Laboratorium z przedmiotu Systemy wbudowane (SW)

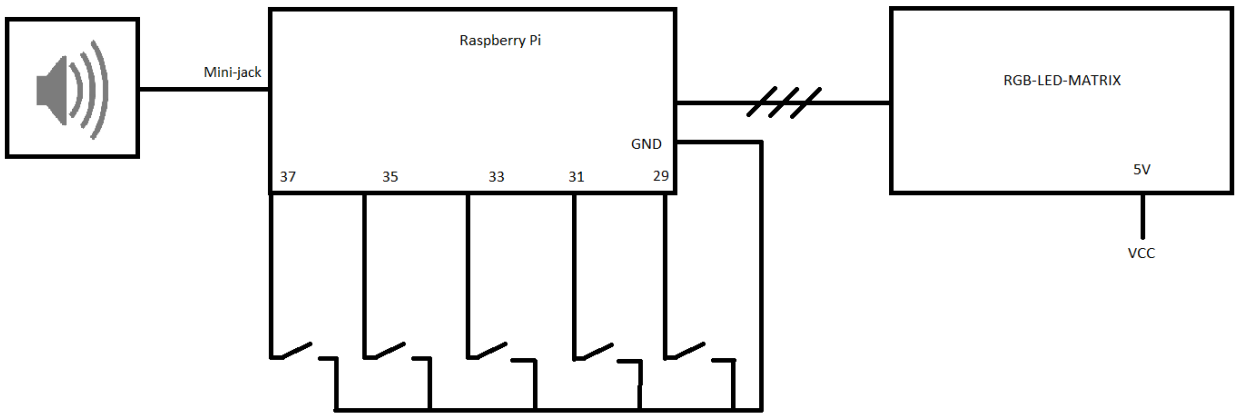
Karta projektu – zadanie 7

Nazwa projektu: Guitar Hero

Prowadzący:	*Autorzy (tylko nr indeksu):*	*Grupa dziekańska:*	I5.2
mgr inż. Ariel Antonowicz	136786 136805	***Ocena:***	

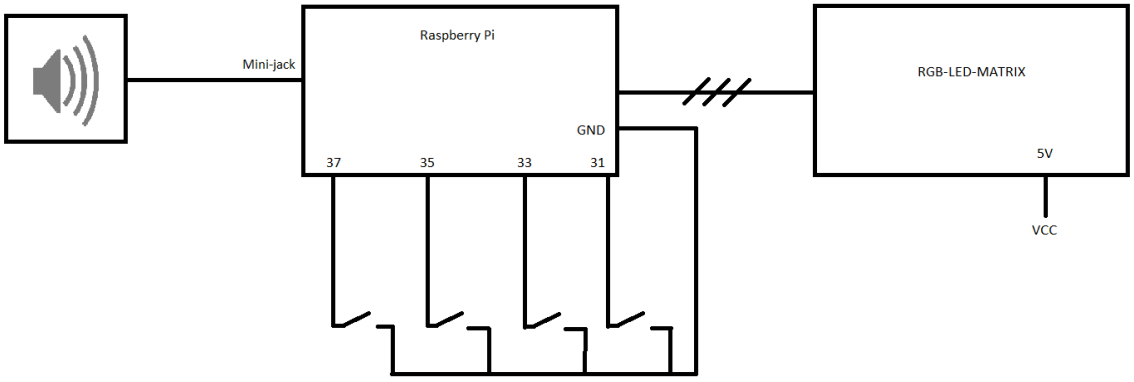
Cel projektu:
Wykonanie prostego klona gry Guitar Hero przy użyciu Raspberry Pi i matrycy LED.

Schemat:

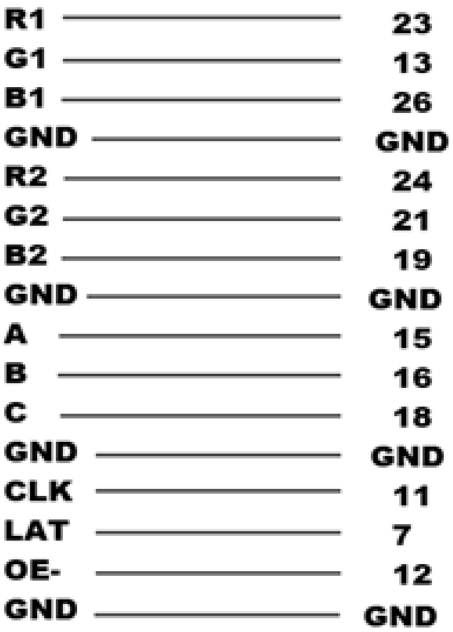


Wykorzystana platforma sprzętowa, czujniki pomiarowe, elementy wykonawcze:
- Raspberry Pi,
- matryca RGB LED,
- głośnik

1. Cel i zakres projektu
   Celem projektu jest wykonanie klona gry Guitar Hero. Gitarę zastępują przyciski, a sekwencje akordów pokazuje wyświetlacz LED. Gra posiada system naliczania punktów oraz możliwość zapisania najlepszego wyniku dla danej piosenki. Program został napisany w języku Python z wykorzystaniem API do sterowania panelem LED dostępnego pod adresem: https://github.com/hzeller/rpi-rgb-led-matrix.
2. Schematy
   a. Schemat połączeniowy



   b. Schemat połączenia matrycy LED z Raspberry Pi



c. Schemat bazy danych

Tabela songs:

- id piosenki,
- nazwa piosenki,
- ścieżka do pliku tekstowego z długościami akordów i odpowiadającymi kolorami,
- ścieżka do pliku dźwiękowego z piosenką,
- ścieżka do obrazka wyświetlanego na panelu,
- najlepszy wynik.

d. Kod inicjalizujący bazę danych

```
with open("sw/DB.db") as inp:
    conn = sqlite3.connect("sw/DB.db")

    if conn is not None:
        conn.execute("""
        CREATE TABLE IF NOT EXISTS songs
        (
        id integer PRIMARY KEY,
        name text,
        textPath text,
        songPath text,
        imagePath text,
        highScore text
        );
        """)
        c = conn.cursor()
```

e. Kod wstawiający rekordy do bazy danych

```
c = conn.cursor()
c.execute( """
 insert into songs(id,name,textPath,songPath,imagePath,highScore)
 Values (0, 'Where is my mind?', 'sw/WIMM/WIMM.txt','sw/WIMM/song.ogg','sw/WIMM/WIMM.png', 0);
 """);
c.execute("""
 insert into songs(id,name,textPath,songPath,imagePath,highScore)
Values (1, 'Test', 'sw/test/test.txt','sw/test/song.ogg','sw/test/test.png', 0);
 """);
conn.commit()
```

3. Projekt a realizacja

Wykorzystanie zaawansowanego sprzętu spowodowało spore problemy z synchronizacją dźwięku z wyświetlaniem. Na początku próbowano korzystać z oryginalnych plików .tab z Guitar Hero, jednak później zdecydowano się na skorzystanie ze zwykłych tabów gitarowych, nieco zmodyfikowanych aby było widać na panelu przerwy między dźwiękami. Zmieniono także ścieżki dźwiękowe na bardziej uproszczone. Dopiero po tych wszystkich zabiegach udało się uzyskać akceptowalną synchronizację dźwięku z obrazem.

Pierwszy zamysł zakładał także użycie pięciu przycisków zamiast czterech, aby móc zasymulować większy poziom trudności. Zrezygnowano z tego pomysłu ze względu na poprawę przejrzystości wyświetlania planszy.

Projekt można rozwinąć poprzez dodanie większej ilości piosenek oraz dodanie systemu użytkowników.

4. Najważniejsze fragmenty kodu
    a. Konwerter przygotowujący pliki do gry
        i. przykładowy mapping tabów gitarowych na kolory

```
self.colorMappingsWIMM = {
    (0, 9, 0, 0, 0, 0): (0),
    (0, 0, 8, 0, 0, 0): (1),
    (0, 0, 9, 0, 0, 0): (2),
    (0, 0, 7, 0, 0, 0): (1),
    (0, 7, 0, 0, 0, 0): (3),
    (0, 10, 9, 0, 0, 0): (0),
    (0, 12, 0, 0, 0, 0): (1),
    (10, 0, 9, 0, 0, 0): (2),
    (12, 0, 0, 0, 0, 0): (3),
    (0, 7, 9, 0, 0, 0): (2,3),
    (0, 0, 1, 2, 2, 0): (0,1),
    (0, 5, 6, 6, 4, 0): (1,2),
    (0, 6, 7, 7, 5, 0): (1,3),
    (0, 0, 0, 2, 2, 0): (0,1),
    (0, 0, 0, 0, 2, 0): (0),
    (0, 0, 0, 9, 0, 0): (0),
    (0, 0, 0, 6, 6, 4): (0,2),
    (0, 0, 5, 6, 6, 4): (0,2),
    (0, 0, 0, 0, 7, 5): (1,2),
    (0, 0, 0, 7, 7, 5): (1,3),
    (0, 0, 6, 7, 7, 5): (2,3),
    (0, 0, 5, 7, 7, 5): (2,3),
    (0, 5, 5, 7, 7, 5): (1,2,3),
    (5, 5, 5, 7, 7, 5): (0,1,2),
    (9, 9, 9, 11, 11, 9): (1,2,3),
    (7,7,8,9,9,7): (0,1),
    (0, 0, 0, 0, 0, 0): -1

}
```

        ii. przykładowy mapping długości nut na ilość wyświetlanych pikseli

```
self.tempoMappingT = {
    "W": 64,
    "H": 32,
    "Q": 16,
    "E": 8,
    "S": 4,
    "T": 2,
}
```

iii. funkcja zapisująca informacje z pliku .tab do macierzy

```
def loadText(self, path,):
    with open(path) as inp:
        j = 0
        triolaCounter = 0
        triola = []
        self.matrix = []
        for line in inp:
            self.matrix.append([])
            if line == 'Duration Legend\n':
                break
            if line[0] == ' ':
                for i in line:
                    if i == ' ':
                        self.matrix[j].append(-1)
                    elif i == '3':
                        triola = self.find(line, '3')
                        triola = list(set(triola))
                        self.matrix[j].append(-1)
                    elif i != '\n':
                        if i == '.':
                            self.matrix[j][-1] += 1/2*self.matrix[j][-1]
                            self.matrix[j].append(-1)
                        elif i in self.tempoMappingX:
                            self.matrix[j].append(self.tempoMapping[i])
                        else:
                            self.matrix[j].append(-1)
                for index in triola:
                    if self.matrix[j][index] > 0:
                        self.matrix[j][index] = self.matrix[j][index]*2/3
                        triolaCounter += self.matrix[j][index] % 1

                       # if triolaCounter%1<0.0003 or triolaCounter%1>0.9997:
                        #    self.matrix[j][index]+=round(triolaCounter)
                        #   triolaCounter=0
                        if triolaCounter > 1:
                            triolaCounter -= 1
                            self.matrix[j][index] += 1
                        self.matrix[j][index] = int(self.matrix[j][index])

            elif line[0] == '\n':
                triola = []
                continue
            else:
                for i in line:
                    if i.isdigit():
                        if self.matrix[j][-1] != -1:
                            tmp = self.matrix[j][-1]
                            self.matrix[j].append(int(i)+tmp*10)
                            self.matrix[j][-2] = 0
                        else:
                            self.matrix[j].append(int(i))
                    elif i == 'L':
                        self.matrix[j].append(-2)

                    else:
                        self.matrix[j].append(0)
                j += 1
```

iv. funkcja transponująca macierz

```
def transpose(self):
    self.transposedMatrix = []
    for i in self.matrix:
        while len(i) < len(max(self.matrix, key=len)):
            if len(i) > 0:
                if i[-1] == 0:
                    i.append(0)
                elif i[-1] == -1:
                    i.append(-1)
            else:
                i.append(-1)

    self.matrix = [i for i in self.matrix if not self.allInList(i, -1)]

    for i in range(0, len(self.matrix), 7):
        tmp = np.array(self.matrix[i:i+7])
        tmp = np.transpose(tmp)
        tmp = tmp.tolist()
        for j in tmp:
            self.transposedMatrix.append(j)

    self.transposedMatrix = [
        i for i in self.transposedMatrix if not i[0] == -1]
    for i in range(len(self.transposedMatrix)):
        for j in range(len(self.transposedMatrix[i])):
            if self.transposedMatrix[i][j] == -2:
                self.transposedMatrix[i][j] = self.transposedMatrix[i-1][j]
    for i in self.transposedMatrix:
        print(i)
```

v. funkcja rysująca obraz na podstawie macierzy

```
def draw(self, title, song):
    width = 0
    for i in self.transposedMatrix:
        width += i[0]
    image = Image.new('RGB', (int(width), 16), color='black')
    img = image.load()
    current = 0
    for i in self.transposedMatrix:
        length = int(i[0])
        color = song[tuple(i[1:])]
        if color != -1:
            if type(color) == tuple:
                for j in range(length-2):
                    for k in color:
                        img[j+current, 1+k*4] = self.RGBMapping[k]
                        img[j + current, 2 + k*4] = self.RGBMapping[k]
            else:
                for j in range(length-2):
                    img[j+current, 1+color*4] = self.RGBMapping[color]
                    img[j + current, 2 + color*4] = self.RGBMapping[color]
        current += length

    image.save(title)
```

vi. funkcja generująca plik tekstowy z długościami dźwięku, i odpowiadającymi kolorami

```
def generateTextFile(self, song, path):
    tempoIterator = 0
    currentTempo = 80
    file = open(path, "w")
    for i in self.transposedMatrix:
        file.write(str(int(i[0])))
        file.write(" ")
        accord = song[tuple(i[1:])]
        if type(accord) == tuple:
            for j in accord:

                file.write(str(j))
                file.write(" ")
        else:
            if (accord == -1):
                file.write("7 ")
            else:
                file.write(str(accord))
                file.write(" ")
        file.write(str(currentTempo))
        tempoIterator += 1
        file.write("\n")
```

b. Właściwy program

i. import bibliotek i inicjacja zmiennych globalnych

```
import time
from samplebase import SampleBase
from rgbmatrix import graphics
from PIL import Image
from rgbmatrix import RGBMatrix, RGBMatrixOptions
import RPi.GPIO as GPIO
import pygame
from timeit import default_timer as timer
import math
import sqlite3

# main vars
green = 26  # left
red = 19  # ok
yellow = 13
blue = 5  # right
orange = 16
songLines = []
timeStamp = 0
```

ii. konfiguracja przycisków

```
def setUpButtons(self):
    # Configures pin numbering to Broadcom reference
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)  # Disable Warnings
    GPIO.setup(green, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    GPIO.setup(red, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    GPIO.setup(yellow, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    GPIO.setup(blue, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    GPIO.setup(orange, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

iii. funkcja podświetlająca odpowiedni tor przy wciśnięciu przycisku

```
def buttonPressed(self, pin, color, double_buffer, i):
    if GPIO.input(pin) == 0:
        for col in range(3, 7):
            double_buffer.SetPixel(col, 4*i, color[0], color[1], color[2])
            double_buffer.SetPixel(
                col, 4*i+3, color[0], color[1], color[2])
    return GPIO.input(pin) == 0
```

iv. pętla wyboru piosenki

```
while(True):
    double_buffer.Clear()
    pos -= 1
    if pos <= -4 * len(songList[songIndex]):
        pos = 31
    my_text = songList[songIndex]
    lenn = graphics.DrawText(
    double_buffer, font, pos, 10, textColor, my_text)
    double_buffer = self.matrix.SwapOnVSync(double_buffer)
    time.sleep(0.2)
    if GPIO.input(green) == 0:
        songIndex -= 1
        if songIndex <= -1:
            songIndex = count-1
    if GPIO.input(blue) == 0:
        songIndex += 1
        if songIndex >= count:
            songIndex = 0
    if GPIO.input(red) == 0:
        break
```

v. funkcja odczytująca dane z pliku tekstowego

```
def loadText(self, path,):
    songLines = []
    with open(path) as inp:
        for line in inp:
            songLines.append(line[:-1].split(" "))
    return songLines
```

vi. funkcja rozpoczynająca odtwarzanie dźwięku

```
def playSound(self, songPath):
    pygame.mixer.init()
    pygame.mixer.music.load(songPath)
    sumOfNotes = sum([float(note[0]) for note in songLines])
    pygame.mixer.music.play()
```

vii. inicjowanie planszy, maksymalnej liczby punktów oraz odtwarzania dźwięku

```
for row in range(0, 16):
    for col in range(0, 32):
        double_buffer.SetPixel(col, row, 0, 0, 0)

for row in range(0, 16):
    double_buffer.SetPixel(7, row, 255, 255, 255)
double_buffer = self.matrix.SwapOnVSync(double_buffer)

songLines = self.loadText(textList[songIndex])

self.image = Image.open(imageList[songIndex]).convert('RGB')
self.image.resize(
    (self.matrix.width, self.matrix.height), Image.ANTIALIAS)
img_width, img_height = self.image.size
timeStamp = (60/(float(songLines[0][-1])*16))

xpos = -10
points = 0
hit = 0
combo = 0
multiplier = 1
start = 0
cnt = 0
cmb = 1
maxPoints = 0
maxHit = 0

for line in songLines:
    if line[1] != 7:
        maxHit += int(line[0]) * (len(line)-2)
        maxPoints += int(line[0]) * (len(line)-2)*cmb
        cnt += 1
        if cnt >= 20 and cmb < 4:
            cnt = 0
            cmb += 1

print(maxPoints)
print(maxHit)
playSound(self, fileList[songIndex])
```

viii. zliczanie punktów i przewijanie obrazka

```
for line in songLines:
    for i in range(int(line[0])):
        if i == 0:
            start = timer()
        xpos += 1
        double_buffer.SetImage(self.image, -xpos)
        double_buffer.SetImage(self.image, -xpos + img_width)

        greenPressed = self.buttonPressed(pin=green, color=(
            0, 255, 0), double_buffer=double_buffer, i=0)
        redPressed = self.buttonPressed(pin=red, color=(
            255, 0, 0), double_buffer=double_buffer, i=1)
        yellowPressed = self.buttonPressed(pin=yellow, color=(
            255, 255, 0), double_buffer=double_buffer, i=2)
        bluePressed = self.buttonPressed(pin=blue, color=(
            0, 0, 255), double_buffer=double_buffer, i=3)

        for row in range(0, 16):
            double_buffer.SetPixel(7, row, 255, 255, 255)

        for row in range(0, 16):
            double_buffer.SetPixel(28, row, 255, 255, 255)
            for col in range(29, 32):
                double_buffer.SetPixel(col, row, 0, 0, 0)
        for note in line[1:-1]:
            if note == "0":
                if greenPressed == True:
                    hit += 1
                    points += multiplier
                    combo += 1
                else:
                    combo = 0
                    multiplier = 1
            if note == "1":
                if redPressed == True:
                    hit += 1
                    points += multiplier
                    combo += 1
                else:
                    combo = 0
                    multiplier = 1
```

cd.

```
    if note == "2":
        if yellowPressed == True:
            hit += 1
            points += multiplier
            combo += 1
        else:
            combo = 0
            multiplier = 1
    if note == "3":
        if bluePressed == True:
            hit += 1
            points += multiplier
            combo += 1
        else:
            combo = 0
            multiplier = 1

if points >= starProgs[len(star)] * maxPoints:
    star += "*"

if combo >= 20 and multiplier < 4:
    multiplier += 1
    combo = 0

for m in range(0, multiplier):
    for n in range(0, 3):
        double_buffer.SetPixel(
            31-n, 15-m, 255, 255, 128 - 128/(4-m))

for m in range(0, len(star)):
        double_buffer.SetPixel(30, 2*m+1, 255, 255, 0)

double_buffer = self.matrix.SwapOnVSync(double_buffer)
end = timer()
time.sleep(timeStamp-(end-start))
start = timer()
```

ix. aktualizowanie najlepszego wyniku w bazie oraz wyświetlanie wyniku

```
pygame.mixer.music.stop()
double_buffer.Clear()
double_buffer = self.matrix.SwapOnVSync(double_buffer)

percentage = int(float(hit)/float(maxHit)*100)
pointsColor = textColor
if highScores[songIndex] < points:
    pointsColor = starColor
    highScores[songIndex] = points
    c = conn.cursor()
    c.execute("Update songs set highScore=" +
              str(points) + " where id = "+str(songIndex)+";")

    conn.commit()
else:
    pointsColor = textColor
while True:
    if GPIO.input(blue) == 0:
        break

    double_buffer.Clear()
    lenn = graphics.DrawText(
    double_buffer, font, 2*(8-len(str(points))), 6, pointsColor, str(points))
    if percentage < 100:
        lenn = graphics.DrawText(
            double_buffer, font, 11, 12, pointsColor, str(percentage)+"%")
    else:
        lenn = graphics.DrawText(
            double_buffer, font, 9, 12, pointsColor, str(percentage)+"%")

    lenn = graphics.DrawText(
        double_buffer, font,  2*(8-len(str(star))), 18, starColor, star)
    double_buffer = self.matrix.SwapOnVSync(double_buffer)
    time.sleep(0.2)
maxPoints = 0
maxHit = 0
songLines = []
```

5. Podsumowanie i wnioski.

   W stosunku do pierwotnego pomysłu gra zyskała na oprawie wizualnej, w zamian za to nieco straciła na jakości dźwięku. Grywalność na wysokim poziomie, w zwiększeniu atrakcyjności gry pomogłoby zastosowanie lepszej jakości przycisków. Początkowo spotkania zespołu zaplanowano na zajęcia laboratoryjne, jednak miejsce spotkań szybko zmieniono na mieszkanie prywatne, gdzie ich częstotliwość i długość mogła być kilkukrotnie zwiększona. Dzięki temu projekt został skończony przed wyznaczonym terminem. Główne trudności sprawiła synchronizacja dźwięku z obrazem, brak doświadczenia w obsłudze panelu LED oraz wymyślenie sposobu wyświetlania akordów.