Unity を用いた 2D ゲーム制作

谷田　裕太　出﨑　琉偉
武入　晴翔　濵戸　大聖

１．研究概要

Unity を用いたゲームの共同開発を通じてプログラミング技術の向上や、共同開発を行う上で重要な事を知ることを目標に制作をした。RPG の機能として敵との戦闘、アイテムの管理や売買などのシステムを実装した。

２．研究の具体的内容

（１）ゲーム概要

今回制作したゲームは、ダンジョンを攻略していき、物語を進めていくものである。ストーリーも作成している。

操作方法は表１の通りである。

表１　操作方法

動作	操作キー
プレイヤーの移動	WASD キーと十字キーで移動 Shift キーを押すことでダッシュで移動
武器の切り替え	１、２キーを押すことで剣と弓を切り替え
攻撃	Space キー
アイテム管理画面を開く	E キー

敵を倒しているとレベルが上がっていき、スキルポイントが溜まるようになっている。そのポイントを使うことでプレイヤーの能力値であるステータスを強化することができ、戦闘を有利に進めていくことができる。

(a) 作成スケジュール

制作を行うにあたって、スケジュールを表２のように機能ごとに計画した。機能ごとに担当を決めて作業を進めたが、計画を予定通り進めることができなかった。

表２　作成スケジュール

４月	プレイヤー動作制作 アイテム管理機能制作 プレイヤー・アイテムデザイン制作
５月	プレイヤーステータス制作 アイテム管理機能制作 敵・武器、背景デザイン制作
６月	アイテムドロップ、戦闘システム制作 武器、敵のアニメーション制作
７月	ショップ機能制作 ダンジョンシステム制作 ストーリー、ボスの制作
８月	システム UI、ダンジョン制作 ストーリー、ボスの制作
９月	セーブ機能、ストーリー制作 BGM 制作
10 月	デバッグ、動作テスト、BGM 制作
11 月	デバッグ、動作テスト、最終テスト 岡工祭展示
12 月	資料作成
１月	課題研究発表準備

(b) 開発環境

ゲーム制作で使用した開発環境は次のとおりである。

・Unity6000.0.47f1
・Microsoft Visual Studio Community 2022 (64 ビット) - CurrentVersion 17.14.20
・EDGE Ver.1.29b（オブジェクト素材制作）

・Cubase LE 14（BGM 制作）
・GitHub（データ共有）

〇使用した Unity・C#の機能
・Tag
Unity でゲーム内のオブジェクトを種類分けする機能。プログラムで Tag ごとに判定をすることができる。主に、オブジェクトの当たり判定で、何と接触したかを判定するために使用した。

・Math.Max()
指定された引数の中から最大値を返すメソッド。

・Math.Min()
指定された引数の中から最小値を返すメソッド。

・NavMesh2D
キャラクターが動く範囲の設定をすることができる。敵キャラクターの移動範囲を指定するために使用した。

・SamplePosition
指定された座標に最も近い NavMesh の座標を返すメソッド。

・Collider
ゲームオブジェクトに当たり判定をつけることができる。例として、図 1 の絵のまわりにある白い四角い線である。

図 1　当たり判定

・OnCollisionEnter2D
Collider 同士がぶつかった時に呼び出されるイベントハンドラ。主に、戦闘システムの、武器と敵キャラクターの衝突判定に使用した。

・ScriptableObject
テキスト、数値、画像などの複数のデータをまとめて管理することができるツール。主にアイテムのデータを管理するために使用した。

・Prefab
オブジェクトを再利用可能にできる機能。同じオブジェクトを複数使用するときに Prefab を複製して使用している。

・アニメーション
イラストなどを少しずつ変化させながら連続的に見せることで、動いているかのように見せる表現方法を指す。主に Player や敵に動きをつけるために使用した。

・タイルマップ
タイル（小さな画像素材）を格子状に並べて、2D ゲームの背景やステージ、ワールドを効率的に作成・管理するための機能。主にゲームの背景の作成に利用した。

（c）制作の役割分担

ゲーム制作には音楽・デザイン・プログラミングなどが必要になってくる。それぞれ役割を振り分けて分担して作業を行った。分担した内容は表３のとおりである。

表３　役割分担

谷田	開発環境構築 アイテム管理機能制作 戦利品・ショップ機能制作 システム UI 制作 セーブ・ロード機能制作
出﨑	プレイヤー操作制作 ステータス機能制作 戦闘システム制作 ダンジョン制作 BGM 制作
武入	ストーリー制作 背景・アイテムデザイン制作
濵戸	プレイヤーデザイン制作 敵デザイン・アニメーション制作

（２）ゲームの機能

（a）キャラクターの動作

○　プレイヤーの動作

プレイヤーが走っている間、減少するスタミナ機能をつけた。スタミナは時間経過で回復をすることができる。

スタミナの計算を行う際に負の値や最大値を超えないようにする必要があったため Math.Max()、Math.Min()を用いることで対策をした。

プレイヤーが歩いているのか、走っているのか、何もしていないのか(待機状態)の３つのパターンと上下左右の４つのパターン、計 12 パターンに分けてアニメーションを切り替えるようにした。

○　敵キャラクターの動作

Unity の Navmesh というものを用いてランダムな移動とプレイヤーへの追従を実現した。実装する際、Navmesh の設定外にキャラクターが移動しようとするとエラーになり動かなくなるため Navmesh の標準機能である SamplePosition を用い一番近くの Navmesh を認識させることで解決した。

図２は Navmesh の設定画面である。青く光っている所がキャラクターの移動可能範囲であり、それ以外の所は、移動できなくしている。

図２　Navmesh 設定画面

○ボスキャラクターの動作

ボスキャラクターは２枚の絵を回転させることで、まるで踊っているかのようなアニメーションを作った。（図３）

特定の座標につくまで２枚の絵を交互に表示し続けながら角度をつけ続け、特定の座標につくと、それぞれの絵を反転させ、また交互に表示しながら回転させる、というようにした。（図４）（図５）

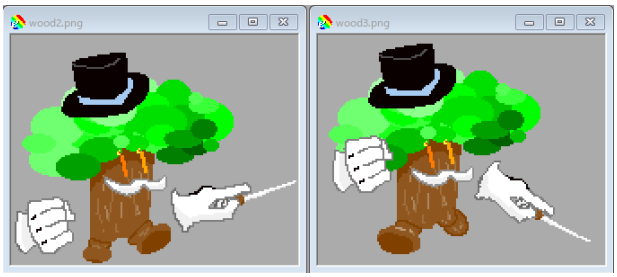

図３　ボスキャラクターのアニメーション

```
using UnityEngine;
using System.Collections;

public class BossMove : MonoBehaviour
{
    [Header("移動設定")]
    public float speed = 2f;              // 移動速度
    public Vector3 leftPos = new Vector3(-6f, 17f, 0f);   // 左座標
    public Vector3 rightPos = new Vector3(8f, 17f, 0f);   // 右座標
    private Vector3 targetPos;            // 現在の目標座標

    [Header("回転設定")]
    public float rotateTime = 0.15f;     // 到達時の180° 回転にかかる時間
    public float spinInterval = 8f / 60f;   // 常時回転の開始間隔（8フレーム）
    public float spinRotateTime = 0.3f;     // 常時回転の180° 回転にかかる時間（以前の0.15→2倍）

    private bool isRotating = false;     // 到達時回転フラグ
    private bool isSpinning = false;     // 常時回転フラグ
    private float spinTimer = 0f;

    void Start()
    {
        transform.position = leftPos;
        targetPos = rightPos;  // 最初は右へ
    }

    void Update()
    {
        // ===== 座標移動 =====
        transform.position = Vector3.MoveTowards(
            transform.position,
            targetPos,
            speed * Time.deltaTime
        );

        // 目標到達時に180° 回転して方向切替
        if (!isRotating && Vector3.Distance(transform.position, targetPos) < 0.01f)
        {
            StartCoroutine(RotateAndChangeTarget());
        }

        // ===== 常時回転 =====
        spinTimer += Time.deltaTime;
        if (!isSpinning && spinTimer >= spinInterval)
        {
            spinTimer = 0f;
            StartCoroutine(VisualSpinOnly());
        }
    }
```

図４　座標の移動

```
// 到達時の180° 回転 + 目標切替
IEnumerator RotateAndChangeTarget()
{
    isRotating = true;

    float timer = 0f;
    float startY = transform.eulerAngles.y;
    float endY = startY + 180f;

    while (timer < rotateTime)
    {
        timer += Time.deltaTime;
        float t = timer / rotateTime;
        float yRot = Mathf.Lerp(startY, endY, t);
        transform.eulerAngles = new Vector3(0, yRot, 0);
        yield return null;
    }

    transform.eulerAngles = new Vector3(0, endY, 0);

    // 目標座標切替
    targetPos = (targetPos == rightPos) ? leftPos : rightPos;

    isRotating = false;
}

// 移動中の常時回転（方向は変えない）
IEnumerator VisualSpinOnly()
{
    isSpinning = true;

    float timer = 0f;
    float startY = transform.eulerAngles.y;
    float endY = startY + 180f;

    while (timer < spinRotateTime)
    {
        timer += Time.deltaTime;
        float t = timer / spinRotateTime;
        float yRot = Mathf.Lerp(startY, endY, t);
        transform.eulerAngles = new Vector3(0, yRot, 0);
        yield return null;
    }

    transform.eulerAngles = new Vector3(0, endY, 0);
    isSpinning = false;
}
```

図５　回転のスクリプト

（b）戦闘システム

戦闘は敵の攻撃を避けながら攻撃をする、アクション方式を採用した。

敵を倒した後の動作は図６のようになっている。経験値を獲得するとレベルが上がり、後述するステータスを強化するために必要なスキルポイントを獲得する。

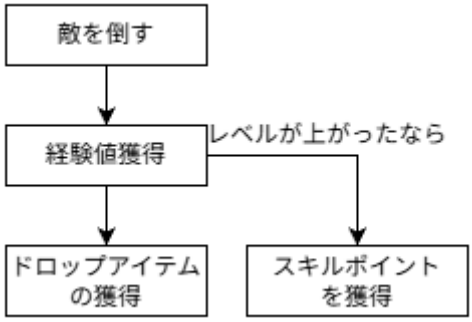

図６　敵を倒した後の動作

○武器の種類

武器は剣と弓の２種類があり、それぞれの特徴は表４の通りである。

1 で剣を、2 で弓を使用する。

表４　武器の特徴

武器	特徴
剣	近距離での範囲攻撃ができる 攻撃力は弓より高い
弓	遠距離での単体攻撃ができる

図７，８のような UI を作ることでどちらを使用しているのかをわかりやすく示した。

図７　選択武器の UI（剣を選択）

図８　選択武器の UI（弓を選択）

○戦闘におけるキャラクターの動作

・プレイヤー

攻撃のタイミングに合わせてプレイヤーが移動できないようにすることで一方的な攻撃を不可能にした。また、連続的な接触による即死を防ぐために攻撃を受けると一定の無敵時間を設けることで大量のダメージを受けることがないようにした。

ダメージを受けた事が分かるようにプレイヤーの色を変更した。

ダンジョンのテーマごとに表５で示す状態異常を実装した。

表５　状態異常の効果

状態	効果
毒	最大体力の 1%のダメージを 10 秒間受け続け、移動速度が低下する。
麻痺	一定時間プレイヤーの動作を完全に停止する。
火傷	火傷状態は 10 の固定ダメージを 5 秒間受け続ける。
凍結	凍結状態は麻痺の動作停止と 6 の固定ダメージを 5 秒間受け続ける。

状態異常は全て時間経過によって解除される。

図９のように HP バーの色を変更することでどの状態異常になっているのかがわかるようにした。

図９　状態異常（毒）

・敵キャラクター

前述したキャラクター動作に加えてプレイヤーからのダメージを受けると少しの間止まるようにした。また、ダメージを受けた事が分かるようにキャラクターの色を変更した。

○ステータス

ステータスは体力、攻撃力、防御力、クリティカル確率、スタミナの５つを実装した。スキルポイントを使用することで好きにステータスの強化をできるようにした。

図 10 はステータス画面である。ここから自分のステータスの確認、強化が行える。強化は５つ全て行えるようにした。

左右のボタンを使う事でどのくらいの強化を行うのかを決める事ができ、右下にある、確定ボタンを押すと強化することができる。また、ボタンが使用できるかどうかを視覚的にわかるように使用できないボタンは暗く表示するようにした。

クリティカル確率は float 型の変数を使用しており、そのまま表示すると小数第２位以下の値があると見栄えが悪かったので元の値を小数第１位で丸めることで見栄えをよくした。

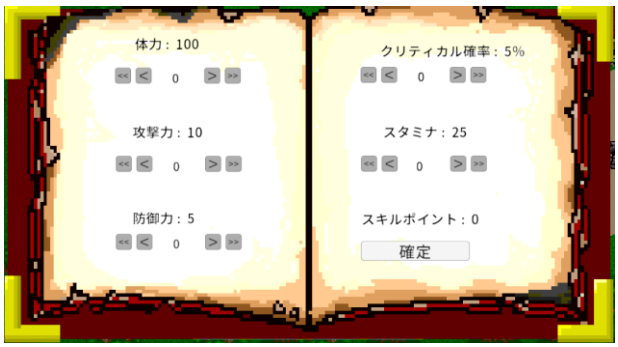

図 10　ステータス画面

○ダメージテキスト

　ダメージを受けた時にどれくらいのダメージを受けたのかが分かるように図 11 のようにダメージテキストをつけた。

図 11　ダメージテキスト

○当たり判定

　マップなどの当たり判定と衝突した時にもプログラムが呼びされてしまったので、tag を使い、図 12 のような考え方をすることで解決した。

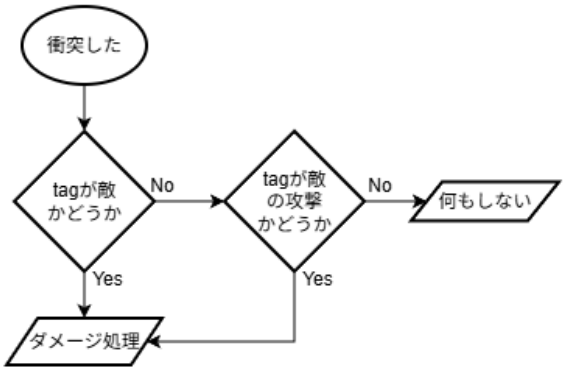

図 12　当たり判定のフローチャート

○ダメージ計算

　攻撃のダメージが一定だと変化がないので変化をつけるために乱数を用いてダメージを変化させた。

　攻撃にクリティカルを発生させるために攻撃の度に確率での抽選を採用した。

　Unity の機能に確率のものがなかったため、図 13 のような考え方で Probability というプログラムを作った。

　クリティカルが発生した時は、普段の 2 倍のダメージが発生するようにした。

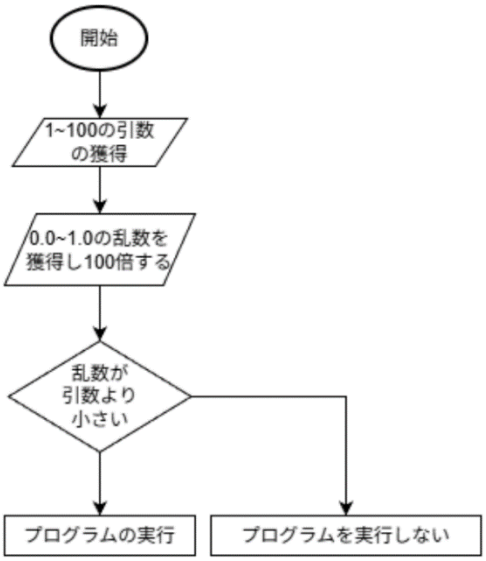

図 13　確率のフローチャート

　ダメージ計算で乱数を使って計算したものをそのまま使うと小数点がでてしまうので値

を float 型から int 型に変更し、整数として扱うようにした。計算後、１を下回ることがあったため、ダメージを最低でも１とするように Math.Max()を使った。

(b) ダンジョン

ダンジョンは 10 階層モデルで１～３階をジャングル、４～６階層を火山、６～９階層を氷山というテーマで開発を行い、最後の 10 階層をラスボスとした。

ダンジョンの内容は入る度に内容が変わる不思議のダンジョン方式にしようとしたが、技術力不足でできなかったため、事前に用意しているものをランダムに使用する形式にした。また、各奇数番号の層には中ボスが存在し、中ボスを倒すことで次の層、ボスの層からの挑戦が可能になるようにした。例として 3 層のボスを倒すと 4 層からの挑戦が可能になる。

ダンジョンに入る前には図 14 のような確認テキストを出して誤動作を防止した。

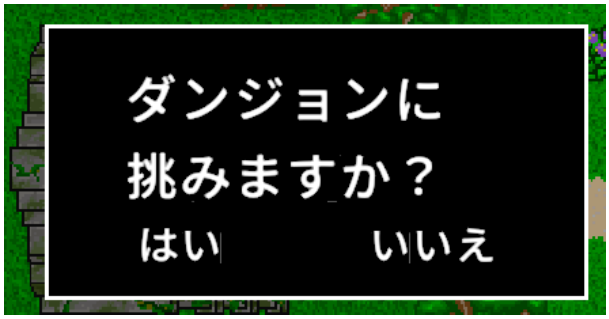

図 14　確認テキスト

用意したクリア条件は以下の３つである。

・一定数の敵の撃破
・鍵を見つけ出し脱出
・迫りくる敵を倒すウェーブ

図 15 のように入場時に何をするのかが分かるようにテキストを表示するようにした。また、図 16 のようにどのくらいの進捗なのかがわかるように、テキストをダンジョンに入った時にのみ出るようにした。入ったダンジョンの内容でテキストの内容が変更される。

図 15　入場時のテキスト UI

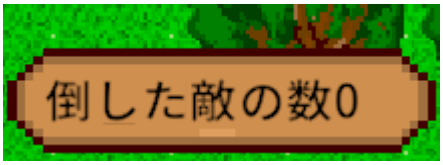

図 16　倒した敵の数のテキスト UI

ダンジョンの宝箱のようにアクションが必要なものは図 17 のようなテキストを出すことで操作を視覚的に分かるようにした。また、宝箱は一回獲得すると、次はテキストの表示も行わないようにした。

図 17　説明テキスト

ランダムでの移動を実現するために前述した Probability を用い実現した。その際、全てに引っかからないことがあることを想定して図 18 のように if 文の最後のように絶対に一つは入るようにプログラムの設計をした。

```
if (Probability.probability(30))
{
    loadScene = loadScene3;
    x = x3;
    y = y3;
}
else if (Probability.probability(30))
{
    loadScene = loadScene2;
    x = x2;
    y = y2;
}
else
{
    loadScene = loadScene1;
    x = x1;
    y = y1;
}
```

図 18　シーン移動のプログラム

場面の切り替え後に出入口の前にキャラクターがいないと不自然なので移動するときに座標を決めておき、決められた座標へ移動させることで不自然さを解消した。また、移動に時間がかかり、キャラクターがシーン移動の前に移動後の座標に飛ばされ不自然だったのでロード画面（図 19）を追加する事で不自然さを解消した。

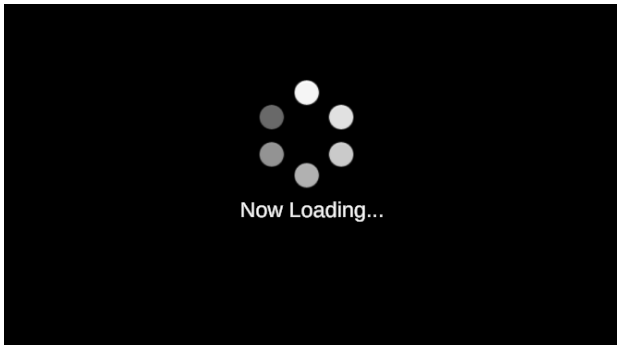

図 19　ロード画面

また、シーンの設定を行うにあたり、テキスト形式ではタイプミスや名前の変更に対応できないので図 20 のようにインスペクタービューからドラッグアンドドロップで変更できるようにした。

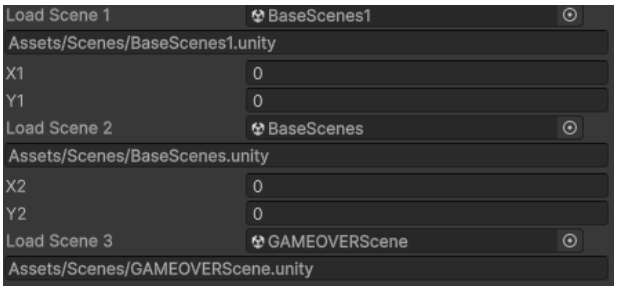

図 20　シーンの設定画面

（c）アイテム管理

○　アイテムデータの管理

プログラム用として、アイテムの情報をまとめて管理できる機能を追加した。

図 21 のようにプルダウンメニューや、チェックを入れるかどうかで表示する項目を変更できるようにした。

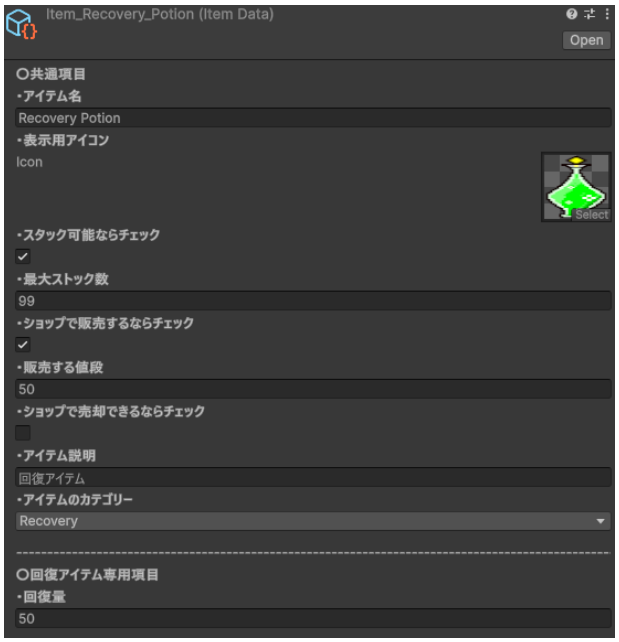

図 21　アイテムの情報画面

○アイテムの使用

アイテムを使用した時の動作は表 6 の通りである。

表６　アイテムのカテゴリー

種類	使用した時の動作
回復アイテム	プレイヤーの体力を回復
武器	見た目・攻撃力を変更
その他（ドロップ品など）	使用できない

アイテムのカテゴリーを変更することで、カテゴリーごとのデータを設定できるようにプログラムした。（図 22）

```
// カテゴリーごとに表示する項目を切り替え
switch (item.category)
{
    case ItemCategory.Recovery:
        EditorGUILayout.LabelField("〇回復アイテム専用項目", EditorStyles.boldLabel);
        EditorGUILayout.LabelField("・回復量", EditorStyles.boldLabel);
        item.RecoveryAmount = EditorGUILayout.IntField(item.RecoveryAmount);
        break;

    case ItemCategory.Weapon:
        EditorGUILayout.LabelField("〇武器専用項目", EditorStyles.boldLabel);
        EditorGUILayout.LabelField("・ダメージ量", EditorStyles.boldLabel);
        item.Damage = EditorGUILayout.FloatField(item.Damage);

        EditorGUILayout.LabelField("・武器の属性", EditorStyles.boldLabel);
        item.weaponattribute = (WeaponAttribute)EditorGUILayout.EnumPopup(item.weaponattribute);
        break;
}
```

図 22　カテゴリーごとのプログラム

〇アイテム管理機能

アイテム管理機能として、インベントリを作成した。インベントリとは、プレイヤーが獲得したアイテムを一覧で管理できる機能である。（図 23）アイテムは最大 24 種類獲得可能である。

プレイヤーが所有しているアイテムのデータは、List を使って管理している。List は、同じ型を複数まとめて扱うことができる。

図 23　インベントリ画面

アイテムごとのスロットは、Prefab を作成している。（図 24）それを複製して List に格納しているアイテムの画像や個数のデータを割り当て、表示している。

図 24　アイテムスロットの Prefab

表示されているアイテムの画像をクリックすると確認画面が出てくるようにした。１スロットに複数所持できるアイテムの場合、プラス・マイナスボタンを押すことで一度に使用するアイテムの個数を変更することができる。（図 25）

アイテムのデータを判定し、アイテムによって表示する内容が変わるようになっている。例えば、アイテムによってテキストが「を使用しますか？」、「を装備しますか？」と変わるようにした。アイテムを一度に複数使用するとき、一度に使用する個数が１つ以上で、所持しているアイテムの個数を超えないように工夫した。（図 256

図 25　アイテム使用確認

```
// plusボタン
1 個の参照
public void Use_PlusButton()
{
    // スタックできるなら
    if (useItemData.data.isStackable)
    {
        if (useItemData.quantity > useAmount)
        {
            useAmount ++;
            SetDisplay_UseCheck(useItemData);
        }
    }
    else
    {
        useAmount = 1;
    }
}

// minusボタン
1 個の参照
public void Use_MinusButton()
{
    // スタックできるなら
    if (useItemData.data.isStackable)
    {
        if (useAmount > 1)
        {
            useAmount --;
            SetDisplay_UseCheck(useItemData);
        }
    }
    else
    {
        useAmount = 1;
    }
}
```

図 26　プラス・マイナスボタンのプログラム

○通貨

　このゲームでの通貨の単位は「JJ」にした。通貨は、アイテムを購入するときに使用する。また、アイテムを売却することで稼ぐことができる。所持金はインベントリ画面か、ショップ画面で確認することができる。

　所持金が増減したときは、どちらの画面でもリアルタイムで金額が変化するようになっている。桁が増えた時を考慮し、テキストボックスの右側を固定し、通貨のアイコンと一緒に左に伸びていくようにした。(図 27、28)

図 27　お金が 1 桁のとき

図 28　お金が 4 桁のとき

○ショップ

　ショップでは、所持金を使い、アイテムを購入・売却することができる。左上にあるボタンを押すことで購入画面、売却画面、終了を切り替えることができる。購入するときも、売却するときも、図 25 と同じような確認画面が出てくるようにした。

・アイテムの購入（図 29）

　アイテムの画像をクリックすることで選択することができる。購入できる場合は確認画面が出てくる。インベントリが埋まっているか、所持金よりも高い値段のものを買おうとした場合、選択できないようにした。

　購入できるアイテムは、回復アイテム、武器となっている。

図 29　アイテム購入画面

・　アイテムの売却（図 30）

　こちらもアイテムの画像をクリックすることで選択することができる。売却できる場合は確認画面が出てくる。売却できるアイテムは、基本的には初期装備以外のアイテムにした。売却できないアイテムは売却価格が×になり、アイコンを押しても反応しない。

ショップで購入できるアイテムは、売却と購入を繰り返して無限にお金が増えることを防ぐために、値段を売値の半分にした。また、自分の所持品ということが分かるようにインベントリと同じレイアウトにした。

図 30　アイテム売却画面

（c）セーブ＆ロード機能

主人公の家のベッドでセーブをすることができる。

セーブができる場所を固定することで保存するデータ量を減らした。セーブできる内容は以下の通りとなっている。（図 31）

・ストーリー、ダンジョンの進行度
・プレイヤーの体力、ステータス
・プレイヤーの所持品、所持金

図 31　セーブできる場所

ロードはタイトル画面から行えるようにした。データを読み込むと、進行状況を引き継いで主人公の家から始まるようになっている。

（３）ゲームデザイン

EDGE を用いて描いた絵を、Unity で Player や敵のアニメーションを作成した。

（a）プレイヤーデザイン・アニメーション

プレイヤーは上下左右の動きに加え、歩く、走る、立ち止まる（待機モーション）を描いたので、数が相当多くなった。プレイヤーにはバッグをそうびさせていたため、ただ反転させるだけでなく、反転させた後にバッグをつけ足したりもした。走るモーションを描く時には、絵全体を１から２ドット程度上に上げた後、足の形を変えた。また、足から砂埃などをつけ足し、早く移動しているような感じを出すようにした。（図 32）

図 32　プレイヤーのアニメーション

（b）敵デザイン

絵を描くときには光源を一か所決めることで、図のように影を配置していくと、立体感が生まれ、ドット絵でも奥行きを表現することができた。（図 33）

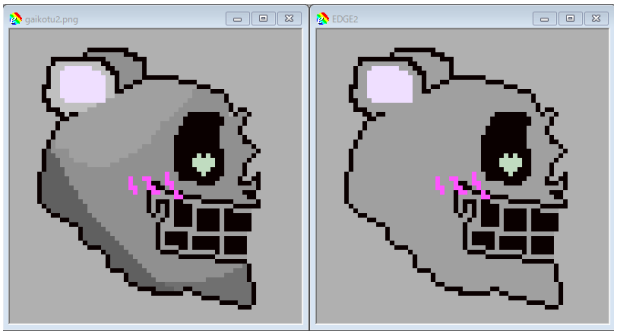

図 33　陰影のつけ方

○魔女のアニメーション

　魔女は、これらの２つの絵を一定時間ごとに交互に表示している。腕を振り上げたときは躍動感を持たせるために、１ドットだけ上に描いた。（図 34）

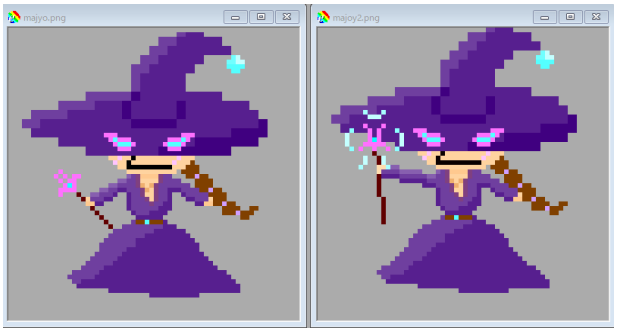

図 34　魔女のアニメーション

　魔女は出現した後、その場から動かないが、攻撃範囲を大きく設定することで、難易度を調整した。

　攻撃範囲は Collider をつけ、当たり判定をなくし、その Collider の範囲内に雷の柱が３本ランダムに出現するようにした。（図 35）

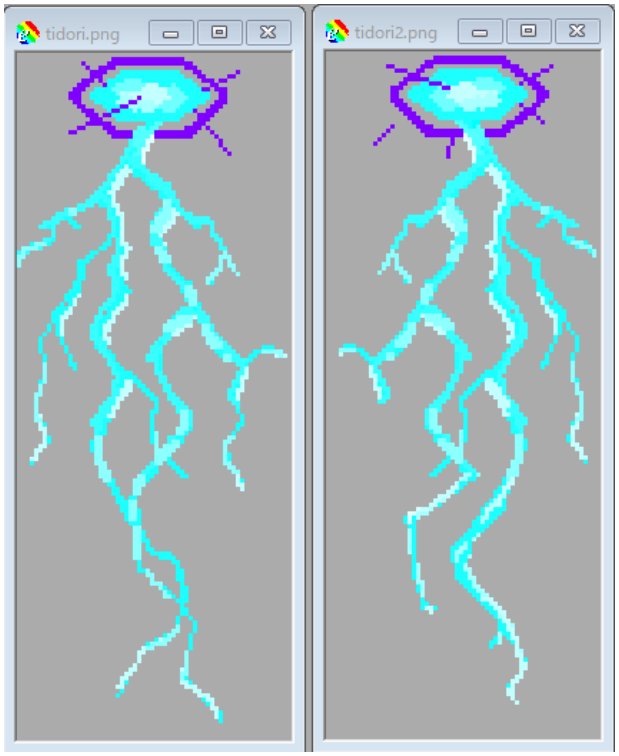

図 35　雷のアニメーション

○爆弾のアニメーション

　ボスが投げる爆弾のアニメーションは、秒数間隔にこだわった。

　早く消えすぎるとプレイヤーにあたる前に消えてしまい、意味がなくなってしまう。しかし、遅すぎると不自然に見えてしまう。秒数間隔をうまく配置するために、何回も試行錯誤した。（図 36）

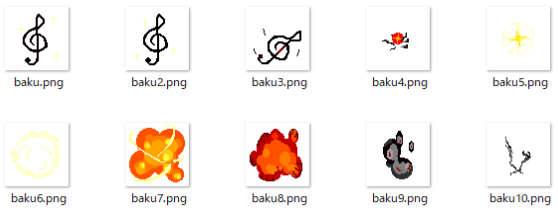

図 36　爆弾のアニメーション

　１から３枚目はゆっくり、４から６枚目は早く、７から８枚目はゆっくり、９から 10 枚目は早くする、というタイミングで表示す

ることで、爆弾がプレイヤーに当たりやすくしている。また、７から８枚目だけに当たり判定をつけることで、ダメージが入るタイミングを調整している。（図 37、38）

図 37　爆弾の表示の秒数間隔（１）

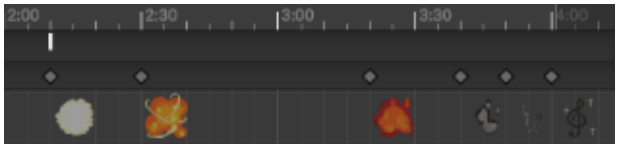

図 38　爆弾の表示の秒数間隔（２）

（c）武器のデザイン

プレイヤーが使用する武器はゲームのプレイ感に直結するため、形状や色を変えて武器ごとの差別化を図った。初期装備のような簡素なものから、上位の武器では金属の光沢や装飾を追加するなど、強さが見た目から伝わるよう意識してデザインを行った。また属性がわかりやすいように、赤は火属性、青は水属性、黄色は光属性など属性が一目で判断できるように色分けをはっきりと行った。武器を作るうえで難しかったのは陰影のつけ方だった。白から黒に陰影をつけるとドット絵としての味がなくなり、リアルな武器になってしまうことがあった。（図 39）そこで陰影をつけないようにして、色を部位ごとに分けることで分かりやすいイラストになった。（図 40）

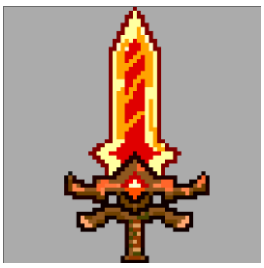

図 39　武器の失敗例　図 40　武器の成功例

（d）ドロップアイテムの制作

敵を倒した際にドロップするアイテムを実装した。薬草・素材・特殊アイテムなど、種類ごとに見分けがつくようにし、ゲーム内でひと目で判別できることを目的としている。

実装にあたっては、種類ごとに色味や形を変えてデザインし、見やすさを重視して制作した。また、市販されているゲームと同じ絵にならないよう注意しながら描いた。さらに、ゲーム全体の世界観に自然になじむよう、影の付け方や縁取りの太さを統一し、縮小表示されてもつぶれにくいシルエットになるよう調整した。見やすく、分かりやすいアイコンになることを意識して制作した（図 41、42）。

図 41　アイテム例（１）

図 42　アイテムの例（２）

（e）UI デザイン

プレイヤーの UI には図 43 のように体力バーとスタミナバー、武器、図 44 のように経験値バーを追加した。

最初は実数値で UI の管理をしていたが値が変化すると UI の挙動がおかしくなったので割合での管理を行うことで、その問題を解決した。

図 43　プレイヤーの UI（１）

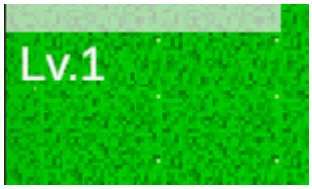

図 44　プレイヤーの UI（２）

（f）タイトル・背景

○タイトル画面の制作

ゲームの第一印象となるスタート画面では、タイトルロゴや背景を制作した。（図 45）マップやストーリーの雰囲気に合う色彩でまとめ、プレイヤーがゲームの世界へ入りやすいようシンプルながら印象に残るデザインになるよう工夫した。背景は奥行きが出るように、絵の後ろになるところから描き、ゲームが始まったのがわかりやすくするために手前には一番最初に出てくる村を配置した。

ダンジョンとしての雰囲気を出すために塔を上にかけていくほど少しずつ細くし、正面よりも少しだけ斜めに描いた。そして斜めに塔を描くことで登っていくという印象を与えられるようにした。

図 45　ゲームタイトルの絵

○背景

ゲームの世界観を表現するために、タイルマップの背景素材を制作した。制作したエリアは「村」「ジャングル」「氷山」「火山」の４種類で、それぞれの地域らしさが伝わるように色彩や配色パターンを工夫している。（表７）（図 46）　こうすることで、マップ全体の雰囲気に統一感が出るように制作した。

特に難しかったことは、色の使い分けである。ドット絵といっても色に統一感を出しすぎて陰影などをつけないでいるとのっぺりとした立体感のない絵が出来上がってしまうことがあったため、影がかかっているところは少し黒くしたり、色の濃度を上げたりする工夫を行った。

表７　マップの配色

マップ	使用押した色の特徴
村	明るい色
ジャンル	緑色
氷山	寒色系
火山	暖色系

図 46　村のタイルマップ

・タイルマップ用オブジェクトの制作

背景だけでは世界に奥行きやゲーム感が生まれないため、マップ上に配置する木・家・神殿･城などのオブジェクトの制作を行った。オブジェクトはタイルとの見た目の整合性を意識し、サイズ･色･陰影を統一することで、マップに自然に溶け込むデザインになるよう心掛けた。

特にこだわったのは「木」である。ダンジョンごとに、村は普通の木、ジャングルは生い茂った木、氷山は凍った木などとの地域の特色に合わせた木の雰囲気にすることでマップごとの違いがはっきりと分かりやすくした。また、配置するときにも一か所に集まらないように自然と配置するように注意した。さらに、絵の反転を行うことで同じ絵でも全く同じ景色にならないようになると気づき、大量に配置するオブジェクトは反転を行った。(図 47、48)

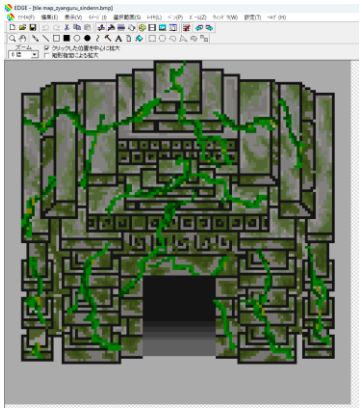

図 47　オブジェクトの例（１）

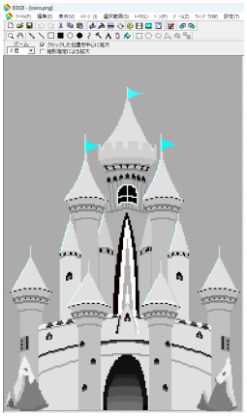

図 48　オブジェクトの例（２）

（g）BGM 制作

ダンジョンごとにテーマが決められているので BGM にもテーマを持たせて制作を行った。音楽はリズム、ハーモニー、メロディの３つの要素からできていて、今回の制作ではこの３つの要素に加えて簡単なアレンジを行うことができた。

１～３層のジャングルでは民族音楽のような雰囲気を出すために、現在の pop ミュージックで使われるギターやベース、ドラムなどは一切使わずに民族楽器を用いて制作を行った。民族楽器についての情報は調べてもなかなかで出てこず、使い方に苦労したが色々試す事で徐々に使い方を理解する事ができ自分の出したい雰囲気をだす事ができた。

図 49 が実際の制作画面である。

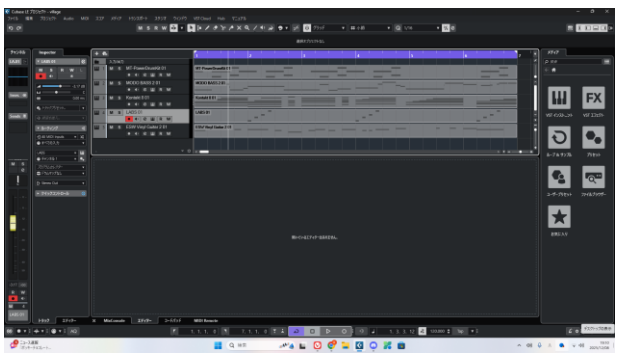

図 49　BGM 制作画面

（h）ストーリー制作

ゲーム全体の統一感を出すために、簡単なストーリーを作成した。物語の背景設定や目的がプレイヤーに伝わるよう構成し、マップやアイテムの雰囲気と合うよう意識した。物語形式にして、ゲームを楽しんでもらえるようにも工夫した。物語を作るのは初めてだったので、実際にスマホで作られている 2DRPG をプレイし、アイデアをもらった。こうすることによって、ゲーム進行や物語の進め方などがわかりやすくなり、ストーリー制作の役に立った。スムーズに戦闘シーンに場面移動するのがとても難しかった。

（i）シーン移動とナレーション

ゲームの導入部分として､図 50 のような流れでシーン移動する演出用スクリプトを制作した。

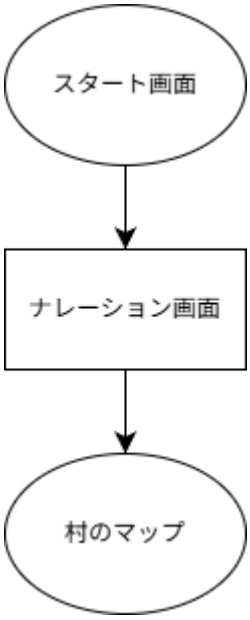

図 50　シーン移動

ゲームの世界観や物語の導入を自然に伝えることを目的とし、いきなり操作を始めるのではなく、ナレーションを挟むことでプレイヤーが物語に入り込みやすくなるよう工夫した。（図 51）

スクリプトでは、ボタン入力をきっかけにナレーションを再生し、ナレーションが最後まで再生されたことを判定してから場面移動を行うようにしている。これにより、物語の流れを途切れさせずに村のシーンへ移動できるようになった。

図 51　ナレーションの例

（j）UI 管理

Unity には UI を表示するための機能として Canvas がある。インベントリに使う UI や、ショップに使う UI など、それぞれの Canvas ごとでまとめて表示・非表示することができる。ただ、このままだとそれぞれの Canvas を同時に開けてしまうという問題があったので、図 52 のように Canvas をまとめて管理するスクリプトを作成し、フラグを立てて同時に開かないようにした。また、Canvas を開いているときに敵から攻撃されると困るので、その間は時間を止めてプレイヤーや敵が動かないようにしている。他にも、Esc ボタンを押すと､どの Canvas でも閉じることができるようにした。

```
// インベントリ開閉
1 個の参照
public void ToggleInventory()
{
    // 表示・非表示を切り替える
    if (inventoryCanvas.activeSelf)
    {
        //閉じる
        CanvasClose();
        Time.timeScale = 1;                    // 時間を動かす
    }
    else if (!inventoryCanvas.activeSelf)
    {
        if(Time.timeScale == 1)
        {
            //開く
            inventoryCanvas.SetActive(true);
            Time.timeScale = 0;                // パネルを開いているときに時間を止める
            InventoryUI.Instance.RefreshUI();  // 開いたときに内容を更新
        }
    }
}
```

図 52　インベントリ Canvas 開閉プログラム

３．研究のまとめ

・谷田

以前 Unity での個人ゲーム制作は行ったことはあるが、共同制作は初めてだった。共同制作を進めていく上で、コミュニケーションを特に意識した。お互いの進行状況を共有して次回までにやっておくこと、次回することを決めておいた。しかし、メンバーの半分が初めて Unity を触るということもあり、なかなかスケジュール通りに進めていくことが難しかった。後半になるにつれてメンバー全員が慣れてきたため､作業スピードが上がった。

プログラムを書いていく上で、自分以外が見たときに理解しやすいように書くことを意識した。メンバーが書いたスクリプトと混在しないような変数名にしたり、コメント文を書くようにした。特に自分以外が作業する可能性があるところは分かりやすいように配慮して制作した。そうすることで、自分やメンバーがバグを直すときや新しい機能を追加するときにスムーズに行うことができた。

今回共同開発をするということで、メンバーと進捗を共有するために Github を使用した。もし同じ Scene やスクリプトに変更をかけるとコンフリクトが起きてしまう｡そこで、自分専用の Scene を作ったり、プロジェクトの作業履歴を分岐させる branch を使うことで想定外のコンフリクトを減らした。

ゲームの共同開発をすることで、コミュニケーションや情報共有、読みやすいコード設計など、様々なことを学ぶことができた。この経験を将来に生かしていきたい。

・出﨑

今回初めて Unity を使って初めてゲーム制作を行ったが、今まで学習したプログラミングの考え方や、分からないことを的確に調べる能力を活用することで、あまり苦戦することなく制作を進めることができた。しかし、計画の甘さや自身の能力の過信などがあり、スケジュール通りに開発が進まなかった。

初めて GitHub というサービスを用いてデータの共有をして共同開発を進めて行ったが最初はいまいち効果的な使い方がわからなかったが調べていくうちに効果的な使い方がわかり、開発を進めていくことができた。

初めて Unity を使ってのゲーム制作、ゲームのグラフィックの作成、BGM 制作を行ったが全ての工程で新しい発見があった。今回の経験したことを忘れずに、これから先も活かしていきたい。

・濵戸

私は課題研究を通して２つのことを学んだ。

１つ目は、チームメイトとコミュニケーションをとることの大切さである。私は Unity を使って本格的な 2D ゲームを制作したことはなく、エラーなどの様々な壁にぶつかることが多かった。Web で調べても、AI に質問してみてもわからなかった時は、チームメイトに相談してみた。すると、嫌な顔一つせず、一緒に解決策を考えてくれて、一人で考えるよりもずっと早く解決することができた。

２つ目は、絵の描き方である。もともと鉛筆などで絵を描くのは得意だったので、デザイン担当で仕事を任せてもらった。デジタルで描くのは初めてのことであり、影のつけ方などがアナログで描くときとは全く異なり、なかなかうまくかけなかったので、チームメイトと一緒に学校の図書室にいき、ドット絵に関する本を一緒に探し、それを参考に描いた。また、チームメイトが自分の絵を褒めてくれることも､私のモチベーションになった。

・武入

今回の課題研究では、Unity を使ってチームでゲーム制作を行った。個人での制作経験がある人や、Unity を初めて触る人もおり、最初はスケジュール通りに進まないことも多かった。しかし、話し合いを重ねていくうちに操作に慣れ、後半は作業スピードが上がった。

共同制作を進めるうえで、コミュニケーションの大切さを強く感じた。分からないことやエラーが出たときは、チームメンバーに相談することで一人で悩むより早く解決できた。また、GitHub を使って進捗を共有し、branch や専用 Scene を活用することで、作業の衝突を減らすことができた。

プログラム制作では、他の人が見ても分かりやすいコードを書くことを意識し、変数名やコメントに気を付けた。そのおかげで、修正や機能追加がしやすくなった。

デザイン面では、EDGE を使ってマップ用オブジェクト、武器、アイテム、スタート画面などを制作した。色や形を統一し、見やすく分かりやすいデザインを心がけた。特に木や武器は、場所や強さが一目で分かるよう工夫した。

また、ゲームの世界観を伝えるために簡単なストーリーを作り、ナレーションを入れてから最初の村へ移動する演出も行った。

この課題研究を通して、チームで協力することの大切さや、ゲーム制作の難しさと楽しさを学ぶことができた。

４．参考文献

note
https://note.com/

Reddit
https://www.reddit.com/

Qiita
https://qiita.com/

侍エンジニア
https://www.sejuku.net/blog/category/development-environment/unity

SoundQuest
https://soundquest.jp/

クランとリオンの【初心者向け DTM 動画】
https://www.youtube.com/@kurantorion

ドット絵教室　著/中川　悠京

作りながら覚える作曲入門２．０
著/monaca：factory

たのしい 2D ゲームの作り方　Unity ではじめるゲーム開発入門 著/STUDIO SHIN

たのしい 2D ゲームの作り方　第２版　Unity ではじめるゲーム開発入門　著/STUDIO SHIN

〇使用した素材サイト
効果音ラボ
https://soundeffect-lab.info/