Final Project
Object-Oriented Programming in Python
M1 MAEF – M1 IMMAEF – QEM – DU MMEF

The final project is based on different variants of the famous game Mastermind.

Rules of the Game

Mastermind is a logic puzzle played by two participants, or one participant against a computer:

- A codemaker (which may be a computer) selects a secret code consisting of m positions, each chosen from n available colors (colors may repeat).
- A codebreaker attempts to guess the codemaker's secret code over multiple attempts.

After each guess, feedback is provided using two types of pegs:

- Black pegs indicate the number of guessed positions that are correct in both color and position.
- White pegs indicate the number of guessed colors that appear in the secret code but are placed in incorrect positions.

Formally, consider the secret code and the guess as vectors $(s_1, s_2, \ldots, s_m)$ and $(g_1, g_2, \ldots, g_m)$, respectively, where each element is an integer representing a color chosen from $\{0, 1, \ldots, n-1\}$. The total number of pegs (black and white combined) is given by:

$$\sum_{c=0}^{n-1} \min\left(\sum_{i=1}^{m} \mathbf{1}_{\{s_i=c\}}, \sum_{i=1}^{m} \mathbf{1}_{\{g_i=c\}}\right).$$

Among these, the number of black pegs is:

$$\sum_{i=1}^{m} \mathbf{1}_{\{s_i=g_i\}}.$$

The objective is to guess the secret code in as few attempts as possible.

You may practice Mastermind online at: `https://micetf.fr/mastermind/`

Variants

You will consider four variants of the standard Mastermind game by potentially combining the following alternative rules:

No Repetition

In this variant, each color in the secret code must be unique. Repetition of colors is prohibited.

Beginner's Mode (Ordered Feedback)

In the beginner's game mode, the number of black and white pegs remains the same as in the standard variant, but feedback is given position-wise. Each position in the guess receives exactly one of the following:

- A black peg (correct color and position),
- A white peg (correct color, incorrect position),
- No peg (color does not appear in the secret code).

More precisely, for each color c, the total number of pegs (black and white) associated with color c is given by

$$\min\left(\sum_{i=1}^{m} \mathbf{1}_{\{s_i=c\}}, \sum_{i=1}^{m} \mathbf{1}_{\{g_i=c\}}\right).$$

Among these, black pegs are first assigned to positions where the guess matches the secret code exactly. The remaining pegs, if any, are assigned as white pegs to the unmatched positions of color c in the guess, proceeding from left to right. This procedure ensures the uniqueness of the feedback and avoids any ambiguity.

The Project

Your project consists of two components:

Part 1: Python Package `mastermind`

Create a Python package called `mastermind` (including an `__init__.py` file and relevant modules). Your implementation must:

- Adhere strictly to object-oriented principles.
- Be concise, efficient, and Pythonic.
- Employ vectorization with NumPy wherever possible to avoid explicit loops.
- Utilize inheritance and abstraction appropriately.
- Provide clear and explicit methods for generating feedback.
- Include comprehensive docstrings for classes and methods.
- Clearly comment your code.

Part 2: Flask Web Application

Develop a Flask web application allowing users to configure and simulate Mastermind games interactively. Users should be able to:

- Select the number of colors ($n \in \{5, 6, 7, 8\}$).
- Choose the length of the secret code ($m \in \{3, 4, 5\}$).
- Specify whether repetition of colors is allowed.
- Select the game mode (Standard or Beginner).
- Specify the number of simulation runs.

Figure 1 shows an example of the application's home page.

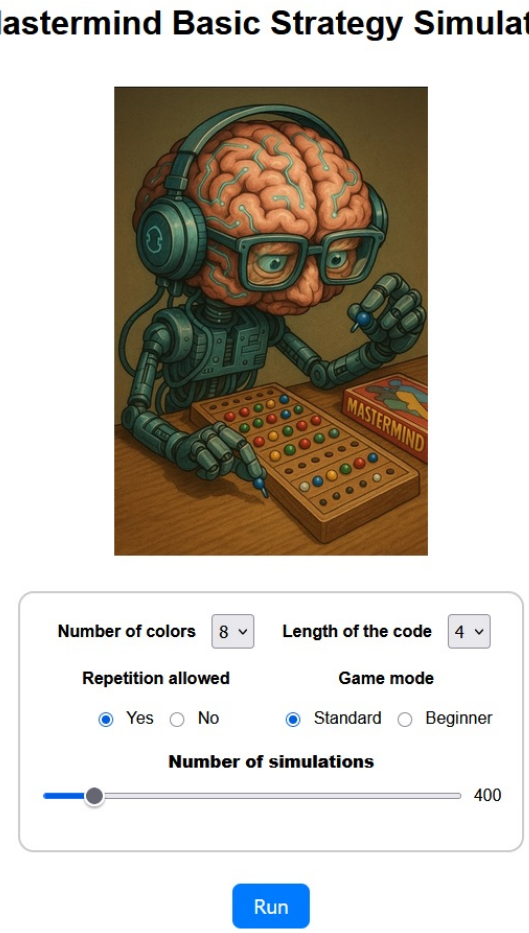

Figure 1: Homepage of the Flask application

The application should simulate games with randomly (uniformly) drawn secret codes and assume that the codebreaker uses a basic guessing strategy: at each step, making random guesses consistent with the previous feedbacks. After running all simulations, the app should display the following statistics clearly:

- Average number of attempts needed.
- Standard deviation of the number of attempts.
- Maximum number of attempts observed.
- (Optional) Visual representation of the distribution of attempts.

Figure 2 illustrates an example of the results page.

Structure your project directory as follows:

```
project/
|-- app.py
|-- templates/
|   |-- index.html
|   `-- results.html
|-- static/
|   `-- (CSS files and images)
`-- mastermind/
    |-- __init__.py
    |-- mastermind_game.py
    `-- (additional modules)
```

Request	
Number of Colors	8
Code Length	4
Repetition Allowed	Yes
Game Mode	Standard
Number of Simulations	400

Statistics on the Number of Guesses	
Average	5.465
Standard Deviation	1.017
Maximum	8

Back to Homepage

Figure 2: Results displayed after simulations

Conda Environment

Your code must execute correctly in a Conda environment created[1] from the file `mastermindenv.yml` provided on the EPI. You are only required to use standard libraries, specifically `numpy` and `flask`; avoid using additional libraries or frameworks.

Evaluation Criteria

Your submission will be evaluated based on:

- Correctness and efficiency of the implementation.
- Effective and proper application of object-oriented programming concepts.
- Clarity, readability, and thorough documentation of the code.
- Quality of the design and user experience provided by the Flask application.

Due Date and Delivery

Please submit your project by emailing `olivier.gueant@univ-paris1.fr` before **May 26th 2025**. Your email should contain a single zip file including:

- `app.py`,
- the directories `static` and `templates`,
- your `mastermind` Python package.

Clearly name your submission as `name_final_project.zip` or `name1_name2_final_project.zip`.

Good luck, and enjoy your coding experience!

[1] `conda env create -f mastermindenv.yml`