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
- *Spike outcome report and working code (with key instructions).*

Technologies, Tools, and Resources used:

- Python IDE(PyCharm) with python 3 installed
- Piglet Documentation http://pyglet.readthedocs.io/en/pyglet-1-3-maintenance/
- Python 3 Documentation http://docs.python.org/
- The code from previous lab task
- Help from peers.

Tasks done:

(Agent.py)

1. Added neighbourhood mode

```
AGENT_MODES = {
    KEY._1: 'seek',
    KEY._2: 'arrive_slow',
    KEY._3: 'arrive_normal',
    KEY._4: 'arrive_fast',
    KEY._5: 'flee',
    KEY._6: 'pursuit',
    KEY._7: 'follow_path',
    KEY._8: 'wander',
    KEY._9: 'neigbourhood',
}
```

2. To indicate we are not a part of neighbourhood

```
self.tagged = False
```

3. Setting the required force around the neighbours in calculate method for the separation, cohesion, and alignment for which we have to calculate the neighbour distance within proximity.

```
115         elif mode == 'neigbourhood':
116             self.find_neighbours(self.world.agents, self.world.radius)
117             force = self.wander(delta)
118             force += self.seperation(self.world.agents) * self.world.seperation
119             force += self.cohesion(self.world.agents) * self.world.cohesion
120             force += self.alignment(self.world.agents) * self.world.alignment
```

```
260     def find_neighbours(self, bots, radius):
261         for bot in bots:
262             bot.tagged = False
263             # get distance between self.pos and bot.pos
264             dis = Vector2D.distance_sq(self.pos, bot.pos)
265             if dis < (radius + bot.bRadius)** 2:
266                 bot.tagged = True
```

4. Separation

```
301     def seperation(self, group):
302         """
303         This returns a steering force away from the center of the group
304         """
305         centerMass = Vector2D()
306         steeringForce = Vector2D()
307         avgCount = 0
308 
309         for agent in group:
310             if self != agent and self.tagged:
311                 centerMass += agent.pos
312                 avgCount += 1
313 
314         if avgCount > 0:
315             centerMass /= float(avgCount)
316             steeringForce += self.flee(centerMass)
317 
318         return steeringForce
319 
```

5. Cohesion

```
282     def cohesion(self, group):
283         """
284         This returns a steering force towards the center of the group
285         """
286         centerMass = Vector2D()
287         steeringForce = Vector2D()
288         avgCount = 0
289 
290         for agent in group:
291             if self != agent and self.tagged:
292                 centerMass += agent.pos
293                 avgCount += 1
294 
295         if avgCount > 0:
296             centerMass /= float(avgCount)
297             steeringForce += self.seek(centerMass)
298 
299         return steeringForce
```

6. Alignment

```
268     def alignment(self, group):
269         avgHeading = Vector2D()
270         avgCount = 0
271 
272         for agent in group:
273             if self != agent and agent.tagged:
274                 avgHeading += agent.pos
275                 avgCount += 1
276 
277         if avgCount > 0:
278             avgHeading /= float(avgCount)
279             avgHeading -= self.heading
280         return avgHeading
281 
```

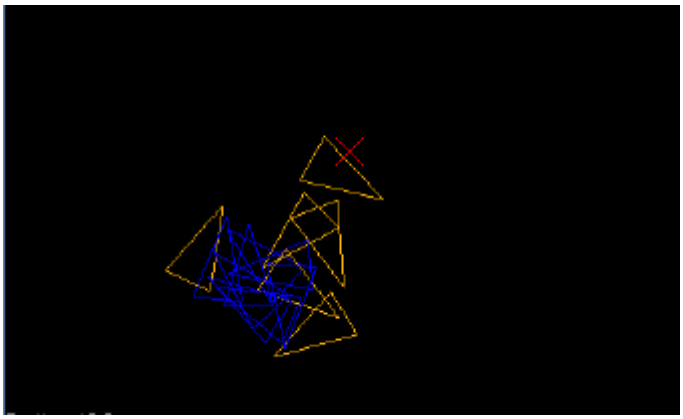
The controls to change the weighting of each of the component parts of the weighted sum and within how large of a radius each agent will consider other to be their neighbours. I used the +, - keys to increase or decrease the modifiers and I used Z, X, Q key t to change which modifiers were being modified.

(World.py)
//Added variables in world.py for the below variables.

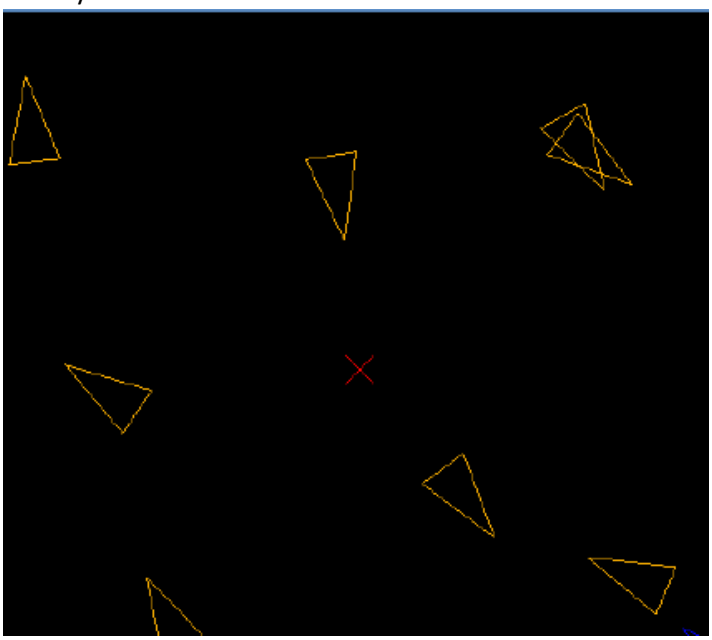
```
# Cohesion/Sepertion/Alignment Variables
self.cohesion = 0.0
self.seperation = 0.0
self.alignment = 0.0
self.radius = 200.0
```

Output we found out:

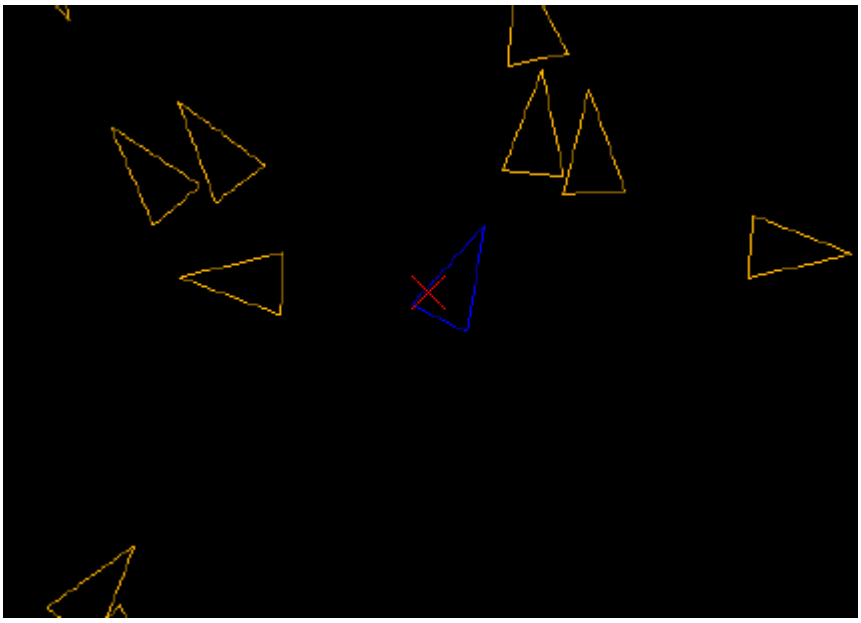
1. Low Alignment and separation but with High cohesion causes the agents to gather and wander about.



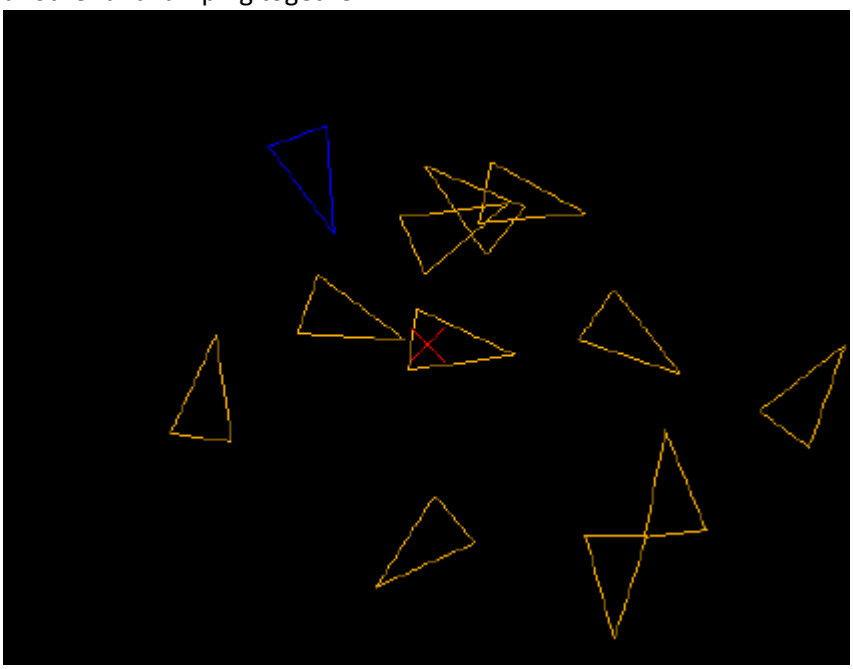
2. Low Alignment or Cohesion with High Separation causes agents to flee towards corners as they wander.



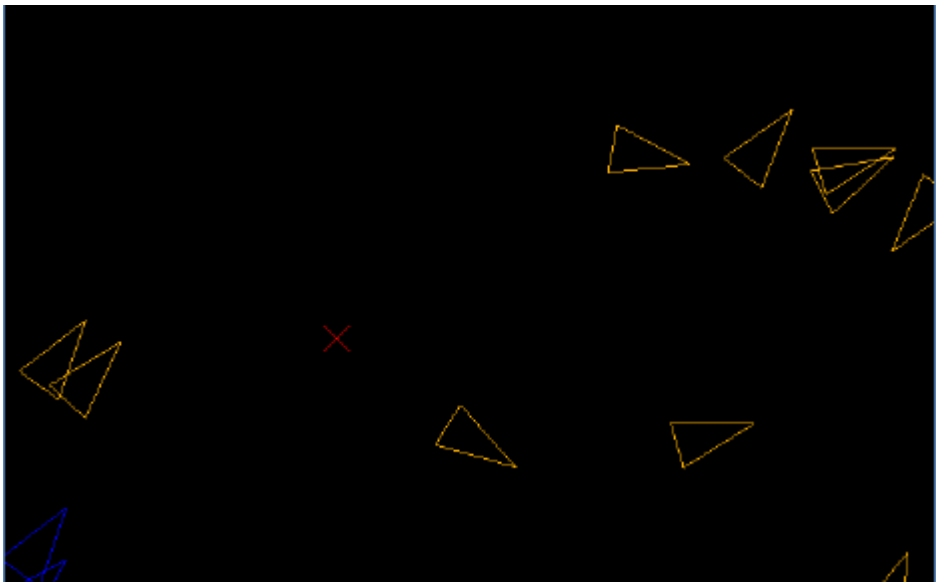
3. Low separation and Cohesion with high alignment causes nearby agents to move in same direction at first but then move away from one another.



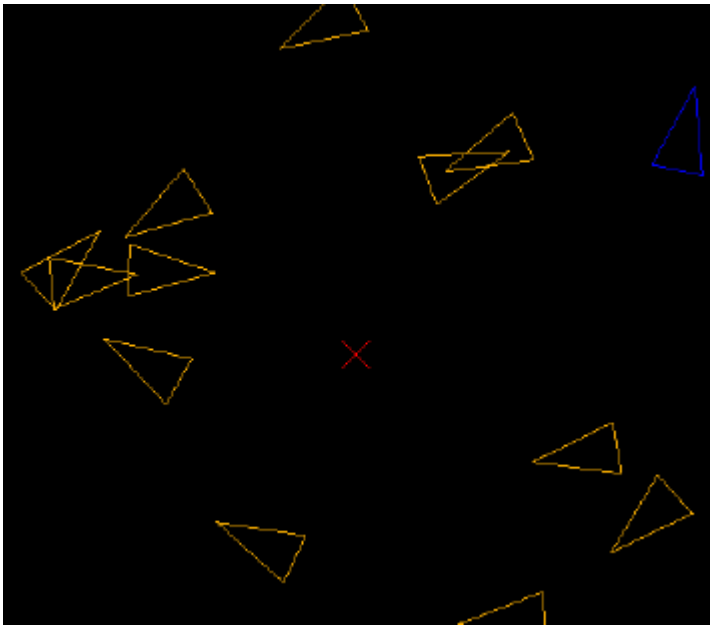
4. High Separation and Cohesion with low Alignment causes agents to move around in circles when within proximity of one another while trying to move away from one another and lumping together.



5. High Alignment and Cohesion with low seperation causes agents to move as a single group in one direction until they over wrap the screen.



6. Unlike just High Separation, High Separation and Alignment with low cohesion causes agents to flee from one another at the edges of the window trying to avoid the centre of the window.



The finding in this task were quite observant and helpful, as the custom project I have in my mind would also utilize the same feature and implement the behaviour where the player can hide from the enemy using the hide objects on the scene, the enemy could utilize the neighbourhood approach to make the AI enemy more smarter and be creative than the player. Plus the logic behind Separation where the group bound by the neighbourhood can separate/steer away from its member can be a useful implementation while patrolling for target as they can cover more space on the scene, while with Alignment the monsters can be set free in a herd or this logic can be utilize when two single handed sword fighters face off with each other within a close combat scene, with Cohesion the thing is this sort of approach where the group members stick or simply seek the centre of its local neighbour positions can be used when in the game the minions try to defend the boss by

forming a protocol circle around the boss monster. With all these combined the movements of AI player/enemy can be clustered in a different way that makes the target to not be able to adapt to AI movements as the patterns can be a mixture of combined Properties of these logics as described above. What I personally felt while exploring these findings was that they have the potential to make a simple strategic bot to choose clever functioning and seek out the target and with all these movements combined it can create one interesting and complex AI movement pattern that can make the target have fun while adapting to counter it.

Notes:

- ✓ Controls for the E.G.B (Emergent Group Behaviour)
  - \+ increases cohesion
  - – decreases cohesion
  - Z + increases radius
  - Z – decreases radius
  - X + increases Separation
  - X – decreases Separation
  - Q + increases Alignment
  - Q – decreases Alignment