Spike: Spike Week 15
Title: Agent Marksmanship
Author: Nguyen Khanh Toan - 104180605

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)

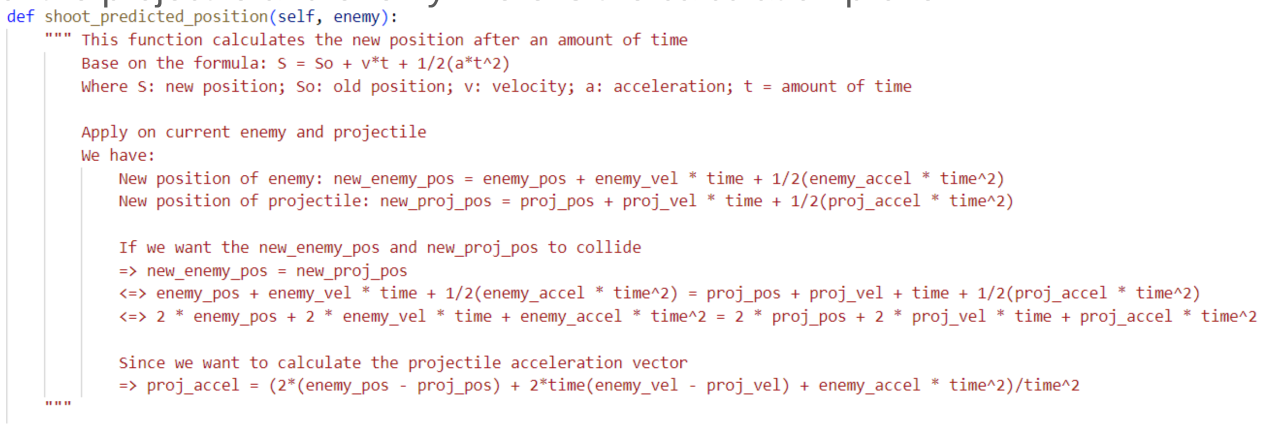
Technologies, Tools, and Resources used:
- Visual Studio Code
- Python 3.12

Tasks undertaken:
- Install Python 3+, Pyglet 2+
- Install and setup compatible IDE for the language, e.g.: Visual Studio Code
- Pay attention to the comment of how the code work and functionality. Can use debug tool to observe the program more clearly.
- Run the code and observing the output.

Planning Notes:
The attacking agent can be static, and the target can be a simple.
• Create a simple weapon for the agent that can shoot projectiles of different types. The projectile (bullet) will
need to be moved each update step and tested for collisions with a target.
• Start by shooting accurate projectiles at a fixed target. Change the state of the target so that a "hit" can be clearly seen.
• Create "inaccurate" projectiles that don't always go where they are aimed.
• Move the target. The shooting agent will need to use the targets position and velocity, together with the
projectile speed, to work out an interception point to shoot at.centre position etc as needed

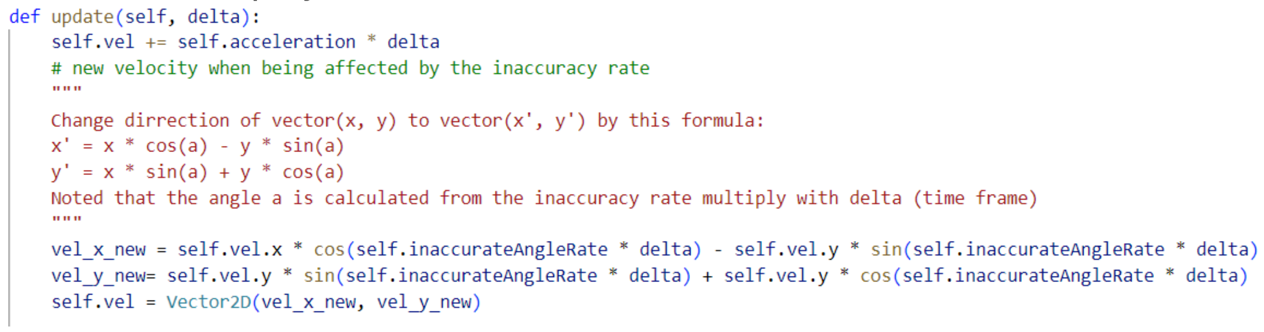
What we found out:

- The program enemy can be add by pressing A, shoot with SPACE and change weapon by C.
- The program now adds two new class, which projectile (bullet), Hunter (Marksman) and modified Enemy class that inherit from Agent class.
- The enemy will follow the path that generated, and the hunter will try to shoot down the enemy.
- In addition to the requirement of the task, which calculate collision base on target position and its velocity, I have found out the way to make it more precise by incorporate with acceleration. The hunter can calculate enemy future position by applying physics formula S = So + vt + (at^2)/2 to calculate acceleration vector for the projectile to the collision of the projectile and enemy. Here is the calculation prove:



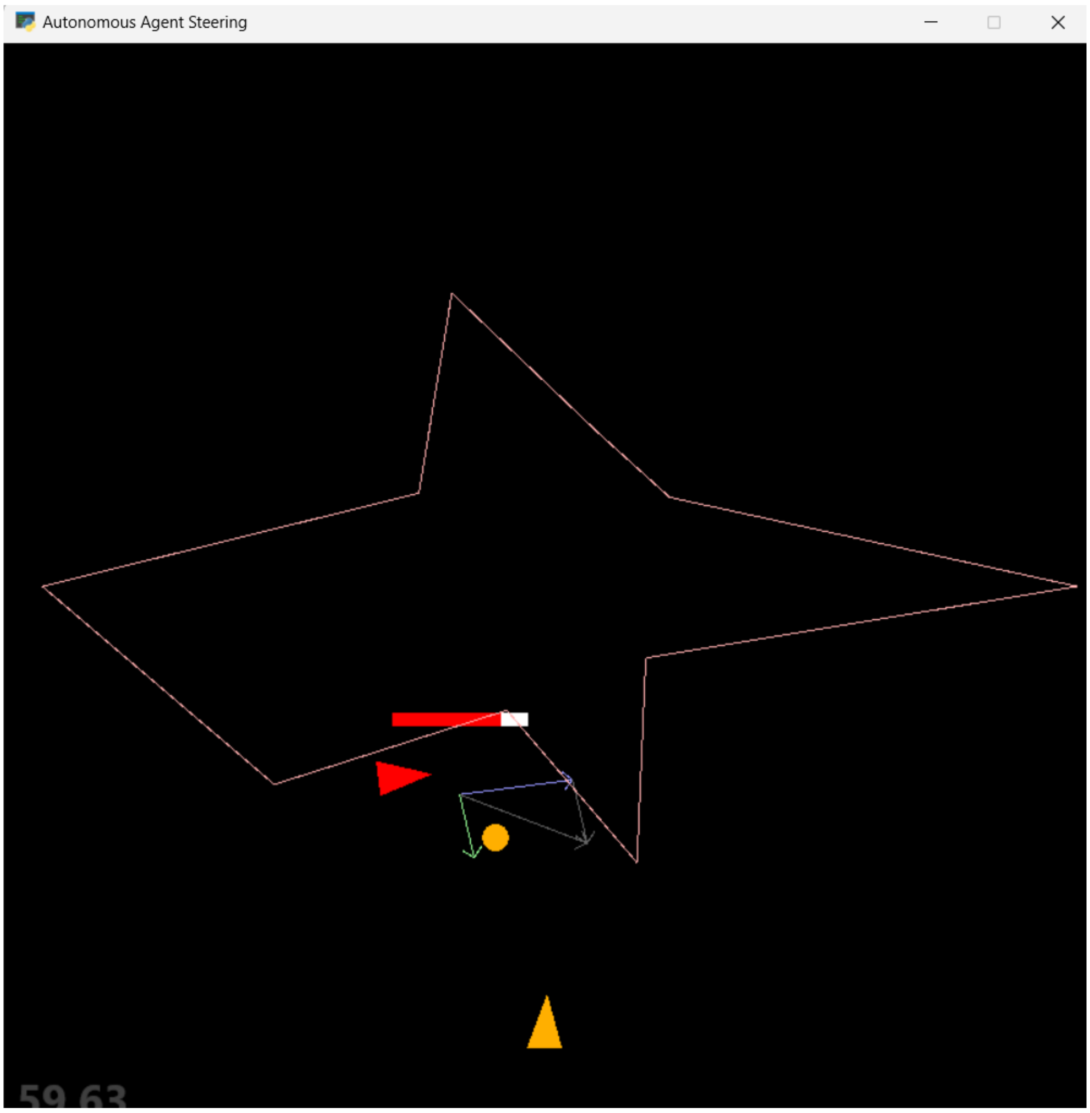
```
def shoot_predicted_position(self, enemy):
    """ This function calculates the new position after an amount of time
        Base on the formula: S = So + v*t + 1/2(a*t^2)
        Where S: new position; So: old position; v: velocity; a: acceleration; t = amount of time

        Apply on current enemy and projectile
        We have:
            New position of enemy: new_enemy_pos = enemy_pos + enemy_vel * time + 1/2(enemy_accel * time^2)
            New position of projectile: new_proj_pos = proj_pos + proj_vel * time + 1/2(proj_accel * time^2)

            If we want the new_enemy_pos and new_proj_pos to collide
            => new_enemy_pos = new_proj_pos
            <=> enemy_pos + enemy_vel * time + 1/2(enemy_accel * time^2) = proj_pos + proj_vel + time + 1/2(proj_accel * time^2)
            <=> 2 * enemy_pos + 2 * enemy_vel * time + enemy_accel * time^2 = 2 * proj_pos + 2 * proj_vel * time + proj_accel * time^2

            Since we want to calculate the projectile acceleration vector
            => proj_accel = (2*(enemy_pos - proj_pos) + 2*time(enemy_vel - proj_vel) + enemy_accel * time^2)/time^2
    """
```

- Each weapon has each inaccuracy rate that affected on the acceleration of projectile.



```
def update(self, delta):
    self.vel += self.acceleration * delta
    # new velocity when being affected by the inaccuracy rate
    """
    Change direction of vector(x, y) to vector(x', y') by this formula:
    x' = x * cos(a) - y * sin(a)
    y' = x * sin(a) + y * cos(a)
    Noted that the angle a is calculated from the inaccuracy rate multiply with delta (time frame)
    """
    vel_x_new = self.vel.x * cos(self.inaccurateAngleRate * delta) - self.vel.y * sin(self.inaccurateAngleRate * delta)
    vel_y_new= self.vel.y * sin(self.inaccurateAngleRate * delta) + self.vel.y * cos(self.inaccurateAngleRate * delta)
    self.vel = Vector2D(vel_x_new, vel_y_new)
```

- Enemy also has the health bar that indicated the health for it. When the projectile hit the target, the health bar will be changed prior to the enemy's current health.


Autonomous Agent Steering
59.63