Spike: Spike Week 14
Title: Emergent Group Behaviour
Author: Nguyen Khanh Toan - 104180605

Goals / deliverables:
Create a group agent steering behaviour simulation that is able to demonstrate distinct modes of emergent group behaviour. In particular, the simulation must:
• Include cohesion, separation and alignment steering behaviours
• Include basic wandering behaviours
• Use a weighted-sum to combine all steering behaviours
• Support the adjustment of parameters for each steering force while running
• Spike outcome report and working code (with key instructions).

Technologies, Tools, and Resources used:

- Visual Studio Code
- Python 3.12

Tasks undertaken:

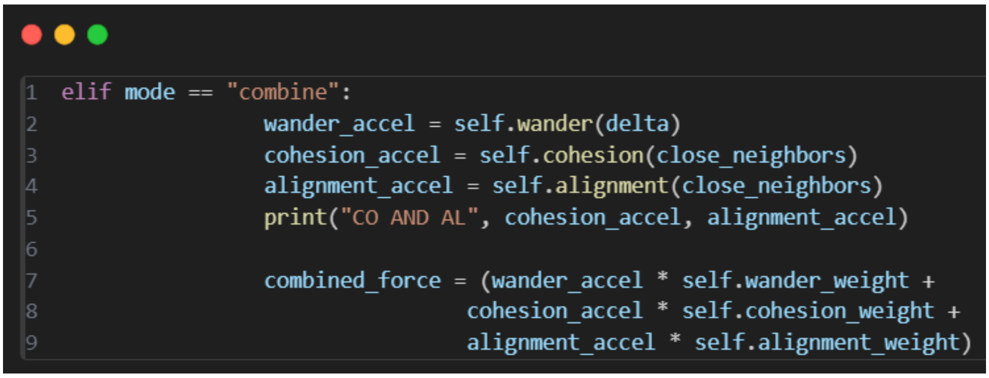
- Install Python 3+
- Install and setup compatible IDE for the language, e.g.: Visual Studio Code
- Pay attention to the comment of how the code work and functionality. Can use debug tool to observe the program more clearly.
- Run the code and observing the output.

Planning Notes:
• Use the existing lab code, copy and create a new project
• Extend the code to support multiple agents and new keyboard input
• Create display code that can show the current parameter values you need
• Create code that can, for each agent, identify its immediate "neighbours" and gather the average heading, the
centre position etc as needed

What we found out:

Combine force: In the calculation function, it will calculate the force of a group of agents. The forces are the combination of wander force, cohesion force and alignment force, using the formular F = m * a where F: force; m: weight; a: acceleration of the object. This a group of agents have the same vector of direction

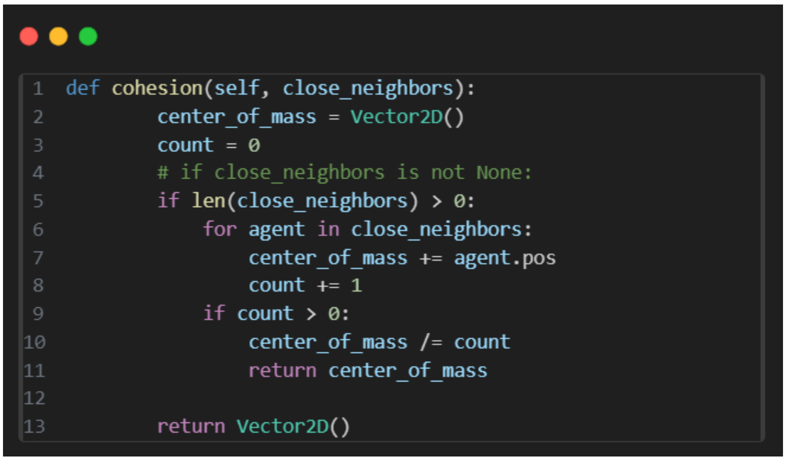
The wander function, introduced in previous tasks, introduces random behavior by initially adding a small random vector to the agent's current target position. This vector is then adjusted to fall within a unit circle with the same radius as the wander circle. Finally, the target position is moved ahead of the agent, and the agent seeks toward this new target.



```
elif mode == "combine":
                wander_accel = self.wander(delta)
                cohesion_accel = self.cohesion(close_neighbors)
                alignment_accel = self.alignment(close_neighbors)
                print("CO AND AL", cohesion_accel, alignment_accel)

                combined_force = (wander_accel * self.wander_weight +
                                cohesion_accel * self.cohesion_weight +
                                alignment_accel * self.alignment_weight)
```

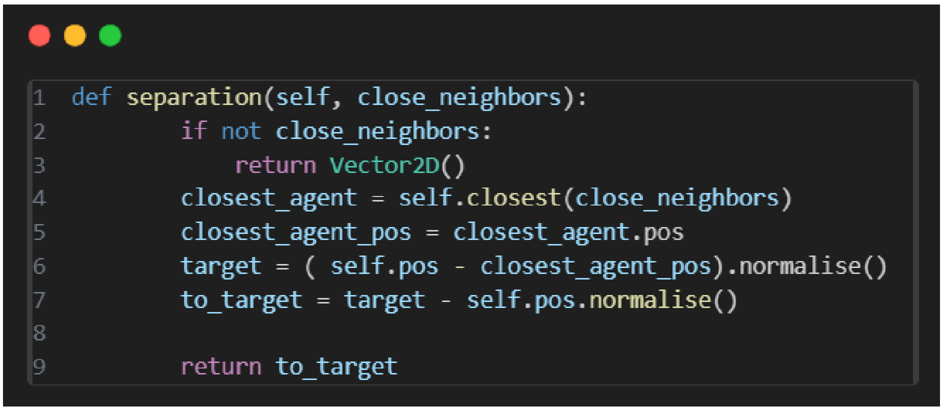
Cohesion: The cohesion function determines the centre point of nearby agents and provides a direction toward that point. This enables agents to move together in groups.



```
def cohesion(self, close_neighbors):
        center_of_mass = Vector2D()
        count = 0
        # if close_neighbors is not None:
        if len(close_neighbors) > 0:
            for agent in close_neighbors:
                center_of_mass += agent.pos
                count += 1
            if count > 0:
                center_of_mass /= count
                return center_of_mass

        return Vector2D()
```

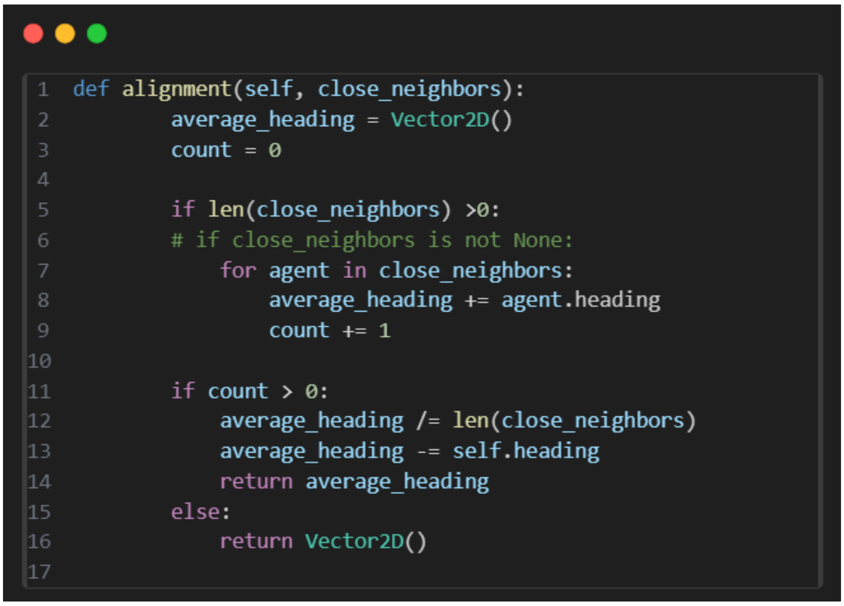
Separation: Separation identifies the nearest neighbour among nearby agents and calculates the force needed to push the agent away from that neighbour. This behaviour helps agents avoid collisions and maintain a safe distance from each other.



```python
def separation(self, close_neighbors):
        if not close_neighbors:
            return Vector2D()
        closest_agent = self.closest(close_neighbors)
        closest_agent_pos = closest_agent.pos
        target = ( self.pos - closest_agent_pos).normalise()
        to_target = target - self.pos.normalise()

        return to_target
```

Alignment: Alignment computes the average direction of neighbouring agents and adjusts the agent's heading to match that average direction. This ensures that agents align their movements with those of their neighbours.



```python
def alignment(self, close_neighbors):
        average_heading = Vector2D()
        count = 0

        if len(close_neighbors) >0:
        # if close_neighbors is not None:
            for agent in close_neighbors:
                average_heading += agent.heading
                count += 1

        if count > 0:
            average_heading /= len(close_neighbors)
            average_heading -= self.heading
            return average_heading
        else:
            return Vector2D()

```