Flooduino

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

January 18, 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 some 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.

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 looses.

To do this we used The Arduino Mega which was beneficial for us because of its increased memory capacity and its many connection pins. In addition we used a 64x64 LED-matrix from Waveshare as visual output, a DFplayer Mini to control the music, a Speaker for auditory output and three buttons as input. The LED-matrix always displays whatever stage the game is in at the current moment. These being the start screen, selection screen, the game and a win/loss screen. The DFplayer Mini functions similarly as it plays different music depending on the different stages of the game. The buttons consist of the arrow keys up and down as well as an enter key. The arrow keys are used to increase or decrease the field size and the amount of colors used in the selection screen and as a way to select the color in the game itself. The enter key is used every where else meaning to switch to the next stage and to initiate the flood fill algorithm.

We first created a prototype using C and raylib which is able to be run on any computer. Then we migrated this code over to the Arduino where changes were made wherever necessary. This includes using an Adafruit library instead of raylib for the visual output. Implementing controls for the DFplayer Mini as well as detecting the button presses. Next everything was build together using two breadboards, one for the buttons and one for the DFplayer Mini. Necessary connections between the devices where made using wires and transistors. Finally a cardboard housing was build which is made out of cardboard boxes that we bought in a hardware store. The shape was cut out and glued together using hot glue such that it resembles an old arcade.

Methodology

Prototype

It is necessary to quickly mention the prototype we had created. It was made completely in C using the raylib graphics library for visual output. At its core it consists of three modules being the input, game logic and the output. The input module's function is to detect any button presses made while the game is running. It does this using a function to detect button presses from the raylib library. 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 as well. The main part is the game logic module in which the main loop is run. It is the entry point and is responsible for most actions. Mainly it is used to initialize everything, parse input and outputs using the aforementioned modules, run the game loop, check whether a button has been pressed and update the screen which shows the current state of the game.

Arduino Implementation

To implement the

Input

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.

For the buttons we had two options. One was to use keyboard switches with breakout boards. This would make for much nicer buttons but would include a higher cost and more work. The second option where regular breadboard buttons which can be bought in any electronics store. We ultimately decided on the second option since it meant less cost and less work for us. 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.

A challenge we faced here where 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.

Visual Output

Music

Housing

Conclusion