Javascipt 網頁設計期末專題

Pizza RPG 書面報告(二版)

姓名：黃綵誼　學號：111613025

目錄

一、 系統功能/摘要:

1. 摘要

一個以 Pizza 廚房為背景的 RPG 遊戲，使用者可以跟場景內的角色進行互動，並進行 Pizza 對戰！

2. 功能

使用者能在場景內自由走動，跟不同角色互動，並走到特定格子切換場景，對戰時能選擇不同招式。

3. 系統架構

於客廳尋找角色進行互動獲得提示➔前往廚房➔
找到敵人➔進入對決場景➔用 pizza 打架!

二、 系統開發平台

1. 利用 VScode 搭配其擴充套件 Live server 開發

三、 執行環境

1. Chrome 為佳 (Edge 會有排版跑掉的問題)

四、 程式說明 (普通場景)

1. overwolrd_test.js

整個遊戲場景的更新，人物走路、與 npc 互動等

i. startGameLoop()

清空畫面並設定要位於畫面中間的人物，更新各角色的方向，畫地圖及角色，並利用 requestAnimationFrame 每隔一段時間重複呼叫，達成不斷更新畫面的效果

```
startGameLoop() {
    const step = () => {
        //clean obj in the canvas
        this.ctx.clearRect(0, 0, this.canvas.width, this.canvas.height);

        //establish the camera person
        const cameraPerson = this.map.gameObjects.hero;

        //update all objs
        Object.values(this.map.gameObjects).forEach(object => {
            object.update({
                arrow: this.directionInput.direction,
                map: this.map,
            })
        })

        //draw lower layer
        this.map.drawLowerImage(this.ctx, cameraPerson);

        //draw game objs
        Object.values(this.map.gameObjects).sort((a, b) => {
            return a.y - b.y;       //if return >0: b->a, <0: a->b
        }).forEach(object => {
            //object.x += 1;
            object.sprite.draw(this.ctx, cameraPerson);
        })

        //draw upper layer
        this.map.drawUpperImage(this.ctx, cameraPerson);

        requestAnimationFrame(() => {
            step();
        })
    }
    step();
}
```

2. map_test.js

宣告 map 的 class 並建立判斷角色與地圖關係的 method，也存放地圖參數

i. window.OverworldMaps

存放地圖資訊，地圖圖片、角色、NPC 固定動作等

```
window.OverworldMaps = {
    DemoRoom: {
        lowerSrc: "./img/maps/DemoLower.png",
        upperSrc: "./img/maps/DemoUpper.png",
        gameObjects: {
            hero: new Person({…
            }),
            npc1: new Person({
                x: utils.withGrid(6),
                y: utils.withGrid(8),
                src: "./img/characters/people/npc1.png",
                behaviorLoop: [
                    { type: "stand", direction: "left", time: 1000 },
                    { type: "stand", direction: "up", time: 500 },
                    { type: "stand", direction: "right", time: 300 },
                    { type: "stand", direction: "up", time: 600 },
```

也包含地圖牆壁、及觸發事件的區域

```
walls: {…
},
cutsceneSpaces: {
    [utils.asGridCoords(7, 4)]: [
        {
            events: [
                { who: "npc2", type: "walk", direction: "left" },
                { who: "npc2", type: "stand", direction: "up", time: 300 },
                { type: "textMessage", text: "You can't be in there!" },
                { who: "npc2", type: "walk", direction: "right" },
                { who: "hero", type: "walk", direction: "down" },
                { who: "hero", type: "walk", direction: "left" },
            ]
        }
    ],
    [utils.asGridCoords(5, 10)]: [
        {
            events: [
                { type: "changeMap", map: "Kitchen" },
            ]
        }
    ],
}
```

ii. isSpaceTaken()

傳入目前位置及方向，判斷前方是否能走，於每次角色要移動時執行

```
isSpaceTaken(currentX, currentY, direction) {
    const { x, y } = utils.nextPosition(currentX, currentY, direction);
    return this.walls[`${x},${y}`] || false;
}
```

iii. startCutscene()

傳入地圖中 cutscene(角色跑固定動作)，要做的 event，利用 async function 及 await 等待 event 執行完後，讓角色重新繼續動作。

```
async startCutscene(events) {
    this.isCutscenePlaying = true;

    //start a loop for async events
    //await each one
    for (let i = 0; i < events.length; i++) {
        const eventHandler = new OverworldEvent({
            event: events[i],
            map: this,
        })
        await eventHandler.init();
    }

    this.isCutscenePlaying = false;

    //reset npc to do behavior
    Object.values(this.gameObjects).forEach(obj => {
        obj.doBehaviorEvent(this);
    })
}
```

iv. checkForAction/FootstepCutscene()

檢查角色是否有跟 NPC 互動，或是踩到有事件的格子

```
checkForActionCutscene() {
    const hero = this.gameObjects["hero"];
    const nextCoords = utils.nextPosition(hero.x, hero.y, hero.direction);

    const match = Object.values(this.gameObjects).find(obj => {
        return `${obj.x},${obj.y}` === `${nextCoords.x},${nextCoords.y}`
    });
    console.log(match);
    if (match && match.talking.length && !this.isCutscenePlaying) {
        this.startCutscene(match.talking[0].events);
    }
}

checkForFootstepCutscene() {
    const hero = this.gameObjects["hero"];
    const match = this.cutsceneSpaces[`${hero.x},${hero.y}`];
    //console.log(match);

    if (match && !this.isCutscenePlaying) {
        this.startCutscene(match[0].events);
    }
}
```

3. obj_test.js

GameObject class，並有 mount(於地圖上建立此 object)及 doBehavior(執行動作)的 method

i. mount()

建立物件時 addWall，確保物件不會被穿透

```
mount(map){
    console.log("mount");
    this.isMounted = true;
    map.addWall(this.x, this.y);

    //if we have a behavior, kick o
    setTimeout(()=>{
        this.doBehaviorEvent(map);
    },10)
}
```

ii. doBehaviorEvent()

讀取 NPC 的固定動作，迴圈執行。

```
async doBehaviorEvent(map){
    //sth more important
    if (map.isCutscenePlaying || this.behaviorLoop.length === 0 || this.isStanding){
        return;
    }

    //console.log("doBehaviorEvent");
    //set up event with relevant info
    let eventConfig = this.behaviorLoop[this.behaviorLoopIndex];
    eventConfig.who = this.id;

    //create an event instance out of our next event config
    const eventHandler = new OverworldEvent({map, event: eventConfig});
    await eventHandler.init();

    //setting the next event to fire
    this.behaviorLoopIndex += 1;
    if (this.behaviorLoopIndex === this.behaviorLoop.length){
        this.behaviorLoopIndex = 0;
    }

    //do it again
    this.doBehaviorEvent(map);
}
```

4. person_test.js

為 GameObject 的延伸，但增加一些人物才會有的 method

i. startBehavior()

利用上述提到的 isSpaceTaken()，判斷是否能往前走，不行的話，隔一段時間後重新執行一次，往前走後 moveWall (移動角色所建立的牆壁)

```
startBehavior(state, behavior) {
    //set character direction to whatever behavior has
    this.direction = behavior.direction;

    if (behavior.type === "walk") {
        //stop if space is not free
        if (state.map.isSpaceTaken(this.x, this.y, this.direction)) {
            //if (behavior.retry)
            behavior.retry && setTimeout(()=>{
                this.startBehavior(state, behavior)
            },10);
            return;
        }

        //ready to walk
        state.map.moveWall(this.x, this.y, this.direction);
        this.movingProgressRemaining = 16;
        this.updateSprite();
    }
```

5. sprite_test.js

依照人物狀態(面向、走動)，設定人物的要擷取照片的哪部分

```
//configuring Animation and initial state
this.animations = config.animations || {
    "idle-down": [[0, 0]],
    "idle-right": [[0, 1]],
    "idle-up": [[0, 2]],
    "idle-left": [[0, 3]],
    "walk-down": [[1, 0], [0, 0], [3, 0], [0, 0]],
    "walk-right": [[1, 1], [0, 1], [3, 1], [0, 1]],
    "walk-up": [[1, 2], [0, 2], [3, 2], [0, 2]],
    "walk-left": [[1, 3], [0, 3], [3, 3], [0, 3]],
}
```

i. updateAnimationProgress()

於每一次重畫畫面時執行，會更新 AnimationFrame 的 index

```
get frame() {
    return this.animations[this.currentAnimation][this.currentAnimationFrame];
}

updateAnimationProgress() {
    //Downtick frameprogress
    if (this.animationFrameProgress > 0) {
        this.animationFrameProgress -= 1;
        return;
    }
    //reset animation progree
    this.animationFrameProgress = this.animationFrameLimit;
    this.currentAnimationFrame += 1;

    if (this.frame === undefined) {
        this.currentAnimationFrame = 0;
    }
}
```

搭配 getframe() 可於畫角色時達到裁切特定方格的效果，製早走路時的動畫。

```
const [frameX, frameY] = this.frame;

if (this.isLoaded) {
    ctx.drawImage(this.image,
        frameX * 32, frameY * 32,
        32, 32,
        x, y,
        32, 32
    )
```

6. directionInput_test.js

使用者按鍵盤方向鍵時，return 該方向，考慮可能同時按多個方向再逐一放開的情況，建立 list 紀錄「正在被按」的鍵盤，按下鍵盤時會加將該方向加入 list，放開時會 pop 掉該方向，只 return 最新按的方向。

```
get direction(){
    return this.heldDirections[0];
}

init(){
    document.addEventListener("keydown", e =>{
        const dir = this.map[e.code];
        if(dir && this.heldDirections.indexOf(dir) === -1){
            this.heldDirections.unshift(dir);
        }

    });
    document.addEventListener("keyup",e=>{
        const dir = this.map[e.code];
        const index = this.heldDirections.indexOf(dir);
        if (index > -1){
            this.heldDirections.splice(index,1);
        }
    })
}
```

此結果會傳入 OverWorld，於每一次重畫畫面時更新每一個 object 的方向。

```
//update all objs
Object.values(this.map.gameObjects).forEach(object => {
    object.update({
        arrow: this.directionInput.direction,
        map: this.map,
    })
})
```

7. text_test.js

TextMessage class

i. createElement()

建立訊息

```
createElement() {
    //create the ele
    this.element = document.createElement("div");
    this.element.classList.add("TextMessage");

    //console.log(this.text);
    this.element.innerHTML = (`
        <p class="TextMessage_p"></p>
        <button class = "TextMessage_button">Next</button>
    `);
```

並於使用者點選 Next 或 enter 時結束該則訊息

```
this.element.querySelector("button").addEventListener("click", () => {
    this.done();
});

this.actionListener = new KeyPressListener("Enter", () => {
    console.log("Enter when text message");
    this.done();
})
```

8. revealText.js

製造打字機動畫

i. revealOneCharacter()

每隔一小段時間幫一個字元新增 revealed 的 class 再顯示(利用 css)。

```
revealOneCharacter(list){
    const next = list.splice(0,1)[0];
    next.span.classList.add("revealed");

    if(list.length > 0){
        this.timeout = setTimeout(()=>{
            //console.log(next);
            this.revealOneCharacter(list);
        }, next.delayAfter)
    }else{
        this.isDone = true;
    }
}
```

```
.TextMessage span{
    opacity: 0;
}

.TextMessage span.revealed{
    opacity: 1;
}
```

ii. wrapToDone()

於使用者按 enter 或 next 時執行，直接顯示完所有文字

```
warpToDone(){
    clearTimeout(this.timeout);
    this.isDone = true;
    this.element.querySelectorAll("span").forEach(s =>{
        s.classList.add("revealed");
    })
}
```

9. event_test.js

存放所有 event 要執行的事

```
class OverworldEvent {
    constructor({ map, event }) {…
    }

    stand(resolve) {…
    }

    walk(resolve) {…
    }

    textMessage(resolve) {…
    }

    changeMap(resolve) {…
    }

    battle(resolve) {…
    }

    init() {
        //console.log("OverworldEvent init");
        return new Promise(resolve => {
            this[this.event.type](resolve);
        })
    }
}
```

event 會以這個形式傳入，當有 event 時就會依照 event 的 type 去做相對應的事，並等待 resolve()，執行完後才會 return

```
{ who: "npc2", type: "stand", direction: "up", time: 300 },
{ type: "textMessage", text: "You can't be in there!" },
{ who: "npc2", type: "walk", direction: "right" },
```

例如：當 event.type === textMessage 時會創立 new TextMessage，並依第 7.的 class 去建立 message 並添加相關的 eventListener

```
textMessage(resolve) {
    if (this.event.faceHero) {
        const obj = this.map.gameObjects[this.event.faceHero];
        //get the direction opposite of the hero's direction to face to hero
        obj.direction = utils.oppositeDirection(this.map.gameObjects["hero"].direction);
    }
    const message = new TextMessage({
        text: this.event.text,
        onComplete: () => resolve(),
    })
    message.init(document.querySelector(".game-container"));
}
```

當使用者按 enter 時會執行 TextMessage .done()，其中的 onComplete()就會是 resolve()，接收到 resolve()之後，這個 event 就被執行完了，才會再繼續其他的事情。

```
done() {
    if (this.reavealingText.isDone) {
        this.element.remove();
        this.actionListener.unbind();
        this.onComplete();
    }else{
        this.reavealingText.warpToDone();
    }
}
```

10. utils_test.js

存放這個遊戲會運用到的小功能

i. 16 個 px 定為一個單位

```
withGrid(n){
    return n*16;
},
asGridCoords(x,y){
    return `${x*16},${y*16}`;
},
```

ii. 根據目前方向及位置 return 下一個位置/相反方向

```
nextPosition(initialX, initialY,direction){
    let x = initialX;
    let y = initialY;
    const size = 16;
    if (direction === "left"){
        x-=size;
    }else if(direction === "right"){···
    }
    else if(direction === "up"){···
    }
    else if(direction === "down"){···
    }
    return {x,y};
},
```

```
oppositeDirection(direction){
    if(direction === "left"){
        return "right";
    }
    if(direction === "right"){
    }
    if(direction === "up"){···
    }
    if(direction === "down"){···
    }
},
```

iii. 客製化 eventListener 及 dalay 功能

```
emitEvent(name, detail){
    const event = new CustomEvent(name,{
        detail
    });
    document.dispatchEvent(event);
},
```

```
wait(ms){
    return new Promise(resolve=>{
        setTimeout(()=>{
            resolve();
        }, ms)
    });
},
```

11. sceneTransition.js

轉場時動畫，添加白色遮罩，
利用 sceneTransition.css 做 fade-in fade-out 效果

```
class SceneTransition {
    constructor() {
        this.element = null;
    }

    createElement() {
        this.element = document.createElement("div");
        this.element.classList.add("SceneTransition");
    }

    fadeOut(){
        this.element.classList.add("fade-out");
        this.element.addEventListener("onanimationend", () => {
            this.element.remove();
        }, { once: true }) //immediateley unbind when done
    }
```

```
.SceneTransition.fade-out{
    position: absolute;
    left: 0;
    top: 0;
    right:0;
    bottom: 0;
    background: #fff;
    opacity: 0;
    animation: sceneTransition-fade-out 1.2s forwards;
}

@keyframes sceneTransition-fade-in {
    from {opacity: 0;}
    to {opacity: 1;}
}

@keyframes sceneTransition-fade-out {
    from {opacity: 1;}
    to {opacity: 0;}
}
```

五、 程式說明 (對戰場景)

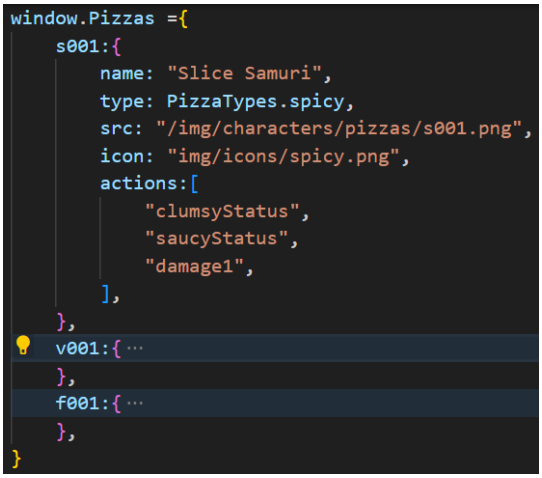
1. actions.js

存放攻擊招式們，success 為攻擊成功需觸發的 event，會利用 battle_event.js 執行

```
window.Actions = {
    damage1: {
        name: "Whomp!",
        description: "Basic attack",
        targetType: "",
        success: [
            { type: "textMessage", text: "{CASTER} uses {ACTION} on {TARGET}" },
            { type: "animation", animation: "spin" },
            { type: "stateChange", damage: 10 },
        ],
    },
    saucyStatus: {
        name: "Tomato Squeeze",
        description: "Recover Hp",
        targetType: "friendly",
        success: [
            { type: "textMessage", text: "{CASTER} uses {ACTION}" },
            { type: "animation", animation: "rotate" },
            { type: "stateChange", status: { type: "saucy", expiresIn: 2 } , },
        ],
    },
```

2. pizzas.js

存放 pizza 們，每個 pizza 會有各自可以用的 action



```
window.Pizzas ={
    s001:{
        name: "Slice Samuri",
        type: PizzaTypes.spicy,
        src: "/img/characters/pizzas/s001.png",
        icon: "img/icons/spicy.png",
        actions:[
            "clumsyStatus",
            "saucyStatus",
            "damage1",
        ],
    },
    v001:{···
    },
    f001:{···
    },
}
```

3. Battle.js

這場 battle 的基本資訊

i. 有哪些人物參與對戰

會直接將 pizza 的參數也導入到人物裡

```
class Battle {
    constructor() {
        this.combatants = {
            "player1": new Combatant({
                ...Pizzas.s001,
                /*
                name: "Slice Samuri",
                type: PizzaTypes.spicy,
                src: "/img/characters/pizzas/s001.png",
                icon: "img/icons/spicy.png",
                */

                team: "player", //enemy
                hp: 40,
                maxHp: 50,
                xp: 70,
                maxXp:100,
                level: 1,
                status: null,
                isPlayerControlled: true,
            }, this),

            "enemy1": new Combatant({…
            }, this),

            "enemy2": new Combatant({…
            }, this),
```

ii. 兩隊上場分別上場的人物

```
this.activeCombatants = {
    player: "player1",
    enemy: "enemy1",
}
```

4. turnCycle.js

有攻擊權交換的 method，並根據現在的攻擊方產生相對應的 menu

i. async turn()

每一輪攻擊後一一執行攻擊成功的 events(利用 await 和 resolve)

```
const resultingEvent = caster.getReplacedEvents(submission.action.success);
/**success: [
    {type: "textMessage", text: "{Pizza} uses Whomp!"},
    {type: "animation", animation: "toBeDefined"},
    {type: "textMessage", text: "Something happend!"},
    {type: "stateChange", damage: 10},
], */

for (let i = 0; i < resultingEvent.length; i++) {
    const event = {
        ...resultingEvent[i],
        submission,
        action: submission.action,
        caster: caster,
        target: submission.target,
    }
    await this.onNewEvent(event);
}
```

onNewEvent 會將傳入的 event 透過 BattleEvent 定義的函式執行(類似第 9.的 event_test.js)，並 resolve() @Battle.js

```
this.turnCycle = new TurnCycle({
    battle: this,
    onNewEvent: event =>{
        return new Promise(resolve=>{
            const battleEvent = new BattleEvent(event, this);
            battleEvent.init(resolve);
        })
    }
});
```

同樣的執行殘留在角色身上的效果(postEvents)

```
const postEvents = caster.getPostEvents();
for(let i = 0; i<postEvents.length; i++){
    const event = {
        ...postEvents[i],
        submission,
        action: submission.action,
        caster: caster,
        target: submission.target,
    }
    await this.onNewEvent(event);
}
```

最後轉換攻擊方，並重新呼叫自己(不斷輪流)

```
this.currentTeam = this.currentTeam === "player" ? "enemy": "player";
this.turn();
```

5. Combatant.js

i. 創立對戰角色

```
class Combatant {
    constructor(config, battle) {

        /*config = {...Pizzas.s001,
            team: "player", //enemy
            hp: 40,
            maxHp: 50,
            xp: 70,
            maxXp:100,
            level: 1,
            status: null,
            isPlayerControlled: true,
            }*/

        Object.keys(config).forEach(key => {
            this[key] = config[key];
        })

        this.battle = battle;
    }
```

```
"enemy1": new Combatant({
    ...Pizzas["v001"],
    team: "enemy",
    hp: 20,
    maxHp: 50,
    xp: 50,
    maxXp:50,
    level: 1,
    status: null,
}, this),
```

@Battle.js

ii. 可以取得其 Hp, Xp, Active 狀態

(若 not active 則不會顯示於畫面)

```
get hpPercent() {
    const percent = this.hp / this.maxHp * 100;
    return percent > 0 ? percent : 0;
}

get xpPercent() {···
}

get isActive() {
    return this.battle.activeCombatants[this.team] === this.id;
}
```

iii. update()

利用 Hp/maxHp 來改變生命值長條的寬度，同理用於 xp

```
//update hp xp bar
this.hpFills.forEach(rect => {
    rect.style.width = `${this.hpPercent}%`
});
this.xpFills.forEach(rect => {
    rect.style.width = `${this.xpPercent}%`
});
```

六、 心得

一直很想做 RPG 遊戲，所以先挑了一個來學，沒想到有點難跟複雜，所以沒有做得很完整，但過程中真的收穫很多。跟著教學去完成的過程中，除了更熟悉 Javascript 的語法及物件導向，這個遊戲中用到許多 async function 和 Promise and resolve() 也是我這次實作的一大收穫，一開始不太懂 resolve() 的意思和時機，所以就去找了很多說明，再透過不斷利用就慢慢抓到那個感覺了，到後來可以自己知道這時候需要等 resolve。

之前寫一些小遊戲的時候都 1、2 份檔案解決，但這份專題讓我知道要寫一個完整、功能齊全的遊戲，要適時宣告新的 class 和 method，甚至利用 window 來存遊戲預設參數。除了主要 Javascript 架構外，這份專題的一些小動畫也是一個特色，例如人物走路時方向改變面向改變，角色選擇不同攻擊方式時，會有相對應的動畫產生，製作這些動畫的過程中，上網找了很多 css 的參數，學到了更多可以使用的參數，也更熟悉 keyframe 的應用。

我覺得透過寫遊戲來更熟悉一個程式語言是個很棒的方式，謝謝老師這學期的教導，帶我們從簡單的網頁開始，一步步寫出網頁遊戲，這堂課帶給我很多收穫及成就感！

七、 參考資料

1. Pizza Legends - JavaScript RPG by Drew Conley
2. Window.requestAnimationFrame() - Web APIs | MDN (mozilla.org)
3. Object.values() - JavaScript | MDN (mozilla.org)
4. Promise.resolve() - JavaScript | MDN (mozilla.org)
5. CustomEvent() - Web APIs | MDN (mozilla.org)
6. Array.prototype.sort() - JavaScript | MDN (mozilla.org)
7. Array.prototype.filter() - JavaScript | MDN (mozilla.org)
8. viewBox - SVG：可缩放矢量图形 | MDN (mozilla.org)
9. Pixilart - Free online pixel art drawing tool