

## Lesson 6 Snake Game

In a very long time ago, when mobile phone screens were still in the black and white era, a game called "Snake" became popular on the streets and alleys. On those low-resolution screens, a few twisted curves seemed to cross the entire childhood.

In this lesson, let's use Scratch to implement this classic game!



### Task Objectives

Play the Snake game on the screen.





### Knowledge points

1. Introduction to Pygame library
2. Learn how to create a game window using Pygame library
3. Learn how to draw graphics, text, and update screen display using Pygame library
4. Learn how to implement keyboard interactions using Pygame library.

## Material List

### Hardware List:

	
UNIHKER x 1	Type-C & Micro Dual-Use USB Cable x 1

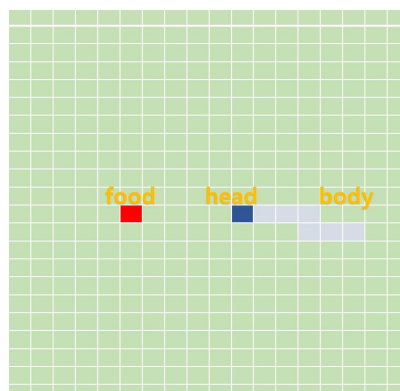
**Software Preparation:** Mind+ Programming Software x 1

## Knowledge background

### 1. Snake Game Implementation Principle and Logic

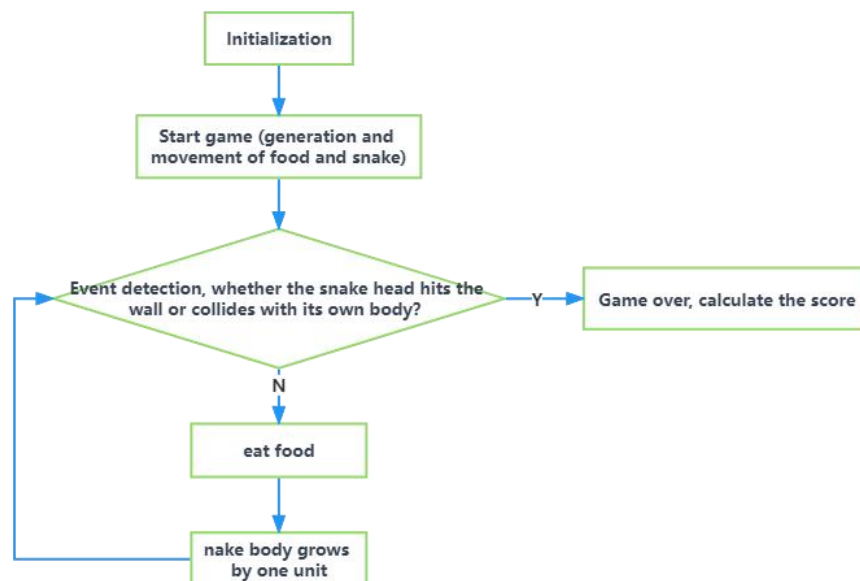
#### (1) Implementation principle

In this Snake game, we divide the entire game area into small squares, and the location of each square can be represented by row and column. A group of connected small squares form a "snake", which is divided into "head" and "body". The "head" is represented by one square, and the "body" is stored in a list. Combined with different colors, a "snake" is formed. The movement of the "snake" is achieved by adding the row and column positions of the next square to the beginning of the list and removing the last element of the list, which is equivalent to the "snake" moving forward one square. The food is randomly presented in the form of squares. When the position of the "head" overlaps with the food, the "snake" eats the food. When the "snake" moves beyond the range or the "head" collides with the "body", the game is over and the number of food is counted.



#### (2) Game logic diagram

Since the game includes multiple functions such as snake movement, food consumption, event monitoring, and game over, we can define each function as a separate function and call it according to the game logic at the required location during programming.



## 2. What is the pygame library

Pygame is a cross-platform Python library designed for electronic game development. With it, we can design electronic games that include elements such as graphics, sound, and more. The advantage of using pygame for game development is that developers don't need to focus too much on low-level details and can instead focus on game logic. Pygame integrates many low-level modules such as accessing display devices, managing events, using fonts, and more. For commonly used modules in pygame, please refer to the table below.

Module Name	Functions
pygame.display	Accessing display device
pygame.event	Managing events
pygame.draw	Drawing shapes, lines and points
pygame.key	Reading keyboard keys
pygame.mouse	Controlling mouse events
pygame.music	Playing audio

## 3. Common functions in Pygame library

There are many functions in the Pygame library, but we only use some of them. When programming, we can use the format "pygame.function name()" by importing the library with "import pygame" to achieve the desired function.

(1) The `init()` function initializes the Pygame module

With the `init()` function, we can initialize the modules in Pygame. In programming, we need to place this command before other Pygame commands, so that the modules can be used after initialization.

```
pygame.init() # Initialize Pygame
```

(2) The `quit()` function to exit pygame library

The `quit()` function is the opposite of the `init()` function, which can exit pygame and terminate its operation. In programming, we usually use it when we need to end the game.

```
pygame.quit() # Exit pygame
```

#### 4. Commonly used methods in the pygame display module

The pygame display module can be used to access the display device for displaying content. There are many methods in the module, but we only use a part of them. During programming, we can implement the functions by using the format 'module\_name.method\_name()'.

(1) `set_mode()` method initializes a window interface ready for display

The `set_mode()` method can create a game window.

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

Here, `size` is the size we set for the game window, which is the same as the UNHIKER screen. `window` is a generated screen Surface object that we can fill with color, paint, add other objects, and perform various operations on.

When filling with color, we can use the "`object.fill()`" command to achieve this.

```
bg_color = (255, 255, 255) # Define the background color as white  
window.fill(bg_color) # fillcolor # Fill the window with the background color
```

To add other objects to the window object, we can use the "`object.blit()`" command to achieve this.

```
window.blit(score, (40, 250)) # Display the score at (40,250) on the window
```

(2) The `flip()` method updates the screen.

The `flip()` method updates the screen with the graphics that are ready to be displayed. Typically, after writing some functions using the pygame.display module, we need to use the flip method to update and display the graphics on the screen.

```
pygame.display.flip() # Refresh all displays to the window # Update all graphics waiting to be  
displayed to the screen
```

#### 5. Common methods in the pygame draw module

The pygame display module can be used to draw various shapes, and there are many methods available to achieve this in programming using the form of 'module\_name.method\_name()'.

(1) The rect() method is used to draw a rectangle

The rect() method is used to draw a rectangle. Here is an example:

```
left = point.col * 15 # Define the distance from the left edge of the small grid
top = point.row * 15 # Define the distance from the top edge of the small grid
pygame.draw.rect(window, color, (left, top, 15, 15)) # Draw a rectangle on the window with
color as the color
```

In this example, window represents the window on which the rectangle is drawn, color refers to the color of the rectangle, left and top respectively refer to the distance between the rectangle and the left and top edges of the window, used to represent the position of the rectangle area.

## 6. Common methods in the pygame font module

The pygame font module enables the use of fonts, and there are many methods available that can be implemented in programming using the format "module\_name.method\_name()".

(1) The SysFont() method creates a font object

The SysFont() method can create a font object.

```
font = pygame.font.SysFont('Arial', 20) # Set the font
```

Here, Arial refers to a specific font type and 20 refers to the font size. Font is a variable used to store the generated font object. After creating the font object, we can draw specific text on it to achieve the display effect. We can use the "object.render()" command to implement it.

```
score = font.render('Your Score is ', False, 'pink') # Calculate the score
```

Here, 'Your Score is ' refers to the specific text to be drawn, False indicates no anti-aliasing is required, pink refers to the color of the text, and score is a variable used to store the generated text.

## 7. In pygame, events are crucial modules that constitute the core of the entire game program.

Events such as mouse clicks, keyboard inputs, game window movements, and game exits can all be considered as "events". Pygame accepts various user operations or events, which can occur at any time and in varying quantities. How does pygame handle these events? Moreover, what are the event types and keyboard events in pygame and how can event detection be performed?

Pygame defines a structure specifically designed to handle events, known as an event queue. This structure follows the basic principle of "first in, first out" for handling events. Through the event queue, we can process user operations (triggered events) in an orderly and sequential manner. For commonly used game events in Pygame, refer to the table below:

Event Type	Description	Member Attributes
------------	-------------	-------------------

QUIT	User presses the close button of the window	None
KEYDOWN	Keyboard key is pressed	unicode, key, mod
KEYUP	Keyboard key is released	key, mod
MOUSEBUTTONDOWN	Mouse button is pressed	pos, button
MOUSEBUTTONUP	Mouse button is released	pos, button

**Tips:** It should be noted that not all of the aforementioned events will be applicable when using pygame for game development.

Keyboard events, among them, will involve a vast number of key operations, such as the game's movement in up, down, left, and right directions or the character's forward and backward movements, all of which require keyboard cooperation for their realization.

Keyboard events provide a "key" attribute that allows for the retrieval of the keyboard's key. Pygame defines the letter keys, numeric keys, and combination keys on the keyboard as constants. The following table provides information on some of the most frequently used constant keys.

The constant names	descriptions are as follows
K_SPACE	the space bar key (Space)
K_RETURN	the enter key (Enter)
K_0...K_9	the numerical keys from 0 to 9
K_a...Kz	the alphabetical keys from a to z
K_UP	the upward-pointing arrow key (up arrow)
K_DOWN	the downward-pointing arrow key (down arrow)
K_LEFT	the leftward-pointing arrow key (left arrow)
K_RIGHT	the rightward-pointing arrow key (right arrow)

**Tips:** UNIHKER has already been mapped to the keyboard keys "a" and "b" during development, so pressing the "a" and "b" keys on UNIHKER can also be detected when getting keyboard events.

Finally, if we want to implement keyboard control of the game, we need to detect events first. So how can we achieve this? The pygame library's event module provides common methods for handling event queues. We can use the get() method to detect events.

```
events = pygame.event.get() # retrieves events
```

which are stored in the "events" variable. After getting the event, we can perform a check on it.

```
if (event.type == pygame.QUIT): # If the event type is to quit (close window)
```

```
pygame.quit() # Quit the game

if (event.type == pygame.KEYDOWN): # If the event type is a keyboard press

    if (event.key == pygame.K_a): # If the key "a" is pressed

        .....

    elif (event.key == pygame.K_b): # If the key "b" is pressed
```

Event.type represents the type of the event, where pygame.QUIT denotes the exit event, pygame.KEYDOWN indicates a key press event, event.key refers to the pressed key on the keyboard, pygame.K\_a denotes the key 'a', and pygame.K\_b denotes the key 'b'.

## Hands-on practice

### Task description 1: Creating the game window.

Create a game window interface using the pygame library.

#### 1. Hardware setup

Connect the UNIHAKER to the computer via a USB cable.

#### 2. program coding

##### STEP1: Creating and Saving Project Files

Launch Mind+ and save the project as "006. Snake Game".

##### STEP2: Creating and Saving Python Files

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

##### STEP3: Programming

###### (1) Import Required Libraries

For this task, we need to use pygame and time libraries to create the game window, so we need to import them first.

```
import pygame # Import the pygame library

import time # Import the time library
```

###### (2) Initialize the game and create a game window with a specific size

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

```
pygame.init() # Initialize the game
```

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

(3) Define background color and initial running status

After creating the game window, we define a background color and initial running status for it to facilitate future settings.

```
bg_color = (255, 255, 255) # Define the background color as white
run = True # Define the initial running state as True, indicating running
```

(4) Fill the background color of the window and keep it displayed

Next, we will fill the window's background with a color. To ensure that the window remains displayed, we need to continuously refresh the content shown on the window screen. Here, we will achieve this by incorporating a loop.

```
while run: # Game loop
    window.fill(bg_color) # Fill the window with the background color
    pygame.display.flip() # Update all the content to the screen
    time.sleep(0.2) # Delay for 0.2 second
```

**Tips:** The complete example program is as follows:

```
'''Create the window'''

import pygame # Import the pygame library
import time # Import the time library

pygame.init() # Initialize the game
W = 240 # Define the width
H = 320 # Define the height
size = (240, 320) # Define the size
window = pygame.display.set_mode(size) # Create the game window with a size of (240, 320)
bg_color = (255, 255, 255) # Define the background color as white
run = True # Define the initial running state as True, indicating running

while run: # Game loop
    window.fill(bg_color) # Fill the window with the background color
    pygame.display.flip() # Update all the content to the screen
```



```
time.sleep(0.2) # Delay for 0.2 seconds
```

### 3. Running the Program

**STEP 1:** Remote connection to UNIIKER

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

**STEP 3:** Observe the Effect

Observe the UNIIKER. Firstly, you will notice that the screen of the UNIIKER turns white, and this is exactly the background color of the game window we created.



### Task Description 2: Adding Game Elements

The generated screen window mentioned above is currently empty. To enhance the game, we will now add the most important character elements of the game - the snake and the food - to it.

#### 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 Writing

Similar to Task 1, at the beginning of the program, we still import the required libraries, initialize the game, and create the game window. Then, we continue writing the remaining parts of the program based on this foundation.

(1) Define the Game Grid

In the game of Snake, the two most important elements are the snake and the food. So how can we represent them on our screen?

Here, we divide the game window into small squares to represent the elements on the screen. Each small square has a width and height of 15 pixels. As a result, since the window size is 240x320, there will be 18 squares horizontally and 24 squares vertically, forming a grid of 24 rows and 18 columns.

```
# Divide the game window into small squares, each with a width and height of 15, totaling 24 rows and 18 columns
ROW=24 # Define the number of rows, 24 rows and 18 columns for each square of size 15x15
COL=18 # Define the number of columns
cell_width=W/COL # Define the width of each cell
cell_height=H/ROW # Define the height of each cell
```

## (2) Define Grid Positions

Next, we define a Point class to represent the squares as points, using rows and columns to indicate their positions.

```
# Define the Point class to represent the position of a square
class Point:
    row = 0 # Row
    col = 0 # Column
    def __init__(self,row,col): # Row, Column
        self.row=row
        self.col=col
```

## (3) Define Positions and Colors for the Snake Head, Snake Body, and Food

Next, we divide the snake into two parts: the snake head and the snake body. We set their initial positions as occupying one square and three squares respectively. The food is also represented as a single element occupying one square.

We proceed by instantiating the classes to define their positions, as well as setting their respective colors. The position of the snake body is represented using a list. The specific process is as follows.

```
#Define the positions and colors of the snake head, body, and food
head = Point(row=int(ROW/2), col=int(COL/2)) # Define the position of the snake head in the 12th row, 9th column
snakes = [
    Point(row=head.row, col=head.col+1), # Define the position of snake body 1 in the 12th row, 10th column
    Point(row=head.row, col=head.col+2), # Define the position of snake body 2 in the 12th row, 11th column
    Point(row=head.row, col=head.col+3) # Define the position of snake body 3 in the 12th row, 12th column
```

```

]
food = Point(row=2, col=3) # Define the position of the food in the 2nd row, 3rd column
head_color = (65, 105, 225) # Define the color of the snake head
snake_color = (204, 204, 204) # Define the color of the snake body
food_color = (255, 10, 10) # Define the color of the food

```

#### (4) Define the Drawing Method for Each Small Square

Next, we use the pygame library to draw rectangles in order to create the small squares. Since the size of each small square is the same, with only the position and color varying, we can define a general function for drawing squares. This function takes the position and color as parameters, and we can determine the position of each square based on its corresponding row and column.

```

# Define the drawing of each small square (takes two parameters: position and color)
def rect(point, color):
    left = point.col * cell_width # Define the distance of the small square from the left edge
    top = point.row * cell_height # Define the distance of the small square from the top edge
    pygame.draw.rect(window, color, (left, top, cell_width, cell_height)) # Draw a rectangle on the window with the given color

```

#### (5) Instantiate the Star Class and Finish Drawing

Finally, we instantiate the Star class at the specified positions and draw squares of the specified colors to represent the snake head, snake body, and food. In order to keep them continuously visible, we place this code within a loop.

```

while run: # Game loop
    window.fill(bg_color) # Fill the window with the background color

    # Define the snake body, snake head, and food using small squares
    for snake in snakes:
        rect(snake, snake_color) # Create the snake body with the specified color
        rect(head, head_color) # Create the snake head with the specified color
        rect(food, food_color) # Create the food with the specified color

    pygame.display.flip() # Update all pending displays to the window
    time.sleep(0.2) # Delay for 0.2 seconds

```

**Tips:** The complete example program is as follows:

```

'''Add snake and food to the game window'''
import pygame # Import the pygame library
import time # Import the time library

```

```

pygame.init() # Initialize the game
W = 240 # Define the width
H = 320 # Define the height
size = (240, 320) # Define the size
window = pygame.display.set_mode(size) # Create the game window with a size of (240, 320)
bg_color = (255, 255, 255) # Define the background color as white

# Divide the game window into small squares, each with a width and height of 15, totaling 24
rows and 18 columns
ROW=24 # Define the number of rows, 24 rows and 18 columns for each square of size 15x1
5
COL=18 # Define the number of columns
cell_width=W/COL # Define the width of each cell
cell_height=H/ROW # Define the height of each cell

# Define the Point class to represent the position of a square
class Point:
    row = 0 # Row
    col = 0 # Column
    def __init__(self,row,col): # Row, Column
        self.row=row
        self.col=col

#Define the positions and colors of the snake head, body, and food
head = Point(row=int(ROW/2), col=int(COL/2)) # Define the position of the snake head in the
12th row, 9th column
snakes = [
    Point(row=head.row, col=head.col+1), # Define the position of snake body 1 in the 12th ro
w, 10th column
    Point(row=head.row, col=head.col+2), # Define the position of snake body 2 in the 12th ro
w, 11th column
    Point(row=head.row, col=head.col+3) # Define the position of snake body 3 in the 12th ro
w, 12th column
]
food = Point(row=2, col=3) # Define the position of the food in the 2nd row, 3rd column
head_color = (65, 105, 225) # Define the color of the snake head
snake_color = (204, 204, 204) # Define the color of the snake body

```

```

food_color = (255, 10, 10) # Define the color of the food

# Define the drawing of each small square (takes two parameters: position and color)
def rect(point, color):
    left = point.col * cell_width # Define the distance of the small square from the left edge
    top = point.row * cell_height # Define the distance of the small square from the top edge
    pygame.draw.rect(window, color, (left, top, cell_width, cell_height)) # Draw a rectangle on t
he window with the given color

run = True # Define the initial running state as True, indicating the game is running
while run: # Game loop
    window.fill(bg_color) # Fill the window with the background color

# Define the snake body, snake head, and food using small squares
for snake in snakes:
    rect(snake, snake_color) # Create the snake body with the specified color
    rect(head, head_color) # Create the snake head with the specified color
    rect(food, food_color) # Create the food with the specified color

pygame.display.flip() # Update all pending displays to the window
time.sleep(0.2) # Delay for 0.2 seconds

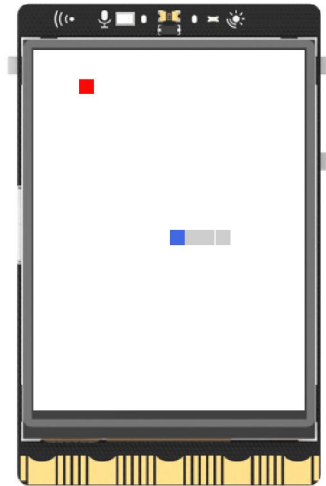
```

## 2. Program Coding

**STEP1:** Remote connection to UNIIKER

**STEP2:** Run the program and observe the effects

After clicking the run button, observe UNIIKER. You will see a snake with a blue head and gray body, as well as a red food displayed within the game window.



### Task Description 3: Setting the Game Mechanics

In the previous task, we added the snake and food to the game interface. However, that's not enough. Now, we will set the game mechanics to allow controlling the snake's left and right movement using the onboard buttons A and B of the UNIIKER. Additionally, we will change the food generation from a fixed position to a random position.

#### 1. Program Coding

##### STEP 1: Create and Save Project File

Create a new Python program file named "main3.py" and open it by double-clicking.

##### STEP 2: Program Coding

We continue to write the program based on Task 2.

##### (1) Defining Random Generation of Food

Here, we modify the previous fixed position of the food and make it generate at a random position within the game window.

```
# Define the position of the food
def gen_food():
    pos = Point(row=random.randint(0, ROW - 1), col=random.randint(0, COL - 1)) # The food is randomly positioned within the window
    return pos
food = gen_food() # Generate the position of the food
```

##### (2) Defining Event Detection

In order to control the movement of the snake using the buttons on the UNIIKER, we first need to define an event detection. The specific process is as follows.

```
# Event Detection
```

```

def detect():
    global direction
    global run
    events = pygame.event.get() # Get events
    for event in events: # Loop through all events
        if event.type == pygame.QUIT: # If the event type is quit (window close)
            pygame.quit() # Quit the game
            run = 0
        if event.type == pygame.KEYDOWN: # If the event type is a key press
            if event.key == pygame.K_a: # If the A key is pressed
                if direction == 'up': # If the current direction is up
                    direction = 'left' # Set the direction to left
                elif direction == 'left':
                    direction = 'down'
                elif direction == 'down':
                    direction = 'right'
                elif direction == 'right':
                    direction = 'up'
            print(direction)

        elif event.key == pygame.K_b: # If the B key is pressed
            if direction == 'up': # If the current direction is up
                direction = 'right' # Set the direction to right
            elif direction == 'right':
                direction = 'down'
            elif direction == 'down':
                direction = 'left'
            elif direction == 'left':
                direction = 'up'
            print(direction)

```

### (3) Defining Movement Method and Setting Initial Direction

Next, we will define the movement method for the snake and set the initial direction to "left". The specific process is as follows.

```

# Movement Definition
def move():
    global direction
    if direction == 'left': # If the direction is left

```

```

        head.col -= 1 # Move the head one cell to the left (decrease the column of the head by 1
    )
    elif direction == 'right':
        head.col += 1
    elif direction == 'up':
        head.row -= 1
    elif direction == 'down':
        head.row += 1
    direction = 'left' # direction is left

```

#### (4) Copying Square Positions

Since the snake's body grows when it eats food, the position of the snake's head becomes the position of the first segment of its body. Therefore, we need to add a "copy" instance method to the Point class to achieve the functionality of copying the position of a square itself, so that we can use it later when the snake eats food.

```

# Define the Point class to represent the position of a square
class Point:
    row = 0 # Row
    col = 0 # Column

    def __init__(self, row, col): # Row, Column
        self.row = row
        self.col = col

    def copy(self): # Copy
        return Point(row=self.row, col=self.col)

```

#### (5) Defining Eating Food

Next, we define the method for the snake to eat food. Here are the specific steps.

```

# Eating Food
def eat():
    global food
    eating = (head.row == food.row and head.col == food.col) # Define eating condition: snake head's position matches the food position
    if eating: # If eating
        food = Point(row=random.randint(0, ROW-1), col=random.randint(0, COL-1)) # Generate a new food at a random position
        snakes.insert(0, head.copy()) # Update snake body: 1. Insert the previous head position at the beginning of the snake body (insert a new element at index 0 in the snakes list)

```



```
if not eating: # If not eating (after eating)
    snakes.pop() # Update snake body: 2. Remove the last element of the snake body (remove the last element in the snakes list)
```

#### (6) Looping Execution

Finally, we put the previously defined event detection, movement, and eating food functions into a loop to execute them indefinitely.

```
while run: # Game loop
    window.fill(bg_color) # Fill the window with background color

    # Define snake body, snake head, and food using small squares
    for snake in snakes:
        rect(snake, snake_color) # Draw snake body with snake_color
    rect(head, head_color) # Draw snake head with head_color
    rect(food, food_color) # Draw food with food_color

    eat() # Eat food
    detect() # Detect events
    move() # Move the snake

    pygame.display.flip() # Update the display
    time.sleep(0.2) # Delay for 0.2 seconds (adjust this value to change snake's movement speed and game difficulty)
```

**Tips:** The complete example program is as follows:

```
'''Control snake movement, generate random food, snake eats food'''

import pygame # Import the pygame library
import random # Import the random library
import time # Import the time library

pygame.init() # Initialize the game
W = 240 # Define the width
H = 320 # Define the height
size = (240, 320) # Define the size
window = pygame.display.set_mode(size) # Create the game window with size (240, 320)
bg_color = (255, 255, 255) # Define the background color as white
# Divide the game window into small squares, each with a width and height of 15, total of 24 rows and 18 columns
```

```

ROW = 24 # Define the number of rows, each square is 15*15, with 24 rows and 18 columns
COL = 18 # Define the number of columns
cell_width = W / COL # Define the width of each cell
cell_height = H / ROW # Define the height of each cell

# Define the Point class to represent the position of a square
class Point:
    row = 0 # Row
    col = 0 # Column

    def __init__(self, row, col): # Row, Column
        self.row = row
        self.col = col

    def copy(self): # Copy
        return Point(row=self.row, col=self.col)

# Define the positions and colors of the snake's head and body
head = Point(row=int(ROW / 2), col=int(COL / 2)) # The snake's head is initially positioned in
the 12th row, 9th column
snakes = [ # The snake's body positions
    Point(row=head.row, col=head.col + 1), # The first body square is in the 12th row, 10th col
umn
    Point(row=head.row, col=head.col + 2), # The second body square is in the 12th row, 11th
column
    Point(row=head.row, col=head.col + 3) # The third body square is in the 12th row, 12th co
lumn
]

# Define the position of the food
def gen_food():
    pos = Point(row=random.randint(0, ROW - 1), col=random.randint(0, COL - 1)) # The food
is randomly positioned within the window
    return pos

food = gen_food() # Generate the position of the food
head_color = (65, 105, 225) # Define the color of the snake's head
snake_color = (204, 204, 204) # Define the color of the snake's body
food_color = (255, 10, 10) # Define the color of the food

```

```

# Define the function to draw each cell (takes two parameters: position and color)
def rect(point, color):
    left = point.col * cell_width # The distance from the left edge of the cell
    top = point.row * cell_height # The distance from the top edge of the cell
    pygame.draw.rect(window, color, (left, top, cell_width, cell_height)) # Draw a rectangle on t
he window with the specified color

# Event Detection
def detect():
    global direction
    global run
    events = pygame.event.get() # Get events
    for event in events: # Loop through all events
        if event.type == pygame.QUIT: # If the event type is quit (window close)
            pygame.quit() # Quit the game
            run = 0
        if event.type == pygame.KEYDOWN: # If the event type is a key press
            if event.key == pygame.K_a: # If the A key is pressed
                if direction == 'up': # If the current direction is up
                    direction = 'left' # Set the direction to left
                elif direction == 'left':
                    direction = 'down'
                elif direction == 'down':
                    direction = 'right'
                elif direction == 'right':
                    direction = 'up'
                print(direction)

            elif event.key == pygame.K_b: # If the B key is pressed
                if direction == 'up': # If the current direction is up
                    direction = 'right' # Set the direction to right
                elif direction == 'right':
                    direction = 'down'
                elif direction == 'down':
                    direction = 'left'
                elif direction == 'left':

```

```

        direction = 'up'
        print(direction)

# Movement Definition
def move():
    global direction
    if direction == 'left': # If the direction is left
        head.col -= 1 # Move the head one cell to the left (decrease the column of the head by 1)
    elif direction == 'right':
        head.col += 1
    elif direction == 'up':
        head.row -= 1
    elif direction == 'down':
        head.row += 1

# Eating Food
def eat():
    global food
    eating = (head.row == food.row and head.col == food.col) # Define eating condition: snake head's position matches the food position
    if eating: # If eating
        food = Point(row=random.randint(0, ROW-1), col=random.randint(0, COL-1)) # Generate a new food at a random position
        snakes.insert(0, head.copy()) # Update snake body: 1. Insert the previous head position at the beginning of the snake body (insert a new element at index 0 in the snakes list)
    if not eating: # If not eating (after eating)
        snakes.pop() # Update snake body: 2. Remove the last element of the snake body (remove the last element in the snakes list)

direction = 'left' # Set initial direction to 'left'
run = True # Set initial run state to True, indicating the game is running
while run: # Game loop
    window.fill(bg_color) # Fill the window with background color

    # Define snake body, snake head, and food using small squares
    for snake in snakes:
        rect(snake, snake_color) # Draw snake body with snake_color

```

```
rect(head, head_color) # Draw snake head with head_color
rect(food, food_color) # Draw food with food_color

eat() # Eat food
detect() # Detect events
move() # Move the snake

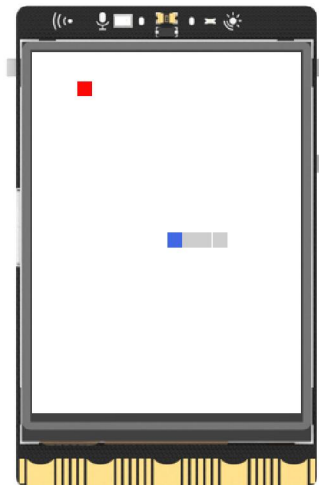
pygame.display.flip() # Update the display
time.sleep(0.2) # Delay for 0.2 seconds (adjust this value to change snake's movement speed and game difficulty)
```

## 2. Running the Program

**STEP1:** Remote connection to UNIIKER

**STEP2:** Run the Program and Observe the Effect

After clicking "run", observe the UNIIKER. You will see a snake with a blue head and gray body displayed in the game window, along with a red food item. The snake will start moving to the left. You can control the snake's left and right turns by pressing the onboard buttons A and B, respectively.



## Task Description 4: End the game and keep score

Finally, we will add the game-ending and scoring mechanism based on the above functionalities.

### 1. Program Coding

**STEP 1:** Create and Save Project File

Create a new Python program file named "main3.py" and open it by double-clicking.

## STEP2: Program Writing

### (1) Import necessary libraries and create a font object

Since we need to display the score, we need to create a font object after defining the background color.

```
font = pygame.font.SysFont('Arial', 20) # Create a font object
```

### (2) Game Over Mechanism

After defining the functionality of eating food, we will now define the game over conditions: either the snake collides with the boundaries of the window or it collides with itself. At the same time, we will display the final game score before ending the game.

```
# Game over function
def game_over():
    global run
    dead = False # Define a dead state
    # Game over condition 1: Hit the wall
    if head.col < 0 or head.row < 0 or head.col >= COL or head.row >= ROW: # If the head is
        outside the screen
        dead = True
    # Game over condition 2: Hit itself
    for snake in snakes:
        if head.col == snake.col and head.row == snake.row: # If the head position is the same as
            a snake body position
            dead = True
            break
    if dead: # If the state is dead
        score = font.render('Your Score is ' + str(10 * len(snakes) - 30), False, 'pink') # Calculate the score
        window.blit(score, (40, 250)) # Display the score on the window
        pygame.display.flip() # Update all displays on the screen
        print("GG") # Good game! ==
        time.sleep(5) # Delay for 5 seconds
        run = False # Set the state to False
```

### (3) Loop Execution

Finally, we will incorporate the defined game over functions into a loop to execute them continuously.

```
while run: # Game loop
```

```

window.fill(bg_color) # Fill the window with the background color

# Use small rectangles to represent the snake body, snake head, and food
for snake in snakes:
    rect(snake, snake_color) # Create snake body with snake_color
rect(head, head_color) # Create snake head with head_color
rect(food, food_color) # Create food with food_color

eat()      # Eat
detect()   # Detect events
move()     # Move
game_over() # Game over check

pygame.display.flip() # Update all pending displays to the screen
time.sleep(0.2) # Delay for 0.2 seconds (You can modify the delay time to adjust the snake's movement speed and game difficulty)

```

**Tips:** The complete example program is as follows:

```

'''Scoring and Game Over'''
import pygame # Import the pygame library
import random # Import the random library
import time   # Import the time library

pygame.init() # Initialize the game
W = 240 # Define the width
H = 320 # Define the height
size = (240, 320) # Define the size
window = pygame.display.set_mode(size) # Create the game window with a size of (240, 320)
bg_color = (255, 255, 255) # Define the background color as white
font = pygame.font.SysFont('Arial', 20) # Create a font object

# Divide the game window into small squares, each square with a width and height of 15, a total of 24 rows and 18 columns
ROW = 24 # Define the number of rows, each square is 15x15, 24 rows and 18 columns in total
COL = 18 # Define the number of columns
cell_width = W / COL # Define the width of a cell

```

```

cell_height = H / ROW # Define the height of a cell

# Define the Point class to represent the position of a cell
class Point:
    row = 0 # Row
    col = 0 # Column
    def __init__(self, row, col): # Row, Column
        self.row = row
        self.col = col
    def copy(self): # Copy
        return Point(row=self.row, col=self.col)

# Define the position and color of the snake's head and body
head = Point(row=int(ROW/2), col=int(COL/2)) # Define the position of the snake's head in the middle of the grid (row 12, column 9)
snakes = [ # Define the snake's body as a list
    Point(row=head.row, col=head.col+1), # Define the position of the first body segment at row 12, column 10
    Point(row=head.row, col=head.col+2), # Define the position of the second body segment at row 12, column 11
    Point(row=head.row, col=head.col+3) # Define the position of the third body segment at row 12, column 12
]

# Define the generation position of the food
def gen_food():
    pos = Point(row=random.randint(0, ROW-1), col=random.randint(0, COL-1)) # Define the position of the food at a random location within the window
    return pos
food = gen_food() # Generate the position of the food
head_color = (65, 105, 225) # Define the color of the snake's head
snake_color = (204, 204, 204) # Define the color of the snake's body
food_color = (255, 10, 10) # Define the color of the food

# Define the drawing of each cell (takes two parameters: position and color)
def rect(point, color):
    left = point.col * cell_width # Define the distance from the left edge of the cell
    top = point.row * cell_height # Define the distance from the top edge of the cell
    pygame.draw.rect(window, color, (left, top, cell_width, cell_height)) # Draw a rectangle on t

```



he window with the specified color

# Define the event detection

def detect():

    global direction

    global run

    events = pygame.event.get() # Get the events

    for event in events: # Iterate over all events

        if event.type == pygame.QUIT: # If the event type is quit (window close)

            pygame.quit() # Quit the game

            run = 0

        if event.type == pygame.KEYDOWN: # If the event type is keydown (key pressed)

            if event.key == pygame.K\_a: # If the 'a' key is pressed

                if direction == 'up': # If the current direction is up

                    direction = 'left' # Set the direction to left

                elif direction == 'left':

                    direction = 'down'

                elif direction == 'down':

                    direction = 'right'

                elif direction == 'right':

                    direction = 'up'

                print(direction)

# Define movement

def move():

    global direction

    if direction == 'left':

        head.col -= 1

    elif direction == 'right':

        head.col += 1

    elif direction == 'up':

        head.row -= 1

    elif direction == 'down':

        head.row += 1

# Define food consumption

def eat():

```

global food
eating = (head.row == food.row and head.col == food.col)
if eating:
    food = Point(row=random.randint(0, ROW - 1), col=random.randint(0, COL - 1))
snakes.insert(0, head.copy())
if not eating:
    snakes.pop()

# Game over function
def game_over():
    global run
    dead = False # Define a dead state
    # Game over condition 1: Hit the wall
    if head.col < 0 or head.row < 0 or head.col >= COL or head.row >= ROW: # If the head is
outside the screen
        dead = True
    # Game over condition 2: Hit itself
    for snake in snakes:
        if head.col == snake.col and head.row == snake.row: # If the head position is the same
as a snake body position
            dead = True
            break
    if dead: # If the state is dead
        score = font.render('Your Score is ' + str(10 * len(snakes) - 30), False, 'pink') # Calculate t
he score
        window.blit(score, (40, 250)) # Display the score on the window
        pygame.display.flip() # Update all displays on the screen
        print("GG") # Good game! ==
        time.sleep(5) # Delay for 5 seconds
        run = False # Set the state to False

direction = 'left' # Initial direction is left
run = True # Initial running state is True, indicating the game is running
while run: # Game loop
    window.fill(bg_color) # Fill the window with the background color

    # Use small rectangles to represent the snake body, snake head, and food

```

```
for snake in snakes:
    rect(snake, snake_color) # Create snake body with snake_color
rect(head, head_color) # Create snake head with head_color
rect(food, food_color) # Create food with food_color

eat()    # Eat
detect() # Detect events
move()   # Move
game_over() # Game over check

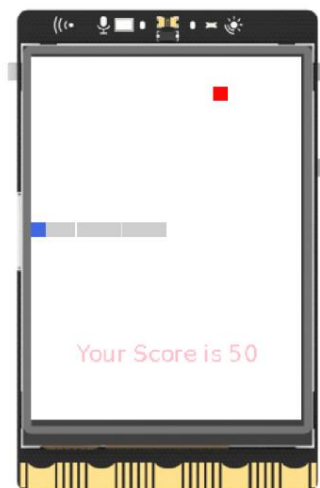
pygame.display.flip() # Update all pending displays to the screen
time.sleep(0.2) # Delay for 0.2 seconds (You can modify the delay time to adjust the snake's
movement speed and game difficulty)
```

## 2. Running the Program

**STEP1:** Remote Connection to UNIHAKER

**STEP2:** Run the Program and Observe the Effect

After running the program, you can control the snake's movement using the onboard buttons A and B. When the snake collides with itself or hits the edges of the window, the game will end, and you will see the final score of the game.



## Challenge Yourself

1. Compare with your classmates and see who can achieve a higher score!
2. Try reducing the screen refresh interval to make the snake move faster and increase the

difficulty. Give it a try!

3. Think about whether we can set levels or stages for different difficulty levels in the game. Modify the program yourself and practice!