Crowd simulation of an ant colony.

# Abstract:

Machine learning and AI enable us to program decision-making for non-player characters (NPCs) in video games. They can also be used to simulate the behavior of real-world animals. In the following research project, I will attempt to simulate the workings of a fire ant colony using crowd simulation. I will use a combination of steering behaviors, decision-making, and influence maps to create the simulation model.

# Introduction:

Did you know that ants can be found nearly everywhere on earth, except 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, which gets disrupted by the ants. But that issue is outside the scope of this project.

The point is that ants display remarkable teamwork. Their nests consist of multiple chambers, each serving their own purpose: A chamber where the queen resides, brood chambers order by the age of the brood, and garbage sites for disposing of leftover food and dead ants. 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 their 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 their 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)

1. **Queen**: the queen ant is a stationary ant that produces new ants every X seconds. She cannot get food herself and thus will need to be fed by worker ants.
2. **Worker ants**: These ants are the most 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.
3. **Soldier ants**: Larger then worker ants these ants are tasked with protecting the colony. They react to external threats to the colony and fight off intruders.

To achieve these behaviors the following components will be used.

* **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 their surroundings without the need for pathfinding.

The project will make use of a framework provided by the teachers of gameplay programming at DAE(ask if you are allowed to mention their names). This framework already contains working code for the previous components. I will use these as a base to make the project.

# Components in more detail

## Steering behaviors:

Steering behaviours are an amalgamation of different behaviours used to manage the movement of an AI. These are often very 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 very 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 which 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 choose what to do in certain situations. This is mostly manually programmed. But doing this for every situation is hard and long work. It also results in code that contains giant 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 which 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 which 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. This helps a group of agents to make decisions more strategically. Different information needs to be stored in separate influence maps. Influence maps also can be used to store information from the past, predict agents future position, and give agents imperfect information. Influence spreads over the map, we call this influence propagation. There are multiple ways to propagate but we will limit ourself to the following 2: convolution filters and map flooding.

[more research needed]

### Convolution filters:

Convolution filter is a blurring technique. Using a small matrix of numbers, also known as a kernel, on each pixel of an image will result in a new image where each pixel is a weighted sum of the pixels in the original image. Using this on an influence map means that each tile on the map will hold the weighted influence of the tiles in the previous iteration.

The advantage of this is that each tile gets processed only once per iteration.

The disadvantage of this is that it can not combine influences into a new higher influence. It also does not work well when there are only a small number of influence sources.

### Map flooding:

Map flooding uses the flood fill algorithm to spread influence across the map. The flood fill algorithm is an algorithm that determines and alters the area connected to a given node. An example of this is the bucket tool in mspaint.

The advantage of this is that it is fast and can use terrain analysis to avoid obstacles.

The disadvantage of this is that it is not good at combining influences because it picks the highest one.

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

Momentum: Indicates how difficult it is to modify the value of something. For example a car whith a high momentum will not stop eaily. Higher momentum will result in slower propagation.

Decay: how much incluence decreases over distance. Lower decay will result in firther spread of the influence.

# Creating the project

I quickly realised that this was going to be a lot more complicated. By simulating an ant colony we are basically trying to create a crowd simulation.

<https://en.wikipedia.org/wiki/Crowd_simulation>

A crowd simulation is the process of simulation the movement and/or dynamics of a large number of entities. There are several approaches to crowd simulation, each with there own advantages and disadvantages based on the following two factors: crowd size and time scale.

. Crowd size: this refers to the number of entities in the simulation. There is no exact number to specify if a crowd is large or small.

. Time scale: this refers to how the objective of the simulation effects the length of the simulation. For example the simulation of how ideologies spread amongst a population will run for a long time as these events can take months or even years.

These 2 factors are used to classify crowd simulations.

* Flow-based Approach: a simulation focused on a crowd as a whole. Individuals have no distinct behavior that occurs due to input from their surroundings. Mostly used to simulate the flow of movement. Crowd size: large, Time Scale: short.
* Entity-based Approach: a simulation that implements a set of physical, predefined, and global laws. The entities do not think for themselves and their movement and actions are determined by the global laws. This model is most often used to research crowd dynamics such as jamming and flocking. Crowd Size: Small to medium, Time Scale: short.
* Agent-based Approach: Each agent in this simulation has a degree of intelligence. They can each react to each situation on their own based on a set of decision rules. They also use information from there surrounding to decide how to react. This is most often used to simulate realistic crowds. Crowd size and time scale depend entirely on what the researcher is trying to simulate.

This project leans more towards the agent-based approach: each ant makes different dicisions based on its surroundings. But there is also a hint of entity-based approach: all ants make decisions based on the predefined global laws. There is no exact guide on how to create the simulation model so I will attempt to make my own. For that I will implement the following steps.

1. Create a base class for all the ants.
2. Create a class for each ants derived from the base class.
3. Create a behavior tree that will be used on all the ants in the simulation.
4. In the tree create a branch for each type of ant.
5. Execute the tree for each ant in the simulation.

## Creating the ants

To create the different ants a base class will be used where the other ants will be derived from.