BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ESP32: SNAKE GAME

ESP32: HRA HAD

TECHNICAL REPORT

AUTHOR
AUTOR PRÁCE

VÍT MRKVICA

BRNO 2025

Contents

Chapter 1

Introduction

This report describes the design and implementation of a simple embedded game console for the game *HAD* (Snake) using a development platform based on the ESP32 microcontroller. The document focuses on the hardware and software technologies used, the implementation of graphical output and user input handling, and the internal game logic of the application.

1.1 Disclaimer

During the development of this project, AI tools were used to support the design process, explain and refactor source code, and assist in the creation of in-code documentation. Additionally, AI tools were used to improve structure and correct grammar in this technical report. The models used include ChatGPT 5.2 and Claude Opus 4.5.

The display multiplexing implementation is based on an example project provided as part of the assignment. The corresponding files or code sections are clearly marked and documented to distinguish them from original work.

Chapter 2

Used Technologies

2.1 Used Hardware

The following hardware components were used in the implementation of the project:

- **Wemos D1 R32 development board** – a development board based on the SoC ESP32.
- **ESP32 microcontroller**
- **Analog RGB LED Shield** – module containing a 16 × 8 matrix of analog RGB LEDs.
- **74HCT154 decoder** – used for selecting individual columns of the LED matrix.
- **TLC5947 PWM driver** – provides 12-bit PWM control of the RGB LED color channels.
- **Push buttons** – four buttons integrated on the LED shield.

2.2 Used software

The following software tools and frameworks were used during the development of the project:

- **Visual Studio Code**
- **PlatformIO**
- **ESP-IDF**
- **FreeRTOS**

Chapter 3

Design and Implementation

This chapter describes the design of the application and the implementation of its key components. It provides information on how the user input is handled, how the information is displayed and how the game loop is executed.

3.1 Displaying and scanning

The display output is generated using time multiplexing, where only a single column of the display is active at any given time. Columns are refreshed sequentially at a rate sufficiently high to appear continuous to the human eye. The pixel data used for rendering is read from the framebuffer `fb_display`.

The display scan is executed periodically using a timer started with `esp_timer_start_periodic`. The timer period is computed as the desired display refresh rate (50 Hz), divided by the number of columns in the display matrix.

Since the display scan loop and the game update loop operate at different frequencies, and because the display is rendered on a per-column basis, it is possible for columns to originate from different logical frames. To avoid this condition, a double-buffering strategy synchronized to the start of the multiplex cycle is used. The game logic renders each new frame into a separate buffer, `fb_draw`. When the display scan begins rendering column zero and a new frame is available, the display and draw buffers are swapped. This ensures that each complete scan cycle is rendered from a consistent framebuffer.

3.2 Handling User Input

User input is processed using hardware interrupts. To minimize contact bounce effects, a simple debounce mechanism is implemented: interrupt events occurring within a minimum time interval are ignored. The time interval is adjusted dynamically based on the current game state, allowing for responsive input during gameplay while preventing unintended double clicks in menus and other contexts.

To maintain consistent game timing, game state updates are performed periodically. As a consequence, multiple input events occurring between two consecutive game updates may be collapsed, resulting in only the most recent input being applied. This behavior leads to a frustrating user experience.

To address this issue, an input queue is introduced. Each valid button press is translated into a corresponding action and pushed into the queue. During each game update cycle, the game loop processes actions by popping them from the queue. This approach ensures that all user inputs are handled deterministically and in sequence. While this mechanism introduces a small input latency proportional to the current queue length, the delay remains unnoticeable.

3.3 Game

3.3.1 Logic

The game consists of a snake moving across a playing field. The primary objective is to increase the length of the snake by consuming food items. When food is eaten, the snake grows, while consuming evil food causes the snake to shrink. The game is won when the snake reaches a predefined winning length.

The game ends in a loss if the snake collides with its own body or if its length decreases below a predefined minimum length.

3.3.2 Implementation Details

All game logic is executed within a periodic update loop driven by a timer started using `esp_timer_start_periodic`. The update frequency is set to 20 Hz and was determined experimentally.

To support different snake movement speeds, the game maintains an internal frame counter. This allows movement updates to occur at configurable intervals relative to the base game loop frequency.

During each applicable update cycle, the snake's movement direction is determined by the first entry in the input direction queue. If the queue is empty, the snake continues moving in its current direction. The snake is then advanced to its new position, after which collisions with food items and with the snake's own body are evaluated.

When the snake consumes food or evil food, the corresponding growth or shrink value is added to an internal buffer. During each update cycle, the snake's length is adjusted by at most one segment until the buffer is empty.

If applicable, new food items are spawned.

Following the state update, the current game state is rendered into the draw framebuffer `fb_draw`. A flag is then set to notify the display scan loop that a new frame is available.

3.3.3 Configuration

Several game parameters can be configured at compile time. The user can select only from three predefined configurations (difficulties).

- `move_T`: Period, in game ticks, between successive snake movement updates.
- `food_T`: Period, in game ticks, at which new food items are considered for spawning.
- `evil_food_T`: Period, in game ticks, at which evil food items are considered for spawning.
- `food_spawn_chance`: Probability (in percent) that a food item is spawned during a spawn attempt.
- `evil_food_spawn_chance`: Probability (in percent) that an evil food item is spawned during a spawn attempt.
- `max_fruit`: Maximum number of food items allowed existing at once.
- `max_evil_fruit`: Maximum number of evil food items allowed existing at once.
- `fruit_ttl`: Time-to-live of a food item, expressed in game ticks.
- `evil_fruit_ttl`: Time-to-live of an evil food item, expressed in game ticks.
- `winning_len`: Snake length required to win.
- `min_snake_len`: Minimum allowed snake length (must not exceed the display width).
- `good_inc`: Amount by which the snake grows when consuming a food item.
- `evil_dec`: Amount by which the snake shrinks when consuming an evil food item.

Chapter 4

Summary

The implemented application is, to the best of my knowledge, fully functional. Testing was performed at the user level by actively playing the game. A demonstration video[1] illustrating the functionality of the application is provided.

One limitation of the current solution is the use of impractical units in the configuration parameters. Replacing these with real-time units would improve clarity and ease of configuration.

[1] https://drive.google.com/file/d/1JIDkEuWejkd0vD0yS87cSNaHFyuzA0hE/view?usp=drive_link

Bibliography

[1] CONTRIBUTORS, W. *Screen tearing* Online resource. N.d. Available at: https://en.wikipedia.org/wiki/Screen_tearing#Vertical_synchronization. Accessed: 2025-02-16.

[2] SYSTEMS, E. *ESP-IDF Programming Guide for ESP32-C3 (v4.3)* Online resource. 2021. Available at: https://docs.espressif.com/projects/esp-idf/en/v4.3/esp32c3/index.html. Accessed: 2025-02-16.