AI challenge

Antonio Carlos Salzvedel Furtado Junior and Luciano Luz

December 5, 2012

University of Alberta
CMPUT 350 L01 Advanced Game Programming
For: Michael Buro

---

1 Introduction

In our project, our goal was to improve an existing bot from the AI challenge[1]. We started by searching open-source bots developed in C++. We prioritized bot that had a readable source code and were ranked reasonably well. This task took more time than we hoped, since most of the source code produced for the challenge had no comments, and was not really readable. We decided to choose a bot called NotGreat[2] to work on. It was ranked 126th in last year's competition.

1.1 Bot overview

The way NotGreat controls the actions of the ants can be basically split in two modes of operation, which are run at each turn of the game. The first mode deals with battles among ants. As battles have a higher priority, this mode is the first to be run at each turn. The second mode is more general, as it is responsible for all the other types of actions, such as food gathering, map exploration and hill hazing. At the following subsections, we aim to give a brief explanation of how these modes work.

---

[1] http://aichallenge.org/

[2] http://aichallenge.org/profile.php?user=12710

1.1.1 Battle mode

The first task of the battle mode is to divide ants between fighting groups and non-fighting ants. The fighting groups are determined by proximity. NotGreat defines an engagement radius, called ENG_RAD. If the distance between an enemy ant and one of our ants less is or equal than ENG_RAD, these ants can be included in the same fighting group. This is true, unless one of these ants already belong to another fighting group, as ants cannot belong to multiple groups.

Once the groups are defined, the bot tries to decide the best movement plan for its ants. It has two algorithms for this task, one for small battles (i.e. for fighting groups with less than 7 ants) and large battles. For small battles, NotGreat uses a minimax search. This algorithm search through all the possible move plans. It considers the survival of an ally to be more valuable than the death of an enemy. One of the disadvantages of this algorithm is that it has a large computational cost. For fighting groups with more than 6 ants, we need to use another algorithm, as the number of movement plans grows exponentially.

The algorithm used for large battles takes a different approach. It basically checks the number of ants in the fighting group. If we have more ants than the enemy, we attack them. In this case, each ally ant is assigned to attack its closest enemy. Otherwise, our group retreats. Even though this algorithm is much faster than the previous one, it is less efficient.

1.1.2 General mode

At each turn, if an ant is not considered a fighter, its move will be decided through diffusion. In opposite to path finding approaches, diffusion is based on an influence map. Thus, the ants don't know why they are following a specific direction, they are guided by a "scent". The scent is created as follows. First, we assign higher values to specific points on the influence map, such as food, enemy hills, etc. Later, we run the diffusion algorithm, which will spread these values to close positions on the map. These values will be used to guide the ants on the map, as they should move to positions with a higher value.

2 Modifications

In our project, two major modifications were performed. Regarding the general mode, we created a hill defense system. On the battle code, we improved the algorithm used for small battles. We explain these modifications in more details in the following subsections.

2.1 Hill Defense

One of the weaknesses of NotGreat is not having any sort of hill defense system. As soon as an ant is born, it is guided by scents that take her away from the hill. Thus leaving the hill unprotected. Sometimes, even one ant can take over one of our hills.

Considering this problem, we decided to develop a hill defense system. In order to do that, we had to create a new scent, and a new type of ant. This ant is guided by the two types of scent. We set this guardian scent to be slightly stronger than the other one, so the ant will keep itself close to the hill most of the times. The strongest values for the guardian scent are within a few steps of our hills. We set a percentage of our ants to be guardians. We choose the guardian ants at each turn, picking ants that are close to our hill.

Even though this system avoids our hills to be taken over by few ants, this causes a impact on exploration and food gathering. As we keep some ants close to our hill, they are unable to search for food or enemy hills. To minimize this impact, a reasonable percentage of guardians must be chosen.

2.2 Battle Code

As mentioned before, NotGreat has one algorithm for small battles, and another for large battles. We noticed that the code used for small battles could be improved. NotGreat simply implements Minimax for small battles. However not cuts are performed on the recursion. As seen in class, we decided to implement the alpha-beta pruning to optimize our results. Our goal was to increase the size of fighting groups the small battle algorithm could deal with. After the implementation, we were able to immediately deal with fighting groups involving up to 8 ants.

There was another further modification performed in order to use bigger fighting groups with the small battle algorithm. Instead of simply looking at the number of ants in each fighting group, we tried to estimate the number of movement plans that could be generated for these ants. We noticed that for some cases, even though there were a lot of ants involved in one battle, the region of the map they were fighting was not open. Therefore not many movement plans were possible. The time taken to run the alpha-beta code depends more on the number of plans involved than just the number of ants. So, if we estimate that the number of plans is smaller than a determined limit, we still use the minimax search with alpha-beta pruning.

3 Conclusion

In order to prove the efficiency of our hill defense system implementation, we ran our bot against the original one. We also ran it against multiple bot types in other maps. Our bot was superior in most of the cases, when compared to the original. We were able to stand against small attacks to our hills much better.

Regarding the alpha-beta implementation, our biggest challenge was to determine how far extend the small battle code. We cannot predict how much of the recursion will be cut by alpha-beta. The average runtime for the battle code was much faster than the original. For example, we measured the average and the maximum runtime for the battle code, and we compared it to the original. Considering that we are dealing with the same fighting group sizes, we achieved the following results:

- Minimax:
  - Average: $71.71ms$
  - Maximum: $519.93ms$
- Alpha-beta pruning:
  - Average: $17.19ms$
  - Maximum: $108.32ms$

Even though we achieved better results, the main problem for the AI challenge is not the average time you can run the battle code. If you cannot accomplish the minimum acceptable time every turn, then your bot is removed from the game, and you are going to lose eventually. We had many problems with timeouts, when uploading our code to the server provided to us in the course. We adjusted some parameters and it seems that the problem is solved.