**AI Challenge 2011 (Ants) post mortem by xathis**

I can’t believe [i won](http://ants.aichallenge.org/).  
I can’t believe i won *decisively* at all.



This is a writeup of the bot i created for the Google-sponsored [AI Challenge 2011](http://ants.aichallenge.org/). I really love programming contests and this was by far the biggest and most interesting one for me.

I want to thank the contest organizers and contributers, tcp server hosts, tool designers and all participants for the amazing previous months.

Here’s a sample game

**TypeError**

|  |  |
| --- | --- |
| **message** | Object doesn't support this property or method |
| **number** | -2146827850 |
| **description** | Object doesn't support this property or method |

In this writeup i want to explain what my code does and how it achieves that. It’s somewhat technical and not so much about strategy in general, but i try to keep it simple.

**about my code**

I started by downloading the [java starter package](https://github.com/aichallenge/aichallenge/tree/epsilon/ants/dist/starter_bots/java), renamed and cleaned up most of the files, created a new Strategy class and soon realized that it would be nice to have an extra class representing an ant.

I put all of my code into the Strategy class that ended up at about 1800 lines of code. Also important are Ant.java and Tile.java both of which are mainly value objects containing lots of variables.

**general**

I kept the design of the starter bot to treat all unknown positions as land. I never calculated the vision of my bot so i actually did not know which tiles were land and which were just unknown.

For each square on the map, exactly one Tile is instantiated before the first turn happened, these instances are kept throughout the rest of the game. Ant objects are created at the beginning of each turn for my ants as well as enemy ants and have a lifespan of one turn only.

If you want to have a look at the java code yourself, be warned, it’is a dirty mess i'm not exactly proud of, but i guess you want to see it anyway, so here's [my code on github](https://github.com/xathis/AI-Challenge-2011-bot).

**Strategy implementation**

There’s been a lot of talking about *overall* strategies. Unfortunately, i don’t really have one. I do not make decisions based on the number of ants i have or the size of my territory, my bot does not play different when it’s losing or winning, it does not even know that. I also never look which turn it is, in the first turn everything is done exactly the same as in the 999th turn. I treat all enemies the same, even in combat situations and i don’t save any hill locations.

Other than moving ants away from my hills via missions, every move i make depends entirely on the local environment of the ant.

So, what exactly is my bot doing each turn? After some initializing and precalculation stuff it calls these methods in the following order:

* initMissions();
* enemyHills();
* food();
* initExplore();
* createAreas();
* fight();
* defence();
* approachEnemies();
* attackDetachedEnemies();
* escapeEnemies();
* distribute(true);
* explore();
* doMissions();
* createMissions();
* distribute(false);
* cleanAreas();

I will try to explain what these methods are doing later, most of them move ants that are meeting specific criterias. Once an ant is moved, the move is instantly send to the game engine through system output, so it cannot be undone in a method called later.

**breadth-first searches**

This is not about one of the methods, it’s about my breadth-first searches in general. To speed up searches through the map, each tile had references to all adjacent tiles excluding water tiles. This way i never needed to access the elements of my 2-dimensional map array, so i didn’t have to use at the row and col properties. A typical breadth-first search (and i used TONS of them) looks like this:

LinkedList<Tile> openList = new LinkedList<Tile>();

LinkedList<Tile> changedTiles = new LinkedList<Tile>();

openList.add(foodTile);

foodTile.dist = 0;

foodTile.isReached = true;

changedTiles.add(foodTile);

while (!openList.isEmpty()) {

    Tile tile = openList.removeFirst();

    if (tile.dist >= 10) break;

    for (Tile n : tile.neighbors) {

            if (n.isReached) continue;

            if (n.type == Type.MY\_ANT) {

                    // found one of my ants at the tile n

            }

            n.isReached = true;

            n.dist = tile.dist + 1;

            changedTiles.add(n);

            openList.add(n);

    }

}

for (Tile tile : changedTiles) tile.isReached = false;

in this example we start at the Tile called foodTile and look for my ants that are reachable within 10 steps.

It would probably be more efficient to use HashMaps to keep track of the tiles reached already, because the changedTiles loop at the end could be left out.

**turn initialization**

At the beginning of each turn I precalculated some relations between ants like the number of close own ants, a list of close enemy ants and their distances. It’s partly confusing since these values are measured in a mixture of Manhatten distances, Euclidean distances and bfs-step distances.

Depending on enemy distances i had a flag indicating whether my ants are dangered or indirectly dangered, for enemies i look if they are detached, meaing they have no other enemies close by.

Another part of the turn initialization was to recognize which enemies stayed on the same position for multiple turns, if they did for more than 5 turns i supposed they would not move this turn as well.

**exploring and missions**

Sending my ants away from own hills, exploring new land and keeping all tiles visible is closely related to eachother in my implementation. This is probably most significant for the total behaviour of my bot.

**exploring**

Every tile on the map has an exploreValue that is increased by one at the beginning of each turn, unless it is reachable within 10 steps by one of my ants (determined via bfs), in that case the exploreValue is resetted to zero. So the exploreValue is an indication of how long the tile has been out of my reach.

Now in the explore method each ant that was not used for another task has a breadth-first search started up to a dist of 11 steps. The exploreValue of tiles reached within the first 10 steps have to be zero, we only care about the last step. If every tile reachable in the 11th step has an exploreValue of zero as well, we know that the ant is surrounded by other own ants or water, otherwise i move the ant in the direction that has the biggest sum of exploreValues on the 11th step (meaning the sum of all exploreValues reachable from the tile i moved to in 10 steps). Because tiles can be reached in exactly 11 steps by up to two different first moves, every tile reached during the bfs has a list containing one or two initial moves which are passed to following tiles.

That’s how i explore new land and keep all tiles visible.

**missions**

So, what to do if an ant is surrounded by other own ants? Move it to the border! What exactly the border is was determined in the createAreas method, more on that in a second. Moving to the border is what i call missions, a mission consists of two things only: an ant and a target destination on the border. Missions are one of the very few things that i keep alive in between my turns, but the path to the destination is recalculated every time using A\*. If an ant already has a mission it continues that mission, otherwise a new mission is created. If the ant has a mission but was used for a different task (food, fighting, defending) the mission is discarded and removed.

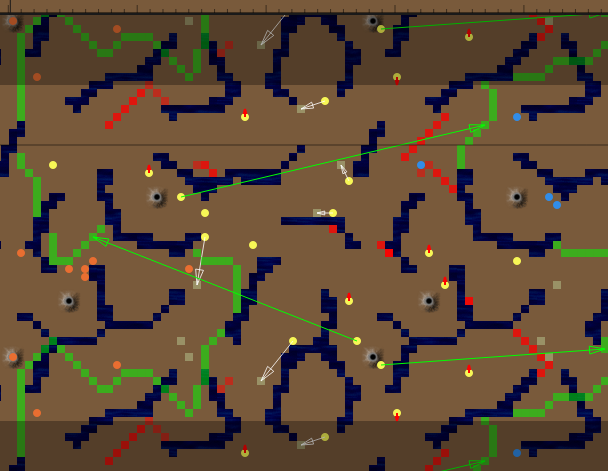
**areas and border**

The createAreas method seperated different areas on the map. I start yet another bfs, this time from all ants including enemies simultaneously. When starting each ant has its own area to which all tiles reached in the search were added. But when two of these areas by the same player collide, they are merged into one new big area, which could later be merged with other areas again. After finishing i loop through all tiles in my own areas and add them to a list of border tiles if at least one of their adjacent tiles is not in the same area. In this area bfs i look at tiles reachable in up to 20 steps, this way two ants with a distance of 39 steps can still be in the same area even if there is fog of war in between. But it assures that there are no border tiles popping up in an area that clearly belongs to me. So border tiles are either tiles that have the same distance to one of my ants and an enemy ant or they are 20 steps away from my exploring ants without enemies in sight. In the latter case they could actually be on a water tile i had not seen yet, in that case i would just remove the mission once i discovered that.

When creating a new mission, we need to choose one of the border tiles as the target destination. If the ant is not on one of my own hills, it chooses the border tile reachable in the lowest amount of steps, looking for it by a bfs. Otherwise if the ant was just spawned sitting on my hill, the target border tile is determined using some complex calculation considering the ratio of own/enemy ants near that border position, the possible distance to enemy hills, how many ants are already on a mission to a near border tile and how long they do need to reach their desitnation... Oh, I'm kidding, it was random. Completely random. Like in

target = area.border.get(turnRandom.nextInt(area.border.size()));

That was my first implementation and it worked out really well so i never bothered to change it. Notice that the overall behaviour was not completely random as once interrupted by another task, when creating a new mission, the target was the closest border tile from the ant, even when the ant was close to one of my hills. Since the border tiles are different ones every turn, missions need to be updated. Therefore they start a bfs from their current target searching for the nearest border tile, this is done every turn if we’re good on time but at least once every 10 turns.

Here are two visualization of everything i just explained. Green tiles are borders, green arrows are missions. Red tiles are not reachable within 10 steps, the little red arrows are explore moves, white arrows are for food.  
  
  


**misc**

One could argue that *reachable in 10 steps* is not exactly the same as *within a viewradius of 77*, but they are pretty close and i never encountered any problems.

Often single ants keep moving back and fourth for a long time. For example if an ant is alone in a cave surrounded by water with one way out. Then there is a situation in that every second turn the ant has to move one step into the cave because it has to explore a tile with a step distance of 11 at the end of the cave, but every other second turn the ant has nothing to explore thus gets assigned a mission to the border, moving one step in direction of the cave exit only to discard this mission the following turn moving back in again. This isn’t a big deal though as creating a mission doesn’t cost much time.

**food**

Nothing really interesting here. I search from all foods via bfs and make the first ant found move into the food direction. I do *not* save food locations and i do *not* consider a single one ant collecting two foods close to eachother.

I do have some special logic to deal with possible enemies near that food though. Normally i try to avoid one on one ant exchanges, but if i have another ant near the food i will sacrifice the closer ant if the opponent choses to approach the food as well.

McLeopold wrote a nice [post in the forums](http://forums.aichallenge.org/viewtopic.php?f=24&t=2210) explaining the way he does it.

**enemy hills**

Very similar to food collection, done using a breadth-first search from the enemy hills. I send up to 4 ants within a dist of 20 steps to the hill. If i have a total number of 10 or less ants i send a single ant to prevent problems in the first few turns on maps with opponent hills very close to eachother. I do never sent more than 4 ants directly to a hill, but because there usually are enemy ants around and i’m very offensive a lot of my ants will most likely go there anyway.

**combat**

Now it gets interesting! Definitely the most-discussed topic was the combat logic.

My approach to combat is to divide fighting ants into groups and then look what could happen one turn ahead. I do this by simulating all (not quite all since version 2, more on that later) possible moves my ants could make and for each possible combination of moves look at all possible moves enemy ants could make, then evaluate the situation and take the moves that result in the best case for me, supposing my enemy knows the moves i take and moves in the way that is the worst for me.

This is quite similar to the [minimax](http://en.wikipedia.org/wiki/Minimax) or [alpha-beta pruning](http://en.wikipedia.org/wiki/Alpha-beta_pruning) algorithm, it just needs to have multiple max nodes successively (one for each of my ants) and multiple min nodes for enemies.

**example**

Let’s consider this simple situation as an example. The green ones (a,b,c) are my ants and my enemies (x,y) are blue.

% % % % % %

% % a % % %

% % . b % %

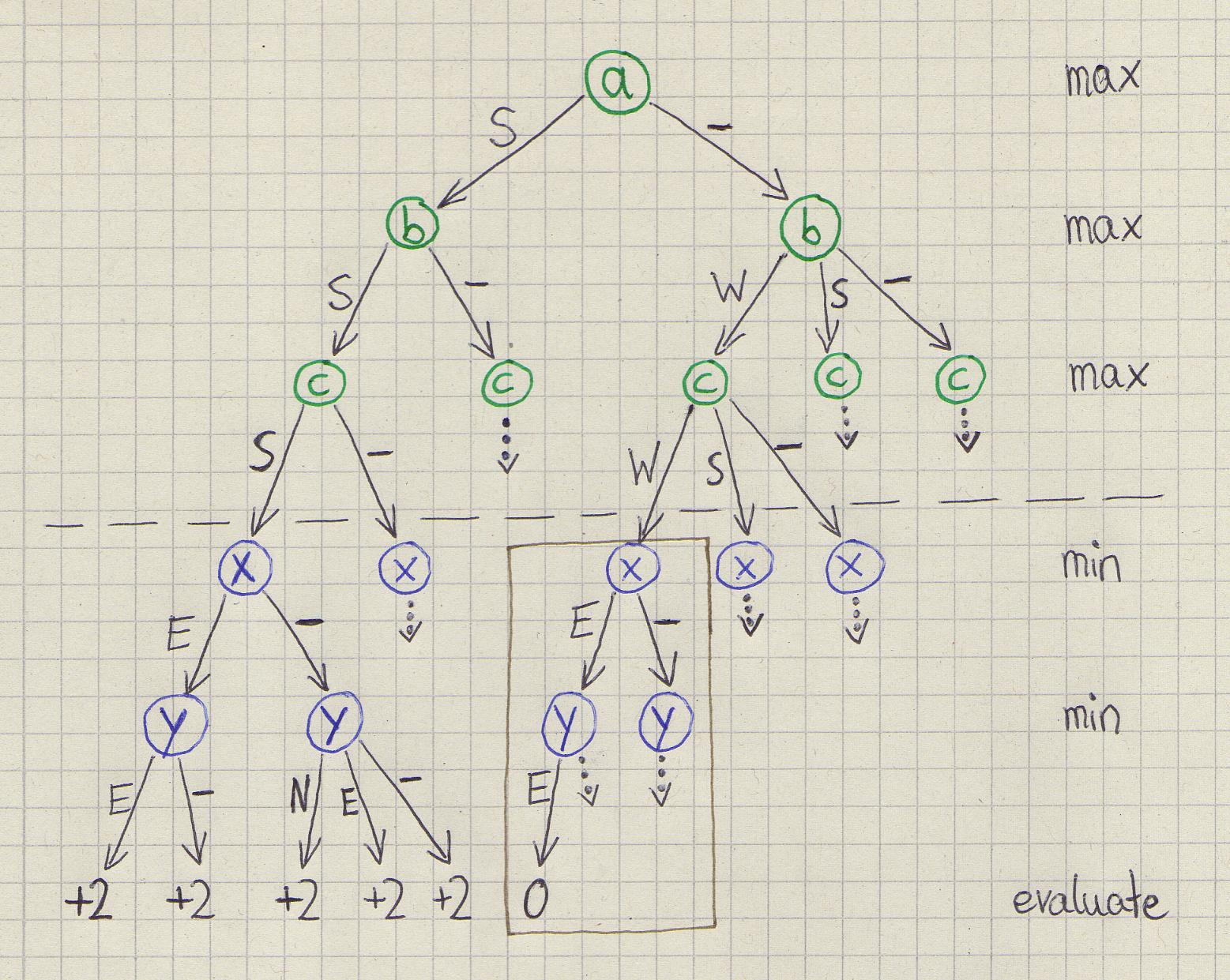
% % . . c %

% x . . . %

% % y . . %

% % % % % %

I took the time to draw you a nice graph visualization to explain what’s going on.



N/E/S/W means moving North/East/South/West and “-” means staying at the same position.

The tree is generated on the fly using recursive depth-first search.

We start by looking at a possible move for a, simulate that move and then continue with the next ant b. After the last enemy has moved the situation is evaluated, here we’re using the simple evaluation function numDeadEnemyAnts - numDeadOwnAnts. The enemy nodes (min) always take the value of the move resulting in the smallest value, our nodes (max) choose the highest value. If we move a, b and c south (left subtree) then both enemies will die regardless of where they move, so the value is +2. But if we let a stay and move b and c west then both enemies can go east which results in a value of 2-2=0. This is very nice, because now we can cut, meaning we don’t need to look at anything else in the brown box. We can cut since we already know there is a possible move combination resulting in a value of 2, but as our opponent chooses the minimal value, the brown box can only result in a value smaller or equal to 0.

As you can see i left out some subtrees, also the numbers of possible moves is very limited and we have 5 ants only, so in reality it would end up growing much larger, exponential with the number of ants.

The actual implementation pseudocoded looks something like this:

List myAnts

List enemyAnts

bestValue = -Infinity

void max(antIndex) {

        if antIndex < myAnts.size

                myAnt = myAnts[antIndex]

                for each possible move of myAnt

                        simulate move

                        max(antIndex+1)

                        undo move

else

        value = min(0)

        if value > bestValue

                bestValue = value

                save the current simulated moves of all my ants

}

int min(antIndex) {

        if antIndex < enemyAnts.size

minValue = +Infinity

                enemyAnt = enemyAnts[antIndex]

                for each possible move of enemyAnt

                        simulate move

                        value = min(antIndex+1)

undo move

                        if value < bestValue

return -Infinity  // cut!

                        if value < minValue

                                minValue = value

                return minValue

else

        return evaluate()

}

The lists myAnts and enemyAnts had to be filled, thereafter we call max(0) and now the moves of my ants resulting in the best value are saved.

**optimizations**

Two neighboring ants both staying at the same tile is exactly the same as both swapping positions with eachother, so the letter can be ignored. To do this, i save the position each ant has before combat and then ignore a move when coming across a possible move from tile A to tile B and there has been an ant on tile B that has currently simulated a move to tile A.

In my version 1 bot i used this algorithm with a fixed number of up to 7 ants, groups of 8 ants sometimes needed a few hundret milliseconds, too much as there could be multiple fights at once.

**move check lists**

In version 2 i experimented with other crazy optimizations, allowing me to look at larger groups in a lot of situations. Let’s have a look at another example, now without water tiles, so there are 5 possible moves for every ant (ignoring cases in which two ants move to the same field).

. . . . . .

. . a . . .

. . . b . .

. . . . c .

. . . . . .

. x y . . .

. . . . . .

Now what can happen with our ant a? If it moves south, it could get into a battle with the enemies, but if it stays were it is it can’t get into trouble. We definitely have to consider both of these moves, because we don’t know yet if attacking is worth it or if we would risk loosing ants. But what about a moving north, east or west? There is no possibility it can get into combat this way, so our evaluation value will be the exact same as for staying on the same tile, which means that looking at those moves is irrelevant. If b and c stay, they can get into battle by y moving north, so they are dangered much more. We could use that for each of b and c moving north results in the same evaluation as moving east, but i did not consider that in my implementation. So for each of our own ants we can precompute a list of move positions that need to be checked, this requires lots of nested loops since we have to look at every adjacent tile of every enemy near every own ant in the combat group.

For enemy ants we can do something similar, but in a different way. Consider a stays and b and c move north. x and y are now completely out of reach, so every move they make results in the same value as staying there. If b stays and only c moves north instead, y has to check the north move as well as staying, because moving north brings it into battle distance with b.

This means that for an enemy to know if a move needs to be looked at is dependent on whether there is one of my ants at a specific tile or not. So i have a dependency table for each enemy that mapped one tile (where my own ant could be) to a list of move tiles for the enemy to check. These tables are created in even more nested loops...

I hope you like examples because here’s another one, demonstration a dependency table, this time with row and column numbers, it’s (row/col).

  0 1 2 3 4 5

0 . . . . . .

1 . . . a . .

2 . . . . b .

3 . . . . . .

4 . x . . . .

5 . . . . . .

If there is an ant at (1/2) x has to check (3/1). If there’s an ant at (2/3), x has to check (3/1) *and* (4/2). For (3/4) it has to check (4/2). If there are ants on (1/2) and (3/4), it has to check both (3/1) and (4/2) of course.

So this is the table for x

(1/2) => [ (3/1) ]

(2/3) => [ (3/1), (4/2) ]

(3/1) => [ (4/2) ]

These check lists and dependency tables greatly reduce the branching factor of our tree. The bad thing is that it’s not easy to estimate how long the calculation might take now. In my bot i do not choose combat groups by limiting the number of ants, but by setting a limit for the number of own moves to look at and the number of enemy ants.

When choosing the combat groups i select the first of my ants arbitrary and add all enemy ants that could attack my ant in the next turn if both my ant and the enemy ant moved. Then i look if one of these enemies has a different ant of me in a dangrous distance. If there are multiple of my ants found this way, i first added the ones closest to my first-selected ant, for each ant of me adding possible new enemies again, until i could add no more own ants because the limit was reached or there were none. I have to assure that for each own ant, every dangerous enemy ant is also in its combat group. In other words, every own ant can only be in a single combat group, but enemies have to be in multiple combat groups if required.

**evaluation**

My evaluation function does not only consider the number of dead ants but also the distance between my ants and enemies. Here it is:

if (beAgressive) return enemyDeadCount \* 300 - myDeadCount \* 180 - distValue;

else return enemyDeadCount \* 512 - myDeadCount \* (512+256) - distValue;

If the beAgressive flag is set to true dead enemies are worth 300 and dead own ants are worth a negative180. So i will sacrifice one of my ants to kill an enemy, i also will sacrifice three ants to kill two enemies, but i will not give up 2 of mine for one enemy only.

When not agressive the bot doesn’t do 1 on 1 exchanges, it likes loosing 2 own ants for 3 enemies though.

The distValue is the sum of the distances each of my ant has to its closest enemy ant, we need to subtract it since small distances are better than big ones.

So when was beAgressive set to true? It has nothing to do with overall ant count or positional advantage, it just depends on the number of own ants that are close to the fight. To be precise it is set to true if the maximum of the number of close own ants of any of my ants in the fight (not only in the combat group, but in the whole fight) was equal to or greater than 14. Why 14? Mostly because 42 is divisible by 14!

**approaching enemies**

The approachEnemies method is called right after fighting and sends those of my ants that are near enemy ants in their direction. For each of those own ants i first start an A\* algorithm that only looks at free tiles (it does not seach through tiles occupied by other own ants) and has a step limit of three times the Manhattan distance to its closest enemy. If there is a path to its closest anemy found this way, the first move of that path is taken, this was the reason my bot often formed lines when fighting, because if there was a free hole in the front line, some ant from the second line went there.

If no path was found this way, i start another A\* algorithm, also looking at occupied tiles this time, so my ant gets close to the enemy even though there are others of my ants fighting in between so it can’t get through right away.

Then there also is the attackDetachedEnemies method, a leftover from the beta phase, which made my ants approach those enemies first that had no other enemies close by.

**defence**

My version 1 had no defence at all, but with version 2, after the multi-hill maps became more popular i introduced some defence mechanisms. I only defend any of my hills if i have not more than 4 hills, i don’t really know if this makes sense but i don’t like having too many of them. I search from my hills for close enemies using - you guessed it - a breadth-first search! I then order them by ascending distance to my hill and try to find a single defender ant for every close enemy. I first look for the defender ant close to the path the enemy has to take to get to my hill. This path is easy to get, because i can just run back all the previous-tiles from the enemy position, which are saved in the bfs. When unsuccessfull i start a new bfs searching for a the closest of my ants from the tile which is on the same path on ¼ of the enemy dist. The defender is send to the tile of the path that has ½ of the enemy dist via A\*. I do not check if the move is safe, so one on one ant exchanges are possible, which is clearly better than losing the hill.

**escaping enemies**

I try to avoid one on one ant exchanges in general, so when a single of my ants encounteres one or more enemy ants it escapes them. I look at all possible moves for my ant and triy to find a safe move. If not available, i try to make a move that could at least kill an enemy ant if my opponent chooses to move in my direction.

Most of the time there are multiple safe moves possible, so to decide which one of those to take i introduced some new logic that was one of the few changes for the last-minute update for version 3. Before this change my ants often escaped into small caves where they were easy targets, escpecially in the maze maps with their jagged walls. I prevented this by starting a bfs up to an ESCAPE\_CHECK\_DIST of 8 from my ant and summed up a value for each possible safe move, close tiles had a greater weight, own ants are good and enemies are bad:

int addValue = (ESCAPE\_CHECK\_DIST+1-tile.dist);

if (tile.type == Type.MY\_ANT) addValue \*= 3;

else if (tile.type.isEnemy()) addValue \*= -3;

**distributing**

This is mostly a leftover from the beta phase i couldn’t get rid of. In the beta when there were no hills fighting wasn’t that important and everyone tried to keep as many tiles reachable as quick as possible, to be the first who gets new food.

I start breadth-first searches from my ant and all close ants for each possible moves, summing up values and then looking for the move that maximizes the number of reachable tiles in the smallest number of steps.

In my final bot this was used rarely, only for single ants near enemies in some situations.

**that’s it!**

I’m sure there are lot of small features i have not explained exactly but you should get the picture of how my bot worked now. I hope you got something out of reading this and i’m sorry i couldn’t present even cooler algorithms.

I’ll keep hanging around in irc and you’ll most likely see me in the next AI Challenge, though i’m not sure if i can spend this much time again.

**meta**

**discuss**

on [the aichallenge.org forums](http://forums.aichallenge.org/viewtopic.php?f=24&t=2225)

or [reddit](http://www.reddit.com/r/programming/comments/nooxo/winner_of_the_2011_ai_challenge_explains_his_code/)

or [Hacker News](https://news.ycombinator.com/item?id=3387959)

**other interesting writeups**

[FlagCapper](http://flagcapper.com/?c2)

[Truusje](http://www.corniel.nl/blog/2011/12/24/truusje-my-lovely-ants/)

[MBCook](http://www.foobarsoft.com/2011/12/second-best-clojure-bot-postmortem/)

[Parasprites](http://forums.aichallenge.org/viewtopic.php?f=24&t=2169)

[delineate](http://www.decompilinglife.com/post/14396418230/2011-google-ai-challenge-ants)

[Daarhuuk](http://forums.aichallenge.org/viewtopic.php?f=24&t=2191)

[runevision](http://blog.runevision.com/2011/12/ai-challenge-my-bot-explained-part-i.html)

[Morloth](http://forums.aichallenge.org/viewtopic.php?f=24&t=2177)

[keithroe](http://www.reddit.com/r/programming/comments/ni1mr/ai_challenge_fall_2011_ants_submissions_closing/c39e76s)

**Contact**

Mathis Lichtenberger

[mathislichtenberger@googlemail.com](mailto:mathislichtenberger@googlemail.com)

[xathis.com](http://xathis.com)