Assignment 3

1. (large) open areas, irregular paths (open areas with lots of isles or straight paths with inlets).
2. Formula for the amount of dropped pheromone:  
   Pheromone is dropped to mark routes that are frequently visited. Shorter routes get more pheromone added because Q is divided by a smaller length . Therefore, over time, high concentrations of pheromone will mark a short route from A to B. However, local optima are possible, so it is not always the shortest route that is marked.
3. Formula for the amount of pheromone on the path from ‘i’ to ‘j’. ‘k’ represents an ant, ‘m’ is the total amount of ants:  
    is the evaporation constant. With every iteration, a factor of the previous amount of pheromone is used in the following amount of pheromone. The evaporation constant is meant to introduce a certain amount of ‘forgetting’ in the algorithm. This will slowly undo pheromone on a route that turns out to be too long.
4. The figure below contains the pseudo code of our main function. Please note that this version of the pseudo code is already the version with ‘special features’, these features will be explained in the next exercise. The ants simulated in this code only drop pheromone on the route they passed when they find a path to the end (unlike ‘real’ ants, which always drop pheromone). This reduces the amount of iterations because only the quickest route found in an iteration is marked.

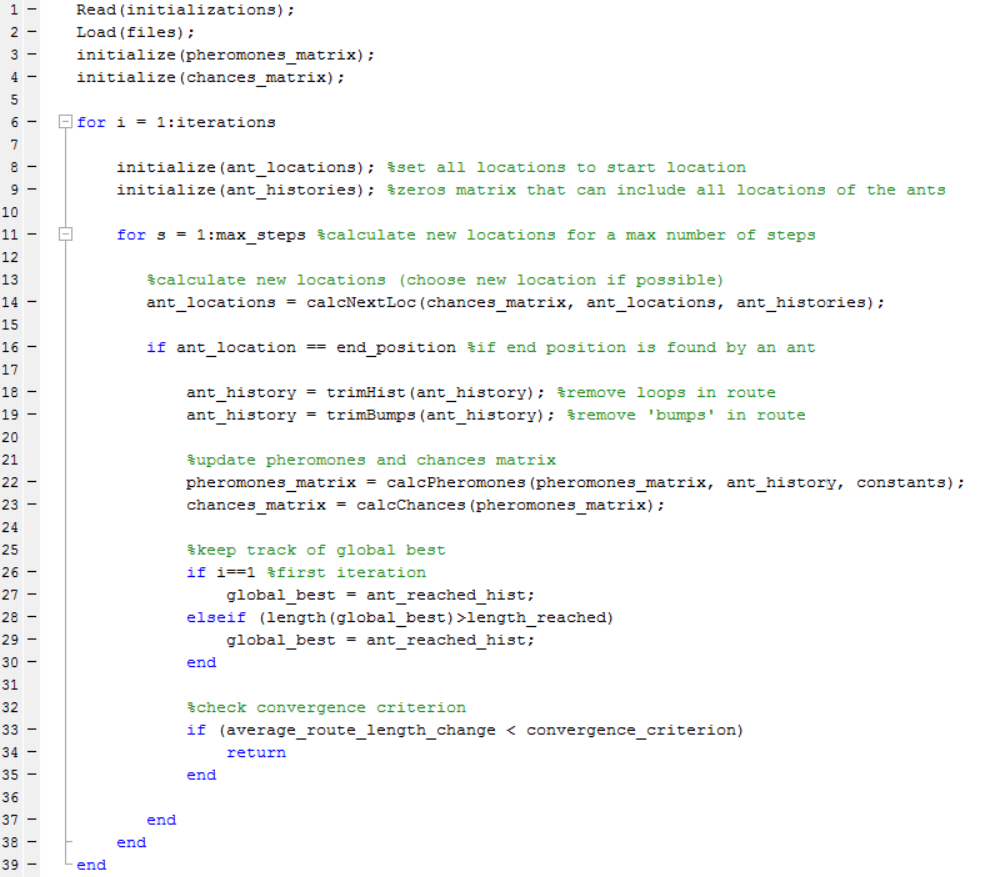
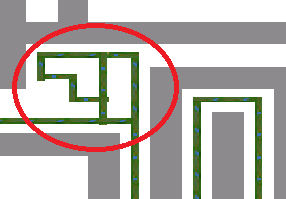


Figure 1: Pseudo-code of the main algorithm

1. The algorithm in the previous exercise was then equipped with three main features.  
   The first involved the *calcNextLoc* function to also look at the history of an ant, and choose randomly (based on the chance of taking that route, of course) only between all new possibilities. If there are no new locations available, the algorithm will simply pick at random (again, based on the chance of picking that route). This feature improves the exploring abilities of the ants, while also enabling them to keep going forward when confronted with long hallways, for example.  
   The second feature was the function *trimHist*, this function is used on the history of an ant that has managed to find a route to the end. This function loops through all the locations an ant has been, starting at the first history element. In every loop, the function checks for the next duplicate of this location, and then deletes all the locations up to and including the found duplicate. This feature removes all loops (see Figure 2), including dead-ends, making the final route of an ant shorter. Quick tests show that the route usually becomes over 60% shorter using this function, with some routes becoming over 90% shorter.  
   The third feature is a function called *trimBumps*. This function is an alternative form of *trimHist*, and looks for the next duplicate of all the neighbouring locations. If a duplicate is found, the locations in between are removed. The result is a much smoother route. Figure 3 shows an example of a bump removed by *trimBumps*.



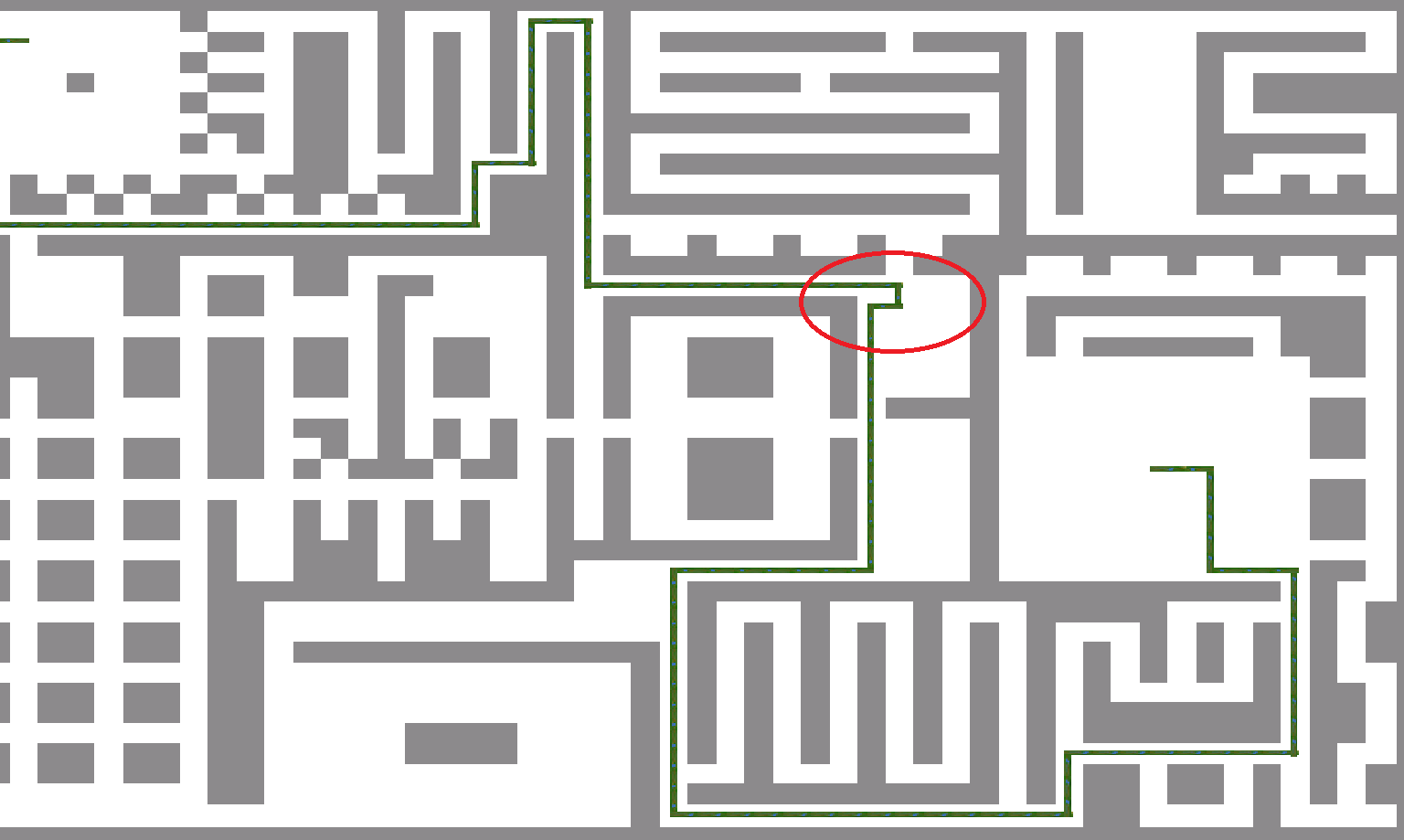


Figure 3: a typical bump removed by *trimBumps*

Figure 2: a typical loop removed by *trimHist*

By including *trimHist* and *trimBumps* in every iteration, the code performance was improved, decreasing calculation times in the medium maze by almost 30% on average. The functions also had a small improvement on the final route length, decreasing the medium routes by roughly 4 steps on average. Performance on the calculation of new locations was not measured, because there was too little time to make and measure a ‘basic’ location picker.

Figure 4: performance increases as result of the trim functions

1. Using rough estimations (10 ants, 100 pheromones, 0.1 evaporation), the code was run on the easy maze first. The results of 5 tests can be found in Table 1. On average, the calculation took less than 2.4 seconds, and every calculation resulted in the shortest route from begin to end, which was 39 steps. Since the algorithm always found the fastest route in a relatively short time, the variables were considered ‘decent’ for this maze, and were therefore not altered any further.

Table 1: results of the easy maze

|  |  |  |  |
| --- | --- | --- | --- |
| Test # | Time (s) | Iterations | Length |
| 1 | 2,850 | 17 | 39 |
| 2 | 1,854 | 13 | 39 |
| 3 | 2,585 | 14 | 39 |
| 4 | 1,993 | 14 | 39 |
| 5 | 2,593 | 18 | 39 |
| AVG | 2,375 | 15 | 39 |

The code was then tested on the medium level maze. Results on this maze can be found in Figure 5, Figure 6 and Figure 7. All measurements are an average of 5 tests. From these results it was concluded that the following variables were considered ‘good’: 10 ants, 300 pheromones per iteration and evaporation 0.05.

Figure 5: performance on the medium maze varying in the amount of ants (300 pheromones, 0.1 evaporation)

Figure 6: performance on the medium maze varying in the amounts of dropped pheromone per iteration (10 ants, 0.1 evaporation)

Figure 7: performance on the medium maze with different evaporation constants (10 ants, 300 pheromones)

The code was then tested on the hard level maze. Results on this maze can be found in Figure 8, Figure 9 and Figure 10. All measurements are an average of 5 tests. From these results it was concluded that the following variables were considered ‘good’: 10 ants, 400 pheromones per iteration and evaporation 0.1. An evaporation constant of 0.2 and 800 pheromones was also considered ‘good’ for faster calculations (These results can be found in Figure 10).

Figure 8: performance on the hard maze varying in number of ants (800 pheromones, 0.1 evaporation)

Figure 9: performance on the hard maze varying in the amounts of dropped pheromone per iteration (20 ants, 0.1 evaporation)

Figure 10: performance on the hard maze with different evaporation constants (20 ants, 800 pheromones)