MAS Project: Self Assembling Swarm

Niels Verleysen
Student number: r0789740
MSc Artificial Intelligence (ECS), KU Leuven
niels.verleysen@student.kuleuven.be

May 2020

1 Introduction

Over the course of history researchers and engineers have found inspiration in nature when facing a hard technological problem, for example the wings of a plane that are inspired by birds, injection needles that are inspired by mosquitoes or neural networks that are inspired by our brains. Swarm intelligence is also one of these technological solutions. A swarm consists of many relatively simple agents that can communicate with eachother. All these agents have a set of behavioral rules and by following these the swarm can solve complex problems. In nature these can be seen as insect colonies, schools of fish and flocks of birds. Insect colonies can show very complex behavior, while each insect on its own is not very smart. There is also no need for central coordination or management in these swarm systems. Swarm systems are very interesting as this means that a group of relatively cheap robots can solve complex tasks without the need of a complex overhead management system. One thing that can be hard is designing the behavior of a single agent such that the desired problem solving behavior of the swarm emerges.

In this project I have researched the self assembling swarm developed by the Wyss Institute at Harvard.[3] This swarm system can form any singly-connected 2D shape that is provided to it by moving the robots inside the shape. This system is initialized by grouping all the agents together and adding a group of four seed agents outside the shape. These seed robots don't do anything but initializing the gradient and coordinate system centre. The other agents in this system are supplied with the shape that needs to be formed and only do five things. They form a gradient which increases when going further from the centre defined by the seeds. They all make sure to have a locally unique identifier. They can locate themselves inside the coordinate system based on their neighbors. An agent can move around the group of agents by following its edge in a clockwise manner. And finally the agents can communicate with other agents in their neighborhood. By combining these abilities the agents can

move around the other agents, find out if they are inside the shape or not and if they are inside the shape determine if they should join the shape or move further.

Research question: What will the system do when the target shape is not a single shape? What if the target exists out of multiple shapes that are not connected?

Hypothesis: The system will behave as normal, but will only take into account the shapes that are directly connected to the swarm itself.

Follow-up research question: Can the idea of bridge-formation like is used by ants, be employed by this system to form multiple shapes that are not connected?

To validate or dismiss my hypothesis of the first research question I have reproduced this swarm system in the Multi-Agent Simulation Toolkit MASON [2] which can be used as a library in Java [1]. I have then simulated the swarm and tasked it with forming different shapes to set a baseline and show that the reproduced system works. Finally I tasked the system to form targets existing of multiple shapes that are not connected.[1] I then extended this system to make use of bridges to form multiple separate shapes.

This paper is further organized as follows. First, in section 2, I will go more into depth about how this swarm system works and what modifications were necessary to make it work in a simulated environment. Then I will introduce the different experiments that I did and show the results in section 3. In section 4 I go more into depth about how the addition of bridge formation could allow this kind of swarm system to form multiple shapes that are not connected. Finally section 5 will contain my conclusion.

2 Reproducing the swarm system

2.1 General state-based behavior

The behavior of a robot can be divided in different states, I therefore modeled the agents as finite state machines. The seed agents start in the *JOINED_SHAPE* state, which is the final state of an agent and signifies that the agent has joined the target shape. All other agents start in the starting state *WAITING_TO_MOVE*. While in this state the agents wait until there are no moving agents visible, then the agent with the highest gradient within a range may start moving. If there are multiple agents in close proximity with the same highest gradient the agent with the highest unique identifier may start moving. Moving agents can be divided in two states. When an agent starts moving its state

[1] The code and the results for this project can be found in this repository

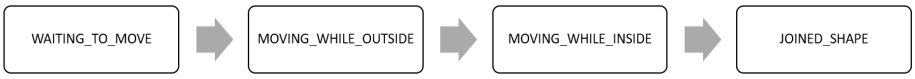


Figure 1: States of an agent

changes to *MOVING_WHILE_OUTSIDE*. Agents in this state are edge-following around the swarm. When an agent in this state is localized and realizes that it is inside the target shape, its state transitions to *MOVING_WHILE_INSIDE*. Agents in this state are still edge-following, but they are checking each step if they should stop and join the shape. These agents join the shape if they are again outside of the shape or if their closest neighbor has an equal or greater gradient. This behavior can be seen in figure 1.

This state based behavior causes the system as a whole to have multiple ending states. If all agents can fit inside the shape, they will all end up in the final state. If the shape is too large to fill by the agents it will be incomplete. On the other hand if the shape is too small to be filled by all agents, there will be some agents that will keep on edge-following around the shape. And finally, if the shape is very small the edge-following agents might inhibit other agents from ever leaving their starting state.

2.2 Gradient formation

To determine its gradient an agent collects the gradients that are communicated in its neighborhood. Then the agent searches for the smallest gradient among these and sets its own gradient as this value plus one. If no gradients are visible, the agent sets its own gradient as the maximum gradient plus one. This maximum gradient value is one of several global parameters that need to be set. The MASON framework supports lookups around a given location. To determine the visible gradients in the neighborhood I let the agent do a lookup to determine all agents within a given radius and collect the gradients from these. The algorithm assumes that the gradients increase with each hop, to accomplish this I set the lookup size to be $1,1$ unit [2] so only direct neighbors can be found. Because of this it was necessary to make a further modification to the algorithm. When an agent is moving it can happen that the agent has no neighbors within this close radius. This would cause the gradient to change to the maximum gradient plus one, which in turn could mess up the decision to join the shape. In my implementation the gradient is therefore not updated if it changes from a normal gradient to the maximum gradient. This has no real impact on how the system behaves as a gradient typically doesn't change after a single step. This change does however make the gradients more stable.

[2] A unit is equal to a step in the coordinate system and to the diameter of an agent.

2.3 Unique identifier

To make sure each agent has a unique identifier we could easily give each agent a unique number when instantiating them in a simulation. However, this is not possible in a physical system. The agents can select new identifiers by picking a random number. A binary variable determines if the agent has a locally unique identifier or not. If this variable is zero, the agent picks a new identifier. At each step the agent does a lookup within the largest visible radius it has and determines if its identifier is unique among these agents, setting the binary variable accordingly.

2.4 Localization

To determine its own location inside the coordinate system an agent does a lookup of the agents within a given radius. If there are at least three noncollinear, localized agents visible, the location of the agent can be determined. For each localized visible agent the agent determines a new location based on the measured distance and the vector between its current location and the location of the other agent. The current location is then updated by moving it one fourth of the way towards the new location. Each agent at the start has a location of $(0, 0)$, but is not localized. Only the four seed robots are localized from the beginning, they determine the centre of the coordinate system. To be able to see at least three noncollinear agents, the visible radius for this algorithm is set to three units. However, after experimentation with this algorithm I observed that the agents were not able to determine a stable location. I solved this instability by not immediately setting the binary variable indicating that the agent is localized to one. Instead the agent needs to have done sixty localization steps before this binary variable may be set to one. This way the localization process will only use locations that have already become somewhat stable.

2.5 Edge-following

To follow the edge of the swarm of agents an agent determines its distance to the closest non-moving agent in its neighborhood. By rotating clockwise or counterclockwise and moving a single step forward it tries to bring this distance closer to a certain value. I have set this desired distance to 1.1 unit.

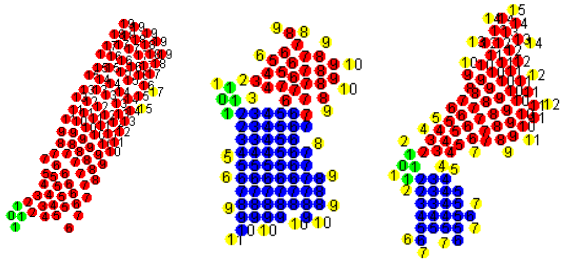
2.6 Visualization

Each agent in the simulation is represented as a circle. To make the underlying processes within the swarm better visible I let the agents draw their representations themselves. This way I could let the agents have a different color based on their state: green agents are seeds, blue agents are waiting to move, yellow

agents are moving outside the shape, orange agents are moving inside the shape and red agents have joined the shape. This also allowed me to display the gradient of each agent inside their representation.

3 Experiments

In the experiments I gradually increased the target complexity from basic shapes to complex shapes to multiple shapes that are not connected.[3] At initialization four seed agents are placed in a cross pattern and one hundred regular agents are placed in a grid below them. All these agents start with an orientation towards the bottom of the screen. In these experiments I observed that all the single target shapes are formed, but they are rotated over a small angle.



(a) Result for a 5 by 20 rectangle target

(b) Result for a 5 by 5 square target

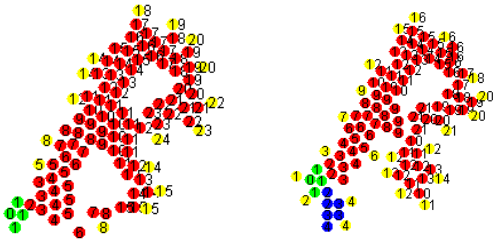
(c) Result for a triangle target

Figure 2: Result of the swarm when forming basic target shapes

The first experiments were with a rectangle, a small square and a triangle. All these shapes are formed and if there are agents left over they continue moving around the swarm. The rectangle has a size of five by twenty units and all agents are needed to fill it. The square and triangle are smaller and some agents are left moving around the swarm or are still waiting to move. The formed shapes are not perfect due to individual mistakes by some agents. The results can be seen in figure 2. I then moved on to some more complex shapes, using the capital letters B and R as targets. Again the shapes are rotated over a small angle, but otherwise formed correctly. These can be seen in figure 3. These experiments show that the reproduced system works as reported in [3].

To determine if my hypothesis is correct I then tasked the swarm system to assemble a target consisting of two shapes. One five by eight rectangle that

[3]Visualizations of these experiments can be found in the Examples folder in the code-base or in the readme in the github repository

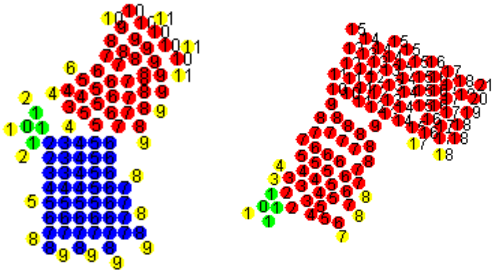


(a) B

(b) R

Figure 3: Result of the swarm when forming capital letters

starts from the seed position and a ten by five rectangle separated by two rows were used as target. In this case the hypothesis is confirmed as the swarm only forms the first shape and ends with agents that are endlessly moving around the swarm or waiting to start moving. However, if these shapes are only separated by a single row two bridges are formed between the rectangles and both are formed. These bridges are formed by agents that move a bit to far from the first rectangle and accidentally enter or think that they have entered the other rectangle and stop moving. By using these bridges the other agents are able to form the second rectangle. These results are shown in figure 4.



(a) Two rows apart

(b) One row apart

Figure 4: Result of the swarm when forming two shapes that are separated

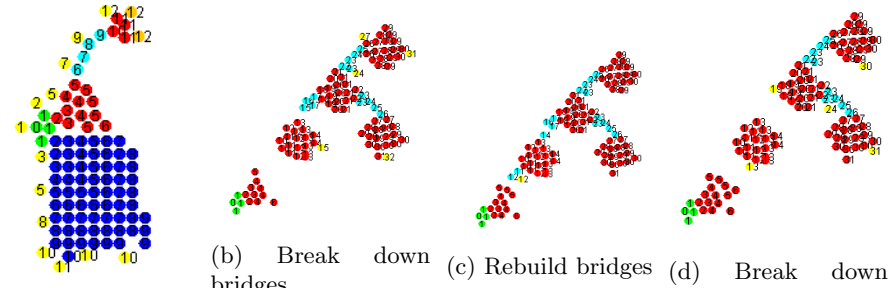
4 Bridge formation

The bridge-forming in the last experiment resembles ants that form bridges to traverse difficult terrain. By basing ourselves on this idea it might be possible to extend this swarm system to use bridges to be able to form multiple shapes that are not connected. A new state would need to be added to mark that an agent is part of a bridge and needs to start moving again later on. This brings up multiple problems that need to be solved. How to determine when a bridge needs to be formed and when it needs to be dissolved? How to determine the order of the shapes that need to be formed? How to determine a seed for the gradient and location in the separated shapes?

To solve this last problem we could stop gradient formation and localization when an agent has joined the shape. To dissolve the bridge we can use the property that remaining agents keep moving around the shapes. If the agents that form a bridge know the way the agents should be moving over the bridge, they could start moving again when they see an agent moving in the opposite direction. The hardest problem still remains, the swarm needs to be modified so they can know when and where to form a bridge and where it should go to. To determine the order of the shapes that need to be formed we could represent the target shapes by a fully connected graph where each node represents a shape. To determine the order we could then employ a shortest path algorithm. We then only need to determine where the bridges need to be located. If we assume that this problem is solved when processing the input image we can add a representation for the bridge to the input shapes.

Using these ideas I modified the agents to be able to form bridges. The bridge locations are given by separate markings in the used shape file. The agents count the amount of steps or time outside a bridge, if this value is high enough and the agent is localized in the bridge it joins this bridge. When inside the bridge the agent again measures how many steps it is part of the bridge, when this value is high enough it can start to check if the bridge can be broken down. The bridge is broken down when an agent is seen that moves in the opposite direction of the bridge, downwards for an upwards bridge and left for a bridge to the right. The bridge is broken down by letting the agent with the lowest gradient move again when no other moving agents are visible. This way the bridge is removed from the beginning to the end, allowing these agents to be used to form a shape further on. Some examples of building and breaking down a bridge are given in figure 5.

The agents that are part of a bridge have a cyan color. The example shows that bridges are build to be able to form other shapes and are broken down when no longer needed. A bridge that has been broken down can be rebuild when necessary to return to a previous shape. When there are no longer enough agents to continue building complete bridges, the remaining agents endlessly join and leave bridges around a shape. A problem with this system is that localization



(a) Use a bridge (b) Break down bridges (c) Rebuild bridges (d) Break down bridges

Figure 5: Swarm using bridges to form five rectangles

can become very unstable as a bridge is singly connected and agents need to see three noncollinear agents to be able to locate themselves. Sometimes the bridge makes a turn and the next shape is formed in the wrong location. A possible solution could be to scale up the system to use more agents and make bridges wider. This would increase the amount of visible localized agents. The experiments done in this project serve as a proof of concept that the idea of bridge building can be used to allow this swarm to form multiple shapes that are not connected.

5 Conclusion

In this project I reproduced and researched the self assembling swarm system from [3] in MASON. I first showed that this reproduced system works as intended and can form complex shapes. Then I confirmed my hypothesis that this system only forms the shapes that are reachable and does not consider the other shapes. These experiments did show a special case where the hypothesis does not hold. Bridges are constructed when the separate shapes are close to eachother, allowing the swarm to form multiple shapes that are not connected. I then used this idea of constructing bridges to extend this swarm system. This extended system is able to build bridges to form shapes that are not connected. The constructed bridges are then broken down when they are no longer necessary.

References

[1] K. Arnold, J. Gosling, and D. Holmes. *The Java programming language*. Addison Wesley Professional, 2005.

[2] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan. Mason: A multiagent simulation environment. *Simulation*, 81(7):517–527, 2005.

[3] M. Rubenstein, A. Cornejo, and R. Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.