Spike: 14
Title: Emergent Group Behaviour

Author: Luke Valentino, 103024456

Goals / deliverables:
The goal of this spike is to implement three distinct emergent group behaviours using basic steering forces and parameter values.

Technologies, Tools, and Resources used:
List of information needed by someone trying to reproduce this work
- Swinburne Lecture Notes
- Swinburne Maths and Physics PDF
- Docs.python.org

Tasks undertaken:
First we must copy over old code to use as a base.
- Then we have to create functionality to be able to get the neighbours of each agent.
  - In agent.py add a list of neighbours and a neighbour's radius.

```
-           self.neighbours = []
-           self.neighbour_radius = 200.0
```

  - Then we can add a method to agent to add any surrounding agents to our neighbours list.

```
-       def TagNeighbours(self, bots, radius):
-           """ Tag neighbours within a given radius """
-           self.neighbours = []
-           for bot in bots:
-               if bot != self:
-                   to = self.pos - bot.pos
-                   gap = radius + bot.bRadius
-                   if to.lengthSq() < gap**2:
-                       self.neighbours.append(bot)
```

  - Then in world.py add a method to call this for every agent.

```
-       def calculate_neighbours(self):
-           for agent in self.agents:
-               agent.TagNeighbours(self.agents, agent.neighbour_radius)
```

- Now that we have a way to determine each agents neighbours, we can begin adding the group behaviours.

  - First let's add separation to agent.py:

```
-       def separation(self, group):
        SteeringForce = Vector2D()
        for bot in group:
            # Don’t include self, only include neighbors (already tagged)
            if bot != self and bot in self.neighbours:
                ToBot = self.pos - bot.pos
                # Scale based on inverse distance to neighbor
                if ToBot.lengthSq() > 0:  # Avoid division by zero
                    SteeringForce += ToBot.normalise() / ToBot.length()
        return SteeringForce
```

- Now for Cohesion:

```
    def cohesion(self, group):
        CentreMass = Vector2D()
        SteeringForce = Vector2D()
        AvgCount = 0
        for bot in group:
            if bot != self and bot in self.neighbours:
                CentreMass += bot.pos
                AvgCount += 1
        if AvgCount > 0:
            CentreMass /= float(AvgCount)
            SteeringForce = self.seek(CentreMass)

        return SteeringForce
```

- And Alignment:

```
    def alignment(self, group):
        AverageHeading = Vector2D()
        NeighborCount = 0

        for bot in group:
            # Don’t include self, only include neighbors
            if bot != self and bot in self.neighbours:
                AverageHeading += bot.heading
                NeighborCount += 1

        if NeighborCount > 0:
            AverageHeading /= NeighborCount
            AverageHeading = AverageHeading.normalise() * self.max_speed
            AverageHeading -= self.vel

        return AverageHeading
```

- Let’s update the calculate method in agent to use these new methods

```python
    def calculate(self, delta):
        # Initialize the steering force
        SteeringForce = Vector2D()

        if self.mode == 'wander':
            # Apply Wander behavior
            SteeringForce += self.wander(delta) * self.wander_amount

        elif self.mode == 'separation':
            # Apply Separation behavior
            SteeringForce += self.separation(self.neighbours) *
self.separation_amount

        elif self.mode == 'alignment':
            # Apply Alignment behavior
            SteeringForce += self.alignment(self.neighbours) *
self.alignment_amount

        elif self.mode == 'cohesion':
            # Apply Cohesion behavior
            SteeringForce += self.cohesion(self.neighbours) *
self.cohesion_amount

        elif self.mode == 'combined':
            # Apply all behaviors with weighted sum
            SteeringForce += self.wander(delta) * self.wander_amount
            SteeringForce += self.separation(self.neighbours) *
self.separation_amount
            SteeringForce += self.alignment(self.neighbours) *
self.alignment_amount
            SteeringForce += self.cohesion(self.neighbours) *
self.cohesion_amount

        # Truncate the steering force to the maximum allowed
        SteeringForce.truncate(self.max_force)

        return SteeringForce
```

- Lets create our parameters for our behaviours.

```python
        # Parameters for steering behaviors
        self.wander_amount = 1.0
        self.separation_amount = 75.0
        self.alignment_amount = 100.0
        self.cohesion_amount = 50.0
```

- Now we can update our agent_modes to reflect these changes:

```
-   AGENT_MODES = {
-       KEY._1: 'wander',
-       KEY._2: 'separation',
-       KEY._3: 'alignment',
-       KEY._4: 'cohesion',
-       KEY._5: 'combined'
-   }
-
```

- Then finally we can display some information on screen and allow for dynamic parameter adjustment.
  - In main.py create this.

```
-
-   PARAMETER_KEYS = {
-       KEY.F1: 'wander',
-       KEY.F2: 'separation',
-       KEY.F3: 'alignment',
-       KEY.F4: 'cohesion'
-   }
```

  - Add this method to main.py as well.

```
-   def adjust_parameter(agents, param, amount):
-       for agent in agents:
-           if param == 'wander':
-               agent.wander_amount += amount
-           elif param == 'separation':
-               agent.separation_amount += amount
-           elif param == 'alignment':
-               agent.alignment_amount += amount
-           elif param == 'cohesion':
-               agent.cohesion_amount += amount
```

  - Then adjust the on_key_press method:

```
-   def on_key_press(symbol, modifiers):
-       if symbol == KEY.P:
-           world.paused = not world.paused
-       elif symbol in AGENT_MODES:
-           for agent in world.agents:
-               agent.mode = AGENT_MODES[symbol]
-       elif symbol in PARAMETER_KEYS:
-           world.selected_param = PARAMETER_KEYS[symbol]
-       elif symbol == KEY.UP:
-           adjust_parameter(world.agents, world.selected_param, 1)
-       elif symbol == KEY.DOWN:
-           adjust_parameter(world.agents, world.selected_param, -1)
-       elif symbol == KEY.A:
-           world.agents.append(Agent(world))
-       elif symbol == KEY.I:
-           for agent in world.agents:
-               agent.show_info = not agent.show_info
```

- And lastly update the render method in world.py to display the information.

```
-       def render(self):
-           for agent in self.agents:
-               agent.render()
-
-           if self.target:
-               egi.red_pen()
-               egi.cross(self.target, 10)
-           if self.show_info:
-                   infotext = ', '.join(set(agent.mode for agent in
   self.agents))
-                   infotext += f" | Wander: {self.agents[0].wander_amount:.2f}"
-                   infotext += f" | Separation:
   {self.agents[0].separation_amount:.2f}"
-                   infotext += f" | Alignment:
   {self.agents[0].alignment_amount:.2f}"
-                   infotext += f" | Cohesion:
   {self.agents[0].cohesion_amount:.2f}"
-                   infotext += f" | Selected Parameter: {self.selected_param}"
-                   egi.white_pen()
-                   egi.text_at_pos(10, self.cy - 20, infotext)  # Position text
   at the top of the screen
```

What we found out:
In this spike I successfully implemented the three emergent group behaviours.

Agents will initially be set to wander.

To change them to desired behaviours:

Press Key 1 for Wander

Press Key 2 for Separation

Press Key 3 for Alignment

Press Key 4 for Cohesion

Press Key 5 for Combined

To adjust the parameter values:

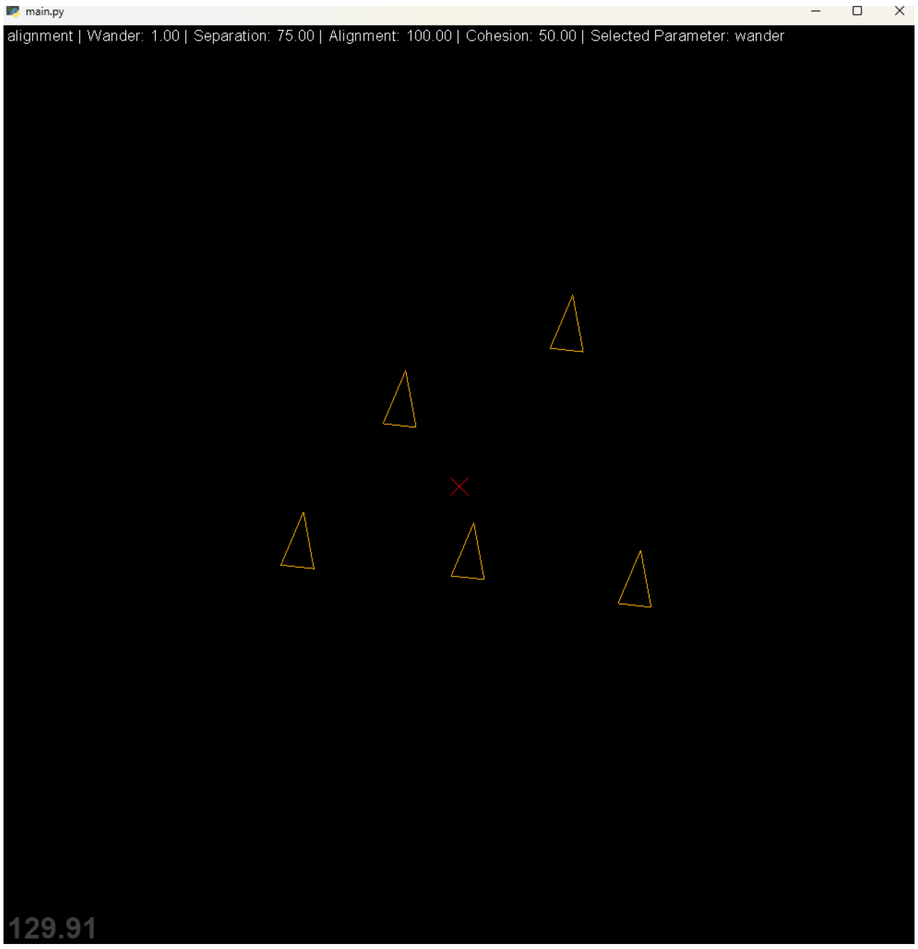
Press F1 to adjust Wander parameter.

Press F2 to adjust Separation parameter.

Press F3 to adjust Alignment parameter.

Press F4 to adjust Cohesion to parameter.

After the desired parameter is selected, use the up and down arrow keys to alter the values.



```
alignment | Wander: 1.00 | Separation: 75.00 | Alignment: 100.00 | Cohesion: 50.00 | Selected Parameter: wander
129.91
```

The outcomes of this task are directly related to several ULO's:
ULO 1: Discuss and Implement software development techniques to support the creation of AI behaviour in game.

- The implementation of the neighbour tagging system combined with the steering behaviours shows the use of multiple techniques to create AI behaviour.

ULO 3: Create realistic movement for agents using steering force models.

- The steering force models for separation, cohesion, and alignment produced a realistic and natural movement simulation.

ULO 5: Combine AI techniques to create more advanced AI.

- The combined behaviour mode shows the ability to combine multiple different forces to create unique and natural behaviour.