NTNU

Norwegian University of
Science and Technology

Artificial intelligence

IE502014

Swarm intelligence

Assignment 3

Author:
Nicolò Pinciroli

March 26, 2020

Introduction

This part of the assignment was not required, but it is part of the first exercise of the first assignment

A **system** is defined as a set of abstract, technological or living organizations cooperating on a common purpose. More formally, a system can be described using the following (set) notation:

$$S = \{B, S(s)\}$$

that means that a system S is formed by several subsystems $S(s)$ and that there exist some relations between the subsystems, indicated by B. A system is complex when it has more than two subsystems. In particular, a complex system can be represented as:

$$S = \{B, \{A, L\}\}$$

where A represents a set of agents and L represents a set of landscapes (the environments in which the agents operate). B, in this context, represents the relationships between the agents and the landscapes, modelled as a network.

An **agent system model** is a model where the agents, the landscapes and the network that models the relationship between agents and landscapes are defined, as described in the definition of complex system. For instance, the agents could be robots, the environment could be a labyrinth and the relationships network could be the set of interactions between the robots and the labyrinth.

An **agent**, therefore, can be defined in different ways. For instance, they can be seen as goal seeking objects (for instance robots looking for an exit from the labyrinth or ants looking for food). In case agents are described in the context of a more complex system, instead, they can be seen as system elements. Another possibility is to see agents as consumer (for instance, a car consumes fuel) and producers (that car moves as a consequence of fuel consumption). A **landscape** can be defined as the environment in which the agents are and operate. Agents can usually interact with the environment (for instance, a robot in a maze might have some sensors to see the location of the walls). A **network** can be modeled as a graph, and a network that evolves over time can be defined as a temporal graph. In this context, a network encodes the relations between the agents and the landscapes.

A **swam** is a group of independent agents that cooperate in order to reach a goal or to perform some actions. For instance, a swarm of birds consists of many birds that think independently but that are also able to remain compact and to perform actions as a group. The same phenomena has been observed in other situations (for instance, for bees and ants, but also for human beings that want to take the best decision on a certain topic).

Swarm intelligence is the intelligent behaviour resulting from the interaction of several agents belonging to a swarm. Those agents behave as described in the previous paragraph. Even if the behaviour of a single agent is not particularly intelligent, the behaviour of the entire swarm can be more intelligent than the one of an individual.

Let's consider an **agent-based system model for an industrial fish farm**. In this context, it is possible to study the problem from different perspectives. For instance, it could be possible to study the system from a business point of view, by considering the different stages of the production. However, I assume the question is referred to the fish behaviour. This means that the agents are fish, therefore they are able to move in a three dimensional space and to change their velocity. Moreover, it is possible to assume that there exist a sort of communication in the fish school, so that they can perform movements and actions in a coordinated way. The landscape, therefore, would be the tank in which the fish are. It is possible to consider, for example, that some food is introduced in the tank, that it is possible to introduce new fish or to remove some fish. In this model there would be only one type of agent, that is a fish. It would be possible to introduce several kinds of fish, with different behaviours, and to model them as different agents.

It is also possible to identify, in this context, the model purpose end ethics. Considering the perspective of a fish, the purpose could be to eat the food provided from the external environment, to follow the other fish in the school, and if we assume that the fish realizes it is in a fish farm, to escape when someone tries to catch it.

Some typical rules for learning can be related to a reward system: the knowledge of a fish improves when examples and consequences are provided. For instance, if a fish sees that all the other fish group together, it will probably learn to do the same. I think, therefore, that reinforcement learning could be a good learning method. Some other rules could include, for instance, trying to reach the food in the fastest time to have more of it.

Producer-consumer agents

This part of the assignment was not required, but it is part of the first exercise of the first assignment

A **producer-consumer agent** is an agent that consumes some resources to produce other resources. For instance, if we consider a panda as an agent, it will consume bamboo in order to move and to remain alive. If we consider a car, that can be modeled as an agent, it consumes fuel to move its wheels and go forward. A producer-consumer agent, therefore, can represent anything that, given some resources, outputs something, obtained after having elaborated the input resources. This transformation can usually be modeled using a mathematical function, such as the logistic function. Changes in this function allow the producer-consumer agent to adapt and to customize the production/consumption capacity in the context of a landscape.

A producer-consumer agent can be seen as a system, and it is also part of a subsystem, in which there can be other agents and the environment. If the agent, for instance, consumes too many resources, then there won't be any resource left for the future. In this case it is possible to describe the behaviour of the agent using a time-dependent representation (for instance by using differential equations, that can be discretized) and to use a PID controller to maintain the overall system stable.

In order to maintain a stable production through time, it is possible to perform an optimization on the mathematical function that describes the behaviour of the system. In order to optimize the value of its production, the same function has to be optimized by considering the output value and not, as in the previous case, the production stability.

1 Goal-oriented agent

A movable goal-oriented agent is an agent that is able to move in a landscape towards a goal. In a graphical context, where the environment can be represented by a tridimensional euclidean space, such an agent can be the model of a bird that flies towards a point (for instance, the top of a tree), that is its goal. Considering again the fish farming example, the fish can be considered as movable goal-oriented agents because they move towards the food, that is the goal. In general, the landscape is not necessarily a tridimensional euclidean space, since it can be a multidimensional space whose meaning is not geometrical, but the idea behind this kind of agent is the same.

This kind of agent can control its speed for instance by increasing it when the goal is far away and by decreasing it when the goal is closer, in order to be more precise when it comes to "hit" the goal. As for the direction, this is problem-dependent. In case the agent knows the location of the goal and the configuration of the environment, it is possible to perform a search in the space (similarly to the search in a labyrinth), while if there are some parts of the environment that are unknown to the agent, it needs to use heuristics and to make assumptions to determine the correct orientation in the space.

If there exist multiple targets, it is possible to select the best target on the basis of an estimate of the "value" of the target and of the resources used to reach it. In practice, it is necessary to have a model that, given a target, is able to return a value that indicates the desirability of the target. For instance, if the model says that the best agent is the closest, then the agent will move towards the closest agent, but if the positions change through time, it has to check the desirability of each target for each iteration.

2 Flocking behaviour

Reynolds boids model is a model that describes the behaviour of social agents, that are individual agents who have a common set of rules. The performance optimization is carried on by each agent (individual), that is part of a larger group, on the basis of the group behaviour. In this way the group performance improves thanks to socialization with other agents (neighbours) without the need of a centralized control.

In particular, boids have been introduced by Reynolds, and they can be used to simulate artificial life, taking inspiration from the flocking behaviour of birds.

In particular, it is important to consider that boids have a geometric shape, which could be used to represent them graphically. Their shape can allow the definition of a direction, similarly to what happens in the case of birds. It can be particularly useful to define a local coordinate system for every boid, so that they can move with respect to other boids according to some rules. In particular, some simple behaviours are generally allowed and they can be encoded by geometrical transformations (such as rototranslations). Moreover, it is possible to change the position, the velocity and the acceleration of boids thorugh time.

Some possible behaviours of boids are:

- **Cohesion**: tendency to stay close to nearby flockmates. This means that, from the set of all the boids of the flock, a subset is considered, and this subset is formed by the boids that, at a given moment, are closer to the boid currently under analysis. This behaviour increments the tendency of the boids to group together.

- **Separation**: this behaviour has the opposite effect with respect to the cohesion behaviour, since it brings a boid further away from other boids. This allows avoiding collisions (such as in a real bird storm).

- **Alignment**: in case a direction is defined (this is not representable clearly in case the boid is a symmetric shape, such as a sphere), the boids can rotate around their axes (this is one of the reasons why it is convenient to define a local reference system for every boid). Alignment is very useful in order to move in the same direction as the other boids with the same (or a similar) velocity and acceleration. This phenomenon is also observable in nature, since bird flocks usually move

compactly towards one direction or following a path, otherwise the flock would be chaotic and not efficient.

2.1 Formulas

The cohesion, separation and alignment behaviours could be expressed also by using mathematical formulas. In particular:

- **Cohesion**: $\kappa.pos = \sum_{i \neq j} \frac{A_j.pos - A_i.pos}{n-1}$. In this case $\kappa.pos$ represents the cohesion center position. Moreover, A_i is the agent considered in this moment and A_j are the other agents, considered one by one. n is the number of agents.

- **Separation**: the separation force should increase when the distance from a given agent decreases. It is necessary, therefore, to calculate the distance of A_j from A_i as $\delta_j = A_j.pos - A_i.pos$. It is possible, then, to normalize this value by dividing it by the square of the distance ($\hat{\delta}_j$. Finally, it is possible to calculate the sum of the distances, agent by agent: $\sigma_i = \sum_j \hat{\delta}_j$

- **Alignment**: $\alpha.speed = \sum_j \frac{A_j.speed}{n-1}$. This means that the speed of the swarm is given by the average of the speeds of all the agents. This also implies that an agent A_i should change its velocity in order to have the same velocity as the rest of the storm: $A_i.speed = \alpha.speed - A_i.speed$, where this formulation can be seen as a line of code (the left-hand side represents the value of the speed after the assignment).

Note: in my practical implementation, the boids are represented as spheres, and rotation hasn't been explicitly defined, but position, velocity and acceleration change according to the flock behaviour anyway.

2.2 Notes on the implementation

The implementation I have proposed is based on some code adaptations with respect to the style I would have used in Unity3D. In particular, handling time is not completely automatic in Python, and my assumption is that the information are updated every timestamp. Considering the formula that determines the position of a body given the initial position, the velocity and the acceleration (for one space component):

$$s = s_0 + v \cdot t + \frac{1}{2} \cdot at^2$$

if $t = 1$ then the numerical value of the space component will be:

$$s = s_0 + v + \frac{a}{2}$$

where s_0 represent the current position of the object, v represents the initial velocity in the previous iteration and a represents the acceleration with respect to the previous iteration (where an iteration is equivalent to 1 timestep).

The cohesion behaviour is implemented by using `cohere`, that takes the cohesion factor as input (the cohesion factor is related to the number of agents involved in the calculation) and the list of all the agents. Then, for each agent, the agent that is making the calculations sorts all the agents by distance, takes the first k agents, where k is the cohesion factor and accelerates towards the agent of the group of the closest agents.

From a programming perspective, this functionality has been implemented as a "centralized" function, but it would be straightforward to associate `cohere`directly to the agent class, since it simply contains a loop that goes through all the agents.

The principle behind separation is similar, but in this case there are two parameters other than the list of agents: `locality`, that refers to the dimension of a cluster of nearby agents, and `minAllowed`, that refers to the minimum distance allowed between two agents. Those parameters are global parameters, valid for the entire simulation.

As for alignment, it isn't completely represented graphically, since the agents are shown as spheres, but is consists in knowing the position of the center of the swarm, that moves towards a target. Firstly, knowing the distance travelled by such central agent, a new fictitious target is set for each agent. This fictitious target is, for each spatial component, at the same distance as the distance between the current and the previous central agent positions for that component. Then, each agent is accelerated to its fictitious target. In this way a sort of rigid translation is implemented, but the translation is not instantaneous.

Note: in the second version of the code `cohere`*,* `align` *and* `separate` *are defined in the Agent class, to design a more object-oriented architecture.*

2.3 Result

The proposed simulation has been developed for a limited number of boids in order to keep a fluid visualization. This is a limitation, since the algorithm could probably be optimized. An example of optimization could consist in using the Barnes-Hut algorithm to subdivide the space into clusters and in using an octree to memorize the positions of the particles. This kind of approach is one of the methods used to solve the n-body problem with a complexity in $O(nlogn)$, while the method I am proposing probably performs some operations (such as sorting) for more times than necessary. Moreover, Barnes-Hut could be used to model cohesion and separation not only when considering particle-particle interactions, but also for cluster-cluster and particle-cluster interactions. This approach would also ease the implementation of a parallelized version of the code (Barnes-Hut, indeed, could be used in complex computer architectures in order to divide the resources among different calculators, generally by considering also a load balancing policy).

Not using an algorithm to have cluster-cluster interactions means that the boids tend to group into isolate clusters. However, it is possible to change the maximum number of elements in each cluster, both for separation and cohesion, to observe some differences. Moreover, considering line 126 in the second version of the code, it is possible to change the hard-coded factor 0.2 `self.acc[l] = 0.2*self.distanceC(centerAgent, l)`. Incrementing it would bring to a faster movement towards the fictitious target and vice-versa.

3 Flocking behaviour applications

Queuing agents are social agents able to show a steering behaviour in absence of a centralized control, so that they can approach, for instance, a narrow slit and pass through avoiding overlapping the ones with the others and with the walls.

A typical example of queuing agents could be some people that have to enter the door of a train. Of course they cannot physically overlap and they don't want to hit the train while they enter. Moreover, there isn't a centralized behaviour able to decide who should enter before. Generally, people can approach the door chaotically and then let the closest person entering.

Queuing agents could, for example, proceed through a slit, or a door, or a narrow passage, one by one, creating a non-chaotic queue. This phenomenon could be used in many fields, from movies to physical and biological phenomena (for instance, ants that should go through a narrow passage to get to their anthill, maybe trying to escape from an anteater).

It is possible, therefore, to model those agents as normal social agents with some additional rules (such as obstacle avoidance, a well-defined goal and a performance measure that is able to measure how the agents have performed in reaching the goal).

It is also possible to use **genetic algorithms** together with queuing agents. In particular, given the problem of having several agents that have to pass through a narrow slit, it could be possible to run several simulations without any rule related to the amount of cohesion and separation and to tune them using GA in order to optimize performance.

My implementation consists of agents that passes through a narrow slit, following a leader. The leader has been represented in red, while the position to which the leader has to arrive has been represented in green. There exist several agents, that are able to identify the position of the leader and the path it has followed, placing themselves in those positions.

I have added some constraints related to the distance the agents should have. This approach is similar to the separation implementation, while following the leader could be seen as a variation of the cohesion implementation. However, I have slightly changed those definition when I have written the actual code, since the problem has an additional constraint, that is the presence of the slit.

One of the challenging parts in programming using vpython is that some

of the functionalities of Unity3D are not natively available, therefore it is necessary to re-implement them. The consequence of this challenge is that a Unity3D project would have been more flexible than the one I have realized. At the same time, I think that those challenges have given me the possibility to study more in depth the problem.

It is possible to change the parameters related to acceleration, for instance, and the consequence is that higher accelerations cause more overcrowding when the boids go from their original positions to an intermediate position.

4 Steering behaviours

The **seek** behaviour consists in moving the character towards a static target, that has a specific position in the space under consideration, that could be represented graphically as a bidimensional or tridimensional space, for example. This behaviour orientates the object towards the target. In order to do so it is necessary to increase the velocity of a quantity equal to the difference between the desired velocity and the current velocity, once the alignment has been set correctly. Moreover, the object might have a maximum speed, therefore it is necessary to check whether this is not exceeded during the calculations. The application of this behaviour does not guarantee that the object stops when it reaches the target, differently from what would happen in the arrival behaviour.

The **flee** behaviour is conceptually the same as the seek behaviour, with the difference that the body and its velocity point to the opposite direction with respect to the target (that acts as a sort of repellent).

The **pursuit** and the **evade** behaviours are similar to the seek and flee behaviours, with the difference that the target is moving. This means that seek and flee are actually special cases of pursuit and evade, where the target velocity is zero.

The **wander** behaviour consists in random steering, without a specific goal. However, just deciding a random direction for each timestep would not produce an interesting behaviour, since on average the body would likely remain in the same place or have a twitchy behaviour. This is the reason why the wander behaviour actually proposes a random walk, based also on the recent history of the body under consideration.

The **arrival** behaviour is similar to the seek behaviour, but in this case the body does not move towards the target at full speed, preferring a slower approach to it. This means that the velocity depends on the current distance from the target, and it decreases when the target gets closer. This phenomenon could be seen as a sort of feedback control.

Obstacle avoidance requires the identification of obstacles in the scene. There can be several approaches to do so. Some environments, such as the one provided by Unity3D, automatically map the obstacles and give the possibility to add physics, so that the bodies don't go through other bodies. Implementing such a behaviour is more difficult when other development environment are used. The principle behind obstacle avoidance is that, given an obstacle, the body does not hit the obstacle, so that, given for instance

a thin slit, the body could pass through it without hitting the walls. A possible approximation could be that both the obstacle and the body are approximated as spheres. This behaviour has also some similarities to the offset pursuit behaviour, that given a target, is able to pass close to it without actually touching it.

The **containment** behaviour is a variation of path following, described below, and consists in remaining in a certain bounded region, for instance between two walls.

The **wall following** can also be considered as a variation of path following. The goal of this behaviour consists in following walls, maintaining a certain offset from them. The idea is that the object gets closer to the wall and then, when a certain distance from it has been reached, it changes its direction and follows the shape of the wall.

The **path following** behaviour consists in the ability to walk on a predesigned path or some intermediate destinations on the basis of calculations.