# Ant Colony Optimisation (ACO)

Nous jalousons nos proches voisins, mais pas le soleil et ses soins.

--- La Fourmi; Charles de Leusse, Fables 1

# Project Details

| | | |
|---|---|---|
| **Ant Colony Optimisation** | https://gitlab-cw5.centralesupelec.fr/groupe-02/aco | Chensheng Luo |
| | | Haonan Lin |
| | | Mingshan Ye |
| | | Raven Bast |
| | | Yue Yang Oo |

# Project Description

- A zero-user simulation game
- Simulates ACO algorithm which aims to find the best path in search of food using pheromone-based communication method.
- Possible usages:
  - Allow user to simulate and understand how ants use group intelligence to find food
  - Can be a model in predicting real life group behaviour, such as human traffic

# Project Structure

- MVP: Simple simulation of ACO on a pre-defined map, with animation by pygame
  - 2 Classes as primary data structures
    - Cell and Ant
    - Respective functions defined for each class
- Final Product: Interactive simulation of ACO with user definition of map
  - Allows the user to generate a random map or draw one
  - 3 Classes as primary data structures
    - Cell, Ant and Map
    - Classes used for the GUI interface as well

# Qualitative Elements

## Modularity

- The representation of the ant, cell (and the map) are all done by self-defined **class**.
- The graphique interface module use the pygame, with several class defined in **gui_utils.py** by ourselves
- Random and math modules for randomly generated map and probablistic selection

## Parameters

- Configuration of map
  - Size of map
  - Cell-related settings (food positions, nest positions, number of food sources etc.)
- Configuration of ants
  - Number of ants generated

```
import pygame as pg
import time

import math as m
import random as r
from random import randint

from ant import Ant
from map import Map
from cell import Cell
from gui_utils import *
```

# Tests Implemented

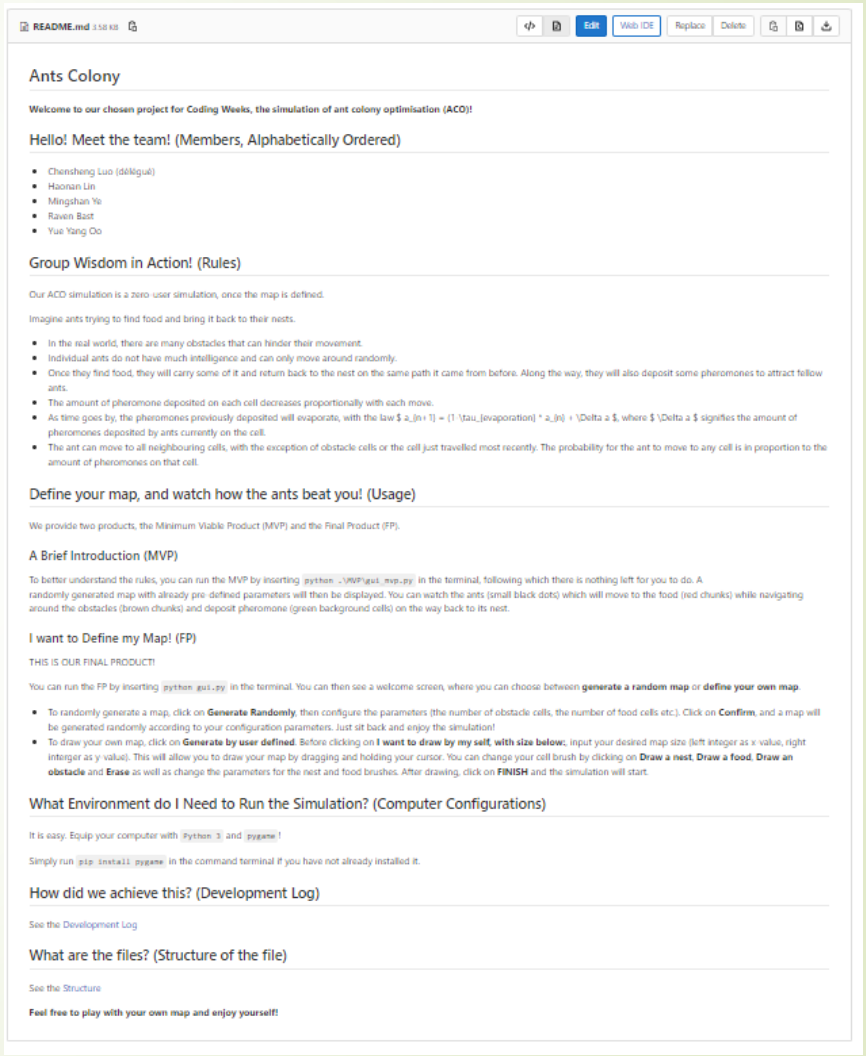
- Tests are made for each class (**ant, cell, map**), the **gui** file and the **gui_utils** module.

Coverage report: 88%

| Module ↑ | statements | missing | excluded | coverage |
|---|---|---|---|---|
| ant.py | 87 | 29 | 0 | 67% |
| cell.py | 53 | 5 | 0 | 91% |
| gui.py | 439 | 51 | 0 | 88% |
| gui_utils.py | 187 | 20 | 0 | 89% |
| map.py | 185 | 11 | 0 | 94% |
| test_ant.py | 33 | 1 | 0 | 97% |
| test_cell.py | 34 | 1 | 0 | 97% |
| test_gui.py | 5 | 1 | 0 | 80% |
| test_gui_utils.py | 23 | 1 | 0 | 96% |
| test_map.py | 37 | 5 | 0 | 86% |
| **Total** | **1083** | **125** | **0** | **88%** |

# Documentation

- **README.md** to show the members, rules, usage, configurations (as a guide for users)
- **Development.md** to show the rules, the sprint defined
- **Structure.md** to show the structure



README.md 3.58 KB

Edit | Web IDE | Replace | Delete

## Ants Colony

**Welcome to our chosen project for Coding Weeks, the simulation of ant colony optimisation (ACO)!**

### Hello! Meet the team! (Members, Alphabetically Ordered)

- Chensheng Luo (dèlìguā)
- Haonan Lin
- Mingshan Ye
- Raven Bast
- Yue Yang Oo

### Group Wisdom in Action! (Rules)

Our ACO simulation is a zero-user simulation, once the map is defined.

Imagine ants trying to find food and bring it back to their nests.

- In the real world, there are many obstacles that can hinder their movement.
- Individual ants do not have much intelligence and can only move around randomly.
- Once they find food, they will carry some of it and return back to the nest on the same path it came from before. Along the way, they will also deposit some pheromones to attract fellow ants.
- The amount of pheromone deposited on each cell decreases proportionally with each move.
- As time goes by, the pheromones previously deposited will evaporate, with the law $ a_{n+1} = (1-\tau_{evaporation}) * a_{n} + \Delta a $, where $ \Delta a $ signifies the amount of pheromones deposited by ants currently on the cell.
- The ant can move to all neighbouring cells, with the exception of obstacle cells or the cell just travelled most recently. The probability for the ant to move to any cell is in proportion to the amount of pheromones on that cell.

### Define your map, and watch how the ants beat you! (Usage)

We provide two products, the Minimum Viable Product (MVP) and the Final Product (FP).

#### A Brief Introduction (MVP)

To better understand the rules, you can run the MVP by inserting `python .\MVP\gui_mvp.py` in the terminal, following which there is nothing left for you to do. A randomly generated map with already pre-defined parameters will then be displayed. You can watch the ants (small black dots) which will move to the food (red chunks) while navigating around the obstacles (brown chunks) and deposit pheromone (green background cells) on the way back to its nest.

#### I want to Define my Map! (FP)

THIS IS OUR FINAL PRODUCT!

You can run the FP by inserting `python gui.py` in the terminal. You can then see a welcome screen, where you can choose between **generate a random map** or **define your own map**.

- To randomly generate a map, click on **Generate Randomly**, then configure the parameters (the number of obstacle cells, the number of food cells etc.). Click on **Confirm**, and a map will be generated randomly according to your configuration parameters. Just sit back and enjoy the simulation!
- To draw your own map, click on **Generate by user defined**. Before clicking on **I want to draw by my self, with size below:** input your desired map size (left integer as x value, right integer as y value). This will allow you to draw your map by dragging and holding your cursor. You can change your cell brush by clicking on **Draw a nest**, **Draw a food**, **Draw an obstacle** and **Erase** as well as change the parameters for the nest and food brushes. After drawing, click on **FINISH** and the simulation will start.

### What Environment do I Need to Run the Simulation? (Computer Configurations)

It is easy. Equip your computer with `Python 3` and `pygame`!

Simply run `pip install pygame` in the command terminal if you have not already installed it.

### How did we achieve this? (Development Log)

See the Development Log

### What are the files? (Structure of the file)

See the Structure

**Feel free to play with your own map and enjoy yourself!**

# Basic Simulation Rules

| | **The simulation is set in a rectangular map with cells.** |
|---|---|
| 1. | All ants start from the nest. All cells are assigned with the same initial level of pheromone (very little). |
| 2. | Ants choose the next cell to travel to according to a probabilistic approach; Higher level of pheromones on the neighbouring cell indicates higher probability. They also memorise their individual paths taken. |
| 3. | Upon reaching the food source and picking up food, they travel back to the nest following the same paths as before. |
| 4. | On their way back, they will deposit decreasing amounts of pheromones on every cell of their path with respect to the distance travelled. |
| 5. | The pheromone trail will start to evaporate as soon as it is deposited. |
| 6. | Once ants deposit food at the nest, they will restart the whole process of finding food again. |
| 7. | A few ants are generated from the nest on every generation. |
| 8. | Ants can only make one cell move per generation. |

# Functionalities Timeline

- **Sprint1: Build cell as a class**
- Initialization of attributes: position_x, position_y, ants, type, pheromone
- Definition of functions: def_type, add_ant, delete_ant, delete_all_ant, updat e_pheromone,update, have_ant, distane_to

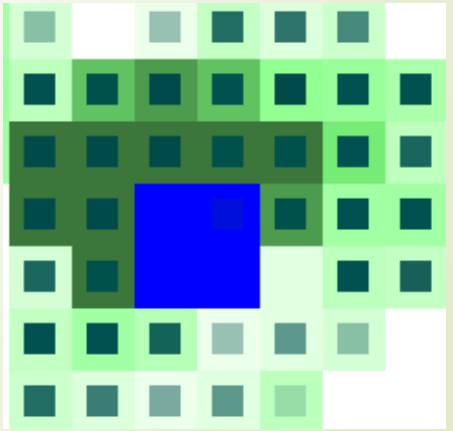
- **Sprint2: Build ant as a class**
- Initialization of attributes: cell, history, carry_food
- Apply rules of ACO simulation to the ants: move_ant

$$\tau_{xy} \leftarrow (1-\rho)\tau_{xy} + \sum_k \Delta\tau_{xy}^k$$

```
#Constant
RATIO_EVAPORATION = 0.1  # The ratio of evaporation
INITIAL_PHEROMONE = 1  # The minimum pheromone level for each cell

# Marker of type
FOOD = 'F'
NEST = 'N'
OBSTACLE = 'O'
OTHER = 'A'
```

```
possible_movements.append(map[neighbour_y][neighbour_x])
 weights_list.append(last_resort_nest.pheromone)
choices(possible_movements, weights=tuple(weights_list), k=1)[0]
```



# Functionalities Timeline

- **Sprint3: Create a map**
- Creation of a map with nest, food and obstacles
- Creation of ants at the nests and update the map for the next move

- **Sprint 4: Simulation of ACO and Display of animation**
- Simulation of ACO with one food, one nest, simulation_MVP.py
- Display animation with a simple user interface, gui_MVP.py

```
######### Parameters of the universe
numFood = 5
numObstacle = 10
foodList = []
food_amount = []
init_food_amount = 50  # Initial amount of food
```

```
def main():
    pg.init()
    pg.display.set_caption("Ant Colony Simulation MVP")
    screen = pg.display.set_mode(
        [int(MAP_WIDTH * scale), int(MAP_HEIGHT * scale)])
```

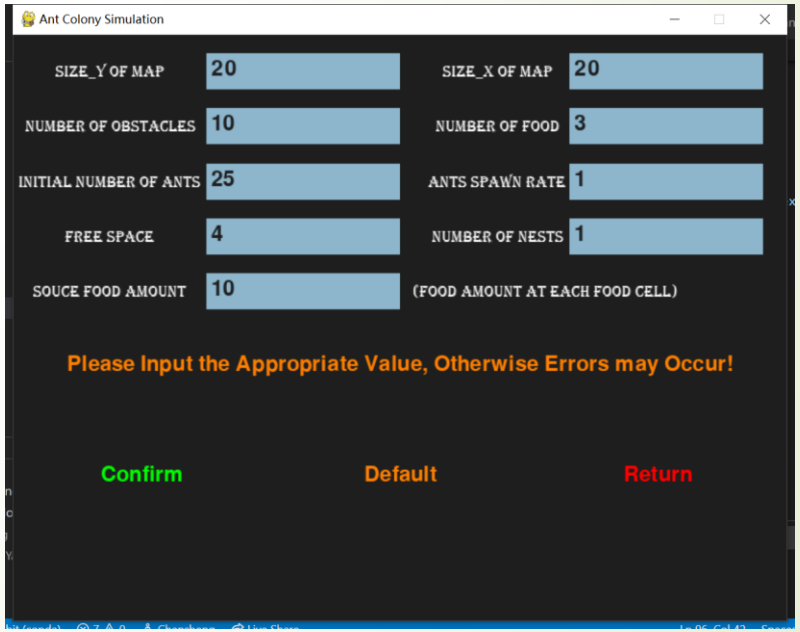
# Functionalities Timeline

- **Sprint 5: Build a graphical user interface**
- Define the different basic widgets in gui_utils.py
- Build the graphical interface by using these widgets

```python
class Button(object):
    def __init__(self, text, color, screen, x_percent=None, y
        self.surface = FONT.render(text, True, color)
        self.bg_color = (225, 225, 225)
        self.color = color
        self.WIDTH = self.surface.get_width()
        self.HEIGHT = self.surface.get_height()
        self.screen = screen
        self.in_click = False
        self.click_loss_time = 0
        self.click_event_id = -1
        self.ctl_id = BFControlId().instance().get_new_id()
        self._click = click
        self.is_hover = False
```

```python
class InputBox(object):

    def __init__(self, x_percent, y_percent, w_percent, h_percent, sc
        w = int(w_percent * SCREEN_MAX_WIDTH)
        h = int(h_percent * SCREEN_MAX_HEIGHT)
        x = max(int(SCREEN_MAX_WIDTH * x_percent - w // 2), 0)
        y = max(int(SCREEN_MAX_HEIGHT * y_percent - h // 2), 0)
        self.rect = pg.Rect(x, y, w, h)
        self.color = COLOR_INACTIVE
        self.txt_color = pg.Color('gray12')
        self.text = text
        self.key = key
        self.txt_surface = FONT.render(text, True, self.txt_color)
        self.active = False
        self.callback = callback
        self.screen = screen
        self.state = 0   # the initial status equal to 0, input statu
```



# Functionalities Timeline

- **Sprint 6: Amelioration of the map**
- Change the map to be a map class so that the information is stored as an entity, by the initialization of attributes (whether generate randomly or not) and the adaptation of the function already written

```
MVP > gui_mvp.py > ...
26
27  ######### Parameters of the universe
28  numFood = 5
29  numObstacle = 10
30  foodList = []
31  food_amount = []
32  init_food_amount = 50  # Initial amount of food
33
```

```
class Map(object):
    '''
    NB: In this file, all of the parameters has been defined as PRIVATE VARIABLE
    as once they are created, they can't be changed random, they can change it or
    '''
    # Map dimension
    __map = [] # The map itself
    __len_y = 0 # The length y of the map
    __len_x = 0 # The length x of the map

    ### Ants in map
    __all_ants = [] # The list of all of ants
    __numAnts = 2000 # Maximum limit for number of ants TODO: do a change
    __numAnts_spawn_per_round = 10 # Number of ants spawned per round

    ### Nest in map
    __numNest = 1 # Amount of nest cells in map
    __nestList = []

    ### Food in map
    __foodList = []
    __numFood = 10 # Amount of food cells in map
    __min_food_dis = 20 # Minimum distance to food on each side
```

```
# These functions gives the parameters needed, without change it
def map(self): return self.__map
def all_ants(self): return self.__all_ants
def init_food_amount(self): return self.__init_food_amount
def foodList(self): return self.__foodList
def nestList(self): return self.__nestList
def len_x(self): return self.__len_x
def len_y(self): return self.__len_y
```

# Functionalities Timeline

- **Sprint 6: Amelioration of the map**
- Adapt the other files already written where use a map, add new parameters in ant
- Generate a map randomly which allows parameter configuration

```
def __random_add_nest(self):
    '''
    Generate nests randomly in the map
    '''
    #Generate num_ants_nest randomly
    num_ants_nest = [r.randint(0, self.__numAnts) for i in range(self.__numNest)]
    num_ants_nest = [0] + sorted(num_ants_nest) + [self.__numAnts]
    num_ants_nest = [num_ants_nest[i + 1] - num_ants_nest[i]
                     for i in range(self.__numNest - 1)]
    num_ants_nest.append(self.__numAnts - sum(num_ants_nest))

    #Generate its coordinate randomly
    for i in range(self.__numNest):
        self.__add_nest((r.randint(self.__freeSpace, self.__len_y-self.__freeSpace-1),
```

```
def __random_add_food(self):
    '''
    Generate food randomly in the map
    '''
    for i in range(self.__numFood):
        stopGenerateFood = False
        while not stopGenerateFood:
            foodX = r.randint(5, self.__len_x - 6)
            foodY = r.randint(5, self.__len_y - 6)
            foodSize = (r.randint(1, 4), r.randint(1, 4))
            foodSuccess = True
            for nest in self.__nestList:
                foodSuccess = foodSuccess and (self.__map[foodY][foodX].
            if foodSuccess:
                if self.__map[foodY][foodX] not in self.__foodList:
                    self.__add_food((foodY, foodX), size=foodSize)
                    stopGenerateFood = True
```

```
def __random_add_obstacle(self):
    '''
    Generate obstacles randomly in the map
    '''
    for _ in range(self.__numObstacles):
        stopGenerateObs = 0
        while stopGenerateObs < self.__numFood/2:
            obstacleX = r.randint(3, self.__len_x - 3)
            obstacleY = r.randint(3, self.__len_x - 3)
            obstacleSize = (r.randint(1, 2), r.randint(1, 2))
            for y in range(obstacleSize[0]):
                for x in range(obstacleSize[1]):
                    current_cell=self.__map[obstacleY+y][obstacleX+x]

                    if current_cell.type in [Cell.OBSTACLE, Cell.FOOD, Cell.NEST]:
                        continue

                    else:
                        nestSuccess = True
                        for nest in self.__nestList:
                            nestSuccess = nestSuccess and (current_cell.distance_to
                        if current_cell.type in [Cell.OBSTACLE, Cell.FOOD,
                            continue

                        else:
                            nestSuccess = True
                            for nest in self.__nestList:
                                nestSuccess = nestSuccess and (current_cell
                            if nestSuccess:
                                stopGenerateObs += obstacleSize[0] * obstac
                            else:
                                stopGenerateObs = 0
                                continue
                            for food in self.__foodList:
                                foodSuccess = current_cell.distance_to(food
                                if foodSuccess:
                                    stopGenerateObs += 1
                                else:
                                    stopGenerateObs = 0
                                    continue
            self.__add_obstacle([obstacleY, obstacleX], size=obstacleSize)
```

# Functionalities Timeline

- **Sprint 7: Allow the user to draw the map by themselves**
- Define the basic draw module in gui_utils.py: CellButton
- Create a clickable interface which can change the colour with user's click
- Store the information of the click and create a map

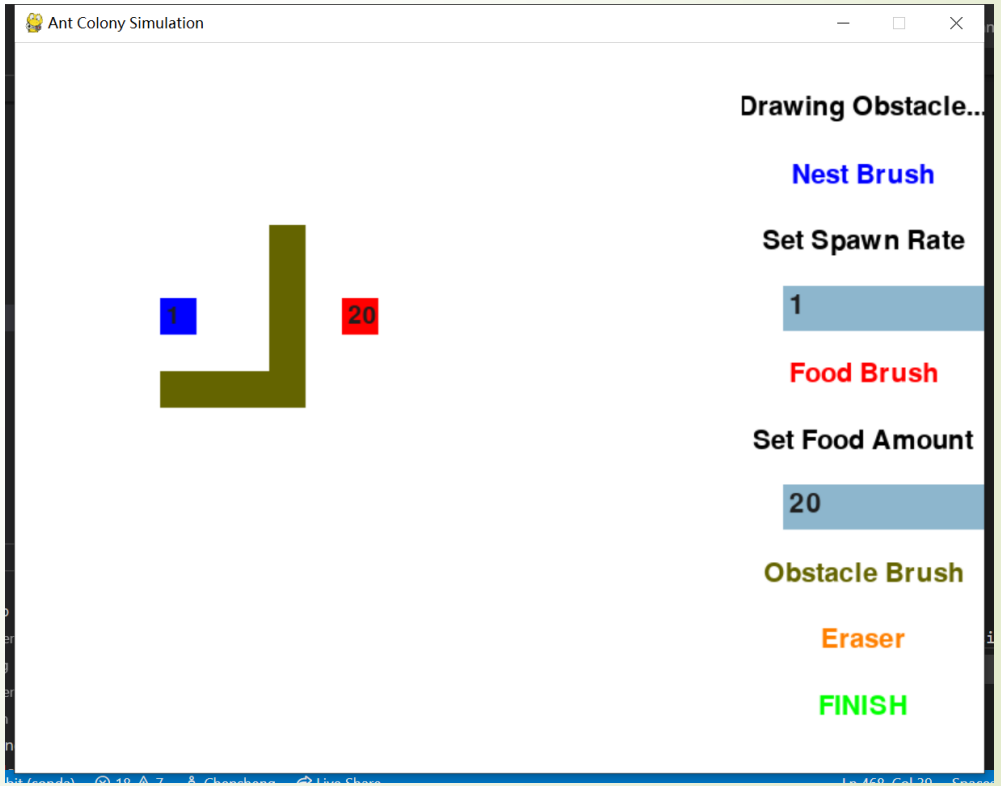
gui_utils.py

```
214
215
216  class CellButton(object):
217
218      def __init__(self, x, y, w, h, color, text=''):
219          self.__x = x
220          self.__y = y
221          self.__w = w
```

```
def userdraw(screen,size_x=20,size_y=20):
    '''
    Ask user to draw a map

    Parameters
    -------
    screen
        The screen to draw on
    size_x
        The size of the map,defalute to be 20
    size_y
        The size of the map,defalute to be 20

    Returns
    -------
    map_drawn
        Map, the map that the user have drawn
    '''
```



# Functionalities Timeline

- **Sprint 8: Amelioration of the rules to increase search food efficiency**
- Change the different parameters already defined to find the best configuration
- Modify rules and test the modifications (for example: the ant can't go to the cell that it had been to during the last 3 iterations (short memory of ant))

```
if map[neighbour_y][neighbour_x] == self.history[-4:-1]:
    last_resort = map[neighbour_y][neighbour_x]
    continue # Ant will not go back to any 3 cells it traveled before
```

# Division of Tasks

| | Monday | Wednesday | Thursday |
|---|---|---|---|
| **MVP** | | | |
| Cell class | Chensheng, Raven, Haonan | | |
| Ant class | Yueyang, Mingshan | | |
| Map (functions) | Chensheng, Raven | | |
| gui.py & test_gui.py | Haonan | Haonan | |
| Analysis of ant strategy | | Raven,Yueyang, Mingshan | |
| **Improvement** | | | |
| Map class | | Chensheng, Haonan | |
| GUI widget user control | | | Chensheng, Haonan |
| Analyse of strategy | | | Raven,Yueyang, Mingshan |
| Test, readme etc. | | | Raven,Yueyang, Mingshan |

# Demonstration

- Live demo