Flooduino

University of Basel, Computer Architecture
Pascal von Fellenberg, Istref Uka, Frederic Weyssow, Alexander Marggraf
January 19, 2025

Abstract

The goal of this project was to create our own implementation of the game Flood-it on an Arduino. The outcome is a fully functioning implementation of the game encased in a cardboard housing which resembles an old arcade machine. To do this we used an Arduino Mega, a LED-matrix, a DFplayer Mini along with a speaker and three Buttons. Furthermore all of our own code is written in C and the project is compiled using the Arduino IDE. Everything is coordinated by the Arduino Mega on which the LED-matrix, a DFplayer Mini and the buttons are connected. The LED-matrix displays the current state of the game, the DFplayer Mini along with the speaker play music depending on the different stages and the three buttons allow the player to interact with the device. At the end of our project, everything worked as intended although not always on the first attempt.

Figure 1: The finished product

Introduction

The goal of the game Flood-it is to fill a grid of differently colored cells with the same color in the maximal number of allowed moves. To start with the player can choose the size of the grid and how many colors should appear on the grid. Then each move the player can choose the color of the top left cell which is then propagated down the grid using a flood fill algorithm at the end of each move. If the player manages to use no more than some predetermined number of moves he wins, however if he uses more moves he loses.

Methodology

Prototype

It is necessary to quickly mention the prototype we had created, because it was an important step of our process, as it allowed us to implement the game logic without having connected all the necessary hardware components. It was made completely in C using the raylib graphics library for visual output and ran on our personal computers. We did this to separate the software and hardware problems as much as we could, as we weren't familiar with the hardware components we were to use. We structured our software into the following modules to make porting it to the Arduino easier: input, game logic and output. The input module's function is to detect any button presses made while the game is running. The output module implements some methods which are used to update the window in which the game is shown. To achieve this we used the raylib library. The game logic module is the entry point of the game and uses the aforementioned modules to bring the whole program together. Additionally we used global variables to share the game state between modules.

Porting of the game to the hardware

We were relieved to find out that the porting was relatively straightforward. By using the Adafruit GFX library as output and three buttons on the breadboard as input we were able to quickly confirm that our hardware was working, although it wasn't perfect yet. During this process we also decided to add some music to the game to improve the experience of playing the game, which also came with some challenges, see **??**.

Schematics and hardware used

- DFPlayer mini MP3 Player Module

- Speaker 4Ohm 3W 50mm
- Buttons for breadboard (3 pcs)
- USB b cable
- Arduino Mega 2560 Rev3
- 9V 1A Power Supply for the Arduino Mega
- RGB P3 Matrix Panel 64x64 HUB75
- 5V DC 5A Power Supply 5.5mm/2.1mm Plug for the Display
- some jumper wires
- breadboards (2 pcs)
- transistors s9018 h331 (2 pcs)
- 1k Ohm resistors (2 pcs)
- 10 Ohm resistor
- some cardboard boxes and hot glue

Course of the game

For the final product we decided to use three buttons. The arrow keys up and down as well as an enter key. These are used as following. At the beginning of the game the player presses enter to switch from the start screen to the selection screen. Here the size of the field is chosen first. The player can press up or down to increase or decrease the size of the field and press enter to select a size. Afterwards the amount of colors can be chosen where just like the field size the number of colors is increased or decreased with the arrow keys and selected with the enter key. After this is done the game switches to the main game. To select the color of the top left cell the up and down arrows are used to switch between the different colors. Enter is used to initiate the flood fill algorithm and marks the end of a move. Depending on whether the player filled the field with the same color within the allowed number of moves he is prompted with won or lost screen. Pressing enter in either will restart the game and switch to the start screen.

Challenges

Input

The buttons work using interrupts. Every time a button is pressed an interrupt is triggered which increases an internal variable which tracks how often a button has been pressed. This allows us to register any button presses regardless of what the processor is doing. The game loop will check for each counter if it is bigger

than zero and perform actions accordingly. Each time the state of such a counter is queried, it is decremented.

A challenge we faced here were the high bounce times of the buttons. A bounce describes the period after a button has been pressed where rapid oscillations trigger multiple unwanted button presses. To deal with this we used a debounce period in which any further button presses are ignored. This was ultimately put at 250 milliseconds which is quite a long time. This leads to the truncation of any button clicks which where made in rapid succession. However, in our experience this is rarely a problem since the game does not require rapid button presses.

Displaying bitmaps with the Adafruit library

One challenge we faced was to figure out how to display images on the LED-matrix. This was especially difficult since there was no clear guide on how to do this. Our basic idea is to convert the image to an array of rgb values, using a Java program, and store this array on the Arduino which can then be displayed with the Adafruit library. However this process was not straight forward as the colors have to be stored as 16-bit values compared to the normal 32 and each value has to be split and stored in two sequential fields of the array. To add to this there still remains some uncertainty about the exact order every value has to have. This process required a lot of testing and experimenting but in the end we managed to get it to work.

Music

We decided at the beginning of the assembly of our arcade that we wanted to implement music for the game to enhance the experience for the player. We ended up deciding that every stage of the game has its own song in a loop. This was to be achieved using the DFplayer Mini which operates independently from the Arduino, playing music saved on an inserted sd card. Additionally to this we acquired a speaker for auditory output.

Initially we intended to use a serial connection to control the speaker but this didn't work in combination with the Adafruit GFX library. To circumvent this we communicated with the DFPlayer mini via it's IO1 and IO2 pins to change volume and skip as well as loop songs. Normally this is achieved using button presses but since the music should work independently of the input we used transistors to simulate button presses. Using this the game increases the volume at the start of the game and skips to the next song if the stage is changed. If a song is over we skip ahead until the same song is playing again to loop the song indefinitely.

After putting all the connected hardware components into the housing, this music control didn't work correctly at times. We suspect that this was caused the wires and transistors that are involved in controlling the DFPlayer Mini being too close to the speaker, which with it's changing magnetic field caused interferences. Placing some cardboard between the speaker and the rest of the hardware fixed this issue.

Housing

The housing was the last thing we decided on and built. Since the game and the hardware used is reminiscent of old arcade games like PAC-MAN we decided to go with a similar look for the housing. It was heavily inspired by old arcades popular in the 1980's. The most difficult part was simply our inexperience with crafting things out of cardboard. We had to make up a design and hope it would work and look nice the first time trying since the process was very time consuming. However we managed to get it to work roughly as we expected it to work and look like.

Conclusion

If we could start over again we would be more careful with the usage of libraries and their side effects on such constrained hardware (e.g. Adafruit GFX making the serial connection impossible). Additionally we would design the housing for easier access to the hardware and with some more space. We used Git and Github for version control which was very beneficial and we will use it again for future projects. Due to our well structured software we could split the labor evenly. Additionally due to our weekly meetings at Friday on 4:15 pm we were able to make continuous progress and not get too stressed close to the deadline. This was very important since we also learned that each stage took longer than expected. Most importantly everyone was motivated and we were able to grow as a group.

Division of Labor

Input and visual design: Alex
Visual output on prototype and porting: Pascal
Game Logic: Frederic and Istref
Hardware, housing and music choice: Everyone

References

Inspiration: https://unixpapa.com/floodit/
libraries used: Adafruit GFX https://github.com/adafruit/Adafruit-GFX-Library
and raylib https://www.raylib.com/
Source of electronics: we ordered from https://www.bastelgarage.ch/ and used
some parts that were made available by the University of Basel
code origin (for raylib prototype): https://github.com/raysan5/raylib-game-template
our repository: https://github.com/AlexMarggraf/Flooduino/