**Domenico Robert Urso**

**Interactive Media**

**Programming for Digital Media II**

**Username: tgz521**

**ID: 303042245**

**Exam No: Y3952739**




# Byronic: Trials of the Hero

## Technical Report




*Implemented Functionality*



Byronic, is a turn-based top-down action-adventure game inspired by traditional Super Nintendo Entertainment System JRPG's; Final Fantasy II, Golden Sun, Chrono Trigger.[1] [2] [3] Upon 'New Game,' the player arrives in a forest they cannot escape from. In order to trigger the door to unlock, 10 enemies must fall. To trigger a combat encounter, the player collides with enemy tiles, represented by shadow-like creatures.



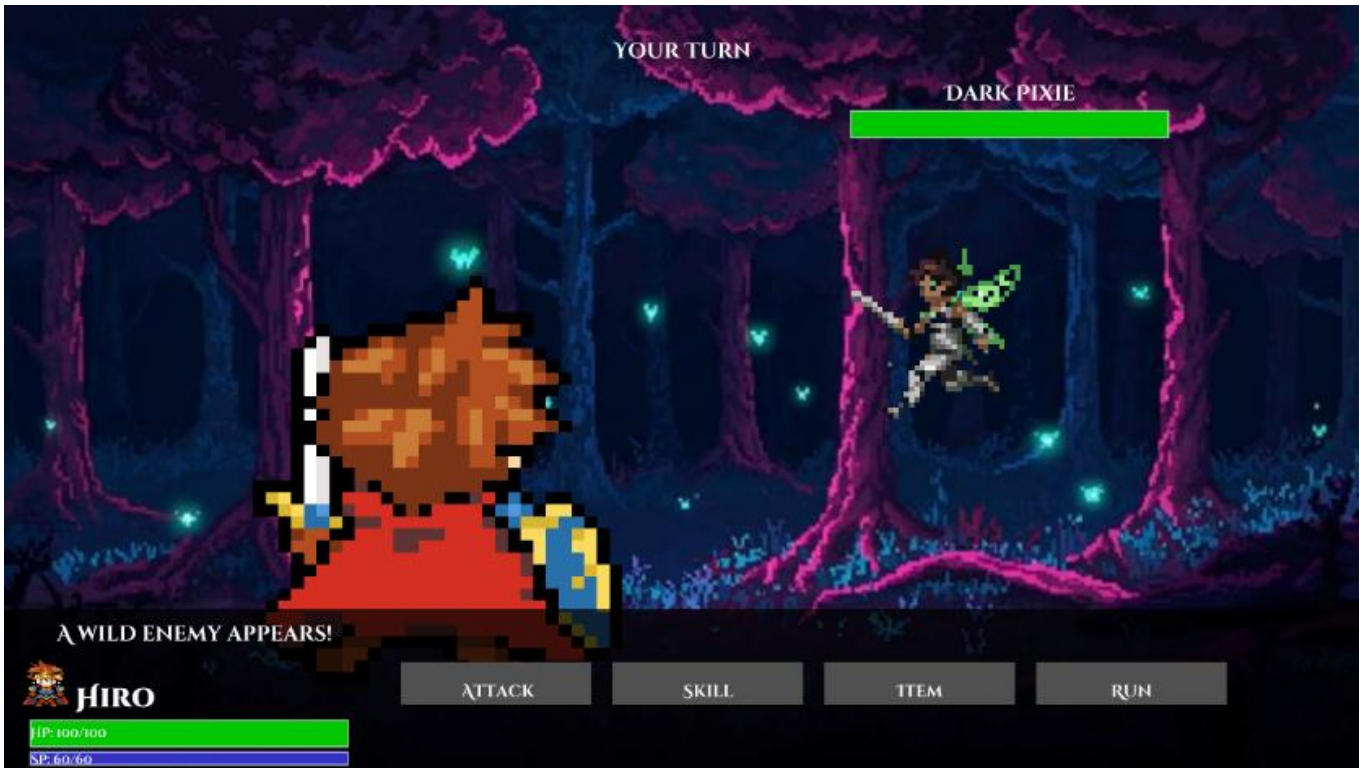

Enemies move throughout empty-spaces in the tilemap. Upon interaction, a combat encounter with one of two random enemies begins, transitioning to the turn-based user-interface. The Slime has a 40% chance to heal 10HP in Level 1, increasing by 10 each area. Dark Pixies do higher damage. Enemy Maximum HP starts at 75, growing a factor of 20 per each level.





[1] https://en.wikipedia.org/wiki/Final_Fantasy_II
[2] https://nintendo.fandom.com/wiki/Golden_Sun_(video_game)
[3] https://purenintendo.com/retro-review-chrono-trigger-snes/

The player has 4 choices each turn; Attack, Skill, Item, Run. Attacking deals random damage between 9 and 16, these increase by 5 each level up. Item allows use of potions or ethers; potions heal for 30 of the current 100HP, ethers heal 15SP of the current 60SP. At Level 10, Medium Potions are unlocked healing 50HP, Level 20 unlocks Large Potions; healing 100HP. Maximum HP increases by 20 per level increase. Run has a 50% chance to end combat; however, the enemy tile will remain and they'll be no profits.




*Skills, Items & NPC's*

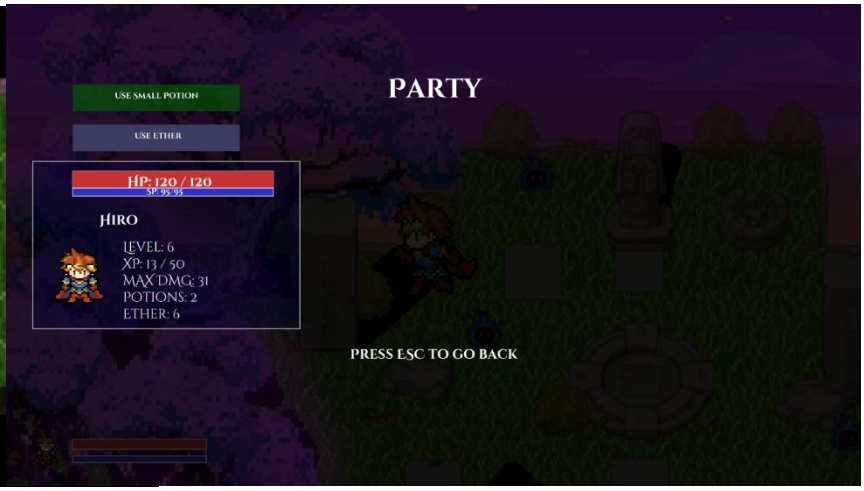
```
messages.push(`Level Up! You are now level ${this.playerLevel}!`);
messages.push(`Max HP increased by 40!`);
messages.push(`Max SP increased by 5!`);
messages.push(`Max DMG increased by 5!`);
messages.push(`Max Skill DMG increased by 6!`);
```

When enemies fall the player earns; 2 potions, 1 ether and upon a level up, stats increase. SP is spent to use skills, either Electro or Inferno. Electro is Hiro's first skill, unlocked at level 7, with a 50% chance to stun the enemy meaning they miss a turn. Inferno at 13, deals 8-14 burn-tick damage, lasting 2-6 turns. Electro deals 16-22 damage initially, with Inferno dealing 13-20.








*Progression*



Upon defeating 10 enemies, Marx, the Warden, will greet you with a cutscene which unlocks the right-side door. In Area 2, enemy HP has increased by 20 and damage range by 8. However, back-tracking is possible by returning to the entrance, allowing grinding. Marx will arrive, after the interaction prompt, the second cutscene will begin. Marx begins to explain the lore of Byronic.




*Unlocking Free Roam & Exploring the Manor*

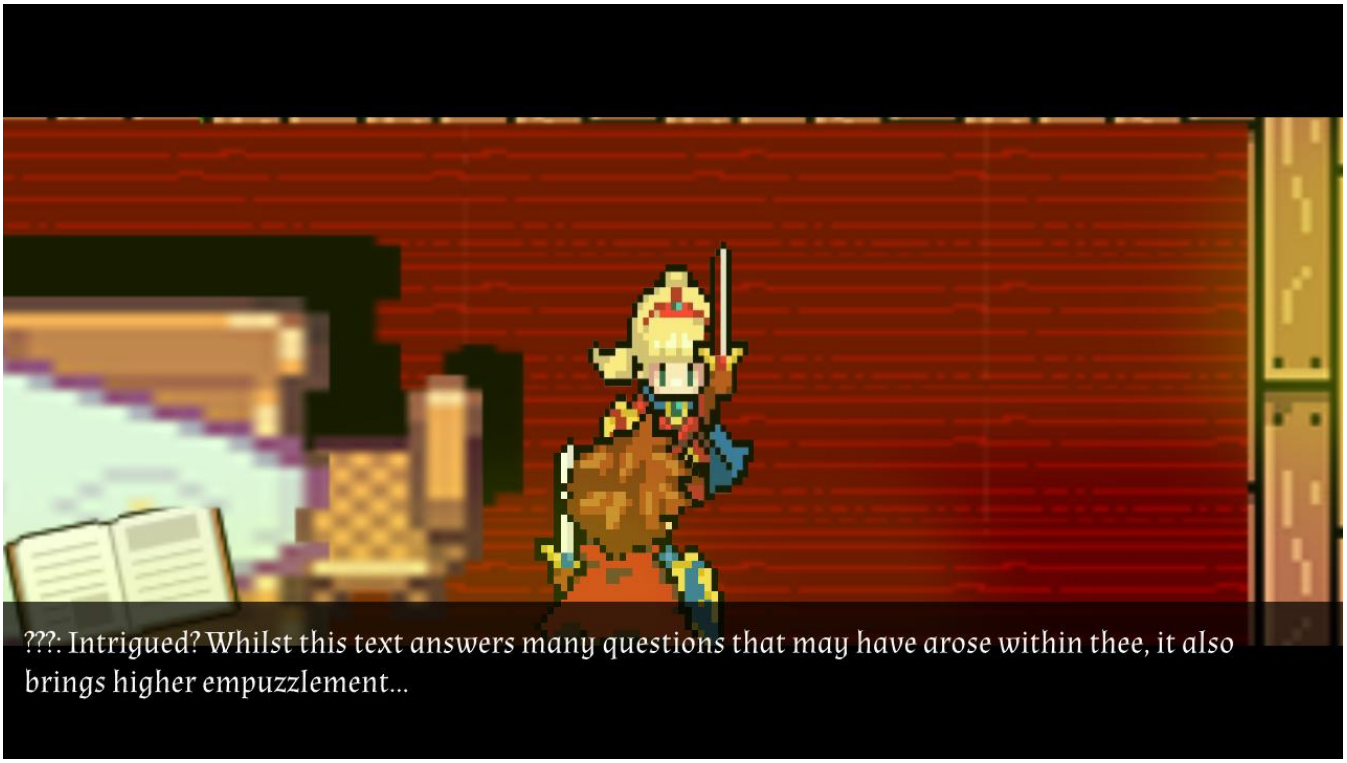


'An open-world game features non-linear game world design where the player is able to freely traverse the environment.'[4] Area 2 onwards is unrestricted; cliff edges allow hopping between connected areas similar to early Pokémon.[5] From here, you 'grow stronger,' as Marx instructs. At any point, a Manor can be discovered near Area 3. You are prompted to

[4] https://gameopedia.com/blogs/an-introduction-to-open-world-games
[5] https://www.serebii.net/pokearth/kanto/3rd/mt.moon.shtml



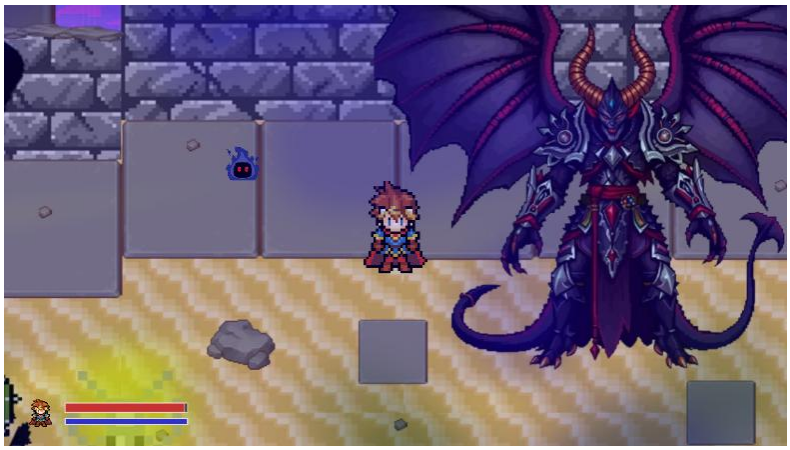
interact with objects, initiating dialogue via the protagonists' thoughts. The Statue is the crucial interaction, initiating Lyn, one of the last Deixis of Symphonia. She'll check if you've interacted with the book, providing a FromSoftware-esc lore extract. Afterwards, Lyn will reveal the final story objective.



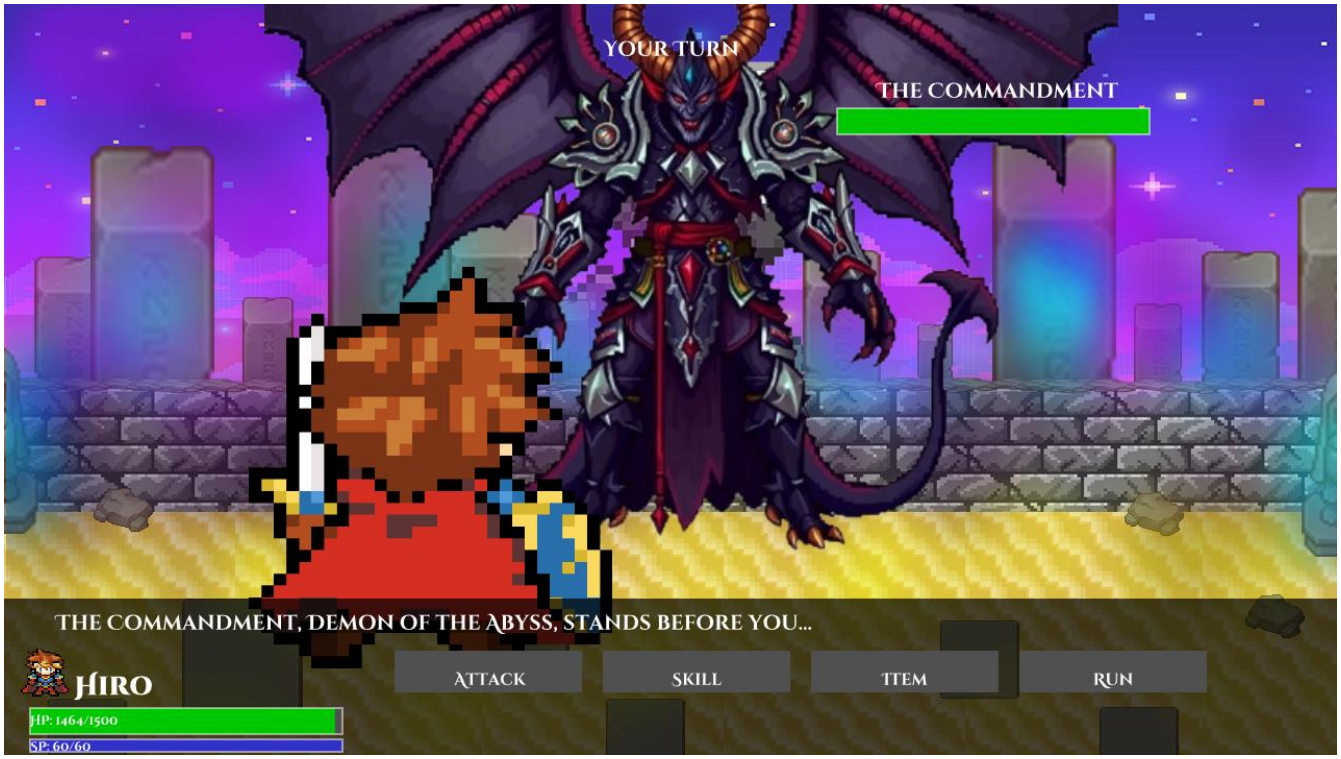

*The Final Act: The Commandment & The Time Loop*

Byronic Asylum past the desert of Area 3 resides the demon, The Commandment. The player must reach Level 30, then the Byronic Key within the inventory will reach Level 30, allowing passage to the final boss.





The Commandment has 650 base HP plus the player's level times 5. So, at Level 30, he'll have 800HP. He does 80-120 fixed damage, meaning he can be overpowered adding a post-game. The player is unable to run here but can infinitely back-track, grind and retry upon a defeat.




*Pause Menu, Party Menu & Player Stats*

The Pause Menu lists controls and prompts to other functions. The Party Menu views Hiro's stats, as well as allows the use of items outside of combat. The Inventory allows viewing of total items and Byronic Key level. Options include debug mode, displaying the tilemap, a reset button and developer god-mode. Save Game allows the game state to save to local storage, loadable from the Main Menu.












*Code Implementation*

*2D-Array Tilemap*

A 2D array made of multiple grids *(32x18)* of strings create the tilemap used for player and environment interaction and collision. Each character (0-9) is a tile representing numbers in **grid.cells[y][x]**. This function conversion occurs within **grid.loadFromArray(levels[index]);**

```
// key for level array:
// 1 = wall
// 2 = spawn
// 3 = door
// 4 = trigger
// 5 = locked door (unlocked when condition flag is true)
// 6 = trap (slows player temporarily)
// 7 = npc trigger (activates npc when stepped on)
// 8 = combat trigger (starts combat when stepped on)
// 9 = interact dialogue tile

let levels = [
[
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000011111111100000000000",
"11110000000011808000011111100011",
"00001111111110000101010800111100",
"01000000000118000100000010100000",
"01008000000100000200011000000000",
"00000000000001011000011011100000",
"08000000000000011000000000000000",
"00008000800000000000800000000000",
"00000011000001000080000000800000",
"00110011000001000000000000000002",
"00100000800000000001000000000085",
"00800000000000000000011000100000",
"00080001000000008000000080100000",
"00000000800000000080001000008000",
"00000800000008000000000000000800",
],

[
"13201000000000001000000000000000",
"10001010010000001000000000000000",
"10601000000000001000000000000000",
"11611000000000001000000000000000",
"11411000800001100111111111111000",
"11611000000011000111000000111111",
"00000000000000008000010110111001",
"00000000000000001000000800000001",
"00080000000008000008000000080001",
"00000000000008000000000100000821",
"00000000110000008008000000000001",
"00008000000000000000008000800001",
"00000000800010000000000000000085",
```

```
function loadLevel(index, playMusic = true) {
    currentLevel = index;

    eventManager = new EventManager();
    doorMap = {};

    for (let door of levelDoors) {
        if (door.level !== index) continue;

        const key = `${door.x},${door.y}`;
        doorMap[key] = {
            ...door,
            locked: door.condition ? true : false
        };
    }

    updateDoorLocks();

    grid.loadFromArray(levels[index]);

    grid.rows = levels[index].length;
    grid.cols = levels[index][0].length;

    // avoid overflow
    let cellW = width / grid.cols;
    let cellH = height / grid.rows;

    grid.cellSize = min(cellW, cellH);

    grid.offsetX = (width - grid.cols * grid.cellSize) /
    grid.offsetY = (height - grid.rows * grid.cellSize)

    npcs = [];

    let npcList = levelNPCs[index] || [];

    for (let i = 0; i < npcList.length; i++) {
        let npcData = npcList[i];

        let worldPath = npcData.path.map(p => {
            let pos = gridToWorld(p.x, p.y);
            return {
                x: pos.x,
                y: pos.y,
                wait: p.wait || 0,
                dialogue: p.dialogue || null
            };
        });
```

```
class Grid {
    /**
     * load grid cells for 2d array tile values
     * @param {Array<Array>} levelArray
     * @returns {void}
     */
    loadFromArray(levelArray) {
        this.rows = levelArray.length;
        this.cols = levelArray[0].length;

        this.cells = [];

        for (let y = 0; y < this.rows; y++) {
            this.cells[y] = [];

            for (let x = 0; x < this.cols; x++) {
                this.cells[y][x] = parseInt(levelArray[y][x]);
            }
        }
    }
```

**loadFromArray();** within **grid.js** contains **this.cells[y][x] = parseInt(levelArray[y][x]);** transforming the string map to a 2D array. Upon **loadLevel(index)**'s calling, the current level is set to the 2D array, loading corresponding data whilst calculating tile size as to fit the 16'9 canvas. **setupEvents()** scans tilemap functionality creating events; doors, traps, triggers…

The player and NPCs move using world/pixel coordinates. Function **gridToWorld(x,y)** is a helper, converting tile positions to tile-centre pixel positions. As collision's grid-based, but the player moves omni-directionally, I needed to bridge this. **player.update()** checks nearby tiles, if walls (1) are detected, the player collides. **EventManager.js**'s trigger function within **protagMove.js** detects player to tile movement, triggering corresponding events.

```
/**
 * grid coords to world pixels
 * @param {number} x
 * @param {number} y
 * @returns {{x:number,y:number}}
 */
function gridToWorld(x,y) {
    return {
        x: grid.offsetX + x*grid.cellSize + grid.cellSize/2,
        y: grid.offsetY + y*grid.cellSize + grid.cellSize/2
    };
}
```

```
// X movement
this.pos.x += moveX;
if (this.grid) {
    this.pos.x = this.grid.resolveMoveX(this.pos.x, this.pos.y, half, moveX);
}

// Y movement
this.pos.y += moveY;
if (this.grid) {
    this.pos.y = this.grid.resolveMoveY(this.pos.x, this.pos.y, half, moveY);
}
```

*Doors, Game Flags, Enemies & Interaction Tiles*

**levelDoors = [ ];** contains object data for mapping exits to entrances. When a door-tile is detected, **nextLevelIndex = door.target;**, **targetSpawn = door.spawn;**, **fading = true;** are set. Some door objects have conditions listed handled via the corresponding **let gameFlags = {};** variable, acting as a larger object that Booleans are assigned to.

```
// doors ------------------------------------------
let levelDoors = [
  // Fantasy Forest (L1) to Sunset Cliff (L2)
  { level: 0, x: 31, y: 13, target: 1, condition:"npc0_done", spawn: { x: 2, y: 0} },
  // Sunset Cliff (L2) to Desert Way (L3)
  { level: 1, x: 31, y: 12, target: 2, condition: null, spawn: { x: 1, y: 8 } },
  // Desert Way (L3) to Sunset Cliff (L2)
  { level: 2, x: 0, y: 5, target: 1, condition: null, spawn: { x: 30, y: 9} },
  // Desert Way (L3) to Lyn's House (L4)
  { level: 2, x: 20, y: 0, target: 3, condition: null, spawn: { x: 14, y: 17} },
  // Lyn's House (L4) to Desert Way (L3)
  { level: 3, x: 16, y: 17, target: 2, condition: null, spawn: { x: 20, y: 0} },
  // Desert Way (L3) to Byronic Asylum (L5)
  { level: 2, x: 31, y: 10, target: 4, condition: "has_level30_byronic_key", spawn: { x: 14, y: 17} },
  // Byronic Asylum (L5) to Desert Way (L3)
  { level: 4, x: 16, y: 17, target: 2, condition: null, spawn: {x: 31, y: 8}},
  // sunset cliff (L2) to fantasy forest (L1)
  { level: 1, x: 31, y: 14, target: 0, condition: null, spawn: {x: 1, y: 0}},
];
```

Enemies operate without physics, pathfinding, complex AI. **movingCombatTiles = [];** array contains static tile-data for enemy tiles (8) including; **x, y, drawX, drawY** coordinates separating logic, render positioning allowing smooth animations using **lerp()**, moving 8% closer each and running **updateCombatTiles();** each frame.

```
// Game update ------------------------
if (!paused) {
    if (!combat.active) {

        // check enemy count (FIXED FLAG NAME)
        if (currentLevel === 0 && gameFlags["enemy_defeated"] >= 10) {
            let npc = npcs.find(n => n.id === 0);

            if (npc && !npc.active) {
                npc.active = true;
                npc.inDialogue = false;
                npc.currentTarget = Math.max(npc.currentTarget, 1);
                gameFlags["npc0_cutscene_started"] = true;
                console.log("Activating NPC after 10 kills");
            }
```

```
movingCombatTiles = [];

for (let y = 0; y < grid.rows; y++) {
    for (let x = 0; x < grid.cols; x++) {
        if (grid.cells[y][x] === 8) {
            movingCombatTiles.push({
                x,
                y,
                drawX: x,
                drawY: y,
                targetX: x,
                targetY: y,
                moveTimer: 0
            });
        }
    }
}
```

```
function updateCombatTiles() {

    let key = `${tile.x},${tile.y}`;
    if (clearedCombatTiles.has(key)) continue;

    tile.moveTimer += deltaTime;

    // smooth movement
    tile.drawX = lerp(tile.drawX, tile.targetX, 0.08);
    tile.drawY = lerp(tile.drawY, tile.targetY, 0.08);
```

**grid.cells[tile.y][tile.x] = 8;** restores the tile following movement, occupying new space. **tile.moveTimer += deltaTime;** engages movement based on elapsed time of 1.2 seconds, not frame-rate. So variable frame-rates don't impact movement, and animations don't break or stack.



```
// enemies
this.enemySprite = null;

// combat
this.playerSP = 60;
this.playerMaxSP = 60;
this.spCost = 30;

// enemy sprites
this.pixieSprite = null;
this.slimeSprite = null;

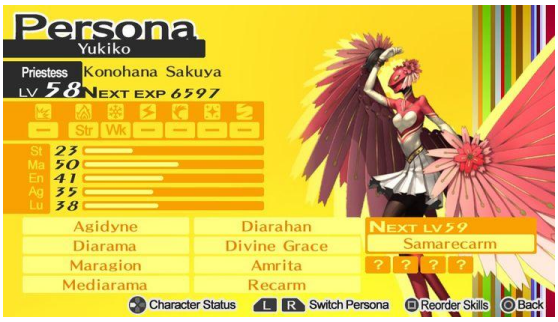
// player progression
this.playerLevel = 5;
this.playerXP = 0;
this.xpToNextLevel = 50;

// damage player
this.playerMinDamage = 9;
this.playerMaxDamageDealt = 16;

// stats
this.playerHP = 100;
this.playerMaxHP = 100;

this.enemyHP = 75;
this.enemyMaxHP = 75;
```







*Turn-based Combat System*

Levels, stats are the 'foundation of RPGs everywhere…'[6] Power output, level, progression of the player, enemies and bosses are handled via the combat system class containing a list of properties dictating each statistic. Simplifying this to a singular class fits this simplified one-on-one repeating turn-based system, making it easy to interchange enemies and override variables. **combat.start()** features the property **this.active = true;** which tells **draw()** to draw and update combat functions. **draw()** is split by the if else statement, **if (!combat.active) else combat.update();** so, the overworld map and menu's will or can be drawn whilst combat isn't active. When active, combat calls in **draw()** will be drawn and update respectively.

```
function draw() {
    if (!paused) {
        if (!combat.active) {

            // check enemy count (FIXED FLAG NAME)
            if (currentLevel === 0 && gameFlags["enemy_defeated"] >= 10) {
                let npc = npcs.find(n => n.id === 0);

                if (npc && !npc.active) {
                    npc.active = true;
                    npc.inDialogue = false;
                    npc.currentTarget = Math.max(npc.currentTarget, 1);
                    gameFlags["npc0_cutscene_started"] = true;
                    console.log("Activating NPC after 10 kills");
                }
            }

            if (!encounterFreeze) {
                updateCombatTiles();
            }

            checkEnemyPlayerCollision();

            updateDoorLocks();

            if (currentLevel === 0) {
                fill(0);
                rect(
                    grid.offsetX,
                    grid.offsetY,
                    grid.cols * grid.cellSize,
                    grid.rows * grid.cellSize
                );
            }

            currentInteractTile = null;
            interactTileInRange = null;

            if (!encounterFreeze) {
                player.update();
            }

            checkInteractTiles();

            for (let npc of npcs) {
                npc.update();
            }
```

Enemy stat property variables generate when **combat.start()** is called, allowing for higher-level areas to load higher property values, and random enemy selection. **this.turn;** equals either **player, enemy, waiting**. Creating a simple state machine following the combat sequence. **combat.update()**'s if statement and **setTimeout** function prevents the enemy AI attacking each frame.

```
class CombatSystem {
    update() {

        if (!this.active) return;

        if (this.turn === "enemy" && !this.enemyActing) {
            this.enemyActing = true;
            this.turn = "waiting";

            setTimeout(() => {
                this.enemyAttack();

                if (this.active) {
                    this.turn = "player";
                }
                this.enemyActing = false;
            }, 1500);
        }
    }
```

```
/**
 * new combat begins
 * @returns {void}
 */
start() {
    this.bossBattle = false;

    this.active = true;
    this.turn = "player";

    this.message = "A wild enemy appears!"

    this.skillMenuOpen = false;
    this.enemyBurning = false;
    this.burnTurns = floor(random(2, 6));

    // scale enemy based on current level
    this.enemyMaxHP = 75 + ((currentLevel + 1) * 20);
    this.enemyHP = this.enemyMaxHP;

    // damage enemy
    this.enemyMinDamage = 1 + (currentLevel + 1 * 8); // starts 9
    this.enemyMaxDamage = 10 + (currentLevel + 1 * 8); // starts 18

    if (random() < 0.5) {
        this.enemySprite = this.pixieSprite;
        this.enemyName = "Dark Pixie";
        this.enemyType = "pixie";
        this.enemyScale = this.pixieScale;
    } else {
        this.enemySprite = this.slimeSprite;
        this.enemyName = "Slime";
        this.enemyType = "slime";
        this.enemyScale = this.slimeScale;
    }
}
```

[6] https://thatsallgames.com/2019/02/05/the-anatomy-of-rpgs-part-1-stats/

```
class CombatSystem {
    /**
     * ui mouse click in combat
     * @param {*} mx
     * @param {*} my
     * @returns {void}
     */
    handleClick(mx, my) {
        if (!this.active || this.turn !== "player") return;

        if (this.skillMenuOpen) {
            for (let b of this.skillButtons) {
                if (
                    mx > b.x &&
                    mx < b.x + b.w &&
                    my > b.y &&
                    my < b.y + b.h
                ) {
                    this.useSkill(b.label);
                    return;
                }
            }
        }

        if (this.itemMenuOpen) {
            for (let b of this.itemButtons) {
                if (
                    mx > b.x &&
                    mx < b.x + b.w &&
                    my > b.y &&
                    my < b.y + b.h
                ) {
                    this.useItem(b.label);
                    return;
                }
            }
        }

        for (let b of this.buttons) {
            if (
                mx > b.x &&
                mx < b.x + b.w &&
                my > b.y &&
                my < b.y + b.h
            ) {
                this.handleAction(b.label);
            }
        }
    }
```

```
    // UI buttons
    this.buttons = [
        { label: "Attack", x: 375, y: height - 100, w: 180, h: 40 },
        { label: "Skill",  x: 575, y: height - 100, w: 180, h: 40 },
        { label: "Item",   x: 775, y: height - 100, w: 180, h: 40 },
        { label: "Run",    x: 975, y: height - 100, w: 180, h: 40 },
    ];
```

Turn-based combat is user-interfaced based interaction, the **combat.buttons = [ ];** properties array allows **combat.draw** to loop through selections using **for (let b of this.buttons)**, streamlining the process. **handleClick()** ensures the rhythm of turn-based remains via disabling the buttons array between turns.

**handleAction()** handles player animation states and properties depending on action. Attack initiates screen-animations at given values and randomly generates a damage value between parameters. Item or Skill trigger a secondary button array via **this.skillMenuOpen** / **this.itemMenuOpen** containing a corresponding for loop for respective choices. Following combat, **this.endCombat** returns the result to functions such as; **eventManager.clear**, removing corresponding combat tiles from the set. **awardXP(amount)** is also triggered, increasing XP variable randomly between parameters, displaying results and stat increases via a messages object handled via a **dialogueBox** class.

```
class CombatSystem {
    /*
    handleAction(action) {
        this.turn = "waiting";

        if (action === "Attack") {

            this.currentAttack = "attack";

            this.playerAttacking = true;

            // effects
            this.flashTime = 120;
            this.shakeTime = 150;
            this.shakeMag = 10;
            this.playerAttackJolt = 20;

            // random damage
            let dmg = floor(random(this.playerMinDamage, this.playerMaxDamageDealt + 1));
            this.enemyHP -= dmg;
            this.enemyHP = max(0, this.enemyHP);

            // messages
            this.message = "You dealt " + dmg + " damage!";

            setTimeout(() => {

                this.playerAttacking = false;
                this.currentAttack = null;

                if (this.enemyHP <= 0) {
                    this.endCombat("win");
                    return;
                }

                this.turn = "enemy";

            }, 400); // speed between attack and enemy turn

            } else if (action === "Skill") {
```

```
// dialoguebBox.js
/**
 * Dialogue box manager for text display and progression.
 */
class DialogueBox {
    constructor() {
        this.active = false;

        this.textLines = [];
        this.currentLine = 0;

        this.charIndex = 0;
        this.textSpeed = 40; // characters per second

        this.timer = 0;

        this.padding = 20;
        this.boxHeight = 150;

        this.onFinish = null;
    }

    /**
     * Start a dialogue sequence.
     * @param {string[]} lines
     * @returns {Promise<void>}
     */
    start(lines) {
        this.textLines = lines;
        this.currentLine = 0;
        this.charIndex = 0;
        this.timer = 0;
        this.active = true;

        return new Promise(resolve => {
            this.onFinish = resolve;
        });
    }

    /**
     * advance the dialogue each frame so it doesnt all show at once
     * @returns {void}
     */
    update() {
        if (!this.active) return;

        const dt = deltaTime / 1000;
        this.timer += dt * this.textSpeed;
```

```
        // random xp amount
        let xpGain = floor(random(15, 30));

        // Mark combat tile as cleared
        if (currentCombatTile) {
            clearedCombatTiles.add(`${currentCombatTile.x},${currentCombatTile.y}`);
            grid.cells[currentCombatTile.y][currentCombatTile.x] = 0;
            eventManager.clear(currentCombatTile.x, currentCombatTile.y)
        }
```

The property variables dictating the player, enemies, stats in combat are handled via the **combatSystem** class. Easily overwritten with new values in the case of the Boss encounter. **startBossBattle()** stores Commandment's unique behaviours, drawn **if (bossEncounter)** in sketch **draw()** is true. Sprite and corresponding animations are interchanged via this method. This only functions due to the one-on-one functionality of my combat system. Various property values attached to enemy types dictate their statistics and individual scale. **NPC.js** also draws different sprites depending on need based on the NPC object array variable specifying path, interaction, conditions and sprite type using **let spriteToDraw =**

```
// combat system ----------
let combat;
let clearedCombatTiles = new Set();
let currentCombatTile = null;
```

```
4   class CombatSystem {
1179    /**
1180     * begin the battle with the commandment
1181     * @returns {void}
1182     */
1183    startBossBattle() {
1184        this.active = true;
1185        this.bossBattle = true;
1186        this.turn = "player";
1187
1188        this.message = "The Commandment, Demon of the Abyss, stands before you...
1189
1190        this.skillMenuOpen = false;
1191        this.itemMenuOpen = false;
1192
1193        this.enemyName = "The Commandment";
1194        this.enemySprite = this.bossSprite;
1195        this.enemyType = "boss";
1196        this.enemyScale = 820;
1197
1198        this.enemyMaxHP = 650 + (this.playerLevel * 5);
```

```
// NPC setup ---------------------------------------
let levelNPCs = [
[
{
    path: [
        {x:32,y:13},
        {x:25,y:12, dialogue:["???: HEY YOU THERE!", "???: HOW LONG YOU GONNA MESS ABOUT HERE FOR!", "???: FOLLOW ME NEW BLOOD!", "Follow him out of the forest"]},
        {x:28,y:13},
        {x:32,y:13}
    ],
    speed:200,
    size:48,
    interaction:false,
    loop:false
}
],
[
{
    path: [
        {x:2,y:0},
        {x:2,y:5},
        {x:10,y:5},
        {x:10, y: 6, dialogue:["???: I'm Warden Marx, you are?",
            "You: uh I'm Hiro",
            "Warden Marx: Hiro... Hiro the Hero? Seriously... well whatever",
            "Warden Marx: Just through here is the gate to Bryonic Asylum",
            "Warden Marx: I have somewhere to be but I expect you'll figure it out, there will be some keys you'll have to find as you go...",
            "Warden Marx: Hardly a challenge for you, you are an adventurer after all, going on those so-called quests...",
            "Warden Marx: ...I won't hold your hand, but I will say this...",
            "You: Hmm?",
            "Warden Marx: Heroes come in and out of this realm everyday, barely any get back to their worlds before it's too late.",
            "You: Where even am I?!",
            "Warden Marx: This is a realm between realms, a prison for those chosen to be brave, a purgatory for those who the council deem unfit as a hero.",
            "You: Helpful... *sigh*",
            "Warden Marx: If you want out... get stronger...",
            "Find Bryonic Asylum and get stronger"]},
            {x:10, y:10},
            {x:15, y:12},
            {x:100, y:12},
    ],
    speed:220,
    size:48,
    interaction:true,
    loop:false
}
```

*Reflective Section*

```
/**
 * ui mouse click in combat
 * @param {*} mx
 * @param {*} my
 * @returns {void}
 */
handleClick(mx, my) {
    if (!this.active || this.turn !== "player") return;

    if (this.skillMenuOpen) {
        for (let b of this.skillButtons) {
            if (
                mx > b.x &&
                mx < b.x + b.w &&
                my > b.y &&
                my < b.y + b.h
            ) {
                this.useSkill(b.label);
                return;
            }
        }
    }
```

Many **.js** files are overcomplicated making them difficult to track. My combat class being responsible for all properties, functions have benefits, but as the project grew I would repeatedly add new overriding 'glue code.'[7] Global state variables; **gameFlags, currentLevel, combat.active, dialogue.active, paused, showInventory, encounterFreeze, bossEncounter** ensure large systems; combat, NPC, grid can coordinate. **combatSystem.js** has extra checks such as **if (!this.active) return** and **if (this.turn !== "player") return;** which balance the flow of combat, logic, items, skills, enemies, stats.

Initially unaware of turn-based's scale, believing one class would be sufficient as code like player class, was pre-established. This was due to my knowledge of RPG-style gameplay outweighing my programming experience. Whenever gameplay, mechanics alter like between traversable open worlds and turn-based interfaces, the previous code is being replaced, overlaying the interactive experience. Therefore, assets have separate functionality, properties and mechanics. Additionally, despite my efforts, I was unable to implement character animations via a sprite sheet, with more time this would've improved the smoothness of player and enemy animation.

```
function draw() {
    background(0);

    // menu condition -----------------
    if (!gameStarted) {
        menu.update();
        menu.draw();

        if (showOptionsMenu) {
            drawOptionsMenu();
        }

        return;

    }
```

In **draw()** a lot of mode-checking is implemented such as **if (!gameStarted), (!paused), (!combat.active).** Each statement acts as glue code, determining which systems allowed to run, rather than calling classes. With my knowledge now, combat would be separated via classes for enemy type, stats, rewards, player functionality. Allowing more manageable code, also easier to debug. Combat functionality would also be less restricted, resulting in more player and enemy types and abilities. This would remove separate functions for using items in combat versus the party menu, as this would be a singular function call. Each sprite wouldn't need interchanging individual scale properties; each would be handled via its own object or class. UI buttons wouldn't need reprogramming for every instance. Similar tilemap functionalities (door, interactables) both detect collision, check conditions, trigger events and set flags, this could be a generalised class system.

```
// npc0 cutscene - marx
if (npc0 && npc0.active && !npc0.requiresInteraction && (npc0.isMoving || npc0.inDialogue)) {
    if (npcCamX === null || npcCamY === null) {
        npcCamX = npc0.pos.x;
        npcCamY = npc0.pos.y;
    }
```

```
// reset camera after cutscene
if (!cutsceneActive) {
    npcCamX = null;
    npcCamY = null;
}

// apply camera smoothing
gameCam.update(targetX, targetY, targetZoom);

if (escapeWindowTimer > 0) {
    escapeWindowTimer -= deltaTime;
}

// Game update ------------------------
if (!paused) {
    if (!combat.active) {
```

```
// doors ------------------------------------------------
let levelDoors = [
    // Fantasy Forest (L1) to Sunset Cliff (L2)
    { level: 0, x: 31, y: 13, target: 1, condition:"npc0_done", spawn: { x: 2, y: 0} },
    // Sunset Cliff (L2) to Desert Way (L3)
    { level: 1, x: 31, y: 12, target: 2, condition: null, spawn: { x: 1, y: 8 } },
    // Desert Way (L3) to Sunset Cliff (L2)
    { level: 2, x: 0, y: 5, target: 1, condition: null, spawn: { x: 30, y: 9} },
    // Desert Way (L3) to Lyn's House (L4)
    { level: 2, x: 20, y: 0, target: 3, condition: null, spawn: { x: 14, y: 17} },
    // Lyn's House (L4) to Desert Way (L3)
    { level: 3, x: 16, y: 17, target: 2, condition: null, spawn: { x: 20, y: 0} },
    // Desert Way (L3) to Byronic Asylum (L5)
    { level: 2, x: 31, y: 10, target: 4, condition: "has_level30_byronic_key", spawn: { x: 14, y: 17} },
    // Byronic Asylum (L5) to Desert Way (L3)
    { level: 4, x: 16, y: 17, target: 2, condition: null, spawn: {x: 31, y: 8}},
    // sunset cliff (L2) to fantasy forest (L1)
    { level: 1, x: 31, y: 14, target: 0, condition: null, spawn: {x: 1, y: 0}},
];
```

```
// interactable tiles ------------
let interactTileInRange = null;

let currentInteractTile = null;

let interactTileDialogue = {
    "3,4,9": { // level + x + y
        lines: ["There's a chest here!", "You opened the chest and found a potion!"], // level 4 (3) chest
        item: "potion",
        amount: 1,
        once: true
    },
    "3,9,10": {
        lines: ["It's someone's bed...",
                "You: Man am I tired...",
        ],
    },
    "3,14,8": {
        lines: ["It's a fireplace, looks like it was recently lit..."],
    },
    "3,19,8": {
        lines: ["It's a grandfather clock, the hands are stuck at 3 AM...",
                "You: But the sun's still out...",
                "You: It could be that time just doesn't pass here...",
                "You: ..... Nah"
        ],
    },
    "3,14,13": {
        lines: ["It's a table...", "You should have a closer look..."],
    },
    "3,20,13": {
        lines: ["It's an old book...",
                "You begin to read the open page...",
                "In the beginning, there was a demon... The realm of Symphonia is one of many found across the
                "The Deixis developed advanced arcane scriptures, through the control of mana, they were able to
                "Whilst great in concept, this rendered the last remaining Deixis unable to protect themselves
                "A faraway realm at this time was facing it's greatest peril, unbeknownst to the Deixis. The Hec
```

[7] https://en.wikipedia.org/wiki/Glue_code

In conclusion, where I to remake it under these conditions, I'd reduce the scale, majority of my time being used to ensure every mechanic I wanted was present, the story was told and with stunning SNES-style pixel graphics, rather than ensuring the existing code was well-organised. As a result, a lot of patchwork was needed, creating multiple functions, variables responsible for similar, if not the same functionality. This made navigation harder, bugs to arise more often and debugging more convoluted, often with a domino effect. Despite this, my code is interconnected, not isolated, focusing on impressive architecture and understanding of event flow and system interaction.

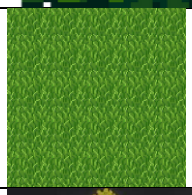
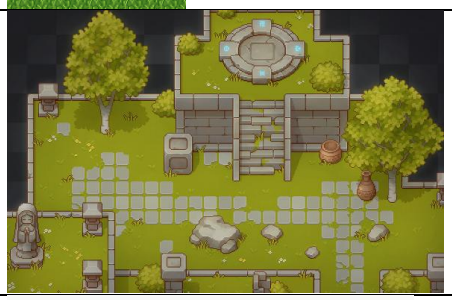
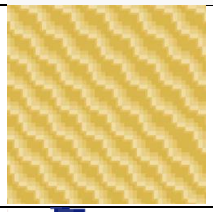


# Bibliography


1. https://en.wikipedia.org/wiki/Final_Fantasy_II (10/05/26) (Wikipedia, Final Fantasy II, Squaresoft, 1988)
2. https://nintendo.fandom.com/wiki/Golden_Sun_(video_game) (10/05/26) (Nintendo Fandom, Golden Sun, Camelot, 2001)
3. https://purenintendo.com/retro-review-chrono-trigger-snes/ (10/05/26) (PureNintendo, Chrono Trigger, Squaresoft, 1995)
4. https://gameopedia.com/blogs/an-introduction-to-open-world-games (10/05/25) (Gameopedia, Vaibhav Manocha, An introduction to open world games)
5. https://www.serebii.net/pokearth/kanto/3rd/mt.moon.shtml (Serebii, Mt Moon, Pokearth, Pokémon Red and Blue, 1996)
6. https://thatsallgames.com/2019/02/05/the-anatomy-of-rpgs-part-1-stats/ (11/05/26) (That's All Games, tagfitty, Anatomy of RPGs Part 1, 2019)
7. https://en.wikipedia.org/wiki/Glue_code (11/05/26) (Wikipedia, Glue Code)


1648 Words

# Appendix

| Asset | URL | Image |
|---|---|---|
| **Purple Fantasy Tree** | https://babanagi.itch.io/10-free-pixel-art-trees |  |
| **Green Bush** | https://gipp-0.itch.io/greenbushes |  |
| **Grass 1** | https://www.deviantart.com/bulwarken/art/Pixel-Art-Grass-692409685 |  |
| **Stone Asset Pack** | https://cainos.itch.io/pixel-art-top-down-basic |  |
| **Stone Wall** | https://pngtree.com/freepng/cartoon-pixel-art-of-a-stone-wall-for-game-interface-design_21390570.html |  |
| **Sand** | https://www.shutterstock.com/image-vector/sand-texture-pixel-art-vector-picture-1827039506 |  |
| **Shadow Icon for Enemies** | https://beaglebits.itch.io/darkflame |  |
| **RPG Hero** | https://mirror.chromaso.net/thread/1073754161 |  |

| | | |
|---|---|---|
| **RPG Hero (ITEM)** | https://ktkm.net/product/izakenzan/ |  |
| **RPG Hero (BACK)** | https://en.ac-illust.com/clip-art/23494156/pixel-art-hero-standing-up |  |
| **RPG Hero (SIDE)** | https://www.stidea.jp/%E9%9B%91%E8%AA%8C-%E6%96%B0%E3%81%97%E3%81%84%E5%B0%B1%E6%B4%BB |  |
| **Interior Pack** | https://gif-superretroworld.itch.io/interior-pack |  |
| **The Comm-andment** | https://pxlart.com/p/sbNibc1xdC3tN849LXn0/high-demon-lord-pixel-art-majestic |  |
| **Battle Back ground** | https://stock.adobe.com/uk/search?k=pixel+forest<br>And below |  |

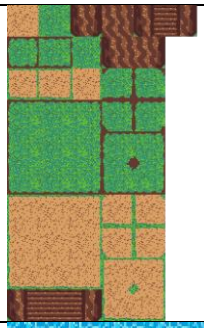
https://www.google.com/search?sca_esv=511efc3390dbf023&lns_surface=44&hl=en-GB&sxsrf=ANbL-n5Q_Cx5K2wJ4Qk-ZxJVkTQy4KyQ8q:1778511648483&udm=44&vsrid=CKeNoeKiiZWCkAEQAhgBIiQyZDUyZWQ4OS02ZWNlLTRhNDMtYWE5My04Mjc1MGRmODFlOGEyeylCd2UoG0JzCi5sZmUtZHVtbXk6OTk5YjU1NWMtZDYzMy00ZWRlLWE4NjgtOWFlZmRjOTJhOGVhEkEKPy9ibnMvd2UvYm9yZy93ZS9ibnMvbGVucy1mcm9udGVuZC1hcGkvcHJvZC5sZW5zLWZyb250ZW5kLWFwaS83Nzj3t6KYwLGUAw&vsint=CAIqDAoCCAcSAggKGAEgATojChYNAAAAPxUAAAA_HQAAgD8lAACAPzABEJUFGOgCJQAAgD8&lns_mode=un&source=lns.web.cntpubb&vsdim=661,360&gsessionid=dYKzcK_-yGhyKfGtTprALN6TI-lgpD_SX77GcwVmw1DJTgq8tZPWeQ&lsessionid=AMU1uwSbTAwRlOtSgq_6zoe5Syj3seHwYn_40Cknu1kpe3qW-iUKdA&q=&sa=X&ved=2ahUKEwirrKWYwLGUAxXBMfsDHRrCHHMQs6gLegQIDhAB&biw=958&bih=394&dpr=1#sv=CAMSVhoyKhBlLUZMVzBDV1dzRHluTE9NMg5GTFcwQ1dXc0R5bkxPTToOX2pFN0JlX3hlUVUyd00qBCocCgZtb3NhaWMSEGUtRkxXMENXV3NEeW5MT00YADABGAcghs6m0wJKCBABGAEgASgB

| | | |
|---|---|---|
| **Menu Screen** | **ChatGTP – prompt = “**Turn this screenshot into a 90’s style fantasy JRPG anime style cover art.” |  |
| **Inferno** | https://en.ac-illust.com/clip-art/24091089/pixel-art-hero-flame-sword |  |

| | | |
|---|---|---|
| **Electro** | https://www.insurtechlab.net/hanger-flight/ |  |
| **Lyn** | http://workingnews117.com/blog-entry-18612.html |  |
| **Lyn Side** | https://www.ac-illust.com/search/ris.php?object_id=3e8161c38290e8922ac5dfdb55abf6f7&id=27094463 |  |
| **Lyn Back** | **ChatGTP = PROMPT = "**Show what this image would look like from behind (Lyn) whilst keeping the same pixelated arty style and pose." |  |
| **RPG Hero (Attack)** | Below |  |

https://www.google.com/search?sca_esv=511efc3390dbf023&lns_surface=44&hl=en-GB&sxsrf=ANbL-n4OaeCmnu4selw7PW1PI9p6BOwhow:1778512208074&udm=44&vsrid=CPmbqbryxoGvTBACGAEiJGVlNjqwZjM4LWIzZWYtNGVhNS05ZDcwLTE3Y2U0YTdlYWIyMjJ7IgJ3ZSgpQnMKLmxmZS1kdW1teTpkNzMzODA4Ni01MzQzLTQ3YjktYThlMS01NmY4MjBiOTQ2OWMSQQo_L2Jucy93ZS9ib3JnL3dlL2Jucy9sZW5zLWZyb250ZW5kLWFwaS9wcm9kLmxlbnMtZnJvbnRlbmQtYXBpLzQ2OOeWjaPCsZQD&vsint=CAIqDAoCCAcSAggKGAEgATojChYNAAAAPxUAAAA_HQAAgD8IAACAPzABEPkGGO8GJQAAgD8&lns_mode=un&source=lns.web.cntpubb&vsdim=889,879&gsessionid=2DgZ3dKKN4StnsqCWhhQfp_JBskeV6PM7wfBnfnTt5QAa92a_q9DZQ&lsessionid=k5yvBGG6PFIr08a224O3pF9Y_f9BdB-RTCBAVB4YX10s9FM1-eb69A&q=&sa=X&ved=2ahUKEwjAiZCjwrGUAxUyT6QEHXbdCncQs6qLegQIEhAB&biw=958&bih=394&dpr=1#sv=CAMSVhoyKhBlLTBOb2NIM2M4REZDMnFNMg4wTm9jSDNjOERGQzJxTToOb19ranFqendRbFhfOU0qBCocCgZtb3NhaWMSEGUtME5vY0gzYzhERkMycU0YADABGAcqgaKyuA9KCBABGAEgASgB

| | | |
|---|---|---|
| **Slime** | https://retrogametalk.com/threads/help-running-psp-iso-on-psp-2000-model.3694/ |  |
| **Dark Pixie** | https://www.sprite-ai.art/gallery/fairy-knight-in-motion-3fcf |  |
| **Cactus** | https://toppng.com/free-image/ixel-cactus-png-clipart-stock-cactus-pixel-art-PNG-free-PNG-Images_442750 |  |

| | | |
|---|---|---|
| **Top-down Stairs** | https://broccolibusiness.itch.io/tiles/devlog/116906/top-down-stairs |  |
| **Water** | https://fonts.google.com/specimen/Cinzel+Decorative?preview.script=Latn |  |
| **Sky** | https://stock.adobe.com/uk/search?k=16+bit&asset_id=911689184 |  |
| **Menu Music** | https://pixabay.com/music/epic-classical-epic-508009/ | |
| **Battle Music** | https://pixabay.com/music/epic-classical-epic-glory-orchestral-loop-517470/ | |
| **Overworld Music** | https://pixabay.com/music/modern-classical-enchanted-fantasy-classical-music-516862/ | |
| **FONT: Cinzel Decorative** | https://fonts.google.com/specimen/Cinzel+Decorative?preview.script=Latn | |
| **Menu Music 2** | https://pixabay.com/music/video-games-pixel-fantasia-355123/ | |
| **Level 5** | https://pixabay.com/music/video-games-8-bit-space-123218/ | |
| **Battle Music 2** | https://pixabay.com/music/video-games-80s-video-game-battle-chiptune-216255/ | |
| **Level 1** | https://pixabay.com/music/synthwave-80s-retro-gaming-through-the-dark-forest-411472/ | |
| **Level 4** | https://pixabay.com/music/synthwave-80s-retro-gaming-through-the-dark-forest-411472/ | |
| **Level 3** | https://pixabay.com/music/video-games-retro-egyptian-theme-187380/ | |
| **Level 2** | https://pixabay.com/music/electronic-game-game-music-508018/ | |