Rust Lab - Tic Tac Toe Engine

Intro to Rust Course

Jordan Hall - jh4020@ic.ac.uk

29/12/21

Summary

Whenever I learn a new language, I usually resort to implementing the game of Tic Tac Toe. I find it makes you think about the language you're working in, and allows you to benchmark its speed in action. In this lab, you'll write a Tic Tac Toe engine for a Tic Tac Toe board of arbitrary size, as well as an AI to play the game perfectly.

Before you get started!

It's worth mentioning that you should be working on a fork of the original repository. That way, when you're done you can submit a pull request on Github and the markers can give feedback directly on the pull request.

Tic Tac Toe

Tic Tac Toe (Noughts and Crosses, Xs and Os, etc) is a very old game usually played by bored children. Traditionally, the game takes place on a 3-by-3 grid where two players (X and O) draw their symbol on an empty space of the grid. The first player to create a vertical, horizontal, or diagonal line with their symbols wins the game. Alternatively, if the whole grid fills up with no winning player, then it is a draw. Tic Tac Toe is famously solved. This means that a game played between two perfect players is entirely predictable and will end in a draw. You can try out the game by typing "tic tac toe" into Google.

A Tic Tac Toe Variant: $the\ m, n, k - game$

Now if I asked you to implement an engine for the standard rule-set, and to program a simple search AI that could play the game perfectly then you'd rightly be bored out of your mind. It's just far too simplistic.
Instead, this lab is aimed to implement a generalized game called $m, n, k - game$ (of which Tic Tac Toe is a variant). Intead of a 3-by-3 board where the aim is to get three-in-a-row ($3, 3, 3 - game$, or Tic Tac Toe), we have a board of size m-by-n and the goal is to get k-in-a-row. Note that $k \leq \max\{m, n\}$ if the game is not to always end in a draw.
Note that for the purposes of this lab, an $m, n, k - game$ has the same ruleset as Tic Tac Toe, and there are no other rules.

The AI

As computing students, there's a large probability that we're not going to be playing the game with other people. This is where an AI would come in handy. For the purposes of this task, a simple Minimax search will do. The AI is only

expected to solve the game efficiently for $m \leq 3, n \leq 3, k \leq 3 - game$, or just the standard Tic Tac Toe game.
Combinatorially speaking, an upper bound for the number of possible states to search for an $m, n, k - game$ is 3^{mn}, which means a more efficient algorithm is needed for larger boards. You may be interested in making the following improvements:

- The Alpha-beta pruning variant of Minimax (significant speedup).
- Using a Tranposition tables (hardest to do well, but best improvement).
- Implementing multithreading using the Rayon Crate (easiest and potential significant speedup).

Testing

For this lab, we have made some simple unit tests in order to show how unit tests are formatted, as well as test small elements of your code's functionality. You may run these tests by opening a terminal in this lab's root directory and typing:

```
$ cargo test
```

The tests given for this lab are quite sparse and do not tests edge cases. We highly encourage you to extend the test suite to account for any edge/corner cases your code] may come accross.

Extensions

For this lab we have elected to run three extensions:

Extension A

There exist other variants of Tic Tac Toe (most notably Quantum Tic Tac Toe). Pick one of these variants and implement it so that:

- It uses the same Game trait used in this lab.
- The AI you developed can still play it (doesn't have to be fast)

Extension B

Notice that the Game trait is generic enough to be used for any discrete, turn-based game. Try implementing an engine for any game that fits this category. For more complex games, the number of possible states becomes too large for a non-heuristic AI to search. Solve this problem by making a HeuristicMinimax, that requires a game to also implement some sort of Heuristic trait in order to be used.

Extension C

Minimax is a great search method for finding the best move to play in a game, but has some major draw-backs such as it's speed or need of a heuristic for larger boards. There also exist other search methods suited towards turn-based games (such as the Monte Carlo tree search). Implement such an AI using a different search strategy and compare how well it works against your minimax algorithm.

Submission

Once you've finished working and you want the piece to be marked, you can create a pull request on Github, and a marker will come along and give you interactive feedback on the pull request. This means that marking can be a two-way conversation between you and the marker.
Before you start a pull request, it is worth rebasing your forked repository with the original:

```
$ git remote add upstream https://github.com/JordanLloydHall/RustCourse
$ git rebase -i upstream/main
```

This can also be used to ensure that you have the lastest spec for the course as well! We are no strangers to mistakes and this allows us to fix them easily.
It is also worth linting and formatting your code before submitting it, as this will make your code easier to read for the markers:

```
$ cargo fmt
$ cargo clippy
```