GOAL | ABSTRACT

Our goal is to prove that artificial intelligence does not need complex algorithms and neural networks - all we need is to teach it rules, give it enough time and drive space, and it can learn anything.

To prove it, in Java programming language, we developed an artificial intelligence based on a MySQL database that never looses in a Tic Tac Toe game. We did not teach it how to play - it came through every possible game combination itself and using statistics, it always knows how to behave to not let the opponent win.

Ultimately, the artificial intelligence seemed so invincible, that its testers demanded a bounty for defeating it. Currently, it stands for 10€. Can you get it?

HYPOTHESIS

For every problem, the answer with this method is just a question of the statistics. We can imagine Tic Tac Toe as a huge tree. In the beginning, it has 9 branches, then 8... and like that, we can predict a number of all the possible endings of the game to be 9!, which is equal to 362880.
Although, for the whole statistics, we must insert all possible situations of the game to the database. And we can calculate that by this formula:
insert formula here:

$$9! + \sum_{n=1}^{8} \frac{9!}{n!}| \quad \text{which is equal to 986409.}$$

Both numbers will be lower in the outcome because these statistics do not count with games shorter than 9 moves.
As we have our numbers predicted, we can calculate the time we have to give to it. Java has 1ms lowest tick timer limit, therefore it can do maximally 1000 operations per second. According to that, generating all game endings would take around 6 minutes 27 seconds, and generating all possible situations would take around 10 minutes 24 seconds.
We also predict that there will be some statistical holes because statistics do not always comply with logic.

Process

At first, we taught the AI the rules of Tic Tac Toe. We started with a 2 player game, where we completed the game logic.
After we connected our MySQL database to the AI, we started with generating the game endings. We numbered the game fields in the following way:

According to that, we can write down combinations in the form of a nine-digit long integer array. Position in the array is equal to the number of the turn, and the numeric value of a certain position is equal to the number of the field that has been taken.

Preconditions:

- Odd positions (pos%2==0) belong to the player X and even (pos%2==1) to the player O
- When a draw occurs, number 9 will be added to the 9th position for the AI to distinguish 9 turns long wins of the player X and draws later
- Minimal length of an array is 5 digits
- Numbers in the array cannot repeat

For an effective generation without statistic holes, we created a system of so-called threads.
In our AI, the threads form a nine-digit long integer array, in which every one's value determines which empty field from 0 to 8 was taken. Thread with a value of 0 takes the first empty field.

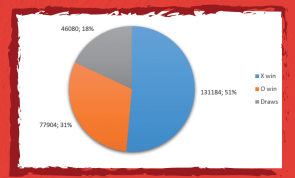
Preconditions:

- 0th position in the array can have a max value of 8 and 8th position can have only the value of 0
- Numbers in the array can repeat
- Threads are not stored into the database, they just help the AI with the generation

For example, the most basic combination 03142 would look like this:

In threads, this combination would be 02010.

After a successful generation, which took approximately 8 hours because of the performance limitations of Java and MySQL (79 times longer than we expected), we can create the first statistic graph:



The number of all possible endings is 255168, which is 30% less than we expected. Our hypothesis turned out to be correct.

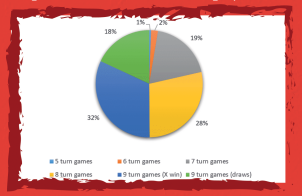
As we have our game endings generated, we can generate game situations - branches.
Branches are the foundation for the decisions of the AI.
In the database, we created a table "decisions", which has columns:

- xwin - % chance of the win of the player X for the branch
- owin - % chance of the win of the player O for the branch
- draw - % chance of the draw for the branch

At first, the AI copies all the game endings to the decision table and following the given rules, it adds to the columns xwin, owin and draw the value of 100%.

Subsequently, it goes from the longest branches to the shortest and with the help of the MySQL functions, it generates branches and inserts % of xwin, owin and draw under every branch.
To test if the branches are relevant, AI constantly checks if the parts of the branch are not in the database of game endings - large load on the database.

The generation took around 36 hours. During that, it was restarted 3 times due to the attempts to raise the performance - 131 times longer than we expected.
From the resulting data, we can make another graph:



The count of the branches is 549945, which is 79% less than we expected.

CONCLUSION

After connecting a finished database to the AI, after around 5 rounds, we found our first bug - AI playing as O lets himself to get into a situation, where it can't prevent player X to win. Probably it knows about this situation, but it reacts with bad defensive rather than offensive (it's programmed to always choose the branch where the opponent has the lowest % chance of winning).

This bug was fixed by adding a table "exceptions", in which there are manually added data which branch should be exchanged for what.
After fixing this bug and making the interface user-friendly, this AI was shifted to the testers, who didn't defeat it to this day.

USAGE IN THE PRACTICE

Of course, this is just an experiment of the usage of this AI method. It can be used anywhere where statistics apply. Many say that this method is inefficient, but this method can make you 100% sure that you have entire statistics covered. With a little help from testing, these statistics can be entirely pure. And also, nowadays with the development of quantum computers, the boundaries for this method are really high. Although for us ordinary programmers, for more complicated statistics, time consumption can be rapidly lowered by using more than one computer and making the generation agile rather than waterfall.

For an example of usage in the practice; in current times, there are many bacterias that are antibiotic-resistant. It is proven that it's more effective to use a combination of antibiotics rather than one antibiotic alone. And this is also just a question of statistics - if we can make a virtual model of the bacteria and antibiotics, AI will give us an answer.