CPE316 – Embedded Systems Final Project Report

Simple Arcade Game : Snake

Semester II (2021-2022)

Course supervisors:	Team STRAWBERRIES members:
Dr. Iman Elawady ENG. Michael B.Khani	Enes YURDATAPAN, 1810205079@ogrenci.karabuk.edu.tr* Ali Eren ERGÜN, 1810205041@ogrenci.karabuk.edu.tr Ali Ramazan TAŞDELEN, 1910205003@ogrenci.karabuk.edu.tr Batıkan CIMBIT, 1810205043@ogrenci.karabuk.edu.tr Kamoliddin FATKHIDDINOV, 1910205506@ogrenci.karabuk.edu.tr Ahmat SOUMAINE, 1810205023@ogrenci.karabuk.edu.tr Sevgican KILIÇ, 1810205078@ogrenci.karabuk.edu.tr Oğuzhan PORTAKAL, 1810205048@ogrenci.karabuk.edu.tr

* Team leader and correspondent member

Contents

1. Introduction

Due to the increase in the demand for the entertainment sector and mobile devices, it is possible to see that the sales in this sector have been at the highest point recently.

One of the most important points of winning in this sector is to be able to promote the product very well and to produce it cheaply.

We aimed to integrate traditional game which is an indispensable childhood enjoyment that people always remember, in a modern way.

We wanted to design it not only as a toy for children, but also for adults as a way of entertainment while waiting in line in anywhere or on a long journey.

We started our project at 10th of April and finished successfully at 17th of May.

2. Related Works

As a result of our research, we can see many such toys in the markets. We noticed a lot of attention despite their simple design. We analyzed the comments on these sales, identified the deficiencies and errors, and concluded that they would sell more when we fixed them.

We realized that the ones on the market usually appeal to children, so we added a challenging mode to our game and made it a place that adults can enjoy as well. There was a deficit in this area in the market, we thought of closing the gap by turning it into profit.

3. Project Design

3.1. Hardware

Analog Joystick

We first researched different button hardware to control the game and found the analog joystick to be the most useful.

Provided connections by connecting the appropriate pins on the analog joystick to the appropriate analog pins on the ESP32.

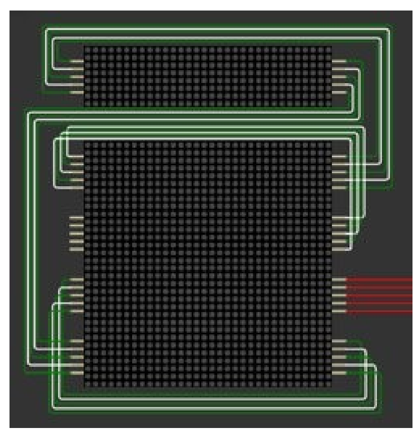
Analog Joystick

Analog Joystick in the WOKWİ

8x8 LED Dot Matrix

In order to view the game, a hardware was needed. We decided that the one suitable for our project was LED Dot Matrix.

Our game and menu screen was created by connecting 16 8x8 matrices. By connecting 4 of them separately to the upper part, it was possible to view the scores. Connecting the cables was difficult.



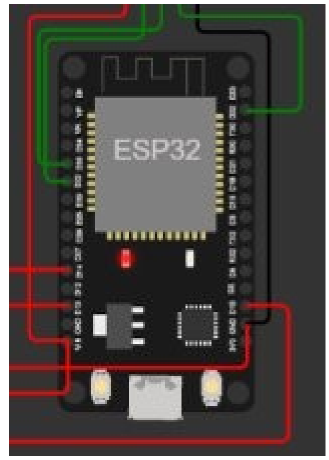
8x8 LED Dot Matrix

20 of 8x8 matrices connected to each other as output hardware

ESP32

We used ESP32 together with the wokwi simulator as it is well applicable with Micropython and simulation.

ESP32



ESP32 in the WOKWI

3.2. Software

MAX7219 Driver

The specially assembled LED monitor used needed a driver to work with ESP32. Open source MAX7219 driver using SPI connection interface was found from internet. But this driver only works for 8x8 single matrix. It was showing the opposite on the special screen we used. We changed some codes and adapted them to our own matrix.

Game Engine

Actually, our entire game consists of game engine. Inspired by the snake game engines we found as a result of our research, we wrote our own game engine from scratch.

Game Engine includes important submodules such as random bait and wall creation, display and mod functions. We tried to improve the code as much as we could.

Game Menu

We have created a menu to select difficulty modes and switch to the game. It can also be developed in the future to select different games.

4. Conclusion and Results

Prototype was used for testing trade-offs and consistency of product. As a result of the our project simulated in Wokwi environment, we came to the conclusion of a physical product is plausible.

Most of project time used for software part was advantage of using a virtual environment and developing the game.

Due of simulation platform and python environment we can't solve problems related to speed of the snake.(real baud rate differs from simulated one)

5. Lesson Learnt

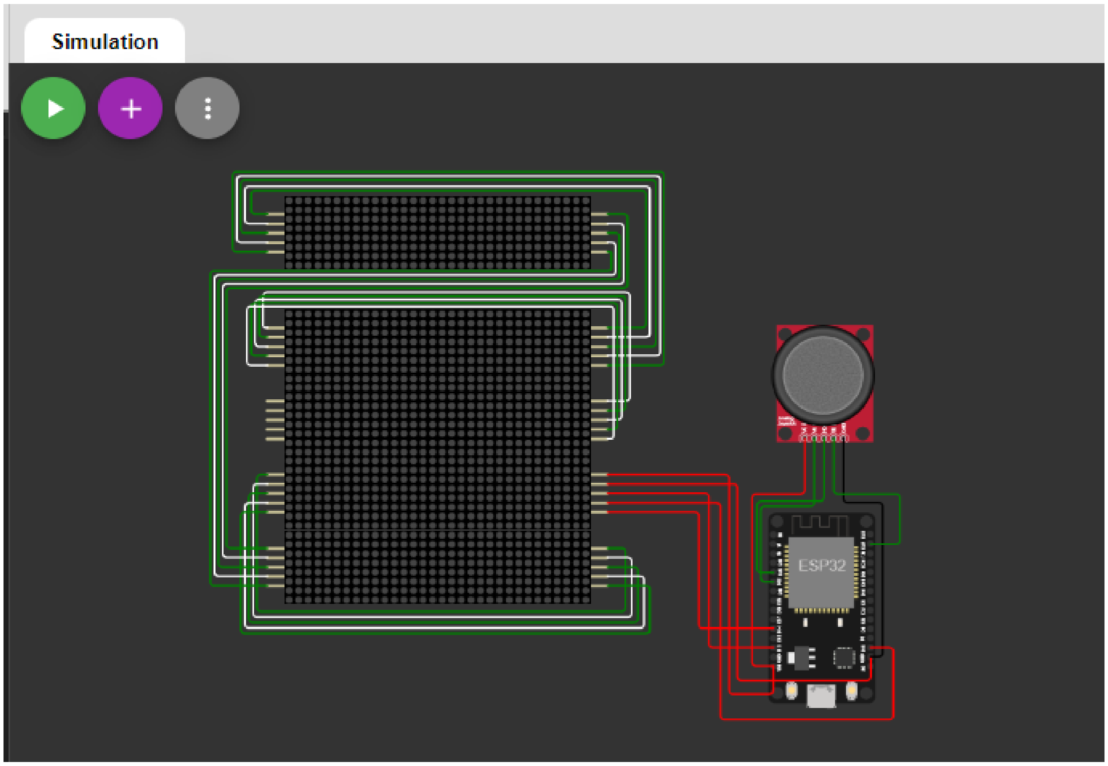
As a result of our project, we got acquainted with the wokwi simulation and micropython environment. We learned how to write our own library and develop it by optimizing it and how to schedule projects better in coordination with teamwork.

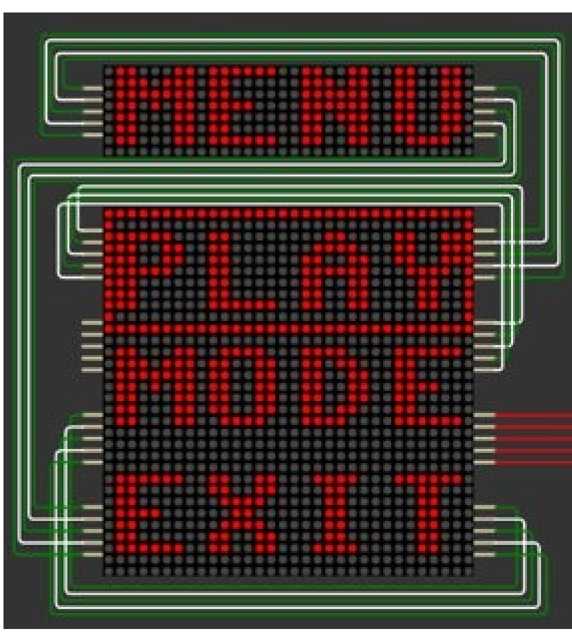
6. References

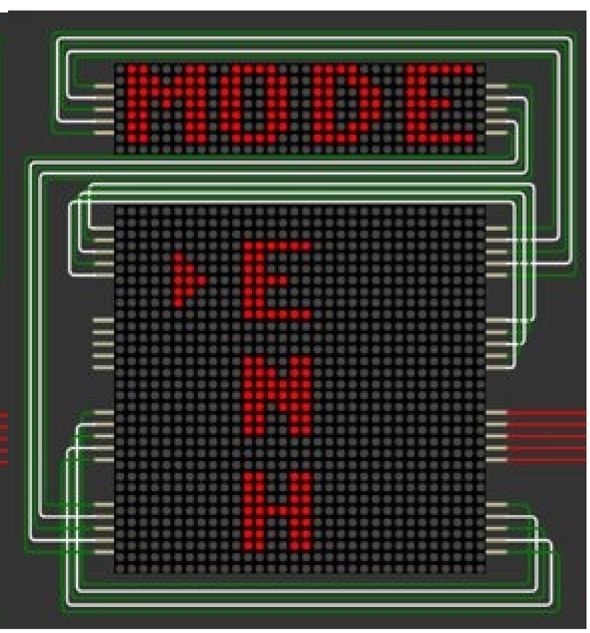
- Arduino Simulator: Uno, Mega, ESP32, FastLED, LCD1602, Servo, Raspberry Pi Pico, Sensors. Designed for makers, by makers. https://wokwi.com/projects/328974406829212243
- https://github.com/csdexter/MAX7219
- https://github.com/coding-world/max7219
- https://github.com/adafruit/Adafruit_CircuitPython_MAX7219
- https://www.w3schools.com/python/

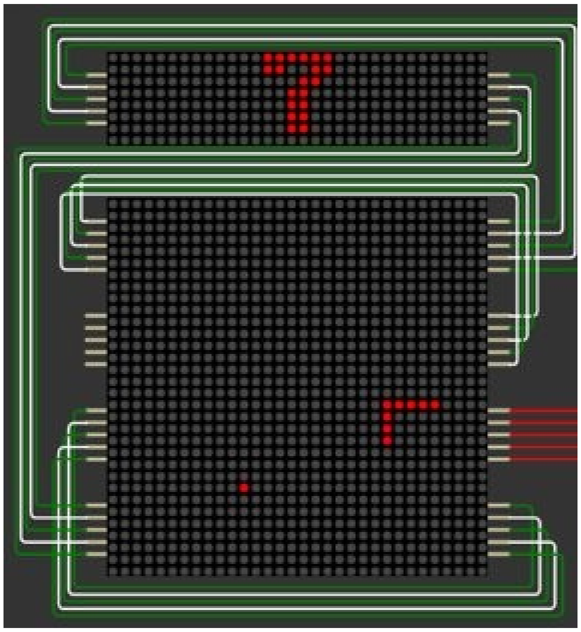
7. Attachments

7.1. Project images

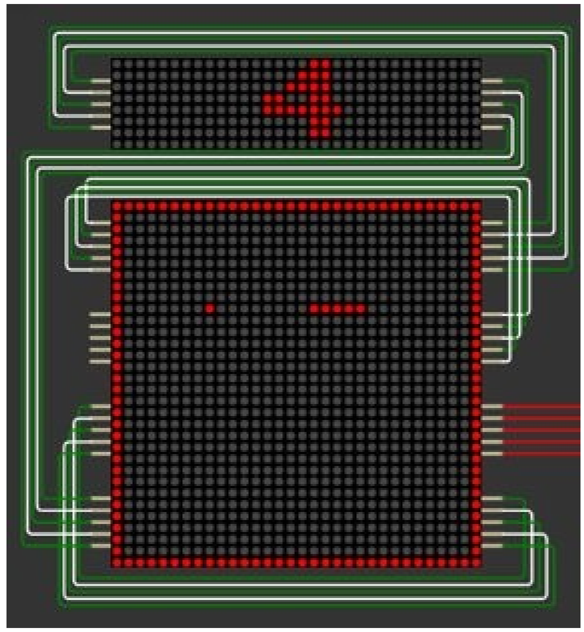




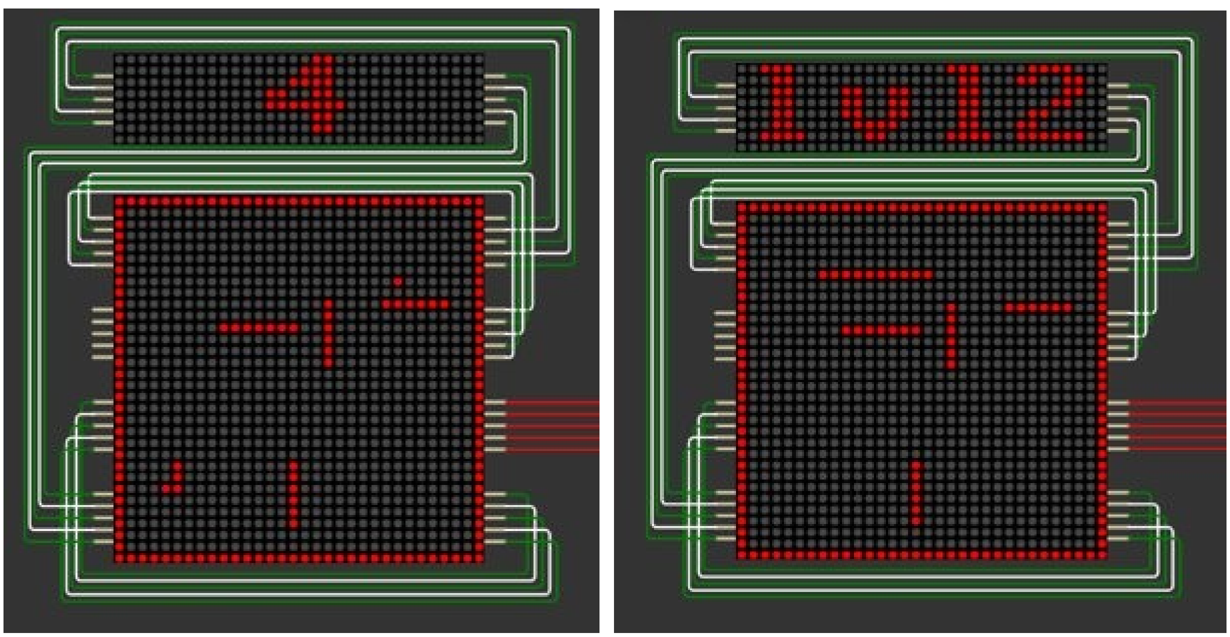




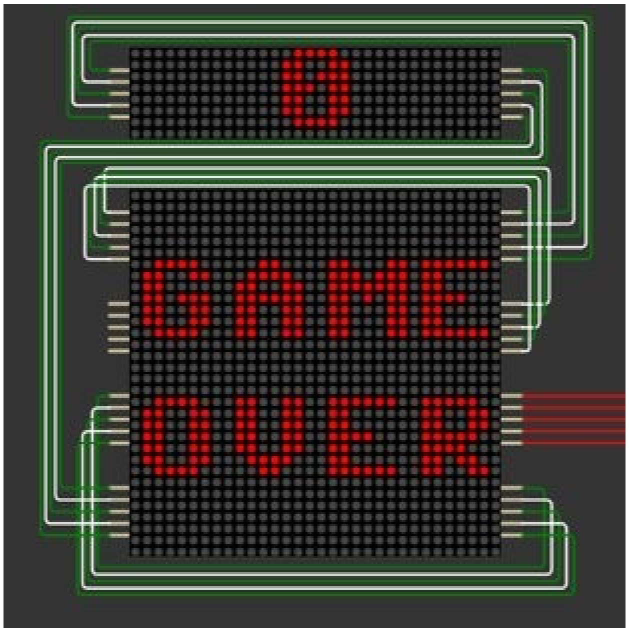
Easy Mode

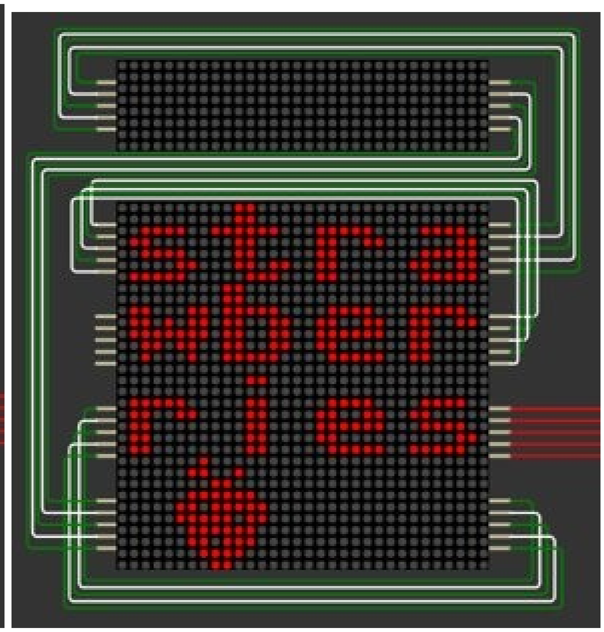


Normal Mode



Hard Mode





7.2. Code Modules

7.2.1. main.py

```
from Max7219 import Max7219
from machine import Pin,SPI,ADC
from engine import Game
from time import sleep
from config import *
import strawberries



class Menu:
  def __init__(self,screen):
    self.screen = screen
    self.menu_index = 0
    self.mode_index = 0
    self.state = ""
    self.arrow =
[[0,0],[0,1],[0,2],[0,3],[0,4],[1,1],[1,2],[1,3],[2,2]]


  def get_joystick_state(self):
    xRef = xAxis.read_u16()
    yRef = yAxis.read_u16()


    if yRef == 0:
      self.state = "down"
```

```
    if yRef == 65535:
      self.state = "up"
    if xRef == 0:
      self.state = "right"
    if xRef == 65535:
      self.state = "left"


  def display_menu_default(self):
    self.screen.fill(0)
    self.screen.text("MENU",0,32,1)
    self.screen.text("PLAY",0,2,1)
    self.screen.text("MODE",0,12,1)
    self.screen.text("EXIT",0,23,1)


  def display_mode_default(self):
    self.screen.fill(0)
    self.screen.text("E",10,3,1)
    self.screen.text("N",10,13,1)
    self.screen.text("H",10,23,1)
    self.screen.text("MODE",0,32,1)




  def draw_pointer(self,index):
    for pixel in self.arrow:
      self.screen.pixel(pixel[0]+5,pixel[1]+index*10+4,1)



  def display_menu(self):
    while 1:
      self.display_menu_default()
      self.get_joystick_state()


      if self.state == "down" and self.menu_index<2:
        self.menu_index +=1
        self.state = ""
      if self.state == "up" and self.menu_index>0:
        self.menu_index -=1
        self.state = ""



      if self.menu_index==0:
        self.display_menu_default()
        self.screen.rect(0,0,32,11,1)
```

```
    if self.menu_index==1:
      self.display_menu_default()
      self.screen.rect(0,10,32,11,1)


    if self.menu_index==2:
      self.display_menu_default()
      self.screen.rect(0,21,32,11,1)


    self.screen.show()


    if not SW.value():

      break


    sleep(0.01)
  return self.menu_index,self.mode_index




def display_mode(self):
  self.display_mode_default()


  self.draw_pointer(self.mode_index)
  self.screen.show()



  while True:
    self.display_mode_default()
    self.get_joystick_state()
    if self.state == "down" and self.mode_index<2:
      self.mode_index +=1
      self.state = ""
    if self.state == "up" and self.mode_index>0:
      self.mode_index -=1
      self.state = ""
    self.draw_pointer(self.mode_index)
    self.screen.show()
    if not SW.value():
      break
```

```
menu = Menu(screen)
game = Game(screen)


strawberries.display(screen)
sleep(0.5)
screen.fill(0)


while True:
  menu_index,mode_index = menu.display_menu()
  if menu_index == 0:
    if mode_index == 0:
      game.easy_mode()
    elif mode_index == 1:
      game.normal_mode()
    elif mode_index == 2:
      game.hard_mode()
  elif menu_index == 1:
    menu.display_mode()
  elif menu_index == 2:
    break


strawberries.display(screen)
sleep(2)
screen.fill(0)
screen.show()
```

7.2.2. MAX7219.py

```
from machine import Pin
from micropython import const
import framebuf
import time
_DIGIT_0 = const(0x1)

```

```python


_DECODE_MODE = const(0x9)
_NO_DECODE = const(0x0)


_INTENSITY = const(0xA)
_INTENSITY_MIN = const(0x0)


_SCAN_LIMIT = const(0xB)
_DISPLAY_ALL_DIGITS = const(0x7)


_SHUTDOWN = const(0xC)
_SHUTDOWN_MODE = const(0x0)
_NORMAL_OPERATION = const(0x1)


_DISPLAY_TEST = const(0xF)
_DISPLAY_TEST_NORMAL_OPERATION = const(0x0)


_MATRIX_SIZE = const(8)



class Max7219(framebuf.FrameBuffer):

    def __init__(self, width, height, spi, cs, rotate_180=False):
        # Pins setup
        self.spi = spi
        self.cs = cs
        self.cs.init(Pin.OUT, True)


        # Dimensions
        self.width = width
        self.height = height
        # Guess matrices disposition
        self.cols = width // _MATRIX_SIZE
        self.rows = height // _MATRIX_SIZE
        self.nb_matrices = self.cols * self.rows
        self.rotate_180 = rotate_180


        # 1 bit per pixel (on / off) -> 8 bytes per matrix
        self.buffer = bytearray(width * height // 8)
        format = framebuf.MONO_HMSB if not self.rotate_180 else framebuf.MONO_HLSB
```

```
        super().__init__(self.buffer, width, height, format)


        # Init display
        self.init_display()


    def _write_command(self, command, data):
        """Write command on SPI"""
        cmd = bytearray([command, data])
        self.cs(0)
        for matrix in range(self.nb_matrices):
            self.spi.write(cmd)
        self.cs(1)


    def init_display(self):
        """Init hardware"""
        for command, data in (
            (_SHUTDOWN, _SHUTDOWN_MODE),  # Prevent flash during
init
            (_DECODE_MODE, _NO_DECODE),
            (_DISPLAY_TEST, _DISPLAY_TEST_NORMAL_OPERATION),
            (_INTENSITY, _INTENSITY_MIN),
            (_SCAN_LIMIT, _DISPLAY_ALL_DIGITS),
            (_SHUTDOWN, _NORMAL_OPERATION),  # Let's go
        ):
            self._write_command(command, data)


        self.fill(0)
        self.show()


    def brightness(self, value):
        """Set display brightness (0 to 15)"""
        if not 0 <= value < 16:
            raise ValueError("Brightness must be between 0 and
15")
        self._write_command(_INTENSITY, value)


    def show(self):
        """Update display"""
        # Write line per line on the matrices
        for line in range(8):
            self.cs(0)


            for matrix in range(self.nb_matrices):
```

	`` # Guess where the matrix is placed row, col = divmod(matrix, self.cols) # Compute where the data starts if not self.rotate_180: offset = 8 * self.cols - row * self.cols * 8 index = col + line * self.cols + offset else: offset = 8 * self.cols - row * self.cols * 8 - 1 index = self.cols * (8 - line) - col + offset self.spi.write(bytearray([_DIGIT_0 + line, self.buffer[index]])) self.cs(1) ```

*engine.py and config.py will be in the zip file.