JavaScript 網頁程式設計-Term Project

109704020 林思圻

壹、遊戲簡介

以交通大學光復校區地圖為靈感，設計一個吃金幣和打怪的小遊戲。在這個遊戲中，玩家可以控制角色在校區的場景中四處穿梭，尋找並收集散落在各處的金幣，同時通過發射火球來打擊怪物，以此來累積分數並提升遊戲體驗。

圖 1.交大光復校區地圖

圖 2.遊戲地圖

貳、開發工具

(一)開發環境

我們使用 VS Code 作為開發環境，利用 Tiled 軟體來設計遊戲地圖，並從 itch.io 下載各種場景元素，例如地磚、家具、牆壁以及人物角色等。

圖 3.開發工具

(二)地圖設計

從草圖開始建構，第二是牆壁與地板、第三是道路、最後是家具和樹木。

圖 4.地圖圖層順序

參、遊戲設定

(一)角色設定

圖 5.角色尺寸大小與幀數

(二)人物移動

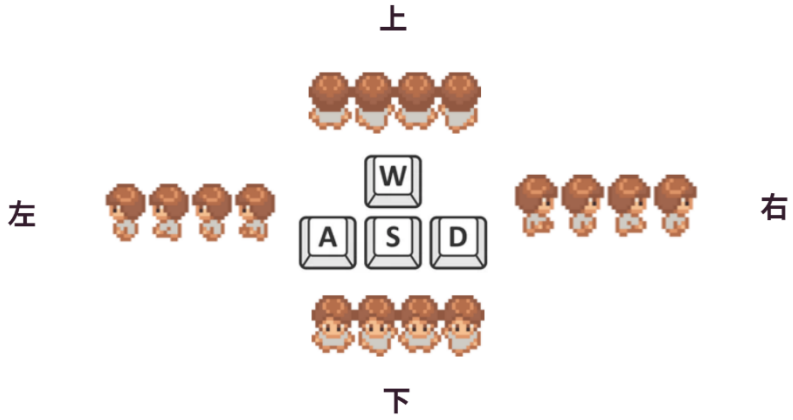

圖 6.角色控制鍵與方向

(三)火球發射

火球會隨著人物方向的轉變，改變發射路徑。由空白鍵控制，且常按可連續發射。

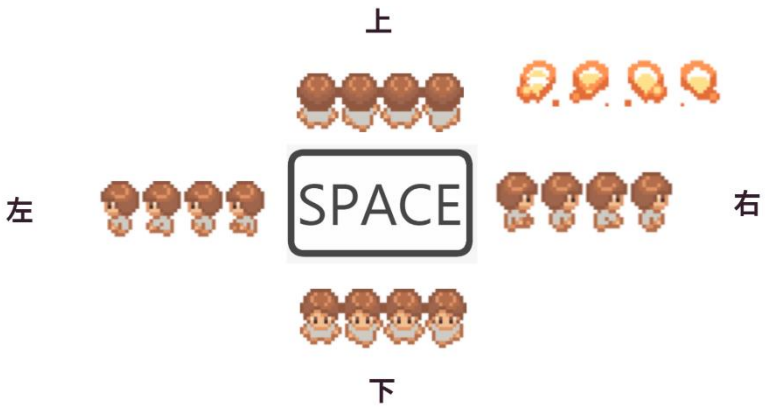

圖 7.火球控制鍵與方向

(四)怪獸移動

會呈現上下來回移動，或是左右移動，去干擾玩家的路徑。

肆、程式碼說明

(一)圖片匯入

這段程式碼用來預載遊戲所需的各種圖像資源，以便在遊戲中使用。具體來說，它創建了一些 Image 對象，並設置它們的 src 屬性以指定圖像的來源。

```
// 載入圖片
const image = new Image();
image.src = 'NYCU final game_2.png';
const playerDownImage = new Image();
playerDownImage.src = './img/playerDown.png';
const playerUpImage = new Image();
playerUpImage.src = './img/playerUp.png';
const playerLeftImage = new Image();
playerLeftImage.src = './img/playerLeft.png';
const playerRightImage = new Image();
playerRightImage.src = './img/playerRight.png';
const coinImage = new Image();
coinImage.src = './img/coin.png';
const fireballImage = new Image();
fireballImage.src = './img/fireball (1).png';
const idleImage = new Image();
idleImage.src = './img/Idle.png';
const HitImage = new Image();
HitImage.src = './img/Hit.png';
let imagesLoaded = 0;
```

圖 8.載入圖片程式碼

(二)遊戲背景+碰撞區域

Tiled 是一款地圖編輯器，它可以將地圖匯出為 JSON 格式。在匯出過程中，Tiled 會將放置了物件的格子賦予一個大於零的數字，並將這些格子的資料存儲在一個陣列中。這樣，我們就可以通過碰撞圖層將地圖轉換為 JSON 格式，並記錄下包含物件的格子。在本次設計的程式碼中，我將這個陣列命名為 collisions。如圖所示，0 代表沒有放置物件，其他數字代表有放置物件，這些有放置物件的區域代表牆壁。

而在這張尺寸為 1600 x 1600 的圖片中，被 16 x 16 的單位方塊切成了 100 x 100 的網格。下圖的陣列可以進一步整理成每 100 個數字為一列的子陣列，總共有 100 列。因此，接下來可以透過有數字的元素是在哪一列 (j) 的哪一個 (i)，對應到整張圖片的位置，從而定位出牆壁的具體位置。

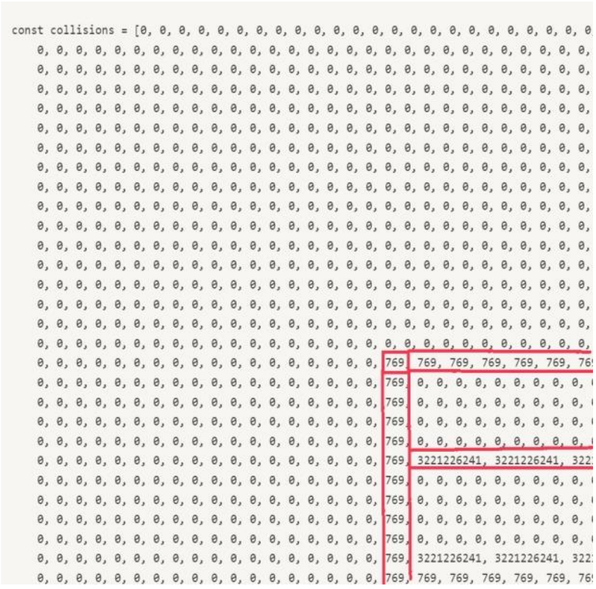

圖 9. 碰撞圖層陣列

圖 10 . 碰撞區域與網格

```
    // 建立碰撞地圖
    const collisionsMap = [];

    // 使用 for 循環來遍歷 collisions 數組，每次增量為 100
    for (let i = 0; i < collisions.length; i += 100) {
        // 使用 slice 方法從 collisions 數組中提取一段長度為 100 的子數組(collisions可以看成
        //100 X 100的矩陣，我希望切成100列為一組)
        // 並將這個子數組添加到 collisionsMap 陣列中
        collisionsMap.push(collisions.slice(i, 100 + i));
    }
class Boundary {
        static height = 16;
        static width = 16;


        constructor({ position }) {

            this.position = position;
            this.width = Boundary.width;
            this.height = Boundary.height;
        }

        // 定義一個 draw 方法，用於繪製邊界
        draw() {
            // 設置繪圖上下文的填充樣式為透明的紅色
            context.fillStyle = "rgba(255,0,0,0.0)";
            // 繪製一個填充矩形，位置和大小由實例的 position、width 和 height 屬性決定
            context.fillRect(this.position.x, this.position.y, this.width, this.height);
        }
    }
```

```
const boundaries = [];
const offset = {
    x: -120,
    y: -200
};

collisionsMap.forEach((row, i) => { //每列
    //每行
    row.forEach((symbol, j) => {
        if (symbol != 0)
            boundaries.push(new Boundary({
                position: {
                    //在第j行，第i列的格子填滿紅色
                    x: j * Boundary.width + offset.x,
                    y: i * Boundary.height + offset.y
                }
            }));
    });
});
```

圖 11 . 碰撞區域繪製程式碼

(三)創建角色

這段程式碼定義了一個名為 Sprite 的 Class，用於處理遊戲中的角色或物件的顯示和動畫。這個 Class 主要包含物件的位置、圖像、幀數、方向和速度等屬性，並提供了一個函數 draw()來繪製物件和更新圖片幀數。

```
class Sprite {
    constructor({ position, image, frames = { max: 1, hold: 10, long: 48 }, direction
        this.position = position;
        this.image = image;
        this.frames = { ...frames, val: 0, elapsed: 0 };
        this.width = image.width / this.frames.max;
        this.height = image.height;
        this.moving = false;
        this.direction = direction;
        this.velocity = velocity || { x: 0, y: 0 };
    }
    // 繪製
    draw() {
        context.drawImage(
            this.image,
            this.frames.val * this.frames.long,
            0,
            this.image.width / this.frames.max,
            this.image.height,
            this.position.x,
            this.position.y,
            this.image.width / this.frames.max,
            this.image.height
        );

        // 更新Frames
        if (this.moving) {
            if (this.frames.max > 1) {
                this.frames.elapsed++;
                if (this.frames.elapsed % this.frames.hold === 0) {
                    this.frames.val = (this.frames.val + 1) % this.frames.max;
```

圖 12. 角色 Class

(四)人物移動(利用移動背景來形成走路的效果)

像是下面程式碼是在處理按下 'a' 鍵時的動作。如果玩家按下 'a' 鍵且上一個按鍵是 'a'，則設置角色為移動狀態，並將角色的圖像方向設置為向左。然後，它遍歷所有的邊界物件，檢查玩家是否與邊界物件發生碰撞。如果碰撞了，則停止移動；否則，將所有可移動的物件向右移動 3 像素，同時更新 walk_x 變量以跟踪角色的水平移動。

```
// 創建玩家
const player = new Sprite({
    position: {
        x: canvas.width / 2,
        y: canvas.height / 2,
    },
    image: playerDownImage,
    frames: { max: 4, hold: 10, long: 36 },
    direction: {
        up: playerUpImage,
        left: playerLeftImage,
        down: playerDownImage,
        right: playerRightImage
    }
});
```

```
else if (keys.a.pressed && lastKey === 'a') {
  player.moving = true;
  player.image = player.direction.left;
  for (const boundary of boundaries) {
      if (recCollision({
          rectangle1: player,
          rectangle2: { ...boundary, position: { x: boundary.position.x + 3, y: boundary.posit
      })) {
          moving = false;
          break;
      }
  }
  if (moving) {
      movables.forEach(movable => movable.position.x += 3);
      walk_x += 3;
```

圖 13 . 人物移動 a 鍵範例

(五)怪獸動畫

程式碼負責更新和繪製怪物。它遍歷 monsters 陣列中的每個怪物，根據怪物的索引來決定它的移動模式：

1. **偶數索引的怪物**：這些怪物會垂直移動。
    - 每隔一秒 (setInterval 設為 1000 毫秒)，改變垂直移動的方向。
    - 若 direction 為 1，則怪物的 y 坐標增加 i 的值；若 direction 為 -1，則 y 坐標減少 2。
    - 每次方向改變後，i 的值從 1 變為 2。
2. **奇數索引的怪物**：這些怪物會水平移動。
    - 每隔一秒 (setInterval 設為 1000 毫秒)，改變水平移動的方向。
    - 若 direction 為 1，則怪物的 x 坐標增加 i 的值；若 direction 為 -1，則 x 坐標減少 2。
    - 每次方向改變後，i 的值從 1 變為 2。

```
const monsters = []; // 儲存怪物的陣列

// 創建怪物
for (let i = 0; i < 5; i++) {
    const monster = new Sprite({
        position: getMonsPosition(i),
        frames: { max: 5, hold: 10, long: 32 },
        image: idleImage,
        velocity: { x: 0, y: 1 }
    });
    monsters.push(monster);
}
```

```
// 更新和繪製怪物
monsters.forEach((monster, index) => {
    if (index % 2 === 0) { // 偶數index的怪物
        let direction = -1;
        let i = 1;

        setInterval(() => {
            // 改變方向
            direction *= -1;
            if (direction === 1) {
                monster.position.y += i;
            }
            if (direction === -1) {
                monster.position.y -= 2;
            }
            i = 2;
        }, 1000); // 每隔一秒改變一次方向
```

```
    } else { // 奇數index的怪物
        let direction = -1;
        let i = 1;

        setInterval(() => {
            // 改變方向
            direction *= -1;
            if (direction === 1) {
                monster.position.x += i;
            }
            if (direction === -1) {
                monster.position.x -= 2;
```

圖 14 . 怪獸水平與垂直移動

根據玩家的最後移動方向 (direction)，設定火球的速度 (velocity)，然後創建一個新的火球對象並將其加入 fireballs 陣列中。

1. **設置速度**：根據方向 (direction) 設置火球的速度：
    - 'w' (上)：速度為 { x: 0, y: -5 }，即火球向上移動。
    - 'a' (左)：速度為 { x: -5, y: 0 }，即火球向左移動。
    - 's' (下)：速度為 { x: 0, y: 5 }，即火球向下移動。
    - 'd' (右)：速度為 { x: 5, y: 0 }，即火球向右移動。
2. **創建火球對象**：利用 Sprite 類創建一個新的火球對象，並設置以下屬性：
    - position：設為玩家當前的位置 。
    - image：火球的圖片。
    - frames：設置火球動畫的幀數、持續時間和每幀的寬度。
    - velocity：設為根據方向計算出的速度。

3. **添加到陣列**：將新創建的火球對象加入 fireballs 陣列中，以便後續更新和繪製。當玩家按下射擊按鈕（例如空格鍵）時，會根據玩家的最後移動方向發射一個火球。

```
function shootFireball(direction) {
    let velocity;
    switch (direction) {
        case 'w':
            velocity = { x: 0, y: -5 };
            break;
        case 'a':
            velocity = { x: -5, y: 0 };
            break;
        case 's':
            velocity = { x: 0, y: 5 };
            break;
        case 'd':
            velocity = { x: 5, y: 0 };
            break;
    }
```

```
    const fireball = new Sprite({
        position: {
            x: player.position.x,
            y: player.position.y,
        },
        image: fireballImage,
        frames: { max: 4, hold: 12, long: 43 },
        velocity: velocity
    });
    fireballs.push(fireball);
}
```

圖 15 . 火球動畫

(六)角色間的交互碰撞偵測

檢查硬幣碰撞並更新分數 (checkCoinCollisions)：

- 這個函式遍歷所有硬幣 (coins)。
- 使用 coin_recCollision 函式檢查玩家 (player) 是否與每個硬幣發生碰撞。
- 如果發生碰撞，將分數增加 1 (score += 1)，並更新頁面上的分數顯示。
- 移除發生碰撞的硬幣（從 coins 陣列中刪除）。

檢查火球和怪物碰撞 (detectMonsterCollision)：

- 這個函式遍歷所有火球 (fireballs) 和所有怪物 (monsters)。
- 使用 mon_recCollision 函式檢查每個火球是否與每個怪物發生碰撞。
- 如果發生碰撞，將分數增加 1，並更新頁面上的分數顯示。
- 將發生碰撞的怪物的圖像更改為被擊中的圖像 (HitImage)，並設置新的動畫幀數。
- 設置一個計時器，在 1 秒後將該怪物從 monsters 陣列中刪除。

檢查玩家和怪物碰撞 (MonsCollisions)：

- 這個函式遍歷所有怪物 (monsters)。

- 使用 coin_recCollision 函式檢查玩家是否與每個怪物發生碰撞。
- 如果發生碰撞，將分數減少 1，並更新頁面上的分數顯示。

```
// 檢查硬幣碰撞並更新分數
function checkCoinCollisions() {
    coins.forEach((coin, index) => {
        if (coin_recCollision({ rectangle1: player, rectangle2: coin })) {
            score += 1;
            document.getElementById('coin').innerText = `Score: ${score}`;
            coins.splice(index, 1);
        }
    });
}

// 檢查火球和怪物碰撞
function detectMonsterCollision() {
    fireballs.forEach((fireball, fireballIndex) => {
        monsters.forEach((monster, monsterIndex) => {
            if (mon_recCollision({ rectangle1: fireball, rectangle2: monster })) {
                score += 1;
                document.getElementById('coin').innerText = `Score: ${score}`;
                monsters[monsterIndex].setImage(HitImage);
                monsters[monsterIndex].setFrames({ max: 7, hold: 5, long: 32 });
                setTimeout(() => {
                    delete monsters[monsterIndex];
                }, 1000);
            }
        });
    });
}
```

```
function recCollision({ rectangle1, rectangle2 }) {
    return (
        rectangle1.position.x + rectangle1.width + 15 >= rectangle2.position.x &&
        rectangle1.position.x + 5 <= rectangle2.position.x + rectangle2.width &&
        rectangle1.position.y + 5 <= rectangle2.position.y + rectangle2.height &&
        rectangle1.position.y + rectangle1.height + 35 >= rectangle2.position.y
    );
}
```

```
// 檢查玩家和怪物碰撞
function MonsCollisions() {
    monsters.forEach((monster, index) => {
        if (coin_recCollision({ rectangle1: player, rectangle2: monster })) {
            score -= 1;
            document.getElementById('coin').innerText = `Score: ${score}`;
        }
    });
}
```

圖 16 . 碰撞檢測

(七)倒數計時、結束頁面

1. **計時器 (setInterval)**：

   - 設置一個計時器，每秒減少 time_count 的值，並更新頁面上的顯示。
   - 當 time_count 小於等於 0 時，停止計時器並調用 clearInterval。

2. **結束遊戲 (endGame 函式)**：

   - 將 gameOver 設為 true，表示遊戲結束。
   - 顯示一個模態彈出窗口，內容包括 "Game Over" 字樣和最終得分。
   - 使模態窗口可見（通過設定 modal.style.display 為 'flex'）。

```javascript
let time_count = 30; // 初始時間設定為30秒

const intervalId = setInterval(() => {
    time_count -= 1;
    document.getElementById('time').innerText = `${time_count}`;

    if (time_count <= 0) {
        clearInterval(intervalId); // 停止計時器
    }
}, 1000);

// 結束遊戲
function endGame() {
    gameOver = true;
    const modal = document.getElementById('myModal');
    const finalScore = document.getElementById('finalScore');
    finalScore.innerText = `Final Score: ${score}`;
    modal.style.display = 'flex';
}
```

```html
<body>
    <canvas id="myCanvas" width="1400" height="576"></canvas>
    <div id="coin">Score: 0</div>
    <div id="time">30</div>
    <!-- The Modal -->
    <div id="myModal" class="modal">
        <div class="modal-content">
            <span class="close">&times;</span>
            <div id="gameOver">Game Over</div>
            <div id="finalScore">Final Score: 0</div>
            <button id="restartButton" onclick="reStart()">Restart</button>
        </div>
    </div>

</body>
```

圖 17 . 計時器與 Html

伍、心得感想

這次的小遊戲實作，把重點都放在 javascript 上。反覆做了許多次修改。最困難的就是不同角色之間的碰撞互動關係，要去做細微的調整。覺得這次很樂在其中，並且也做出了理想的效果!