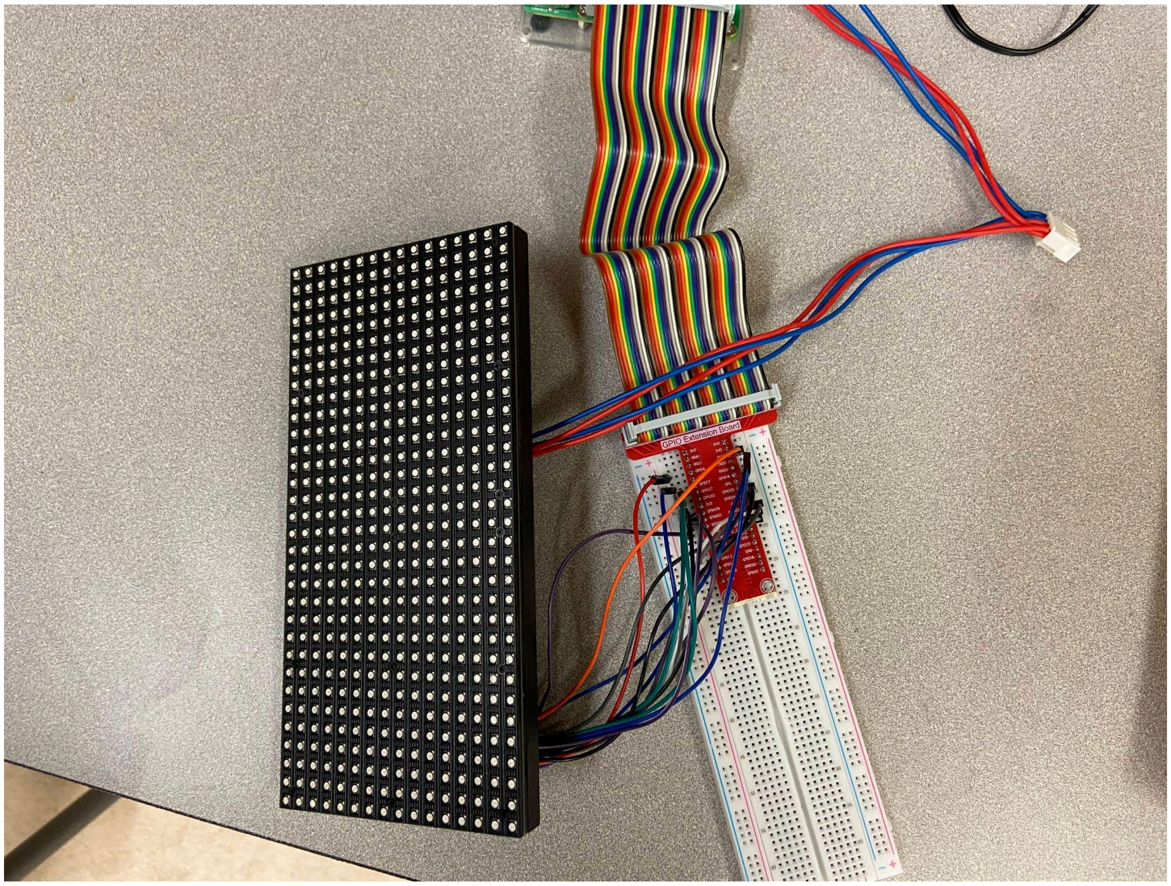
TEJ4M - Computer Engineering Summative Report
Raspberry Pi RGB LED Matrix Game

Prepared by:	Jinyuan Zhao
Prepared for:	Mr. Palmer
Date:	Jan 17, 2024



Oakville Trafalgar High School
1460 Devon Road
Oakville, ON L6J 3L6
Phone: (905) 845-2875
Fax: (905) 845-3532

Executive Summary

This project aims to create a simple gaming system involving the RGB LED Matrix Display. By using the `rpi-rgb-led-matrix` library by `hzeller` and its Python bindings, we have created 2 simple games utilizing the keyboard on the computer as input and the matrix display and its various beautiful colours as output.

This project demonstrated our skill and ability to solve complex problems and learn difficult concepts and components all by ourselves. Throughout the project, we utilized sources such as forums or documentation to build a better understanding of the technology.

Overall, this project demonstrated our ability to wire, learn and troubleshoot a complex system involving Raspberry Pi, LED matrix display and foreign libraries. We have gained valuable knowledge and experience in coding and problem-solving. This project's outcome not only showcases our technical skills but also as a foundation for next year's projects, where we will utilize this display to its full potential.

Table of Contents

1 INTRODUCTION

This summative performance task serves as a final check of the students' ability to understand and apply the knowledge they learned earlier in the year and a statement of their ability to learn without the teacher's supervision.

Our project aims to explore the ability to utilize an RGB matrix display with the Raspberry Pi. By making 2 simple games on the matrix, we explored the possibilities and built the foundations for the next semester.

This project is focused more on code development and exploring the details of how a Raspberry Pi functions, as the wiring is relatively simple and provided on the `rpi-lgb-led-matrix` library.

The following sections, Challenges will be dedicated to the details of this project. The problems I encountered while wiring and programming the display will be explained extensively in the following section.

2 CHALLENGES

Post-wiring, our major challenge was programming the display. The driver, written in C, presented a steep learning curve due to its complexity, as evidenced in the example code. We overcame this hurdle by utilizing hzeller's Python bindings for the library, which streamlined our coding process. Despite the smooth execution of the example with the command.

```
Unset
sudo examples-api-use/demo -D0 —led-rows=16 —led-cols=32
—led-no-hardware-pulse
```

We foresee potential difficulties in fully leveraging these bindings for our specific display requirements, marking our next significant challenge.

We discovered a method to install the necessary library for our display. In the library's directory, we initiated the installation process using the `make` command in the terminal. This step was essential to compile the library for our use. To ensure all dependencies were met, we updated our system and installed Python development tools and the Pillow library:

Unset
```
sudo apt-get update

sudo apt-get install python3-dev python3-pillow -y
```

We then built the Python bindings with:

Unset
```
make build-python PYTHON=$(which python3)

sudo make install-python PYTHON=$(which python3)
```

Successfully executing these commands made the `rgbmatrix` Python module accessible system-wide, a critical step in enabling us to programmatically control our display.

In the journey of our project's construction phase, we stumbled upon a critical and somewhat unexpected technical hurdle related to the `--led-no-hardware-pulse` parameter. This challenge emerged when our program consistently failed to run, and the root cause was traced back to a conflict with the Raspberry Pi's sound module. The sound module, we discovered, was utilizing a critical timing circuit, which was indispensable for the proper functioning of our display system. The issue manifested in the form of compatibility problems, evidenced by the following error message:

Unset
```
=== snd_bcm2835: found that the Pi sound module is loaded. ===
Don't use the built-in sound of the Pi together with this lib; it is known to be
incompatible and cause trouble and hangs (you can still use external USB sound
      adapters).

See Troubleshooting section in README how to disable the sound module.
You can also run with --led-no-hardware-pulse to avoid the incompatibility,
but you will have more flicker.
Exiting; fix the above first or use --led-no-hardware-pulse
```

Initially, our approach to circumvent this issue involved the use of the `--led-no-hardware-pulse` flag in our example files. This method, though effective in those instances, proved to be a dead end when applied to our specific Python file. Consequently, we were compelled to explore a more direct solution—disabling the sound module itself.

Our first step in addressing this issue was to modify the Raspberry Pi's boot configuration:

We accessed the boot configuration file using the nano editor:

```
Unset
sudo nano /boot/config,txt
```

Inside the configuration file, we pinpointed and disabled the sound module:

```
Unset
dtparam=audio=off
```

Post-editing, a system reboot was necessary to apply these changes:

```
Unset
sudo reboot now
```

However, this initial attempt did not yield the desired outcome for our unique setup. The sound module, despite being disabled in the boot configuration, continued to present a conflict. This led us to undertake a more radical approach—completely disabling the sound system at the kernel level:

To confirm the active status of the sound system, we used the list command:

```
Unset
ls
```

The `bcm2835` module's presence confirmed our suspicions.

```Unset
...
bcm2835_codec          49152 0
...
```

The next course of action involved blacklisting the sound module to prevent it from loading:

```Unset

cat <<EOF | sudo tee /etc/modprobe.d/blacklist-rgb-matrix.conf
blacklist snd_bcm2835
EOF

sudo update-initramfs -u
```

A final reboot was required to solidify these changes:

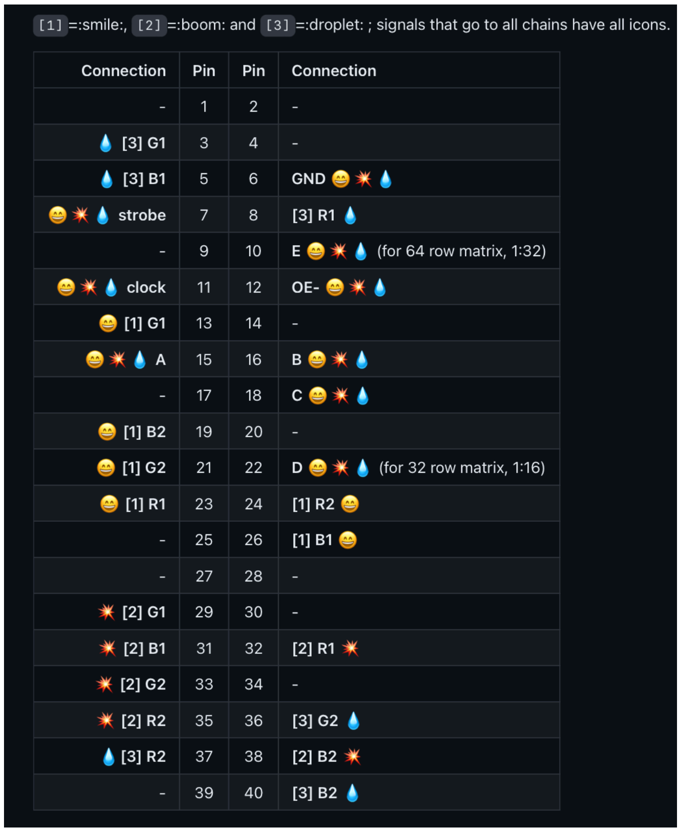
```Unset
sudo reboot now
```

Upon completion of these steps, our display system began to operate flawlessly, rendering the initial `--led-no-hardware-pulse` workaround obsolete.

Embarking on the construction phase, our plan to integrate joystick inputs for user interaction encountered unforeseen difficulties. The initial steps seemed straightforward, with the joystick setup and a rudimentary detection program ready. However, an unexpected runtime error quickly halted our progress, questioning the compatibility of our code with the Raspberry Pi:

```Unset
esk@raspberrypi:~/Desktop/rpi-lgb-led-matrix-project $ sudo /bin/python3
      /home/esk/Desktop/rpi-lgb-led-matrix-p
roject/game.py
Traceback (most recent call last):
 File "/home/esk/Desktop/rpi-lgb-led-matrix-project/game.py", line 15, in
      <module>
  ADC0834.setup()
 File "/home/esk/Desktop/rpi-lgb-led-matrix-project/ADC0834.py", line 31,
      in setup
  GPIO.setup(ADC_CS, GPIO.OUT)       # Set pins' mode is output
RuntimeError: Not running on a RPi!
```

This error, though initially bewildering, led us to a critical examination of the display's wiring setup.



`[1]`=:smile:, `[2]`=:boom: and `[3]`=:droplet: ; signals that go to all chains have all icons.

Connection	Pin	Pin	Connection
-	1	2	-
💧 [3] G1	3	4	-
💧 [3] B1	5	6	GND 😄 💥 💧
😄 💥 💧 strobe	7	8	[3] R1 💧
-	9	10	E 😄 💥 💧 (for 64 row matrix, 1:32)
😄 💥 💧 clock	11	12	OE- 😄 💥 💧
😄 [1] G1	13	14	-
😄 💥 💧 A	15	16	B 😄 💥 💧
-	17	18	C 😄 💥 💧
😄 [1] B2	19	20	-
😄 [1] G2	21	22	D 😄 💥 💧 (for 32 row matrix, 1:16)
😄 [1] R1	23	24	[1] R2 😄
-	25	26	[1] B1 😄
-	27	28	-
💥 [2] G1	29	30	-
💥 [2] B1	31	32	[2] R1 💥
💥 [2] G2	33	34	-
💥 [2] R2	35	36	[3] G2 💧
💧 [3] R2	37	38	[2] B2 💥
-	39	40	[3] B2 💧

The wiring diagram revealed a significant oversight: the display's expansive use of GPIO pins left us with no available pins for input peripherals like joysticks. This realization was a pivotal moment, highlighting the importance of thoroughly understanding hardware interdependencies in system design.

With GPIO inputs no longer viable, we pivoted to keyboard inputs. Pygame, with its user-friendly reputation, was our chosen library for this task. Following its installation, we implemented keyboard event handling mechanisms:

```
Unset
sudo python3 -m pip install pygame
```

However, the transition to Pygame was not smooth. Running our script resulted in a critical error, confounding us as we were already executing it with root privileges:

```
Unset
esk@raspberrypi:~/Desktop/rpi-lgb-led-matrix-project $ sudo python3
      game.py
pygame 1.9.6
Hello from the pygame community. https://www.pygame.org/contribute.html
No protocol specified
No protocol specified
No protocol specified
Fatal Python error: pygame_parachute: (pygame parachute) Segmentation
      Fault
Python runtime state: initialized

Current thread 0xf796c640 (most recent call first):
  File "/home/esk/Desktop/rpi-lgb-led-matrix-project/game.py", line 16
      in <module>
Aborted
```

Despite following common troubleshooting advice, the issue persisted, suggesting a deeper underlying problem. This failure, while frustrating, was instrumental in teaching us the unpredictability of software dependencies and the complexities of permissions in a Linux environment.

In our quest for a viable input method, we next explored the Python keyboard library, known for its straightforward approach to capturing keypresses. Optimistic, we implemented `keyboard.is_pressed` to detect input. Unfortunately, this attempt also encountered significant roadblocks:

```
Unset
Traceback (most recent call last):
 File "/home/esk/Desktop/rpi-lgb-led-matrix-project/game.py", line 53, in
      <module>
  if keyboard.is_pressed('w'):
 File "/usr/local/lib/python3.9/dist-packages/keyboard/__init__.py", line
      410, in is_pressed
  _listener.start_if_necessary()
 File "/usr/local/lib/python3.9/dist-packages/keyboard/_generic.py", line
      35, in start_if_necessary
  self.init()
 File "/usr/local/lib/python3.9/dist-packages/keyboard/__init__.py", line
      196, in init
  _os_keyboard.init()
 File "/usr/local/lib/python3.9/dist-packages/keyboard/_nixkeyboard.py",
      line 113, in init
  build_device()
 File "/usr/local/lib/python3.9/dist-packages/keyboard/_nixkeyboard.py",
      line 109, in build_device
  ensure_root()
 File "/usr/local/lib/python3.9/dist-packages/keyboard/_nixcommon.py",
      line 174, in ensure_root
  raise ImportError('You must be root to use this library on linux.')
ImportError: You must be root to use this library on linux.
```

Despite running the script with root privileges, as the error suggested, we faced the same impasse. A switch to non-root mode led us back to compatibility issues with the `rgbmatrix` library, creating a catch-22 scenario.

In our final endeavour to resolve the input handling dilemma, we turned to a custom solution involving Python's `termios` library. We crafted a separate Python file, `getch.py`, to capture keyboard inputs in a non-blocking manner. This approach allowed us to bypass the limitations we faced with other libraries:

```python
Python
# getch.py
import sys
import tty
import termios

def getch():
  fd = sys.stdin.fileno()
  old_settings = termios.tcgetattr(fd)
  try:
    tty.setraw(sys.stdin.fileno())
    ch = sys.stdin.read(1)
  finally:
    termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
  return ch
```

The `getch` function, a simple yet elegant solution, performed flawlessly in our preliminary tests:

```python
# keyboard_input.py
import getch

while True:
  char = getch.getch()

  if char.lower() == 'w':
    print("Up")
  elif char.lower() == 'a':
    print("Left")
  elif char.lower() == 's':
    print("Down")
  elif char.lower() == 'd':
    print("Right")
  elif char.lower() == 'q':
    print("Quitting")
    break
  else:
    print("Unknown key:", char)
```

Below is our modified code in the main game file handling input.

Python
```python
while True:
      char = getch.getch()

      if char.lower() == 'w':
          direction = "UP"
      elif char.lower == 'a':
          direction = "LEFT"
      elif char.lower() == 's':
          direction = "DOWN"
      elif char.lower() == 'd':
          direction = "RIGHT"
```

However, integrating getch into our main game code revealed a critical flaw: its blocking nature. In a real-time game like Snake, this was impractical. To circumvent this, we leveraged Python's threading module, creating a separate thread for handling keyboard input. This thread updated a shared variable, input_direction, allowing seamless integration with the game's main loop:

Python
```python
# use function to handle in separate thread
def read_keyboard_input():
    global input_direction
    while True:
        char = getch.getch()
        if char.lower() == 'w':
            input_direction = "UP"
        elif char.lower() == 'a':
            input_direction = "LEFT"
        elif char.lower() == 's':
            input_direction = "DOWN"
        elif char.lower() == 'd':
            input_direction = "RIGHT"

# Start the keyboard input thread
input_thread = threading.Thread(target=read_keyboard_input)
input_thread.daemon = True  # Daemonize thread
input_thread.start()

while True:
    direction = input_direction

    # rest of my code
```

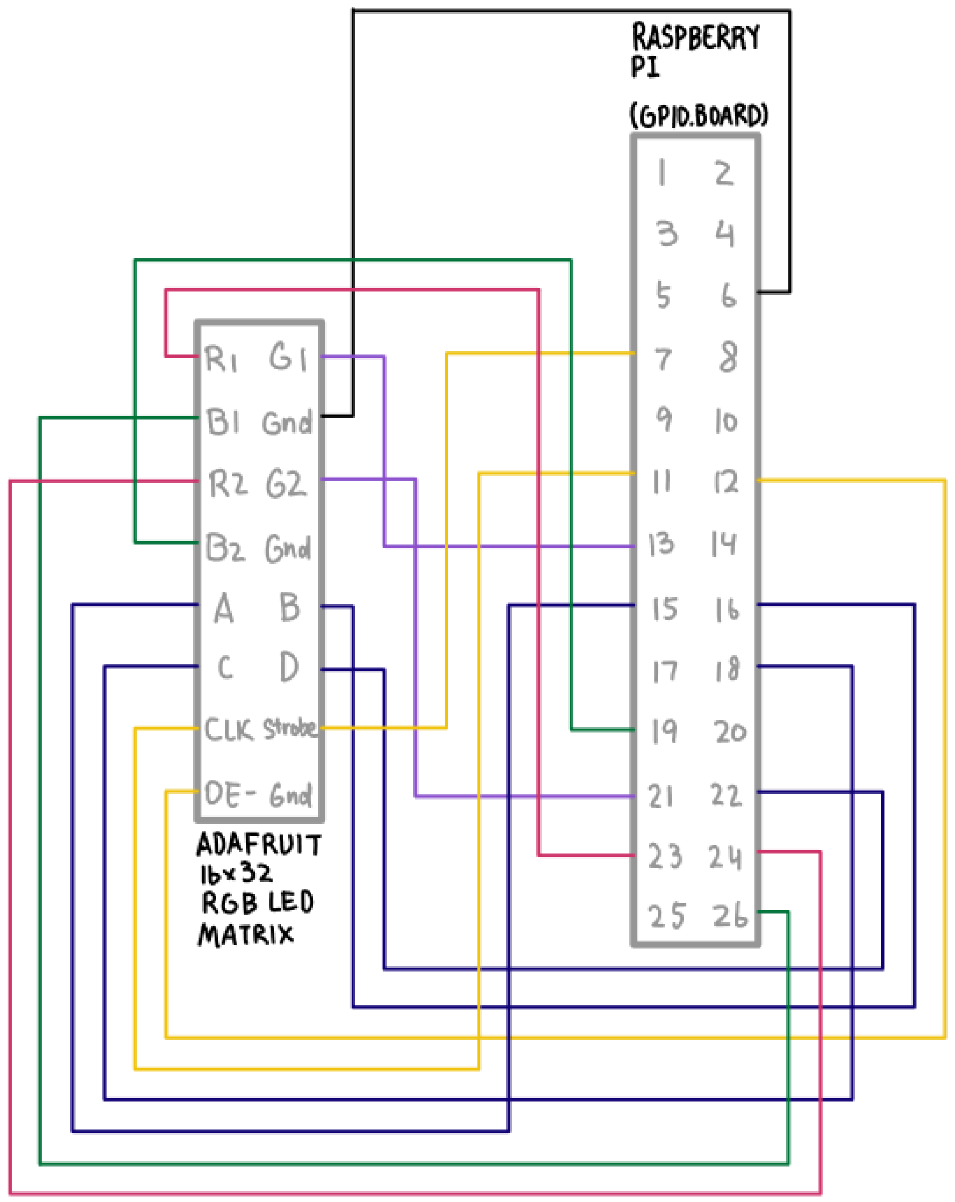
This threading solution elegantly resolved our input challenges, marking a significant milestone in our project. It not only addressed the immediate technical hurdle but also served as a testament to our team's resilience and innovative spirit. This breakthrough allowed us to proceed with a fully functional and responsive game, ready for deployment and playtesting.

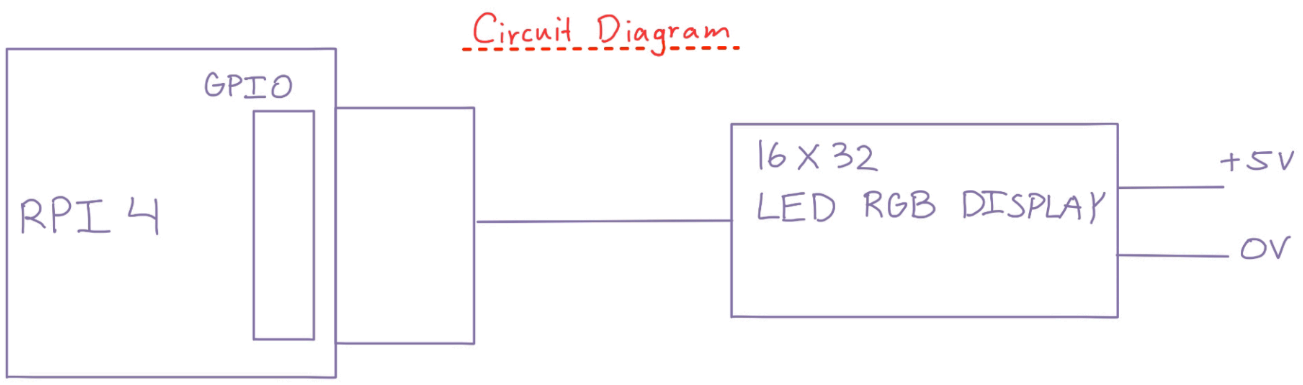
3 CONSTRUCTION

Connection	Pin	Pin	Connection
-	1	2	-
	3	4	-
	5	6	**GND**
strobe	7	8	
	9	10	
clock	11	12	**OE-**
G1	13	14	
A	15	16	**B**
	17	18	**C**
B2	19	20	
G2	21	22	**D**

R1	23	24	**R2**
	25	26	**B1**

ADAFRUIT 16x32 RGB LED MATRIX WIRING DIAGRAM





4 PROJECT BREAK-DOWN

Our group consists of 2 people. Jinyuan Zhao [Student] and Mary Li [Student]

Mary, with her expertise in hardware wiring and wire management, took on the job of constructing the physical aspect of the project and handled the programming of one of the two games. With attention to detail, she ensured the correct wiring and powering of the display.

Jinyuan took charge of most of the coding aspects of the project. With prior knowledge of Linux, Jinyuan solved all the software issues without great difficulty. Jinyuan's ability to debug, learn new concepts quickly, and write efficient code became a strong foundation for the project. By leveraging his coding expertise, Jinyuan ensured a stable and reliable software implementation for controlling the project as well as research into unfamiliar territories such as the RGB LED Matrix, and successfully integrated them.

The collaboration between Mary and Jinyuan proved instrumental in the success of the project. Mary's wiring skills complemented Jinyuan's coding proficiency, enabling them to tackle both the physical construction and the programming challenges with ease. Their combined knowledge and expertise created a well-rounded team capable of overcoming obstacles and achieving the desired functionality of the LED RGB Matrix Games.

5 COMPLETED PROJECT VS PROPOSAL

In transforming our proposal into a finished project, we encountered several deviations that shaped our final creation.

Initially, we envisioned an interface featuring joysticks and buttons, aiming for a tactile and interactive experience. However, as previously detailed, technical constraints led us to pivot to keyboard-based controls. This adaptation, while not our first choice, demonstrated our flexibility in problem-solving.

We also aspired to implement a dynamic default screen and a comprehensive game menu, enhancing user engagement. Time constraints, however, limited our ability to realize these features, reminding us of the importance of project management and prioritization.

Our most ambitious goal was to link two displays, adaptable in orientation based on the game selected. Whether stacked or side-by-side, this configuration would have offered a unique gaming experience. Yet, practical limitations guided us to focus on a single-display setup, which still delivered an enjoyable and functional gaming environment.

These deviations from our original proposal were not just challenges, but opportunities for learning and growth, underscoring the dynamic nature of project development.

Appendix A

GitHub - hzeller/rpi-rgb-led-matrix: Controlling up to three chains of 64x64, 32x32, 16x32 or similar RGB LED displays using Raspberry Pi GPIO

Medium 16x32 RGB LED matrix panel - 6mm Pitch : ID 420 : $24.95

https://cdn-learn.adafruit.com/downloads/pdf/raspberry-pi-led-matrix-display.pdf

Our program code is located at

https://github.com/1zhaojin2/rpi-lgb-led-matrix-game/tree/main

Appendix B

Adafruit 16x32 RGB LED Matrix x1

Jumper Wires - Male to Female x16

Raspberry Pi 4B x1

Compatible keyboard, mouse and monitor for Raspberry Pi x1

Appendix C

Python
```python
# getch.py
# by Jinyuan Zhao
# Jan 15, 2024
# This file serves as input for the rest of the game code

import sys
import tty
import termios
import atexit

def restore_settings(saved_settings):
       termios.tcsetattr(sys.stdin.fileno(), termios.TCSADRAIN,
       saved_settings)

def getch():
       fd = sys.stdin.fileno()
       old_settings = termios.tcgetattr(fd)
       try:
       tty.setraw(sys.stdin.fileno())
       ch = sys.stdin.read(1)
       finally:
       restore_settings(old_settings)
       return ch

# Restore terminal settings at exit
atexit.register(restore_settings, termios.tcgetattr(sys.stdin.fileno()))
```

Python

```python
# snake.py
# by Jinyuan Zhao
# Jan 16, 2024
# This is the code for the snake game

from rgbmatrix import RGBMatrix, RGBMatrixOptions, graphics
import getch
import threading
import time
import random

# Matrix configuration
options = RGBMatrixOptions()
options.rows = 16
options.cols = 32
options.chain_length = 1
options.parallel = 1
options.hardware_mapping = 'regular'
matrix = RGBMatrix(options=options)


# Game setup
snake = [(1, 3), (1, 4), (1, 5)]
food_pos = (random.randint(0, 15), random.randint(0, 31))
direction = "RIGHT" # Initial direction
input_direction = "RIGHT" # Direction based on keyboard input

# Gradient colors
start_color = graphics.Color(255, 0, 0) # Red
end_color = graphics.Color(0, 0, 255)   # Blue

# Food color
food_color = graphics.Color(255, 255, 0) # Yellow

def interpolate_color(start_color, end_color, ratio):
      new_red = int(start_color.red * (1 - ratio) + end_color.red * ratio)
      new_green = int(start_color.green * (1 - ratio) + end_color.green *
      ratio)
      new_blue = int(start_color.blue * (1 - ratio) + end_color.blue * ratio)
      return graphics.Color(new_red, new_green, new_blue)

def draw_snake(canvas):
      snake_length = len(snake)
      for i, segment in enumerate(snake):
      ratio = i / snake_length
      color = interpolate_color(start_color, end_color, ratio)
      canvas.SetPixel(segment[1], segment[0], color.red, color.green,
      color.blue)

def draw_food(canvas):
      canvas.SetPixel(food_pos[1], food_pos[0], food_color.red,
      food_color.green, food_color.blue)

def update_food_position():
      global food_pos
      while True:
      new_food_pos = (random.randint(0, 15), random.randint(0, 31))
      if new_food_pos not in snake:
        food_pos = new_food_pos
        break

def read_keyboard_input():
        global input_direction
        while True:
        char = getch.getch() # Get a single character from keyboard
        if char.lower() == 'w':
        input_direction = "UP"
        elif char.lower() == 'a':
        input_direction = "LEFT"
        elif char.lower() == 's':
        input_direction = "DOWN"
        elif char.lower() == 'd':
        input_direction = "RIGHT"

input_thread = threading.Thread(target=read_keyboard_input)
input_thread.daemon = True # Daemonize thread
input_thread.start()

direction = "RIGHT" # Set an initial direction
input_direction = direction # Synchronize input direction with initial
        direction

try:
        while True:
        print(f"Current direction: {direction}, Input direction:
        {input_direction}")

        direction = input_direction

        # Update snake position
        head = snake[0]
        print(f"Current head position: {head}")

        if direction == "RIGHT":
        head = (head[0], head[1] + 1)
        elif direction == "LEFT":
        head = (head[0], head[1] - 1)
        elif direction == "UP":
        head = (head[0] - 1, head[1])
        elif direction == "DOWN":
        head = (head[0] + 1, head[1])

        print(f"New head position: {head}")

        # Check for game over conditions
        if not (0 <= head[0] < 16 and 0 <= head[1] < 32):
        print("Game over condition met")
        break # Snake hits the wall or itself

        snake.insert(0, head)

        # Check for food collision
        if head == food_pos:
        print("Food collision")
        update_food_position() # Place new food
        else:
        snake.pop() # Move the snake

        # Rendering
        canvas = matrix.CreateFrameCanvas()
        draw_snake(canvas)
        draw_food(canvas)
        canvas = matrix.SwapOnVSync(canvas)

        time.sleep(0.1) # Control game speed
except KeyboardInterrupt:
        print("Game stopped by user")
except Exception as e:
        print("An error occurred:", e)
finally:
        print("Exiting game...")
```

Python

```
# dino.py
# by Mary Li
# Jan 16, 2024
# This is the program for Mary's Dinosaur game
```

```
from rgbmatrix import RGBMatrix, RGBMatrixOptions, graphics
from time import sleep
import random
import getch # Import the getch module
import threading

# Matrix configuration
options = RGBMatrixOptions()
options.rows = 16
options.cols = 32
options.chain_length = 1
options.parallel = 1
options.hardware_mapping = 'regular'
matrix = RGBMatrix(options=options)

# Game setup
dino_pos = [14, 3] # Initial position of the dino
obstacles = []
score = 0
is_on_ground = True

# Colors
dino_color = graphics.Color(255, 0, 0) # Red
obstacle_color = graphics.Color(0, 255, 0) # Green

def draw_dino(canvas):
      canvas.SetPixel(dino_pos[1], dino_pos[0], dino_color.red,
      dino_color.green, dino_color.blue)

def draw_obstacles(canvas):
      for obstacle in obstacles:
      canvas.SetPixel(obstacle[1], obstacle[0], obstacle_color.red,
      obstacle_color.green, obstacle_color.blue)

def update_obstacles():
      # This function now only spawns new obstacles
      while True:
      sleep_time = random.uniform(1.0, 2.0)
      sleep(sleep_time)
      obstacles.append([14, 31]) # Spawn new obstacle at the right edge
```

```
def read_keyboard_input():
        while True:
        char = getch.getch()
        if char.lower() == ' ':
        jump_thread = threading.Thread(target=perform_jump)
        jump_thread.start()

# Modify the perform_jump function
def perform_jump():
        global is_on_ground
        if is_on_ground:
        is_on_ground = False
        # Ascent with decreasing sleep time (faster movement)
        ascent_intervals = [0.1, 0.08, 0.05, 0.03, 0.01]
        for interval in ascent_intervals:
        dino_pos[0] -= 1
        sleep(interval)

        # Descent with increasing sleep time (faster movement)
        descent_intervals = [0.01, 0.03, 0.05, 0.08, 0.1]
        for interval in descent_intervals:
        dino_pos[0] += 1
        sleep(interval)

        is_on_ground = True


def draw_score(canvas):
        global score
        score_str = str(score)

        # Load the font from the correct path
        font = graphics.Font()
        font.LoadFont("/home/esk/rpi-rgb-led-matrix/fonts/6x10.bdf")

        textColor = graphics.Color(255, 255, 0) # Bright yellow color for
        visibility
        x_pos = options.cols - len(score_str) * 8 # Adjust x_pos based on
        score length
        y_pos = 9 # Set y_pos to a value within the upper area of the
        matrix

        graphics.DrawText(canvas, font, x_pos, y_pos, textColor, score_str)


input_thread = threading.Thread(target=read_keyboard_input)
input_thread.daemon = True # Daemonize thread
input_thread.start()

obstacle_thread = threading.Thread(target=update_obstacles)
obstacle_thread.daemon = True
obstacle_thread.start()

try:
        while True:
        # Move obstacles to the left
        for obstacle in obstacles:
        obstacle[1] -= 1 # Move each obstacle left

        # Remove obstacles that have left the screen
        obstacles = [obstacle for obstacle in obstacles if obstacle[1] >= 0]

        # Check for game over conditions
        if [dino_pos[0], dino_pos[1]] in obstacles:
        break # Dino hits an obstacle

        # Check for game over conditions
        if [dino_pos[0], dino_pos[1]] in obstacles:
        break # Dino hits an obstacle

        for obstacle in obstacles:
        if obstacle[1] == dino_pos[1] - 1 and dino_pos[0] < 14:
        score += 1 # Increment score when dino jumps over an obstacle
        print(f'Score: {score}')


        # Rendering
        canvas = matrix.CreateFrameCanvas()
        draw_dino(canvas)
        draw_obstacles(canvas)
        draw_score(canvas) # Draw the score
        canvas = matrix.SwapOnVSync(canvas)

        sleep(0.06) # Control game speed

except KeyboardInterrupt:
        print("Game stopped by user")
except Exception as e:
        print("An error occurred:", e)
finally:
        print("Exiting game...")
```

Python

```python
# lgb_test.py
# by Jinyuan Zhao
# Jan 12, 2024
# This is a test file to see if our display was wired correctly, and if
        the python library was correctly installed

from rgbmatrix import RGBMatrix, RGBMatrixOptions, graphics

# Configuration for the matrix
options = RGBMatrixOptions()
options.rows = 16
options.cols = 32
options.chain_length = 1
options.parallel = 1
options.hardware_mapping = 'regular' # If you're using an Adafruit hat,
        use 'adafruit-hat'


matrix = RGBMatrix(options = options)

# Create a canvas to draw on
canvas = matrix.CreateFrameCanvas()

# Define the square properties
square_size = 6
```

```
start_x = (canvas.width - square_size) // 2
start_y = (canvas.height - square_size) // 2
color = graphics.Color(255, 0, 0) # Red color

# Draw a square
for x in range(start_x, start_x + square_size):
      for y in range(start_y, start_y + square_size):
      canvas.SetPixel(x, y, color.red, color.green, color.blue)

# Update the display
canvas = matrix.SwapOnVSync(canvas)

# Keep the square displayed
import time
time.sleep(10)
```