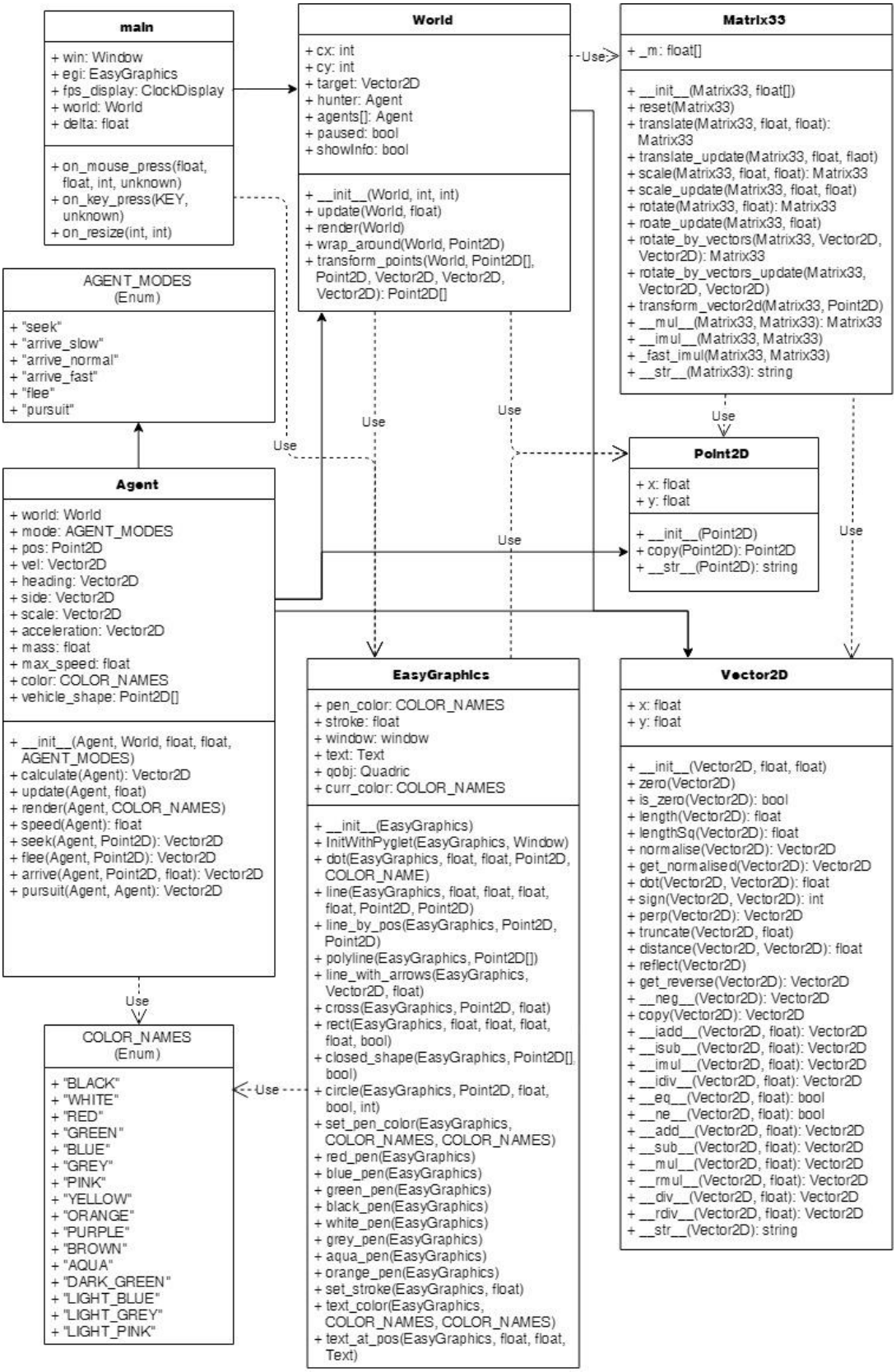
Task 8: Autonomous Steering

UML Diagram



Lab Coding

Multiple, Varied Agents

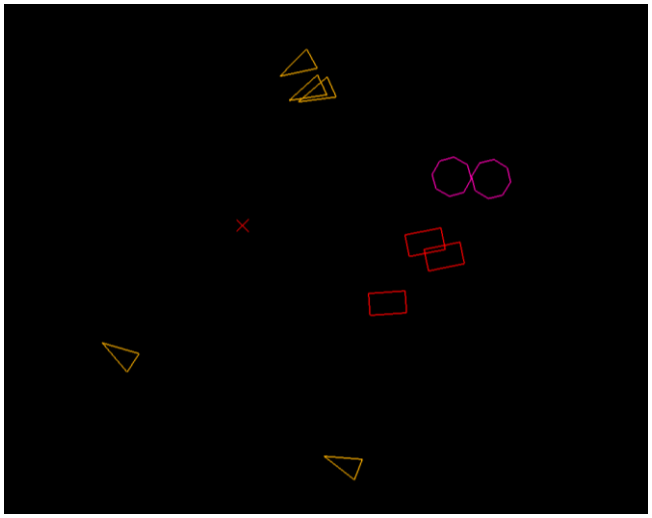


Figure 1: A screenshot of the program featuring multiple agents of different shapes. Each has different attributes for mass and max forward / sideways speed.

```
def apply_friction(self):
    future_pos = self.pos + self.vel * 0.1
    accel_to_future_pos = self.seek(future_pos)
    friction = accel_to_future_pos * -self.friction
    return friction
```

Figure 2: The apply_friction method

```
if model == "dart":
    self.mass = 1.0
    # limits?
    self.max_forward_speed = 5000.0
    self.max_sideways_speed = 4000.0
    self.max_reverse_speed = 1000.0
    self.friction = 0.1
    # data for drawing this agent
    self.color = 'ORANGE'
    self.vehicle_shape = [
        Point2D(-1.0,  0.6),
        Point2D( 1.0,  0.0),
        Point2D(-1.0, -0.6)
    ]
elif model == "block":
    self.mass = 1.5
    # limits?
    self.max_forward_speed = 4000.0
    self.max_sideways_speed = 3200.0
    self.max_reverse_speed = 1000.0
    self.friction = 0.2
    # data for drawing this agent
    self.color = 'RED'
    self.vehicle_shape = [
        Point2D(-1.0,  0.6),
        Point2D( 1.0,  0.6),
        Point2D( 1.0, -0.6),
        Point2D(-1.0, -0.6)
    ]
else:
    self.mass = 2.0
    # limits?
    self.max_forward_speed = 3000.0
    self.max_sideways_speed = 2400.0
    self.max_reverse_speed = 1000.0
    self.friction = 0.3
    # data for drawing this agent
    self.color = 'PURPLE'
    self.vehicle_shape = [
        Point2D( 0.4,  1.0),
        Point2D(-0.4,  1.0),
        Point2D(-1.0,  0.4),
        Point2D(-1.0, -0.4),
        Point2D(-0.4, -1.0),
        Point2D( 0.4, -1.0),
        Point2D( 1.0, -0.4),
        Point2D( 1.0,  0.4)
    ]
```

Figure 3: The attributes of each model of agent.

Flee with Panic Distance

```
def flee(self, hunter_pos):
    ''' move away from hunter position '''
    panic_range = 100

    if self.distance(hunter_pos) > panic_range:
        return Vector2D(0, 0)

    desired_vel = (self.pos - hunter_pos).normalise() * self.max_forward_speed
    return (desired_vel - self.vel)
```

Figure 4: the flee method adapted from the lecture slides, with panic distance implemented.

Deceleration Speeds

```
def calculate(self):
    # reset the steering force
    mode = self.mode
    if mode == 'seek':
        accel = self.seek(self.world.target)
    elif mode == 'arrive_slow':
        accel = self.arrive(self.world.target, 'slow')
    elif mode == 'arrive_normal':
        accel = self.arrive(self.world.target, 'normal')
    elif mode == 'arrive_fast':
        accel = self.arrive(self.world.target, 'fast')
    elif mode == 'flee':
        accel = self.flee(self.world.target)
    elif mode == 'pursuit':
        accel = self.pursuit(self.world.evader)
    else:
        accel = Vector2D()
    self.acceleration = accel
    return accel
```

Figure 5: the arrive method being called with different deceleration speeds implemented successfully.

```
DECELERATION_SPEEDS = {
    'slow': 0.9,
    'normal': 0.6,
    'fast': 0.3
}
```

Figure 6: the deceleration speeds.

Pursuit

```
def pursuit(self, evader):
    ''' this behaviour predicts where an agent will be in time T and seeks
        towards that point to intercept it. '''
    to_evader = evader.pos - self.pos
    relative_heading = self.heading.dot(evader.heading)

    if (to_evader.dot(self.heading) > 0) and (relative_heading < 0.95):
        return self.seek(evader.pos)

    future_time = to_evader.length()/(self.max_forward_speed + evader.speed())
    #future_time += (1 - self.heading.dot(evader.vel))
    future_pos = evader.pos + evader.vel * future_time
    return self.seek(future_pos)
```

Figure 7: the pursuit code adapted from the lecture slides.

```
elif symbol in AGENT_MODES:
    mode = AGENT_MODES[symbol]

    if mode == 'pursuit' and len(world.agents) == 1:
        return

    world.agent_mode = mode

    for agent in world.agents:
        agent.mode = mode

    if mode == 'pursuit':
        world.agents[0].mode = 'flee'
        world.evader = world.agents[0]
```

Figure 8: the code setting the pursuing agents to pursue world.agents[0], which is made to flee the mouse clicks.

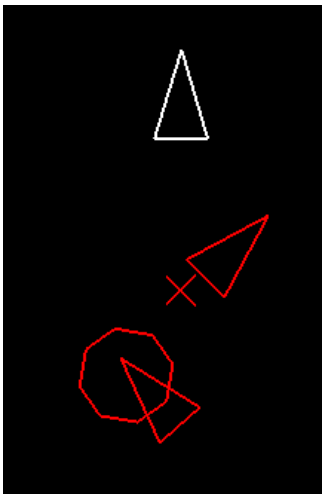


Figure 9: a poor triangle being harassed by two triangles, an octagon, and a mouse-spawn X.

Varied steering limits

```
def enforce_speed_limit(self, vel):
    result = vel

    if result.x >= 0:
        result.x = min(result.x, self.max_forward_speed)
    else:
        result.x = max(result.x, -self.max_reverse_speed)   # max_reverse_speed is asigned as +ve
                                                            # when applying, it needs to be -ve
    if result.y >= 0:
        result.y = min(result.y, self.max_sideways_speed)
    else:
        result.y = max(result.y, -self.max_sideways_speed)

    return result
```

Figure 10: the method for enforcing varied steering and speed limits. See fig. 3 for the values assigned to each model of agent.