README - Advance Wars Inspired Game in Unreal Engine 5.4

Descrizione del Progetto

Questo progetto è un gioco strategico a turni ispirato ad *Advance Wars*, sviluppato in *Unreal Engine 5.4* utilizzando il linguaggio *C++*. Il campo di gioco è composto da una griglia di **25x25 tile** (625 tile totali), su cui si muovono e combattono unità appartenenti a due giocatori.

Struttura dei Giocatori

I giocatori sono rappresentati da tre diverse classi:

- **Pawn** (classe base)
- **HumanPlayer** (controllato dall'utente)
- **ComputerPlayer** (controllato dall'AI)
    - **StarComputerPlayer** (variante avanzata dell'AI)

Tutte queste classi ereditano da una classe comune: **Flutter Interface**.

Struttura del Campo di Gioco

Il campo di gioco è una griglia quadrata di **625 tile** (25x25). Ogni **tile** ha uno *status* definito da una `enum class`, che ne determina la condizione:

- **Libera** (può essere occupata da un'unità)
- **Occupata** (già assegnata a un'unità)
- **Ostacolo** (non attraversabile)

Gli ostacoli vengono **spawnati casualmente** sul **20% delle tile** in SpawnRandomObstacles (che sono di 3 tipi , 2 alberi e una montagna) , con un controllo per evitare la creazione di zone inaccessibili (IsGameFieldAccessible).

Selezione della Difficoltà

Prima di iniziare la partita, il giocatore sceglie la difficoltà dell'AI tramite un widget apposito chiamato **ChooseAI**.

Simulazione del Coin Toss

L'inizio della partita è determinato da un **coin toss** simulato fisicamente. La moneta viene lanciata con un **impulso** verso l'alto e una **forza torcente** di intensità casuale per garantire un risultato non prevedibile, l'esito è stabilito confrontando il verso della normale a una faccia della moneta quando quella si assesta .

Gestione delle Tile e Posizionamento

Quando un attore (che sia un **Pawn** o un **ostacolo**) viene posizionato su una tile, ne diventa **automaticamente figlio**, facilitando la gestione della scena.

Gestione dei Turni

La gestione dei turni è affidata alla **Game Mode**. I primi **4 turni** sono dedicati alla **fase di posizionamento delle unità**, gestita dalla funzione `SetUnitPlacement` all'interno della Game Mode.

La Game Mode itera su un **array di players** in cui:

- La posizione `0` è occupata dall'**Human Player**
- La posizione `1` è occupata dal **Player AI**

I giocatori vengono mandati `OnTurn` uno alla volta.

Algoritmi di Movimento e Attacco

Le unità **Brawler** e **Sniper** utilizzano due algoritmi basati su **BFS** per determinare:

- Le tile raggiungibili per il movimento
- Le tile da cui possono attaccare

La logica di **pathfinding** viene gestita nella **Game Field** e utilizza l'algoritmo *A (Astar)** per trovare il percorso ottimale e a costo minimo tra due tile accessibili.

Il movimento effettivo è gestito direttamente da ciascuna unità. La funzione `MoveUnit` imposta il percorso, mentre lo spostamento viene eseguito frame per frame nella funzione `Tick` del **Brawler** e dello **Sniper**.

Intelligenza Artificiale

Il gioco presenta due tipi di AI:

1. **AI Random**:
    - Controlla i propri **Brawler** e **Sniper** tramite un metodo della `GameMode`.
    - Recupera le **tile raggiungibili**.
    - Ne sceglie una **casuale**.
    - Se dalla nuova posizione può attaccare, lo fa; altrimenti passa il turno.
2. ***AI A** (StarComputerPlayer)**:
    - Si muove sempre verso l'avversario utilizzando un *algoritmo A per il percorso ottimale**.
    - Se non trova una posizione da cui attaccare, cerca di avvicinarsi al nemico.
    - Lo **Sniper** della AI cerca sempre di avvicinarsi **quanto basta** per attaccare le unità nemiche, dando priorità a quella con meno vita.
    - Mantiene comunque una distanza di sicurezza per evitare il contraccolpo e per non essere raggiunto facilmente.

Gestione del Combattimento e Condizioni di Vittoria

Quando un'unità subisce danni, il metodo `TakeDamage` verifica se la sua **vita scende sotto zero**:

- Se l'unità muore, viene **rimossa** dall'array di unità del giocatore.

- Viene chiamato il metodo `CheckWinCondition` della `GameMode`, che controlla se uno degli array è vuoto.

- Se un giocatore non ha più unità, la partita termina e l'altro giocatore vince.

Notifica della Vittoria

Alla fine della partita, il vincitore viene notificato tramite un **widget** chiamato **End Game Widget**. Questo mostra il nome del giocatore vincente su uno sfondo che raffigura il campo di battaglia privo di ostacoli e unità.

Widget visibili nel gioco

Nel gioco ci sono 4 widget che appaiono nelle varie fasi del gioco.

ChooseAI : due bottoni per scegliere quale livello di difficoltà della AI

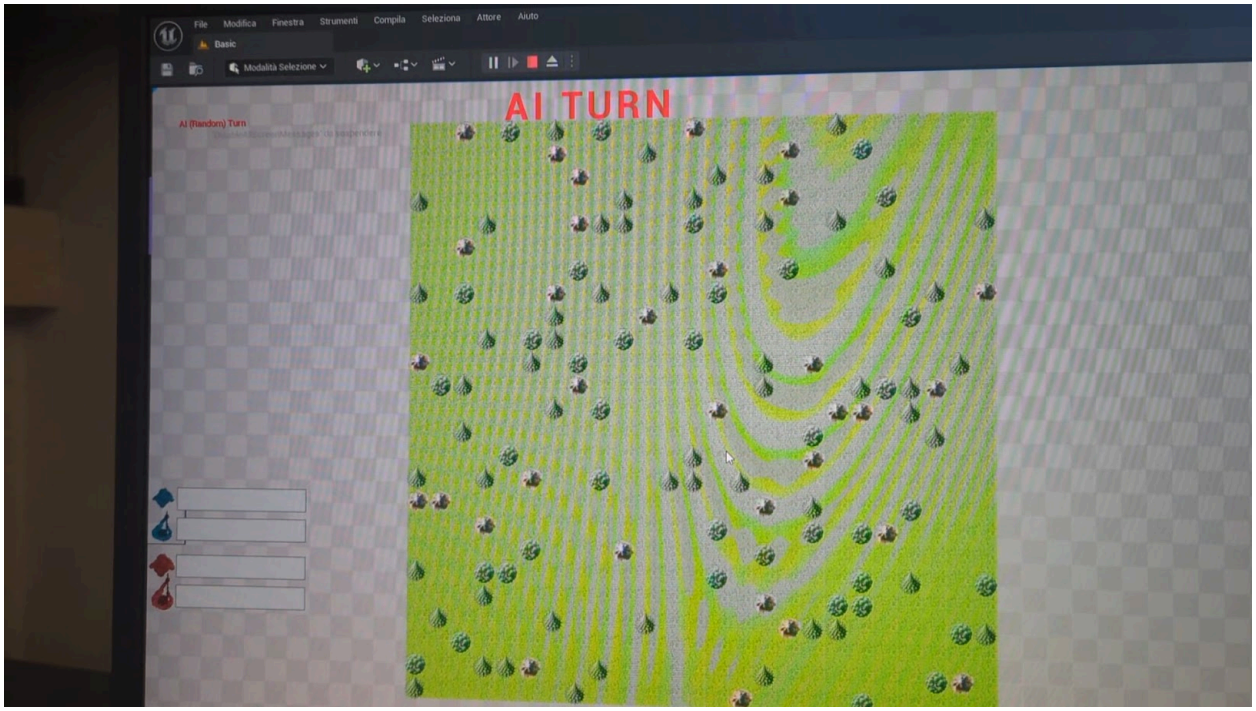
InGameHud : 4 progress bar per visualizzare la vita dei soldatini , parte vuota e viene riempita quando si spawna l'unità sulla griglia, in alto una text block che in blu e rosso comunica il turno del giocatore

HistoryMoves : storico delle mosse in alto a sinistra , una listview con 5 voci al massimo (ogni entry è un widget textblock historyentry) , la piùù recente è inserita in cima mentre la meno a lista piena viene cancellata, si differenziano i player per colore del testo Blu umano, rosso ia

EndGameHud : text block che comunica vincitore (classe c++ ha anche funzioni commentate che prevedevano l'uso di un bottone per ricominciare la partità)

Si consiglia qualora non fosse visibile lo storico delle mosse di ridimensionare la viewport dell'editor di unreal in modo tale che non lo copra .

Schermata iniziale di gioco .



Schermata di gioco duranta il posizionamento.