Titel: Simulation of an ant colony using steering behaviors, decision making and influence maps.

# Abstract:

Machine learning an AI allow us to program decision making for non player character (NPCs) in video games. It can also be used to simulate the behaviour of animals from the real world. In the following research project I will attempt to simulate the workings of an ant colony, more specifically a fire ant colony, using a combination of steering behaviors, decision making and influence maps.

# Introduction:

Did you know that you can find ants nearly everywhere on earth. The only places where you can not find them is Antarctica and a few inhospitable islands. This clearly indicates that ants are great at spreading to new places and surviving there. This is of course horrible for the local ecosystem, wich gets disrupted by the ants. But that issue is outside the scope of this project. The point is that for a group of tiny insects they work together quit well. They have different areas in their nest each with their own purpose: (own knowledge) A chamber where the queen resides, multiple brood chambers order by the age of the brood there and garbage sites where left over food and dead ends are dropped. This last one is interesting because it shows that ants have a basic concept of hygiene. They create these garbage sites so that the smell does not attract predators and to prevent the growth of mould in the nest itself. Ants also have the ability to store food. To achieve this ants make use of there social stomachs to store food. They then use a process called trophallaxis to regurgitate food from one ant to another. They achieve this and more all while being nearly blind. Instead of relying on sight and sound to convey meaning to one another they instead use specialised pheromones. To indicate where an ant found food they lay down a pheromone trail. They also use this trail to find their way back home. When an ant dies they start releasing a pheromone that alerts nearby ants of the dead ant. They then collect the body and bring it to the garbage site. There are around 10 to 20 pheromones that ants use, each with their own meaning. It is clear that ants are more complex then they look like. And this is only a general description. There are hundreds of ant species each with their own unique quirks. Leaf cutter ants that make there nests in trees. Dracula ants that survive by drinking a small amount of blood from their own brood. For this project I will limit myself to the following behaviours.

(changes to this are still posible)

Queen: the queen ant is a stationary ant that produces new ants every X seconds. She can not get food herself and thus will need to be fed by worker ants.

Worker ants: These ants are the mot common ants and do most of the work. They go out and forage for food. They collect food in their social stomach. They bring food back to the colony and feed the queen and other ants. They have a limited life span and will die. They need to keep the colony clean by moving dead ants to the designated garbage area.

Soldier ants: Larger then worker ants these ants are tasked with protecting the colony. The react to external threats to the colony and fight off intruders.

To achieve these behaviors the following components will be needed.

Steering behaviors: A way to control how the ants move.

Decision making: A way to device what the ant does depending on its current situation.

Influence map: A substitute for the pheromones. This allows the ant to map out there location without the need for pathfinding.

# Steering behaviors:

Steering behaviours are an amalgamation of different behaviours used to manage the movement of an AI. These are often verry simple behaviours using external factors such as goal, other agents and obstacles to move the agent. A few examples:

* Seek: move to a specific location.
* Flee: move away from a specific location.
* Pursuit: predict the location where another moving agent will be in the near future and move to that location.
* Evade: opposite of pursuit. Move away from the predicted location of another moving agent.
* Wander: choose a random location in front of the agent and move towards it.

On their own these behaviours are verry limited, but combining them opens up a whole new world of possibilities. We will limit us to the following 2 ways of combining steering behaviours: priority steering and blended steering.

## Priority steering:

Priority steering uses extra rules and checks to see wich steering behaviour to use. For example if you want an agent that wanders around randomly but also evades obstacles you create a priority steering behavior using wander and evade. By default the agent will use the wander behaviour to move around but the moment the agent detects an obstacle it will switch to the evade behaviour to move away from it.

## Blended steering:

Blended steering gives each steering behaviour a certain weight. The higher the weight of the behaviour the more influence it will have on the movement of the agent. For example if you combine the wander behaviour and the seek behaviour each with a weight of 50% you will get drunk seek: an agents that tries to reach a specific point while never following a straight path.

# Decision making:

An agent should have the ability to chose what to do in certain cituations. This is mostly manually programmed. But doing this for every situation is hard and long work. It also results in code that contains gaint if else statements. To avoid this several ways of decision making have been made to simplify and clean up this process. We will limit ourselves to Finite State Machine and Behavior trees.

## Finite State Machine:

A finite state machine is a directed graph with each node representing a state in wich the agent can be and every link a transition to another state. An agent can only be in 1 state at a time and can only go to another state to wich it has a link. An example is the finite state machine used for deciding what animation to play at what time. This is of course very simplified but it suffices for the current project.

## Behaviour tree:

A behaviour tree is an directed acyclic graph, a directed graph with no cycles. The tree always starts at the same point, the root, and then makes it way down the tree. De pending on the result of the nodes the agent will perform certain actions or the program will return back down the tree. The node that return a result are called action nodes and return 1 of the following 3 actions: succeeded, failed or running. This result is send to the composite node. This node decides what to do depending on the result of the action node. We will limit ourselves to the following 2 composite nodes: the sequence node and the selector node.

### The sequence node:

The sequence node attempts to execute each action node in the giving order. Depending on the result of the action nodes the sequence executes a specific action.

* Action node returns success: the sequence node start axecuting the next action node.
* Action node returns running: the sequence node aborts the execution and resturns running.
* Action node returns failed: the sequence node aborts the execution and returns failed.

You can compare this node the an AND operation.

### The selector node:

The selector node attempts to execute each action node until a certain result is received.

* Action node returns success: the selector node stops and returns success.
* Action node returns running: the selector node stops and returns running.
* Action node returns failed: the selector node starts executing the next child.

Just like the sequence node you can compare this node to the OR operation.

Action nodes are used to make the agent do something an can not have child nodes. This can be very limiting especially if in a sequence you want to check if a certain condition is met before executing the rest of the sequence. To solve this we use Conditional nodes. Condition nodes are nodes used to check a condition. They return either true of false.

Combining this with the movement behavior from before will handle most of the decision making and movement.

# Influence maps

Influence maps are the collection of the influence that agents have on the world. It is a spatial storage of influence that agents can use to make decisions. For example if a lot of enemy agents have walked through the same area their influence on that area will be big and an agent may know not to go there. Influence maps also can be used to store information from the past, predict agents future position and give agents imperfect information. Influence also spreads over the map. This way this influence spreads over the map is called influence propagation. We will limit ourself to 2 types of influence propagation: convolution filters and map flooding.

[more research needed]

Convolution filters:

Convolution filter is a bluring technique, each tile of the map only gets processed once per blur iteration.

Map flooding:

Map flooding is the slow spread of influence over the map. Slowing down and decreasing in power the further away it goes from the source.

Combining these two will result in a influence propagation that will: Take the highest influence value of all neighbors and interpolate with the current influence. To controle this 2 parameters are used: Decay and momentum.

Momentum: Indicates how difficult it is to modify the value of something.

Decay: how much incluence decreases over distance.