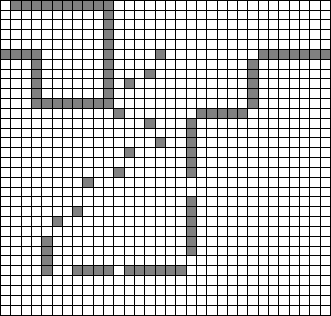
## Introduction:

The implementation should be done in **MATLAB 2020a**. **The codes should be run by pressing MATLAB RUN bottom (without requiring any changes in the codes)**. If your codes consist of more than one .m file, the name of the main file should be “*main.m*”. The output should be two plots. Copy your code (including “*main.m*”, “*simulate\_ant.m*”, “*muir\_world.txt*”) in a folder “P\_ \_firstname”.

In the C2, you will implement a Genetic Algorithm (GA) to solve a specific problem, and show and analyse your results. These are to be done in **MATLAB 2020a**.

# Problem Definition

You will evolve a controller for a simulated ant. Each ant must survive on its own in a world represented by a 2D grid of cells by following trails of food. Each cell in the world either has a piece of food or is empty and the cells wrap-around (so, moving up when in the top row leaves the ant in the bottom row of the grid). Shown below is an environment (called the “John Muir” trail) that consists of a 32 by 32 grid containing 89 food cells (shown in grey).



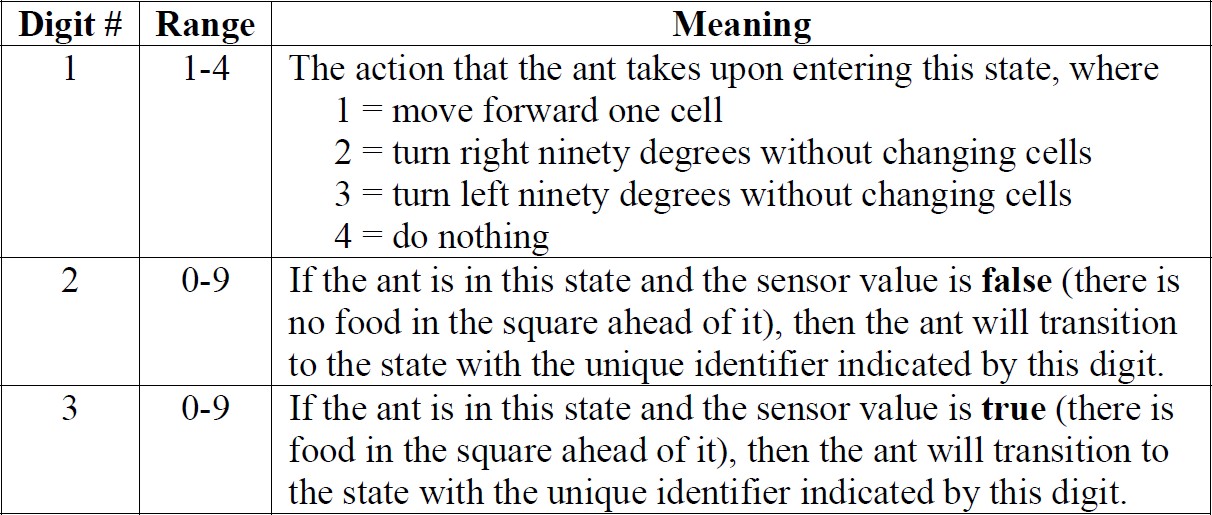
The ant’s position at any point in time can be specified by a cell location and a heading (north, south, east, or west). The ant always starts in the cell in the upper left corner, facing right (east). At the beginning of each time-step it gets one bit of sensory information: whether there is food in the cell in front of the cell it currently occupies (i.e., the cell it would move to if it moved forward). At each time-step it has one of four possible actions. It can move forward one cell; turn right ninety degrees without changing cells; turn left ninety degrees without changing cells; or do nothing. If an ant moves onto a food-cell, it consumes the food and the food disappears; when the ant leaves that cell, the cell is empty. The fitness of the ant is rated by counting how many food elements it

consumes in 200 time-steps. (An ant that consumes 10 cells worth of food receives a fitness score of 10.)

The controller for our ant will consist of a 10-state finite state machine (FSM). At each time step, the ant takes the following actions:

1. Read the sensor value.
2. The controller changes state based on the sensor value/
3. The ant takes an action indicated by the new state (which may result in a change in position).
4. If the ant is in a cell with food, the ant eats the food.

Each of the ten states in the FSM controller has a unique identifier (a number ranging from 0 to 9) and the content of that state can be represented by three digits:



The ant begins its life with the controller in state 0. The entire genetic material for an ant thus consists of 10 states, each of which is represented by three digits, for a total of 30 digits.

# Simulator

We provide you with a simulator function that takes as input the genetic encoding for an ant (i.e. string\_controller containing 30 digits) and the file containing the environment, and will produce as output the trail that the ant takes and an overall fitness value as below:

[fitness, trail] = simulate\_ant(dlmread(‘muir\_world.txt’,' '), string\_controller)

The file called “*simulate\_ant.m*” on Moodle does this. In the file, the variable “*ant\_ori*” specifies the ant’s orientation, and 1, 2, 3, 4 represents east, north, west, south respectively.

There is also a file called “*muir\_world.txt*” containing the Muir world shown above. The numbers in this file describe the 32×32 grid world, where 0 represents no food and 1 represents food.

What you have to do here: Perform an example run of a test ant (with a genetic encoding of your own choosing or the genetic encoding given below) on the Muir world. Try to understand how the simulator works.

[1 1 3 2 7 6 3 1 5 1 8 0 1 7 6 3 4 6 1 1 0 1 5 8 4 4 9 2 9 3]

# Implement Genetic Algorithm

Your main task is to implement with MATLAB a genetic algorithm that attempts to build a better ant through evolution. You cannot use MATLAB's “ga” function, so you have to implement something similar to what you did in the labs in week 6 & week 7. Note that there is no need to change the simulator file (i.e., “*simulate\_ant.m*”).

Your algorithm should make use of appropriate crossover and mutation operators. You will need to make many design decisions on how to implement the algorithm and what parameter values to use.

### Your code should output a plot showing the fitness score of the most-fit ant in each generation using the following code:

hf = figure(1); set(hf,'Color',[1 1 1]); hp = plot(1:Ngen,100\*fitness\_data/89,'r'); set(hp,'LineWidth',2);

axis([0 Ngen 0 100]); grid on; xlabel('Generation number'); ylabel('Ant fitness [%]');

title('Ant fitness as a function of generation');

Ngen is the number of generations and fitness\_data is a 1×Ngen vector of fitness values of the most-fit ant in Ngen generation. One possible plot has shown below.



### Your code should also output a plot showing the trail of the most-fit ant in the final generation using the following codes:

% read the John Moir Trail (world) filename\_world = 'muir\_world.txt'; world\_grid = dlmread(filename\_world,' ');

% display the John Moir Trail (world) world\_grid = rot90(rot90(rot90(world\_grid))); xmax = size(world\_grid,2);

ymax = size(world\_grid,1);

hf = figure(2); set(hf,'Color',[1 1 1]); for y=1:ymax

for x=1:xmax

if(world\_grid(x,y) == 1) h1 = plot(x,y,'sk'); hold on

end

end

end

grid on

% display the fittest Individual trail for k=1:size(trail,1)

h2 = plot(trail(k,2),33-trail(k,1),'\*m'); hold on

end

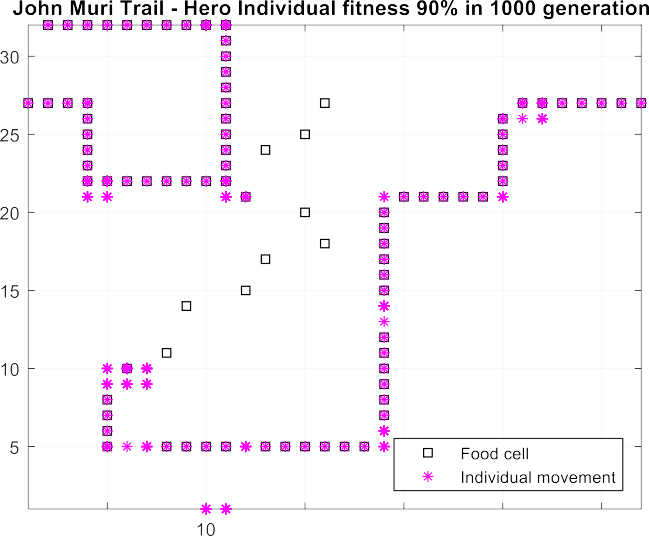
axis([1 32 1 32])

title\_str = sprintf('John Muri Trail - Hero Ant fitness %d%% in %d generation ',uint8(100\*best\_fitness/89), Ngen);

title(title\_str)

lh = legend([h1 h2],'Food cell','Ant movement'); set(lh,'Location','SouthEast');

Ngen is the number of generations and best\_fitness is the fitness value of the most-fit ant in in the final generation. trail is a 200×2 matrix, which is the second output of the *simulate\_ant.m* function for the most-fit ant in in the final generation. One possible plot has shown below.



### Important Notes:

1. You need to implement 3 different selections, 3 appropriate cross-over operators and 3 appropriate mutation operators. Then, try to find the best selection, cross- over and mutation operators by tuning all GA parameters.
2. At the beginning, the code should ask the user for using a default setting (i.e., the setting which you get the best results based on your experiments, for example, Tournament selection, uniform cross-over, swap mutation, etc.) or an optional setting. If the user chooses the default setting, then the code should output 1) the parameters used in the default setting (e.g., selection type, cross-over type, mutation type, number of generations etc.) and 2) the two plots as mentioned above. If the user chooses the optional setting, then the code should ask the user to select what types of selection, cross-over and mutation operators should be used to run the code. The user should also be asked to enter the number of generations. Based on the selected settings, the code should output the two plots as mentioned above.
3. I expect the code runs for around 10 seconds and produces the required outputs