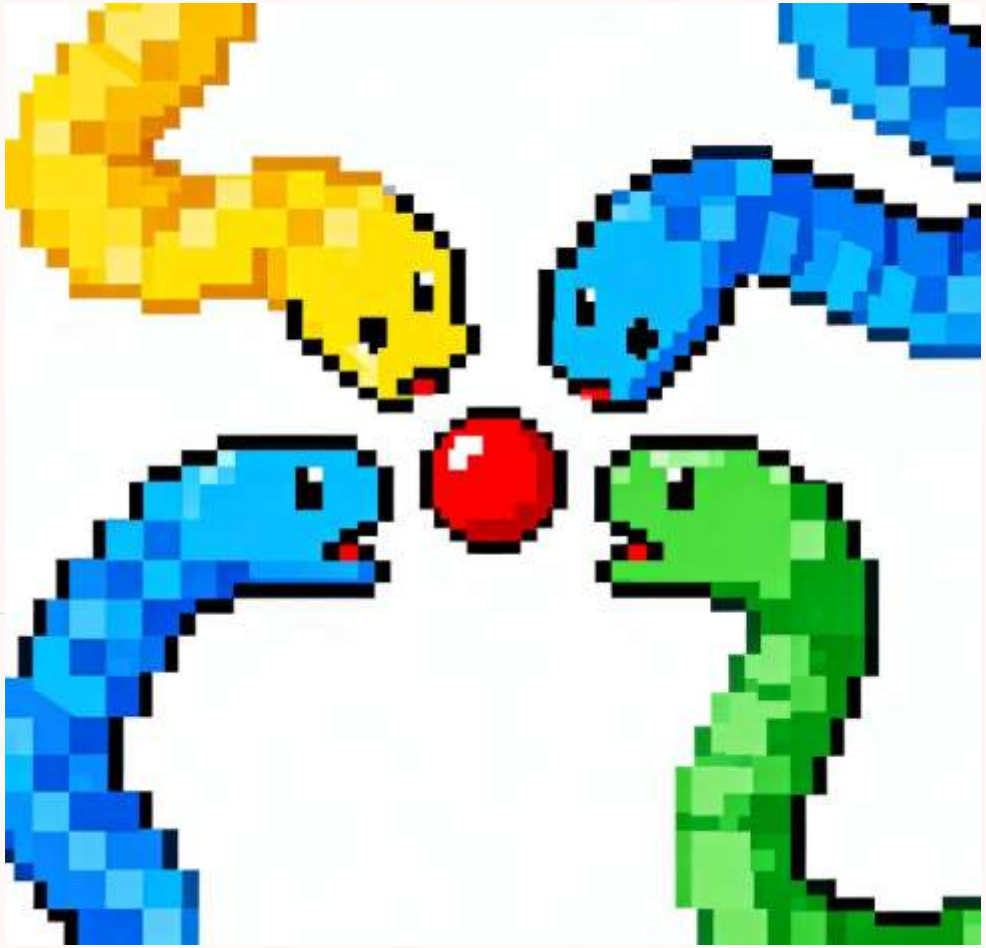
Multiplayer Snake Over the Internet

- Cute Style Multiplayer Snake Game
- Project Overview & Technical Breakdown



Team :12

Team Leader: Xu Hao
Team Member:
Li Fengdu
Zhang Haoyi
Xu Nuo
Yue Xiaotian
Bai Junfu

Video Demonstration

Agenda

Presentation Outline

01 Project Introduction

02 Server Part

03 Backend Part 1: Room Management , Preparation Interface Display

04 Frontend Part 1: Monitoring the backend, status callback

05 Backend Part 2: Backend communication, maintaining synchronization between frontend and backend

06 Frontend Part 2: Maps, snakes, virtual keyboards, and web content generation

07 Frontend Part 3: Module reference, web framework generation

08 Project Difficulties

09 Tools Used

Project Introduction

What is This Project?

- A multiplayer online Snake game with concise visual style
- Real-time synchronization between multiple players
- Key features: Room creation, cross-player interaction, virtual controls
- Tech stack: Rust (backend), WebAssembly/WebSocket (communication)

Server Part

Server Infrastructure

- Hardware Configuration: Cloud server (2vCPU,4GiB, 50GB SSD)
- Environment Setup: Ubuntu 22.04, Rust 1.70+, PostgreSQL

Server Part

Server Infrastructure

- Firewall Configuration: Allow TCP (8080) , TCP(3000) AND TCP (80) ports; Block unused ports

协议	端口范围	来源IP
TCP	80	0.0.0.0/0
TCP	443	0.0.0.0/0
TCP	22	0.0.0.0/0
ICMP	-1（全部）	0.0.0.0/0
TCP	8080	0.0.0.0/0
TCP	3000	0.0.0.0/0

Server Part

Necessary libraries and components

- 前端：yew，serde + serde_json，web-sys，console_error_panic_hook
- 后端：tokio，tokio-tungstenite，rand serde + serde_json，async-trait，once_cell tracing + tracing-subscriber

```toml
[package]
name = "snake-server"
version = "0.1.0"
edition = "2021"

[dependencies]
# WebSocket服务器框架（启用ws特性）
axum = { version = "0.6", features = ["ws"] }
tokio = { version = "1.0", features = ["full"] }

# 数据序列化/反序列化
serde = { version = "1.0", features = ["derive"] }
serde_json = "1.0"

# 异步流处理（正确启用stream/sink所需的特性）
futures-util = { version = "0.3", features = ["std", "sink"] }

# 随机数生成
rand = "0.8"

# 日志调试
tracing = "0.1"
tracing-subscriber = "0.3"

# 同步原语
parking_lot = "0.12"
```

```toml
[package]
name = "snake-game"
version = "0.1.0"
edition = "2021"

[lib]
crate-type = ["cdylib", "rlib"]

[dependencies]
yew = { version = "0.21", features = ["csr"] }
wasm-bindgen = "0.2"
js-sys = "0.3"
web-sys = { version = "0.3", features = ["console", "WebSocket", "MessageEvent", "Event", "HtmlElement", "KeyboardEvent", "MouseEvent", "Window", "Document"] }
serde = { version = "1.0", features = ["derive"] }
serde_json = "1.0"
console_error_panic_hook = "0.1"
gloo-timers = "0.3"

[dev-dependencies]
trunk = "0.17"

[profile.release]
lto = true
```

Backend Part 1: Room Management , Game Content Generation

Room & Game State Handling

Room Management

Room Management Module,Player Preparation Interface

```rust
impl GameRoom {
    pub fn new(room_id: String) -> Self {
        eprintln!("[房间{}] 创建新房间，所需玩家数：{}", room_id, 2);
        Self {
            room_id,
            snakes: HashMap::new(),
            foods: Vec::new(),
            map_size: (30, 30),
            next_snake_id: 1,
            game_started: false,
            game_over: false,
            ready_players: HashSet::new(),
            required_players: 2,
            countdown: None, // 初始无倒计时
            countdown_started: false,
            last_countdown_update: None,
        }
    }

    eprintln!("[房间{}] 玩家加入，分配蛇ID：{}", self.room_id, snake_id);
    snake_id
}
```

Room Creation and Player ID Assignment

```rust
/// 标记玩家准备，返回值：
/// - Some(GameState) ：如果本次准备导致倒计时开始
/// - None ：如果游戏尚未开始（仅更新 ready 状态）
pub fn player_ready(&mut self, snake_id: usize) -> Option<GameState> {
    self.ready_players.insert(snake_id);
    eprintln!("[房间{}] 蛇{} 标记为准备（ready={}/{}）", self.room_id, snake_id, self.ready_players.len(), self.required_players);

    if self.check_start_condition() && !self.countdown_started {
        // 开始倒计时而不是立即开始游戏
        self.countdown = Some(3);
        self.countdown_started = true;
        self.last_countdown_update = Some(Instant::now());
        eprintln!("[房间{}] 所有玩家准备就绪，开始3秒倒计时！", self.room_id);
        Some(self.get_state())
    } else {
        eprintln!("[房间{}] 等待玩家准备：当前准备数{} / 所需{}", self.room_id, self.ready_players.len(), self.required_players);
        None
    }
}

fn check_start_condition(&mut self) -> bool {
    // 只有当在线玩家数等于 required_players 且所有这些玩家均 ready 时才能开始
    let connected = self.snakes.len();
    let ready = self.ready_players.len();
    if connected == self.required_players && ready == self.required_players && !self.game_started {
        true
    } else {
        false
    }
```

Room Preparation Logic and Player Preparation Interface

Backend Part 1: Room Management , Game Content Generation

Room & Game State Handling

Logic Processing

Random Food Spawning and Collision Detection

```
    // 检查碰撞
    let hit_wall = new_head.x < 0 || new_head.x >= self.map_size.0 ||
                   new_head.y < 0 || new_head.y >= self.map_size.1;
    let hit_body = initial_snake_positions.contains(&new_head);

    if hit_wall || hit_body {
        println!("[房间{}] 蛇{} 死亡（撞墙: {}, 撞身体: {}）", self.room_id, snake_id, hit_wall, hit_body);
        new_snake.alive = false;
        updated_snakes.insert(*snake_id, new_snake);
        continue;
    }

    new_snake.body.insert(0, new_head);

    let mut ate_food = false;
    for (i, food) in new_foods.iter().enumerate() {
        if new_head == food.position {
            new_snake.score += 1;
            new_foods.remove(i);
            ate_food = true;
            println!("[房间{}] 蛇{} 吃食物！当前得分: {}", self.room_id, snake_id, new_snake.score);
            break;
        }
    }

    if !ate_food {
        new_snake.body.pop();
    }

    updated_snakes.insert(*snake_id, new_snake);
}

self.snakes = updated_snakes;
self.foods = new_foods;

// 补充食物
let current_positions = self.get_all_snake_positions();
self.generate_foods_with_positions(current_positions, 5);
```

```
    if self.foods.len() < required {
        println!("[房间{}] 警告: 食物生成不足（需要{}, 实际生成{}）", self.room_id, required, self.foods.len());
    }
}

pub fn handle_input(&mut self, snake_id: usize, direction: Direction) {
    if !self.countdown_started || self.game_over {
        println!("[房间{}] 忽略输入: 倒计时未开始或游戏已结束（蛇{}）", self.room_id, snake_id);
        return;
    }

    let snake = self.snakes.get_mut(&snake_id);
    if let Some(snake) = snake {
        if !snake.alive {
            println!("[房间{}] 忽略输入: 蛇{} 已死亡", self.room_id, snake_id);
            return;
```

1,Snake Collision Detection
2,Food Spawning After Consumption
3,Log for Ensuring Loop Continuity

Backend Part 1: Room Management , Game Content Generation

Room & Game State Handling

Game Over and Restart

Game Ranking and Game Repeat Logic

```
    // 检查游戏结束
    let alive_count = self.snakes.values().filter(|s| s.alive).count();
    if alive_count <= 1 {
        self.game_over = true;
        let rankings = self.get_rankings();
        println!("[房间{}] 游戏结束！存活蛇数：{}，排名：{:?}", self.room_id, alive_count, rankings);
    }
}

pub fn get_state(&self) -> GameState {
    GameState {
        room_id: self.room_id.clone(),
        snakes: self.snakes.values().cloned().collect(),
        foods: self.foods.clone(),
        game_started: self.game_started,
        game_over: self.game_over,
        countdown: self.countdown, // 添加倒计时到游戏状态
    }
}
```

Display of Game Rankings and Scores

```
    }

    let mut new_snakes = HashMap::new();
    for (i, &snake_id) in snake_ids.iter().enumerate() {
        let start_pos = new_positions[i];
        new_snakes.insert(
            snake_id,
            Snake {
                id: snake_id,
                body: vec![
                    start_pos,
                    Position { x: start_pos.x - 1, y: start_pos.y },
                    Position { x: start_pos.x - 2, y: start_pos.y },
                ],
                direction: Direction::Right,
                alive: true,
                score: 0,
            }
        );
    }

    self.snakes = new_snakes;
    self.foods.clear();
    self.game_started = false;
    self.game_over = false;
    self.ready_players.clear();
    self.countdown = None;
    self.countdown_started = false;
    self.last_countdown_update = None;

    println!("[房间{}] 重置游戏状态，等待玩家重新准备", self.room_id);
}

pub fn remove_player(&mut self, snake_id: u32) -> Option<Snake> {
    println!("[房间{}] 蛇{} 离开房间", self.room_id, snake_id);
    let removed = self.snakes.remove(&snake_id);
    self.ready_players.remove(&snake_id);
    removed
}
```

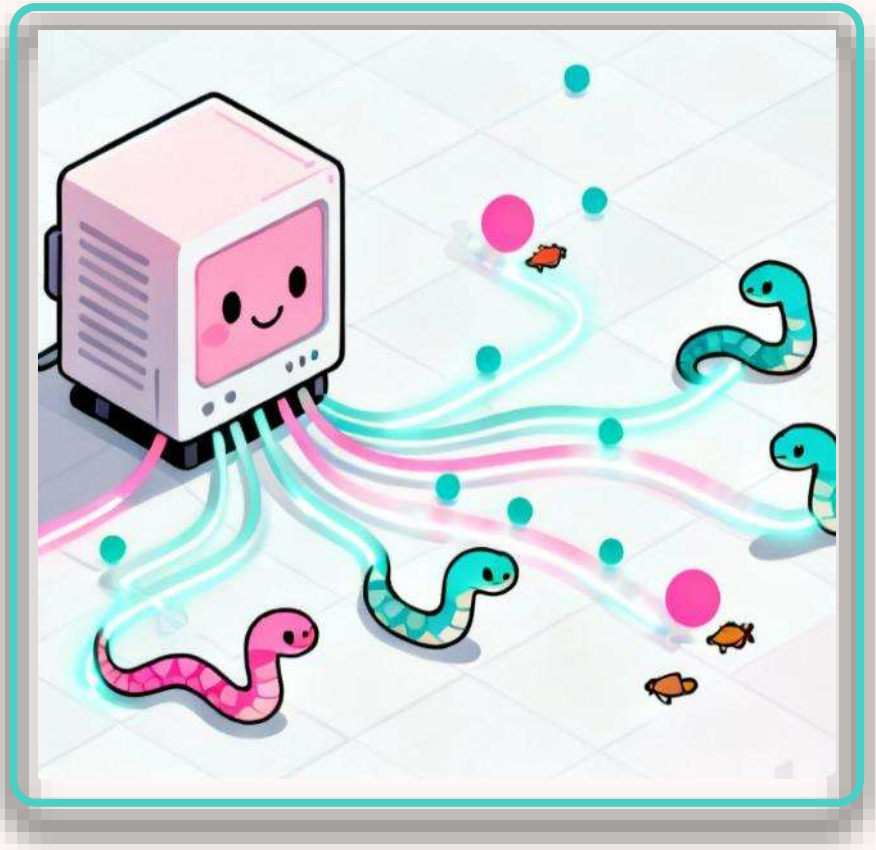
Player Repeat Game Operations

Room Management After Player Exit

Backend Part 2: Communication(\src\websocket.rs)

01

```rust
async fn handle_socket(socket: WebSocket, mut state: AppState) {
    let (room_id, snake_id) = state.game_manager.join_or_create_room();
    state.player_rooms.lock().unwrap().insert(player_id, room_id.clone());

    let (mut sender, mut receiver) = socket.split();
    let room_bc = state.get_room_broadcaster(&room_id);
    let mut room_rx = room_bc.subscribe();
}
```



Handle new player connections

Backend Part 2: Communication (\src\websocket.rs)

02

```
while let Some(Ok(msg)) = receiver.next().await {
    if let Ok(GameMessage::PlayerInput(direction)) = from_str(&text) {
        recv_state.game_manager.handle_input(&room_id_copy, snake_id, direction);
    }
}
```

Receive input from the client

03

Handle player preparation status

```
if let Ok(GameMessage::Ready) = from_str(&text) {
    let maybe_state = recv_state.game_manager.player_ready(&room_id_copy, snake_id);

    if let Some(initial_state) = maybe_state {
        let room_bc = recv_state.get_room_broadcaster(&room_id_copy);
        let msg = GameMessage::GameState(initial_state);
        let _ = room_bc.send(serde_json::to_string(&msg).unwrap());
    }
}
```

Backend Part 2: Communication (\src\websocket.rs)

04

Regular broadcast of game status

```
tokio::spawn(async move {
    let mut interval = tokio::time::interval(tokio::time::Duration::from_millis(16));
    loop {
        interval.tick().await;
        let room_states = game_state_clone.game_manager.update_all_rooms();

        for (room_id, game_state) in &room_states {
            let room_bc = game_state_clone.get_room_broadcaster(room_id);
            let msg = GameMessage::GameState(game_state.clone());
            let _ = room_bc.send(serde_json::to_string(&msg).unwrap());
        }
    }
});
```

Frontend Part 3: Backend Listener **(\src\websocket.rs)**

```rust
pub fn new(url: &str) -> Self {
    let ws = WebSocket::new(url).expect("Failed to create WebSocket connection");
    ws.set_binary_type(web_sys::BinaryType::Arraybuffer);

    let open_closure = Closure::wrap(Box::new(|| {
        console::log_1(&"WebSocket connection established successfully!".into());
    }) as Box<dyn FnMut()>);
    ws.set_onopen(Some(open_closure.as_ref().unchecked_ref()));
    open_closure.forget();

    Self { ws: Some(ws), ... }
}
```

Establish a WebSocket connection

Frontend Part 3: Backend Listener

```rust
pub fn on_game_state(mut self, callback: Callback<GameState>) -> Self {
    self.on_game_state = Some(callback);
    self
}
```

Registration status callback function

Send messages to the backend

```rust
pub fn send(&self, msg: GameMessage) {
    if let Some(ws) = &self.ws {
        let msg_str = serde_json::to_string(&msg).unwrap();
        ws.send_with_str(&msg_str).unwrap();
    }
}
```

Frontend Part 3: Backend Listener

Start message listening

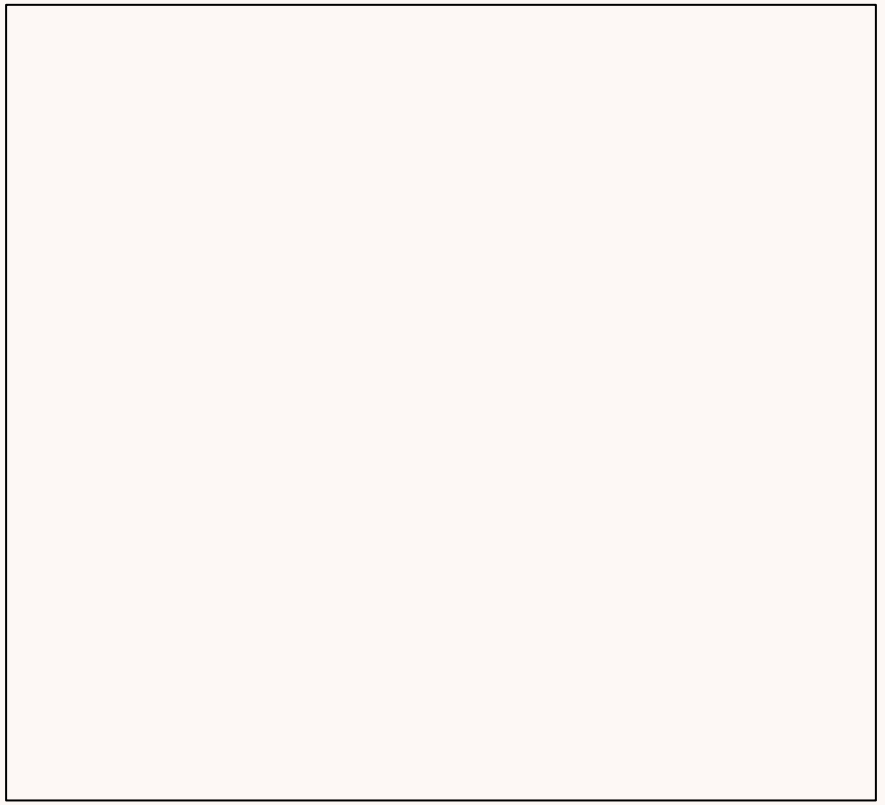
```rust
pub fn start_listening(&mut self) {
    let on_game_state = self.on_game_state.clone();

    let msg_closure = Closure::wrap(Box::new(move |e: MessageEvent| {
        let text = e.data().as_string().unwrap();
        match serde_json::from_str::<GameMessage>(&text) {
            Ok(GameMessage::GameState(state)) => {
                if let Some(cb) = on_game_state.clone() {
                    cb.emit(state);
                }
            }
            // ... 其他消息处理
        }
    }) as Box<dyn FnMut(MessageEvent)>);

    ws.set_onmessage(Some(msg_closure.as_ref().unchecked_ref()));
    msg_closure.forget();
}
```

Frontend Part 1: Map Rendering (1/5)

Map Rendering Code

```
<div class="game-map"
style=("border: 2px solid #333; "
       "width: 700px; height: 700px; "
       "position: relative; "
       "margin: 0 auto;")>
```

Square Map



Code Explanation

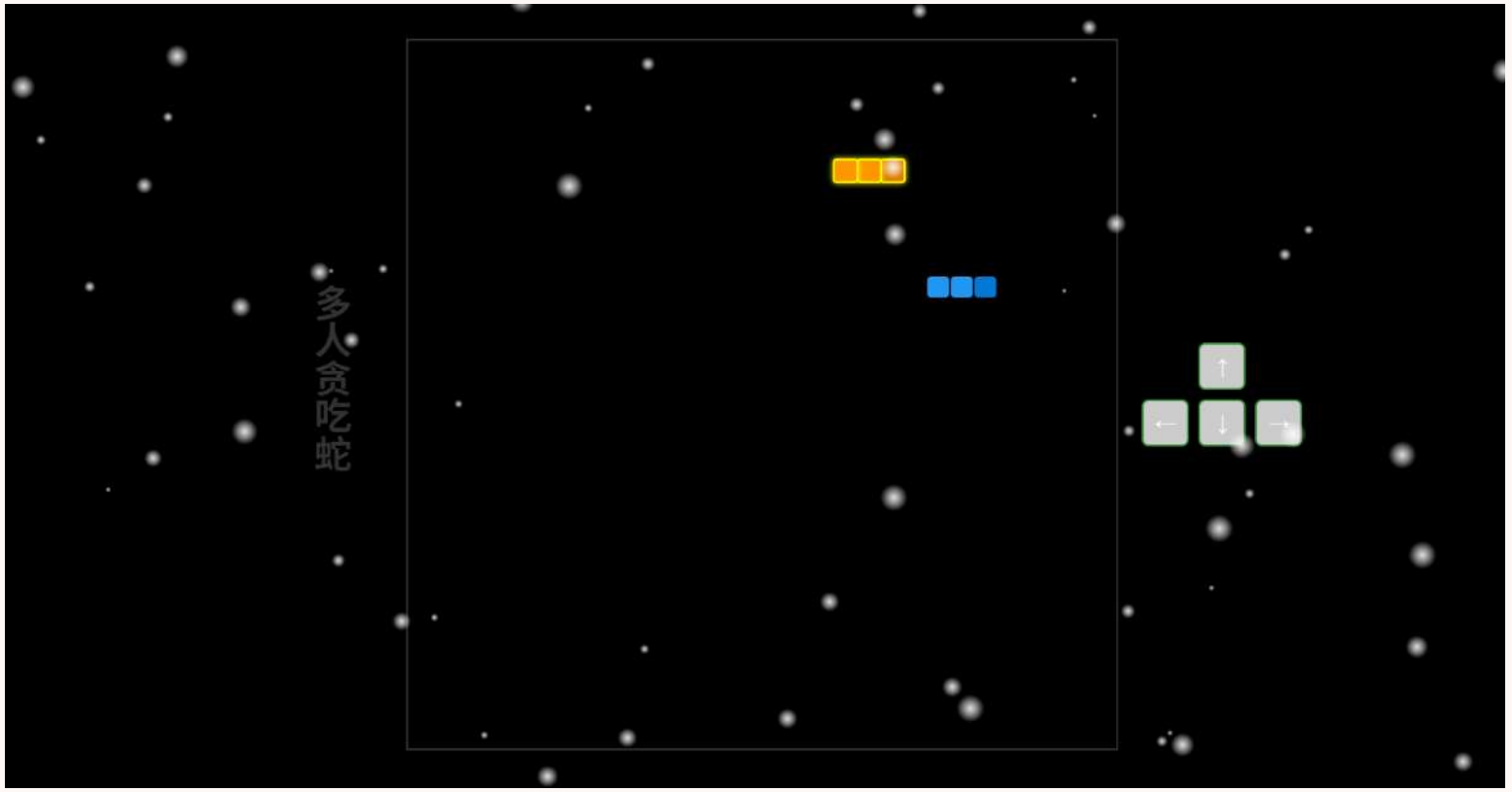
style： CSS

position: relative： Establishes relative positioning

contextmargin: 0 auto： Horizontal center alignment

Frontend Part 1: Map Rendering (1/5)

Effect display



unscramble

Add a non-intrusive snowflake falling animation to a web page without affecting the original interaction of the page, and adapt to different browsers and window sizes.

Frontend Part 1: Map Rendering (1/5)

Code Showcase

```
function snowFall(snow) {
    snow = snow || {};
    this.maxFlake = snow.maxFlake || 200;    /* 最多片数 */
    this.flakeSize = snow.flakeSize || 10;   /* 雪花形状（最大尺寸） */
    this.fallSpeed = snow.fallSpeed || 1;    /* 基础坠落速度 */
}
```

Let's first look at the entrance to the code - the snow Fall constructor, which is the "configuration center" of the entire animation

```
pub fn snow() -> &'static str {
    r##"
        /* 控制下雪 */
        function snowFall(snow) {
            /* 可配置属性 */
            snow = snow || {};
            this.maxFlake = snow.maxFlake || 200;    /* 最多片数 */
            this.flakeSize = snow.flakeSize || 10;   /* 雪花形状 */
            this.fallSpeed = snow.fallSpeed || 1;    /* 坠落速度 */
        }
```

Code interpretation

Here are two key design points:
Parameter default value handling: via snow = snow || {} to avoid errors caused by passing in undefined, and then use logic or (||) sets default values for the three core parameters - maximum number of snowflakes 200, maximum snowflake size 10px, and base fall speed of 1. This design allows users to flexibly configure and ensures the availability of default effects;
Instance property mounting: Mounting configuration parameters to this so that subsequent prototype methods (such as start) can be directly accessed, which is the classic practice of "separating data and methods" in JavaScript object-orientation.

Frontend Part 1: Map Rendering (1/5)

Code Showcase

```
// 请求动画帧兼容
requestAnimationFrame = window.requestAnimationFrame ||
    window.mozRequestAnimationFrame ||
    window.webkitRequestAnimationFrame ||
    window.msRequestAnimationFrame ||
    window.oRequestAnimationFrame ||
    function(callback) { setTimeout(callback, 1000 / 60); };

// 取消动画帧兼容
cancelAnimationFrame = window.cancelAnimationFrame ||
    window.mozCancelAnimationFrame ||
    window.webkitCancelAnimationFrame ||
    window.msCancelAnimationFrame ||
    window.oCancelAnimationFrame;
```

In front-end development, browser compatibility is an unavoidable issue, and this code has been fully compatible with the animation API:

Code interpretation

This piece of compatible code may seem simple, but it embodies the core idea of "progressive enhancements" in front-end development - prioritizing the use of native high-performance APIs, and then providing downgrade solutions for low-end browsers, taking into account experience and compatibility.

Frontend Part 1: Map Rendering (1/5)

Code Showcase

4. Start method: start prototype method and process in series

```
snowFall.prototype.start = function(){
    /* 创建画布 */
    snowCanvas.apply(this);
    /* 创建雪花形状 */
    createFlakes.apply(this);
    /* 画雪 */
    drawSnow.apply(this)
};
```

Code interpretation

The execution order of this method strictly follows the "dependency pre-dependency" principle:
Create a Canvas canvas (snowCanvas) first: you can't draw without a canvas, this is the basics;
then create a snowflake object pool (createFlakes): generate the initial state of all snowflakes to prepare for drawing;
Finally, start the draw loop (drawSnow): continuously update the position of the snowflakes and draw them to form an animation effect。

Frontend Part 1: Map Rendering (1/5)

Code Showcase

5. Canvas creation: snowCanvas function

```
function snowCanvas() {
    // 1. 创建Canvas元素
    var snowcanvas = document.createElement("canvas");
    snowcanvas.id = "snowfall";
    // 2. 设置Canvas尺寸为窗口大小
    snowcanvas.width = window.innerWidth;
    snowcanvas.height = document.body.clientHeight;
    // 3. 设置Canvas样式：绝对定位、顶层显示、不阻挡交互
    snowcanvas.setAttribute("style", "position:absolute; top: 0; left: 0; z-ind
    // 4. 添加到页面body中
    document.getElementsByTagName("body")[0].appendChild(snowcanvas);
    // 5. 挂载到实例上，供后续使用
    this.canvas = snowcanvas;
    this.ctx = snowcanvas.getContext("2d");
    // 6. 窗口大小改变时更新Canvas尺寸
    window.onresize = function() {
        snowcanvas.width = window.innerWidth;
    }
}
```

Code interpretation

Dynamically create a Canvas: Dynamically generate a canvas without relying on the original elements of the page, using document.createElement
Size setting: width with window.innerWidth (the window viewable width ensures that the canvas can cover the entire viewable area;
Style configuration: position:absolute; top:0; left:0: Fix the canvas in the top left corner of the page, covering the entire window;
Context mounting: Mount the Canvas element and the 2D drawing context to the instance, which will be used for subsequent drawings.
Window resize processing: When the window size changes, update the width of the canvas to ensure that the snowflakes always move in the viewable area and avoid "blank space".

Frontend Part 1: Map Rendering (1/5)

Code Showcase

6. Snowflake object: flake Move constructor

```
function flakeMove(canvasWidth, canvasHeight, flakeSize, fallSpeed) {
    // 初始位置：随机分布在画布内
    this.x = Math.floor(Math.random() * canvasWidth);    /* x坐标 */
    this.y = Math.floor(Math.random() * canvasHeight);   /* y坐标 */
    // 尺寸：2px到最大尺寸之间随机
    this.size = Math.random() * flakeSize + 2;           /* 形状 */
    this.maxSize = flakeSize;                            /* 最大形状 */
    // 速度：基础速度+随机偏移，让雪花下落速度有差异
    this.speed = Math.random() * 1 + fallSpeed;          /* 坠落速度 */
    this.fallSpeed = fallSpeed;                          /* 基础坠落速度 */
    this.velY = this.speed;                              /* Y方向速度（垂直下落） */
    this.velX = 0;                                       /* X方向速度（水平摆动） */
    this.stepSize = Math.random() / 30;                  /* 水平摆动步长 */
    this.step = 0                                        /* 摆动步数（用于计算余弦值） */
}
```

1. **Update method: Update the snowflake motion status**

2. **Render method: Snowflake drawing**

3. **Reset method: Snowflake reset**

Code interpretation

flake moves are the "data model" of snowflakes, each snowflake is a flake move instance that stores its own position, size, speed, and other states, and provides update and rendering methods

Frontend Part 1: Map Rendering (1/5)

Code Showcase

7. Snowflake pool creation and animation cycle

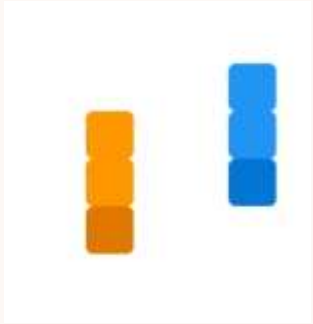
```
function createFlakes() {
    var maxFlake = this.maxFlake,
        flakes = this.flakes = [],
        canvas = this.canvas;
    // 循环生成maxFlake个雪花实例，存入flakes数组
    for (var i = 0; i < maxFlake; i++) {
        flakes.push(new flakeMove(canvas.width, canvas.heig
    }
```

Code interpretation

The core here is "object pool design": creating all the required snowflake instances at once, storing them in the this.flakes array (mounted on the snowFall instance), and only updating the state of these instances in subsequent animation loops without repeatedly creating and destroying objects - this is the key to front-end performance optimization to avoid performance loss caused by frequent DOM manipulation or object creation.

Frontend Part 1: Snake Rendering (2/5)

Snake Rendering Code



```
snake body
{ for snake.body.iter().enumerate().map(|(idx, segment)| {
    let bg_color = if idx == 0 { darken_color(color.clone()) } else { color.clone() };
    html! {
        <div style={format!(
            "position: absolute;
            width: 20px; height: 20px;
            background: {};
            left: {}px; top: {}px;",
            bg_color, segment.x * 20, segment.y * 20
        )}></div>
    }
})}
```

Code Explanation

Function Analysis:

iter().enumerate
map
darken_color function
format!():

Actual Benefits:

Distinguishing the head from the body of a snake

Frontend Part 1: Virtual Keyboard Event Handling (3/5)

Virtual Keyboard Event Handling Code

```
// Using closures and Callback to handle direction button clicks
let handle_click = |direction: Direction| {
    let on_direction = on_direction.clone();
    Callback::from(move |e: MouseEvent| {
        e.prevent_default();
        on_direction.emit(direction);
    })
};
```

Code Explanation

Function Analysis:

Callback::from():
move keyword:
prevent_default():
emit():

Actual Benefits:

event handling system,
prevents memory leaks.

Frontend Part 1: Game State Conditional Rendering (4/5)

Game State Conditional Rendering Code


```
// Using pattern matching and conditional statements to handle game state
let (snakes, foods, game_started, game_over) = match state {
    Some(s) => (s.snakes.clone(), s.foods.clone(), s.game_started, s.game_over),
    None => (vec![], vec![], false, false),
};

// Conditional rendering of game tips
if !game_started && snakes.is_empty() {
    <div class="game-tip">{"等待玩家加入..."}</div>
}
if game_over {
    <div class="game-over-tip">{"游戏结束! "}</div>
}
```

Code Explanation

Function Analysis:

match expression:

clone():

Conditional rendering:

Actual Benefits:

enhances user experience.

Frontend Part 1: Color Processing Functions (5/5)

Color Processing Functions Code

```rust
// Custom color generation and adjustment functions
fn get_snake_color(snake_id: usize) -> String {
    let colors = ["#4CAF50", "#2196F3", "#FFC107", /*...*/];
    let color_idx = snake_id % colors.len();
    colors[color_idx].to_string()
}

fn darken_color(color: String) -> String {
    let r = i32::from_str_radix(&color[1..3], 16).unwrap_or(0);
    // ... Color calculation logic
    format!("#{:02X}{:02X}{:02X}", r_dark, g_dark, b_dark)
}
```

Code Explanation

Function Analysis:

from_str_radix():
% module operation:
format!():

Actual Benefits:

game visual effects.

Frontend Part 2: Module reference and web framework generation(lib.rs)

Yew Component Declaration

THE core of Rust WebAssembly front-end development - functional components and state management

```rust
#[function_component(App)]
fn app() -> Html {
    let ws_client = use_state(|| None::<WsClient>);
    let game_state = use_state(|| None::<GameState>);
    let matching_status = use_state(|| (0, 2));
    let game_over_rankings = use_state(|| None::<Vec<(usize, u32)>>);
    let is_ready = use_state(|| false);
```

Frontend Part 2: Module reference and web framework generation(lib.rs)

```rust
use_effect_with((), move |_| {
    let mut client = WsClient::new("ws://47.100.220.180:3000/ws");

    let game_state_cb = Callback::from(move |state: GameState| {
        game_state_clone.set(Some(state));
    });
    client = client.on_game_state(game_state_cb);

    let matching_cb = Callback::from(move |(current, required): (usize, usize)| {
        matching_status_clone.set((current, required));
    });
    client = client.on_matching_status(matching_cb);
```

WebSocket connection and callback settings

The core of real-time multiplayer gaming - WebSocket communication and event driven architecture

Conditional rendering logic

Display responsive UI design and state driven rendering logic

```rust
let show_matching = {
    let game_state = game_state.clone();
    game_state.as_ref().map_or(false, |s| !s.game_started)
};

// 检查游戏是否正在进行（显示虚拟键盘的条件）
let show_virtual_keyboard = {
    let game_state = game_state.clone();
    game_state.as_ref().map_or(false, |s| s.game_started && !s.game_over)
};
```

Frontend Part 2: Module reference and web framework generation(lib.rs)

Keyboard event handling
Display user input processing and game control logic

```rust
let handle_keydown = {
    let send_message = send_message.clone();
    let game_state = game_state.clone();
    Callback::from(move |e: KeyboardEvent| {
        let state = game_state.as_ref();
        if state.map_or(true, |s| !s.game_started || s.game_over) {
            return;
        }
        match e.key().as_str() {
            "ArrowUp" => send_message(GameMessage::PlayerInput(Direction::Up)),
            "ArrowDown" => send_message(GameMessage::PlayerInput(Direction::Down)),
            "ArrowLeft" => send_message(GameMessage::PlayerInput(Direction::Left)),
            "ArrowRight" => send_message(GameMessage::PlayerInput(Direction::Right)),
            _ => {}
        }
    })
;
```

```rust
let handle_virtual_direction = {
    let send_message = send_message.clone();
    let game_state = game_state.clone();
    Callback::from(move |direction: Direction| {
        let state = game_state.as_ref();
        if state.map_or(true, |s| !s.game_started || s.game_over) {
            return;
        }
        send_message(GameMessage::PlayerInput(direction));
    })
};
```

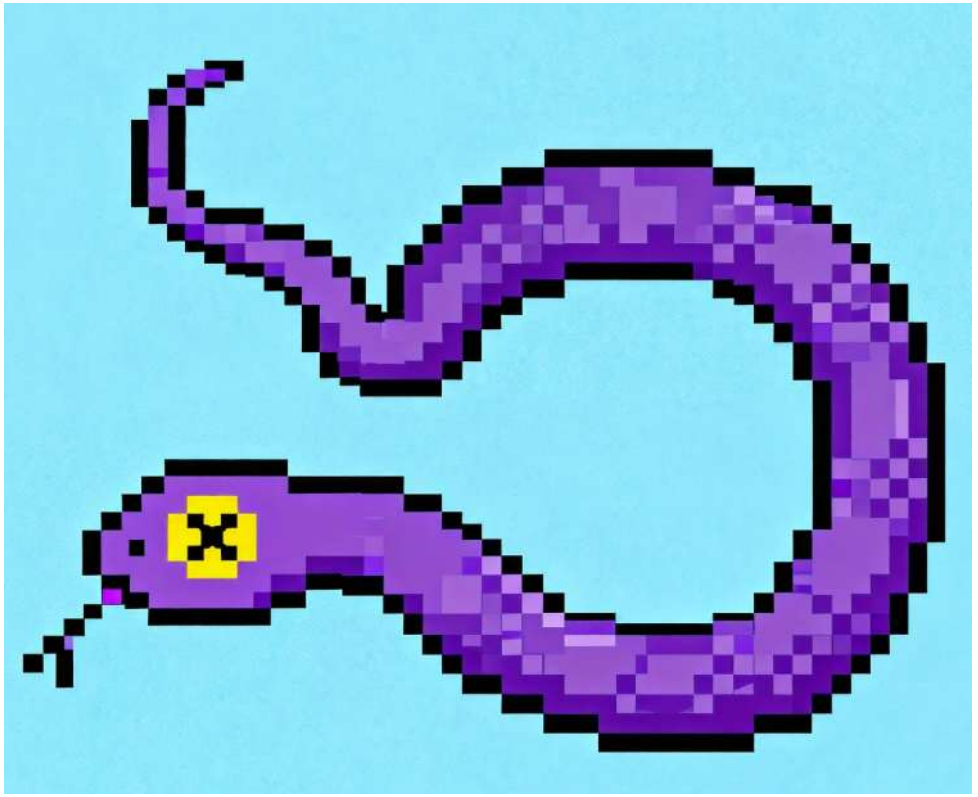
Virtual keyboard processing
Display mobile adaptation and component communication modes

Frontend Part 2: Module reference and web framework generation(lib.rs)

```
html! {
    <div class="app" onkeydown={handle_keydown} tabindex="0" style="outline: none;">
        <h1>{"多人贪吃蛇游戏"}</h1>

        if show_matching {
            <MatchingStatus
                current={matching_status.0}
                required={matching_status.1}
                on_ready={handle_ready}
                is_ready={*is_ready}
            />
        }

        <GameMap state={(*game_state).clone()} />

        // 添加虚拟键盘（仅在游戏进行中显示）
        if show_virtual_keyboard {
            <VirtualKeyboard on_direction={handle_virtual_direction} />
        }

        if let Some(rankings) = &*game_over_rankings {
            <GameOver
                rankings={rankings.clone()}
                on_restart={handle_restart}
            />
        }
        <style>
            { r#"
                .app {
                    text-align: center;
                    margin: 20px auto;
                    max-width: 500px;
                    font-family: Arial, sans-serif;
                }
                h1 {
                    color: #333;
                    margin-bottom: 30px;
                }
            "# }
            { styles() }
        </style>
    </div>
}
}
```

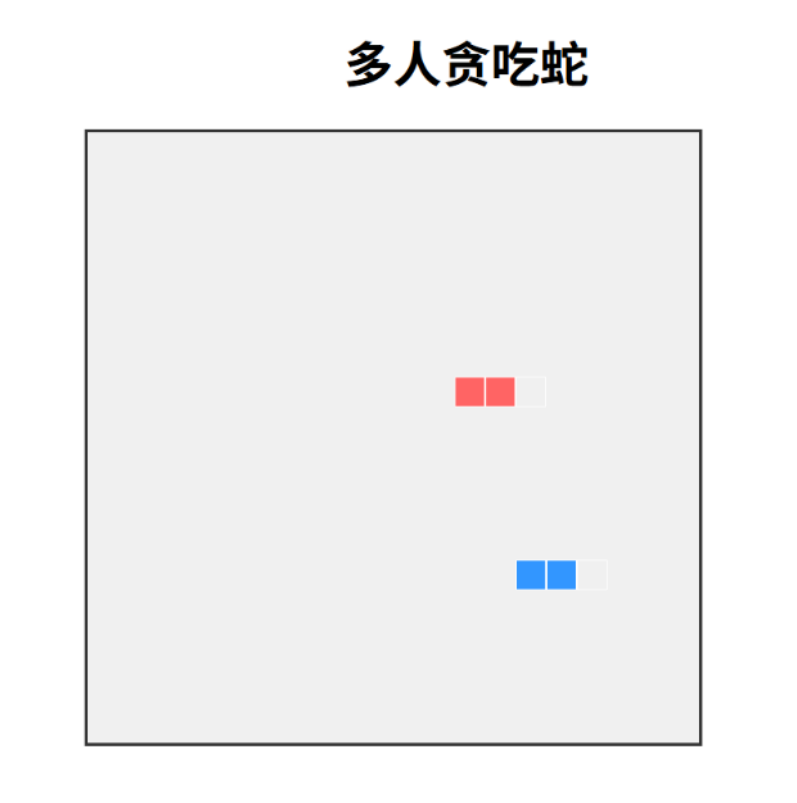
HTML rendering section

Yew's JSX like syntax and complete UI component structure



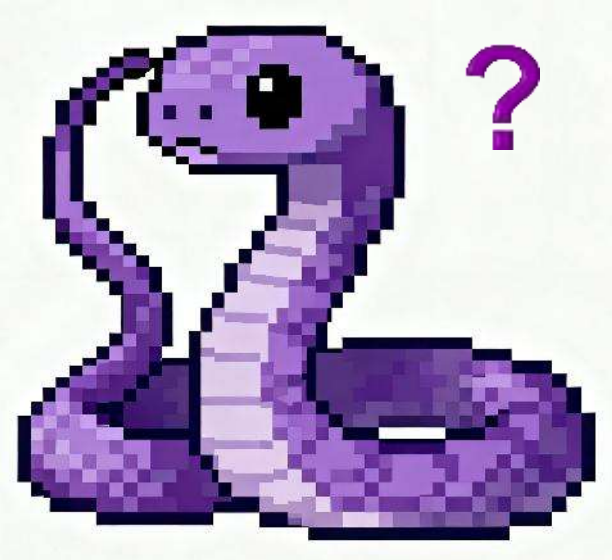
Project Difficulties

Difficulty1:
The interface is visible but unresponsive, as if the screen were frozen solid.



Solutions

1. **the backend server**
2. **the game loop**
3. **the link of WebSocket**



Project Difficulties

Difficulty2:

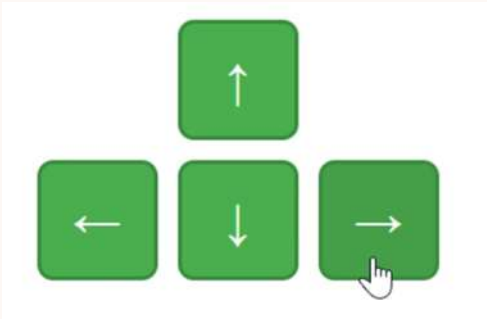
The snakes keep moving, but can't control by keyboard.

Solution:

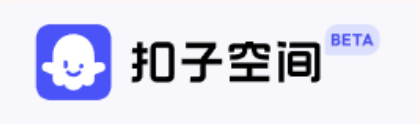
keyboard monitoring and response ×

Spending about a whole week

Li Fengdu gave an idea： Virtual Keyboard √



Tools Used



Backend

Rust, Actix-Web, PostgreSQL

Frontend

WebAssembly, Canvas API, Tailwind CSS (for cute UI)

DevOps

Docker, GitHub Actions (CI/CD), AWS EC2

Collaboration

Git, Discord, Figma (UI design)

Main division of responsibilities among project members

Li Fengdu：Configuration of the server and environment, development of the main frontend and backend programs, ensuring the server's operation and communication functionality.

Xu Nuo: implemented the code for the snow background, and also completed the map rendering section of the PowerPoint presentation.

Xu Hao: Assign tasks and monitor team members' progress; resolve the issue of the initial screen being unresponsive and create a virtual keyboard; provide a comprehensive PowerPoint template and integrate it; engage in patient discussions with team members and complete the final project.

Yue Xiaotian: Participated in debugging the code and creating the PPT lib section.

Zhang Haoyi: Debug the project and optimize the user experience. Fix the bug where the server backend receives requests but the front end does not render the game. Implement a highlighted border for the snake to emphasize the main body; design a countdown; optimize the speed of the main body.


Thank
You