Mastermind Board Game

Atiyeh Keshavarz Safi
atiyeh.keshavarz@gmail.com

February 2020

Mastermind Board Game **0**

About **1**
- Live Demo 1
- Set-Up 1
- Running The Game 1
- Game Rules 1
- Extensions Implemented 2
  - Extension 1) Replay button 2
  - Extension 2) Difficult Level 2

Documentation **3**
- Documentation of the thought process and progress 3
  - Phase 0 - Understanding the problem, goal, and stretch goals 3
  - Phase 1 - Choose a framework and set up the environment 3
  - Phase 2 - Simple local random sequence generator 3
  - Phase 3 - Working with API 3
  - Phase 3 - External API integration 4
  - Phase 4 - Input component and Feedback components 4
  - Phase 5 - Evaluation Engine 4
  - Phase 6 - Added Replay Button 4
  - Phase 7 - Added Difficulty 4
  - Phase 8 - Final UI Touches 5
  - Phase 9 - Deploy, Document, Share! 5
- System Design 5
  - Components 5
  - MVC Model - Separate Controller: Separation of concern 5
  - Handling the UI behavior of Asynchronous Call 6
  - Page State vs Router Pages 6
- Code structure 7

About

Live Demo

https://mastermind-number-game.herokuapp.com/

Set-Up

Run the following in the root directory:

```
npm i
```

Running The Game

Run the following in the root directory:

```
npm start
```

Game Rules

This project is an implementation of the Mastermind board game. You will be playing against a computer agent and your goal is to guess a sequence of **N** numbers, between 0 and **K** each (inclusive), that are randomly picked by the computer at the beginning of each game. (Default N = 4, and K = 7).

You will have **L** attempts. (Default L = 10). At each attempt, you need to enter N numbers, and upon the submission, you will receive a feedback message.

Feedback
Unless you win (guess the sequence correctly) or you lose (run out of attempts), the feedback message will contain two numbers:

- **Partially right:** It shows the amount of number(s) you guessed correctly, but not in order yet
- **Exactly right:** It shows the amount of in order number(s) you guessed correctly, both in terms of the number and the order/position.

Extensions Implemented

The following extensions are implemented into the game:

Extension 1) Replay button

You can click on the Playback button whenever in your game, to reset everything at any point in time.

Extension 2) Difficult Level

At the beginning of the game, an option to choose a difficulty level is provided. To keep it simple in the User Interface, we only show words “EASY”, “MEDIUM”, and “HARD” as buttons.

The meaning of them is as below:

- **Easy:** Guess **3 digits number**. Each digit can be between **0 to 5**. You will win if you guess the number up to **10 attempts**.
- **Medium:** Guess **4 digits number**. Each digit can be between **0 to 7.** You will win if you guess the number up to **10 attempts**.
- **Hard:** Guess **5 digits number.** Each digit can be between **0 to 9**. You will win if you guess the number up to **12 attempts**.

Documentation

Documentation of the thought process and progress

Phase 0 - Understanding the problem, goal, and stretch goals

The first phase for me was to understand the problem, inputs that I have, and minimum requirements. Since I didn't have former knowledge about the game and it was the first time I heard about it, I also downloaded an iOS app of Mastermind on my phone and played with it for a while.

Phase 1 - Choose a framework and set up the environment

I decided to build my application based on React framework. I had prior experience with React in my other projects and it could suit here as well.

To build the project and set up the initial scaffolding, I used react-scripts which is based on create-react-app (link: https://github.com/facebook/create-react-app) It helped to kick off the project easily and start a server very quickly. I successfully set up the react and have a "hello world" on my localhost server.

Phase 2 - Simple local random sequence generator

I initially started with a local controller (as in MVC, more details below) to be the engine of the game. To start off, the controller was generating the sequence using `Math.random` locally and I tested communicating between the main component and the controller.

Phase 3 - Working with API

Since one of the requirements of the project was working with random.org API to generate the random numbers, I first created an account over there, got a private API key, and tested their API endpoints on Postman to confirm everything works as expected. (I later decided to go with their public API, which doesn't require an API key, for easier setup and deploys.)

Phase 3 - External API integration

Once I got the API working, I tried embedding it into the code. Since it is an AJAX call (and inherently, asynchronous), I needed to maintain a loading state to display a loading indicator in the interface.

Phase 4 - Input component and Feedback components

I continued by creating two more components for input (user entering their guess) and feedback (sharing the evaluation result of their input back).

I established communication between these components and the Main component in React using props (parent to child messaging) and callback functions (child to parent communication).

Phase 5 - Evaluation Engine

To properly compute the result of a user entry, I created an evaluation function in the game controller. It computed exacts and partials properly. I also pick an emoji based on how close their guess is, to the result! 😇

Phase 6 - Added Replay Button

I refactored the initialization of the game/controller to be reusable upon user's click on the new button of "replay" at any time.

Phase 7 - Added Difficulty

To challenge myself, I decided to introduce a feature of "difficulty" (explained above) and allow the user to pick the difficulty level of the game at the beginning. To support that, I refactored my controller to get the configs (N, K, L, etc.) not at the construction time, but at the setup time (which will be called after user's difficulty pick).

I also found that since the "difficulty selection" component had its own stand-alone UI, I needed the whole state of the app to be a tri-state variable (Loading, showing the game, showing the difficulty selection.) Previously, I only had the first two, so I was handling it with one single boolean variable and my render function was just a ternary operation.

It was challenging to get it to work properly (and reset other properties of the engine correctly), but once it worked, it was so nice!

Phase 8 - Final UI Touches

Finally, to make it look more beautiful, I played with different UI frameworks (Bootstrap, Material, …) but finally decided to go with a simple local stylesheet.css to keep it lightweight and easy to customize.

Phase 9 - Deploy, Document, Share!

And as the last piece, I deployed the code to Heroku, and finalized this document to be sent out and shared!

System Design

Components

I draw the UI of my application on my whiteboard to have a better understanding of my final product. Then I designed all possible components to start off with the project.

The UI components I currently have are:

1. **MainBoard:** Is the parent component and the rest would be children.
2. **Keypad:** Is the component which will take care of a player entry
3. **Feedback:** Is in charge of providing feedback to users
4. **Replay:** restarting the game
5. **Difficulty:** handles three types of level in the game

MVC Model - Separate Controller: Separation of concern

While it's a React app with modular components, I decided to have an external controller for the game-related stuff, as the game engine and put all the logic related to how the game works under the hood, over there. It helped with keeping the code clean and helping me easily figure out when I was in non-UI parts of the codebase (e.g. evaluation function).

Another benefit that I initially had in mind was to separate the logic as much as possible from the UI, so when/if I had a server-side compartment for the project, I could easily replace that.

Handling the UI behavior of Asynchronous Call

Since the initial call to generate the random sequence (internally, called “secret”) is an asynchronous call the random.org API, there would be a delay in the user’s workflow that needed to be compensated with proper elements in the UI.

Thinking about it, I had two options on how to handle it.

- **Option A) Blocking with Loading UI**: It’s showing the “loading” state during the time that the API request is on the fly. So technically, nothing is running parallel and we are handling one task at a time.
- **Option B) [More advanced] Letting user submit their first entry and making the API call in the background:** Since we didn’t need the entire sequence to be there to accept the first entry from the user, we could let the API call happen in the background (kinda, in parallel) to save user’s time.

Assuming the API call would take 2 seconds and it usually takes at least 3 seconds for the user to type in their first guess, option B would result in a seamless User Experience! But handling the case that the API is taking longer (i.e. it’s not back when the user has submitted their first entry) would be quite challenging. That’s why I decided to go with option A for now and KISS! (Keep it Stupid Simple!) :) (link to: https://en.wikipedia.org/wiki/KISS_principle)

Page State vs Router Pages

I initially thought that the whole thing can fit into one single page and I wouldn’t need a Router to maintain my Single Page Application. That’s the impression I had got from playing with the simple iOS app as well. However, down the road, I first found that I need a loading page/state and then I found I had a third page/state for difficulty selection.

I am juggling with that and holding all of them into one Main component, using a render function that picks which component to render and display based on this.state.pageState. However, I believe the right way of handling this would be having a Router and 3 separate pages for these cases.

I didn’t get the time to make that happen, but it can be a good refactoring for future.

Code structure

My application has the following folders:

- **public:** The HTML file that shapes the single page
- **src**: It has all components and controller, CSS, and images
    - **components:** the React components that I created
    - **configs:** the configurations of the app, including the API key/path
    - **controllers:** the stand-alone controllers (e.g. game engine)