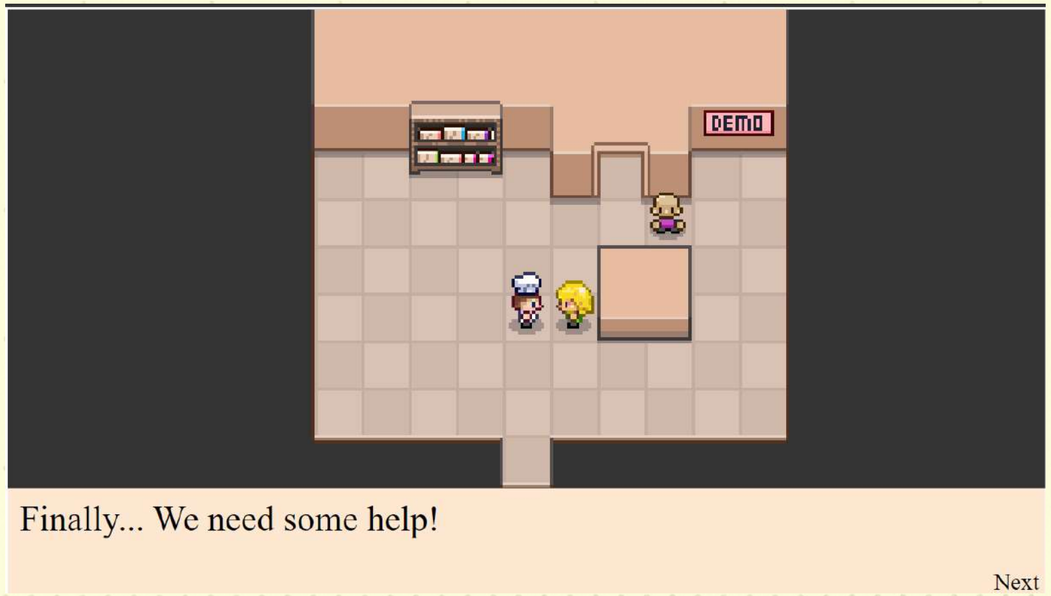
PIZZA RPG

黃綵誼 111613025

START

遊戲說明

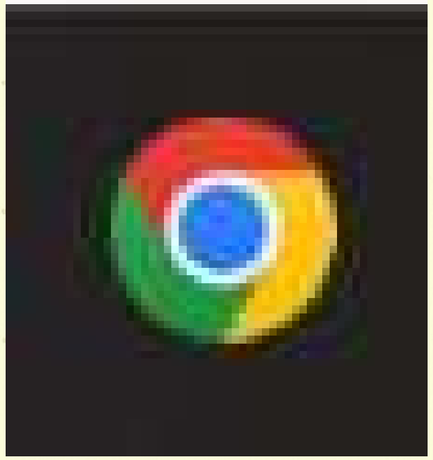
使用者要在場景內跟角色互動，完成任務，打敗壞人



開發環境

VS CODE

LIVE SERVER



CHROME

程式碼說明 (RPG動畫)

程式碼說明

class Overworld -> startGameLoop() -> step
負責不斷更新畫面，製造動畫的效果

JS overworld_test.js > Overworld > startGameLoop

```
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);
```

每一個step都先清空畫面

update每一個物件的位置

畫底層地圖

程式碼說明

JS overworld_test.js > Overworld > startGameLoop

```
        //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.sprite.draw(this.ctx, cameraPerson);
        })

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

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

依照物件的y值畫每一個物件
(避免上下重疊的順序錯誤)

畫上層地圖

用requestAnimationFrame
不斷執行step更新畫面

程式碼說明

JS sprite_test.js > Sprite > draw

讓角色位在正中央，其他東西以角色的相對位置下去畫

```
//draw img
draw(ctx, cameraPerson) {
    const x = this.gameObject.x - 8 + utils.withGrid(10.5) - cameraPerson.x;
    const y = this.gameObject.y - 18 +utils.withGrid(6) - cameraPerson.y;

    if (this.isShadowLoaded) {…
    }
    const [frameX, frameY] = this.frame;
    if (this.isLoaded) {
        ctx.drawImage(this.image,
            frameX * 32, frameY * 32,
            32, 32,
            x, y,
            32, 32
        )
        //console.log("draw successful");
    } else {…
    }
    this.updateAnimationProgress();
}
```

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

只裁切特定格子

程式碼說明

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

JS sprite_test.js > Sprite > constructor

```
//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]],
}
this.currentAnimation = config.currentAnimation || "idle-down";
this.currentAnimationFrame = 0;
```

依照人物狀態和方向，決定要擷取照片的哪部分

程式碼說明

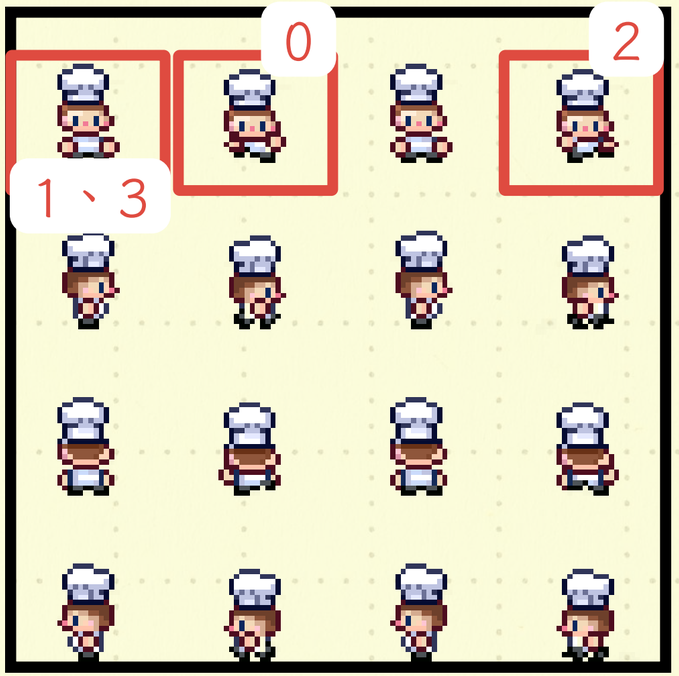
sprite_test.js > Sprite > updateAnimationProgress

```
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;
    }
}
```

更新animationFrame，擷取動作的下一個部分

0 1 2 3

```
"walk-down": [[1, 0], [0, 0], [3, 0], [0, 0]],
```



程式碼說明

程式碼說明 (RPG事件)

程式碼說明

```
cutsceneSpaces: {
    [utils.asGridCoords(5, 10)]: [
        {
            events: [
                { type: "changeMap", map: "Kitchen" },
            ]
        }
    ],
```

```
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: "walk", direction: "left" },
                    { type: "walk", direction: "up", },
                    { type: "stand", direction: "right", time: 500 },
                    { type: "stand", direction: "left", time: 500 },
                    { type: "walk", direction: "up", },
```

Overworldmaps

存放地圖參數

ex: npc固定動作、事件格子

程式碼說明

JS overworld_test.js > Overworld > bindAcionInput

```
bindAcionInput() {
    new KeyPressListene("Enter", () => {
        //is there a person here to talk to
        this.map.checkForActionCutscene();
    })
}
```

當玩家按下enter時執行
checkForActionCutscene

JS KeyPress_test.js > KeyPressListener > constructor

```
class KeyPressListener {
    constructor(keyCode, callback) {
        let keySafe = true;
        this.keydownFunction = function (event) {
            if (event.code === keyCode) {
                if (keySafe) {
                    keySafe = false;
                    callback();
                }
            }
        }
        this.keyupFunction = function (event) {
            if (event.code === keyCode) {
                keySafe = true;
            }
        }

        document.addEventListener("keydown", this.keydownFunction);
        document.addEventListener("keyup", this.keyupFunction);
    }
```

利用class KeyPressListener
可以單純看一個特定按鍵有沒有被觸發，
並在觸發後執行相應的function

程式碼說明

map_test.js > OverworldMap > checkForActionCutscene

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

如果玩家前方有可以互動的物件
執行相對應的事件

```
npc1: new Person({
    x: utils.withGrid(4),
    y: utils.withGrid(7),
    src: "./img/characters/people/npc1.png",
    behaviorLoop: [ ···
    ],
    talking: [
        {
            events: [
                { type: "textMessage", text: "Finally... We need some help!"
                { type: "textMessage", text: "Go find and beat the bad guy i
                { who: "npc1", type: "stand", direction: "down", time: 500 }
            ],
        },
    ]
```

程式碼說明

JS map_test.js > OverworldMap > checkForFootstepCutscene

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

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

如果玩家踩到的格子是cutsceneSpace
執行相對應的事件

```
cutsceneSpaces: {
    [utils.asGridCoords(5, 10)]: [
        {
            events: [
                { type: "changeMap", map: "Kitchen" },
            ]
        }
    ],
```

程式碼說明

JS map_test.js > OverworldMap > startCutscene

```
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);
    });
```

用class OverworldEvent 一一執行事件
利用await()

JS event_test.js > OverworldEvent > init

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

程式碼說明

JS event_test.js > OverworldEvent > changeMap

```
changeMap(resolve) {
    const sceneTransition = new SceneTransition();

    //container, callback
    sceneTransition.init(document.querySelector(".game-container"), () => {
        console.log("animation end");
        this.map.overworld.startMap(window.OverworldMaps[this.event.map]);
        resolve();

        sceneTransition.fadeOut();
    })
}
```

事件函數
執行完後resolve()

JS event_test.js > OverworldEvent > init

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

resolve()成功後就會return

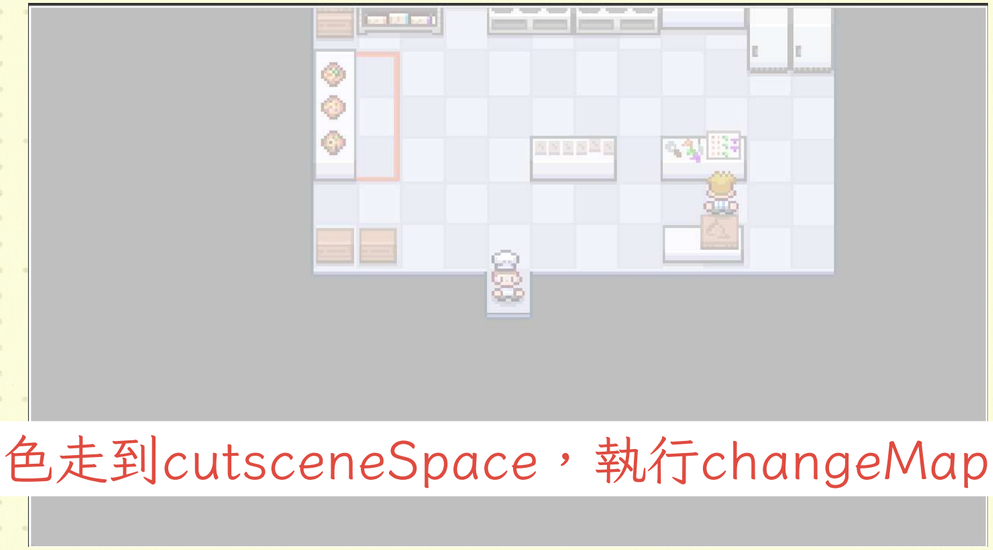
程式碼說明

JS map_test.js > OverworldMap > startCutscene

```
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);
    });
```

await 到了之後才會繼續執行下面的程式

程式碼說明



角色走到cutsceneSpace，執行changeMap的事件

角色面向npc後按enter，
執行npc talking裡的事件

程式碼說明

JS obj_test.js > GameObject > mount

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

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

物品建立時於物品位置建立wall

物品移動時移除前一個位置的wall於新位置建立wall

JS map_test.js > OverworldMap > removeWall

```
addWall(x, y) {
    this.walls[`${x},${y}`] = true;
}
```

程式碼說明

person_test.js > Person > startBehavior

```
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) && this.movingProgress
            if (behavior.retry) {
                setTimeout(() => {
                    this.startBehavior(state, behavior)
                }, 10);
            }
            return;
        }
```

如果角色要走的方向前面是wall，則不走

過一段時間再試一次(如果前面的人走掉的case)

map_test.js > OverworldMap > isSpaceTaken

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

程式碼說明

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

走的時候移除原本位置的牆壁，於新位置建立牆壁

```
removeWall(x, y) {
    delete this.walls[`${x},${y}`];
}

moveWall(wasX, wasY, direction) {
    //console.log("move wall");
    this.removeWall(wasX, wasY);
    const { x, y } = utils.nextPosition(wasX, wasY, direction);
    this.addWall(x, y);
}
```

程式碼說明

npc行徑路線被玩家建立的wall擋住，停止行走

玩家走掉後wall被移走，npc得以繼續前進

程式碼說明 (對戰模式)

程式碼說明

```
class PlayerState {
    constructor() {
        this.pizzas = {
            "p1": {
                pizzaId: "s002",
                hp: 30,
                maxHp: 30,
                xp: 95,
                maxXp: 100,
                level: 1,
                status: null,
            },
            "p2": {…
            },
            "p3": {…
            },
        };

        this.lineup = ["p1", "p2", "p3"];
```

playerState.js -> class PlayerState

建立玩家，內含

1. 玩家可以用的pizza，及各參數
2. 預設pizza的出場順序
3. 玩家有的特殊物品(items)

```
this.items = [
    { actionId: "item_recoverStatus", itemId: "i1",},
    { actionId: "item_recoverStatus", itemId: "i2",},
    { actionId: "item_recoverHp", itemId: "i3",},
    { actionId: "item_protect", itemId: "i4",},
];
```

程式碼說明

g > content > JS enemies.js > ...

```
window.enemies = {
    "Bob": {···
    },
    "Jackie": {
        name: "Jackie",
        src: "/img/characters/people/erio.png",
        pizzas: {
            "a": {
                pizzaId: "f001",
                hp: 30,
                maxHp: 30,
                xp: 0,
                maxXp: 50,
                level: 1,
                status: null,
            },
            "b": {
                pizzaId: "v001",
                maxHp: 30,
```

window.enemies
定義每個敵人及其可以派出的pizza

pg > content > JS pizzas.js > v002

```
window.Pizzas ={
    s001:{
        name: "Slice Samuri",
        description: "Classic",
        type: PizzaTypes.spicy,
        src: "/img/characters/pizzas/s001.png",
        icon: "/img/icons/spicy.png",
        actions:["damage1","saucy","damage2","clumsyStatus","sp
    },
    s002:{···
    },
    v001:{···
    },
    v002:{···
    },
    f001:{···
    },
}
```

window.Pizzas
定義每個pizza還有其可以用的攻擊招式

程式碼說明

```
battle > JS Battle.js > Battle
class Battle {
    constructor({ enemy, onComplete }) {
        this.enemy = enemy;
        //enemy: enemies[this.event.enemyId],
        this.onComplete = onComplete;

        this.combatants = {};
        this.activeCombatants = {
            player: null,
            enemy: null,
        };
        //dynamically add the player team
        window.playerState.lineup.forEach(id => {
            this.addCombatant(id, "player", window.playerState.pizzas[id]);
        });

        //dynamically add the ene
        Object.keys(this.enemy.pizzas).forEach(key => {
            this.addCombatant("e_" + key, "enemy", this.enemy.pizzas[key]);
        });
        this.items = [];
        this.usedItemIds = {};
        //dynamically add items for player team
        window.playerState.items.forEach(item => {
            this.items.push({
                ...item,
                team: "player",
            })
        });
```

class Battle
對戰的相關資訊
ex: 對戰角色跟其參數

記錄用掉的物品

從playerState加入玩家有的物品

從playerState及window.enemies
加入對戰有的角色

程式碼說明

battle > JS Battle.js > Battle > addCombatant

```
addCombatant(id, team, config) {
    this.combatants[id] = new Combatant({
        ...Pizzas[config.pizzaId],
        ...config,
        team: team,
        isPlayerControlled: team === "player" ? true : false,
    }, this);

    let activePlayerId = this.activeCombatants[team];
    let pizza = window.playerState.pizzas[activePlayerId];
    if (pizza) {
        this.activeCombatants[team] = (pizza.hp > 0) ? activePlayerId : id;
    } else {
        this.activeCombatants[team] = id;
    }
```

添加的同時
設定activeCombatant

程式碼說明

```
class TurnCycle {
    constructor({ battle, onNewEvent, onWinner }) {
        this.battle = battle;
        this.onNewEvent = onNewEvent;
        this.onWinner = onWinner;
        this.currentTeam = "player"; // enemy
    }

    async turn() {
        const casterId = this.battle.activeCombatants[this.currentTeam];
        const caster = this.battle.combatants[casterId];
```

class TurnCycle
紀錄現在是誰攻擊，
並於turn中執行每一輪的事件(攻擊/受效果影響)

程式碼說明

attle > JS turnCycle.js > TurnCycle > turn

```
async turn() {
    const casterId = this.battle.activeCombatan
    const caster = this.battle.combatants[caste

    const enemyId = this.battle.activeCombatant
    const enemy = this.battle.combatants[enemy
    //console.log(`enemy.name from turn(): ${en

    const submission = await this.onNewEvent({
        type: "submissionMenu",
        caster: caster,
        enemy: enemy,
    });
```

每一輪先產生選單

選單被提交後，執行相對應的event

```
const resultingEvent = enemy.getReplacedEvents(submission.action)

for (let i = 0; i < resultingEvent.length; i++) {
    const event = {
        ...resultingEvent[i],
        submission,
        action: submission.action,
        caster: caster,
        target: submission.target,
    }
    // console.log(`event.caster,enemy.name from turnCycle: ${even
    await this.onNewEvent(event);
}
```

attle > JS Combatant.js > Combatant > getReplacedEvents

```
getReplacedEvents(originalAction) {
    if(this.status?.type === "cheesy"){
        return originalAction.fail || originalAction.success;
    }
    return originalAction.success;
}
```

如果攻擊對象被cheese保護，
就執行攻擊失敗的event，否則執行成功的

程式碼說明

再來執行自己身上受效果影響的事件

```
//check for post event
// do things after original turn submission
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);
}
```

ttle > JS Combatant.js > Combatant > getPostEvents

```
getPostEvents() {
    if (this.status?.type === "saucy") {
        return [
            { type: "textMessage", text: "Feeling Saucy!" },
            { type: "stateChange", recover: 5, onCaster: true },
        ]
    }
    if (this.status?.type === "clumsy") {…
    }
    if(this.status?.type === "spicy"){…
    }
    if(this.status?.type === "magic"){…
    }

    return [];
}
```

class Combatant ->getPostEvents
依據角色身上的效果回傳事件

程式碼說明


Slice Samuri
1
clumsy
Portobello Express
1
受到效果影響
Feeling Clumsy...


Slice Samuri
1
Portobello Express
1
cheesy
有起司保護，所以攻擊失敗
Portobello Express is protected!!
Next

程式碼說明

attle > JS Team.js > Team > createElement

```
createElement() {
    this.element = document.createElement("div");
    this.element.classList.add("team");
    this.element.setAttribute("data-team", this.team);
    this.combatants.forEach(cmbt => {
        let icon = document.createElement("div");
        icon.setAttribute("data-combatant", cmbt.id);
        icon.innerHTML = (`
            <img class="alive-pizza" src="/img/icons/alive-pizza.png"></>
            <img class="dead-pizza" src="/img/icons/dead-pizza.png"></>
            <img class="indicator" src="/img/icons/indicator.png"></>
        `);
        this.element.appendChild(icon);
    });
}
```

Team.js -> class Team -> createElement()
為每一塊pizza創建一個小icon，
後續利用css控制該icon顯示的狀態

程式碼說明

ttle > JS Team.js > Team > update

```
update() {
    this.combatants.forEach(cmbt=>{
        //console.log(cmbt.id, cmbt.isActive);
        const icon = this.element.querySelector(`[data-combatant=${cmbt.id}]`)
        icon.setAttribute("data-dead", cmbt.hp<=0);
        icon.setAttribute("data-active", cmbt.isActive);
    })
}
```

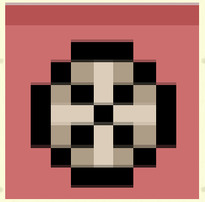
Team.js -> class Team -> update()
依據pizza狀態設立html attribute

g > styles > # team.css > .team [data-dead="true"] .dead-pizza

```
.team [data-dead="true"] .dead-pizza{
    display: block;
}

.team [data-active="true"] .indicator{
    display: block;
}
```

team.css
利用attribute來更改icon顯示的樣式

.indicator



.dead-pizza

程式碼說明

JS turnCycle.js > TurnCycle > turn

```
//target die?
const targetDie = submission.target.hp <= 0;
if (targetDie) {
    await this.onNewEvent( ···
    );

    if(submission.target.team === "enemy"){ ··
    }

    //winning team? end battle
    const winner = this.getWinner();
    if (winner === "player") {
        await this.onNewEvent({ ···
        });
        this.onWinner(winner);
        return;
    }else if (winner === "enemy") { ···
    }
    else { ···
    }
}
```

```
else {
    const replacement = await this.onNewEvent({
        type: "replacementMenu",
        team: submission.target.team,
    });

    await this.onNewEvent({
        type: "replace",
        replacement: replace
    });
    await this.onNewEvent({ ···
    });
}
```

如果還有可以上場的pizza
就呼叫替換選單，
讓使用者選擇要替換的pizza

turnCycle.js -> class TurnCycle -> async turn()
執行完攻擊事件後判斷是否有贏家產生，
有的話呼叫onWinner

程式碼說明


戰鬥狀態
目前使用的pizza及還有幾片pizza
1
saucy
Bacon Brigade
1
替換選單
Pizza簡介
Mushroom paradise
Portobello Express
Friarielli Broccoli

程式碼說明

JS Battle.js > Battle > init > onWinner

```
},
onWinner: winner => {
    if (winner === "player") {
        const playerState = window.playerState;
        Object.keys(playerState.pizzas).forEach(id => {
            const playerStatePizza = playerState.pizzas[id];
            const combatant = this.combatants[id];
            if (combatant) {
                playerStatePizza.hp = combatant.hp;
                playerStatePizza.maxHp = combatant.maxHp;
                playerStatePizza.xp = combatant.xp;
                playerStatePizza.maxXp = combatant.maxXp;
                playerStatePizza.level = combatant.level;
            }
        });

        //delete used items
        playerState.items.filter(i => {
            return this.usedItemIds[i.itemId] !== true
        });
    }
    this.element.remove();
    this.onComplete();
}
```

onWinner 會更新玩家的狀態
(可以保留到下一場對戰)

利用Array.filter() 過濾已經用過的items
(下一場對戰就沒有該物品可以用了)

程式碼說明

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

class Combatant ->update method

依據hp/xp值來更新hp/xp條的寬度

程式碼說明

```
window.Actions = {
    //attack
    damage1: {
        name: "Whomp!",
        description: "Basic attack (damage: 5)",
        targetType: "",
        alwaysAvailable: true,
        success: [
            { type: "textMessage", text: "{CASTER} uses {ACTION}" },
            { type: "animation", animation: "spin" },
            { type: "stateChange", damage: 5 },
        ],
        fail: [
            { type: "textMessage", text: "{CASTER} uses {ACTION}" },
            { type: "animation", animation: "spin" },
            { type: "animation", animation: "protect" },
            { type: "textMessage", text: "{TARGET} is protected!!" },
        ],
    },

    damage2: {···
    },
```

alwaysAvailable紀錄該招式是不是一直都可以用

actions.js -> window.Actions
存放攻擊招式及攻擊成功/失敗要產生的event

程式碼說明

```
getPages() {
    const backOption = {
        label: "Go Back",
        description: "Return to previous page",
        handler: () => {
            this.keyboardMenu.setOptions(this.getPages().root);
        }
    };

    return {
        root: [···
        ],
        attacks: [
            ...this.caster.actions.map(key => {
                const action = window.Actions[key];
                return {
                    label: action.name,
                    description: action.description,
                    disabled: action.alwaysAvailable?
                            false : utils.randomFromArray([true,false])
                    handler: () => {
                        this.menuSubmit(action);
                    }
```

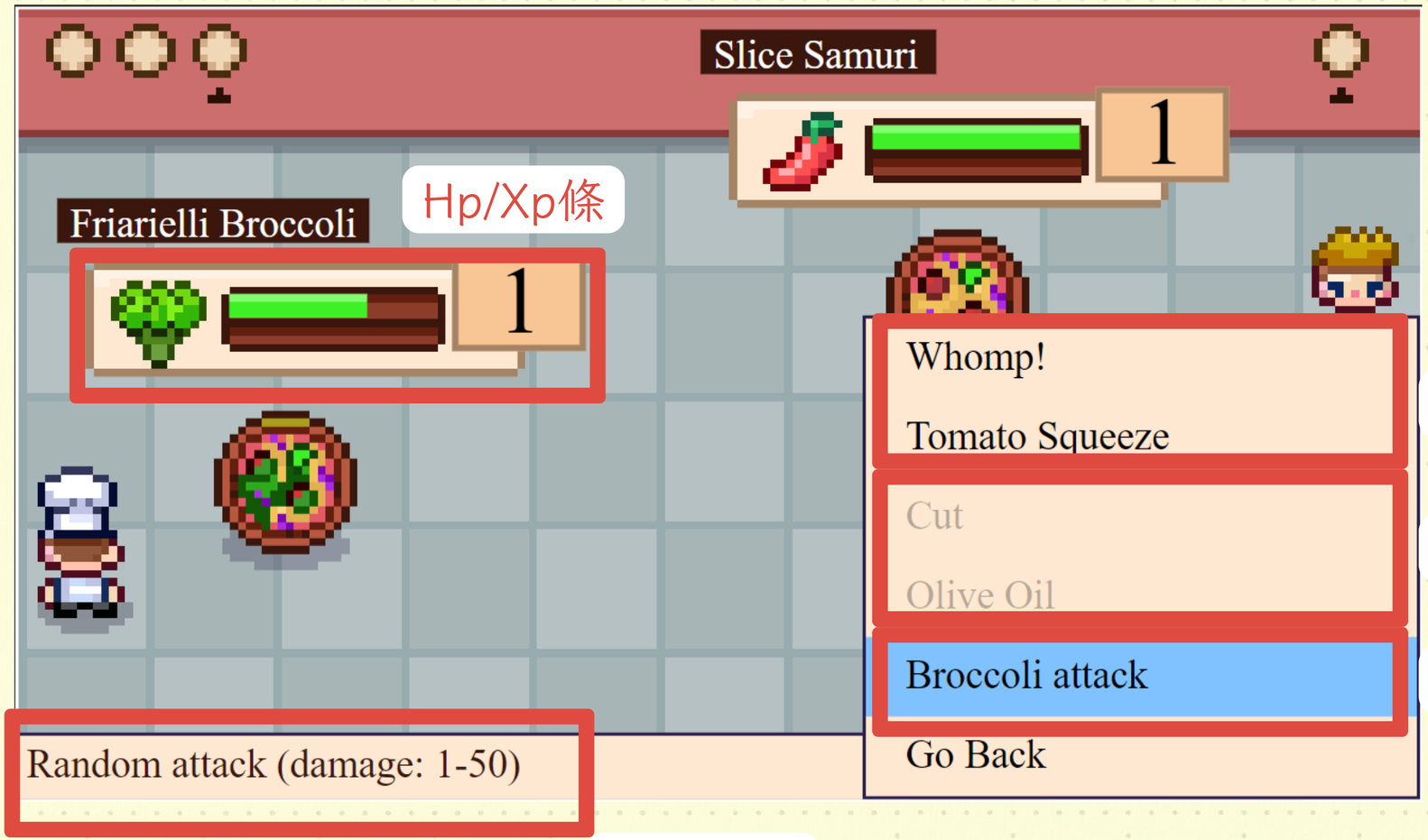
每一頁都添加backOption
讓使用者可以回到主選單

class SubmissionMenu ->getPages()

依據角色現在擁有的攻擊招式
return menu要有的選項

如果該招式的alwaysAvailabe是false，
則每一輪有50%的機會可以使用，
透過將button設為disabled來達成此效果

程式碼說明


Slice Samuri
1
Friarielli Broccoli
1
Hp/Xp條
Whomp!
Tomato Squeeze
Cut
Olive Oil
Broccoli attack
Go Back
Random attack (damage: 1-50)
有兩個固定普通招式
這一輪無法使用的特殊招式
這一輪可以用的特殊招式
招式說明

特殊功能

玩家可自由選擇攻擊招式


```
attle > JS SubmissionMenu.js > SubmissionMenu > decide

decide() {
    this.menuSubmit(Actions[utils.randomFromArray(this.caster.actions)]);
}
```

```
randomFromArray(arr){
    return arr[Math.floor(Math.random()*arr.length)];
}
```

電腦會隨機選擇攻擊招式
利用Math.random()

場景跟著角色移動 (角色維持在畫面正中間)

打字機效果

場景轉換動畫

攻擊動畫

心得

1. 更熟悉 async function
2. 學會宣告需要的class及method以便重複使用
3. 學到很多css動畫的參數，更熟悉@keyframe
4. RPG好難，但越寫越上手

未來展望

1. 完整故事線
2. 新增除了對戰外的小遊戲 (製作PIZZA等)

更多

1. 遊戲ZIP檔

2. PDF書面報告
(內含更多程式碼說明、心得、參考資料)

THANK YOU

See you next time!