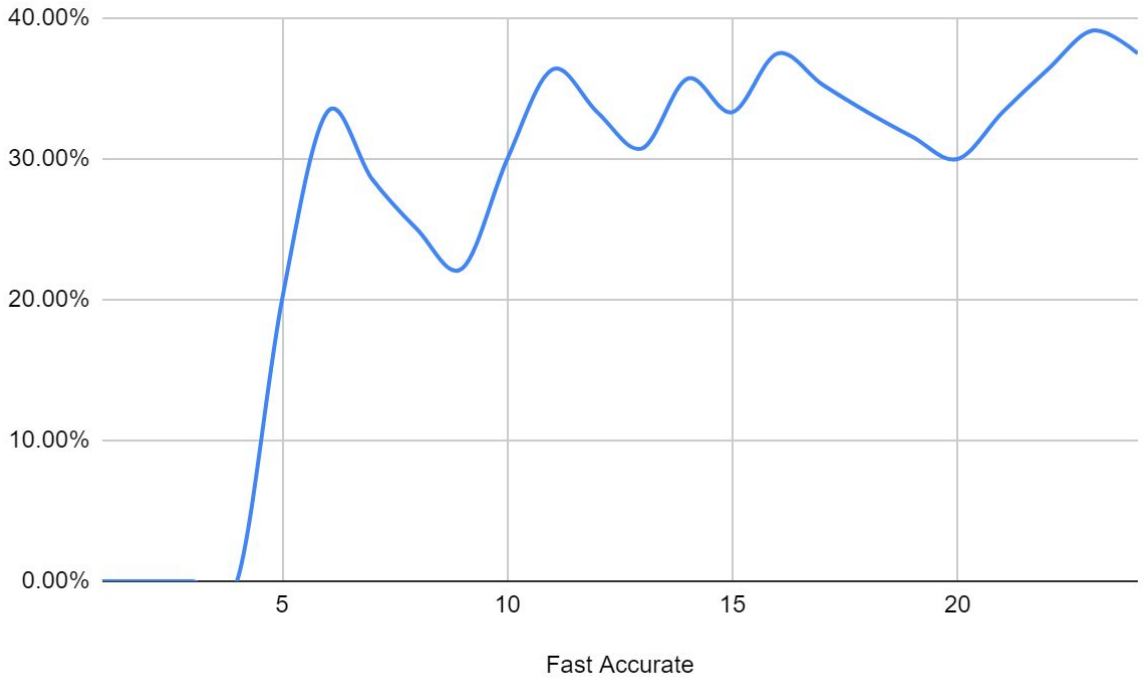
For two moving agents where one is shooting and the other is moving, we can predict their location with the following formula:

```
def get_look_ahead(self,agent):
      to_target = agent.pos - self.pos
      dist = to_target.length()
      look_ahead = dist * agent.speed() / self.weapon.proj_speed
      return look_ahead
```

Detailed analysis of results

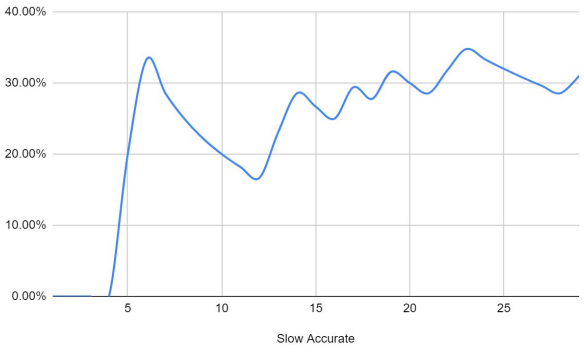
I produced detailed graphs for each of the following situations which represented how many shots it required land shots on 10 enemies:

Fast Accurate



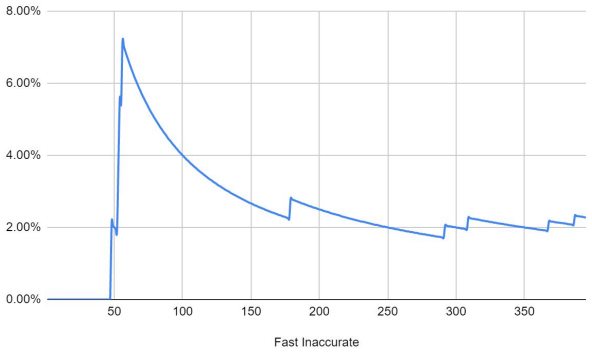
Final accuracy: 37.5%

Slow Accurate



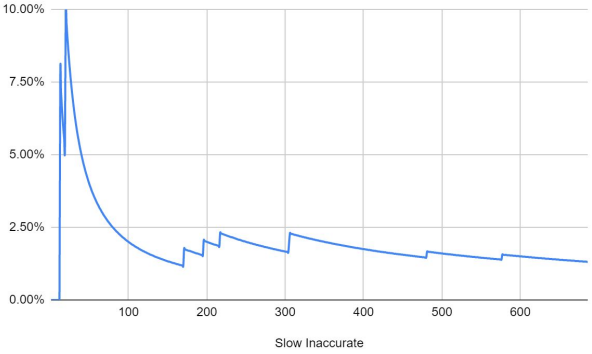
Final accuracy: 31.03%

Fast Inaccurate



Final accuracy: 2.28%

Slow Inaccurate



Final accuracy: 1.31%

Spike 15

Include "rate of fire" and reloading

I included a rate of fire and reloading into the shooting properties

```
        # Cool down timer between shots
        if self.cooling_down:
            self.fire_rate_tmr -= 1
        if self.fire_rate_tmr == 0:
            self.cooling_down = False
            self.fire_rate_tmr = self.fire_rate
```

Separation for multiple agents patrolling

After enough travelling, multiple patrolling agents would end up overlapping because they all had similar forces being applied. Hence, I used the separation technique from the previous lab to overcome this problem.

```
    def separate(self):
        # Moves away from nearby agent
        closeby = list(filter(lambda a: a is not self and (a.pos -
self.pos).length() < self.separation,self.world.agents))
        if (closeby):
            closest_agent_pos = min(closeby,key = lambda a: (a.pos -
self.pos).length()).pos
            target = (2 * self.pos - closest_agent_pos)
            to_target = target - self.pos
            length = to_target.copy().length()
            to_target.normalise()
            to_target *= self.separation - length
            return to_target
        return Vector2D()
```

Hiding behind dead bodies

When the agent is reloading, his mode is updated to hide, hence he would go and look for a dead body to hide behind:

```
    def hide(self):
        '''Goes to nearest dead body and hides behind it'''
        alive = list(filter(lambda e: e.alive, self.world.enemies))
        enemy = min(alive, key = lambda e: (e.pos - self.pos).length() *
e.health)
        dead = list(filter(lambda e: e.alive == False,
self.world.enemies))
        if dead:
```

```
            closest = min(dead, key = lambda e: (e.pos -
self.pos).length())
            # Uses angle to calculate best spot behind the hiding object
            position = enemy.pos - closest.pos
            position.normalise()
            hiding_point = closest.pos - closest.scale * 2 * position
            return self.arrive(hiding_point)
        return Vector2D()
```

Spike 16

Picking up weapons and body armour

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')
```