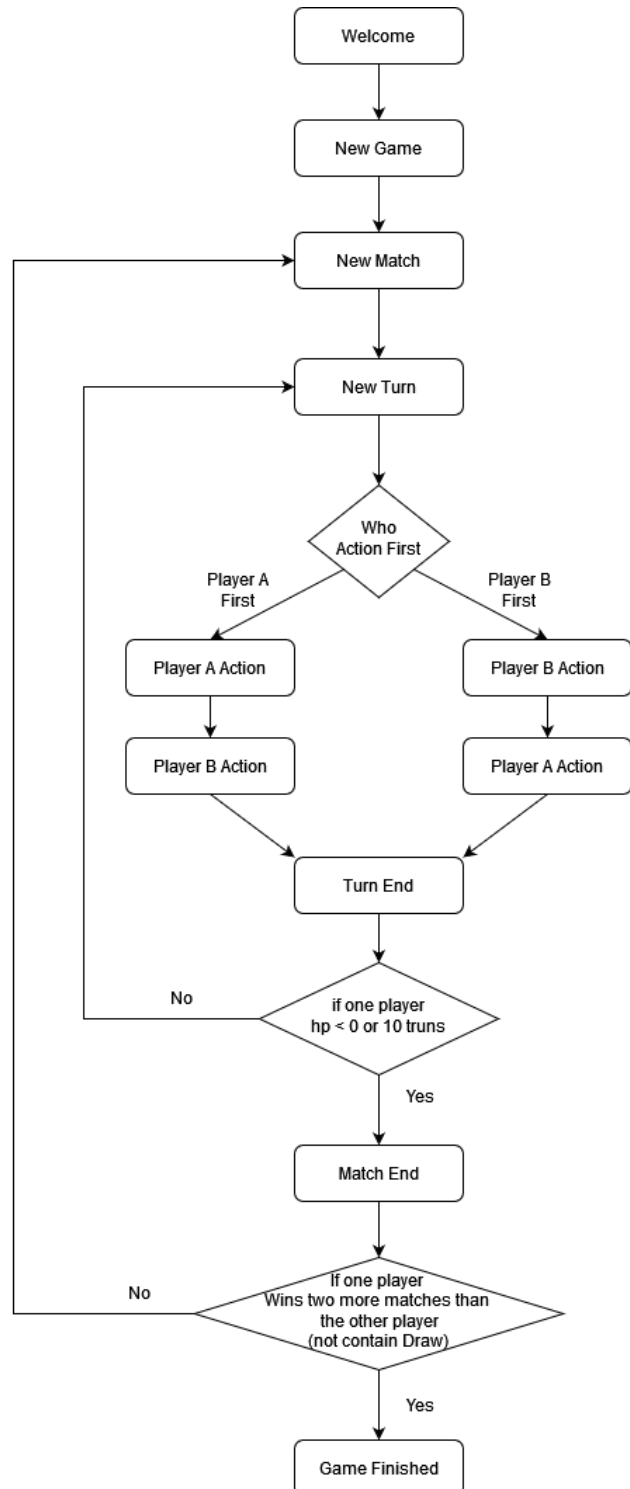


互動程式設計三 期中專案

資工四 110590034 楊榮鈞

期中專案的 Repo : https://github.com/kyynk/RPG_mid

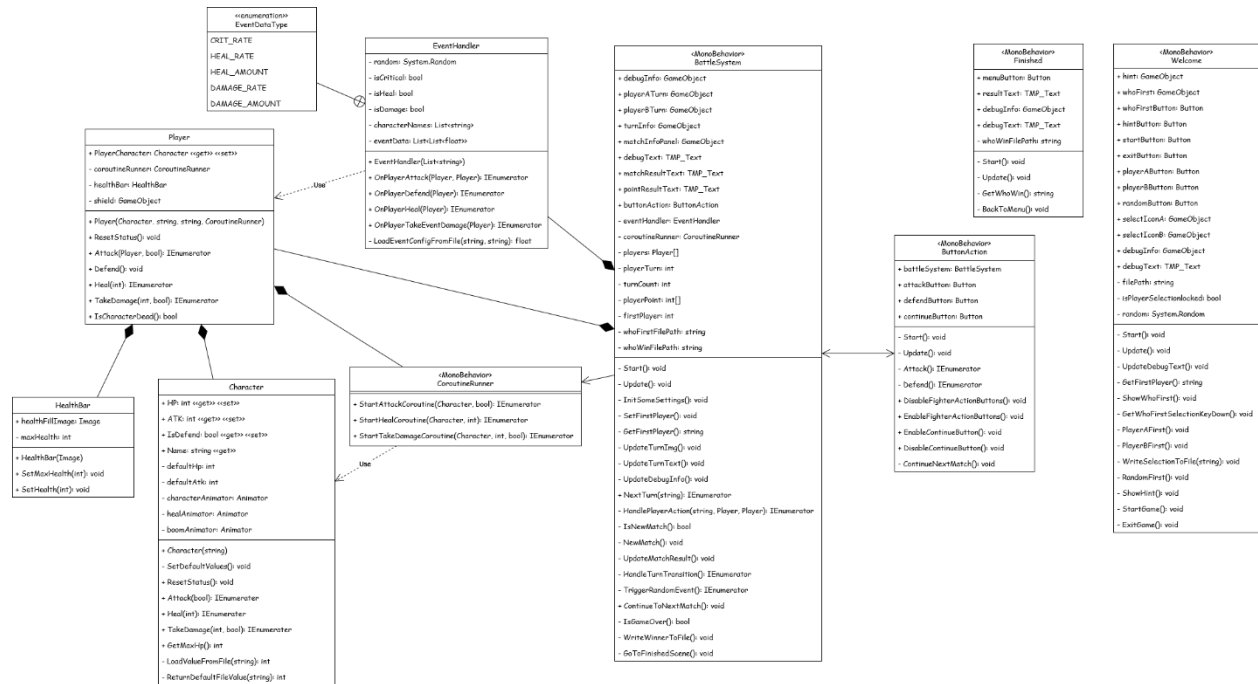
Flowchart



Class Diagram

如想看放大圖片可至

https://raw.githubusercontent.com/kyynk/RPG_mid/refs/heads/main/DocumentAndGraph/SomeGraph/class_diagram.png



Welcome.cs

```
using TMPro;
using System.IO;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

namespace RPGBattle
{
    public class Welcome : MonoBehaviour
    {
        public GameObject hint;
        public GameObject whoFirst;
        public Button whoFirstButton;
        public Button hintButton;
        public Button startButton;
        public Button exitButton;
        public Button playerAButton;
        public Button playerBButton;
        public Button randomButton;
        public GameObject selectIconA;
        public GameObject selectIconB;
        public GameObject debugInfo;
        public TMP_Text debugText;

        private string filePath;
        private bool isPlayerSelectionlocked;
        private System.Random random;

        void Start()
        {
            filePath = Path.Combine(Application.streamingAssetsPath,
"who_first.txt"); // Set file path
            random = new System.Random();

            whoFirstButton.onClick.AddListener(ShowWhoFirst);
            hintButton.onClick.AddListener(ShowHint);
            startButton.onClick.AddListener(StartGame);
            exitButton.onClick.AddListener(ExitGame);
        }
    }
}
```

```
playerAButton.onClick.AddListener(PlayerAFirst);
playerBButton.onClick.AddListener(PlayerBFirst);
randomButton.onClick.AddListener(RandomFirst);
PlayerAFirst();
isPlayerSelectionlocked = true;
hint.SetActive(false);
whoFirst.SetActive(false);
debugInfo.SetActive(false);
}
```

// Update is called once per frame

```
void Update()
{
    if (Input.GetKeyDown("w"))
    {
        ShowWhoFirst();
    }
    else if (Input.GetKeyDown("h"))
    {
        ShowHint();
    }
    else if (Input.GetKeyDown("s"))
    {
        StartGame();
    }
    else if (Input.GetKeyDown("e"))
    {
        ExitGame();
    }
    else if (Input.GetKeyDown(KeyCode.Tab))
    {
        if (debugInfo.activeSelf)
        {
            debugInfo.SetActive(false);
        }
        else
        {
            debugInfo.SetActive(true);
        }
    }
}
```

```
        else if (!isPlayerSelectionlocked)
        {
            GetWhoFirstSelectionKeyDown();
        }
        UpdateDebugText();
    }

    private void UpdateDebugText()
    {
        debugText.text = "State: Welcome\n" +
            "Who First:\n" +
            GetFirstPlayer();
    }

    private string GetFirstPlayer()
    {
        if (File.Exists(filePath))
        {
            return File.ReadAllText(filePath);
        }
        else
        {
            Debug.LogError("File not found!");
            return "Player A";
        }
    }

    private void ShowWhoFirst()
    {
        hint.SetActive(false);
        whoFirst.SetActive(true);
        isPlayerSelectionlocked = false;
    }

    private void GetWhoFirstSelectionKeyDown()
    {
        if (Input.GetKeyDown("a"))
        {
            PlayerAFirst();
        }
    }
}
```

```

        else if (Input.GetKeyDown("b"))
        {
            PlayerBFirst();
        }
        else if (Input.GetKeyDown("r"))
        {
            RandomFirst();
        }
    }

    private void PlayerAFirst()
    {
        selectIconA.SetActive(true);
        selectIconB.SetActive(false);
        WriteSelectionToFile("Player A");
    }

    private void PlayerBFirst()
    {
        selectIconA.SetActive(false);
        selectIconB.SetActive(true);
        WriteSelectionToFile("Player B");
    }

    private void WriteSelectionToFile(string selection)
    {
        try
        {
            File.WriteAllText(filePath, selection);
        }
        catch (IOException ex)
        {
            Debug.LogError($"Failed to write to file: {ex.Message}");
        }
    }

    private void RandomFirst()
    {
        if (random.Next(0, 2) == 0)
        {

```

```
        PlayerAFirst();
    }
    else
    {
        PlayerBFirst();
    }
}

private void ShowHint()
{
    hint.SetActive(true);
    whoFirst.SetActive(false);
    isPlayerSelectionlocked = true;
}

private void StartGame()
{
    SceneManager.LoadScene("BattleScene");
}

private void ExitGame()
{
    Application.Quit();
}
}
```

BattleSystem.cs

```
using TMPro;
using System;
using System.IO;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

namespace RPGBattle
{
    public class BattleSystem : MonoBehaviour
    {
        // UI references
        public GameObject debugInfo;
        public GameObject playerATurn;
        public GameObject playerBTurn;
        public GameObject turnInfo;
        public GameObject matchInfoPanel; // MatchInfo panel
        public TMP_Text debugText;
        public TMP_Text matchResultText; // Match result text
        public TMP_Text pointResultText; // Point result text
        public ButtonAction buttonAction;

        private EventHandler eventHandler;
        private CoroutineRunner coroutineRunner;
        private Player[] players;
        private int playerTurn; // 0 or 1 (player 1 or player 2)
        private int turnCount;
        private int[] playerPoint;
        private int firstPlayer;
        private string whoFirstFilePath;
        private string whoWinFilePath;

        private void Start()
        {
            GameObject runnerObject = new GameObject("CoroutineRunner");
            coroutineRunner = runnerObject.AddComponent<CoroutineRunner>();
            eventHandler = new EventHandler(new List<string> { "Giant",
"Paladin" });
        }
    }
}
```



```

        players = new Player[2];
        players[0] = new Player(new Character("Giant"), "L_HP", "L_Shield",
coroutineRunner);
        players[1] = new Player(new Character("Paladin"), "R_HP", "R_Shield",
coroutineRunner);
        playerPoint = new int[2] { 0, 0 };
        whoFirstFilePath = Path.Combine(Application.streamingAssetsPath,
"who_first.txt");
        whoWinFilePath = Path.Combine(Application.streamingAssetsPath,
"who_win.txt");
        debugInfo.SetActive(false);
        InitSomeSettings();
    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Tab))
        {
            if (debugInfo.activeSelf)
            {
                debugInfo.SetActive(false);
            }
            else
            {
                debugInfo.SetActive(true);
                UpdateDebugInfo();
            }
        }
    }

    private void InitSomeSettings()
    {
        turnCount = 1;
        SetFirstPlayer();
        matchInfoPanel.SetActive(false);
        turnInfo.SetActive(true);
        foreach (Player player in players)
        {
            player.ResetStatus();
        }
    }

```

```

        UpdateTurnText();
        UpdateDebugInfo();
    }

    private void SetFirstPlayer()
    {
        string firstPlayerName = GetFirstPlayer();
        if (firstPlayerName == "Player A")
        {
            firstPlayer = 0;
            playerTurn = 0;
        }
        else
        {
            firstPlayer = 1;
            playerTurn = 1;
        }
        UpdateTurnImg();
    }

    private string GetFirstPlayer()
    {
        if (File.Exists(whoFirstFilePath))
        {
            return File.ReadAllText(whoFirstFilePath);
        }
        else
        {
            Debug.LogError("File not found!");
            return "Player A";
        }
    }

    private void UpdateTurnImg()
    {
        if (playerTurn == 0)
        {
            playerATurn.SetActive(true);
            playerBTurn.SetActive(false);
        }
    }

```

```

        else
        {
            playerATurn.SetActive(false);
            playerBTurn.SetActive(true);
        }
    }

    private void UpdateTurnText()
    {
        TMP_Text turnInfoText = turnInfo.GetComponent<TMP_Text>();
        if (turnInfoText != null)
        {
            turnInfoText.text = "Round " + turnCount;
        }
        else
        {
            Debug.LogError("Text component is missing on TurnText
GameObject!");
        }
    }

    private void UpdateDebugInfo()
    {
        debugText.text = "State: " + (playerTurn == 0 ? "Player A" : "Player B")
+ "\n" +
            "Player A: \nHP=" + players[0].PlayerCharacter.HP +
            ", ATK=" + players[0].PlayerCharacter.ATK +
            ", DEFEND=" + (players[0].PlayerCharacter.IsDefend ?
"true" : "false") + "\n" +
            "Player B: \nHP=" + players[1].PlayerCharacter.HP +
            ", ATK=" + players[1].PlayerCharacter.ATK +
            ", DEFEND=" + (players[1].PlayerCharacter.IsDefend ?
"true" : "false");
    }

    public IEnumerator NextTurn(string action)
    {
        // every turn need two players to attack each other, so we need to get the
        current player and the opponent
        Player currentPlayer = players[playerTurn];

```

```

    Player opponent = players[(playerTurn + 1) % 2];
    buttonAction.DisableFighterActionButtons();
    yield return HandlePlayerAction(action, currentPlayer, opponent);
    if (firstPlayer != playerTurn)
    {
        turnCount++;
    }
    UpdateDebugInfo(); // update debug info for player property
    if (IsNewMatch())
    {
        NewMatch();
    }
    else
    {
        yield return HandleTurnTransition();
    }
}

private IEnumerator HandlePlayerAction(string action, Player
currentPlayer, Player opponent)
{
    if (action == "attack")
    {
        yield return eventHandler.OnPlayerAttack(currentPlayer, opponent);
    }
    else if (action == "defend")
    {
        yield return eventHandler.OnPlayerDefend(currentPlayer);
    }
    else
    {
        Debug.LogError("Invalid action!");
    }
}

private bool IsNewMatch()
{
    // if 10 turn or one player hp <= 0, then the match is over
    return turnCount > 10 || players[0].IsCharacterDead() ||
players[1].IsCharacterDead();

```

```

    }

    private void NewMatch()
    {
        UpdateMatchResult();
        matchInfoPanel.SetActive(true);
        turnInfo.SetActive(false);
        buttonAction.DisableFighterActionButtons();
        buttonAction.EnableContinueButton();
    }

    private void UpdateMatchResult()
    {
        // has results: p1 win, p2 win, draw
        if (players[1].IsCharacterDead())
        {
            matchResultText.text = "Player A Win";
            playerPoint[0]++;
        }
        else if (players[0].IsCharacterDead())
        {
            matchResultText.text = "Player B Win";
            playerPoint[1]++;
        }
        else
        {
            matchResultText.text = "Draw";
        }
        pointResultText.text = playerPoint[0] + " - " + playerPoint[1];
    }

    private IEnumerator HandleTurnTransition()
    {
        UpdateTurnText();
        playerTurn = (playerTurn + 1) % 2;
        UpdateTurnImg();
        UpdateDebugInfo(); // update debug info for state
        yield return TriggerRandomEvent();
        if (IsNewMatch()) // since maybe the random event cause the match over
        {

```

```

        NewMatch();
    }
    else
    {
        buttonAction.EnableFighterActionButtons();
    }
}

private IEnumerator TriggerRandomEvent()
{
    Player currentPlayer = players[playerTurn];
    yield return eventHandler.OnPlayerHeal(currentPlayer);
    UpdateDebugInfo(); // update debug info for player property (hp)
    yield return eventHandler.OnPlayerTakeEventDamage(currentPlayer);
    UpdateDebugInfo(); // update debug info for player property (hp)
}

public void ContinueToNextMatch()
{
    buttonAction.EnableFighterActionButtons(); // Re-enable buttons
    buttonAction.DisableContinueButton(); // Hide Continue button
    if (IsGameOver())
    {
        WriteWinnerToFile();
        GoToFinishedScene();
    }
    else
    {
        InitSomeSettings();
    }
}

private bool IsGameOver()
{
    return Math.Abs(playerPoint[0] - playerPoint[1]) == 2;
}

private void WriteWinnerToFile()
{
    try

```

```
{
    string winner = playerPoint[0] > playerPoint[1] ? "Player A" : "Player
B";
    File.WriteAllText(whoWinFilePath, winner);
}
catch (IOException ex)
{
    Debug.LogError($"Failed to write to file: {ex.Message}");
}
}

private void GoToFinishedScene()
{
    SceneManager.LoadScene("FinishedScene");
}
}
```

Finished.cs

```
using TMPro;
using System.IO;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class Finished : MonoBehaviour
{
    public Button menuButton;
    public TMP_Text resultText;
    public GameObject debugInfo;
    public TMP_Text debugText;

    private string whoWinFilePath;

    // Start is called before the first frame update
    void Start()
    {
        whoWinFilePath = Path.Combine(Application.streamingAssetsPath,
"who_win.txt");
        menuButton.onClick.AddListener(BackToMenu);
        debugInfo.SetActive(false);

        resultText.text = GetWhoWin() + "!!!";
        debugText.text = "State: Finished\n" +
            "Winner:" + GetWhoWin();
    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Tab))
        {
            if (debugInfo.activeSelf)
            {
                debugInfo.SetActive(false);
            }
            else
            {

```



```
        debugInfo.SetActive(true);
    }
}
else if (Input.GetKeyDown("m"))
{
    BackToMenu();
}
}

private string GetWhoWin()
{
    if (!File.Exists(whoWinFilePath))
    {
        return "File Not Found";
    }
    return File.ReadAllText(whoWinFilePath);
}

private void BackToMenu()
{
    SceneManager.LoadScene("WelcomeScene");
}
}
```

ButtonAction.cs

```
using System.Collections;
using UnityEngine;
using UnityEngine.UI;

namespace RPGBattle
{
    public class ButtonAction : MonoBehaviour
    {
        public BattleSystem battleSystem;
        public Button attackButton;
        public Button defendButton;
        public Button continueButton;

        void Start()
        {
            // Add listener for mouse clicks
            attackButton.onClick.AddListener(() => StartCoroutine(Attack()));
            defendButton.onClick.AddListener(() => StartCoroutine(Defend()));
            continueButton.onClick.AddListener(ContinueNextMatch);

            // Initially disable ContinueButton
            continueButton.gameObject.SetActive(false);
        }

        // Update is called once per frame
        void Update()
        {
            if (attackButton.interactable && defendButton.interactable)
            {
                if (Input.GetKeyDown("a"))
                {
                    StartCoroutine(Attack());
                }
                else if (Input.GetKeyDown("d"))
                {
                    StartCoroutine(Defend());
                }
            }
            else if (Input.GetKeyDown("c"))
            {
                StartCoroutine(ContinueNextMatch());
            }
        }
    }
}
```

```
        {
            ContinueNextMatch();
        }
    }

    private IEnumerator Attack()
    {
        yield return battleSystem.NextTurn("attack");
    }

    private IEnumerator Defend()
    {
        yield return battleSystem.NextTurn("defend");
    }

    public void DisableFighterActionButtons()
    {
        attackButton.interactable = false;
        defendButton.interactable = false;
    }

    public void EnableFighterActionButtons()
    {
        attackButton.interactable = true;
        defendButton.interactable = true;
    }

    public void EnableContinueButton()
    {
        continueButton.gameObject.SetActive(true); // Show Continue button
    }

    public void DisableContinueButton()
    {
        continueButton.gameObject.SetActive(false); // Hide Continue button
    }

    private void ContinueNextMatch()
    {
        battleSystem.ContinueToNextMatch();
    }
}
```

```
}  
}  
}
```

Character.cs

```
using System;
using System.IO;
using System.Collections;
using UnityEngine;

namespace RPGBattle
{
    public class Character
    {
        public int HP { get; set; }
        public int ATK { get; set; }
        public bool IsDefend { get; set; }

        public string Name { get; }

        private int defaultHp;
        private int defaultAtk;
        private Animator characterAnimator;
        private Animator healAnimator;
        private Animator boomAnimator;

        public Character(string _name)
        {
            Name = _name;
            GameObject gameObject =
GameObject.FindGameObjectWithTag(Name);
            if (gameObject == null)
            {
                Debug.LogError($"GameObject for {Name} not found!");
            }
            characterAnimator = gameObject.GetComponent<Animator>();
            gameObject = GameObject.FindGameObjectWithTag(Name + "Heal");
            if (gameObject == null)
            {
                Debug.LogError($"GameObject for {Name}Heal not found!");
            }
            healAnimator = gameObject.GetComponent<Animator>();
            gameObject = GameObject.FindGameObjectWithTag(Name + "Boom");
            if (gameObject == null)
```

```

    {
        Debug.LogError($"GameObject for {Name} Boom not found!");
    }
    boomAnimator = gameObject.GetComponent<Animator>();
    SetDefaultValues();
    ResetStatus();
}

private void SetDefaultValues()
{
    defaultHp = LoadValueFromFile("hp");
    defaultAtk = LoadValueFromFile("atk");
}

public void ResetStatus()
{
    HP = defaultHp;
    ATK = defaultAtk;
    IsDefend = false;
    characterAnimator.Play("idle");
    healAnimator.Play("hidden");
    boomAnimator.Play("hidden");
}

public IEnumerator Attack(bool isCritical)
{
    if (isCritical)
    {
        characterAnimator.Play("crit_attack");
    }
    else
    {
        characterAnimator.Play("attack");
    }
    yield return new
    WaitForSeconds(characterAnimator.GetCurrentAnimatorStateInfo(0).length);
}

public IEnumerator Heal(int amount)
{

```

```

        healAnimator.Play("heal");
        HP += amount;
        yield return new
WaitForSeconds(healAnimator.GetCurrentAnimatorStateInfo(0).length);
    }

    public IEnumerator TakeDamage(int amount, bool isEventDamage)
    {
        if (isEventDamage)
        {
            boomAnimator.Play("boom");
            yield return new
WaitForSeconds(boomAnimator.GetCurrentAnimatorStateInfo(0).length);
        }

        if (IsDefend)
        {
            HP -= amount / 2;
            IsDefend = false;
        }
        else
        {
            HP -= amount;
        }

        characterAnimator.Play("injure");
        yield return new
WaitForSeconds(characterAnimator.GetCurrentAnimatorStateInfo(0).length);
    }

    public int GetMaxHp()
    {
        return defaultHp;
    }

    private int LoadValueFromFile(string fileName)
    {
        string filePath = Path.Combine(Application.streamingAssetsPath,
"PlayerConfig", fileName + ".csv");

```

```

        if (!File.Exists(filePath))
        {
            Debug.LogError($"File {fileName} not found!");
            return ReturnDefaultFileValue(fileName);
        }

        try
        {
            string[] lines = File.ReadAllLines(filePath); // Reads all lines from the
file
            foreach (string line in lines)
            {
                string[] values = line.Split(',');
                // Check if the name matches the first column
                if (values.Length > 1 && values[0] == Name)
                {
                    if (int.TryParse(values[1], out int targetValue))
                    {
                        return targetValue; // Return parsed value for HP or ATK
                    }
                }
            }
        }
        catch (Exception ex)
        {
            Debug.LogError($"Error reading {fileName}.csv: {ex.Message}");
        }

        Debug.LogWarning($"Value for {Name} not found in {fileName}.csv.
Using default value.");
        return ReturnDefaultFileValue(fileName);
    }

    private int ReturnDefaultFileValue(string type)
    {
        return type == "atk" ? 10 : 100;
    }
}

```


CoroutineRunner.cs

```
using System.Collections;  
using UnityEngine;
```

```
namespace RPGBattle
```

```
{
```

```
    public class CoroutineRunner : MonoBehaviour
```

```
    {
```

```
        public IEnumerator StartAttackCoroutine(Character character, bool  
isCritical)
```

```
        {
```

```
            yield return StartCoroutine(character.Attack(isCritical));
```

```
        }
```

```
        public IEnumerator StartHealCoroutine(Character character, int amount)
```

```
        {
```

```
            yield return StartCoroutine(character.Heal(amount));
```

```
        }
```

```
        public IEnumerator StartTakeDamageCoroutine(Character character, int  
amount, bool isEventDamage)
```

```
        {
```

```
            yield return StartCoroutine(character.TakeDamage(amount,  
isEventDamage));
```

```
        }
```

```
    }
```

```
}
```

EventHandler.cs

```
using System;
using System.IO;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace RPGBattle
{
    public class EventHandler
    {
        private enum EventType
        {
            CRIT_RATE,
            HEAL_RATE,
            HEAL_AMOUNT,
            DAMAGE_RATE,
            DAMAGE_AMOUNT
        }

        private System.Random random; // using System.Random, since Unity's
Random not random enough
        private bool isCritical;
        private bool isHeal;
        private bool isDamage;
        private List<string> characterNames;
        private List<List<float>> eventData;

        public EventHandler(List<string> _characterNames)
        {
            random = new System.Random();
            isCritical = false;
            isHeal = false;
            isDamage = false;
            eventData = new List<List<float>>();
            characterNames = new List<string>(_characterNames);
            foreach (var name in characterNames)
            {
                eventData.Add(new List<float>
                {
```

```

        LoadEventConfigFromFile("crit_rate", name),
        LoadEventConfigFromFile("heal_rate", name),
        LoadEventConfigFromFile("heal_amount", name),
        LoadEventConfigFromFile("damage_rate", name),
        LoadEventConfigFromFile("damage_amount", name)
    });
}
}

public IEnumerator OnPlayerAttack(Player player, Player enemy)
{
    int whichPlayer = characterNames.FindIndex(x => x ==
player.PlayerCharacter.Name);
    isCritical = random.NextDouble() <
eventData[whichPlayer][EventDataType.CRIT_RATE.GetHashCode()];
    yield return player.Attack(enemy, isCritical);
}

public IEnumerator OnPlayerDefend(Player player)
{
    player.Defend();
    yield return null;
}

/// <summary>
/// random event for player to heal
/// </summary>
/// <param name="player"></param>
public IEnumerator OnPlayerHeal(Player player)
{
    int whichPlayer = characterNames.FindIndex(x => x ==
player.PlayerCharacter.Name);
    isHeal = random.NextDouble() <
eventData[whichPlayer][EventDataType.HEAL_RATE.GetHashCode()];
    if (isHeal)
    {
        yield return
player.Heal((int)eventData[whichPlayer][EventDataType.HEAL_AMOUNT.Get
HashCode()]);
    }
}

```

```

    }

    /// <summary>
    /// random event for player to take damage
    /// </summary>
    /// <param name="player"></param>
    public IEnumerator OnPlayerTakeEventDamage(Player player)
    {
        int whichPlayer = characterNames.FindIndex(x => x ==
player.PlayerCharacter.Name);
        isDamage = random.NextDouble() <
eventData[whichPlayer][EventDataType.DAMAGE_RATE.GetHashCode()];
        if (isDamage)
        {
            yield return
player.TakeDamage((int)eventData[whichPlayer][EventDataType.DAMAGE_A
MOUNT.GetHashCode()], true);
        }
    }

    private float LoadEventConfigFromFile(string fileName, string
characterName)
    {
        string filePath = Path.Combine(Application.streamingAssetsPath,
"EventConfig", fileName + ".csv");

        if (!File.Exists(filePath))
        {
            Debug.LogError($"File {fileName} not found!");
            return 0;
        }
        try
        {
            string[] lines = File.ReadAllLines(filePath); // Reads all lines from the
file
            foreach (string line in lines)
            {
                string[] values = line.Split(',');
                // Check if the name matches the first column
                if (values.Length > 1 && values[0] == characterName)

```

```
        {
            if (float.TryParse(values[1], out float targetValue))
            {
                return targetValue; // Return parsed value for HP or ATK
            }
        }
    }
}
catch (Exception ex)
{
    Debug.LogError($"Error reading {fileName}.csv: {ex.Message}");
}
Debug.LogWarning($"Value for {characterName} not found in
{fileName}.csv");
return 0;
}
}
}
```

HealthBar.cs

```
using UnityEngine.UI;

namespace RPGBattle
{
    public class HealthBar
    {
        public Image healthFillImage; // Reference to the health bar Image

        private int maxHealth;

        public HealthBar(Image _healthFillImage)
        {
            healthFillImage = _healthFillImage;
        }

        public void SetMaxHealth(int maxHealth)
        {
            this.maxHealth = maxHealth;
            healthFillImage.fillAmount = 1f; // Set to full at the start
        }

        public void SetHealth(int currentHealth)
        {
            healthFillImage.fillAmount = (float)currentHealth / maxHealth;
        }
    }
}
```

Player.cs

```
using System.Collections;
using UnityEngine;

namespace RPGBattle
{
    public class Player
    {
        public Character PlayerCharacter { get; set; }
        private CoroutineRunner coroutineRunner;
        private HealthBar healthBar;
        private GameObject shield;

        public Player(Character _character, string _healthBarTag, string _shieldTag,
        CoroutineRunner _coroutineRunner)
        {
            PlayerCharacter = _character;
            GameObject healthBarImg =
        GameObject.FindGameObjectWithTag(_healthBarTag);
            if (healthBarImg == null)
            {
                Debug.LogError($"Health bar image for {_healthBarTag} not
        found!");
            }
            GameObject shieldObject =
        GameObject.FindGameObjectWithTag(_shieldTag);
            if (shieldObject == null)
            {
                Debug.LogError($"Defend image for {_shieldTag} not found!");
            }
            shield = shieldObject;
            shield.SetActive(PlayerCharacter.IsDefend);
            healthBar = new
        HealthBar(healthBarImg.GetComponent<UnityEngine.UI.Image>());
            healthBar.SetMaxHealth(PlayerCharacter.GetMaxHp());
            coroutineRunner = _coroutineRunner;
        }

        public void ResetStatus()
        {
```

```

        PlayerCharacter.ResetStatus();
        healthBar.SetHealth(PlayerCharacter.HP);
        shield.SetActive(PlayerCharacter.IsDefend);
    }

    public IEnumerator Attack(Player enemy, bool isCritical)
    {
        yield return coroutineRunner.StartAttackCoroutine(PlayerCharacter,
isCritical);
        int damage = isCritical ? PlayerCharacter.ATK * 2 :
PlayerCharacter.ATK;
        yield return enemy.TakeDamage(damage, false);
    }

    public void Defend()
    {
        PlayerCharacter.IsDefend = true;
        shield.SetActive(PlayerCharacter.IsDefend);
    }

    public IEnumerator Heal(int amount)
    {
        yield return coroutineRunner.StartHealCoroutine(PlayerCharacter,
amount);
        if (PlayerCharacter.HP > PlayerCharacter.GetMaxHp())
        {
            PlayerCharacter.HP = PlayerCharacter.GetMaxHp();
        }
        healthBar.SetHealth(PlayerCharacter.HP);
    }

    public IEnumerator TakeDamage(int amount, bool isEventDamage)
    {
        yield return
coroutineRunner.StartTakeDamageCoroutine(PlayerCharacter, amount,
isEventDamage);
        shield.SetActive(PlayerCharacter.IsDefend);
        if (PlayerCharacter.HP < 0)
        {
            PlayerCharacter.HP = 0;

```



```
    }  
    healthBar.SetHealth(PlayerCharacter.HP);  
}  
  
public bool IsCharacterDead()  
{  
    return PlayerCharacter.HP <= 0;  
}  
}  
}
```