**ARTIFICIAL INTELLIGENCE FUNDAMENTALS**

**PROJECT INTRODUCTION:**

Finding the optimal tower defense path  
using the Ant Colony Optimization Algorithm

**15.03.2017**

**Ksawery Jasieński & Victor Fontaine**

**Contents:**

1. Description of problem
2. Analysis of the problem
3. Analysis of existing solutions
4. Description of preferred solution
5. Possible implementation problems
6. Expected simulation results



Responsibilities :

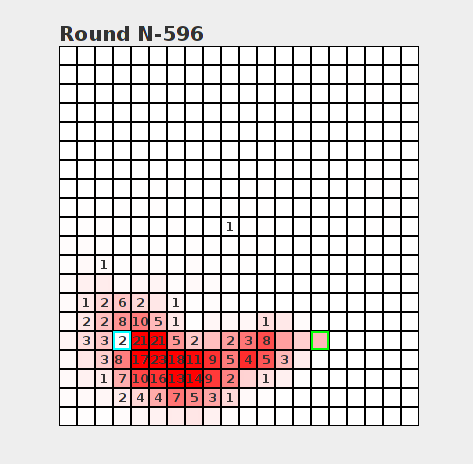
|  |  |
| --- | --- |
|  |  |
| Ksawery Jasieński | * Graphical User Interface of the application * Description of the problem * Possible implementation problems * Expected simulation results |
| Victor Fontaine | * Description of preferred solution * Implementation of the algorithm * Simulation results from application * An Analysis of the problem |

1. Description of problem

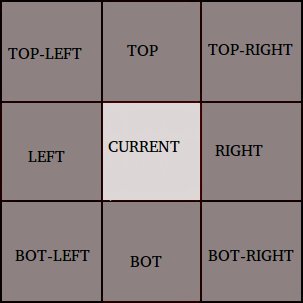
This project is set in a setting resembling a tower defense game. In tower defense (TD) games, the goal of the player is to “defend a player's territories or possessions by obstructing the enemy attackers, usually achieved by placing defensive structures on or along their path of attack”[[1]](#footnote-2). Defensive structures - usually referred to as *towers* - automatically shoot down the incoming attackers - typically called *creeps*, or *minions* - in order to stop them from reaching their destination. The path of attack taken by the attackers can be either predefined by the creator of the game, or it can be computed by the game itself. In the latter case, the path is usually individually chosen for each creep so that the distance that it has to travel is minimised.

In the case of this project, we will take the sides of both the attackers and the defenders: firstly by placing the defensive towers, and then secondly, and most importantly, finding the most optimal path that the attackers should take in order to reach their destination unscathed, and as quickly as possible. The problem given in this project is therefore more advanced than in most TD games, as in there the creeps usually don’t bother taking a safe path - they only care about reaching their destination as fast as possible. The task of finding the shortest possible path leading from the starting point to the end point while avoiding all towers for the purpose of this project will be performed by the Ant Colony Optimization Algorithm. Hence, the attackers in the context of this application will be called *ants*.

The simulation discussed in this project will take place on a rectangular grid. The user will be able to chose the positions of towers and the starting position of the ants.



Analysis of the problem



Every steps, the ant have to choose the next case it wants to go at the next step. For this, it’s going to choose one of the five cases (dark-grey on the picture) around the current one (light grey). It can not go back, so it can not go to the left, bottom-left or top-left. To choose the next one, the algorithm is going to check numbers of pheromones on the five possible next case, and then, generate a probability that the ant goes to the i-th case in the next step, for every possible next case. If we denote the number of pheromones in the i-th case by , then the probability will be: . Now the ant is going to “choose” the i-th case with probability .

Moreover, the algorithm is also going to create some “crazy ants” (maybe one crazy ant for 10 normal ants) which are not going to follow the pheromones but who are going to choose the next case completely randomly (with a probability for every next possible case), like this, we can find others paths, may be shorter, to reach the stop case. If we don’t do that, when we find a path, next ants are all going to choose this path, and we will not find another path, and probably don’t find the shortest path.

Every time an ant is reaching the stop case, the algorithm is going to update every case the ant was traveling to increase the number of pheromones, in regards with the length of the path the ant took. For example, let say that every ants have 1000 pheromones to put on the path. If an ant reach the stop case with a path of 100 cases, then on each case the algorithm is going to add pheromones. In general, let say that each ants have pheromones, and the length of the path is , then on each case where the ant was traveling, the algorithm is going to add pheromones.

1. Analysis of existing solutions

1. Description of preferred solution

First of all, some ants go through the board (a graph) starting where the user choose previously. They travel randomly, choosing the next case with a probability equal to 1/5 at the beginning (for each case, there is 5 possible next cases), to find the stop case. When one of the ants find it, some pheromones are deposed on the path the ant used (according to the length of the path because we are looking for the shortest one).

Then, next ants starting from the beginning choose the next case (at every iterations) with a different probability than previously because the probability is now depending on the number of pheromones on the next case, so they are going to choose the same case as their “ancestors” with a higher probability than the other cases.

If the ant is killed during its travel, there is no pheromones deposed on the path to avoid that next ants died systematically. We also use some “crazy ants” who don't care about the pheromones to find possible new paths.

Finally, the shortest path should be the one with the biggest number of pheromones, and almost all ants should take this path to reach the final case without dying.

1. Possible implementation problems

As things stand, we do not foresee many sizeable roadblocks that could considerably impact the development of our application. Among things to consider will be, definitely, the values of the parameters which will be used to run the Ant Colony algorithm: the number of iterations to be performed, the speed of decay of the pheromones, the probability values which will be assigned to different nodes etc.

Furthermore, it might so happen that due to the positioning of the towers selected by the user, there could be no path for ants to take that would allow them to reach the right sight of the board. Proper handling of this circumstance will have to be implemented so as to avoid having the algorithm run indefinitely.

As it often happens during development of a project, unexpected problems may arise at any time. We will seek to valiantly deal with those through flexibility, inspired by modern software development methodologies such as Agile.

1. Expected simulation results

Due to the lack of already completed implementations of this problem, there does not exist a handy benchmark for us to use for evaluating the efficiency of our solution. We will, however, try to compare the speed at which our implementation of the Ant Colony algorithm converges towards a reasonable solution with a brute force solution, which should provide a solution after a much longer period of time than the Ant Colony algorithm.

1. After Wikipedia: <https://en.wikipedia.org/wiki/Tower_defense> [↑](#footnote-ref-2)