Universität Bielefeld
Technische Fakultät

AG Genominformatik
Dr. Lucas Robidou

The Rust programming language
Summer 2024 / 2025

MENACE: Matchbox Educable Noughts and Crosses Engine

1 Introduction

1.1 tic-tac-toe

The tic-tac-toe is a simple turn-based game played on a 3×3 grid. Each player is represented by a symbol, either "X" or "O", and plays by putting their symbol on an empty space in the grid.

X	O	O
X	X	
X		O

The first player to align 3 of their symbols (either vertically, horizontally, or diagonally) wins. If the grid is full and there are no winners, the game is a tie.

The goal of this project is to write a program that learns how to play tic-tac-toe.

1.2 MENACE

MENACE [2] (Matchbox Educable Noughts and Crosses Engine) is a "machine" built with matchboxes, operated by a human, that learns to play tic-tac-toe.

1.2.1 MENACE setup

1. Associate a unique number for each position in a tic-tac-toe grid, e.g.:

1	2	3
4	5	6
7	8	9

2. Write these numbers on matches.
3. Associate each valid grid of tic-tac-toe with a matchbox.
4. For each matchbox, place some matches that correspond to a valid move in the corresponding grid.

1.2.2 MENACE turn

When playing tic-tac-toe, the human operating MENACE:

1. Identifies the matchbox that corresponds to the current state of the grid.
2. Opens the matchbox.
3. Randomly selects a match in the matchbox.

4. Plays the move indicated by the match.

At the end of the game, if MENACE loses, the human operating it discards the matches it just played. If it wins, it adds more of these matches to the open matchboxes, then closes them. This way, the losing moves are discouraged, while the winning moves are now more probable.

1.3 Goal

Of course, we can emulate MENACE using a program, rather than operating it manually.

Your goal is to write a program that teaches MENACE how to play tic-tac-toe by making it play against an opponent. Since a naive implementation of MENACE needs about 10^6 games to learn how to play, you'll also have to implement a random player to act as an opponent to MENACE.

Your program should output a JSON array in a file named "results.json", with a 1 for every victory of MENACE against the random player, 0 for a tie, and -1 for each defeat, e.g.:

```
> cat results.json
[-1, -1, -1, 0, 1, 0, -1, 1, 1, 1, 1]
```

If you cannot complete the entire project, leave it in a polished state, including unit tests that highlight the bugs, comments that guide me in understanding your approach, and so on.

1.4 Ressources

You can use this Python script to analyse your result output: `https://gist.github.com/lrobidou/4368868acac63214750a0a0ac66cb784`.

This project is supposed to be solved by you, not by an LLM. If, however, you are stuck and need to use one, **use the prompt given in the address mentioned above**. While studies hint that using an LLM prevents you from learning, a study showed that using specific prompts might improve learning instead [1]. The given prompt nudges LLMs into teaching you the solution, instead of giving it right away.

This project is inspired by a similar project given by Mr. John Chaussard in 2017 `https://www.math.univ-paris13.fr/~chaussar/index.html`.

2 Asssignment

Your project should be made of a few modules:

- One module containing the game's logic, which includes:
  - The tic-tac-toe grid.
  - The symbols in the grid.
- One module containing the players, which includes:
  - The human player.
  - The MENACE player.
  - The random player.

2.1 The game module

The tic-tac-toe game consists of a grid and some symbols.

2.1.1 The symbols module

1. How many states can a cell of a tic-tac-toe grid be in?
2. Represent these states in a Rust type named `Symbols`.
3. Add a method to turn a `Symbol` to a `&str` (or a `String` if you're stuck), so that you can print `Symbols` later.

2.1.2 The grid module

4. Write a Grid type that represents a single grid being played. Pick one internal representation: hashmap, array of Symbols, vector, etc. Justify your choice in a comment in your code.
5. Add a `print` method that prints the grid in a human-readable format.
6. Add a method that checks if someone won last turn, one that checks if the grid is full, and one that adds a symbol to a desired location.

2.2 The player module

Each player needs a symbol and a name to be instantiated. They need to be able to play one turn in the grid, and they need to be informed when they win or lose (so that, e.g., the MENACE player can update itself).

7. Write an interface for this common behavior in Rust.
8. Write a function `play` that takes two players and makes them play a game of tic-tac-toe.

2.2.1 The random player

The random player picks a position at random.

9. Implement the aforementioned interface for the random player.
10. Can you make the random player pick a move without using a `loop {}`?

2.2.2 The human player

The human player inputs a number corresponding to a position in the grid.

11. How to get a number from the human player?
12. Implement the aforementioned interface for the human player.

2.2.3 The MENACE player

The MENACE player needs to remember the number of "matches" in every "matchbox", and which "matches" were played during a game.

13. Write a function that takes any grid of tic-tac-toe and returns a unique integer. Also write the inverse function, which takes one number and returns the corresponding grid.

If you want to go further, for an advanced challenge:

- Take into account rotations and symmetry to reduce the number of identifiers. Be careful to also update the move you play later on.
- Only encode valid grids to reduce the number of identifiers.

Implement the MENACE player:

14. Pick a type for the "matchbox". Represent the matchboxes in a vector and use the function that maps grids to integers to access them.
15. How can MENACE remember which matchboxes need to be updated?
16. Implement the MENACE player and update its knowledge after each game. Be wary of overflows! Find a strategy to play when a matchbox is empty.

2.3 The main function

Remember that `cargo` uses the debug mode by default. If your program is taking a long time to execute, turn on the release mode!

17. Write a main function that makes the MENACE player play against a random player multiple times.
18. Starting from an empty vector of integers, for every game:
    - If MENACE wins, add a 1 to it.
    - In case of a tie, add 0.
    - For every defeat of MENACE, -1.
19. Write this vector in a JSON file named `results.json` and use the Python script to check that your MENACE player learns while playing (the victory rate should increase).
20. The victory rate might actually decrease at the very beginning. Why so?

I got some good results starting with 255 matches for every legal move. I add one match per victory and remove 5 per defeat. My MENACE player wins more than 85% of the time after 10,000,000 games. The training takes about 2 seconds in release mode (the Python script, however, takes 30 seconds to parse the results, so you might want to subsample the points to speed up your debugging).

References

[1] Greg Kestin, Kelly Miller, Anna Klales, Timothy Milbourne, and Gregorio Ponti. Ai tutoring outperforms in-class active learning: an rct introducing a novel research-based design in an authentic educational setting. *Scientific Reports*, 15(1):17458, 2025.

[2] Donald Michie. Experiments on the mechanization of game-learning part i. characterization of the model and its parameters. *The Computer Journal*, 6(3):232–236, 1963.