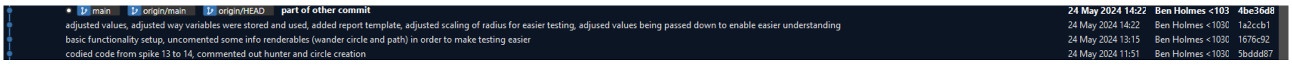
Spike: 15
Title: Agent Marksmanship

Author: Ben Holmes, 103024841

Goals / deliverables:
Create an agent targeting simulation with:
(a) an attacking agent (can be stationary),
(b) a moving target agent (can simply move between two-way points), and
(c) a selection of weapons that can fire projectiles with different properties.
Be able to demonstrate that the attacking agent that can successfully target (hit) with different weapon properties:
(a) Fast moving accurate projectile. (Rifle)
(b) Slow moving accurate projectile. (Rocket)
(c) Fast moving low accuracy projectile (Hand Gun)
(d) Slow moving low accuracy projectile (Hand grenade)

Technologies, Tools, and Resources used:
- Visual Studio Code
- Python 3.12.2
- Pyglet

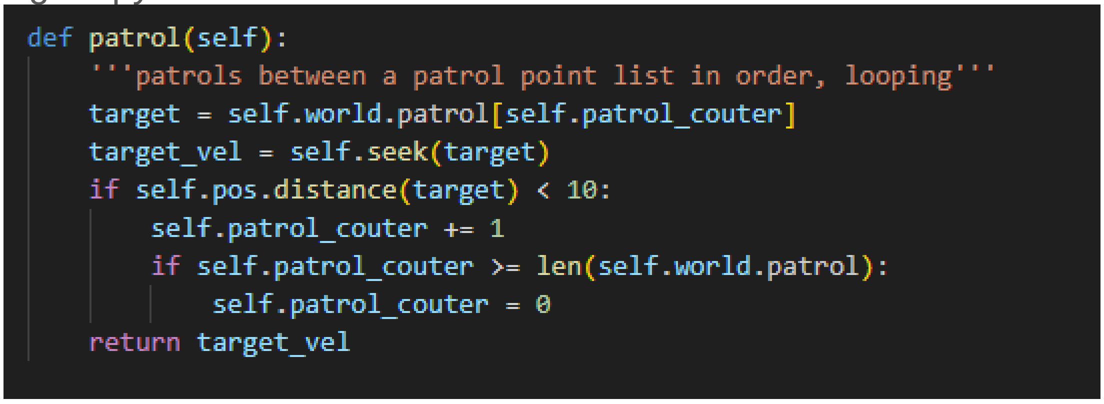
Tasks undertaken:



```
main  origin/main  origin/HEAD  part of other commit    24 May 2024 14:22  Ben Holmes <103  4be36d8
adjusted values, adjusted way variables were stored and used, added report template, adjusted scaling of radius for easier testing, adjused values being passed down to enable easier understanding    24 May 2024 14:22  Ben Holmes <1030  1a2ccb1
basic functionality setup, uncomented some info renderables (wander circle and path) in order to make testing easier    24 May 2024 13:15  Ben Holmes <1030  1676c92
codied code from spike 13 to 14, commented out hunter and circle creation    24 May 2024 11:51  Ben Holmes <1030  5bddd87
```

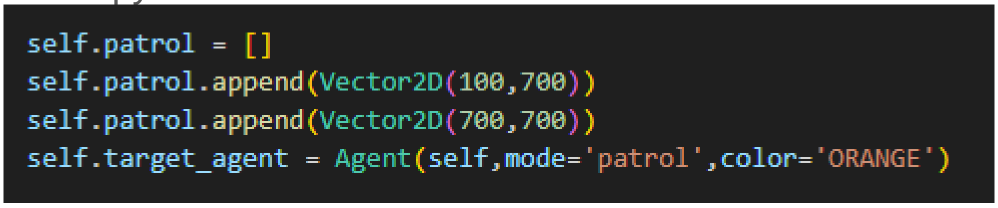
- Copied code from spike 13
- Commented out circle creation code
- Created the patrol functionality with preplace patrol points and added a target agent set to patrol

Agent.py



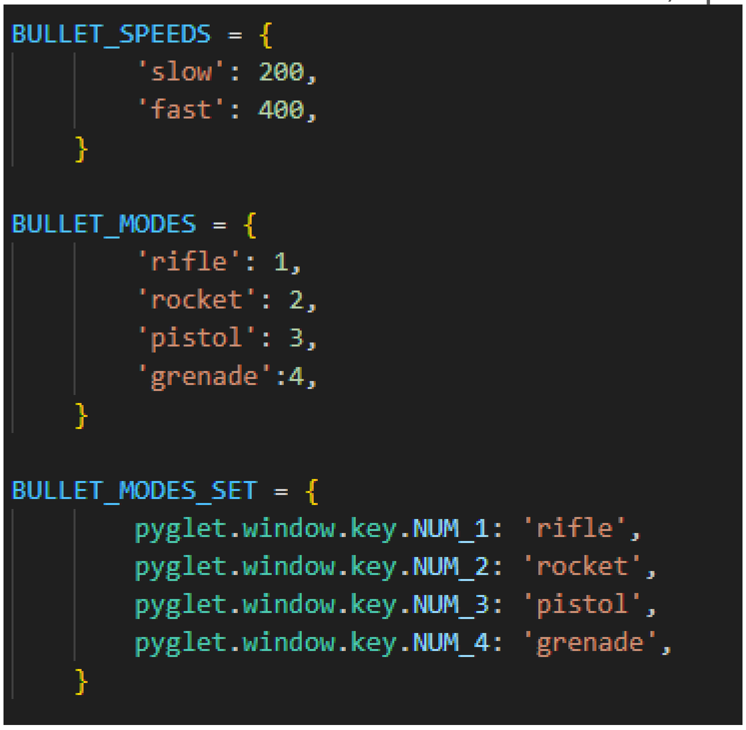
```python
def patrol(self):
    '''patrols between a patrol point list in order, looping'''
    target = self.world.patrol[self.patrol_couter]
    target_vel = self.seek(target)
    if self.pos.distance(target) < 10:
        self.patrol_couter += 1
        if self.patrol_couter >= len(self.world.patrol):
            self.patrol_couter = 0
    return target_vel
```

World.py



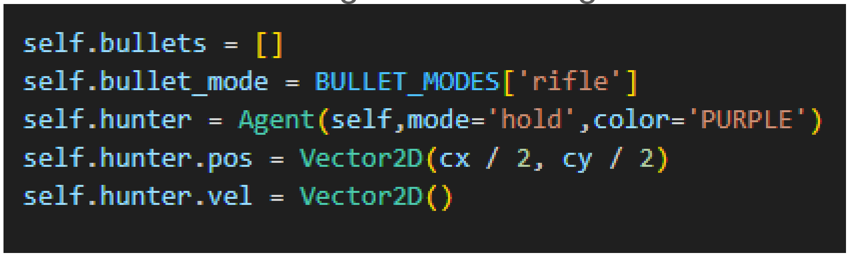
```
self.patrol = []
self.patrol.append(Vector2D(100,700))
self.patrol.append(Vector2D(700,700))
self.target_agent = Agent(self,mode='patrol',color='ORANGE')
```

- Created the variables for bullet mode, speed and mode selection



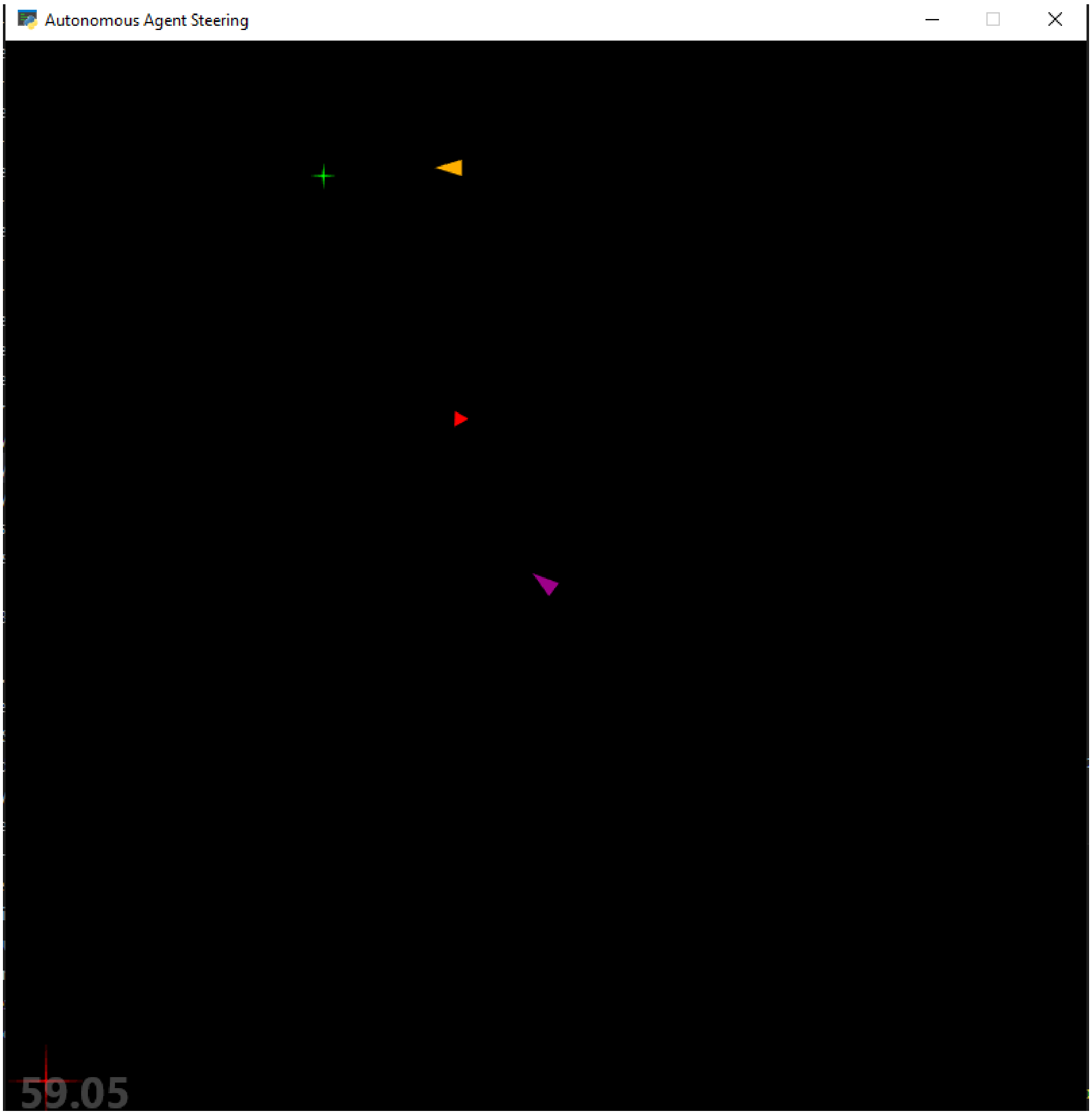
```
BULLET_SPEEDS = {
        'slow': 200,
        'fast': 400,
    }

BULLET_MODES = {
        'rifle': 1,
        'rocket': 2,
        'pistol': 3,
        'grenade':4,
    }

BULLET_MODES_SET = {
        pyglet.window.key.NUM_1: 'rifle',
        pyglet.window.key.NUM_2: 'rocket',
        pyglet.window.key.NUM_3: 'pistol',
        pyglet.window.key.NUM_4: 'grenade',
    }
```

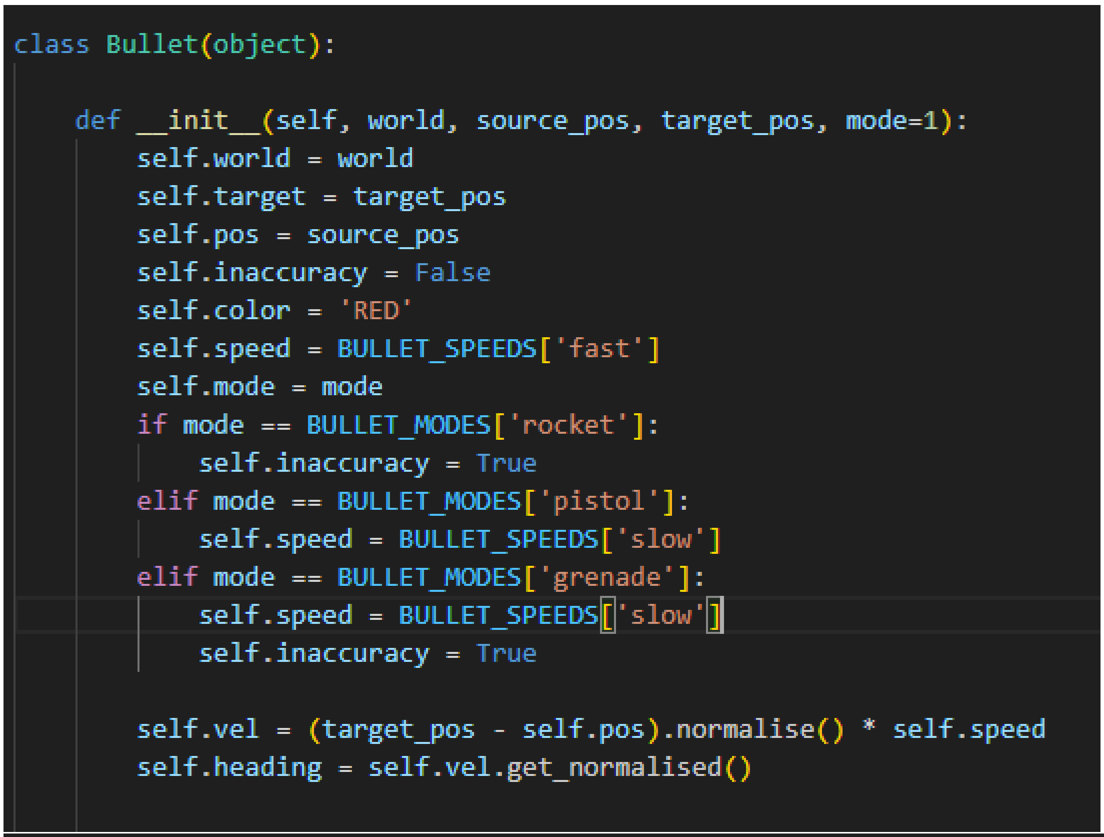
- Adjusted the hunter agent to start in a hold mode (no motion) in the centre of the screen and created an empty array for bullets and a world bullet mode storage for switching between



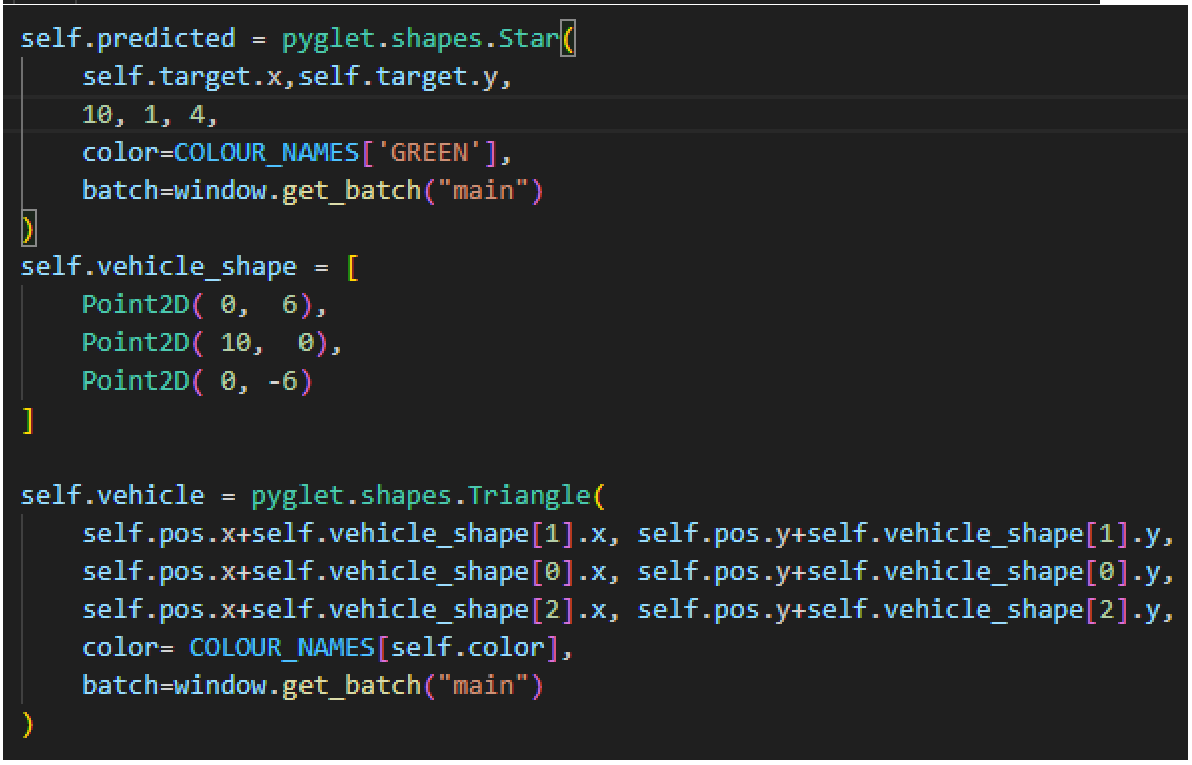
```
self.bullets = []
self.bullet_mode = BULLET_MODES['rifle']
self.hunter = Agent(self,mode='hold',color='PURPLE')
self.hunter.pos = Vector2D(cx / 2, cy / 2)
self.hunter.vel = Vector2D()
```

- Created the bullet class in separate file bullet.py with a vehicle of a smaller red arrow and a render of the predicted position as a green cross



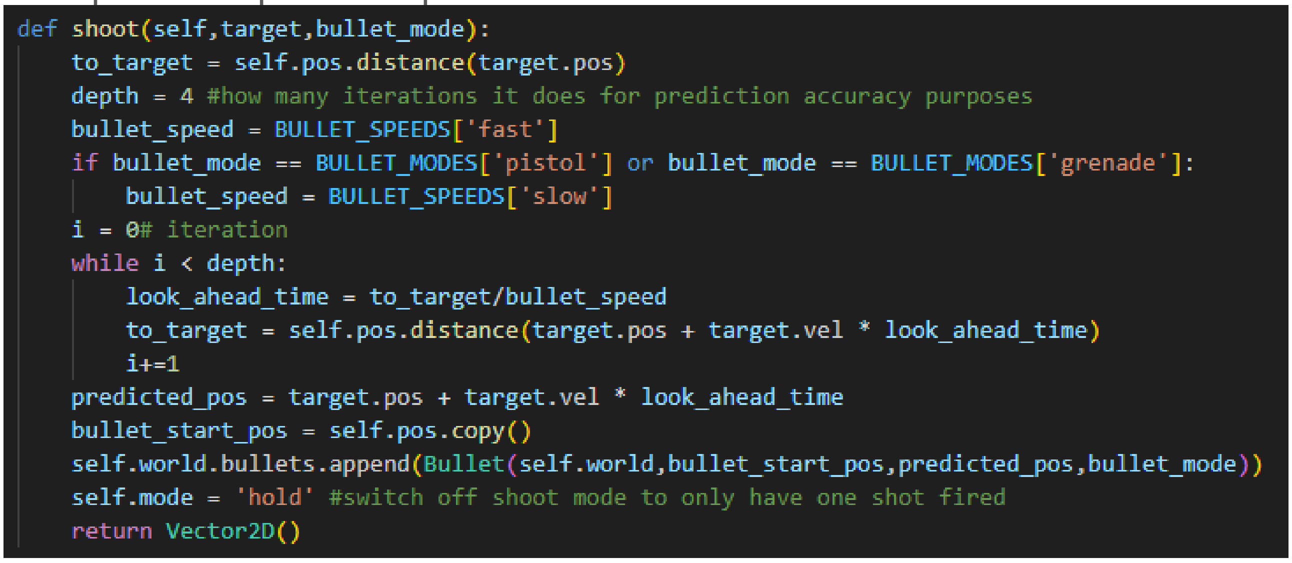


```python
class Bullet(object):

    def __init__(self, world, source_pos, target_pos, mode=1):
        self.world = world
        self.target = target_pos
        self.pos = source_pos
        self.inaccuracy = False
        self.color = 'RED'
        self.speed = BULLET_SPEEDS['fast']
        self.mode = mode
        if mode == BULLET_MODES['rocket']:
            self.inaccuracy = True
        elif mode == BULLET_MODES['pistol']:
            self.speed = BULLET_SPEEDS['slow']
        elif mode == BULLET_MODES['grenade']:
            self.speed = BULLET_SPEEDS['slow']
            self.inaccuracy = True

        self.vel = (target_pos - self.pos).normalise() * self.speed
        self.heading = self.vel.get_normalised()
```



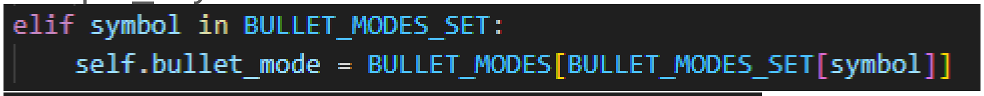
```python
self.predicted = pyglet.shapes.Star(
    self.target.x,self.target.y,
    10, 1, 4,
    color=COLOUR_NAMES['GREEN'],
    batch=window.get_batch("main")
)
self.vehicle_shape = [
    Point2D( 0,  6),
    Point2D( 10,  0),
    Point2D( 0, -6)
]

self.vehicle = pyglet.shapes.Triangle(
    self.pos.x+self.vehicle_shape[1].x, self.pos.y+self.vehicle_shape[1].y,
    self.pos.x+self.vehicle_shape[0].x, self.pos.y+self.vehicle_shape[0].y,
    self.pos.x+self.vehicle_shape[2].x, self.pos.y+self.vehicle_shape[2].y,
    color= COLOUR_NAMES[self.color],
    batch=window.get_batch("main")
)
```

- Added a shoot mode and function to the agent and adjusted the math around it to improve its aim including adding an iteration method to improve the predicted position

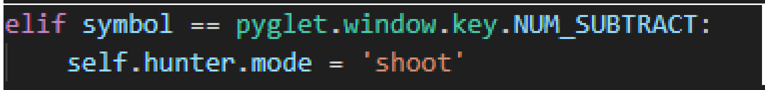


```python
def shoot(self,target,bullet_mode):
    to_target = self.pos.distance(target.pos)
    depth = 4 #how many iterations it does for prediction accuracy purposes
    bullet_speed = BULLET_SPEEDS['fast']
    if bullet_mode == BULLET_MODES['pistol'] or bullet_mode == BULLET_MODES['grenade']:
        bullet_speed = BULLET_SPEEDS['slow']
    i = 0# iteration
    while i < depth:
        look_ahead_time = to_target/bullet_speed
        to_target = self.pos.distance(target.pos + target.vel * look_ahead_time)
        i+=1
    predicted_pos = target.pos + target.vel * look_ahead_time
    bullet_start_pos = self.pos.copy()
    self.world.bullets.append(Bullet(self.world,bullet_start_pos,predicted_pos,bullet_mode))
    self.mode = 'hold' #switch off shoot mode to only have one shot fired
    return Vector2D()
```

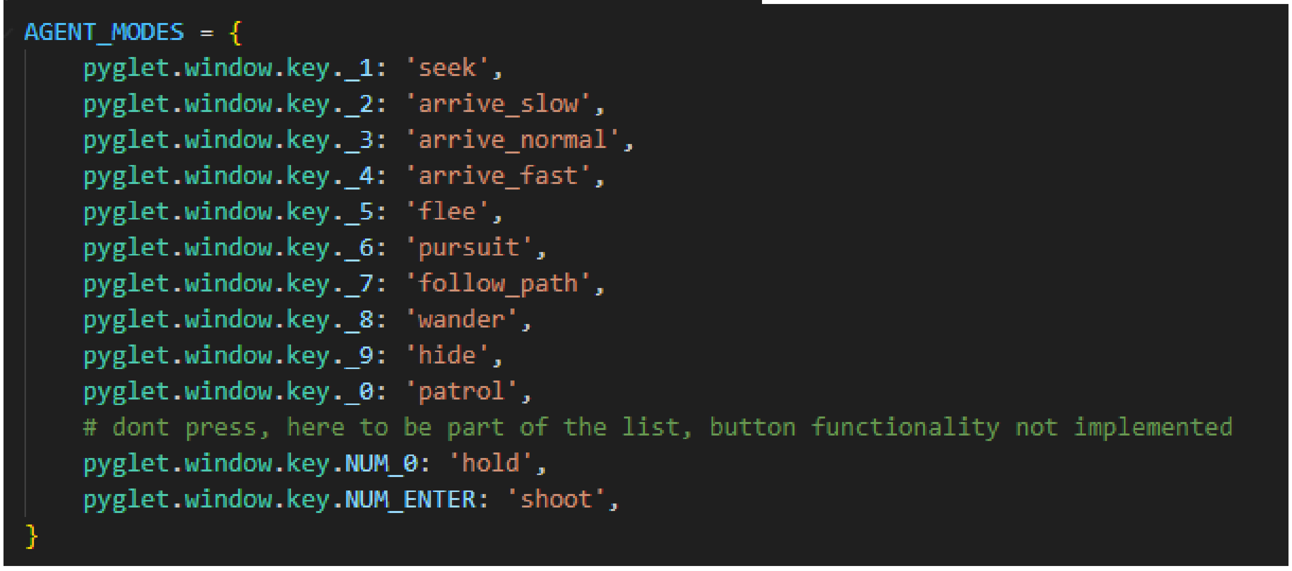
- Added the mode change buttons and shoot button to the world input_keyboard



```python
elif symbol in BULLET_MODES_SET:
    self.bullet_mode = BULLET_MODES[BULLET_MODES_SET[symbol]]
```

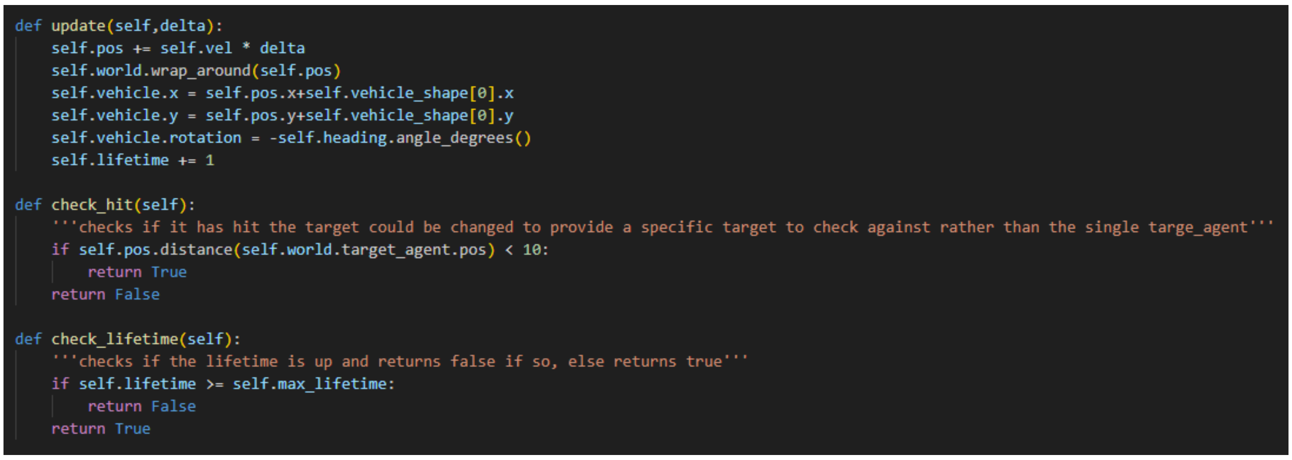


```python
elif symbol == pyglet.window.key.NUM_SUBTRACT:
    self.hunter.mode = 'shoot'
```



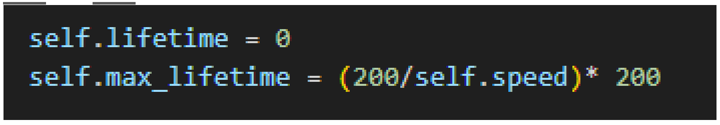
```python
AGENT_MODES = {
    pyglet.window.key._1: 'seek',
    pyglet.window.key._2: 'arrive_slow',
    pyglet.window.key._3: 'arrive_normal',
    pyglet.window.key._4: 'arrive_fast',
    pyglet.window.key._5: 'flee',
    pyglet.window.key._6: 'pursuit',
    pyglet.window.key._7: 'follow_path',
    pyglet.window.key._8: 'wander',
    pyglet.window.key._9: 'hide',
    pyglet.window.key._0: 'patrol',
    # dont press, here to be part of the list, button functionality not implemented
    pyglet.window.key.NUM_0: 'hold',
    pyglet.window.key.NUM_ENTER: 'shoot',
}
```

- Added update, check_hit and check_ lifetime to bullet to enable removal of bullet after set time (based on speed) and ability to check hit on bullet level



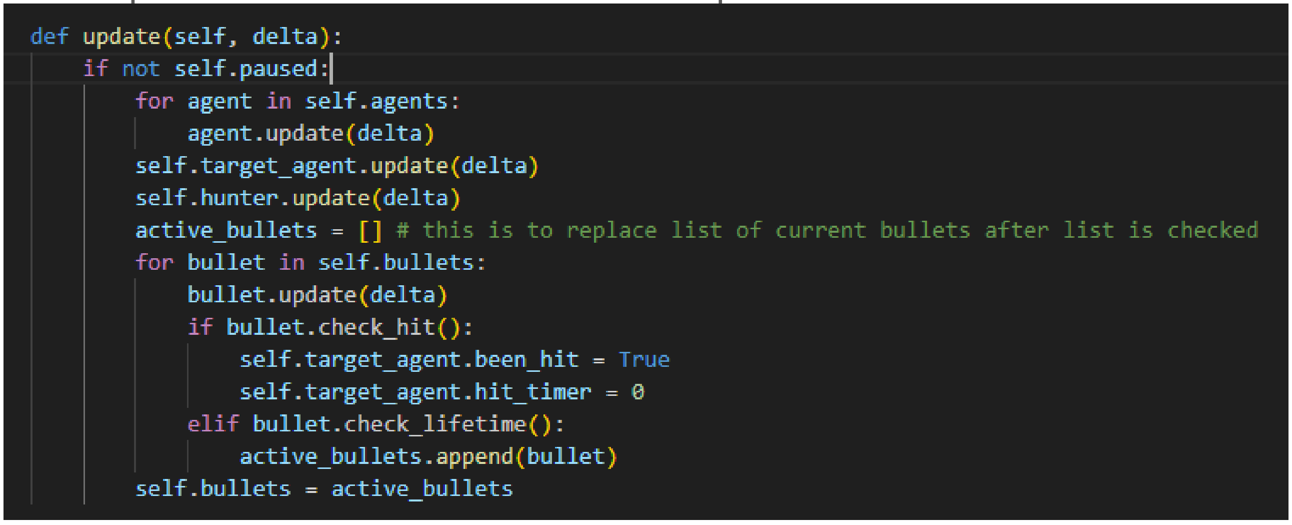
```
def update(self,delta):
    self.pos += self.vel * delta
    self.world.wrap_around(self.pos)
    self.vehicle.x = self.pos.x+self.vehicle_shape[0].x
    self.vehicle.y = self.pos.y+self.vehicle_shape[0].y
    self.vehicle.rotation = -self.heading.angle_degrees()
    self.lifetime += 1

def check_hit(self):
    '''checks if it has hit the target could be changed to provide a specific target to check against rather than the single targe_agent'''
    if self.pos.distance(self.world.target_agent.pos) < 10:
        return True
    return False

def check_lifetime(self):
    '''checks if the lifetime is up and returns false if so, else returns true'''
    if self.lifetime >= self.max_lifetime:
        return False
    return True
```

__init__



```
self.lifetime = 0
self.max_lifetime = (200/self.speed)* 200
```

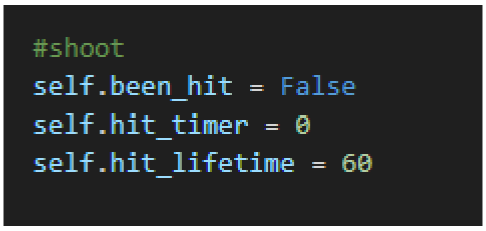
- Setup hit check and bullet life check in update of world



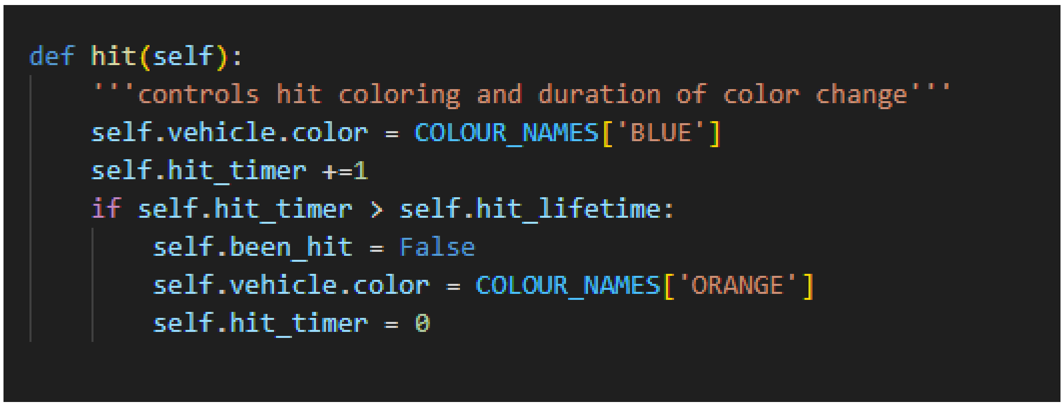
```
def update(self, delta):
    if not self.paused:
        for agent in self.agents:
            agent.update(delta)
        self.target_agent.update(delta)
        self.hunter.update(delta)
        active_bullets = [] # this is to replace list of current bullets after list is checked
        for bullet in self.bullets:
            bullet.update(delta)
            if bullet.check_hit():
                self.target_agent.been_hit = True
                self.target_agent.hit_timer = 0
            elif bullet.check_lifetime():
                active_bullets.append(bullet)
        self.bullets = active_bullets
```

- Added hit functionality to agent.

Init

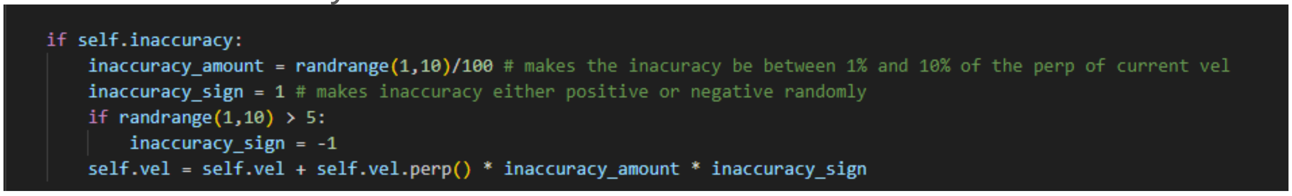


```
#shoot
self.been_hit = False
self.hit_timer = 0
self.hit_lifetime = 60
```



```python
def hit(self):
    '''controls hit coloring and duration of color change'''
    self.vehicle.color = COLOUR_NAMES['BLUE']
    self.hit_timer +=1
    if self.hit_timer > self.hit_lifetime:
        self.been_hit = False
        self.vehicle.color = COLOUR_NAMES['ORANGE']
        self.hit_timer = 0
```

- Added inaccuracy to bullet in bullet init



```python
if self.inaccuracy:
    inaccuracy_amount = randrange(1,10)/100 # makes the inacuracy be between 1% and 10% of the perp of current vel
    inaccuracy_sign = 1 # makes inaccuracy either positive or negative randomly
    if randrange(1,10) > 5:
        inaccuracy_sign = -1
    self.vel = self.vel + self.vel.perp() * inaccuracy_amount * inaccuracy_sign
```

-

What we found out:

Patrol was incredibly easy to implement, as was the bullet class and basic functionality of getting it to shoot at a point. Getting the hit accuracy when the bullet was accurate was difficult however.
It was adjusting the code to have an iterative calculation and adjusting the pursuit provided calculation to use distance divided by only bullet speed that allowed the accuracy to be good, it has trouble with the accelerating target but once the velocity is stable its perfectly accurate.
To help with the accuracy while accelerating I adjust the agent to be a little slower and doubled the speed of the bullets that I originally selected.

Figuring out how to implement inaccuracy was a little difficult as adjusting the angle of the vector directly was too tricky so instead I added a 1-10% strength positive or negative perp of the bullet vector to add the inaccuracy which can be seen by looking at the predicted green star

Testing buttons:

Numpad minus shoots a bullet and numpad:
1: rifle (fast accurate)
2: rocket (fast inaccurate)
3: pistol (slow accurate)
4: grenade (slow inaccurate)