Joel Aldridge (n9472827)

IGB 383

Assignment 2

Statement of completeness:

The following tasks were completed

Scouts set up to look for resource rich areas then if they find an area report back to the mothership.

Foragers set up to move to a resource area, harvest an amount of resource.

Elite Foragers forage while also checking for new resources. If a new resource is found and is better than the current foraging area, then the elite forager becomes a scout.

Attacking and fleeing occurs via a utility function.

All drones are selected using heuristics, the selection of which resource site to forage is done through systematic approach of sending the best foragers to the highest valued patches. A utility function is applied for the drone's choice between attacking or fleeing that uses hunger and a version of hunting variables.

Harvesting resources occurs as required.

Endgame situation occurs when the mothership has collected enough resources through foragers.

The following tasks were not completed

The search zone shrinking after forager fails to find resource. This could be implemented by parameters within the mothership code that once activated after receiving a zero-cargo value from a drone enable restrictions which scout drones use for their random searches – effectively keeping them within a new bounded domain.

The mothership decision to abandon a search site after several failed attempts was not correctly implemented as per the instructions. Instead when a bee harvests an asteroid and reduces its value to a negative it decides for the mothership to abandon the asteroid as a search zone. The desired result could be achieved by using the mothership game object data instead of setting a target to the mothership then have the mothership keep track of failed forages for each asteroid and if they passed a certain threshold abandon the asteroid.

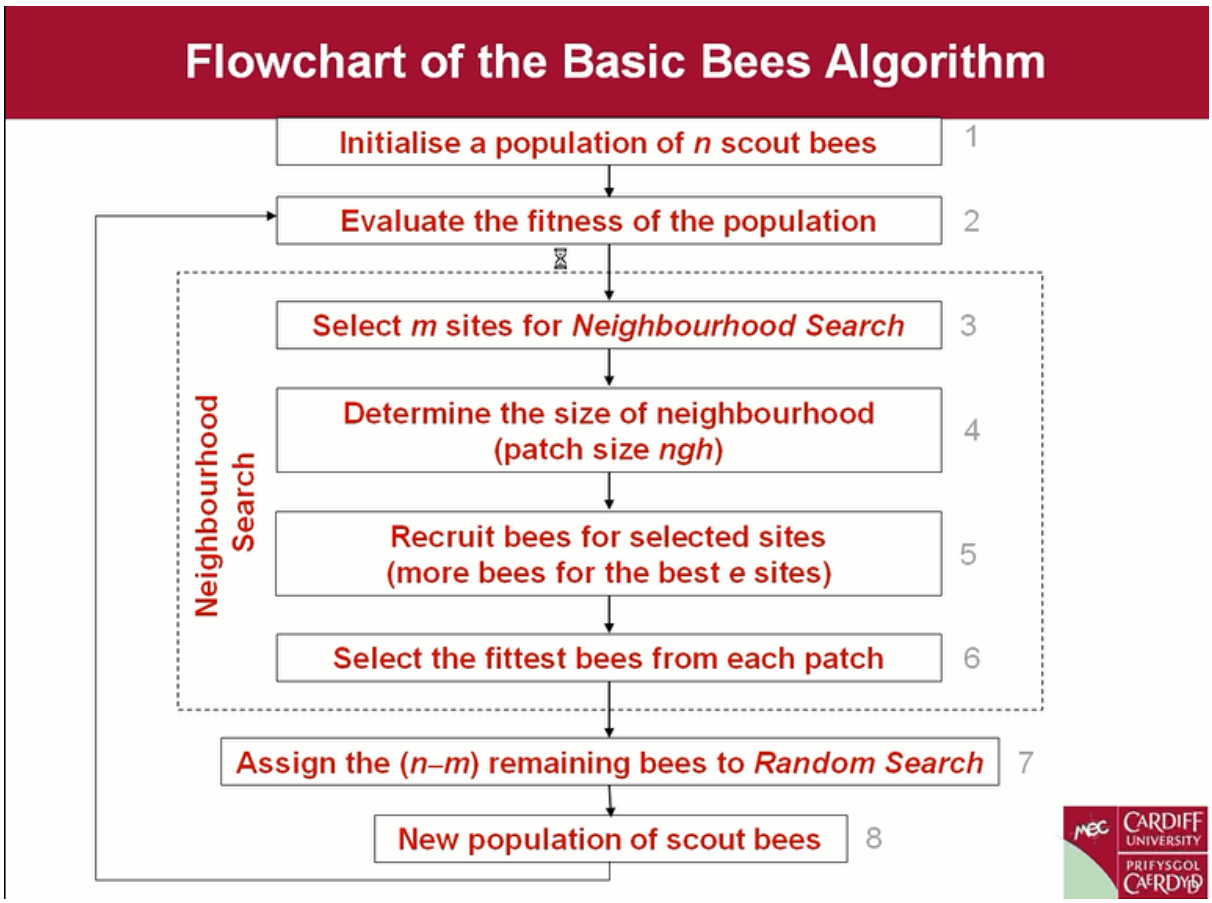
Fear was not implemented in the project. This could be implemented via a variable keeping track of other drones being killed and their proximity to the current drone to then influence the decision of attacking or fleeing.

Hunting was semi implemented. The value for hunting was combined with the hunger value in the implementation such that when attacking the hunger value lowers.

Bee's Algorithm:

This project applies a simplified version of the bee's algorithm. The central mothership assigns a limited number of drones (based on a heuristic considering aspects that will positively influence the drone's behaviour such as speed) to be scouts. Scouts move in a random direction and periodically check if they are within range of an asteroid that will provide a resource. When they find such an asteroid they return to the mothership which sends out an elite or regular forager to harvest this asteroid. The regular forager simply moves to this asteroid, takes a portion of its resources and returns it to the mothership. Elite foragers act as both foragers and scouts, moving to a designated resource while also checking for more profitable resources on the way. The flow chart below details how the

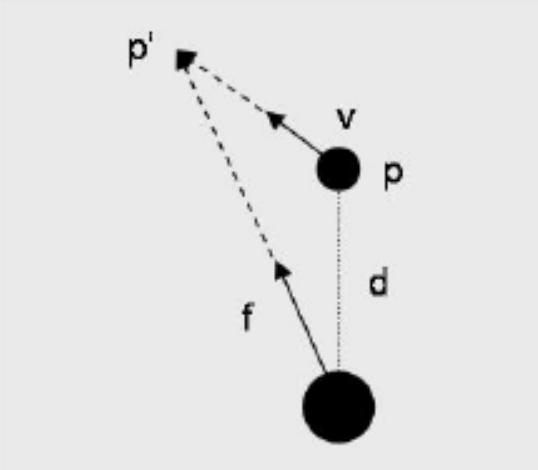
bee's algorithm was implemented with scouts providing locations of areas for foragers to visit.



The implementation of the bee's algorithm ran into only 1 small setback that was solved with the use of lists and modulo operator. This problem was the assignment of which asteroid to send foragers too. As there was no way of telling if a forager has been sent to any given patch the mothership used the sorted list of resource patches and the number of foragers and elite foragers sent out. An Elite forager is sent out to the highest patch – with the next elite forager knowing that there is one elite forager ahead of them and hence going to the second patch. The same method was applied to regular foragers who checked how many elite and normal foragers were already in the field and then chose the first free patch assuming each other drone was at one of the highest scoring patches. A modulo arithmetic method was applied on top of this so if there were less resource patches than elite and regular foragers any drone sent foraging would loop back to the highest scoring patch with the smallest number of drones already foraging there.

Predator Prey Models:

This project uses a simplified version of the predator prey model. Drones use a utility function to decide whether it is more important to attack the player or to move away and regroup or heal. When attacking the drone calculates the expected position of the player and moves to intercept. This is done using the players current position (p on the diagram below), the velocity (v) of the player and the distance (d) between the drone and the player to find the projected location of the player (p') and move in its direction (f).



The fleeing model was applied in the same formula, finding the velocity and distance and using it to calculate the projected location of the player then moving away from this location.

The utility function controlling when to attack and when to flee uses a value, hunger, to be more in the line with a real-world predator prey scenario. This value hunger increases when the drone is fleeing and decreases when it is attacking. As this number changes increases the likelihood of the drone changing state. This provides a more real-world approach to the predator prey scenario as external factors such as hunger influence the likelihood a predator or prey to take on the other state role.

The current model has some problems with transitioning between attacking or fleeing via the utility function. The function value fluctuates rapidly each update, and this can lead to drones switching between attacking and fleeing frequently without any input from the player. This could be fixed by limiting the amount the utility function can change in a given amount of time – forcing the drones to remain in attacking or fleeing for a set time to stop the rapid flipping between attacking or fleeing.

The aliens currently attack via a simple copy of the players shooting mechanics with damage removed in case of friendly fire. This negates the effect of drones in the back of the squadron shooting their squad mates but does not stop this behaviour – to effectively fix this a ray casting system could be implemented where the drones check to see if their line of sight to the player is clear and only then fires.

Drones currently can attempt to pass through the mothership if they are in a state requiring them to be on the other side. These drones will slowly slide up till they are over the mothership. This Could be fixed by adding an input check before assigning a direction to see if facing an object and if so move target position to avoid this behaviour.

Conclusion:

The use of the bee's algorithm in a real time game scenario provides an effective way to mirror real world hive environments and can be extended to simulate behaviour for a variety of large NPC groups. Using a predator prey model is very useful in a variety of video game genres as it creates a dynamic element to the AI which if extended could provide memorable game experiences. When combined these two algorithms produce a believable effect of simulating a large crowd of NPCs.