Spike: 14
Title: Emergent Group Behaviour

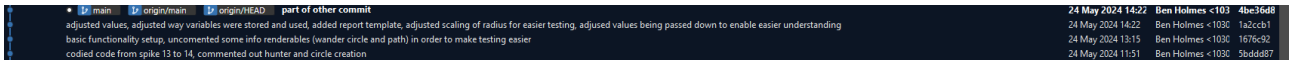
Author: Ben Holmes, 103024841

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 3.12.2
- Pyglet

Tasks undertaken:



- Copied code from spike 13
- Commented out hunter and circle creation code, as well as wander circle render and path render for ease of use
- Added the 4 needed group functions of tag_neighbour, separation, alignment and cohesion

```
def tag_neighbours(self,agents,radius):
    # lets find 'em
    for agent in agents:
        # untag all first
        agent.tagged = False
        # get the vector between us
        to = self.pos - agent.pos
        if to.length() < radius:
            agent.tagged = True
```

```
def separation(self,group):
    steering_force = Vector2D()
    for agent in group:
        # don't include self, only include neighbours (already tagged)
        if agent != self and agent.tagged:
            to_agent = self.pos - agent.pos
            # scale based on inverse distance to neighbour
            steering_force += to_agent.normalise() / to_agent.length()
        return steering_force
```

```
def alignment(self, group):
    avg_heading = Vector2D()
    avg_count = 0
    for agent in group:
        if agent != self and agent.tagged:
            avg_heading += agent.heading
            avg_count += 1

    if avg_count > 0:
        avg_heading /= float(avg_count)
        avg_heading -= self.heading
    return avg_heading
```

```
def cohesion(self,group):
    centre_mass = Vector2D()
    steering_force = Vector2D()
    avg_count = 0
    for agent in group:
        if agent != self and agent.tagged:
            centre_mass += agent.pos
            avg_count += 1
    if avg_count > 0:
        centre_mass /= float(avg_count)
        steering_force = self.seek(centre_mass)
    return steering_force
```

- Added necessary world.py variables for scaling of the vectors to enable weighted sum

```
wander_amount = 1.0
seperation_amount = 1.0
cohesion_amount = 1.0
alignment_amount = 1.0
radius = 10

self.group_variable_mode = 1 # storage variable for group variable change
#stored values in dictionary as to not cause issues, it stores ints, but they are initalised as variables for easy understanding of which variable is which
self.group_variables = { # used in concert with group_variable_mode to reduce number of lines of code
    1: wander_amount,
    2: seperation_amount,
    3: cohesion_amount,
    4: alignment_amount,
    5: radius,
}
```

- Added label functionality for the types of parameters as well as group label enum

```
self.group_info = pyglet.text.Label('', x=5, y=self.cy-20, color=COLOUR_NAMES['WHITE'],batch=window.get_batch("label"))
self.update_label()

def update_label(self):
    self.group_info.text = GroupLabels(self.group_variable_mode).name +': '+ str(self.group_variables[self.group_variable_mode])
```

```
class GroupLabels(Enum):
        Wander_Amount =     1
        Seperation_Amount = 2
        Cohesion_Amount =   3
        Alignment_Amount =  4
        Radius = 5
```

- Added functionality to change parameters on key presses (all but radius increase and decrease by 1, radius by 10, they all cannot go below 0)

```
elif symbol == pyglet.window.key.M:
    self.group_variable_mode += 1
    if self.group_variable_mode > len(self.group_variables):
        self.group_variable_mode = 1
    self.update_label()
elif symbol == pyglet.window.key.N:
    self.group_variable_mode -= 1
    if self.group_variable_mode < 1:
        self.group_variable_mode = len(self.group_variables)
    self.update_label()
elif symbol == pyglet.window.key.RIGHT:
    if self.group_variable_mode == 5:
        self.group_variables[self.group_variable_mode] += 10
    else:
        self.group_variables[self.group_variable_mode] += 1.0
    self.update_label()
elif symbol == pyglet.window.key.LEFT:
    if self.group_variables[self.group_variable_mode] > 0.0:
        if self.group_variable_mode == 5:
            self.group_variables[self.group_variable_mode] -= 10
        else:
            self.group_variables[self.group_variable_mode] -= 1.0
    self.update_label()
```

- Added overall group behaviour calculation

```
def group_movement(self, delta, radius, wander_amount, seperation_amount,alignment_amount,cohesion_amount):
    steering_force = self.wander(delta) * wander_amount
    self.tag_neighbours(self.world.agents, radius)
    seperation = self.separation(self.world.agents)
    steering_force += seperation * seperation_amount
    alignment = self.alignment(self.world.agents)
    steering_force += alignment * alignment_amount
    cohesion = self.cohesion(self.world.agents)
    steering_force += cohesion * cohesion_amount
    return steering_force
```

Calculate function:

```
elif mode == 'group':
    force = self.group_movement(delta, radius, wander_amount, seperation_amount,alignment_amount,cohesion_amount)
```

Update function:

```
def update(self, delta, wander_amount, seperation_amount,alignment_amount,cohesion_amount, radius):
    ''' update vehicle position and orientation '''
    # calculate and set self.force to be applied
    ## force = self.calculate()
    force = self.calculate(delta, radius, wander_amount, seperation_amount,alignment_amount,cohesion_amount)  # <-- delta needed for wander
```

World.py update function:

```
def update(self, delta):
    if not self.paused:
        #self.hunter.update(delta)
        for agent in self.agents:
            agent.update(delta, self.group_variables[1], self.group_variables[2], self.group_variables[3], self.group_variables[4],self.group_variables[5])
```

- Then tested and adjusted any mistakes

What we found out:

The basic functionality was relatively simple to implement however I had some issues around testing and various other parameter implementation, mainly to do with creating the dictionary and not understanding exactly what was being stored there.

The testing was seemingly rotating around the same point constantly, which I figured out was first the parameters not adjusting as a result of the dictionary mistake and second, the radius increments were too large for the scale.

Testing buttons:

M switches parameter (or group variable modes) increasing through the list, with N doing the inverse

Left decreases and Right increases the parameter value