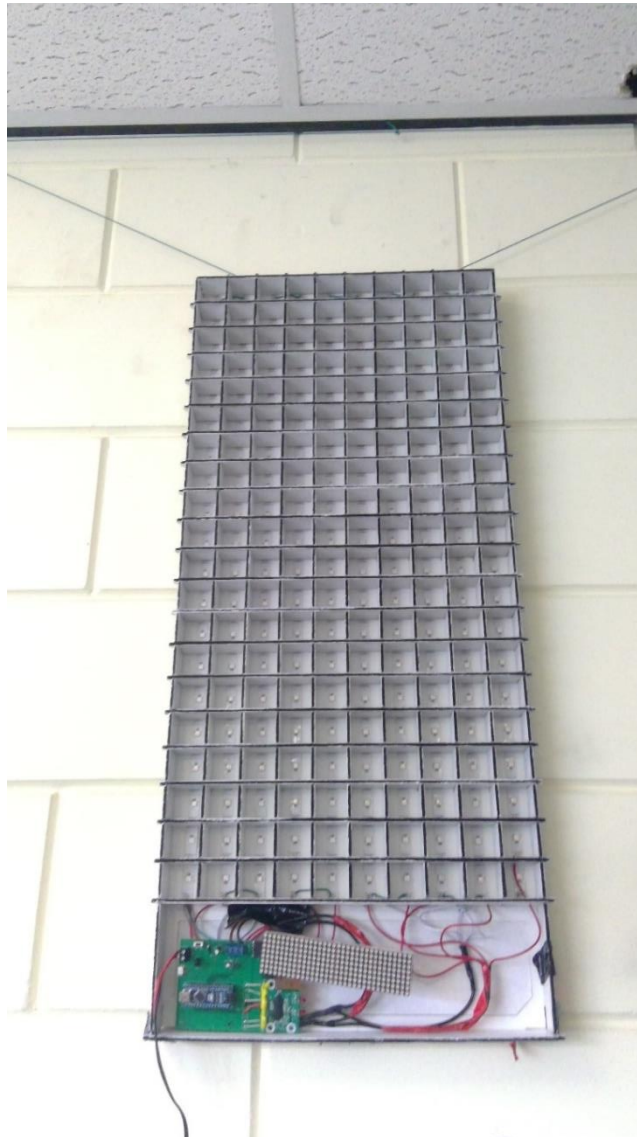


Arexx Game Board



Author: Sjors Smit

Date: June 17th 2020

What is the Arexx Game Board:

Arexx Game Board is a large display made of WS2812b/NeoPixel RGB smart LEDs. On which multiple games can be played.

The display is driven by an Arduino Nano development board. Also attached to the Arduino Nano is a wireless receiver for a game controller and a smaller 32x8 pixel dot-matrix display, controlled via the SPI bus. The main display will be used to show the menu, play games or show other images, for example a clock. The small dot-matrix display is used to show short strings of text (max 6 characters) and the score in a game.

The entire board receives power from a 5V/3A source, either via a micro USB port or a screw terminal block. The large RGB display is also connected via a screw terminal block and the small dot-matrix is connected to the board via a 5-pins female header.

What can Arexx Game Board do?

Currently the first game on Arexx Game Board has been fully implemented: A complete version of the popular game "*Tetris*" is playable on the large display. Currently also in development is a version of "*Pong*" and "*Snake*". Besides playing games there are also plans to include a feature to display a large clock on the display.

Component overview:

Electronics:

Arduino Nano:

Arduino Nano is a very tiny (hence the name nano) microcontroller development board from the Arduino line of products. The board is based on the Atmel Atmega328p microcontroller, a small 8-bit AVR processor with 32kB of flash memory (of which 2 kB is reserved for bootloader), 2kB of SRAM, 22 GPIO pins, of which 6 pins have PWM output capabilities and 8 pins have a 10-bit ADC built in. Besides that the processor has built in hardware for UART, SPI and TWI/I²C interfaces for serial communication.

The Arduino can be easily programmed in C/C++ with the Arduino IDE.

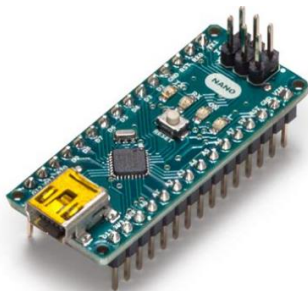


Image 1.1: an Arduino Nano

ws2812b/NeoPixel:

A ws2812b “smart” LED is an array of a red, green and blue LEDs, with a built in controller to adjust the brightness of each LED in a single 5050 SMD package. The smart LEDs are connected in series and are controlled via a serial data protocol, which gets sent on through all the LEDs.

The package consists of 4 pins: V_{DD}, GND, DI (data in) and DO (Data Out). The Data in pin of the first smart LED is connected to a GPIO pin of the processor. Then the second, third etc.. are connected to the Data Out pin of the *previous* LED, in a so called “Daisy-chain” setup:



Image 1.2: A single ws2812b smartLED

Cascade method:

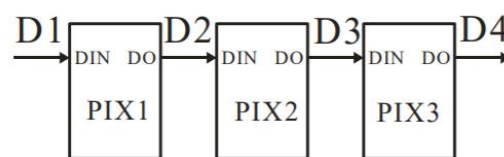


Image 1.3: the daisy chain setup

The ws2812b smart LEDs can be controlled easily using Arduino with the NeoPixel library provided by Adafruit.

Dot-matrix display:

The dot-matrix display consists of four 8x8 dot-matrices, each driven by a MAX7219CWG SPI display driver. A matrix consists of 64 LEDs connected in an 8x8 common-cathode array. The display controller uses a technology called “scanning” to drive the display. This means actually one row (horizontal line) of the display is visible at a time, but because it scrolls through all the lines really quickly, our eyes perceive it as if all the LEDs are on at the same time (commonly known as persistence of vision or POV). The display driver is connected to the Arduino Nano via the SPI bus. The Arduino “tells” the display driver which LEDs to turn on and off on the display. Similarly to the ws2812b, the MAX7219CWG display drivers are connected in series in a daisy-chain setup.

The display comes with its own easy-to-use Arduino library, which is compatible with the graphics primitives provided by the “Adafruit_GFX” library, for drawing small images and text to the display.



Image 2.4: the 4 8x8 dot matrices and underneath them the circuit board with de display drivers

Dualshock 2 controller/receiver:

To be able to play games on the display, a separate (wireless) controller is connected to the Arduino Nano. In this case, it is a wireless gamepad, heavily “inspired” (AKA copied) from the game controller of the Sony Playstation 2 game console: the “Dualshock 2”.

This is a wireless controller with a small simple receiver, the receiver can be connected to the circuit with a 9 pin connector. (of which only 6 pins are actually used). The controller has a “data in” (PS2DAT), “data out/command” (PS2CMD), “clock”, “select/activate” pins, and pins for power (3.3V/5V) and GND (ground).

Although the data stream, using command (MOSI), data in (MISO) ,clock and chip select pins, is very similar to SPI, it is actually a little different and thus requires its own separate connection pins to the Arduino board, instead of being connected to the SPI bus, like the dot-matrix display is.

To use the controller, an Arduino library called “ps2xlib” is provided to easily read which buttons on the controller are pressed at any given time.



image 2.5: the wireless game controller and receiver

Mechanical:

The electronics will be mounted to a laser cut plywood panel. The panel consists of 2 sections. The top section is 340x675mm (w x h), and consists of a grid of 10x20 small squares, each 30x30mm in size.

In each of these small squares, one of the smart LEDs of the ws2812b strip is placed, turning this entire section into the main screen. The bottom left LED is the first pixel of the strip, it goes up to the top and then goes down again, turning the LEDs daisy chain into a zig-zag pattern:

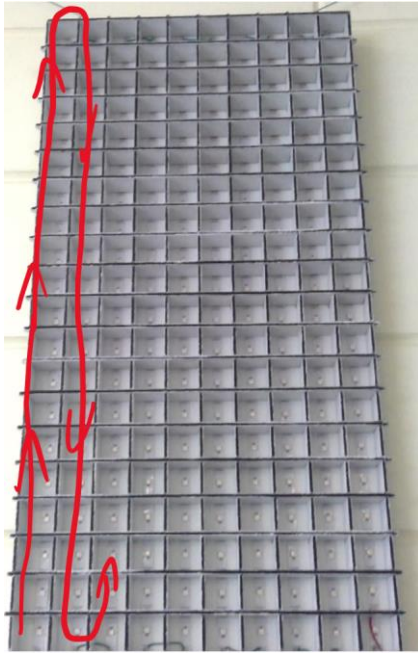


Image 2.6: The display section of the panel, with the arrangement in which the LEDs are set up.

In the bottom section is a larger area, covering the full width of the panel (340mm) and with a height of 100mm. In this section of the panel all the other electronics will be stored.

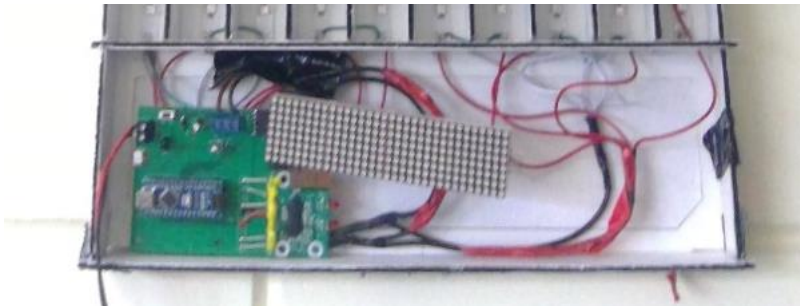


Image 2.7: The bottom section of the panel, housing the electronics. (Lay-out not final)

Software:

The software for Arexx Game Board is written in C++ using the Arduino IDE. The Arduino IDE is an easy to use programming environment that makes it easy to find and include software libraries made for the family of Arduino Development boards, like the Arduino Nano used in this project, and it can easily flash the program into the Arduino's memory with the use of the pre-installed Arduino-Bootloader.

To make the programming of the display easier, a few libraries have been included:

Library	Purpose
<Arduino.h>	The base library automatically included in every Arduino program. It provides all the basic functionality used in the Arduino IDE.
<SPI.h>	A library to be able to send and receive data over the SPI bus.
<Adafruit_NeoPixel.h>	A library to easily control the ws2812b/Neopixel LEDs.
<Adafruit_NeoMatrix.h>	An extension to the NeoPixel library, giving features and commands for controlling NeoPixel strips arranged in a matrix setup.
<Adafruit_GFX.h>	A library that integrates graphics primitives like drawing lines, simple shapes like squares, circles and triangles, and displaying text on a multitude of compatible displays.
<PS2X_lib.h>	A library to easily read out the buttons pressed on a wireless PS2 controller.
<Max72xxPanel.h>	A library to display text and other graphics on dot-matrix panels using a MAX72xx driver IC.
<DSRTClib.h>	A library to control the DS1339 RTC module

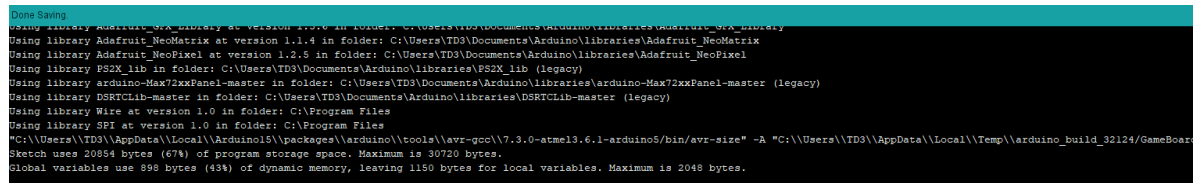
```
Multiple Games for Arexx Game Board
1  /*
2   Multiple games for on Arexx Game Board
3   Author: Sjors Smit
4   Date: June 2nd 2020
5   AREXX ENGINEERING
6   */
7
8   //used libraries:
9   #include <Adafruit_GFX.h>
10  #include <Adafruit_NeoMatrix.h>
11  #include <Adafruit_NeoPixel.h>
12  #include <PS2X_lib.h>
13  #include <Max72xxPanel.h>
14  ..
```

Image 3.1: the libraries included in the code

Limitations:

Due to the low amount of SRAM in the Arduino, the code had to be written memory efficiently. Out of the 2kB of SRAM, a little over 600 bytes were already in used just for the framebuffer of the RGB display. For the remainder of the program almost 900 bytes are used for global variables. These include all 28 shapes for the Tetris shapes, and all their respective colours, variables for positions of objects on screen, timers, score, wireless controller, dot-matrix and a few other variables. This

leaves a total of a little under $2048 - 600 - 900 = 548$ bytes of SRAM for local variables, which are used within the functions, for example: the current shape takes up 2 bytes of local memory, but then also when it has to be drawn on screen, 3 more bytes are used to bit-shift to determine to proper location, for collision detection there's also space needed. But even with all this, there is enough space left to include one or two games and features onto the display without the need of upgrading the processor or memory capabilities.



```
Done Saving.
Using library Adafruit_NeoMatrix at version 1.10.6 in folder: C:\Users\TD3\Documents\Arduino\libraries\Adafruit_NeoMatrix
Using library Adafruit_NeoMatrix at version 1.1.4 in folder: C:\Users\TD3\Documents\Arduino\libraries\Adafruit_NeoMatrix
Using library Adafruit_NeoPixel at version 1.2.5 in folder: C:\Users\TD3\Documents\Arduino\libraries\Adafruit_NeoPixel
Using library PS2X_lib in folder: C:\Users\TD3\Documents\Arduino\libraries\PS2X_lib (legacy)
Using library arduino-Max72xxPanel-master in folder: C:\Users\TD3\Documents\Arduino\libraries\arduino-Max72xxPanel-master (legacy)
Using library DSRTClib-master in folder: C:\Users\TD3\Documents\Arduino\libraries\DSRTClib-master (legacy)
Using library Wire at version 1.0 in folder: C:\Program Files
Using library SPI at version 1.0 in folder: C:\Program Files
"C:\Users\TD3\AppData\Local\Arduino15\packages\arduino\tools\avr-gcc\7.3.0-atmel3.6.1-arduino5/bin/avr-size" -A "C:\Users\TD3\AppData\Local\Temp\arduino_build_32124/GameBoard
Sketch uses 20854 bytes (67%) of program storage space. Maximum is 30720 bytes.
Global variables use 898 bytes (43%) of dynamic memory, leaving 1150 bytes for local variables. Maximum is 2048 bytes.
```

Image 3.2: the amount of flash and SRAM memory used by the program in its current state.

Software Overview

The software is split into different sections: The declaration and setup, the main menu and a section for every feature.

First off, all libraries are included, constants are defined, global variables are declared and objects are created.

Next up is the setup() code. In here all objects and ports are initialised. Pins are set to be either an input or output. The SPI, I2C and UART data protocols are setup and the displays get setup in the proper orientation.

The main()/loop() section is used for the main menu screen. In here the controller can be used to select what feature should be played. Option 0 will launch the clock, option 1 is to launch TETRIS and option 2 launches PONG.

To exit any of the options once launched, the **Select** button can be pressed.

In the next sections will be a more detailed description of the working of these options:

Clock:

The clock uses the RTC module to keep track of real time, even when the display is turned off, so that the time does not have to be set every time the device is turned on. The Arduino reads the actual time from this device and displays it on the matrix screen in both analogue and digital form. The analogue clock is made of a red circle, representing the dial. A blue line indicates the minutes on the clock, and the smaller green line points to the hour. The digital clock displays a two digit seven-segment number for both the hours and the minutes, the hours being on the top left and the minutes on the bottom right. The colours of the digits are the same as the colours of the analogue indicators.

To set the clock, the “x” (cross) button on the controller can be pressed. The ring of the clock will now turn to yellow, indicating that the “**Set Time**” mode has been enabled. Now with the **square** button, the minutes can be set at increments of 1 minute, or at increments of 5 minutes using the **R1** button. Using the **triangle** button the hours can be set. To save the set time to the RTC module the **start** button must be pressed, or to cancel setting the time, and returning to the clock screen without making changes, the **circle** button can be pressed.

Tetris:

The classic game. Create full horizontal lines to score points. The more lines that are filled in at the same time, the more points that are scored. (one line = 100 points, 2 lines = 250 points (125 points per line), 3 lines = 400 points (~133 points per line) and 4 lines = 600 points (150 points per line))

To move the block left or right, use the left and right arrows on the gamepad. To make it go down faster the down-arrow can be pressed. With either **cross** or **circle** the block can be rotated.

It's game over when the pile of blocks reaches the top of the screen.

Pong:

Pong is one of the oldest games ever created. Move the bar side to side with the arrow buttons and try to hit the bouncing ball back to your opponent.

Appendix 1: abbreviations

abbreviation	meaning
RGB	“Red Green Blue”, The three primary colours, used in electronics to be able to make full-range lights
LED	“Light-emitting diode”, A device that can turn electric energy into light.
SPI	“Serial Peripheral Interface” A synchronous serial bus protocol using 3 wires for full duplex communication and a separate hardware line for device selection.
MOSI	“Master Out Slave In” a term used in SPI to describe the data line where the processor sends data and the peripheral receives data
MISO	“Master In Slave Out” a term used in SPI to describe the data line where the peripheral sends data and the master receives data
I ² C/IIC	“Inter-IC Communication” A synchronous serial bus protocol using 2 wires, half duplex communication and in-line addresses for device selection
TWI	“Two Wire Interface”, a variation of I ² C developed by Microchip Technology
UART	“Universal Asynchronous Receiver Transmitter”, an asynchronous serial data protocol between 2 devices with full duplex communication.
AVR	A series of RISC-based microprocessors developed by Atmel™
RISC	“Reduced Instruction Set Computer” a computer instruction set with a low amount of cycles per instruction.
(S)RAM	“Random Access Memory” A computer’s “short-term memory” used for quickly reading and writing data in a computer
PWM	“Pulse Width Modulation” Changing the width of a pulse in a fixed frequency to limit the amount of power used. Also used to digitally recreate analog values when combined with a low pass filter
ADC	“Analog to Digital Converter” A device to convert an analog signal/voltage into a digital value to be read by a microprocessor or other digital device
GPIO	“General Purpose Input/Output” a pin on a computer/processor that can be used to connect to any other device or peripheral (hence the “general purpose”).

SMD/SMT	“Surface Mount Device/Technology”, a process where components are soldered directly to the <i>surface</i> of a circuit board, instead of through a hole.
IDE	“Integrated Development Environment” a computer program used to develop software
PS2	“PlayStation 2” a videogame console made by Sony Entertainment™
IC	“Integrated Circuit” An electronic circuit integrated in a small package
RTC	“Real Time Clock” An electronic device capable of keeping track of time very precisely

Appendix 2: Main Code

```
/*
    Multiple games for on Arexx Game Board
    Author: Sjors Smit
    Date: June 2nd 2020
    AREXX ENGINEERING
*/

//used libraries:
#include <Adafruit_GFX.h>
#include <Adafruit_NeoMatrix.h>
#include <Adafruit_NeoPixel.h>
#include <PS2X_lib.h>
#include <Max72xxPanel.h>

//Amount of selectable items in the main menu (zero indexed!)
#define gameAmount 2

//pins for PS2 controller
#define PS2_CMD    7
#define PS2_DAT    8
#define PS2_CLK    9
#define PS2_SEL    10

// Color definitions
#define BLACK      0x0000 //turns a pixel off
#define BLUE       0x001F
#define RED        0xF800
#define GREEN      0x07E0
#define DARKGREEN  0x0362
#define CYAN       0x07FF
#define MAGENTA    0xF81F
#define YELLOW     0xFFE0
#define ORANGE     matrix.Color(0xff, 0xA5, 0x00)
#define WHITE      0xFFFF
#define BOARDCOLOUR matrix.Color(0x00, 0xCE,0xBF) //The colour a pixel will turn into once it has had a bottom collision in Tetris

//Starting position:
int8 t Xpos = 3;
```