PROJECT REPORT
STUDY WEEK "FASCINATING INFORMATICS"

Game Platform Mastermind

L. Meili[1], N. Pross[2], L. Stampfli[3]

[1]Kantonsschule Büelrain, Winterthur, Switzerland, [2]Scuola Arti e Mestieri Bellinzona, Ticino, Switzerland, [3]Kantonsschule Wattwil, St. Gallen, Switzerland.

Supervised by: Dominik Link
Date: 13 September 2017

Abstract

The Mastermind Game Platform is a multi use arcade electronic platform designed and built during a study week organized by the Swiss Youth in Science Foundation. The goal of the project is to design both a hardware and a software for an electronic version of the classic board game Mastermind, to then extend by adding an algorithm that plays on its own. The software is implemented in Python 3 running on a MicroPython compatible microcontroller ESP32 by Espressif while the hardware is built with a PCB (Printed Circuit Board) or a VeroBoard for prototyping. Many technical challenges were encountered such as limited dynamic memory and hardware failures but the final prototype was complete with all of the previously mentioned features.

1 Introduction

The goal for our project is to prepare a working prototype for an electronic version of a classic board game called Mastermind. The secondary goal is to implement an algorithm that plays the game on its own, possibly better than how human would. Lastly a third objective is to extend the platform be able to run other games such as Tetris or Snake.

1.1 Mastermind Game Rules

Mastermind is a two player game, where one player known as the code maker prepares a secret code, and the other is the code breaker whose job is to guess the code within a finite number of attempts. The code is represented by colors with plastic or wood pegs. The classic variant has a code of 5 pegs of 6 colors and a maximum of 12 attempts. To begin, the code breaker sets a code, then the code maker gives a feedback (hint) based on the following rules by placing a peg on the side of the board.

- There is a correctly colored peg in the correct position.
- There is a correctly colored peg, but in the wrong position.

Note that the hint does not indicate which pegs are correct or wrong. The code breaker wins when he discovers the code, or loses if he runs of out of attempts.

1.2 Platform requirements

The game platform will need to replicate the features of the original game. Since it is electronic, the platform should also augment the game to be be playable only with one player, with the computer being either the code maker or the code breaker. Finally the platform should offer intuitive controls that need little or no explanation.

2 Materials and methods

2.1 Hardware and software

The main processing unit for the Digital Mastermind is an ESP32 microcontroller by Espressif, and the pegs are represented on a display composed of 3 Adafruit NeoPixel LED Matrices which in total give a screen of 8 by 17 pixels. To receive the users input there are 4 pushbuttons on the top with an integrated LED in each of them. The software for the project is written entirely in Python 3 with JetBrains IDE PyCharm CE. On the microcontroller board the code is run by MicroPython, a small implementation of Python 3 that offers a subset of the Python standard library and many feature to control lower level hardware components. The case for the electronic Mastermind is entirely 3D printed with an Ultimaker printer. The materials used are a mix of a PLA and PETG type plastic. Other software tools were also used. MobaXTerm is a set of tools for remote computing, in our case it was used as a serial terminal to communicate with the microcontroller. AMPY, which stands for Adafruit MicroPython Tool, allows to write, read and list files that are present in the MicroPython File System; We used it to upload our code onto the microcontroller.

2.2 Design decisions

Because of some hardware limitations the implementation of Mastermind differs slightly from the original game. First, the display has 8 columns instead of 10 reducing the length of the code to 4 and consequently making the game easier. Another difference from the classic version is in the number of tries, in our version the Code Breaker has 16 attempts before losing.

2.3 Team organization

The work was split up in 3 parts: Hardware, Software and Algorithm. Naoki designed the electrical scheme and a printed circuit board. Unfortunately, it wasnt possible to the print it in time for the end of the week, so we had solder the parts on a prototyping board. Lukas had to implement the game logic for the microcontroller. And Luke implemented an algorithm that solves Mastermind in the least possible number of turns.

2.4 Solver algorithm

The algorithm is composed by two parts. The first part validates the responses from the other players and keep all still possible combinations of colors in a list. The second part then chooses one of the remainings codes depending on the situation. We first tried to implement Donald Knuth's algorithm [5] which basically creates a huge tree with the best colors to take in every situation. But because of the limited memory on the microcontroller we had to take another simpler attempt. Our final Algorithm of choosing a good color code to guess took the code just took the element that was at the position of the square root of the lists length minus one. We found that solution by bruteforcing through every color code that was possible and then comparing the result with other constants. At the end our algorithm was able to solve mastermind with an average of 4.82 turns which is nearly as good as the far more memory consuming alternatives.

2.5 Software design pattern

To write the software a pattern called State-machine [3][4] was used since Mastermind and many other similar games can be represented as a sequence of states with predetermined actions. A secondary reason is that State Machines make it easier to add new code to the program. This design decision comes handy if someone wants to add a new game or another feature to the project. In our implementation of Mastermind there are 9 states: `codeSetting`, `autoCodeSetting`, `codeGuess`, `autoCodeGuess`, `check`, `won`, `lost`, `waitingForReset` and `reset`.

- `codeSetting`: Here the Code Maker sets his code. After the Player sets up his code, the program will hide the code and switch to the codeGuess state.
- `autoCodeSetting`: As mentioned before this state emulates the Code Maker. It generates 4 random colors for the code for a single player mode. As soon as the code is set, the program will clear the code and then switch to the `codeGuess` state. This state can be entered by pressing button 4 while in `codeSetting` state.
- `codeGuess`: Here the codebreaker can try his best to break the code. Like in the `codeSetting` state the player can skip through the various colors and

confirm with the same buttons with the exception of button 4 that now works as a game reset. It switches to the `reset` state.

- `autoCodeGuess`: This state implements Lukes algorithm that plays the game. It is more a show of than of any use.
- `check`: In this state, the program checks the input from the Code Breaker and compares it to the color code of the Code Maker and gives a feedback according to the rules. Green means correct color and correct place, white means just correct color. After this process the the program can switch to either state `won`, if all colors are correct and on the right place or state `outOfTries`, if the player runs out of attempts. Or if none of the previous conditions are met, the state returns to `codeGuess`.
- `won`: This state is simple. It fills the the next row with green. It indicates that the Code Breaker has won. Then switches to `waitingForReset`.
- `outOfTries`: In this case the Code Breaker loses and the Code Maker wins. It colors all LEDs in red to indicate a GameOver. After that, the program switches to the `waitingForReset` state.
- `waitingForReset`: The program will simply waits for button 4 to be pressed.
- `reset`: Once this state is called, it resets all variables and LEDs and goes back to the `codeSetting` state.

2.6 Easter egg

Since we had some spare time the LED Matrix is able to show moving GIFs. Any image can be uploaded with a GIF to byte array conversion tool we programmed. The program converts any a 16 by 8 pixel animated GIF file to an array of frames that contain 3 bytes for each pixel.

This should be a secret state that is only accessible through a secret color combination while in codeSetting state. When the correct combination is entered a secret moving low resolution GIF will appear on the LED Matrix as easter egg.

3 Results

In the end, a working Mastermind electronic game prototype was produced. Player one can set a color code and Player two place guesses on how the color code looks. The system returns a feedback consisting of the amount of correct placed colors (green) and wrong placed colors (white). There is also a single player mode implemented were the computer generates the color-code on its own with a random one random. Also some small secret features were implemented.

4 Discussion

4.1 Technical challenges

4.1.1 Debugging

Debugging the software was difficult because the code was executable only on the microcontroller due to missing libraries. A software debugger couldnt been attached because the program could only run on the microcontroller.

4.1.2 Random number generator

The random number generator in the ESP32 microcontroller does not generate pure random numbers. Like in many electronic devices actually it uses a pseudo random generator function that needs a starting seed value. To get a truly random value to use as a seed the uptime counter was used since the measure, in microseconds, depends on the time the user spends in the `codeSetting` state.

4.1.3 Memory limitations

As mentioned before, the platform was intended to implement the auto solve algorithms. But unfortunately the ESP32 is not able to perform this task because it does not have enough dynamic memory (RAM) to store the informations needed to compute the solution. So this part was not implemented in in the main program. It is still possible to run this code on a separate program that does not allow to play the game manually.

Also during development the of lack of memory didnt allow to run the code. The problem was solved by invoking manually the Micropython garbage collector, the ESP32 was then able to run the main program.

4.2 Hardware failures

Problems from hardware failures were also present. Originally the platform control were intended to be button 1 and button 2 to loop forward or backward through the colors and button 3 to confirm. But button 3 suddenly stopped working in the middle of the project, so it was changed to have button 1 alone to circle through all the various colors while button 2 confirms the selection.

5 Acknowledgements

We especially thank the Swiss Youth in Science that made possible to organize this great experience. We also want to thank Markus Knecht and Claude Rubattel for coordinating the study week and our supervisors Dominik Link, Flavio Müller and Manuel Schlatter for supporting us. Last but not least we thank the University

of Applied Sciences and Arts, Northwestern Switzerland, School of Engineering for offering the infrastructure to host the event.

Source code

The source code and all documents related to the project can be found in a Git repository hosted on Github at: `https://github.com/flavio99/Mastermind`