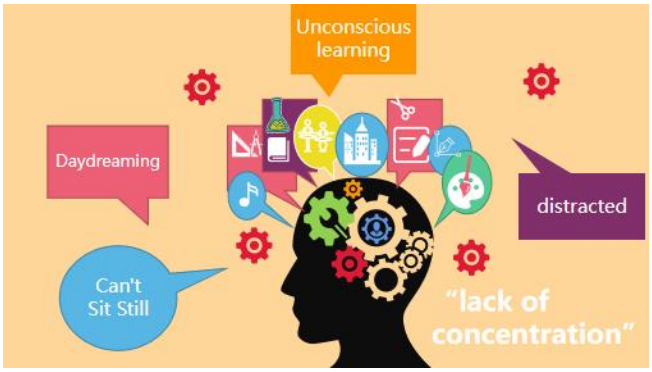
Lesson 7 Schulte Grid Game

Attention is the foundation of all learning and is a fundamental requirement for the brain's perception, learning, thinking, and cognitive activities. However, both children and adults often struggle with distractions and difficulty concentrating. It is essential to find a suitable method to train our attention.

The Schulte Grid is one of the simplest, most effective, and scientifically proven methods for attention training used worldwide.

The Schulte Grid is a square card divided into 25 squares, each measuring 1cm * 1cm. The squares are filled with Arabic numerals 1 to 25 in random order, with each number appearing only once. During the training, the participant is required to sequentially locate and verbally recite the position of each number while using their finger. The examiner records the time taken to complete all 25 numbers. The shorter the time taken, the higher the level of attention.

Let's design a Schulte Grid game together to train our attention!



Task Objectives

Play the Schulte Grid game on the screen.

Knowledge Points

1. Learning how to load images using the pygame library.
2. Learning how to play sound effects using the pygame library.
3. Learning how to implement mouse interaction using the pygame library.

Materials List

Hardware List:

		
UNIHIKER x 1	Type-C & Micro Dual-Use USB Cablex1	USB speakerx1

Knowledge background

Common methods in the pygame library's image module.

The pygame image module is primarily used for image processing. In programming, you can use the syntax "module_name.method_name()" to perform various functionalities.

(1) The load() method loads an image file.

The load() method allows you to load an image from a specified location.

```
screen.blit(pygame.image.load("pic/start-5.png"), (30, 190)) # Display the image start-5.png at co
ordinates (30, 190)
```

In the above example, "pic/start-5.png" represents the path and file name of the specific image. Here, it refers to an image file named start-5.png located in the pic folder. After loading the image, you can use the blit() method to display it at the desired position.

1. Common methods in the pygame music module

The pygame music module is closely related to the mixer module and allows for control over audio and sound.

(1) The load() method loads an audio file.

The load() method is used to load an audio file.

```
wavFileName = 'sounds/fire.wav' # Set the path to the sound file
sndTrack = pygame.mixer.music.load(wavFileName) # Load the sound file
```

In the above example, wavFileName refers to the audio file path. sndTrack is a variable used to store the loaded audio object.

(2) The play() method plays the audio.

The play() method is used to play the loaded audio file.

```
pygame.mixer.music.play() # Play the music
```

2. Common methods in the pygame mouse module

The pygame mouse module is used to obtain the current state of the mouse device. In this game, we simulate mouse control by touching the screen. When programming, you can use the "module_name.method_name()" format to achieve the desired functionality.

(1) The get_pos() method retrieves the mouse position.

The get_pos() method is used to retrieve the x and y coordinates of the mouse's current position.

```
t_x, t_y = pygame.mouse.get_pos() # Retrieve the x and y coordinates of the mouse position and store them in variables t_x and t_y.
```

In the above example, t_x and t_y are variables used to store the obtained x and y coordinates of the mouse position.

3. Mouse Events and Event Detection in the pygame Library

The mouse is one of the most important peripheral devices for computers and an essential tool for gamers. For example, in games, we often need to perform actions like clicking and releasing on images, which require the use of a mouse.

Pygame provides three mouse events: MOUSEMOTION (mouse movement), MOUSEBUTTONDOWN (mouse button press), and MOUSEBUTTONUP (mouse button release). Each event type has different attributes.

For all three mouse events, pygame provides a "pos" attribute, which represents the current coordinates (x, y) of the mouse relative to the top-left corner of the window.

To implement mouse control in a game, we need to first detect the events and then examine their properties.

```
if event.type == pygame.MOUSEBUTTONUP and 30 <= t_x <= 200 and 190 <= t_y <= 250: # If the mouse is released and its coordinates are within the range of the "Start Game" image.
```

Here, event.type represents the type of event, pygame.MOUSEBUTTONUP represents the mouse release event, and 30 <= t_x <= 200 and 190 <= t_y <= 250 represents that the mouse's x-coordinate is between 30 and 200, and the y-coordinate is between 190 and 250, indicating that the mouse is within the range of the "Start Game" image.

4. What is the numpy library and its common functions

Numpy is a scientific computing library in Python that is commonly used for array manipulation. When programming, after importing the library using "import numpy," you can use functions by calling "numpy.function_name()".

(1) The array() function creates an array

We can create an array using the array() function.

```
Xpts = [0, 48, 96, 144, 192]  # x-coordinates
Ypts = [0, 48, 96, 144, 192]  # y-coordinates
ha = itertools.product(Xpts, Ypts)  # Generates all possible combinations of x and y coordinates, resulting in 25 sets of data
haha = list(ha)  # Convert the generated combinations into a list
'''
Output:
[(0, 0), (0, 48), (0, 96), (0, 144), (0, 192), (48, 0), (48, 48), (48, 96), (48, 144),
(48, 192), (96, 0), (96, 48), (96, 96), (96, 144), (96, 192), (144, 0), (144, 48), (144, 96),
(144, 144), (144, 192), (192, 0), (192, 48), (192, 96), (192, 144), (192, 192)]
'''
map = np.array(haha)  # Convert the list of combinations into an array
```

In the above code snippet, we first create two lists, Xpts and Ypts, to store the x and y coordinates. Then, using the product() function from the itertools library, we generate all possible combinations of the values from the two lists and convert the result into a list. Finally, we use the array() function from the numpy library to convert the list into an array and store it in the variable "map".

5. The time() function in the time library to check the current time

The time() function in the time library can be used to check the time and return the current timestamp.

```
time_start = time.time()  # Start timing
time_end = time.time()  # End timing
time_c = round(time_end - time_start, 1)  # Calculate the time taken, rounded to 1 decimal place
```

In the code snippet above, we use the time.time() function to record the current time at the start and end points. Then, we calculate the difference between the two timestamps to get the elapsed time. The round() function is used to round the result to 1 decimal place.

Hands-on Practice

Task Description 1: Creating a Game Window and Start Interface

Create a game window using the pygame library and display the game's start interface on it.

1. Hardware Setup

Connect the UNIHIKER to the computer using a USB cable.

2. Program Coding

STEP 1: Creating and Saving Project Files

Launch Mind+ and save the project with the name "007. Schulte_Grid_Game".

STEP 2: Creating and Saving Python File

Create a Python program file named "main1.py" and double-click to open it.

STEP 3: Importing Resources

Import two resource folders, including images and sound effects, into the project folder. Follow these steps:

(1) Drag and drop the resources into the project folder

From the 'pictures' folder, drag and drop the 'Pic' and 'Sound' folders into this area.

(2) After dragging and dropping, you should see the following:

STEP 4: Programming

(1) Import the necessary libraries

In this task, we need to use the pygame library to create the game window. Therefore, we need to import it.

```
import pygame # Import the pygame library
```

(2) Initialize the game and create a game window with specified dimensions

When using pygame for games, we need to initialize it first. Then, to match the UNIHIKER screen, we create a game window with a size of (240, 320).

```
pygame.init()  # Initialize pygame
width = 240   # Define the width
height = 320   # Define the height
size = (width, height)  # Define the size
screen = pygame.display.set_mode(size)  # Create the game window with a size of (240, 320)
```

(3) Define the game start page

After creating the game window, we define a game start page. On this start page, we need to iterate through all events and set the functionality to exit the game when the window is closed. Additionally, we display a background image saying "Start Game".

```
# Define the start page
def start(start_page):
    while start_page:  # When on the start page
        for event in pygame.event.get():  # Iterate through all events
            if event.type == pygame.QUIT:  # If the window is closed, exit
                pygame.quit()  # Quit pygame
            screen.blit(pygame.image.load("pic/start-5.png"), (30, 190))  # Display the image "start-5.png" at (30,190)
        pygame.display.flip()  # Update the entire display
```

(4) Display the start page

To keep the start page displayed at all times, we use a variable to track the state of the start page and initially set it to True. Then, we use a loop to continuously display the start page.

```
start_page = True  # Set the initial state of the start page to True
while True:  # Main game loop
        start(start_page)  # Call the start function
```

Tips: The complete example program is as follows:

```
'''Creating the Window and Displaying the Start Page'''

import pygame  # Import the pygame library

pygame.init()  # Initialize pygame
width = 240  # Define the width
height = 320  # Define the height
size = (240, 320)  # Define the size
screen = pygame.display.set_mode(size)  # Create the game window with size (240, 320)

# Define the start page
def start(start_page):
    while start_page:  # When entering the start page
        for event in pygame.event.get():  # Iterate through all events
            if event.type == pygame.QUIT:  # If the window is closed
                pygame.quit()  # Quit pygame
            screen.blit(pygame.image.load("pic/start-5.png"), (30, 190))  # Display the image "start-
```

```
5.png" at coordinates (30, 190)
        pygame.display.flip()  # Update the entire display

start_page = True  # Set the initial state of the start page to True
while True:  # Main game loop
   start(start_page)  # Call the start function
```

3. Running the Program

STEP 1: Remote connection to UNIHIKER

STEP 2: Click the "Run" button in the upper right corner

STEP 3: Observe the result

Observe the UNIHIKER board and you will see the words "Start Game" displayed on the screen, which is the start page we set up before entering the game.

Tips: Press and hold the Home button on the UNIHIKER for 5 seconds to exit the program.

Task Description 2: Enter the Game Interface

The previous start page was a static image. To enhance the game, we will now add dynamic effects to it. When the mouse pointer (or finger) moves to the text area, the content will turn green. And when clicked, it will enter the game interface.

1. Program Coding

STEP 1: Create and Save Project Files

Create a new Python program file named "main2.py" and double-click to open it.

STEP 2: Program Coding

(1) Import the necessary libraries and create the game window.

```
import pygame  # Import pygame librar
import random  # Import random library
import numpy as np  # Import numpy library
import itertools  # Import itertools library

pygame.init()  # Initialize pygame
width = 240    # Define width
```

```
height = 320     # Define height
size = (240, 320)  # Define size
screen = pygame.display.set_mode(size)  # Create game window with size (240, 320)
```

(2) Setting Image Coordinates

In this game, we will click on 25 images with numerical symbols in sequential order. Therefore, in order to display these 25 number images on the screen, we need to determine their coordinates.

Here, we set the size of each number image to 48x48 pixels. Therefore, on the horizontal axis of the screen, we can display exactly 5 images in a row, and the same goes for each column vertically. Additionally, the x-coordinate of each column of number images ranges from 0 to 4 times 48, and the y-coordinate follows the same pattern.

To achieve this in programming, we can generate all possible combinations of the two lists [0, 48, 96, 144, 192] for x and y coordinates, resulting in 25 sets of coordinate data. We then represent these coordinates as an array to represent the positions of the 25 number images on the screen.

```
# Set image coordinates
Xpts = [0, 48, 96, 144, 192]  # x coordinates
Ypts = [0, 48, 96, 144, 192]  # y coordinates
# map = np.array(list(itertools.product(Xpts, Ypts)))  # Image coordinates for 25 images
ha = itertools.product(Xpts, Ypts)  # Generate all combinations of x and y coordinates, resulting in 25 sets of data
haha = list(ha)  # Convert the resulting combinations to a list
'''Result: [(0, 0), (0, 48), (0, 96), (0, 144), (0, 192), (48, 0), (48, 48), (48, 96), (48, 144),
 (48, 192), (96, 0), (96, 48), (96, 96), (96, 144), (96, 192), (144, 0), (144, 48), (144, 96),
 (144, 144), (144, 192), (192, 0), (192, 48), (192, 96), (192, 144), (192, 192)]'''
map = np.array(haha)  # Convert the list of coordinates to an array
```

(3) Define a preparation function to determine the images

After determining the coordinates for the 25 number images, how can we determine which images to display randomly?

Here, we use a little trick. We name the 25 number images to be initially displayed as pic0.png to pic24.png, and the images to be displayed after being clicked as qic0.png to qic24.png.

With this naming convention, we can represent the two sets of images, each with unique images, using the same prefix name (pic/qic) followed by different numbers from 0 to 24 and the png format. In programming, we can represent these numbers as a list.

Therefore, here we will create a list and shuffle its elements randomly so that we can use them to represent the sequence part of the image names.

```
# Define the preparation function to determine the images
def ready():
    global list1  # Define a global variable list1
    list1 = [[i] for i in range(25)]  # List comprehension to generate another list based on the comp
rehension of an existing list
    '''Result:[[0], [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12],
    [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24]]'''
    random.shuffle(list1)  # Randomly shuffle all elements in the list
```

(4) Detecting Mouse Position

Next, we will add the functionality to get the mouse position on the start page. We will set it up so that when the cursor (mouse) hovers over the "Start Game" image, we can switch to another image with the same text but a different color, giving the impression that the text has changed color.

```
global t_x, t_y  # Define two global variables t_x and t_y
            t_x, t_y = pygame.mouse.get_pos()  # Get the x and y coordinates of the mouse and store
them in variables t_x and t_y
            if 30 <= t_x <= 200 and 190 <= t_y <= 250:  # If the mouse is within the range of the "Sta
rt Game" image
                screen.blit(pygame.image.load("pic/start-
6.png"), (30, 190))  # Switch the image to start-6.png at coordinates (30,190)
            if event.type == pygame.MOUSEBUTTONUP and 30 <= t_x <= 200 and 190 <= t_y <= 25
0: # If the mouse button is released and the mouse is within the range of the "Start Game" image
                start_page = False  # Set the start page state to False, exit the start page
                game_page = True    # Set the game page state to True, enter the game page


            pygame.display.flip()  # Update the entire display
```

(5) Detecting Mouse Events

Next, we will add mouse detection events to the start page. We will set it up so that when the mouse is released and moved within the range of the "Start Game" image, it will transition to the game page and exit the start page.

Here, we use two variables, start_page and game_page, to mark the entry and exit states of the start page and game page, respectively.

```
            if event.type == pygame.MOUSEBUTTONUP and 30 <= t_x <= 200 and 190 <= t_y <= 25
```

```
0: # If the mouse button is released and the mouse is within the range of the "Start Game" image
                start_page = False  # Set the start page state to False, exit the start page
                game_page = True    # Set the game page state to True, enter the game page


                pygame.display.flip()  # Update the entire display
```

(6) Define Game Interface

Next, we define the game interface, where the state is set to display the 25 initial number images. The images are represented using a combination of a prefix, the index from the list, and the .png file extension.

The coordinates of the images are represented as an array. Additionally, we use a variable called pic_zero to track the state of number display, where 1 indicates the number is ready to be displayed and 0 indicates that it does not need to be displayed.

```
# Define the game interface
def gamePage(game_page):
    pic_zero = 1  # Question page state, ensures that the questions are displayed only once
    while game_page: # When in the game interface
        while pic_zero: # When the question page state is 1
            for i in range(25):  # Loop 25 times
                screen.blit(pygame.image.load("pic/pic" + str(*list1[i -
 1]) + ".png"), map[i]) # Display the initial images of the specified 25 numbers at the coordinates given in the map array
            pic_zero = 0 # Set the question page state to 0 (indicating that all 25 number images have been displayed)
            pygame.display.flip() # Update the entire display
```

(7) Loop Execution

Finally, we set the initial states of start_page and game_page to True, and call them within a loop to ensure that the window remains displayed.

```
start_page = True # Set the initial start page state to True
game_page = True  # Set the initial game page state to True
while True: # Loop indefinitely
    ready() # Call the ready function
    start(start_page) # Call the start function
```

```
    gamePage(game_page) # Call the gamePage function
```

Tips: The complete example program is as follows:

```
'''Display game interface Click "Start Game" to enter the game interface'''
import pygame  # Import pygame librar
import random  # Import random library
import numpy as np  # Import numpy library
import itertools  # Import itertools library

pygame.init()  # Initialize pygame
width = 240    # Define width
height = 320    # Define height
size = (240, 320)  # Define size
screen = pygame.display.set_mode(size)  # Create game window with size (240, 320)

# Set image coordinates
Xpts = [0, 48, 96, 144, 192]  # x coordinates
Ypts = [0, 48, 96, 144, 192]  # y coordinates
# map = np.array(list(itertools.product(Xpts, Ypts)))  # Image coordinates for 25 images
ha = itertools.product(Xpts, Ypts)  # Generate all combinations of x and y coordinates, resulting in
 25 sets of data
haha = list(ha)  # Convert the resulting combinations to a list
'''Result: [(0, 0), (0, 48), (0, 96), (0, 144), (0, 192), (48, 0), (48, 48), (48, 96), (48, 144),
 (48, 192), (96, 0), (96, 48), (96, 96), (96, 144), (96, 192), (144, 0), (144, 48), (144, 96),
 (144, 144), (144, 192), (192, 0), (192, 48), (192, 96), (192, 144), (192, 192)]'''
map = np.array(haha)  # Convert the list of coordinates to an array

# Define the preparation function to determine the images
def ready():
    global list1  # Define a global variable list1
    list1 = [[i] for i in range(25)]  # List comprehension to generate another list based on the compr
ehension of an existing list
    '''Result:[[0], [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12],
    [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24]]'''
    random.shuffle(list1)  # Randomly shuffle all elements in the list
```

```
# Define the start page
def start(start_page):
    while start_page:  # When in the start page
        for event in pygame.event.get():  # Iterate through all events
            if event.type == pygame.QUIT:  # If the close window button is clicked, quit the program
                pygame.quit()  # Quit pygame
            screen.blit(pygame.image.load("pic/start-5.png"), (30, 190))  # Display the image start-5.png at coordinates (30,190)

            global t_x, t_y  # Define two global variables t_x and t_y
            t_x, t_y = pygame.mouse.get_pos()  # Get the x and y coordinates of the mouse and store them in variables t_x and t_y
            if 30 <= t_x <= 200 and 190 <= t_y <= 250:  # If the mouse is within the range of the "Start Game" image
                screen.blit(pygame.image.load("pic/start-6.png"), (30, 190))  # Switch the image to start-6.png at coordinates (30,190)
            if event.type == pygame.MOUSEBUTTONUP and 30 <= t_x <= 200 and 190 <= t_y <= 250: # If the mouse button is released and the mouse is within the range of the "Start Game" image
                start_page = False  # Set the start page state to False, exit the start page
                game_page = True    # Set the game page state to True, enter the game page
            pygame.display.flip()  # Update the entire display

# Define the game interface
def gamePage(game_page):
    pic_zero = 1  # Question page state, ensures that the questions are displayed only once
    while game_page: # When in the game interface
        while pic_zero: # When the question page state is 1
            for i in range(25):  # Loop 25 times
                screen.blit(pygame.image.load("pic/pic" + str(*list1[i - 1]) + ".png"), map[i]) # Display the initial images of the specified 25 numbers at the coordinates given in the map array
            pic_zero = 0 # Set the question page state to 0 (indicating that all 25 number images have been displayed)
            pygame.display.flip() # Update the entire display
```

```
start_page = True # Set the initial start page state to True
game_page = True  # Set the initial game page state to True
while True: # Loop indefinitely
    ready() # Call the ready function
    start(start_page) # Call the start function
    gamePage(game_page) # Call the gamePage function
```

2. Running the Program

STEP 1: Remote connection to UNIHIKER

STEP 2: Run the Program and Observe the Effects

After clicking the "Run" button, observe the Row Board. You will see that when you move your finger and the cursor over the "Start Game" icon, the text changes from blue to green. When you move away, it reverts back to blue. When you click on the green "Start Game" icon, you will enter the game interface, where a set of gray randomly ordered numeric images from 1 to 25 will be displayed.

Tips: Press and hold the Home button on the UNIHIKER for 5 seconds to exit the program.

Task Description 3: Setting Up Game Mechanics

Next, we will set up the complete game mechanics to achieve color switching of numbered images when clicked in order. We will also record the time taken to click all the images, where shorter time indicates higher concentration.

1. Program Coding

STEP 1: Creating and Saving Project Files

Create a new Python program file named "main3.py" and open it.

STEP 2: Programming

(1) Adding Sound Effects and Text Objects

In order to further enhance the game experience, we will add background sound effects to the program mentioned above. When the numbered images are not clicked in the correct order, a background sound will be triggered. Additionally, we will create a font object to display the elapsed time.

```
# Load sound effect
```

```
wavFileName = 'sounds/fire.wav'  # Set the sound effect file path
sndTrack = pygame.mixer.music.load(wavFileName)  # Load the sound effect file
# Timer text preparation
font = pygame.font.SysFont('Arial', 60)  # Create a Font object for the timer text
```

(2) Start Timer

To accurately record the time, we will add code to the start page program to start the timer once the game page is entered. Additionally, to ensure that the time is only displayed after the game is over, we will fill the screen with black.

```
    global time_start  # Define the global variable for starting time
    screen.fill((0, 0, 0))  # Fill the screen with black
    time_start = time.time()  # Start the timer, returns the current time as a timestamp
```

(3) Set the Mechanism

Next, we will add a mechanism in the game interface to switch to another image with the same number but a different color when a number image is clicked at the same position.

```
for i in range(25):  # Loop 25 times
            screen.blit(pygame.image.load("pic/pic" + str(*list1[i -
 1]) + ".png"), map[i])  # Display the initial images of the specified 25 numbers at the coordinates in the map
        pic_zero = 0  # Set the problem setting page status to 0 (indicating that all 25 images of numbers have been displayed)
    for event in pygame.event.get():  # Traverse all events
        if event.type == pygame.QUIT:  # If the window is closed, quit
            pygame.quit()  # Quit pygame
        for i in range(25):  # Loop 25 times
            # If the mouse is released and within the range of a certain number image
            if event.type == pygame.MOUSEBUTTONUP and map[i][0] <= event.pos[0] <= map[i][0] + 48 and map[i][1] <= event.pos[1] <= map[i][1] + 48:
                if int(*list1[i-1]) == zero:  # If the clicked image is the 0th image in the list
                    screen.blit(pygame.image.load("pic/qic" + str(*list1[i-
1]) + ".png"), map[i])  # Display the specified image of the clicked number at the coordinates in the map
                    zero = zero + 1  # Increase the number by 1
                    print(zero)  # Print the number
```

(4) Displaying Time

After that, when the last number image (25) is clicked, we will display the recorded time on the screen. Here, to ensure that the text is centered, we first create a rectangular area centered at a given position, and then display the time text on it.

```
if zero == 25:  # If the number reaches 25
                    time_end = time.time()  # Stop timing
                    time_c = round(time_end -
 time_start, 1)  # Calculate the time taken, rounded to 1 decimal place
                    print('time cost:', int(time_c), 's')  # Print the time taken in seconds
                    text = font.render(str(time_c) + 's', True, (0, 255, 0), (0, 0, 128))  # Render the tex
t about the time, with green text color and blue background color
                    text_rect = text.get_rect(center=(120, 290))  # Create a text-
filled rectangle with the given position (120,290) as the center
                    screen.blit(text,text_rect)  # Display the time text on the filled rectangle
```

(5) Restart the game

After one round of the game is over, we set it to display the start game interface again so that the game can be restarted.

```
                if event.type == pygame.MOUSEBUTTONUP and 30 <= t_x <= 210 and 200 <= t
_y <= 250:  # If the mouse is released and moves to the "Start Game" image range
                    start_page = True  # Set the start page status to True, enter the start page
                    game_page = False  # Set the game page status to False, exit the game page
```

(6) Looping Call

Finally, we add the sound effect to play in the background when the numbers are not clicked in the correct order.

```
                pygame.display.flip()  # Update all displays
```

Tips: The complete example program is as follows:

```
'''Set up game mechanics for a complete Schulte Grid game where the player needs to click on th
e images of numbers 1-25 in sequence and measure the time'''

import pygame # Import the pygame library
import random # Import the random library
import numpy as np # Import the numpy library
```

```
import itertools # Import the itertools library
import time # Import the time library

pygame.init()  # Initialize pygame
width = 240  # Define the width
height = 320  # Define the height
size = (240, 320)  # Define the size
screen = pygame.display.set_mode(size)  # Create the game window with the size (240,320)

# Load sound effect
wavFileName = 'sounds/fire.wav'  # Set the sound effect file path
sndTrack = pygame.mixer.music.load(wavFileName)  # Load the sound effect file
# Timer text preparation
font = pygame.font.SysFont('Arial', 60)  # Create a Font object for the timer text

# Set image coordinates
Xpts = [0, 48, 96, 144, 192]  # x coordinates
Ypts = [0, 48, 96, 144, 192]  # y coordinates
# map = np.array(list(itertools.product(Xpts, Ypts)))  # 25 image coordinates
ha = itertools.product(Xpts, Ypts)  # Generate all permutations of x and y coordinates, resulting in
 25 sets of data
haha = list(ha)  # Convert the permutations to a list
'''Result: [(0, 0), (0, 48), (0, 96), (0, 144), (0, 192), (48, 0), (48, 48), (48, 96), (48, 144),
 (48, 192), (96, 0), (96, 48), (96, 96), (96, 144), (96, 192), (144, 0), (144, 48), (144, 96),
 (144, 144), (144, 192), (192, 0), (192, 48), (192, 96), (192, 144), (192, 192)]'''
map = np.array(haha)  # Convert the list data to an array

# Define the ready function to determine the image sequence
def ready():
    global list1  # Define a global variable list1
    list1 = [[i] for i in range(25)]  # List comprehension to generate a new list based on another list
    '''Result: [[0], [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12],
    [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24]]'''
    random.shuffle(list1)  # Shuffle all elements in the list
```

```
# Define the start page
def start(start_page):
    while start_page:  # When on the start page
        for event in pygame.event.get():  # Iterate through all events
            if event.type == pygame.QUIT:  # If the window is closed, exit
                pygame.quit()  # Quit pygame
            screen.blit(pygame.image.load("pic/start-5.png"), (30, 190))  # Display the image "start-5.png" at (30,190)

            global t_x, t_y  # Define two global variables t_x and t_y
            t_x, t_y = pygame.mouse.get_pos()  # Get the x and y coordinates of the mouse and store them in variables t_x and t_y
            if 30 <= t_x <= 200 and 190 <= t_y <= 250:  # If the mouse is within the range of the "start game" image
                screen.blit(pygame.image.load("pic/start-6.png"), (30, 190))  # Switch the image to "start-6.png" at (30,190)
            if event.type == pygame.MOUSEBUTTONUP and 30 <= t_x <= 200 and 190 <= t_y <= 250:  # If the mouse is released and the coordinates are within the range of the "start game" image
                start_page = False  # Set the start page status to False, exit the start page
                game_page = True  # Set the game page status to True, enter the game page
                global time_start  # Define the global variable for starting time
                screen.fill((0, 0, 0))  # Fill the screen with black
                time_start = time.time()  # Start the timer, returns the current time as a timestamp
            pygame.display.flip()  # Update the display

# Define the game interface
def gamePage(game_page):
    zero = 0  # Variable added here to ensure starting from the smallest number
    pic_zero = 1  # Problem setting page status to ensure only one problem is displayed
    while game_page:  # When entering the game page
        while pic_zero:  # When the problem setting page status is 1
            for i in range(25):  # Loop 25 times
                screen.blit(pygame.image.load("pic/pic" + str(*list1[i - 1]) + ".png"), map[i])  # Display the initial images of the specified 25 numbers at the coordinates in the map
            pic_zero = 0  # Set the problem setting page status to 0 (indicating that all 25 images of n
```

```
umbers have been displayed)
        for event in pygame.event.get():  # Traverse all events
            if event.type == pygame.QUIT:  # If the window is closed, quit
                pygame.quit()  # Quit pygame
            for i in range(25):  # Loop 25 times
                # If the mouse is released and within the range of a certain number image
                if event.type == pygame.MOUSEBUTTONUP and map[i][0] <= event.pos[0] <= map[i][0] + 48 and map[i][1] <= event.pos[1] <= map[i][1] + 48:
                    if int(*list1[i-1]) == zero:  # If the clicked image is the 0th image in the list
                        screen.blit(pygame.image.load("pic/qic" + str(*list1[i-1]) + ".png"), map[i])  # Display the specified image of the clicked number at the coordinates in the map
                        zero = zero + 1  # Increase the number by 1
                        print(zero)  # Print the number

                        if zero == 25:  # If the number reaches 25
                            time_end = time.time()  # Stop timing
                            time_c = round(time_end - time_start, 1)  # Calculate the time taken, rounded to 1 decimal place
                            print('time cost:', int(time_c), 's')  # Print the time taken in seconds
                            text = font.render(str(time_c) + 's', True, (0, 255, 0), (0, 0, 128))  # Render the text about the time, with green text color and blue background color
                            text_rect = text.get_rect(center=(120, 290))  # Create a text-filled rectangle with the given position (120,290) as the center
                            screen.blit(text,text_rect)  # Display the time text on the filled rectangle
                            #screen.blit(text, (40, 250))  # Display the time text at position (40,250) on the window
                            if event.type == pygame.MOUSEBUTTONUP and 30 <= t_x <= 210 and 200 <= t_y <= 250:  # If the mouse is released and moves to the "Start Game" image range
                                start_page = True  # Set the start page status to True, enter the start page
                                game_page = False  # Set the game page status to False, exit the game page
                            pygame.display.flip()  # Update all displays
                    else:
                        pygame.mixer.music.play()  # Play music when there is an error
            pygame.display.flip()  # Update all displays
```

```
start_page = True  # Define the initial start page status as True
game_page = True  # Define the initial game page status as True
while True:  # Loop
    ready()  # Call the ready function
    start(start_page)  # Call the start function
    gamePage(game_page)  # Call the gamePage function
```

2. Running the Program

STEP 1: Remote connection to UNIHIKER

STEP 2: Connect the USB Speaker

Connect the USB speaker to the side USB port of the UNIHIKER.

STEP 3: Run the Program and Observe the Effects

After clicking "Run," observe the UNIHIKER. When you touch the green "Start Game" icon, the game interface will be displayed. Then, click the number images in order. The clicked numbers will turn pink. If you click the numbers out of order, the background music will play from the speaker.

After completing a game, you can click on the screen again to display the text "Start Game" and continue playing.

Challenge Yourself

1. Compare yourself and see how much time it takes to complete three consecutive games!
2. Download a music track from the internet and add it as background music. How would you adjust the program?