Matrice de neopixeli cu animatii

1. Parte Teoretica

Un prim pas in realizarea proiectului a fost configurarea animatiilor pe care dorim sa le vizualizam. Astfel, folosind programul PixelFontEdit 2.7 am trasat diversele stari prin care fiecare animatie ar trebui sa treaca pentru a fi mai usor sa scriem codul corespunzator.

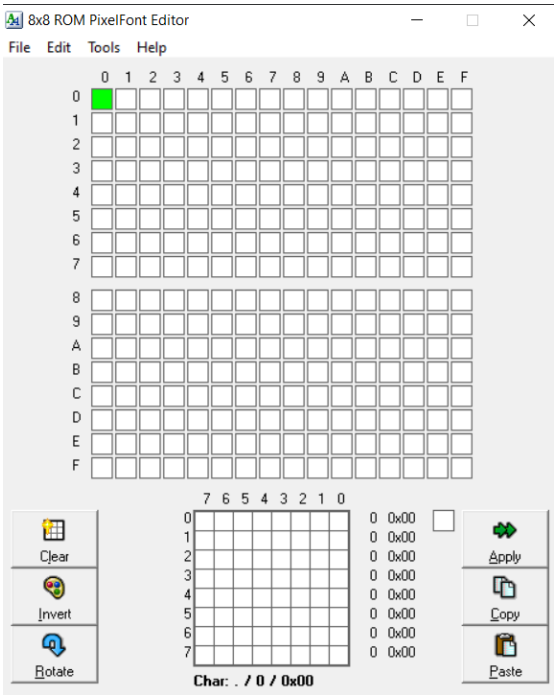

Figure 1 Pixel Font Edit 2.7

Folosind acest editor am desenat astfel 2 dintre cele 4 animatii existente, etapele specificate fiind urmatoarele:

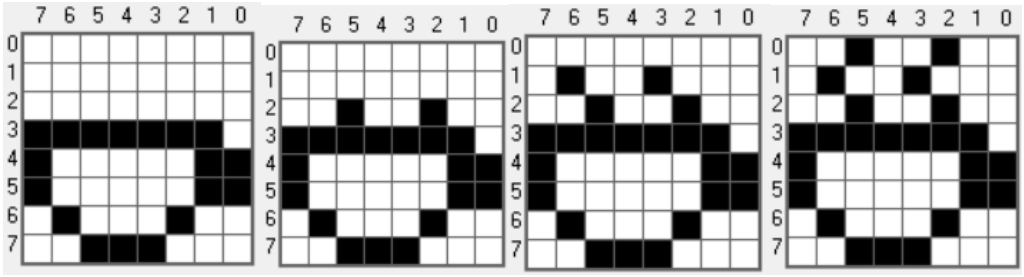

Figure 2 Animatia ,,Coffee,,

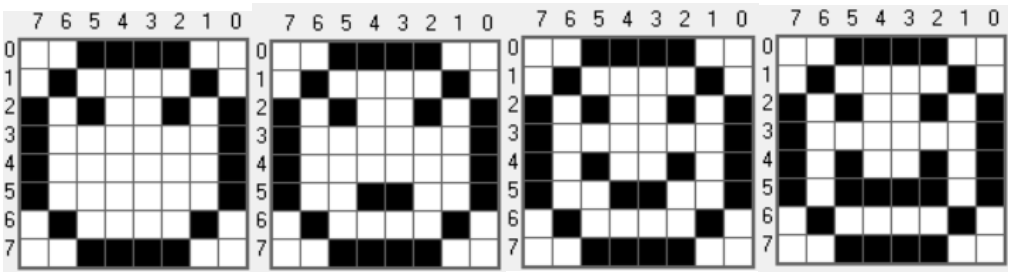

Figure 3 Animatia ,,Happy,,

Implementarea primelor 2 animatii este facilitata de folosirea librariei Adafruit_NeoMatrix, sustinuta de Adafruit_GFX si Adafruit_NeoPixel deoarece matricea noastra este in esenta o banda de neopixeli inlantuita. Acestea ajuta la creerea unui display grafic bidimensional, avand primitive care ne scutesc de adresarea individuala a fiecarui pixel a matricii, putand folosii functiile de desenare de forme. In desenarea animatiilor cele mai uzuale functii folosite de mine au fost drawLine-pentru desenarea liniilor din animatii, drawPixel-pentru desenarea unui pixel individual, fillScreen(0)-pentru oprirea tuturor pixelilor la tranzitiile dintre etapele animatiilor si show-pentru afisarea pe matrice la fiecare pas a schimbarilor efectuate.[2]

Dificultatea alegerii coordonatelor x si y ale punctelor au constat in faptul ca matricea utilizata de mine are numerotare inversa pentru linii pare, respectiv impare, conform figurii, reprezentand astfel un impediment pentru desenarea liniilor pe coloane.

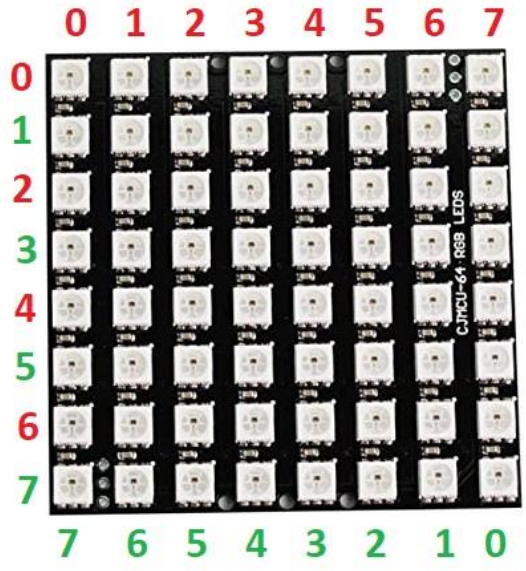

Figure 4 Orientarea ledurilor matricii

Odata cunoscute sabloanele pentru etapele animatiei si inteleasa utilizarea corecta a matricii, specificarea animatiilor s-a efectuat fluid conform urmatoarelor tipare:

Animatia 1 “Coffee” – apeleaza cele fiecare dintre cele patru sabloane care reprezinta tranzitiile animatiei la interval de jumatate de secunda, intre fiecare apel fiind folosite functiile fillScreen(0) si show() pentru o afisare corecta.

Animatia 2 “Happy” – apeleaza cele fiecare dintre cele sase sabloane care reprezinta tranzitiile animatiei la interval de jumatate de secunda, intre fiecare apel fiind folosite functiile fillScreen(0) si show() pentru o afisare corecta.

Animatia 3 “Dots” – aceasta animatie a fost aleasa din considerente estetice deoarece succesiunea repetata a activarii a cate treizeci de pixeli (prin buclele for) la pozitie si cu culoare aleatoare.

Animatia 4 “Line” – scopul acestei animatii este de a valorifica si functiile puse la dispozitie de libraria FastLed. Pentru fiecare dintre cei saizeci si patru adresati individual setam culoarea cu o nuanta hue incrementata la fiecare pas, precum si Saturation si Value maxime. Dupa setarea tuturor ledurilor se reface procesul in ordine inversa. FastLed dispune de un spectru mult mai bogat de culori si mai optim, fiecare componenta avand valori intre 0-255, generand astfel un cod mai rapid si mai compact. Matricea este considerata de fapt o banda de neopixeli continua. [3]

Fiecare pixel este colorat corespunzator pozitiile fiind specificate conform figurii:

Figure 5 Pozitiile neopixelilor

Avand, asadar, toate cele 4 animatii pregatite urmatoarea etapa este de a realiza conexiunea la modulul bluetooth Bluefruit LE si de a pregati mediul.[1]

In cadrul functiei de setup() se initializeaza starea matricii si a sirului de leduri FastPixels, setandu-le totodata luminozitatea. Ulterior, se aprind toate ledurile pentru a indica finalizarea incarcarii pe placa, apoi se activeaza comunicarea seriala cu baud rate 115200 pentru a fi compatibil cu modulul.

Configurarea modulului bluetooth Bluefruit LE declarat inainte de setup() incepe prin initializarea acestuia (daca modulul nu este setat in modul Command initializarea esueaza si se afiseaza un mesaj de eroare, iar executia este oprita). Ulterior se efectueaza o resetare din fabrica a modulului pentru a ne asigura ca modulul este intr-o stare consistenta, stergand setarile care persistau de la utilizarile anterioare. In cazul unui esec se afiseaza un mesaj corespunzator si se stopeaza continuarea, programul necesitand sa fie reincarcat. Suplimentar se pot afisa informatii despre modulul utilizat apeland functia info(). Pasul urmator este de a ne conecta la modul folosind aplicatia mobila Bluefruit Connect disponibila in Magazin Play sau AppStore. Se asteapta pana cand se efectueaza conectarea, dupa care se poate trece in modul DATA, folosind functionalitatile de tip Controller si Color Picker. Pe parcursul efectuarii acestor operatii in interfata seriala au fost scrise mesaje ajutatoare care indica in ce pas ne aflam.

In cadrul functiei loop() se citesc informatiile primite de la modulul Bluefruit, folosind functia readPacket(&ble, BLE_READPACKET_TIMEOUT) preluata din clasa packetParser preimplementata (vezi Anexa A). In functie de packetbuffer[1] stabilim daca am folosit Color Picker sau controlul butoanelor. In cazul in care acesta este ‘C’ inseamna ca am folosit Color Picker asa ca citim valorile RGB si le afisam pe ecran.

Daca acesta este ‘B’ atunci a fost apasat un buton din cadrul lui Controller, asa ca citim numarul butonului si verificam daca este apasat, afisand un mesaj corespunzator. Starea animatiei se schimba in functie de numarul butonului apasat, pentru fiecare dintre cele 4 posibile cazuri apeland animatia specificata.

Toate aceste functionalitati ale modulului Bluefruit au putut fi folosite utilizand librariile Adafruit_BLE si Adafruit_BluefruitLE_UART, precum si clasa de parsare specificata in cadrul anexei A, permitandu-ne astfel sa realizam conexiunea, sa citim, prelucram si afisam datele pentru a alege animatia si culoarea acesteia intr-un mod mult mai interactiv, gratie interfetei grafice oferite de aplicatia Bluefruit Connect.

2. Circuit electric

Deoarece nu am gasit toate componentele utilizate in simulatorul Proteus/Tinkercad, am realizat schema electrica folosing programul Fritzing.

In cadrul acestui proiect am folosit 3 componente:

- Placa Arduino Mega 2560: necesara incarcarii programului si alimentarii celorlalte doua componente
- Modul Bluefruit LE Uart Friend: necesar folosirii aplicatiei Bluefruit Connect pentru a trimite prin bluetooth datele dorite
- Matricea 8x8 de neopixeli WS28112B-64: necesara afisarii animatiilor

Modul in care acestea se conecteaza este surprins in schema urmatoare:

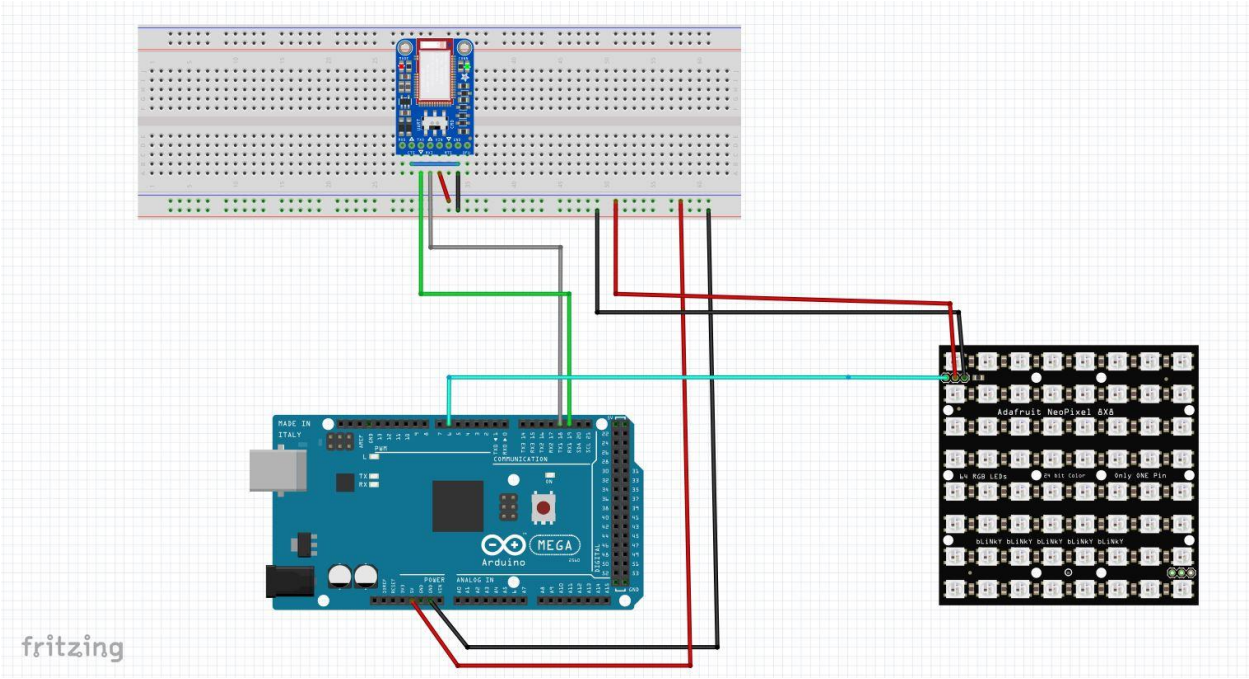

Figure 6 Conectarea componentelor

3. Arduino IDE

```
#include <Adafruit_NeoPixel.h>
#include <Adafruit_GFX.h>
#include <Adafruit_NeoMatrix.h>
#include <FastLED.h>
#define PIN 6

#include <string.h>
#include <Arduino.h>
#include <SPI.h>
#include <SoftwareSerial.h>

#include "Adafruit_BLE.h"
#include "Adafruit_BluefruitLE_SPI.h"
#include "Adafruit_BluefruitLE_UART.h"

#include "BluefruitConfig.h"

#define FACTORYRESET_ENABLE     1
#define NUMPIXELS      64

//declaram sirul de pixeli folosit cu libraria FastLed
CRGB fastPixels[NUMPIXELS];

//declaram matricea de 8x8
Adafruit_NeoMatrix matrix = Adafruit_NeoMatrix(8, 8, PIN,
                    NEO_MATRIX_TOP     + NEO_MATRIX_LEFT +
                    NEO_MATRIX_COLUMNS + NEO_MATRIX_ZIGZAG,
                    NEO_GRB            + NEO_KHZ800);

//declaram modululbluetooth utilizat
Adafruit_BluefruitLE_UART                              ble(BLUEFRUIT_HWSERIAL_NAME,
BLUEFRUIT_UART_MODE_PIN);

//afisam eroarea si ramanem in bucla infinita pentru a nu se continua cat timp nu este remediata
void error(const __FlashStringHelper*err) {
  Serial.println(err);
  while (1);
}
```

```
// prototipul functiilor din packetparser.cpp utilizate pentru parsarea informatiilor primite prin bluetooth
uint8_t readPacket(Adafruit_BLE *ble, uint16_t timeout);
float parsefloat(uint8_t *buffer);
void printHex(const uint8_t * data, const uint32_t numBytes);

// bufferul in care pastram informatiile primite
extern uint8_t packetbuffer[];

//culoare predefinita
uint8_t red = 100;
uint8_t green = 100;
uint8_t blue = 100;

//starea de inceput
uint8_t animationState = 1;


void setup(void)
{

  //adaugam ledurile FastPixels
  LEDS.addLeds<WS2812,PIN,RGB>(fastPixels, NUMPIXELS);
  LEDS.setBrightness(80);

  //initializam starea matricii
  matrix.begin();
  matrix.setBrightness(80);

  //matricea se aprinde ca intampinare
  matrix.fillScreen(matrix.Color(red, green, blue));
  delay(1000);
  matrix.show();

//activam comunicarea seriala
  Serial.begin(115200);
  Serial.println("Proiect: Matrice de neopixeli cu animatii");
  Serial.println("-----------------------------------------------");


  Serial.print("Initializam modulul Bluefruit LE: ");

 //pentru a se putea initializa modulul acesta trebuie sa fie setat in modul Cmd
 //daca intervine o eroare programul se opreste
  if ( !ble.begin(VERBOSE_MODE) )
  {
```

```
    error(F("Modulul nu a fost gasit, verificati daca acesta este setat in Command Mode precum si firele!"));
  }
  Serial.println( F("Modul initializat!") );

 //se recomanda efectuarea unui reset din fabrica pentru a ne asigura ca modulul este intr-o stare consistenta, stergand setarile care persistau de la utilizarile anterioare
  if ( FACTORYRESET_ENABLE )
  {

    Serial.println("Se efectueaza resetarea din fabrica");
    if ( ! ble.factoryReset() ){
      error(F("Eroare la incercarea resetarii, va rugam inchideti fereastra si reincercati!"));
    }
  }
//dezactivam comanda echo a modulului deoarece nu o utilizam
  ble.echo(false);

  Serial.println("Informatii despre Bluefruit:");

  ble.info();

  Serial.println("Porniti aplicatia Bluefruit Connect si accesati Controller");


//setat pe false pentru a ajuta la efectuarea unui eventual debug
  ble.verbose(false);

//asteptam pana cand aplicatia mobila se conecteaza la modulul Bluefruit
  while (! ble.isConnected()) {
      delay(500);
  }

  Serial.println(F("-------------------------------------"));
  Serial.println("Acum puteti folosi color picker/controller-ul!");

  // modulul Bluefruit trebuie setat in Data mode pentru a prelua informatii

  Serial.println( F("Setati modulul in modul Data,folosind switch-ulacestuia!") );
  ble.setMode(BLUEFRUIT_MODE_DATA);

  Serial.println(F("-------------------------------------"));
}

void loop(void)
{
```

```
//citim informatiile primite de la modulul Bluefruit
uint8_t len = readPacket(&ble, BLE_READPACKET_TIMEOUT);

//in functie de packetbuffer[1] stabilim daca am folosit color picker sau controlul butoanelor
//culoare
if (packetbuffer[1] == 'C') {

  //daca am primit o culoare citim valorile RGB si le afisam pe ecran
  red = packetbuffer[2];
  green = packetbuffer[3];
  blue = packetbuffer[4];
  Serial.print ("RGB #");
  if (red < 0x10) Serial.print("0");
  Serial.print(red, HEX);
  if (green < 0x10) Serial.print("0");
  Serial.print(green, HEX);
  if (blue < 0x10) Serial.print("0");
  Serial.println(blue, HEX);
}

// butoane
if (packetbuffer[1] == 'B') {

  //daca am accesat un buton citim numarul acestuia si verificam cand este apasat
  uint8_t number = packetbuffer[2] - '0';
  boolean pressed = packetbuffer[3] - '0';
  Serial.print("Button ");
  Serial.print(number);

  //starea animatiei se schimba in functie de numarul butonului apasat
  animationState = number;

  if (pressed) {
    Serial.println(" pressed");
  } else {
    Serial.println(" released");
  }
}
  //daca am ajuns aici inseamna ca se cunoaste atat animatia dorita cat si culoarea

  //animatia butonului 1 in forma de ceasca de cafea si culoare primita de la modul
    if (animationState == 1){
      matrix.fillScreen(0);
      coffeeAnimation(matrix.Color(red, green, blue));
      matrix.show();
    }
```

```
   //animatia butonului 2 in forma de expresie faciala (fericire) si culoare primita de la modul
    if (animationState == 2){
      matrix.fillScreen(0);
      happyAnimation(matrix.Color(red, green, blue));
      matrix.show();
    }

    //animatia butonului 3 (sageata in jos) afiseaza multiple puncte cu culori generate aleator
    if (animationState == 3){
      matrix.fillScreen(0);
      dotsAnimation();
      matrix.show();
    }

     //animatia butonului 4 (sageata la stanga)foloseste libraria FastLed si coloreaza succesiv liniile
matricii
    if (animationState == 4){
      lineAnimation();
    }


}


//functii folosite la crearea animatiei de tip cana de cafea
void Coffee1(uint32_t c) {
  matrix.drawLine(3, 1, 3, 7, c);
  matrix.drawLine(4, 0, 5, 0, c);
  matrix.drawLine(3, 7, 5, 7, c);
  matrix.drawLine(7, 3, 7, 5, c);
  matrix.drawPixel(6, 1, c);
  matrix.drawPixel(6, 5, c);
  matrix.drawPixel(4, 6, c);
  matrix.drawPixel(5, 1, c);
}

void Coffee2(uint32_t c) {
  matrix.drawLine(3, 1, 3, 7, c);
  matrix.drawLine(4, 0, 5, 0, c);
  matrix.drawLine(3, 7, 5, 7, c);
  matrix.drawLine(7, 3, 7, 5, c);
  matrix.drawPixel(6, 1, c);
  matrix.drawPixel(6, 5, c);
  matrix.drawPixel(4, 6, c);
  matrix.drawPixel(5, 1, c);
  matrix.drawPixel(2, 2, c);
```

```
  matrix.drawPixel(2, 5, c);
}
void Coffee3(uint32_t c) {
  matrix.drawLine(3, 1, 3, 7, c);
  matrix.drawLine(4, 0, 5, 0, c);
  matrix.drawLine(3, 7, 5, 7, c);
  matrix.drawLine(7, 3, 7, 5, c);
  matrix.drawPixel(6, 1, c);
  matrix.drawPixel(6, 5, c);
  matrix.drawPixel(4, 6, c);
  matrix.drawPixel(5, 1, c);
  matrix.drawPixel(2, 2, c);
  matrix.drawPixel(2, 5, c);
  matrix.drawPixel(1, 3, c);
  matrix.drawPixel(1, 6, c);
}

void Coffee4(uint32_t c) {
  matrix.drawLine(3, 1, 3, 7, c);
  matrix.drawLine(4, 0, 5, 0, c);
  matrix.drawLine(3, 7, 5, 7, c);
  matrix.drawLine(7, 3, 7, 5, c);
  matrix.drawPixel(6, 1, c);
  matrix.drawPixel(6, 5, c);
  matrix.drawPixel(4, 6, c);
  matrix.drawPixel(5, 1, c);
  matrix.drawPixel(2, 2, c);
  matrix.drawPixel(2, 5, c);
  matrix.drawPixel(1, 3, c);
  matrix.drawPixel(1, 6, c);
  matrix.drawPixel(0, 2, c);
  matrix.drawPixel(0, 5, c);
}

//animatia 1: la fiecare jumatate de secunda se afiseaza cate o etapa a animatiei
void coffeeAnimation(uint32_t c) {
  matrix.fillScreen(0);
  Coffee1(c);
  matrix.show();
  delay(500);
  matrix.fillScreen(0);
  Coffee2(c);
  matrix.show();
  delay(500);
  matrix.fillScreen(0);
  Coffee3(c);
```

```
  matrix.show();
  delay(500);
  matrix.fillScreen(0);
  Coffee4(c);
  matrix.show();
  delay(500);
}

//functii folosite la crearea animatiei de tip expresie faciala
void Face1(uint32_t c) {
  matrix.drawLine(0, 2, 0, 5, c);
  matrix.drawLine(7, 2, 7, 5, c);
  matrix.drawLine(2, 7, 3, 7, c);
  matrix.drawLine(2, 0, 3, 0, c);
  matrix.drawLine(4, 0, 5, 0, c);
  matrix.drawLine(5, 7, 4, 7, c);
  matrix.drawPixel(1, 1, c);
  matrix.drawPixel(1, 6, c);
  matrix.drawPixel(2, 2, c);
  matrix.drawPixel(2, 5, c);
  matrix.drawPixel(6, 1, c);
  matrix.drawPixel(6, 6, c);
}

void Face2(uint32_t c) {
  matrix.drawLine(0, 2, 0, 5, c);
  matrix.drawLine(7, 2, 7, 5, c);
  matrix.drawLine(2, 7, 3, 7, c);
  matrix.drawLine(2, 0, 3, 0, c);
  matrix.drawLine(4, 0, 5, 0, c);
  matrix.drawLine(5, 7, 4, 7, c);
  matrix.drawPixel(1, 1, c);
  matrix.drawPixel(1, 6, c);
  matrix.drawPixel(2, 2, c);
  matrix.drawPixel(2, 5, c);
  matrix.drawPixel(6, 1, c);
  matrix.drawPixel(6, 6, c);
  matrix.drawLine(5, 3, 5, 4, c);
}

void Face3(uint32_t c) {
  matrix.drawLine(0, 2, 0, 5, c);
  matrix.drawLine(7, 2, 7, 5, c);
  matrix.drawLine(2, 7, 3, 7, c);
  matrix.drawLine(2, 0, 3, 0, c);
  matrix.drawLine(4, 0, 5, 0, c);
```

```
  matrix.drawLine(5, 7, 4, 7, c);
  matrix.drawPixel(1, 1, c);
  matrix.drawPixel(1, 6, c);
  matrix.drawPixel(2, 2, c);
  matrix.drawPixel(2, 5, c);
  matrix.drawPixel(6, 1, c);
  matrix.drawPixel(6, 6, c);
  matrix.drawLine(5, 3, 5, 4, c);
  matrix.drawPixel(4, 2, c);
  matrix.drawPixel(4, 5, c);
}

void Face4(uint32_t c) {
  matrix.drawLine(0, 2, 0, 5, c);
  matrix.drawLine(7, 2, 7, 5, c);
  matrix.drawLine(2, 7, 3, 7, c);
  matrix.drawLine(2, 0, 3, 0, c);
  matrix.drawLine(4, 0, 5, 0, c);
  matrix.drawLine(5, 7, 4, 7, c);
  matrix.drawPixel(1, 1, c);
  matrix.drawPixel(1, 6, c);
  matrix.drawPixel(2, 2, c);
  matrix.drawPixel(2, 5, c);
  matrix.drawPixel(6, 1, c);
  matrix.drawPixel(6, 6, c);
  matrix.drawLine(5, 3, 5, 4, c);
  matrix.drawPixel(4, 2, c);
  matrix.drawPixel(4, 5, c);
  matrix.drawPixel(5, 2, c);
  matrix.drawPixel(5, 5, c);
}

//animatia 2: folosita la afisarea unei fete care incepe sa zambeasca
void happyAnimation(uint32_t c) {
  matrix.fillScreen(0);
  Face1(c);
  matrix.show();
  delay(500);
  matrix.fillScreen(0);
  Face2(c);
  matrix.show();
  delay(500);
  matrix.fillScreen(0);
  Face3(c);
  matrix.show();
  delay(500);
```

```
  matrix.fillScreen(0);
  Face4(c);
  matrix.show();
  delay(500);
}
//animatia 3: de tip puncte genereaza aprinderea ledurilor intr-o culoare si la pozitie aleatoare
void dotsAnimation() {
  for(int j = 0; j < 8; j++)
  for(int i = 0 ; i < 64 ; i++){
      int randX = random(0, 8);
      int randY = random(0, 8);
      int randR1 = random(0, 255);
      int randG1 = random(0, 255);
      int randB1 = random(0, 255);
      matrix.drawPixel(randX, randY, matrix.Color(randR1, randG1, randB1));
  }
}

//animatia 4
void lineAnimation(){
  int hue = 0;
  //pentru fiecare led din cei NUMPIXELS existenti setam culoarea si incrementam nuanta pentru
configurarea unui efect de trecere gradata
  for(int i = 0; i < NUMPIXELS; i++) {
    //seteaza culoarea ledului i
    fastPixels[i] = CHSV(hue++, 255, 255);
    //afiseaza schimbarile ledurilor
    FastLED.show();
    delay(10);
  }

  //dupa setarea tuturor ledurilor se reincepe in ordine inversa
  for(int i = NUMPIXELS - 1; i >= 0; i--) {
    fastPixels[i] = CHSV(hue++, 255, 255);
    FastLED.show();
    delay(10);
  }
}
```

4. Implementare reala

Pentru a utiliza corect acest proiect se recomanda efectuarea urmatorilor pasi:

- Setam modulul Bluefruit in modul Command si incarcam programul pe placa
- Dupa incarcarea pe placa si confirmarea efectuarii resetarii din fabrica se porneste aplicatia Bluefruit Connect de pe telefonul mobil si se apasa butonul “Connect” pentru conectarea la modul

- Modulul Bluefruit se seteaza in modul Data pentru a putea prelua informatiile, dupa care se pot folosi Controller si Color Picker

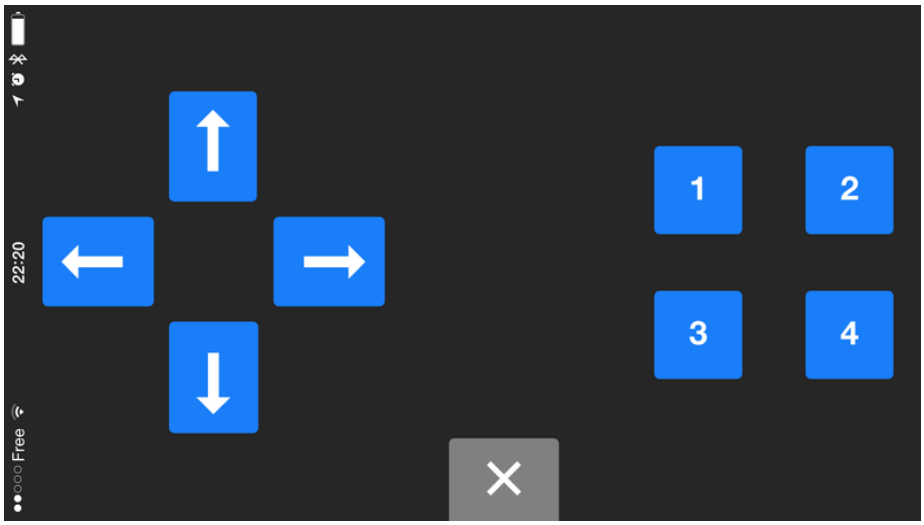

Figure 7 Control Pad

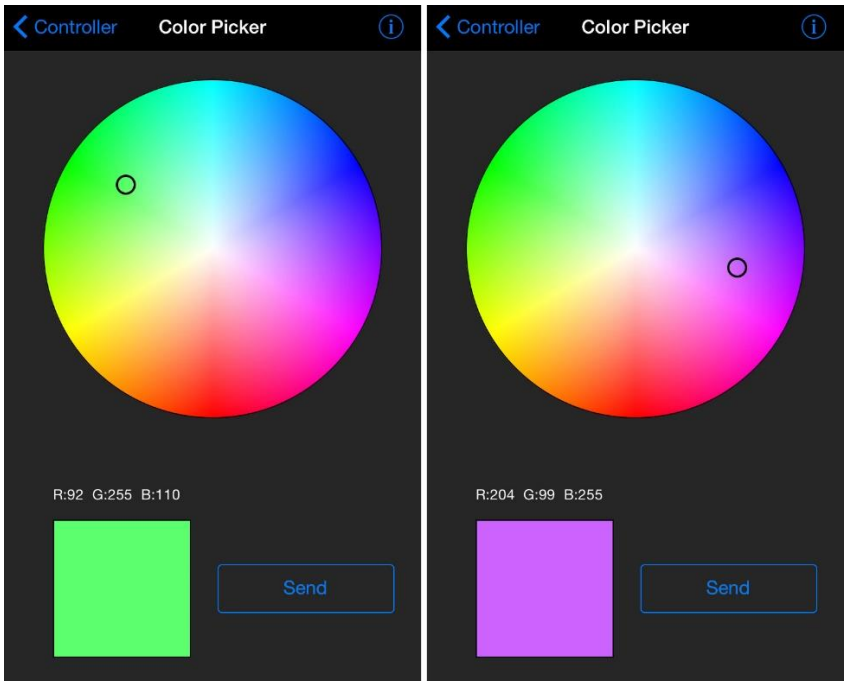

Figure 8 Color Picker

- Culorile primelor cinci animatii sunt configurabile prin trimiterea unei nuante alese in Color Picker
- Animatia poate fi schimbata folosind butoanele astfel:
    - Butonul 1: Animatia “Coffee”

    - Butonul 2: Animatia “Happy”
    - Butonul 3: Animatia “Dots”
    - Butonul 4: Animatia “Lines”
- Odata aleasa animatia si culoarea dorita aceasta va fi afisata pe matrice pana la noi modificari

Rezultatele obtinute sunt in urmatoarele imagini:

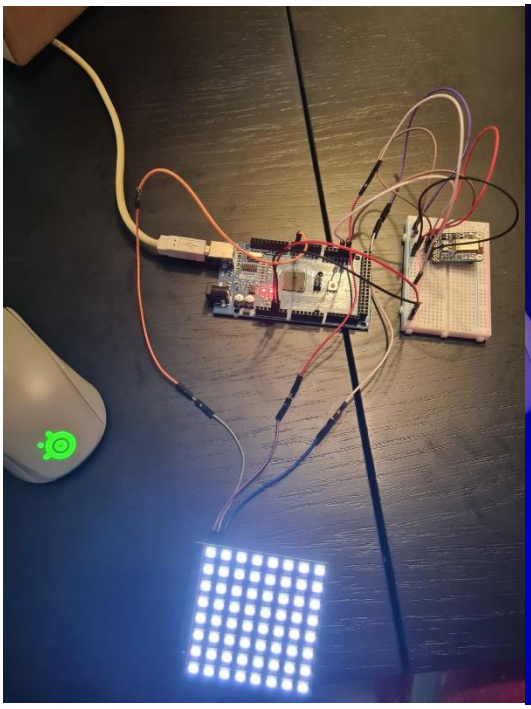

Figure 9 Confirmarea incarcarii pe placa

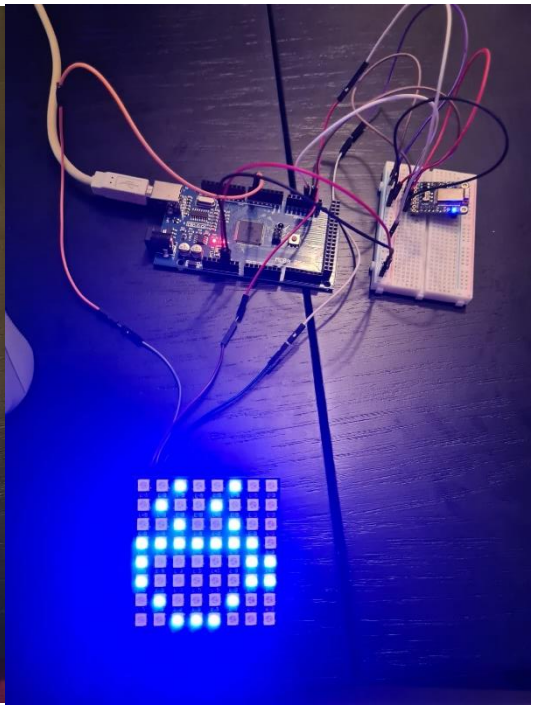

Figure 10 Prima animatie

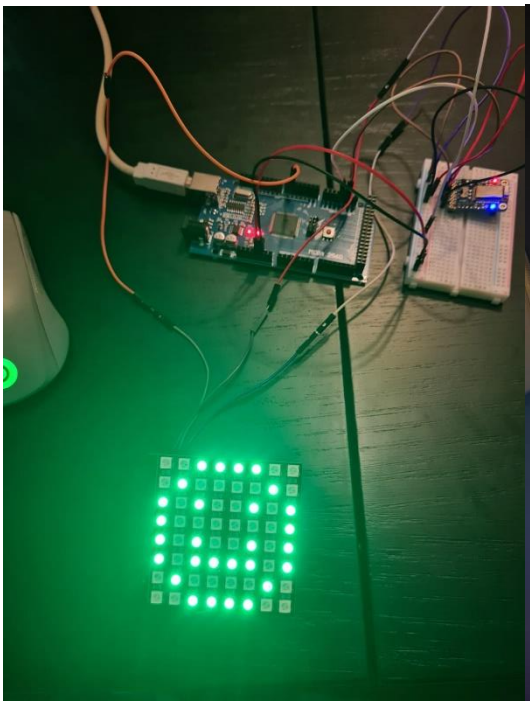

Figure 11 A doua animatie

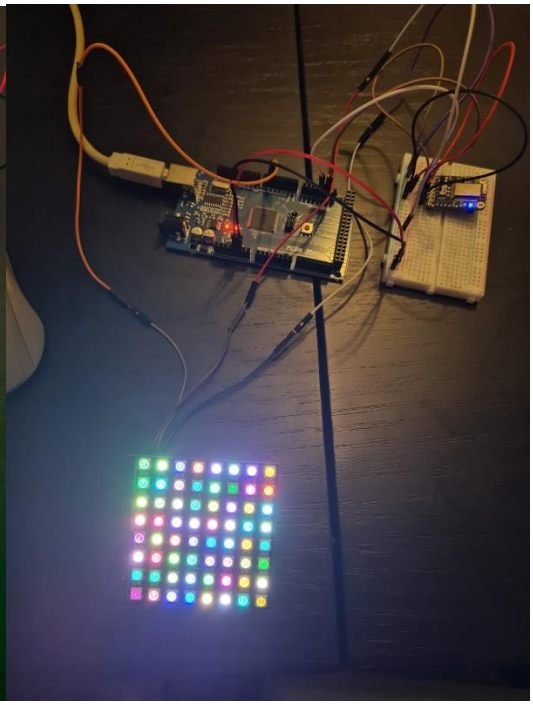

Figure 22 A treia animatie

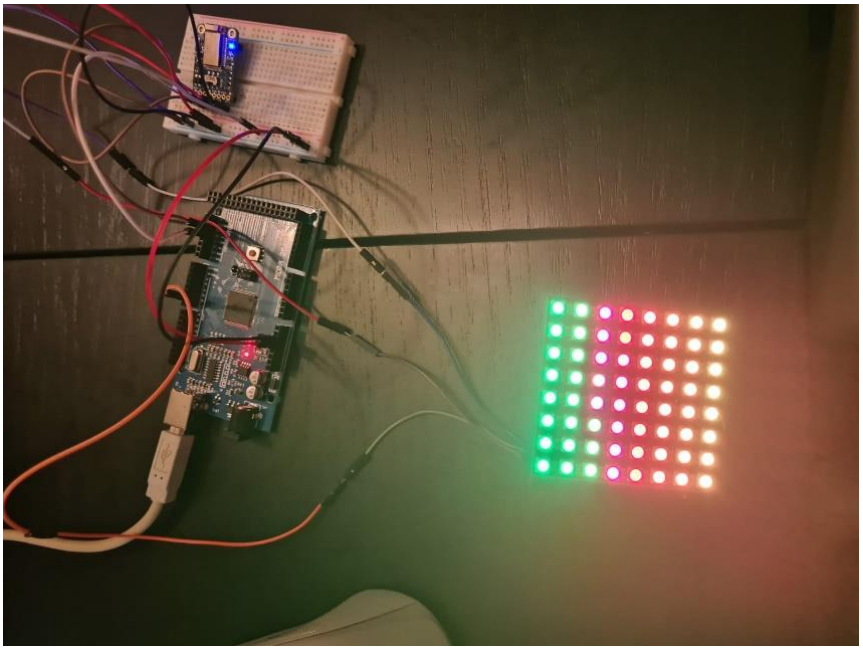

Figure 33 A patra animatie

5. Video

Mai jos am inserat link-ul catre folderul unde este atasat video-ul cu demonstratia implementarii si resursele aditionale folosite(BluefruitConfig.h si packetParser.cpp)

https://drive.google.com/drive/folders/16BrIdUIVixHgSLC5V_RK5kYWb9PR-KcP?usp=sharing

Bibliografie

[1] https://learn.adafruit.com/introducing-the-adafruit-bluefruit-le-uart-friend?view=all

[2] https://learn.adafruit.com/adafruit-neopixel-uberguide/neomatrix-library

[3] https://blog.claytonk.com/2019/10/16/working-with-fastled-library/

[4] https://learn.adafruit.com/neopixel-matrix-snowflake-sweater?view=all

Anexe

Anexa A
Clasa PacketParser

```
#include <string.h>
#include <Arduino.h>
#include <SPI.h>
#if not defined (_VARIANT_ARDUINO_DUE_X_) && not defined
(_VARIANT_ARDUINO_ZERO_) && not defined(__SAMD51__)
  #include <SoftwareSerial.h>
#endif

#include "Adafruit_BLE.h"
#include "Adafruit_BluefruitLE_SPI.h"
```

```
#include "Adafruit_BluefruitLE_UART.h"


#define PACKET_ACC_LEN             (15)
#define PACKET_GYRO_LEN            (15)
#define PACKET_MAG_LEN             (15)
#define PACKET_QUAT_LEN            (19)
#define PACKET_BUTTON_LEN           (5)
#define PACKET_COLOR_LEN            (6)
#define PACKET_LOCATION_LEN        (15)

//    READ_BUFSIZE            Size of the read buffer for incoming packets
#define READ_BUFSIZE                (20)


/* Buffer to hold incoming characters */
uint8_t packetbuffer[READ_BUFSIZE+1];

/**************************************************************************
***/
/*!
    @brief  Casts the four bytes at the specified address to a float
*/
/**************************************************************************
***/
float parsefloat(uint8_t *buffer)
{
  float f;
  memcpy(&f, buffer, 4);
  return f;
}

/**************************************************************************
***/
/*!
    @brief  Prints a hexadecimal value in plain characters
    @param  data      Pointer to the byte data
    @param  numBytes  Data length in bytes
*/
/**************************************************************************
***/
void printHex(const uint8_t * data, const uint32_t numBytes)
{
  uint32_t szPos;
  for (szPos=0; szPos < numBytes; szPos++)
  {
```

```
    Serial.print(F("0x"));
    // Append leading 0 for small values
    if (data[szPos] <= 0xF)
    {
      Serial.print(F("0"));
      Serial.print(data[szPos] & 0xf, HEX);
    }
    else
    {
      Serial.print(data[szPos] & 0xff, HEX);
    }
    // Add a trailing space if appropriate
    if ((numBytes > 1) && (szPos != numBytes - 1))
    {
      Serial.print(F(" "));
    }
  }
  Serial.println();
}

/**************************************************************************
***/
/*!
    @brief  Waits for incoming data and parses it
*/
/**************************************************************************
***/
uint8_t readPacket(Adafruit_BLE *ble, uint16_t timeout)
{
  uint16_t origtimeout = timeout, replyidx = 0;

  memset(packetbuffer, 0, READ_BUFSIZE);

  while (timeout--) {
    if (replyidx >= 20) break;
    if ((packetbuffer[1] == 'A') && (replyidx == PACKET_ACC_LEN))
      break;
    if ((packetbuffer[1] == 'G') && (replyidx == PACKET_GYRO_LEN))
      break;
    if ((packetbuffer[1] == 'M') && (replyidx == PACKET_MAG_LEN))
      break;
    if ((packetbuffer[1] == 'Q') && (replyidx == PACKET_QUAT_LEN))
      break;
    if ((packetbuffer[1] == 'B') && (replyidx == PACKET_BUTTON_LEN))
      break;
    if ((packetbuffer[1] == 'C') && (replyidx == PACKET_COLOR_LEN))
```

```
    break;
  if ((packetbuffer[1] == 'L') && (replyidx == PACKET_LOCATION_LEN))
    break;

  while (ble->available()) {
    char c =  ble->read();
    if (c == '!') {
      replyidx = 0;
    }
    packetbuffer[replyidx] = c;
    replyidx++;
    timeout = origtimeout;
  }

  if (timeout == 0) break;
  delay(1);
 }

 packetbuffer[replyidx] = 0;  // null term

 if (!replyidx)  // no data or timeout
  return 0;
 if (packetbuffer[0] != '!')  // doesn't start with '!' packet beginning
  return 0;

 // check checksum!
 uint8_t xsum = 0;
 uint8_t checksum = packetbuffer[replyidx-1];

 for (uint8_t i=0; i<replyidx-1; i++) {
  xsum += packetbuffer[i];
 }
 xsum = ~xsum;

 // Throw an error message if the checksum's don't match
 if (xsum != checksum)
 {
  Serial.print("Checksum mismatch in packet : ");
  printHex(packetbuffer, replyidx+1);
  return 0;
 }

 // checksum passed!
 return replyidx;
}
```