SPIKE REPORT LAB 06.2 (T7)

Jake Lucic

2103435 AI for Games

Number: Lab 06 Spike 2
Spike Title: Emergent Group Behaviour
Personal: Jake Lucic (2103435)

Goals:
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 weighted-sum to combine all steering behaviours
- Support the adjustment of parameters for each steering force while running

Technologies, Tools, and Resources used:

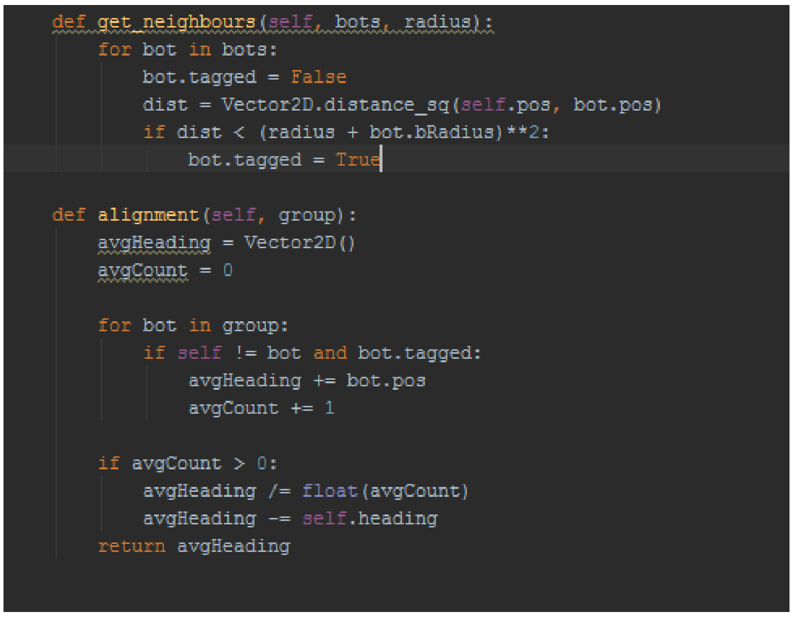
In this task, the technologies used are listed below:

- Lab 06 base code from lab task 06.1
- Python IDLE v3.6.4 / Python language or PyCharm IDE
- AI for Games Lecture material

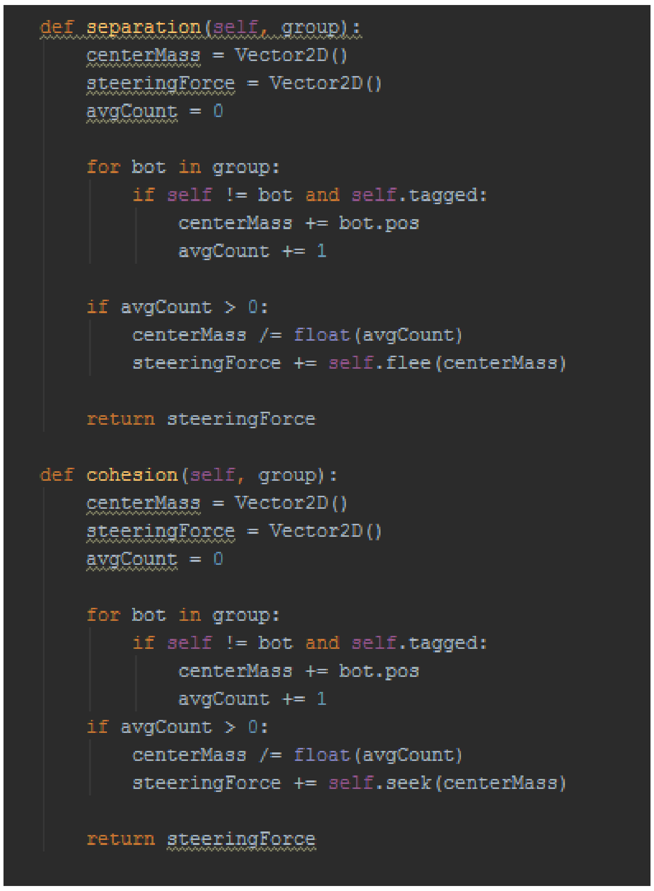
Tasks undertaken:

- Read the code base and understand what each function does
- Study the different key words and variable names
- Read the Lecture notes and watch lecture, gathering knowledge on cohesion, separation and alignment steering behaviours
- Understand key press behaviours and modifiers in pyglet, for easy access to changing parameters
- Add in 'neighbourhood mode' for bot on button click, with changeable radius
- Add cohesion, alignment, separation modes that act on a changeable value, that you can adjust in game.
- Create a weighted sum function that if in neighbourhood mode, all 3 steering behaviours calculate and operate
- Draw the values of each parameter to screen

CODE SNIPPETS



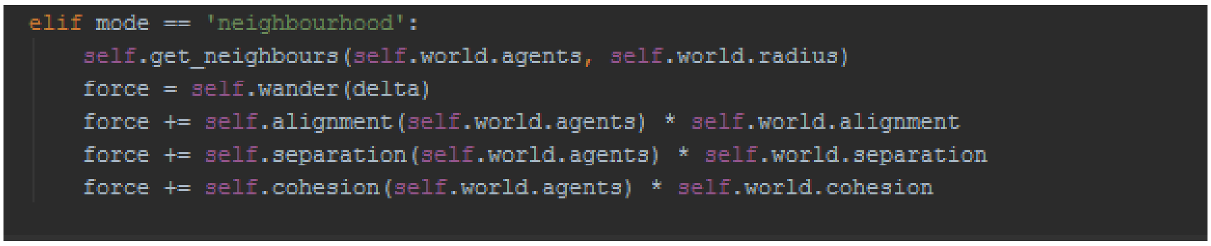
```
def get_neighbours(self, bots, radius):
    for bot in bots:
        bot.tagged = False
        dist = Vector2D.distance_sq(self.pos, bot.pos)
        if dist < (radius + bot.bRadius)**2:
            bot.tagged = True

def alignment(self, group):
    avgHeading = Vector2D()
    avgCount = 0

    for bot in group:
        if self != bot and bot.tagged:
            avgHeading += bot.pos
            avgCount += 1

    if avgCount > 0:
        avgHeading /= float(avgCount)
        avgHeading -= self.heading
    return avgHeading
```



```
def separation(self, group):
    centerMass = Vector2D()
    steeringForce = Vector2D()
    avgCount = 0

    for bot in group:
        if self != bot and self.tagged:
            centerMass += bot.pos
            avgCount += 1

    if avgCount > 0:
        centerMass /= float(avgCount)
        steeringForce += self.flee(centerMass)

    return steeringForce

def cohesion(self, group):
    centerMass = Vector2D()
    steeringForce = Vector2D()
    avgCount = 0

    for bot in group:
        if self != bot and self.tagged:
            centerMass += bot.pos
            avgCount += 1
    if avgCount > 0:
        centerMass /= float(avgCount)
        steeringForce += self.seek(centerMass)

    return steeringForce
```

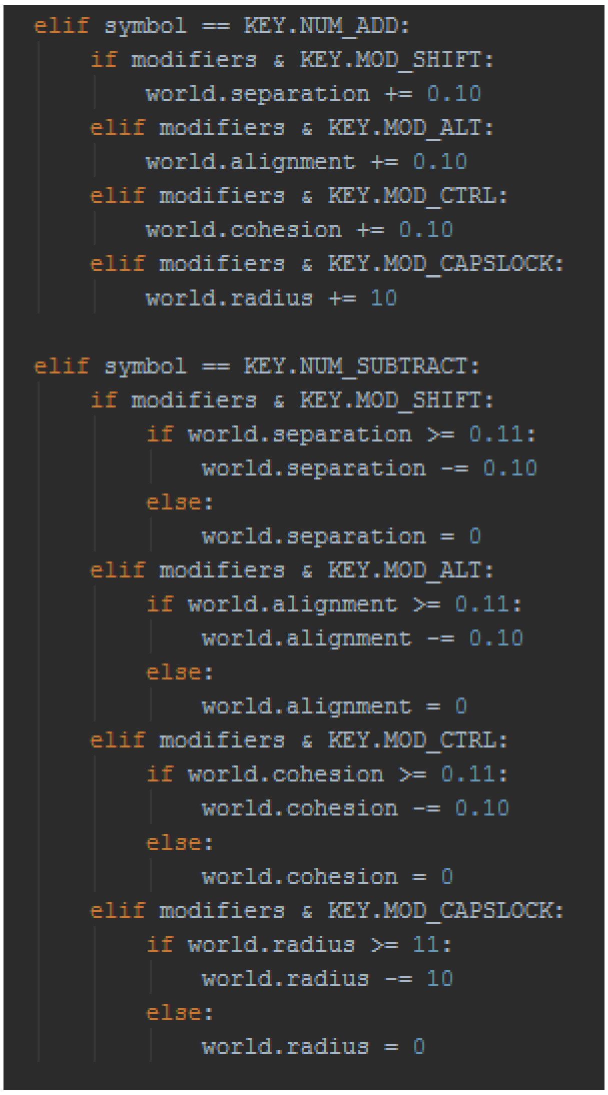
These are the bot functions for each of the steering behaviours, including the get_neighbours function that checks if the bot has any close by friends to steer with.

This is the weighted sum that adds the force of all 3 steering behaviours



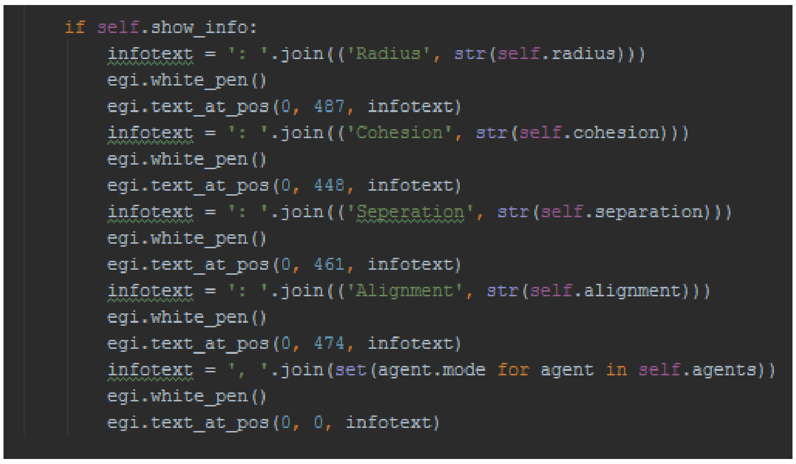
```
elif mode == 'neighbourhood':
    self.get_neighbours(self.world.agents, self.world.radius)
    force = self.wander(delta)
    force += self.alignment(self.world.agents) * self.world.alignment
    force += self.separation(self.world.agents) * self.world.separation
    force += self.cohesion(self.world.agents) * self.world.cohesion
```

CONTROLS



```python
elif symbol == KEY.NUM_ADD:
    if modifiers & KEY.MOD_SHIFT:
        world.separation += 0.10
    elif modifiers & KEY.MOD_ALT:
        world.alignment += 0.10
    elif modifiers & KEY.MOD_CTRL:
        world.cohesion += 0.10
    elif modifiers & KEY.MOD_CAPSLOCK:
        world.radius += 10

elif symbol == KEY.NUM_SUBTRACT:
    if modifiers & KEY.MOD_SHIFT:
        if world.separation >= 0.11:
            world.separation -= 0.10
        else:
            world.separation = 0
    elif modifiers & KEY.MOD_ALT:
        if world.alignment >= 0.11:
            world.alignment -= 0.10
        else:
            world.alignment = 0
    elif modifiers & KEY.MOD_CTRL:
        if world.cohesion >= 0.11:
            world.cohesion -= 0.10
        else:
            world.cohesion = 0
    elif modifiers & KEY.MOD_CAPSLOCK:
        if world.radius >= 11:
            world.radius -= 10
        else:
            world.radius = 0
```

A big help to making the controls function easier was to read about pyglets keyboard clicks and modifiers.



```python
if self.show_info:
    infotext = ': '.join(('Radius', str(self.radius)))
    egi.white_pen()
    egi.text_at_pos(0, 487, infotext)
    infotext = ': '.join(('Cohesion', str(self.cohesion)))
    egi.white_pen()
    egi.text_at_pos(0, 448, infotext)
    infotext = ': '.join(('Seperation', str(self.separation)))
    egi.white_pen()
    egi.text_at_pos(0, 461, infotext)
    infotext = ': '.join(('Alignment', str(self.alignment)))
    egi.white_pen()
    egi.text_at_pos(0, 474, infotext)
    infotext = ', '.join(set(agent.mode for agent in self.agents))
    egi.white_pen()
    egi.text_at_pos(0, 0, infotext)
```

Looking at how the infotext agent.mode was written, I was able to figure out how to draw the parameters on screen.