ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso di Laurea in Ingegneria e Scienze Informatiche

PROGETTAZIONE E SVILUPPO DI UN EDITOR PER L'INTELLIGENZA ARTIFICIALE DI PERSONAGGI IN UN ROGUE-LITE GAME

Elaborato in
PROGRAMMAZIONE AD OGGETTI

Relatore
Prof. ALESSANDRO RICCI

Presentata da
MANUEL JULIO
BUIZO MONTESINOS

Anno Accademico 2022 – 2023

La ricerca e la conoscenza sono intrinsecamente collegate. Più conoscenza hai, meno cerchi una risposta di cui hai bisogno; più ricerca puoi fare, meno conoscenza hai bisogno.

Indice

Introduzione

L'ecosistema dei giochi Rogue-Lite si è affermato come un terreno fertile per sfide avvincenti e esperienze di gioco innovative, grazie alla loro capacità di offrire un equilibrio dinamico tra casualità e strategia. In questo contesto, emerge "FarAfter", un titolo che non solo abbraccia l'essenza dei dungeon procedurali, ma si distingue ulteriormente grazie all'implementazione di un sistema di combattimento a turni profondo e ricco di sfumature tattiche. Durante i combattimenti ogni mossa assume una nuova profondità, diventando un tassello cruciale nel puzzle strategico, contribuendo a plasmare il corso delle battaglie e a definire la strategia vincente.

L'obiettivo principale di questa tesi è affinare ulteriormente l'esperienza di gioco di "FarAfter" mediante l'integrazione di sistemi avanzati di Intelligenza Artificiale all'interno del modello di combattimento a turni. Queste complesse strutture di AI mirano a delineare un confronto tattico raffinato, creando una sinergia fluida tra le azioni pianificate dal giocatore e le risposte adattive dei nemici. Mediante un'analisi approfondita del sistema di combattimento attuale e l'adozione di soluzioni innovative basate sull'AI, il metodo per conseguire l'obiettivo consiste nell'istituire uno strato intermedio che si interpone tra la complessa realizzazione dell'intelligenza artificiale che governa le azioni dei nemici e la delicata concezione del loro design.

Attraverso questa prospettiva, si conferisce al designer di giochi l'intrigante opportunità di plasmare una vasta gamma di strategie, agendo come architetto di un'esperienza ludica unica. In base a questa metodologia, si darà luogo alla creazione di uno spazio dedicato appositamente al designer, il quale acquisirà la facoltà di creare e plasmare l'intelligenza artificiale dei nemici. Tramite questo editor, si potrà darà origine a un'esperienza di gioco che sia in grado di catturare e mantenere l'attenzione del giocatore, offrendo un costante senso di sfida e intrigo, evitando qualsiasi forma di monotonia.

Capitolo 1

Introduzione al Contesto della Tesi: "FarAfter" Rogue-Lite Game

Nel primo capitolo, verrà fornita un'esaustiva panoramica del gioco "FarAfter", esplorando in dettaglio i complessi sistemi di combattimento a turni e le componenti fondamentali di questa esperienza. Un'enfasi particolare sarà posta sull'approfondimento del concetto di giocatore, noto come "Fighter", al fine di delineare in modo esaustivo il ruolo cruciale che tale figura riveste all'interno dell'ecosistema del gioco. In parallelo, verrà offerta un'analisi approfondita del concetto di mossa, esplorando sia le dinamiche interne che il suo significato all'interno del contesto di gioco.

In aggiunta, saranno esaustivamente esaminate le componenti di rilievo che costituiscono il cuore del sistema di combattimento, apportando un livello significativo di complessità e sottigliezza al meccanismo complessivo. Questi elementi, noti come "Stati", "Dot" e "Hot", saranno oggetto di un'analisi dettagliata al fine di comprendere appieno il loro impatto e la loro interazione all'interno dell'ambito del combattimento.

Per concludere, si procederà all'analisi esaustiva della problematica legata alla realizzazione di un Editor di sviluppo completo, all'interno del quale il designer avrà la possibilità di modellare e interagire con tutti i sistemi complessi del gioco. L'obiettivo dell'Editor è la creazione dell'intelligenza artificiale dei nemici in sintonia con il loro design intrinseco. Tale processo richiede l'integrazione sinergica delle sfide sia di natura tecnica che creativa, finalizzate a plasmare una dinamica di gioco coinvolgente e fluida.

1.1 Introduzione

FarAfter rappresenta un gioco Rogue-Lite caratterizzato da robusti tratti tipici dei giochi di ruolo (RPG), il cui scenario è collocato in un mondo post-apocalittico. Al centro del gioco si colloca l'esplorazione di dungeon generati proceduralmente, in cui i giocatori affronteranno sfide impegnative tramite incontri con nemici("Mob") e si immergeranno in coinvolgenti esperienze di combattimento.

1.1.1 Fighters

I Fighters assumono la responsabilità di personificare i protagonisti durante le situazioni di gioco, ognuno dotato di statistiche particolari progettate per il sistema di combattimento a turni adottato. Inoltre, ciascun personaggio presente nel contesto del gioco possiede una serie di abilità esclusive, da utilizzare durante il proprio turno con l'obiettivo di plasmare lo sviluppo del combattimento.

Statistiche

Le statistiche di ciascun Fighters, pur essendo uniformi, sono state attentamente progettate per servire a scopi distinti all'interno del sistema di combattimento, contribuendo in modo specifico e dettagliato alla configurazione dei vari aspetti complessi delle dinamiche di combattimento:

- Vita "Hp": indica il valore della vita corrente del fighter.
- Attacco "Atk": descrive la potenza di un personaggio nell'infliggere danni.
- Difesa "Def": capacità di ridurre i danni subiti dagli attacchi avversari.
- Velocità "Vel": rappresenta la rapidità di agire all'interno del combattimento, influenzando attivamente sull'ordine dei turni.
- Precisione "Pre": riflette la probabilità con cui in fighter riesce a colpire il bersaglio durante un attacco.
- Evasione "Eva": determina la capacità di un personaggio di evitare gli attacchi avversari.
- Critico "Crt": è legata alla probabilità che un attacco possa causare danni superiori, al suo normale danno.

Nel contesto del gioco, si trovano sette personaggi, di cui tre sono controllabili dal giocatore, mentre gli altri quattro avranno a disposizione l'intelligenza artificiale sviluppata nell'ambito di questa tesi:

- Alleati: Suggo, Deezel e Rhaadi
- Nemici: Slyzard, Mob Pesce 1, Mob Pesce 2, Boss Pesce

1.1.2 Mosse

Ciascun personaggio è dotato di mosse comuni e di mosse distintive, le quali sono in perfetta sintonia con il design unico del personaggio stesso e apportano al fighter azioni dal carattere unico e irripetibile. Le mosse dei personaggi costituiscono un insieme diversificato di scelte disponibili a un fighter nel corso del suo turno, spaziando tra attacchi, potenziamenti, indebolimenti e cure. Queste diverse tipologie di azioni possono variare dalla semplicità di una guarigione o un attacco infliggente danni, fino a complessi meccanismi che generano comportamenti articolati, innescando una serie concatenata di azioni intricate. All'interno delle azioni intrise di complessità, emergono le combo, che mediante l'articolata successione di mosse, danno vita a un comportamento inedito.

Bersaglio "Target"

Inoltre, è opportuno evidenziare che ogni singola mossa è dotata di un elemento di fondamentale rilevanza, ovvero la logica del bersaglio, indicata come "Target". Questo aspetto assume un'importanza cruciale poiché stabilisce in maniera altamente specifica e dettagliata le direttive che governano quali sono i personaggi coinvolti nell'azione intrapresa, da parte del fighter nel suo turno.

All'interno delle varie categorie di "Target", è possibile individuare tre principali tipologie in cui possono essere classificate:

- Singolo: comprendono tutte le tipologie di bersaglio con un singolo obbiettivo, possono essere un nemico, se stesso, un alleato, ecc ...
- Gruppo: questa categoria abbraccia tutte i tipi di Target che ahnno come bersaglio un gruppo di Fighters, per esempio tutti i nemici, alleati, e cosi via ...
- Speciali: sono tute i generi di bersagli unici, creati ad hoc per le mosse.

1.2 Design del sistema di combattimento

Nel contesto del sistema di combattimento, la dinamica è regolata da una struttura a turni, in cui la sequenza è determinata dalla statistica di velocità dei personaggi. Un algoritmo altamente responsivo e flessibile non solo gestisce la disposizione temporale dei personaggi nel campo di gioco, ma predice anche l'ordine futuro dei turni. Questo meccanismo assicura che qualsiasi variazione nelle statistiche di velocità si rifletta immediatamente nelle modifiche dell'ordine dei turni.

Le variazioni che si verificano, rappresentando modifiche nello stato attuale del combattimento, vengono gestite attraverso componenti attive, tra cui stati, dot e hot. Questi elementi offrono un'opportunità per apportare modifiche alle statistiche dei personaggi, al danno inflitto durante le azioni e agli effetti derivanti dalle mosse compiute, introducendo al contempo meccanismi unici e peculiari che arricchiscono l'esperienza di gioco.

1.2.1 Componenti: Stati, Dot e Hot

Come già menzionato in precedenza, i concetti di stati, DOT (Damage Over Time) e HOT (Heal Over Time) si configurano come elementi essenziali e attivi all'interno dell'ambito di gioco. Alla base di tutto ciò, tutti questi elementi trovano un contenitore all'interno della mossa, il che implica che la loro attivazione è strettamente legata a essa.

Esplorando una prospettiva di analisi più dettagliata, emergono i concetti specifici di DOT e HOT, entrambi con proprietà che durano nel tempo, i quali rappresentano due aspetti interconnessi di un'unica realtà:

- DOT "Damage Over Time": componente di danno, infligge gradualmente danno ai personaggi nel corso del tempo, seguendo una logica predefinita e ben definita.
- HOT "Heal Ove Time": contrariamente al DOT, l'HOT rappresenta un elemento che gradualmente fornisce cura a un Combattente nel corso del tempo, anch'esso guidato da una logica ben definita e precisa.

1.2.2 Stati

Gli stati rappresentano l'aspetto più dinamico e coinvolgente, caratterizzato da una serie di meccanismi intricati che contribuiscono in modo sostanziale ad arricchire l'esperienza di combattimento. Essi introducono una profondità notevole al gioco, consentendo di modellare in maniera estremamente flessibile la struttura stessa degli elementi in gioco, che spaziano dai Fighters alle loro

mosse, ai DOT, agli HOT, e persino coinvolgendo gli stati stessi. In tal modo, i giocatori si trovano di fronte a un panorama di possibilità praticamente illimitate, in grado di plasmare l'intero sistema di gioco secondo le loro preferenze, tattiche e visione strategica.

Al fine di offrire un'analisi completa del comportamento connesso agli stati, è possibile classificare il comportamento in quattro distinte tipologie di azioni:

- Statistiche: includono tutte le azioni che influiscono sulla variazione del valore di una statistica dei personaggi coinvolti nel combattimento.
- Danno: implica l'alterazione del danno inflitto da parte di un personaggio o, al contrario, può influenzare la quantità di danno subito.
- Attivazione di DOT e HOT: azioni che, secondo a una logica specifica, avvia l'attivazione di particolari effetti DOT e/o HOT.
- Azioni Speciali: si classifica come una categoria di azione che incorpora la capacità di apportare modifiche a qualsiasi componente presente nel contesto di gioco.

Ogni singolo componente, sia esso uno stato, un DOT o un HOT, è caratterizzato da una durata specifica che è misurata attraverso l'avanzamento dei turni nel corso del gioco. Tale durata è determinata da un conto alla rovescia in termini di turni, durante i quali l'effetto permane prima di svanire.

1.3 Problema e scopo

La complessità intrinseca dei sistemi legati al combattimento e le sue componenti, hanno sollevato una sfida cruciale riguardante la creazione di un editor di sviluppo, atto alla creazione di un'intelligenza artificiale per il controllo dei personaggi avversari.

L'ambiente di sviluppo deve essere dotato di una capacità essenziale che gli permetta di interpretare e influire su ciascuno dei molteplici sistemi che costituiscono le intricanti dinamiche del combattimento. Tale necessità sorge dalla volontà di garantire un grado massimo di flessibilità e portabilità a tutto il contesto del combattimento complessivo.

1.3.1 Personalità

Una delle sfide primarie nell'elaborazione dell'editor di sviluppo per l'intelligenza artificiale dei nemici, risiede nel riuscire a creare componenti per l'ambiente, che riescano a conciliare in modo appropriato il personaggio con le

sue mosse personalizzate, concentrando l'attenzione nella selezione delle mosse stesse in modo da riflettere fedelmente la caratterizzazione individuale del personaggio e consentire la trasmissione efficace dei dettagli peculiari a esso associati.

1.3.2 Mosse e Componenti

In aggiunta, acquisisce un'importanza considerevole il compito di elaborare comportamenti standard che derivano sia dalla configurazione stessa della mossa, sia dalle circostanze scaturite da eventi, quali l'attivazione di componenti specifici ("Stati, Dot e Hot") o la mutazione del contesto di gioco.

Questo processo conduce alla necessità di create elementi per l'ambiente, che conducano un'analisi approfondita di ogni singolo evento all'interno del gioco, con l'obiettivo di comprenderne in modo esauriente le dinamiche, e quindi poter adeguatamente adottare tali conoscenze nel comportamento delineato.

1.3.3 Scopo

L'obbiettivo centrale di tale ambiente di sviluppo consiste nell'incorporare al suo interno la capacità di interpretare e reagire all'intero sistema di combattimento e garantire la manipolazione delle principali caratteristiche dei personaggi ("Personalità").

Inoltre, uno dei suoi scopi primari è quello di fornire le basi per la creazione e l'orchestrazione di comportamenti predefiniti, che vengono attivati in risposta a specifici eventi che si verificano durante il combattimento. Allo stesso tempo, si pone l'obiettivo di incorporare anche comportamenti dall'apparenza pseudo-casuale, al fine di garantire la flessibilità necessaria per attribuire una vasta gamma di reazioni e risposte ai personaggi avversari presenti all'interno dell'esperienza di gioco.

Un ulteriore aspetto di considerevole importanza risiede nella sua intrinseca propensione ad agevolare con notevole facilità ulteriori implementazioni ed espansioni, attraverso un sistema che assicura la flessibilità necessaria per estendere il suo raggio d'azione in sintonia con le mutevoli e contingenti necessità che possono emergere.

Queste caratteristiche, a sua volta, si traduce in un potere decisionale esteso ai game-designer, che potranno così esercitare un controllo assoluto sul panorama del gioco. L'obiettivo che guida tale sforzo è la creazione di un'intelligenza artificiale intrinsecamente adattabile, dotata della capacità di modulare dinamicamente le proprie risposte e reazioni di fronte a uno scenario mutevole e a repentini cambiamenti che si verificano all'interno del combattimento.

1.3.4 Validazione

Un altro aspetto di notevole importanza da sottolineare risiede nell'abilità di effettuare con successo un processo di validazione e controllo, al fine di verificare in maniera accurata e dettagliata che l'ambiente di sviluppo stia operando in accordo con le giuste modalità. Questa pratica non solo contribuisce a conferire un grado di solidità e robustezza fondamentale al processo di sviluppo, ma ne amplifica anche le potenzialità di espansione e crescita.

In questo modo, si libera da qualsiasi apprensione legata all'operatività dell'ambiente, consentendo di focalizzare le attenzioni esclusivamente sulla definizione e realizzazione del design desiderato, senza essere intralciati dalle dinamiche del suo comportamento interno.

Pertanto, si procederà con la creazione e l'implementazione di un sistema dedicato, il quale assumerà il ruolo cruciale di partecipare attivamente al processo di convalida e controllo. Questo sistema, concepito con un fine preciso, sarà volto a supervisionare con meticolosa attenzione sia l'ambiente di sviluppo nella sua interezza, sia le sue complesse componenti interconnesse. In tal modo, si mira a garantire che ogni aspetto del processo sia sottoposto a una rigorosa verifica, affinché l'integrità operativa e la coerenza funzionale dell'intero sistema siano preservate.

Capitolo 2

Analisi dei Potenziali Algoritmi di AI Applicabili

In questo capitolo, verrà eseguita un'analisi dettagliata dei principali algoritmi di intelligenza artificiale, consentendo una panoramica approfondita dei loro meccanismi e degli obiettivi che perseguono. Questo studio consentirà di tracciare le molteplici direzioni prese dall'AI nell'ambito dei videogiochi, aprendo una via alla comprensione delle sfide e delle opportunità coinvolte nel loro processo di sviluppo e perfezionamento.

In aggiunta, va sottolineato che la presente ricerca si propone di condurre un'analisi approfondita, orientata a conseguire una conoscenza completa e dettagliata. Tale conoscenza è mirata a identificare e catalogare le varie opzioni di algoritmi nell'ambito dell'intelligenza artificiale, mettendo in luce le loro peculiarità e potenzialità. Parimenti, l'indagine si estende all'apprendimento di strategie altamente efficaci per l'integrazione sinergica e proficua di tali algoritmi all'interno di un editor di sviluppo.

Questo processo di integrazione, governato da una profonda comprensione delle dinamiche coinvolte, assume un ruolo fondamentale nella realizzazione dell'intelligenza artificiale destinata a governare i comportamenti dei personaggi avversari del gioco "FarAfter".

2.1 Decision Tree

Il Decision Tree [6] [11] è tra le più semplici tecniche decisionali da implementare nei video giochi, ampiamente utilizzata per controllare il comportamento dei personaggi e altri processi di scelta. Un albero decisionale è composto da punti decisionali collegati, chiamati scelte o nodi, traendo conoscenza sullo stato attuale del gioco, del personaggio e sulle possibili azioni che quest'ultimo può compiere, l'obbiettivo del processo decisionale è di analizzare ed elaborare, le possibili scelte a disposizione, per richiedere l'azione più appropriata. Dal lato teorico, l'albero decisionale è un grafo formato da una struttura gerarchica di nodi, che consiste di un nodo radice, di rami, nodi interni e nodi foglia.

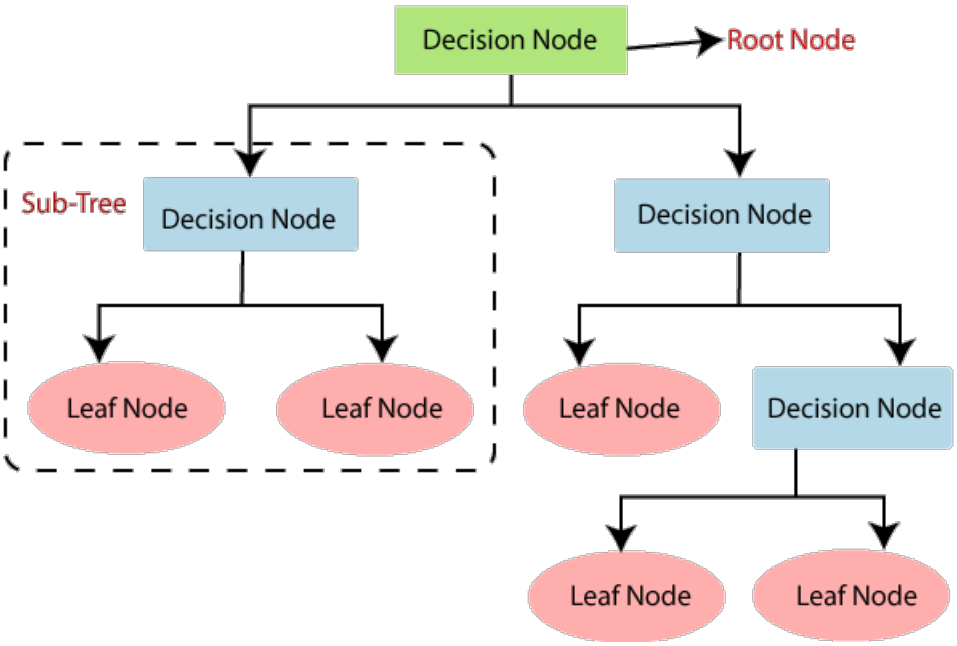

Figura 2.1: Esempio di albero decisionale

Per analizzare meglio il processo decisionale, si riporta un esempio generale delle decisioni che deve prendere un personaggio dentro un gioco di guerra. Mostrando per ogni nodo le possibili scelte date, dalla conoscenza sullo stato attuale del gioco e le possibili azioni intelligenti che può compiere durante il combattimento.

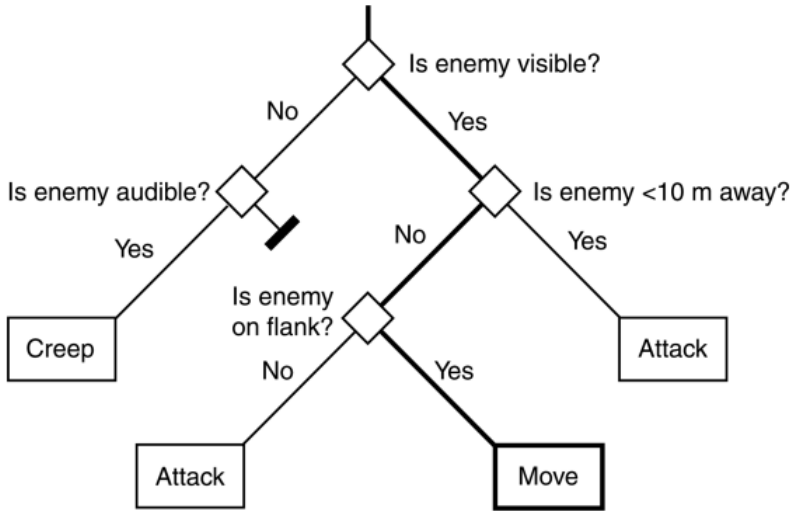

Figura 2.2: Esempio di un processo decisionale

In questo caso come riportato in figura, il personaggio potrà scegliere di attaccare a meno che non riesca a vedere il nemico o sia affiancato. Il percorso preso, evidenzia la singola azione che verrà eseguita dal personaggio, dato dalle conoscenze attuali del gioco.

Struttura dell'albero

Per strutturare al meglio un albero decisionale e avere la massima efficienza, occorre equilibrarlo, mantenendo all'incirca lo stesso numero di foglie su ciascun ramo. In genere ogni nodo deve eseguire semplici test ma non sempre è cosi, ci sono decisioni che richiedono molto tempo per essere eseguite. La strategia più comune è quella di mettere in ritardo le decisioni più costose, verificando prima se sono assolutamente necessarie. E accorciare i rami usati comunemente rispetto a quelli usati raramente. Riguardo ai test logici, ponendo i nodi in serie nell'albero, si rendono possibili qualsiasi combinazione logica tra i nodi, come AND e OR.

Performance

La performance degli alberi decisionali è lineare con il numero di nodi visitati, assumendo che per ogni decisioni richieda un tempo costante e che l'albero sia ben bilanciato, allora le prestazioni dell'algoritmo sono O(log2 n), dove n è il numero di nodi decisionali. Nei peggiore dei casi è O(n).

2.2 State Machine

Una state Machine [6] [4] [12] è una rappresentazione di un sistema reattivo basato su eventi, che attivano la transizione da uno stato all'altro, solo nel caso in cui venga soddisfatta la condizione, che controlla il cambiamento. I dati presi in considerazione tengono conto, sia della conoscenza del mondo che li circonda che della loro composizione interna. Normalmente le azioni o i comportamenti sono associati con ogni stato, quindi finché lo stato è attivo, le azioni da svolgere saranno solo le medesime di quello stato. L'obbiettivo della State Machine mira a creare un sistema generale, che supporti macchine a stato arbitrario con qualsiasi condizione di transizione, occupando solo uno stato alla volta.

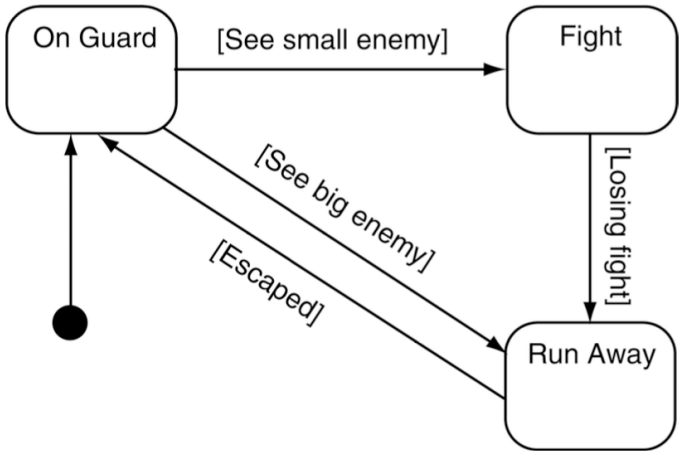

Figura 2.3: Esempio di state machine

La figura mostra una semplice macchina di stato con tre stati: On Guard, Fight e Run Away. Si noti che ogni stato ha il proprio insieme di transizioni. In una macchina di stato, vengono considerate solo le transizioni dallo stato corrente, quindi non tutte le azioni possono essere raggiunte in un momento particolare: solo lo stato corrente e i suoi vicini.

Performace

L'algoritmo della State Machine richiede solo memoria per contenere una transizione attivata e lo stato corrente. È O(1) in memoria e O(m) nel tempo, dove m è il numero di transizioni per stato. L'algoritmo chiama altre funzioni sia nello stato che nelle classi di transizione, e nella maggior parte dei casi il tempo di esecuzione di queste funzioni rappresenta la maggior parte del tempo trascorso nell'algoritmo.

State Machine combinata al Decision Tree

L'implementazione delle transizioni ha più di una somiglianza passeggera con l'attuazione degli alberi decisionali. Gli alberi decisionali sono un modo efficiente per abbinare una serie di condizioni, e questo ha un'applicazione nelle macchine di stato per le transizioni di corrispondenza. Possiamo combinare i due approcci sostituendo le transizioni da uno stato con un albero decisionale. Le foglie dell'albero, piuttosto che essere azioni come prima, sono transizioni verso nuovi stati.

2.3 Naive Bayes Classifiers

Naive Bayes [6] [13] [7] è un algoritmo di apprendimento automatico supervisionato, utilizzato per attività di classificazione, fondato sul teorema di Bayes. I classificatori costruiti dall'algoritmo generano modelli che assegnano etichette di classe a istanze di problemi, rappresentate come vettori di valori di caratteristiche, in cui le etichette di classe sono tratte da un insieme finito. Questo approccio si basa sul presupposto che il valore di una particolare caratteristica sia indipendente dal valore di qualsiasi altra caratteristica, data la variabile di classe, consentendo all'algoritmo di effettuare previsioni in modo rapido e accurato. Uno dei vantaggi di Naive Bayes è che richiede solo un piccolo numero di dati di addestramento per stimare i parametri necessari per la classificazione.

Esempio

Attraverso un esempio si andrà a spiegare meglio i classificatori di Naive Bayes. Si supponga che un personaggio AI, impari lo stile di un giocatore che gira intorno alle curve, dove per semplicità si analizza quando il giocatore decide di rallentare in base solo alla sua velocità e distanza da un angolo.

brake?	distance	speed
Y	2.4	11.3
Y	3.2	70.2
N	75.7	72.7
Y	80.6	89.4
N	2.8	15.2
Y	82.1	8.6
Y	3.8	69.4

Figura 2.4: Valori iniziali del giocatore

La prima cosa da inquadrare sono i dati, è importante rendere i modelli dei dati il più evidenti possibile, cosi da facilitare l'apprendimento. Guardando i valori in tabella, emergono alcuni schemi chiari, i giocatori sono vicini o lontani dall'angolo e vanno veloci o lenti. Quindi si andrà ad etichettare ciascun valore, le distanze inferiori a 20.0 come "vicino" e "lontano" altrimenti. E per le velocità inferiori a 10.0 sono considerate "lente", altrimenti "veloci". Questo porta alla seguente tabella di attributi discreti binari:

brake?	distance	speed
Y	near	slow
Y	near	fast
N	far	fast
Y	far	fast
N	near	slow
Y	far	slow
Y	near	fast

Figura 2.5: Valori iniziali etichettati

Ora risulta molto più facile anche per un essere umano, vedere le connessione tra i valori degli attributi e le scelte di azione. Questo renderà l'apprendimento veloce e non richiederà troppi dati, ovviamente in un esempio reali ci saranno molti più dati da considerare e i modelli potrebbero non essere cosi evidenti. Ma la conoscenza del gioco rende abbastanza facile sapere come semplificare le cose. In questo esempio si vuole imparare la probabilità condizionale che un giocatore decida di frenare data la sua distanza e velocità verso un angolo. La formula per questo è:

$$P(brake?|distance, speed)$$

Il prossimo passo è applicare la formula di Bayes:

$$P(A|B) = \frac{P(B|A)*P(A)}{P(B)}$$

Il punto importante della regola di Bayes è che ci permette di esprimere la probabilità condizionale di A dato B, in termini di probabilità condizionale di B dato A. Prima di applicarla, si andrà a riaffermare leggermente la formula di Bayes come:

$$P(A|B) = \alpha * P(B|A) * P(A)$$

Dove $\alpha = \frac{1}{P(B)}$, risulta essere più facile da usare per i calcolo.
Ecco la versione della regola Bayes applicata all'esempio:

$$P(brake?|distance, speed) = \alpha * P(distance, speed|brake?) * P(brake?)$$

In seguito si applica un'assunzione di indipendenza condizionale per dare:

$$P(distance, speed|brake?) = P(distance|brake?) * P(speed|brake?)$$

Unendo queste due formule nella regola di Bayes e l'assunzione ingenua dell'indipendenza condizionale dà la seguente formula finale:

$$P(brake?|distance, speed) = \\ \alpha * P(distance|brake?) * P(speed|brake?) * P(break?)$$

Il punto focale di questa versione finale della formula, è di poter usare i valori nella tabella presentata precedentemente, per cercare varie probabilità. Per vedere come si analizzerà il caso in cui, un personaggio AI cerchi di decidere se frenare o meno, in una situazione in cui la distanza dalla curva è 79.2 e la sua velocità è 12.1. Si vuole calcolare la probabilità condizionale che un giocatore umano freni nella stessa situazione e usarla per prendere la decisione del personaggio AI. Date solo due possibilità, o frenare o no, si prende in considerazione ognuna a turno. Per prima cosa, si calcola la probabilità di frenata:

$$P(brake? = Y|distance = 792, speed = 121)$$

Etichettando i valori dati in input e usando la formula derivata in precedenza, si trova:

$$P(brake? = Y|far, slow)$$

$$P(brake? = Y|far, slow) = \alpha * P(far|brake? = Y) * P(slow|brake? = \\ Y) * P(break? = Y)$$

Osservando la tabella, si contano che per 5 casi in cui le persone stavano frenando, ci sono 2 casi in cui erano lontani. Quindi si stimano:

$$P(far|brake? = Y) = \frac{2}{5}$$

Allo stesso modo, si contano 2 casi su 5 in cui le persone hanno frenato mentre andavano a bassa velocità, risulta:

$$P(slow|brake? = Y) = \frac{2}{5}$$

Ancora una volta dalla tabella, in totale c'erano 5 casi su 7 quando la persona stava frenando, per dare:

$$P(brake? = Y) = \frac{5}{7}$$

Questo valore è noto a priori poiché rappresenta la probabilità di frenare, prima di qualsiasi conoscenza della situazione attuale. Un punto importante sul precedente è che se un evento è intrinsecamente improbabile, allora il precedente sarà basso. Pertanto, la probabilità complessiva, dato ciò che si sa sulla situazione attuale, può essere ancora bassa. Tornando all'esempio della frenata, si può mettere insieme tutti questi calcoli per calcolare la probabilità condizionale che un giocatore umano freni nella situazione attuale:

$$P(brake? = Y|far, slow) = \alpha * \frac{4}{35}$$

Il valore di α risulta non essere importante, poiché calcolando la probabilità di non frenare, emerge:

$$P(brake? = N|far, slow) = \alpha * \frac{1}{14}$$

Ed è il motivo per cui α risulta non necessaria, è perché si annulla:

$$\alpha * \frac{4}{35} > \alpha * \frac{1}{14} => \frac{4}{35} > \frac{1}{14}$$

Quindi, calcolando la probabilità della frenata e della non frenata, risulta maggiore quella di frenare. Pertanto se il personaggio AI, volesse comportarsi come gli umani da cui abbiamo raccolto i dati, allora dovrebbe frenare.

Criticità del Naive Bayes

Uno dei punti critici dell'applicare il classificatore di Naive Bayes è la moltiplicazione di piccoli numeri (come probabilità), che a causa della precisione finita della virgola mobile, perdono rapidamente precisione e alla fine diventano zero. Una delle strategie per risolvere il problema è rappresentare tutte le probabilità, come logaritmi e poi, invece di moltiplicare, sommare.

2.4 Decision Tree Learning

Gli alberi decisionali suddetti sono una serie di decisioni che generano un'azione da intraprendere sulla base di una serie di osservazioni. Questi ultimi possono essere maggiormente di due tipi: molto specifici o molto generici, i primi prendono decisioni basate sull'intricato dettaglio delle loro osservazioni, mentre i secondi sono poco profondi, con pochi rami e danno comportamenti ampi e generali.

I decision tree learning [6] [5] [2] abbattono questa inefficienza, i quali possono essere costruiti dinamicamente da una serie di considerazioni e azioni fornite attraverso una forte supervisione. Esistono diversi algoritmi di apprendimento degli alberi decisionali, ma si andrà ad analizzare quelli più diffusi nell'ambito video-ludico, tra cui:

- ID3
- ID4

ID3 di Quinlan

ID3 sta per "Inductive Decision Tree Algorithm 3" o "Iterative Dichotomizer 3". Inventato da Ross Quinlan, ID3 è un algoritmo di apprendimento dell'albero decisionale, semplice da implementare e relativamente efficiente. Come per qualsiasi altro algoritmo ha tutta una serie di ottimizzazioni utili in diverse situazioni, più generalmente ID3 viene utilizzato solo per problemi di classificazione con caratteristiche nominali.

In statistica, i dati nominali [8] servono per non fornire alcun valore quantitativo, etichettando le variabili. A differenza dei dati quantitativi, i valori nominali non possono né essere ordinati, né misurati, inoltre data la loro caratteristica non possono essere manipolati usando gli operatori matematici, ma l'unica misura per tali dati è la moda.

Pertanto partendo da un data-set di dati nominali, ID3 segue un approccio greedy dall'alto verso il basso, in parole semplici, la creazione dell'albero inizia dall'alto, iterando progressivamente fino al raggiungimento delle foglie. Per spiegare al meglio l'approccio greedy, si andrà a dare un significato di guadagno d'informazione (Information Gain) e entropia (Entropy), per risultare il meno forviante possibile.

- Entropia:

  Quest'ultima è la misura del disordine, l'entropia di un set di dati nominali è la misura del disordine nella caratteristica target del set di dati. Cioè misura i dati nominali di una serie di esempi, nel caso di una classificazione arbitraria, se tutti gli esempi hanno la stessa azione, l'entropia sarà 0. Ma se le azioni sono distribuite uniformemente, allora l'entropia sarà 1.

  $$Entropia(S) = -\sum p_i * log_2(p_i)$$

  - S è il nostro set di dati.
  - n è il numero totale di classi nella colonna di target.
  - p_i è la probabilità della classe 'i' o il rapporto tra " numero di righe con classe i nella colonna target" al " numero totale di righe" nel set di dati.

  Nei sistemi dove non si ha un logaritmo di base 2 dedicato, possiamo cambiare la base, in quella che ottimizza al meglio il processo (tipicamente, la base e è più veloce), con la seguente formula:

$$log_a(b) = \frac{log_c(b)}{log_c(a)}$$

- Guadagno d'informazione:

  E' semplicemente la riduzione dell'entropia complessiva, si può pensare che le informazioni in un insieme siano il grado in cui l'appartenenza all'insieme determina l'output. Se si ha una serie di esempi con tutte le azioni diverse, allora essere nel set non ci dice molto su quale azione intraprendere. Idealmente, si vuole raggiungere una situazione in cui essere in un set ci dice esattamente quale azione scegliere.

$$IG(S, A) = Entropia(S) - \sum((\frac{|S_v|}{|S|}) * Entropia(S_v))$$

  - dove S_v è l'insieme delle righe in S per le quali la colonna target ha valore v
  - $|S_v|$ è il numero di righe in S_v e similmente $|S|$ è il numero di righe in S.

Quindi l'approccio greedy sta a significare, che per ogni iterazione al momento della creazione del nodo, verifica la caratteristica migliore calcolando il guadagno d'informazione data dalla riduzione dell'entropia e misurando la capacità della caratteristica di dividere il data-set, riducendo sempre di più il nostro set di dati ad ogni iterazione. La caratteristica con il più alto guadagno d'informazione sarà la migliore scelta.

Esempio pratico ID3

Si andrà a dipanare l'algoritmo ID3 con un esempio pratico, mostrando come costruire l'albero decisionale per un personaggio in guerra. In primis si mostra il data-set S di partenza:

Attributo	Attributo	Attributo	TARGET
Healthy	In Cover	With Ammo	Attack
Hurt	In Cover	With Ammo	Attack
Healthy	In Cover	Empty	Defend
Hurt	In Cover	Empty	Defend
Hurt	Exposed	With Ammo	Defend

Figura 2.6: Data-set di dati nominali

Ci sono due possibili esiti, attaccare e difendere, partendo dall'attributo target, si calcola l'entropia del data-set(S), dato da:

$$Entropia(S) = -\sum p_i * log_2(p_i)$$
$$Entropia(S) = -[(p_A * log_2(p_A)) + (p_D * log_2(p_D))]$$
$$0.9709505945 = -[(\tfrac{2}{5} * log_2(\tfrac{2}{5})) + (\tfrac{3}{5} * log_2(\tfrac{3}{5}))]$$

Al primo nodo l'algoritmo esamina ogni possibile attributo a turno, divide l'insieme del data-set e calcola l'entropia associata ad ogni divisione, per semplicità si andrà a sviluppare un singolo attributo, mostrando il risultato finale per i restanti.

$$Entropia(Healthy) = -[(p_A * log_2(p_A)) + (p_D * log_2(p_D))]$$
$$1 = -[(\tfrac{1}{2} * log_2(\tfrac{1}{2})) + (\tfrac{1}{2} * log_2(\tfrac{1}{2}))]$$

$$Entropia(Hurt) = -[(p_A * log_2(p_A)) + (p_D * log_2(p_D))]$$
$$0.9182958341 = -[(\tfrac{1}{3} * log_2(\tfrac{1}{3})) + (\tfrac{2}{3} * log_2(\tfrac{2}{3}))]$$

I calcoli dimostrano che l'entropia dell'attributo "Healthy" risulta 1, perché le azioni (attack e defense) sono distribuite uniformemente.

Health	$E_{\text{healthy}} = 1.000$	$E_{\text{hurt}} = 0.918$
Cover	$E_{\text{cover}} = 1.000$	$E_{\text{exposed}} = 0.000$
Ammo	$E_{\text{ammo}} = 0.918$	$E_{\text{empty}} = 0.000$

Figura 2.7: Risultati dell'entropia dei vari attributi

Calcolata l'entropia del data-set iniziale e degli attributi nelle rispettive data-set figli, si andrà a determinare il guadagno di informazioni (Information Gain) che per ogni divisione è la riduzione dell'entropia dall'attuale date-set (0.971), alle entropie dei data-set figli. Per non essere prolissi, si esaminerà l'attributo "ammo" e saranno riportati i risultati dei restanti.

- $|S|$: Sono le righe totali del data-set S
- $|S_v|$: Sono il numero di righe dove l'attributo v compare nel data-set S.
- $Entropia(S_v)$: è il valore dell'entropia dell'attributo v, data dal data-set figlio di S.

$$IG(S, A) = Entropia(S) - \sum((\tfrac{|S_v|}{|S|}) * Entropia(S_v))$$
$$IG(S, Ammo) = Entropia(S) - [(\tfrac{|S_{withAmmo}|}{|S|} * Entropia(S_{withAmmo})) + (\tfrac{|S_{empty}|}{|S|} * Entropia(S_{empty}))]$$
$$0.4201505945 = 0.9709505945 - [(\tfrac{3}{5} * 0.918) + (\tfrac{2}{5} * 0)]$$

$$G_{\text{health}} = 0.020$$
$$G_{\text{cover}} = 0.171$$
$$G_{\text{ammo}} = 0.420$$

Figura 2.8: Risultati del guadagno di informazioni dei vari attributi

Trovati i valori del guadagno di informazioni dei tre attributi, le munizioni ("Ammo") risultano il miglior indicatore di quale azione intraprendere (dal momento che non si può attaccare senza munizioni). Con il principio di imparare prima le cose più generali, si usano le munizioni come primo ramo dell'albero delle decisioni.

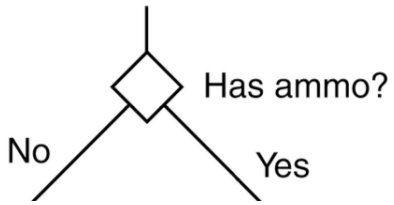

Figura 2.9: Creazione del primo nodo di decisione

Dopo aver trovato il primo nodo di decisione, si itera di nuovo il calcolo dell'entropia e del guadagno di informazioni, però dividendo il data-set di partenza in due insiemi. Un insieme contenente solo i record che hanno l'attributo "With Ammo" S_A, mentre il secondo insieme avente solo i record con attributo "Empty" S_E. In questo modo si analizzerà per entrambi i data-set, quale attributo ha il miglior guadagno di informazioni.

S_A

Healthy	In Cover	With Ammo	Attack
Hurt	In Cover	With Ammo	Attack
Hurt	Exposed	With Ammo	Defend

S_E

Healthy	In Cover	Empty	Defend
Hurt	In Cover	Empty	Defend

Figura 2.10: Divisone del data-set di partenza

Si calcola ora l'entropia e il guadagno di informazioni entrambi i data-set:

- Data-set S_A

$$E(S_A) = 0.9182958341$$
$$E(Healthy) = 0 \qquad E(Hurt) = 1 \qquad IG(Health) = 0.2516291674$$
$$E(InCover) = 0 \qquad E(Exposed) = 0 \qquad IG(Cover) = 0.9182958341$$

- Data-set $S_E = 0$

$$E(Healthy) = 0 \quad E(Hurt) = 0 \quad IG(Health) = 0$$
$$E(InCover) = 0 \quad E(Exposed) = 0 \quad IG(Cover) = 0$$

Si noti che l'albero generato dalla seconda iterazione ha una foglia, contenente l'azione difesa ("defend"). Questo risultato è dato dal data-set S_E, come suggeriscono i dati dell'entropia e del guadagno di informazione, è pari a 0. Questo porta a creare il nodo foglia con l'azione difesa ("defense").

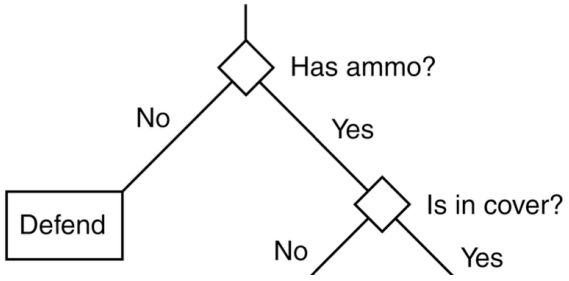

Figura 2.11: Fine della seconda iterazione dell'algoritmo ID3

Continuando nell'iterazione, si divide il data-set S_A in due insiemi rispetto agli attributi "InCover" e "Exposed". Guardando meglio il data-set, si osserva che in un insieme risulteranno solo le azioni attacco ("attack"), mentre nel secondo solo difesa ("defend"). Questo ci porterà a creare due nodi fogli con le azioni descritte.

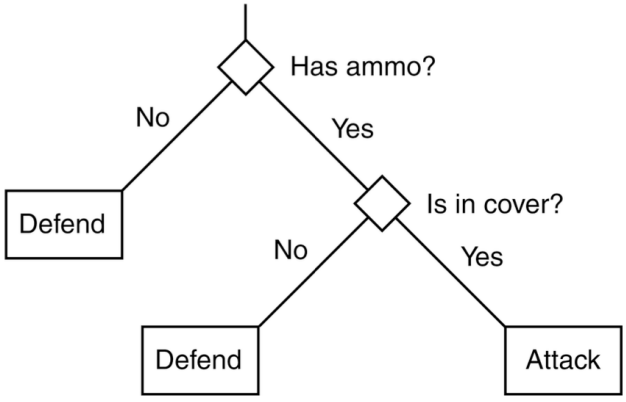

Figura 2.12: Creazione dell'albero decisionale ID3

In fine si noti che la salute del personaggio non è presente nell'albero, dal data-set analizzato, emerge che tale attributo non è rilevante per la decisione. Analizzato un insieme di dati differenti, si potrebbe incontrare una situazione in cui è rilevante, per cui l'albero delle decisioni lo userebbe.

Capitolo 3

Ideazione di un Editor per l'IA

All'interno di questo capitolo, verrà esaminato con particolare rigore l'approccio risolutivo che si prefigge di plasmare un editor di sviluppo ottimale, mirato a favorire la creazione dell'intelligenza artificiale dei nemici nel contesto di gioco. Questo processo implicherà una sinergia profonda tra sistemi complessi, integrati in maniera sinuosa all'interno delle dinamiche di combattimento, con l'obiettivo principale di garantire un elevato livello di manovrabilità e controllo.

In questo studio, verranno analizzate approfonditamente le strategie e le ragioni che hanno spinto l'ambiente di sviluppo a concentrarsi sull'ideazione di un editor appositamente progettato per la creazione di alberi decisionali. Tale editor sarà affiancato da strumenti di modifica progettati con l'obiettivo di semplificare il processo di creazione stessa. Il fine principale di quest'editor è abilitare la generazione di alberi decisionali, destinati a essere impiegati nell'ambito dell'intelligenza artificiale dei nemici, nell'ottica di potenziarne le capacità decisionali in maniera significativa.

Inoltre, si procederà all'integrazione nell'editor di un sistema di personalità, affiancato da strategie volti a modulare questa personalità in risposta al livello di vita del personaggio, nel corso del combattimento. L'obiettivo ultimo è la creazione di un design caratteriale articolato e dinamico per il personaggio stesso.

3.1 Approccio al problema: Ambiente di sviluppo

All'interno di questa sezione, si condurrà un'analisi approfondita allo scopo di individuare quale tra i diversi algoritmi di intelligenza artificiale utilizzati nei giochi, visti nel capitolo precedente, risulti particolarmente adeguato per adattarsi in maniera ottimale alle complesse dinamiche del sistema di combattimento in questione.

Successivamente, si procederà all'esame delle strategie volte a espandere e potenziare l'ambiente di sviluppo, al fine di conferirgli una natura scalabile che possa consentirgli di integrarsi in modo ottimale sia con gli attuali sistemi di combattimento che con quelli che verranno sviluppati in futuro.

3.1.1 Scopo dell'Editor: Decision Tree

L'obiettivo dell'editor consiste nel creare un'intelligenza artificiale in grado di guidare le decisioni dei nemici del gioco. Tale sistema sarà progettato con l'obiettivo primario di consentire ai nemici di compiere scelte ponderate e strategiche, basate su un'analisi approfondita delle molteplici variabili in gioco, al fine di determinare con precisione le azioni da intraprendere.

In aggiunta, è imperativo che l'editor di sviluppo sia strutturato in modo da agevolare l'emanazione di decisioni predeterminate, le quali vengono attivate da specifici eventi nel contesto del combattimento. Ciò si tradurrà in un completo dominio sulla selezione delle mosse avviate dal nemico, consentendo un controllo totale e mirato su tali dinamiche. Oltre a ciò, è essenziale che l'editor sia in grado di rispondere prontamente a qualsiasi evento verificantesi durante il combattimento, al fine di abilitare la formulazione di decisioni specifiche in relazione a tali eventi.

Tra le diverse metodologie algoritmiche esaminate, ovvero Decision Tree, State machine, Naive Bayes Classifiers e Decision Tree Learning, l'algoritmo che si dimostra più consono alle esigenze di sviluppo di un'intelligenza artificiale per i nemici risulta essere il Decision Tree. Questo algoritmo consente di costruire strutture ad albero, all'interno delle quali vengono definiti sia i nodi decisionali che quelli corrispondenti alle azioni da intraprendere, fornendo così uno spazio di lavoro altamente adattabile e configurabile.

Gli algoritmi che incorporano il concetto di state machine risultano inadeguati per concorrere agli obiettivi prefissati dall'editor, poiché la natura stessa di una state machine si fonda su un sistema reattivo che si attiva in risposta a eventi specifici, i quali innescano transizioni tra diversi stati.

Il Naive Bayes Classifiers costituisce un algoritmo di apprendimento automatico che, attraverso un esiguo insieme di dati di addestramento in ingresso,

stima i comportamenti da adottare. Tuttavia, questa metodologia elimina la prospettiva di creare comportamenti predefiniti per gli eventi durante il combattimento, sottraendo così una componente essenziale all'obiettivo perseguito dall'editor.

Nell'ambito dei metodi di apprendimento automatico, il Decision Tree Learning, pur facendo parte della medesima famiglia, non si allinea con lo scopo precedentemente descritto. In aggiunta, questo algoritmo consente la generazione di alberi decisionali dinamici tramite l'analisi dei guadagni informativi ottenuti dai dati in forma tabulare. Tuttavia, tale approccio non consente una modellazione personalizzata degli alberi secondo preferenze specifiche, andando a minare il fulcro stesso dell'intento dell'editor.

Di conseguenza, la focalizzazione dell'ambiente di sviluppo deve essere posta sulla creazione di alberi decisionali destinati a governare le azioni dei nemici nel contesto del combattimento, arricchendo la loro struttura con l'obiettivo di renderla facilmente espandibile e adattabile alle diverse esigenze.

Nelle sezioni successive, verranno esaminate le strategie volte a configurare un editor di sviluppo idoneo alla creazione di alberi decisionali, nonché le metodologie finalizzate ad arricchire la struttura di tale ambiente e a potenziarne la capacità di adattamento e espansione.

3.1.2 Modello dell'Editor

Nel contesto della formulazione di un editor per alberi decisionali, risulta imprescindibile esaminare le varie componenti che possono essere integrate al suo interno. Questa analisi consente di acquisire una comprensione approfondita delle capacità operative e delle possibilità di ampliamento connesse a ciascuna di tali componenti.

L'algoritmo del decision tree, descritto nel capitolo precedente, espone tre componenti di rilevanza: il nodo principale ("Root"), un nodo decisionale e un nodo d'azione. Partendo da questi tre elementi chiave, si esaminerà in dettaglio il loro funzionamento per poter affinarne la struttura stessa e ampliare le opportunità di sviluppo dell'editor. Nel perseguire questo obiettivo, si manterrà salda l'intenzione originaria dell'algoritmo. L'obiettivo finale è plasmare un editor con un'architettura intrinsecamente predisposta alla scalabilità, in grado di adattarsi alle esigenze in evoluzione.

Per l'obiettivo descritto precedentemente, si procede all'analisi dettagliata degli scopi intrinseci di ciascuna componente. Tale esame mira a una scomposizione delle responsabilità attribuite a ciascun elemento, al fine di ottenere una comprensione approfondita delle loro dinamiche interne, riflettendo così uno uno dei principi "SOLID" [3] della progettazione, il "Single Responsibility Principle"(SRP) [3]:

- Nodo Radice: come indicato nell'algoritmo originale, il nodo radice è semplicemente un punto di inizio da cui si avvia l'intero albero decisionale.

- Nodo Azione: in modo analogo, il nodo d'azione si dedica esclusivamente all'esecuzione di un'azione singola, selezionata all'interno del percorso dell'albero decisionale al momento della sua attuazione, attivato dal nodo decisionale.

- Nodo Decisione: come indicato dall'algoritmo del decision tree, al nodo di decisione è affidato la responsabilità di prendere decisioni in base a condizioni valutate. Questo processo conduce a percorrere un ramo decisionale specifico, al fine di eseguire l'azione desiderata che discende da tale decisione.

L'analisi delle componenti, come precedentemente esposta, offre un'ampia disamina delle rispettive responsabilità. È evidente che sia il nodo radice che il nodo d'azione presentano una singola responsabilità, il che rende tali elementi non suscettibili di ulteriori decomposizioni in parti distinte.

Diversamente, nel contesto del nodo di decisione si manifestano due distinte responsabilità: il primo coinvolge l'analisi delle condizioni, mentre il secondo comprende l'elaborazione individuale di ciascuna condizione particolare. È evidente che la struttura del nodo può essere disassemblata in diverse componenti, una scelta che si dimostra preziosa nell'ottica di arricchire l'articolazione dell'albero decisionale stesso.

Attraverso l'applicazione del principio di progettazione noto come 'Common Closure Principle' (CCP) [3], è stato effettuato uno specifico intervento di suddivisione del nodo decisionale in due componenti separati. Questa decisione è stata presa in stretta aderenza al principio fondamentale che sottolinea la necessità di assicurare che le classi all'interno di un componente debbano essere suscettibili di modifiche per le stesse ragioni e non in risposta a motivazioni differenti o a tempi diversi.

Dopo aver eseguito un'approfondita analisi, è possibile identificare quattro elementi distinti che risultano fondamentali nella genesi degli alberi decisionali: il nodo radice, il nodo di azione, il nodo di decisione e il nodo di condizione. Tra questi, è il nodo di condizione a ricoprire un ruolo cruciale poiché si assume la responsabilità unica di valutare una determinata condizione.

Attraverso questa nuova struttura, si apre la possibilità di una ridefinizione delle competenze assegnate al nodo decisionale; in questo scenario, il nodo stesso si focalizza esclusivamente sull'analisi delle diverse condizioni che ospita, delegando ad esse la propria responsabilità principale.

Da ciò scaturisce il riconoscimento di un'altra responsabilità che richiede una definizione precisa: quella della chiave di lettura necessaria per valutare le diverse condizioni in questione.

Al fine di acquisire una prospettiva chiara sulla metodologia di sviluppo delle chiavi di lettura, l'algebra booleana emerge come un prezioso punto di riferimento. Secondo questa teoria, è possibile generare un numero di chiavi di lettura pari al conteggio delle porte logiche esistenti. Attraverso questo approccio, ciascun nodo di condizione assume il compito essenziale di produrre un risultato booleano valido. Questa sinergia contribuisce in maniera significativa alla robustezza e all'affidabilità delle chiavi di lettura che vengono formulate.

Mediante questa dettagliata analisi della decomposizione delle responsabilità, emergono cinque componenti fondamentali per la formulazione dell'albero decisionale:

- Nodo Radice: questo particolare nodo dà il via al procedimento dell'albero decisionale.
- Nodo Azione: nel corso del processo decisionale, porta a compimento un'azione, attivata dalla valutazione del nodo di verifica.
- Nodo Condizione: effettua la verifica di una condizione che produce un valore booleano all'interno del contesto decisionale.
- Porta Logica: esamina l'insieme di tutti i nodi condizione, valutando i rispettivi valori al fine di restituire un unico valore booleano.
- Nodo di Verifica: questo nuovo nodo di decisione si presenta come un'entità costituita da una chiave di lettura e una quantità variabile di nodi di condizione.

Il risultante sistema conduce a un livello di flessibilità strutturale estremamente evoluto per la generazione di alberi decisionali. Le varie componenti, ognuna investita di una singola responsabilità, si interconnettono per agevolare, adattamenti e estensioni in un'ottica di scalabilità e espandibilità.

Affinché si possa conseguire incremento di flessibilità tra le varie componenti, diventa imperativo consentire ai nodi di verifica, non solo di includere nodi di condizione, bensì anche altri nodi di verifica. Attraverso questa strategia, si realizza un livello di flessibilità nella creazione che raggiunge uno stadio altamente evoluto.

Sistemi per la verifica dei Nodi e la distribuzione degli alberi

Un aspetto di cruciale importanza per qualsiasi editor risiede nell'assicurare l'integrità complessiva del sistema che viene generato. In tale ottica, si

sottolinea l'indispensabile necessità di garantire non solo l'integrità, ma anche la solidità intrinseca dell'albero decisionale che prende forma attraverso l'operatività dell'editor.

In questa circostanza, diviene essenziale implementare sistemi che siano in grado di assicurare tali requisiti. Tali sistemi possono essere realizzati mediante l'esecuzione di test mirati, facendo ricorso anche a metodologie dei test basati sullo sviluppo (TDD).
In questo modo si garantisce il corretto funzionamento delle diverse componenti costituenti l'albero decisionale.

In particolare, è possibile condurre test mirati sui nodi di verifica e le porte logiche per la lettura delle condizioni. Questa strategia consente di esaminare in dettaglio il nucleo essenziale del processo decisionale. Ciò consente di convalidare qualsiasi struttura ad albero decisionale.

Inoltre, è di fondamentale importanza non solo instaurare sistemi che salvaguardino l'integrità dell'editor, bensì anche istituire complessi dispositivi di controllo operanti durante l'esecuzione, allo scopo di monitorare attentamente il procedere del processo decisionale in tempo reale. Questi sofisticati strumenti non solo semplificano il compito degli ideatori, ma permettono anche di esaminare e certificare in maniera accurata il percorso decisionale intrapreso durante l'effettiva fruizione del sistema.

In aggiunta, una volta che l'albero decisionale è stato concepito, assume rilevanza cruciale l'istituzione di un sistema di distribuzione specifico, il quale abiliti l'utilizzo dello stesso albero in una pluralità di contesti che ne richiedano l'applicazione, mantenendo costante e inalterato l'albero originale.

3.1.3 Custom Node

Il sistema di combattimento a turni è caratterizzato da un intricato insieme di elementi che guidano e regolano l'andamento degli scontri. Affinché sia possibile una sinergica e personalizzabile incorporazione di ogni aspetto dell'esperienza del gioco all'interno dell'albero decisionale, è necessario innanzi tutto individuare in modo dettagliato quali specifiche componenti dell'albero sono coinvolte in tale intento.

Nodo di condizione

Per avviare questo processo, si pone in primo piano l'indispensabile esigenza di acquisire la competenza necessaria per interpretare con precisione gli eventi che si scatenano durante il susseguirsi degli scontri. Questa abilità costituisce un elemento cruciale in quanto rappresenta un punto nodale per l'individuazione dei momenti specifici nel corso del combattimento. Di conseguenza, si mette

a disposizione una chiara e dettagliata prospettiva sulla dinamica sottostante al sistema di combattimento stesso. Per incorporare tale comportamento all'interno dell'albero decisionale, considerando la sua struttura, il componente che si dimostra più idoneo a tale scopo è il nodo di condizione.

Il nodo di condizione, come precedentemente illustrato, restituisce un valore booleano in base alla valutazione della condizione in questione. Questo aspetto ci consente di plasmare una vasta gamma di condizioni che interpretano gli eventi che si susseguono nel corso del combattimento, valutandone l'accuratezza. In breve, qualsiasi evento relativo allo svolgimento del combattimento potrà essere interpretato e tradotto in forma di condizione.

Come già esposto in precedenza, il nodo di condizione giunge a una valutazione espressa sotto forma di valore booleano in base all'analisi effettuata sulla condizione stessa. Questa particolarità conferisce la capacità di definire una varietà estesa di condizioni, ognuna delle quali è in grado di interpretare gli eventi che si verificano durante il corso del combattimento, e di giudicarne la pertinenza. In altre parole, qualsiasi avvenimento coinvolgente l'ambito del combattimento potrà essere tradotto e compreso attraverso l'implementazione di una condizione specifica.

Nodo di Azione

Oltre alla rilevanza di interpretare gli eventi che si manifestano nel contesto del combattimento, assume pari rilevanza la capacità di rispondere in modo attivo a tali eventi. Questa dinamica mira a generare comportamenti reattivi e adeguati in risposta agli sviluppi che emergono durante il progredire del combattimento stesso.

Uno dei pilastri fondamentali dell'albero decisionale risiede nei nodi d'azione, che svolgono un ruolo cruciale nell'emanare azioni. Questi nodi costituiscono il cuore pulsante del sistema, poiché sono incaricati di generare azioni in risposta alle condizioni rilevate. La loro struttura e funzione sono intrinsecamente predisposte per adattarsi e rispecchiare il principio di reattività, permettendo così una risposta mirata e appropriata agli eventi che si verificano nel corso del combattimento.

Pertanto, si procederà alla creazione di un insieme variegato di nodi d'azione, destinati a rispondere in modo mirato a condizioni specifiche all'interno del contesto di gioco. La strategia di adottare i nodi di condizione come meccanismo di interpretazione degli eventi che si manifestano durante lo svolgimento del combattimento, e i nodi d'azione come modalità di risposta a tali eventi, si configura come l'elemento cardine che sottende la costruzione stessa dell'intelligenza artificiale dei nemici. In questo quadro, i nodi di condizione acquisisco-

no la responsabilità di definire le circostanze necessarie affinché i nodi d'azione possano prendere la scelta di una determinata mossa da eseguire.

3.1.4 Editor lato grafico

Oltre all'articolata fase di modellazione degli aspetti logici nell'ambito dell'editor, è di pari rilevanza affrontare con meticolosa attenzione l'aspetto grafico ad esso correlato. Si intende che ogni singola componente integrante l'albero dovrà essere dotata di un'identità visuale ben definita, progettata con l'intento di illustrarne chiaramente il suo ruolo specifico e, in tal modo, eliminare ogni eventuale ambiguità intrinseca nell'intero albero decisionale.

Dalla considerazione della natura intrinseca dei nodi di verifica, i quali producono risposte booleane rappresentate da valori veri o falsi, emerge la necessità di concepire e realizzare un sistema grafico che agevoli la percezione visuale delle componenti connesse tra loro. Questo sistema dovrebbe essere progettato in modo tale da mettere in evidenza le relazioni intrinseche tra le diverse parti dell'albero e i rami che si collegano ai nodi di verifica. L'obiettivo è quello di offrire una rappresentazione visiva chiara e immediata che faciliti notevolmente la comprensione dell'intera struttura dell'albero decisionale e che permetta una veloce e precisa analisi dei percorsi e delle opzioni possibili. Questo sistema, in un'ottica statica, consente di visualizzare l'insieme completo dei possibili percorsi tracciabili nell'albero decisionale, senza però fornire una chiara rappresentazione del percorso effettivamente percorso durante l'attuazione del processo decisionale.

Per colmare questa lacuna, si rende necessario ampliare il sistema di controllo in tempo reale sull'albero decisionale. Tale sistema non solo garantisce un controllo a livello del modello, ma estende la sua efficacia anche al lato grafico, agevolando così sia il processo di controllo che di creazione dell'albero decisionale. In questo modo, è possibile accelerare il processo di verifica e di analisi, offrendo una panoramica completa sia dell'aspetto strutturale che dell'esecuzione del sistema.

3.2 Fighter

Nel contesto di questa sezione, si procederà a esaminare dettagliatamente l'approccio all'utilizzo degli alberi decisionali, i quali sono stati generati dall'editor di gioco, al fine di modellare il comportamento dei nemici noti come "Fighter". Saranno meticolosamente esplorate diverse strategie al fine di definire chiaramente sia la finalità intrinseca degli alberi decisionali sia l'obiettivo che sottende a essi. In altre parole, attraverso l'impiego dell'albero decisio-

nale, si cercherà di determinare in maniera precisa la mossa ottimale che il personaggio "Fighter" dovrà eseguire in ogni situazione di gioco.

Successivamente, si procederà con un'approfondita analisi della struttura adottata per la creazione di un modello di personalità, il quale avrà il compito di conferire tratti distintivi e peculiari al singolo personaggio. Subito dopo, verrà condotto uno studio approfondito sull'integrazione di questo modello all'interno dell'editor degli alberi decisionali. L'obiettivo di tale integrazione sarà quello di plasmare e influenzare il processo decisionale, contribuendo in modo significativo alla definizione delle scelte di azione ottimali da intraprendere.

Alla fine di questa sezione di ricerca, sarà sviluppata un'apposita metodologia che, in sinergia con l'algoritmo esaminato nel capitolo precedente, ossia la state machine, permetterà di formulare una strategia volta a modificare i tratti caratteriali della personalità del personaggio, in base a uno stato di vita specifico.

L'effetto concreto di tale metodologia sarà un comportamento non statico, ma in continua evoluzione, da parte del personaggio stesso. Questo, a sua volta, avrà un impatto profondo e tangibile sul processo decisionale degli alberi durante gli scontri in-game. In sostanza, si creerà un sistema dinamico in cui le scelte intraprese dal personaggio saranno sensibilmente influenzate dalle variazioni dei suoi tratti di personalità, arricchendo così l'esperienza di gioco complessiva.

3.2.1 Strategie per l'uso dell'AI: Decision Tree

Gli alberi decisionali si pongono come strumento precipuo per l'instaurazione di un processo decisionale all'interno del contesto ludico, volto a condurre alla selezione ottimale di azioni e mosse da intraprendere. Tuttavia, il loro contributo non si esaurisce in questa prerogativa. È di altrettanta rilevanza cruciale porre sotto lente di ingrandimento un processo che riveste notevole importanza, ossia la determinazione del bersaglio da colpire o influenzare tramite l'azione scelta.

In questo scenario, emerge un approccio bifocale all'interno della gestione del personaggio durante gli scontri: da una parte, la creazione di un albero decisionale per le mosse da adottare, dall'altra, un albero accurato e ponderato per la scelta del soggetto o obiettivo su cui concentrare l'azione intrapresa. Questa duplice prospettiva di analisi e controllo si coniuga per conferire un livello di maestria e padronanza nell'esperienza di combattimento, assicurando un'interazione profonda e completa con le variabili in gioco.

Scelta della Mossa

Il percorso iniziale nell'analisi dei vari alberi decisionali trova il suo primo punto focale nell'esame dell'albero che governa le diverse mosse che il personaggio può intraprendere. All'interno di questo albero specifico, attraverso l'utilizzo dell'editor messo a disposizione, verranno plasmate e delineate tutte le diramazioni, le sottolineature e le dinamiche complesse che concorrono a determinare il processo di selezione della mossa che il personaggio sarà chiamato a eseguire. Questo processo implica la creazione di una struttura decisionale articolata e interconnessa, il cui scopo è regolare con precisione il comportamento del personaggio in relazione alle diverse situazioni che possono emergere all'interno del contesto di gioco.

In questo processo decisionale, non soltanto si terranno in considerazione le molteplici situazioni che si presentano durante gli scontri, ma si darà altrettanto spazio alla creazione di scelte predefinite che si adattano a condizioni predeterminate. Questo duplice approccio avrà l'effetto di conferire una forma ben definita ai comportamenti del personaggio. Da un lato, si terrà conto della fluidità delle situazioni mutevoli che caratterizzano gli scontri, dall'altro si darà spazio a comportamenti costanti e preordinati, creando così un equilibrio tra adattabilità e struttura all'interno del sistema decisionale.

L'implementazione di questa particolare metodologia apre le porte alla creazione di design articolati per le diverse mosse disponibili. Questi design non solo influenzano direttamente il processo decisionale legato alla scelta delle azioni da intraprendere, ma offrono anche un'opportunità per modellare il comportamento del personaggio in modo specifico. A titolo di esempio, si potrebbe ideare un design di mossa che richiede al personaggio di eseguirla due volte in successione, un'approccio che si dimostra necessario affinché l'azione abbia l'effetto desiderato. Questo livello di dettaglio nell'articolazione delle mosse contribuisce a fornire una maggiore varietà e complessità al sistema decisionale, arricchendo ulteriormente l'esperienza di gioco e le dinamiche di combattimento.

Scelta del Target

Una volta che è stata compiuta la determinante decisione riguardante l'azione da intraprendere, si apre la strada alla fase successiva: la creazione di un albero decisionale specifico destinato a guidare la scelta del bersaglio. In questa fase ulteriore, emerge la necessità di elaborare un processo decisionale dettagliato, il quale si prefigge l'obiettivo di definire con meticolosità quale individuo debba essere preso in considerazione al fine di eseguire con precisione l'azione precedentemente selezionata. In tal modo, si completa un ciclo

di decisioni interconnesse che contribuiscono in maniera cruciale a definire le azioni e le direzioni intraprese dal personaggio nell'ambito del combattimento.

L'essenza fondamentale della logica che governa la selezione dei bersagli trova la sua incanalatura all'interno di una componente altamente specifica all'interno di ogni singola mossa. Tuttavia, è importante sottolineare che questa logica rivela la sua presenza in modo esplicito solamente in situazioni in cui le scelte di bersaglio sono predefinite e non soggette a variazioni.

È in quest'ottica che l'albero decisionale entra in azione in un ruolo di sostegno cruciale, agendo come strumento di modellazione e di delineamento del processo di selezione del bersaglio. Questo ruolo acquisisce una rilevanza accentuata nei momenti in cui la logica relativa al bersaglio per una specifica mossa non mostra una logica predefinita per la scelta del target. In tali circostanze, l'albero decisionale offre una struttura articolata per affrontare e gestire la complessità delle scelte di bersaglio in evoluzione, garantendo una gestione coerente e adattabile delle situazioni che emergono durante il combattimento.

Componente per gestire Gli alberi decisionali

Nel quadro dell'analisi dei due alberi decisionali destinati a controllare il comportamento del personaggio durante gli scontri, emerge la necessità di individuare e definire un nuovo elemento chiave. Questo componente aggiuntivo assume un ruolo centrale: è chiamato a svolgere il compito fondamentale di inizializzare gli alberi decisionali specifici del personaggio. Questo processo di inizializzazione si dimostra essenziale per garantire che gli alberi decisionali siano pronti all'uso e pienamente operativi quando il personaggio è coinvolto nel combattimento.

In pratica, questo nuovo elemento introduce un livello di organizzazione e coordinamento nell'ambito delle dinamiche delle decisioni, consentendo la distribuzione più agevole ed efficiente degli alberi decisionali del personaggio.

3.2.2 Personalità

Dopo aver impiegato con successo gli alberi decisionali, si perviene a un livello di controllo estremamente completo sulle azioni del personaggio nel corso dei combattimenti. Tuttavia, emerge un aspetto di fondamentale importanza che riguarda l'approfondimento della personalità intrinseca del medesimo personaggio. Di conseguenza, si pone l'imperativo di ideare e implementare una strategia accurata finalizzata alla creazione di un sofisticato sistema di personalità.

Questo sistema dovrà essere integrato in modo sinergico e organico all'interno della struttura preesistente dell'albero decisionale. L'obiettivo di tale in-

tegrazione consiste nel permettere alla personalità stessa di esercitare un ruolo attivo e influente nel corso del complesso processo decisionale che orienta le azioni e le scelte del personaggio in questione.

Tratti

Al fine di ottenere un efficace modellazione delle personalità dei personaggi, si rende indispensabile l'attività di individuazione e definizione dei tratti intrinseci che le compongono. Attraverso questa fase di analisi e categorizzazione, è possibile creare un insieme di dati nominale che agisce quale rappresentazione identificativa di tali tratti, contribuendo così alla caratterizzazione profonda e articolata delle personalità stesse.

Di conseguenza, la personalità di ciascun personaggio risulterà costituita da una molteplicità di tratti distinti, ognuno dei quali opererà come un fattore chiave. Questo scenario suscita l'esigenza di determinare la strategia ottimale per attribuire a ciascun tratto un valore corrispondente, al fine di stabilire una relazione di associazione chiave-valore, ovvero tra il tratto individuale e il suo valore intrinseco.

Logica del valore del tratto

Nell'ottica di collegare un valore specifico a ciascun tratto, emerge la necessità di discernere quale tipologia di valore sia più appropriata da adottare. In questa circostanza, si propende per l'impiego di un valore numerico, poiché questo facilita la comprensione e agevola la valutazione dettagliata del tratto stesso. Tale scelta consente, inoltre, il confronto diretto con altri valori numerici, amplificando le opportunità di interpretazione. In definitiva, l'utilizzo di valori numerici amplifica il panorama di letture possibili, favorendo un approccio più ampio e articolato all'analisi dei tratti della personalità.

Nell'ambito della ricerca del valore associato a ciascun tratto, è possibile esplorare svariate modalità e tattiche volte a ottenere tale valore. Nel contesto specifico, al fine di sviluppare un sistema di valutazione che risulti congruente con l'obiettivo, si rivela indispensabile introdurre un modello noto come "Strategy". [1] L'implementazione di questo pattern offre l'opportunità di istituire un sistema strutturato, capace di configurare diverse metodologie atte a determinare il valore intrinseco del tratto. In tal modo, si apre la strada a una molteplicità di approcci volti a conseguire una valutazione accurata e approfondita, coniugando varietà e precisione all'interno del processo di assegnazione dei valori ai tratti.

Integrazione con l'albero decisionale

Con l'obiettivo di assicurare un'integrazione ottimale del sistema chiave-valore legato alla personalità, è essenziale procedere con un'esaustiva delucidazione del suo funzionamento ottimale all'interno dei complessi alberi decisionali che governano il personaggio in questione. Alla luce del modello di personalità delineato, risulta evidente che le scelte intraprese dal suddetto individuo vengono intrinsecamente plasmate e indirizzate dalla configurazione unica della propria personalità.

Uno degli elementi costituenti l'architettura dell'albero decisionale, che dà origine al comportamento precedentemente descritto, s'identifica nel nodo di condizione. Tale nodo, oggetto di valutazione da parte del nodo di verifica, si configura come il motore scatenante dell'azione appropriata, in accordo con le circostanze del contesto. In questa configurazione, si può osservare che il valore insito nel tratto di personalità rappresenta un nodo di tipo condizione. Al fine di poter eseguire una valutazione precisa del valore associato a tale tratto all'interno del nodo di condizione, risulta importante capire come valutate il valore stesso, consentendo cosi una corretta operatività del nodo di condizione nell'ambito dell'albero decisionale.

In relazione a questo argomento, nel punto di valutazione del nodo di condizione è necessario un valore booleano che possa essere vero o falso. Pertanto, per comprenderne il significato del valore dei tratti della personalità, diventa essenziale considerare la probabilità associata a tale valore. La strategia ottimale per effettuare questo calcolo consiste nell'adozione di un intervallo di valori compreso tra 0 e 100, allo scopo di ottimizzare il processo di calcolo.

Pertanto, il valore associato al tratto delle personalità acquisisce una scala compresa tra 0 e 100. Questa disposizione consente al nodo di condizione di effettuare una valutazione probabilistica su tale valore, ritornando un valore vero o falso. Quest'ultimo valore costituisce un elemento indispensabile affinché il nodo possa essere adeguatamente considerato all'interno della struttura dell'albero decisionale.

3.2.3 Stato di salute

Dopo aver individuato il sistema inerente alle personalità, va sottolineato che questo sistema si caratterizza come statico, mantenendo la sua immutabilità nel corso delle fasi di combattimento. Tuttavia, questa caratteristica ha suscitato la necessità di introdurre un elemento nuovo, capace di conferire una dimensione dinamica ai valori delle personalità, modificandoli in tempo reale durante lo svolgimento del combattimento.

Al fine di affrontare questa esigenza, è stato elaborato un nuovo componente, il cui sviluppo si fonda sull'applicazione dell'algoritmo di una state machine. In questo contesto, l'intento è quello di creare una state machine che sia in grado di reagire in relazione al livello di salute del personaggio. Ciò consente di apportare mutamenti mirati ai valori dei tratti della personalità, adattandoli al mutare delle circostanze all'interno del combattimento.

L'implementazione di questa state machine implica una ristrutturazione dei valori dei tratti di personalità, basata su una logica specifica. Di conseguenza, emerge una dinamica in cui i valori delle personalità non restano statici, ma subiscono alterazioni coerenti e pertinenti. Questo processo, a sua volta, incide attivamente sul flusso decisionale del personaggio, influenzandone le scelte e le azioni lungo l'intera durata del combattimento.

Verifica dello stato di salute

All'interno di ciascuna personalità, si collocherà una state machine specifica, progettata per ospitare una sequenza articolata di stati. Questi stati, però, si attiveranno unicamente in relazione alla percentuale di vita del personaggio. Per ogni stato concepito, saranno generati due parametri distinti, i quali conterranno i valori percentuali di riferimento. Questi valori delineeranno con precisione l'intervallo all'interno del quale il livello di salute del personaggio dovrà collocarsi affinché lo stato corrispondente sia innescato.

L'insorgenza di tale stato comporterà l'introduzione di un nuovo strato, un livello aggiuntivo di complessità, che si frapporrà in modo distintivo tra il nodo condizionale e il valore intrinseco del tratto di personalità. Di conseguenza, lo stato agirà come una variabile di mediazione, inserendosi in modo significativo tra questi due elementi cardine del sistema. La sua funzione primaria sarà quella di apportare modifiche mirate e direzionali ai valori dei tratti di personalità. Queste modifiche saranno governate da una logica predefinita, la quale garantirà un intervento tempestivo, precedente all'elaborazione del nodo di condizione e alla conseguente valutazione della probabilità associata al valore specifico del tratto modificato. In tal modo, si instaura un meccanismo sofisticato e stratificato che consente un'attiva influenza sulla dinamica decisionale del personaggio lungo l'intero corso del combattimento.

Logica per cambiare i valori dei tratti della personalità

Una volta che uno stato è stato attivato all'interno della state machine, emerge la necessità di delineare un componente specifico che si prenderà direttamente carico di apportare alterazioni ai valori dei tratti della personalità. Questo processo, che si dispiega in seguito all'attivazione dello stato, richiede

un approccio paragonabile a quello impiegato inizialmente per la generazione dei valori stessi dei tratti. In questo contesto, è di primaria importanza adottare il noto pattern denominato "Strategy". [1]

La scelta di implementare il pattern "Strategy" [1] è guidata dalla volontà di istituire un sistema che si caratterizzi per la sua flessibilità espansiva e la sua capacità di adattarsi in modo scalabile alle mutevoli esigenze. In particolare, tale approccio consentirà di realizzare nuove logiche di modifica dei valori dei tratti di personalità con relativa agevolezza. Questo meccanismo rende possibile l'aggiunta e l'integrazione di nuove strategie, ciascuna con il proprio insieme di regole e operazioni, che agiranno sulla trasformazione dei valori dei tratti. In definitiva, il pattern "Strategy" [1] costituirà un tassello cruciale nell'architettura complessiva, permettendo di manipolare e influenzare in modo dinamico e strutturato i tratti della personalità in risposta alle variazioni dello stato di salute del personaggio.

3.3 Influenza degli elementi base del combattimento

In questa sezione, si procederà a un'analisi approfondita delle modalità con cui i sistemi di combattimento possono esercitare un'influenza determinante sulle dinamiche degli alberi decisionali. Questo studio permetterà di comprendere come sia possibile personalizzare e integrare con flessibilità diverse logiche di gioco all'interno del complesso processo decisionale, mettendo a disposizione gli strumenti necessari all'editor e offrendo così ampie opportunità per adattare e ottimizzare le strategie di gioco in modo aderente alle esigenze specifiche.

3.3.1 Mosse

Nel contesto del sistema di gioco, è cruciale riconoscere che le mosse relative ai personaggi rappresentano un pilastro centrale e imprescindibile. Esse costituiscono l'essenza stessa del design delle azioni che i personaggi devono compiere durante i combattimenti, svolgendo un ruolo di primaria rilevanza nell'esperienza di gioco complessiva. Pertanto, assume un'importanza cruciale la capacità delle logiche interne delle mosse di esercitare un'influenza attiva e ponderata sui processi decisionali dei personaggi, poiché questo consente di plasmare con precisione e dettaglio il design delle azioni, adattandole in modo specifico a comportamenti desiderati in linea con gli obiettivi di gioco predefiniti.

Per supportare appieno il complesso design delineato in precedenza, è possibile sfruttare appieno le ampie capacità offerte dall'editor degli alberi deci-

sionali. Questo sofisticato strumento mette a disposizione degli sviluppatori la possibilità di creare nodi di condizione e azione completamente personalizzati, consentendo così di plasmare tali nodi con una precisione meticolosa. Questa personalizzazione mirata mira a influenzare in modo profondo e accurato il processo decisionale all'interno del contesto di gioco, utilizzando le logiche intrinseche alle singole mosse come leva per modellare l'esperienza di gioco. In tal modo, si raggiunge un livello di integrazione sinergica delle azioni all'interno del processo decisionale, il che si traduce in un'esperienza di gioco estremamente adattabile, coerente e in linea con il design e gli obiettivi prefissati.

3.3.2 Target

Nel contesto dei combattimenti, è di rilevante importanza riconoscere il ruolo attivo e cruciale svolto dai personaggi all'interno del complesso processo decisionale. Ciò comporta la necessità di un'approfondita elaborazione dei dati forniti da questi stessi personaggi all'interno dell'articolato albero decisionale.

Questa analisi comprende vari aspetti, tra cui il monitoraggio del loro stato di salute rappresentato dal livello di vita, l'esplorazione dei tratti distintivi della loro personalità, e la valutazione delle opzioni disponibili per le mosse da essi effettuate, tra molte altre variabili pertinenti. In sintesi, è fondamentale considerare e integrare accuratamente tutte le informazioni messe a disposizione dai personaggi, poiché queste informazioni costituiscono un elemento chiave per una presa di decisione consapevole e ponderata durante il combattimento.

In accordo con il design previamente delineato per le mosse, è essenziale applicare la medesima metodologia anche al design delle logiche derivate dai personaggi. Ciò implica l'utilizzo coerente di nodi di decisione e azione personalizzati al fine di garantire l'integrazione fluida e sinergica di qualsiasi logica derivante dai comportamenti dei personaggi durante le fasi di combattimento. Tale approccio consente di assicurare una connessione profonda tra le caratteristiche intrinseche dei personaggi e le scelte compiute all'interno del gioco, contribuendo a creare un'esperienza di combattimento ricca e altamente personalizzata, in cui le dinamiche interne dei personaggi interagiscono in modo dinamico con il sistema decisionale complessivo.

3.3.3 Componenti: Stati, Dot e Hot

Nell'ambito del gioco, è cruciale riconoscere il ruolo determinante svolto dai componenti chiave come gli stati, i dot e gli hot, i quali influenzano in modo sostanziale le dinamiche di gioco. Pertanto, risulta imprescindibile integrare tali elementi nel processo decisionale dei personaggi durante le fasi di

combattimento. Questa integrazione è essenziale al fine di catturare qualsiasi logica intrinseca a tali componenti, consentendo, ad esempio, di determinare se un personaggio è affetto da uno stato o un dot specifico. Le logiche derivate da questi componenti rivestono un'importanza fondamentale, in quanto sono in grado di influire attivamente sul processo decisionale dei personaggi, modellando le loro scelte e azioni in maniera coerente con la situazione di gioco e gli effetti derivanti da tali elementi.

Grazie all'utilizzo sapiente dei nodi di condizione e azione personalizzati all'interno dell'editor degli alberi decisionali, si apre la porta alla creazione di una varietà considerevole di nodi specifici in grado di incapsulare in maniera completa e dettagliata le intricate logiche associate agli stati, dot e hot presenti nel contesto di gioco. Questa flessibilità di progettazione consente di modellare con precisione e adattare in modo specifico il comportamento dei personaggi rispetto a una vasta gamma di circostanze, creando così un'esperienza di gioco ricca e sfaccettata, in cui la comprensione e l'applicazione delle logiche relative agli stati, dot e hot sono fondamentali per il successo e la strategia nel contesto del combattimento.

3.4 Integrazione dei sistemi: Editing

In questo capitolo, si è proceduto all'analisi dettagliata dei complessi sistemi che sottendono al combattimento. Tale analisi ha coinvolto l'esame approfondito delle logiche che regolano ciascuna delle componenti coinvolte in questo intricato ecosistema (stati, dot e hot), il sistema di personalità dei personaggi e le logiche sottostanti alle diverse mosse.

Questo approfondimento ha fornito una solida base di conoscenza per lo sviluppo e la messa a punto degli strumenti messi a disposizione dall'editor degli alberi decisionali. Grazie a questi strumenti avanzati, l'editor offre la capacità di esercitare un controllo completo e totale sull'intero ecosistema del combattimento. Questa padronanza delle dinamiche interne ci permette di plasmare l'esperienza di gioco in modo mirato, adattando il sistema decisionale e le azioni dei personaggi per raggiungere gli obiettivi prefissati con precisione e coerenza, garantendo così una partita ricca di sfumature e altamente personalizzata.

Capitolo 4

Design e Implementazione dell'Editor e dei Tool a Supporto

All'interno di questo capitolo, sarà esposto in dettaglio l'iter evolutivo della progettazione, nonché dei vari algoritmi adoperati per lo sviluppo dell'editor, atto alla creazione dell'intelligenza artificiale dei nemici del gioco. Ovvero, si tratta dell'ambiente di sviluppo progettato per offrire al game designer la capacità di generare intelligenza artificiale con strategie personalizzate, destinate ad arricchire il comportamento dei nemici all'interno del contesto di gioco.

Il progetto fa affidamento su uno dei motori di sviluppo più diffusi nell'ambito dei videogiochi, ovvero Unity. La realizzazione avviene attraverso l'utilizzo del linguaggio di programmazione C# e sfrutta la vasta gamma di strumenti messi a disposizione dalla ambiente di sviluppo Unity. Inoltre questa trattazione sarà supportata da frammenti di codice e da diagrammi UML, con l'obiettivo di offrire una rappresentazione ottimale del design sviluppato. Il percorso espositivo avrà inizio dall'approfondimento dello sviluppo dell'albero decisionale, congiuntamente al suo editor, e procederà successivamente all'analisi dettagliata dell'implementazione della metodologia scelta per conseguire l'obiettivo prestabilito.

Per realizzare ciascun elemento relativo all'ambiente Unity, sono stati consultati documenti[10] e forum[9] al fine di sviluppare tutti i componenti essenziali richiesti per la creazione dell'editor.

4.1 Decision Tree Tool

Iniziando dall'albero decisionale, verrà esposto l'intero percorso evolutivo del principale algoritmo adoperato per lo sviluppo dell'ambiente atto alla creazione dell'intelligenza artificiale dei nemici, affiancato da uno strumento di editing.

Questa fase di editing si configura come un'opportunità strategica volta all'ottimizzazione e al progresso congiunto della creazione e del perfezionamento dell'intelligenza artificiale. Tuttavia, il suo valore si estende ulteriormente, offrendo un duplice beneficio che si riflette anche nella fruizione dell'esperienza da parte del game-designer durante l'utilizzo di tale strumento.

4.1.1 Assets

Nel contesto del progetto e nell'ambiente di sviluppo di Unity, il termine "Asset" è impiegato per riferirsi al dato persistente.

L'inizio del percorso di sviluppo coinvolge la creazione dei nodi dell'albero, ciascuno dei quali assume un compito specifico. Al fine di assicurare una scalabilità efficiente dell'albero, sono stati individuati e definiti quattro nodi fondamentali:

- Nodo Radice: costituisce il nodo principale da cui prende avvio l'intero albero;
- Nodo di Condizione: la sua funzione consiste nel restituire un valore booleano, destinato all'elaborazione da parte del nodo di verifica;
- Nodo di Verifica: è incaricato di esaminare l'insieme dei nodi figli di condizione, sfruttando una specifica porta logica per la valutazione;
- Nodo d'Azione: includerà un'azione ben definita, la quale sarà innescata tramite i nodi di verifica;

Per la creazione e l'evoluzione di tali nodi, è essenziale disporre di dati persistenti in grado di conservare l'intera struttura degli alberi. All'interno dell'ecosistema di Unity, sono disponibili vari elementi per mantenere dati persistenti, come ad esempio scene, prefab e ScriptableObject. È interessante notare che persino Unity fa uso degli ScriptableObject per la creazione dei propri editor interni, poiché questi si rivelano estremamente flessibili nella creazione di editor di sviluppo personalizzati.

Ciò conduce all'elaborazione individuale di un oggetto scriptableObject dedicato all'albero decisionale nel suo insieme, oltre a generare oggetti scriptableObject distinti per ciascuna categoria di nodo presente nell'albero stesso.

Albero

All'interno dell'albero, verranno mantenuti i riferimenti a tutti i nodi da esso contenuto, inclusa la sua radice fondamentale che orchestrerà l'intero flusso decisionale. Questo sistema di riferimenti interni consentirà la gestione agevole dei suoi nodi, oltre a permettere la clonazione dell'albero stesso, un meccanismo cruciale per la distribuzione della sua struttura. Con questa metodologia, si garantisce di evitare modifiche dirette ai dati persistenti dell'albero collegato al suo scriptableObject.

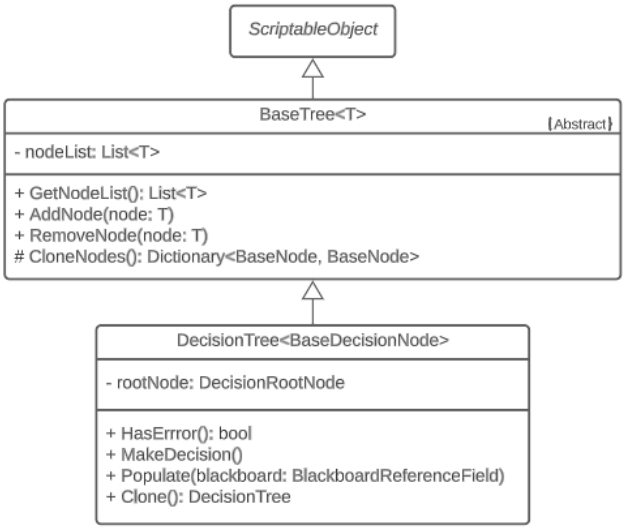

Figura 4.1: Schema UML dell'albero decisionale

Nodo Radice

Il nodo radice ha il compito di mantenere il riferimento al nodo iniziale di verifica, consentendo così l'inizio del processo decisionale complessivo. Naturalmente, nell'ambito dell'intero algoritmo, si è scelto di adottare un singolo nodo di verifica; tuttavia, è possibile estendere questa logica creando un nodo radice in grado di accogliere più nodi di verifica. In aggiunta, il nodo è investito del ruolo di effettuare la duplicazione sia di sé stesso che dei suoi collegamenti, rappresentati dai nodi interconnessi. Questa operazione è fondamentale per implementare il processo mediante il quale si consolida il meccanismo di distribuzione dell'Asset.

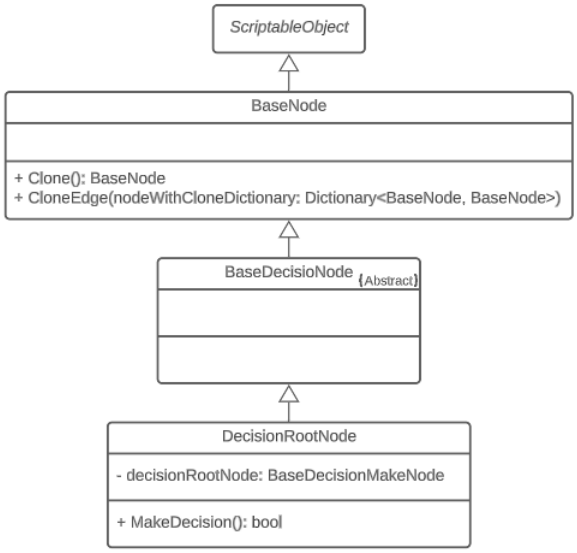

Figura 4.2: Schema UML del nodo radice

Nodo di Condizione

Incorpora la realizzazione di nodi in grado di restituire un valore booleano, che può essere vero o falso. Inoltre, è possibile ampliare la procedura di creazione per consentire varie forme di elaborazione dei dati, finalizzate all'ottenimento di un valore booleano. Inoltre, sono integrate le funzioni atte a supportare la duplicazione dell'oggetto.

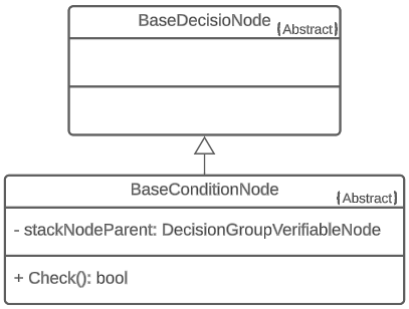

Figura 4.3: Schema UML del nodo di condizione

Nodo d'Azione

Abbraccia la creazione di nodi in grado di eseguire azioni senza produrre alcun valore di ritorno. Offre inoltre la flessibilità di espandere questa procedura per ottenere nodi di diversi tipi, ciascuno con la capacità di accettare

una varietà di campi per eseguire azioni specifiche, anch'esso con funzionalità di clonazione.

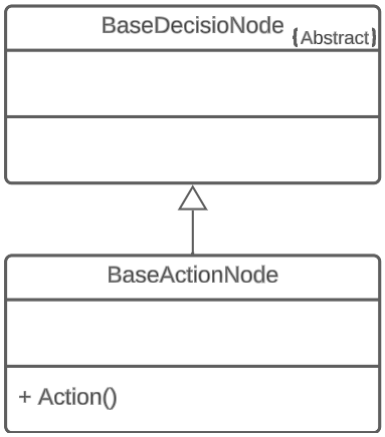

Figura 4.4: Schema UML del nodo di azione

Nodo di Verifica

Viene presentato il nodo centrale all'interno dell'albero decisionale, il quale assume il ruolo di conduttore del processo decisionale, e si compone di tre elementi essenziali.

- Contenitore di condizioni:
  Il nodo di verifica, simile al nodo di condizione, produce un valore booleano e, inoltre, effettua l'elaborazione di tutti i nodi figli, che possono includere sia nodi di verifica che nodi di condizione. Questo procedimento è finalizzato all'acquisizione del valore di ritorno desiderato. Pertanto, il contenitore è formato da ogni tipo di nodo che produce un valore booleano, che in questa situazione include sia i nodi di verifica che quelli di condizione.

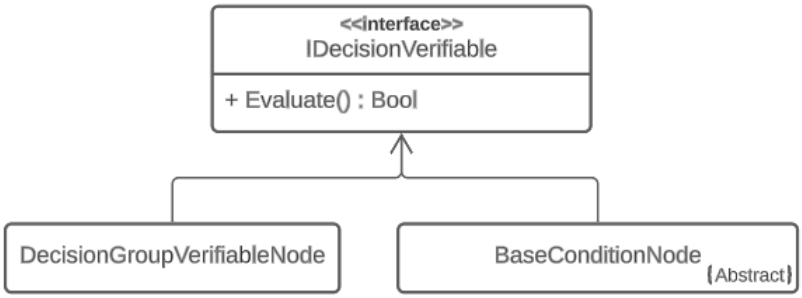

Figura 4.5: Schema UML dei nodi figli del nodo di verifica

- Porta Logica:
  Rappresenta l'entità incaricata di fornire la chiave di interpretazione per l'elaborazione di tutti i nodi contenuti nel contenitore delle condizioni. Include le varie porte logiche dell'algebra booleana, come ad esempio AND, OR, NAND e NOR.

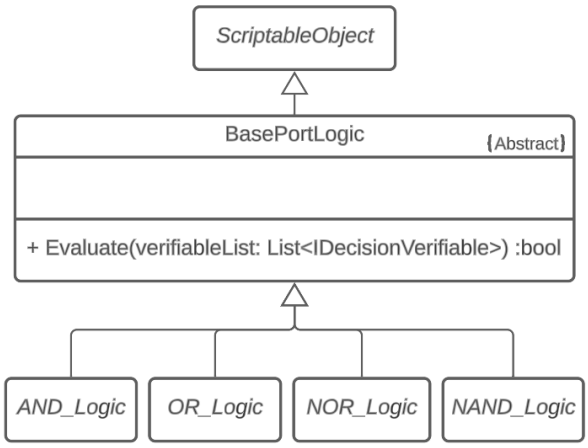

Figura 4.6: Schema UML delle porta logiche

- Porta di Connessione:
  Considerando ad esempio un nodo di verifica che ha la capacità di restituire solamente un valore, vero o falso, la porta di connessione viene associata a uno specifico risultato del valore di ritorno. In questo contesto, vengono istituite due porte: una per il valore vero e una per il valore falso. Queste porte consentono l'associazione con nodi di azione o nodi di verifica. Nel primo scenario, la porta è utilizzata per attivare il nodo che eseguirà un'azione, mentre nel secondo caso, essa permette ulteriormente di elaborare un insieme di condizioni, quindi di dare continuità al processo decisionale. Per estendere questo procedimento in maniera generale, viene associata una porta di connessione a ciascun possibile valore assunto dal risultato di ritorno.

4.1.2 Test degli Assets

All'interno di questo studio, si procederà con l'effettuazione di una serie di test mirati alla valutazione critica della fase di creazione e del comportamento dei componenti precedentemente sviluppati. L'intento principale di tali test è quello di conferire una solida e affidabile struttura dei diversi elementi che compongono l'editor, contribuendo così a consolidare la stabilità e l'integrità complessiva dell'editor stesso. Inoltre, verrà svolta un'analisi esaustiva del

comportamento globale dell'albero decisionale, con l'obiettivo di garantire la corretta creazione e il corretto funzionamento.

Al fine di sviluppare test affidabili per i componenti che compongono l'albero decisionale, l'ambiente Unity mette a disposizione un framework "NUnit" in C# per la creazione di test sia statici che dinamici che vengono eseguiti durante l'esecuzione del programma.

Nodo d'Azione

Per avviare questa fase di testing, s'incomincerà con la verifica del nodo azione, iniziando con l'analisi della sua fase iniziale di creazione e proseguendo con l'esame della logica relativa alla clonazione del componente e al suo funzionamento durante l'esecuzione dell'azione. Questa approfondita analisi ci consentirà di convalidare in modo completo e accurato il corretto funzionamento del nodo azione e la sua clonazione.

```
[Test]
public void ActionNodeTest()
{
    var action = ScriptableObject.CreateInstance<DebugActionNode>();
    var actionClone = Object.Instantiate(action);

    Assert.NotNull(action);
    Assert.NotNull(actionClone);

    action.SetMessage("Test Action");
    actionClone.SetMessage("Test Action 1");

    action.Action();
    actionClone.Action();
    Assert.AreNotEqual(action.message, actionClone.message);
}
```

Listato 4.1: Codice sorgente dei test del nodo di azione

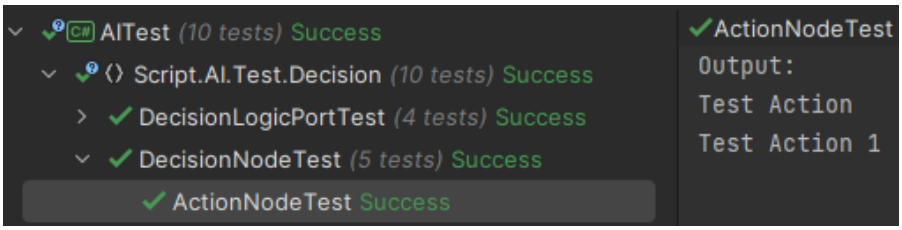

Figura 4.7: Controllo dei test del nodo di azione

Nodo di Condizione

Nella presente fase dell'indagine, si procederà con l'analisi approfondita del nodo di condizione. Saranno esaminati con attenzione i vari aspetti legati alla sua creazione, alle procedure di clonazione e, in particolare, al valore booleano generato in seguito. Per valutare la correttezza del comportamento di questo nodo, si effettueranno diverse modifiche nei parametri interni del nodo stesso, al fine di testare una gamma completa di scenari possibili. Attraverso questo processo esaustivo, si propone di convalidare in maniera approfondita il corretto funzionamento del nodo di condizione e la sua intrinseca affidabilità e stabilità.

```
[Test]
public void ConditionNodeTest()
{
    var conditionIntEquals =
        ScriptableObject.CreateInstance<IntEqualsConditionNode>();
    var conditionIntEqualsClone =
        Object.Instantiate(conditionIntEquals);
    Assert.NotNull(conditionIntEquals);
    Assert.NotNull(conditionIntEqualsClone);

    conditionIntEquals.SetValue(1, 1);
    conditionIntEqualsClone.SetValue(0, 1);
    Assert.AreEqual(true, conditionIntEquals.Evaluate());
    Assert.AreEqual(false, conditionIntEqualsClone.Evaluate());

    conditionIntEquals.SetValue(1, 0);
    conditionIntEqualsClone.SetValue(0, 0);
    Assert.AreEqual(false, conditionIntEquals.Evaluate());
    Assert.AreEqual(true, conditionIntEqualsClone.Evaluate());

    conditionIntEquals.SetValue(0, 1);
    conditionIntEqualsClone.SetValue(1, 1);
    Assert.AreEqual(false, conditionIntEquals.Evaluate());
    Assert.AreEqual(true, conditionIntEqualsClone.Evaluate());

    conditionIntEquals.SetValue(0, 0);
    conditionIntEqualsClone.SetValue(1, 0);
    Assert.AreEqual(true, conditionIntEquals.Evaluate());
    Assert.AreEqual(false, conditionIntEqualsClone.Evaluate());
}
```

Listato 4.2: Codice sorgente dei test del nodo di condizione

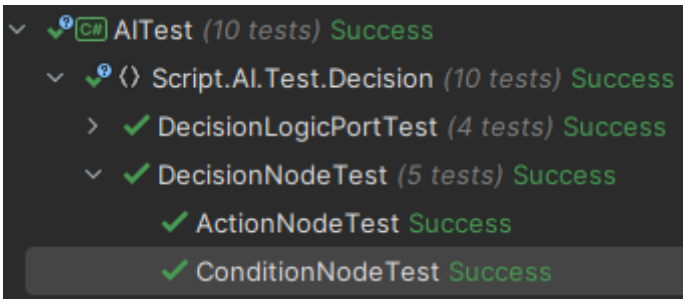

Figura 4.8: Controllo dei Test del nodo di condizione

Nodo di Verifica

Per condurre in modo accurato e completo un test sul nodo di verifica, è imperativo avviare il processo con un attento esame del suo componente interno, che in questo caso è rappresentato dalla porta logica. Questa porta logica svolge un ruolo cruciale nel contesto dei nodi figli, fornendo una chiave di lettura specifica. La sua funzione principale consiste nel ricevere parametri in ingresso e restituire un valore booleano in base a tali parametri. Attraverso un'analisi meticolosa e una serie di test eseguiti su questo componente, l'obiettivo fondamentale è confermare l'efficacia e la coerenza del meccanismo di controllo che è associato a ciascuna porta logica. La validazione di tale comportamento è essenziale per garantire il corretto funzionamento complessivo del sistema del nodo di verifica.

```
[Test]
public void Test_OR()
{
    var or = ScriptableObject.CreateInstance<OR_Logic>();
    Assert.AreEqual(false, or.Evaluate(false, false));
    Assert.AreEqual(true, or.Evaluate(true, false));
    Assert.AreEqual(true, or.Evaluate(false, true));
    Assert.AreEqual(true, or.Evaluate(true, true));
}
[Test]
public void Test_AND()
{
    var and = ScriptableObject.CreateInstance<AND_Logic>();
    Assert.AreEqual(false, and.Evaluate(false, false));
    Assert.AreEqual(false, and.Evaluate(true, false));
    Assert.AreEqual(false, and.Evaluate(false, true));
    Assert.AreEqual(true, and.Evaluate(true, true));
}
```

```
[Test]
public void Test_NAND()
{
    var nand = ScriptableObject.CreateInstance<NAND_Logic>();
    Assert.AreEqual(true, nand.Evaluate(false, false));
    Assert.AreEqual(true, nand.Evaluate(true, false));
    Assert.AreEqual(true, nand.Evaluate(false, true));
    Assert.AreEqual(false, nand.Evaluate(true, true));
}
[Test]
public void Test_NOR()
{
    var nor = ScriptableObject.CreateInstance<NOR_Logic>();
    Assert.AreEqual(true, nor.Evaluate(false, false));
    Assert.AreEqual(false, nor.Evaluate(true, false));
    Assert.AreEqual(false, nor.Evaluate(false, true));
    Assert.AreEqual(false, nor.Evaluate(true, true));
}
```

Listato 4.3: Codice sorgente dei test delle porte logiche

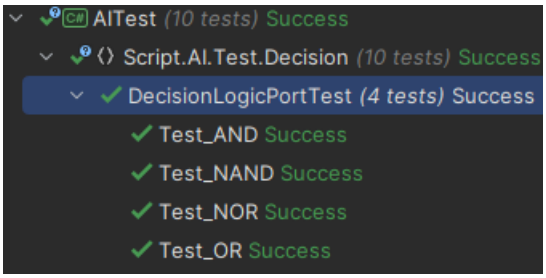

Figura 4.9: Controllo dei test delle porte logiche

Ulteriori test di importanza decisiva si concentrano sulla valutazione della creazione, della clonazione e, non meno significativo, sulla stabilizzazione dei collegamenti che si instaurano tra i nodi figli. Questi collegamenti rivestono un ruolo fondamentale nel contesto del controllo logico e nella sequenza di attivazione dei nodi successivi, contribuendo in modo sostanziale alla progressione del processo decisionale complessivo.

```
[Test]
public void GroupDecisionNodeTest()
{
    // Action
```

```
    var trueAction =
        ScriptableObject.CreateInstance<DebugActionNode>();
    trueAction.SetMessage("True Action");
    var falseAction =
        ScriptableObject.CreateInstance<DebugActionNode>();
    falseAction.SetMessage("False Action");

    // Condition
    var conditionIntEquals =
        ScriptableObject.CreateInstance<IntEqualsConditionNode>();
    var conditionIntEquals1 =
        ScriptableObject.CreateInstance<IntEqualsConditionNode>();
    var conditionIntEquals2 =
        ScriptableObject.CreateInstance<IntEqualsConditionNode>();

    // Decision
    var groupDecision =
        ScriptableObject.CreateInstance<DecisionGroupVerifiableNode>();
    groupDecision.AddChild(conditionIntEquals);
    groupDecision.AddChild(conditionIntEquals1);
    groupDecision.AddChild(conditionIntEquals2);
    groupDecision.AddTrueAction(trueAction);
    groupDecision.AddFalseAction(falseAction);
    var groupDecisionClone = Object.Instantiate(groupDecision);

    // Exist
    Assert.NotNull(groupDecision);
    Assert.NotNull(groupDecisionClone);

    // Children
    Assert.AreEqual(3, groupDecision.GetChildrenCount());
    Assert.AreEqual(3, groupDecisionClone.GetChildrenCount());

    groupDecision
        .SetLogicPort(ScriptableObject.CreateInstance<AND_Logic>());
    groupDecisionClone
        .SetLogicPort(ScriptableObject.CreateInstance<AND_Logic>());
    conditionIntEquals.SetValue(1, 1);
    conditionIntEquals1.SetValue(1, 1);
    conditionIntEquals2.SetValue(1, 1);
    Assert.AreEqual(true, groupDecision.MakeDecision());
    Assert.AreEqual(true, groupDecisionClone.MakeDecision());

    groupDecision
```

```
    .SetLogicPort(ScriptableObject.CreateInstance<AND_Logic>());
     groupDecisionClone
    .SetLogicPort(ScriptableObject.CreateInstance<AND_Logic>());
     conditionIntEquals.SetValue(1, 1);
     conditionIntEquals1.SetValue(1, 0);
     conditionIntEquals2.SetValue(1, 1);
     Assert.AreEqual(false, groupDecision.MakeDecision());
     Assert.AreEqual(false, groupDecisionClone.MakeDecision());

     groupDecision
    .SetLogicPort(ScriptableObject.CreateInstance<OR_Logic>());
     groupDecisionClone
    .SetLogicPort(ScriptableObject.CreateInstance<OR_Logic>());
     conditionIntEquals.SetValue(1, 1);
     conditionIntEquals1.SetValue(1, 0);
     conditionIntEquals2.SetValue(0, 1);
     Assert.AreEqual(true, groupDecision.MakeDecision());
     Assert.AreEqual(true, groupDecisionClone.MakeDecision());

     groupDecision
    .SetLogicPort(ScriptableObject.CreateInstance<OR_Logic>());
     groupDecisionClone
    .SetLogicPort(ScriptableObject.CreateInstance<OR_Logic>());
     conditionIntEquals.SetValue(0, 1);
     conditionIntEquals1.SetValue(1, 0);
     conditionIntEquals2.SetValue(0, 1);
     Assert.AreEqual(false, groupDecision.MakeDecision());
     Assert.AreEqual(false, groupDecisionClone.MakeDecision());
}
```

Listato 4.4: Codice sorgente dei test del nodo di verifica

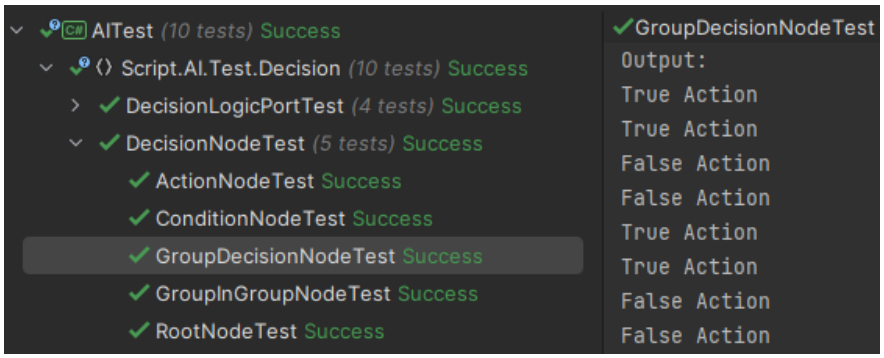

Figura 4.10: Controllo dei test del nodo di verifica

Questi test hanno anche lo scopo di confermare la corretta operatività del nodo clonato. Dato che il nodo clonato eredita gli stessi riferimenti dei nodi figli del nodo dal quale è stato duplicato, si procede a eseguire una verifica accurata per assicurarsi che i risultati prodotti dal nodo originale siano in perfetta coincidenza con quelli generati dal nodo clonato. Questa procedura sottolinea l'importanza di una clonazione senza discrepanze, contribuendo in modo sostanziale a garantire un impeccabile comportamento nel processo di replicazione.

Un altro test di notevole rilevanza si concentra sull'analisi approfondita del comportamento peculiare relativo alla capacità non solo di attribuire nodi di condizione come figli al nodo di verifica, ma anche nodi di verifica autonomi. Mediante l'esecuzione dei test, l'obiettivo principale è condurre una valutazione dettagliata che metta in luce la flessibilità e l'integrità del sistema dell'editor stesso. Si mira a evidenziare la sua competenza nel generare nodi di verifica in varie configurazioni, assicurando così la sua versatilità e affidabilità nell'ambito della gestione dei nodi di verifica.

```
[Test]
public void GroupInGroupNodeTest()
{
    // Action
    var trueAction =
        ScriptableObject.CreateInstance<DebugActionNode>();
    trueAction.SetMessage("True Action");
    var falseAction =
        ScriptableObject.CreateInstance<DebugActionNode>();
    falseAction.SetMessage("False Action");
    // Condition
    var conditionIntEquals =
        ScriptableObject.CreateInstance<IntEqualsConditionNode>();
    var conditionIntEquals1 =
        ScriptableObject.CreateInstance<IntEqualsConditionNode>();
    // Decision
    var groupDecision =
        ScriptableObject.CreateInstance<DecisionGroupVerifiableNode>();
    var groupDecision1 =
        ScriptableObject.CreateInstance<DecisionGroupVerifiableNode>();
    groupDecision
   .SetLogicPort(ScriptableObject.CreateInstance<AND_Logic>());
    groupDecision1
   .SetLogicPort(ScriptableObject.CreateInstance<AND_Logic>());

    Assert.NotNull(groupDecision);
```

```
    Assert.NotNull(groupDecision1);
    groupDecision1.AddChild(conditionIntEquals1);
    groupDecision.AddChild(conditionIntEquals);
    groupDecision.AddChild(groupDecision1);
    groupDecision.AddTrueAction(trueAction);
    groupDecision.AddFalseAction(falseAction);
    // Children
    Assert.AreEqual(2, groupDecision.GetChildrenCount());

    groupDecision
   .SetLogicPort(ScriptableObject.CreateInstance<AND_Logic>());
    groupDecision1
   .SetLogicPort(ScriptableObject.CreateInstance<AND_Logic>());
    conditionIntEquals.SetValue(1, 1);
    conditionIntEquals1.SetValue(1, 1);
    Assert.AreEqual(true, groupDecision.MakeDecision());

    groupDecision
   .SetLogicPort(ScriptableObject.CreateInstance<AND_Logic>());
    groupDecision1
   .SetLogicPort(ScriptableObject.CreateInstance<AND_Logic>());
    conditionIntEquals.SetValue(1, 1);
    conditionIntEquals1.SetValue(1, 0);
    Assert.AreEqual(false, groupDecision.MakeDecision());

    groupDecision
   .SetLogicPort(ScriptableObject.CreateInstance<OR_Logic>());
    groupDecision1
   .SetLogicPort(ScriptableObject.CreateInstance<OR_Logic>());
    conditionIntEquals.SetValue(1, 1);
    conditionIntEquals1.SetValue(1, 0);
    Assert.AreEqual(true, groupDecision.MakeDecision());

    groupDecision
   .SetLogicPort(ScriptableObject.CreateInstance<OR_Logic>());
    groupDecision1
   .SetLogicPort(ScriptableObject.CreateInstance<OR_Logic>());
    conditionIntEquals.SetValue(0, 1);
    conditionIntEquals1.SetValue(1, 0);
    Assert.AreEqual(false, groupDecision.MakeDecision());
}
```

Listato 4.5: Codice sorgente dei test del nodo di verifica dentro un nodo di verifica

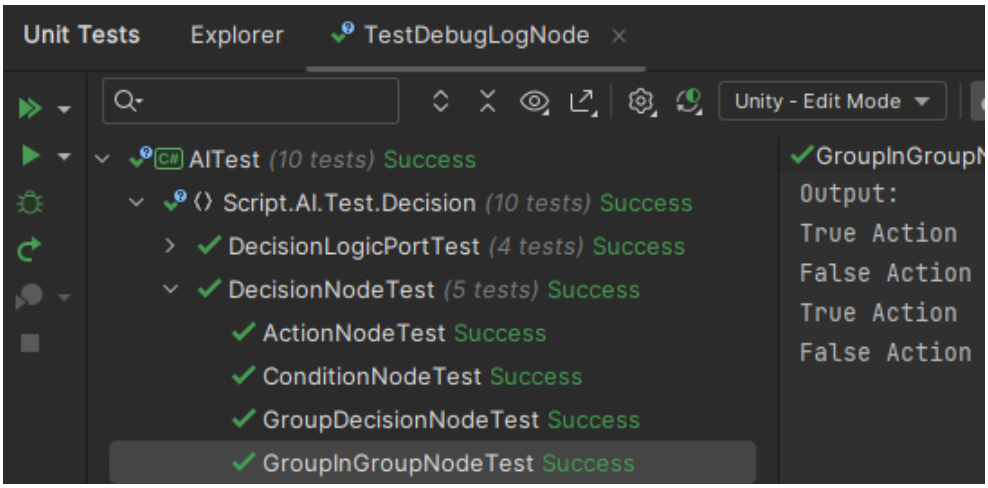

Figura 4.11: Controllo dei Test del nodo di verifica dentro un nodo di verifica

Nodo Radice

Il nodo radice rappresenta il punto di partenza fondamentale all'interno del processo dell'albero decisionale, avviando così la catena di nodi di verifica. È quindi di vitale importanza sottoporre il nodo radice a rigorosi test al fine di verificare la sua corretta operatività, nonché valutare la precisione del suo processo di creazione e clonazione. Questo approccio permette di attestare la robustezza dell'intero albero decisionale creato, garantendo la sua affidabilità e stabilità.

```
[Test]
public void RootNodeTest()
{
    // Action
    var trueAction =
        ScriptableObject.CreateInstance<DebugActionNode>();
    trueAction.SetMessage("True Action");
    var falseAction =
        ScriptableObject.CreateInstance<DebugActionNode>();
    falseAction.SetMessage("False Action");
    // Condition
    var conditionIntEquals =
        ScriptableObject.CreateInstance<IntEqualsConditionNode>();
    var conditionIntEquals1 =
        ScriptableObject.CreateInstance<IntEqualsConditionNode>();
    var conditionIntEquals2 =
        ScriptableObject.CreateInstance<IntEqualsConditionNode>();
    // Decision
```

```
    var groupDecision =
        ScriptableObject.CreateInstance<DecisionGroupVerifiableNode>();
    groupDecision.AddChild(conditionIntEquals);
    groupDecision.AddChild(conditionIntEquals1);
    groupDecision.AddChild(conditionIntEquals2);
    groupDecision.AddTrueAction(trueAction);
    groupDecision.AddFalseAction(falseAction);
    // Root
    var root = ScriptableObject.CreateInstance<DecisionRootNode>();
    root.SetDecision(groupDecision);
    var rootClone = Object.Instantiate(root);

    groupDecision
        .SetLogicPort(ScriptableObject.CreateInstance<AND_Logic>());
    conditionIntEquals.SetValue(1, 1);
    conditionIntEquals1.SetValue(1, 1);
    conditionIntEquals2.SetValue(1, 1);
    Assert.AreEqual(true, root.MakeDecision());
    Assert.AreEqual(true, rootClone.MakeDecision());
    groupDecision
        .SetLogicPort(ScriptableObject.CreateInstance<AND_Logic>());
    conditionIntEquals.SetValue(1, 1);
    conditionIntEquals1.SetValue(1, 0);
    conditionIntEquals2.SetValue(1, 1);
    Assert.AreEqual(false, root.MakeDecision());
    Assert.AreEqual(false, rootClone.MakeDecision());
    groupDecision
        .SetLogicPort(ScriptableObject.CreateInstance<OR_Logic>());
    conditionIntEquals.SetValue(1, 1);
    conditionIntEquals1.SetValue(1, 0);
    conditionIntEquals2.SetValue(0, 1);
    Assert.AreEqual(true, root.MakeDecision());
    Assert.AreEqual(true, rootClone.MakeDecision());
    groupDecision
        .SetLogicPort(ScriptableObject.CreateInstance<OR_Logic>());
    conditionIntEquals.SetValue(0, 1);
    conditionIntEquals1.SetValue(1, 0);
    conditionIntEquals2.SetValue(0, 1);
    Assert.AreEqual(false, root.MakeDecision());
    Assert.AreEqual(false, rootClone.MakeDecision());
}
```

Listato 4.6: Codice sorgente dei test del nodo radice

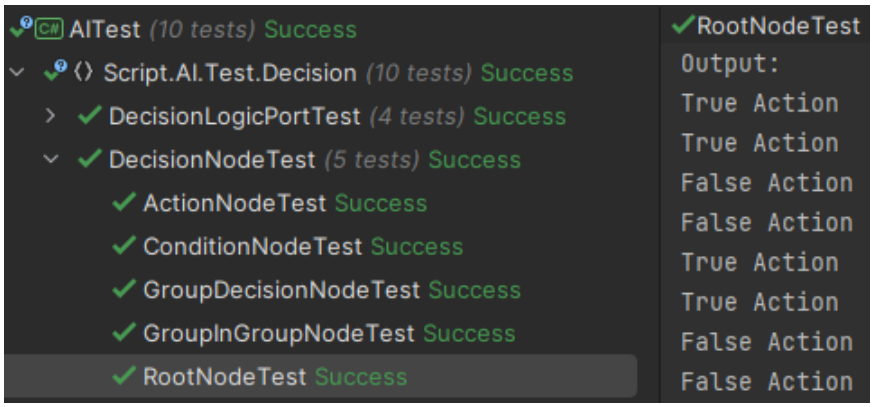

Figura 4.12: Controllo dei test del nodo radice

Albero decisionale

L'ultimo passo cruciale nel processo di verifica, volto a garantire la solidità e l'integrità di un albero decisionale, implica l'esame di un albero decisionale complesso, ovvero un albero che presenta una serie di percorsi decisionali distinti. L'obiettivo principale di questa fase di verifica è determinare con certezza se il risultato ottenuto è in perfetta conformità con le specifiche condizioni definite nei nodi di verifica lungo tutti i percorsi decisionali. Attraverso questa approfondita analisi, si conferma e si assicura il corretto funzionamento e il comportamento impeccabile degli elementi costituenti l'albero decisionale, contribuendo in modo sostanziale a garantire la sua stabilità e affidabilità complessiva.

```
[Test]
public void SimpleTreeTest()
{
    // Action
    var trueAction =
        ScriptableObject.CreateInstance<DebugActionNode>();
    trueAction.SetMessage("True Action");
    var falseAction =
        ScriptableObject.CreateInstance<DebugActionNode>();
    falseAction.SetMessage("False Action");
    // Condition
    var conditionIntEquals =
        ScriptableObject.CreateInstance<IntEqualsConditionNode>();
    var conditionIntEquals1 =
        ScriptableObject.CreateInstance<IntEqualsConditionNode>();
    var conditionIntEquals2 =
        ScriptableObject.CreateInstance<IntEqualsConditionNode>();
```

```
// Decision
var groupDecision = ScriptableObject
    .CreateInstance<DecisionGroupVerifiableNode>();
var groupDecision1 = ScriptableObject
    .CreateInstance<DecisionGroupVerifiableNode>();
var groupDecision2 = ScriptableObject
    .CreateInstance<DecisionGroupVerifiableNode>();

groupDecision
    .SetLogicPort(ScriptableObject.CreateInstance<AND_Logic>());
groupDecision1
    .SetLogicPort(ScriptableObject.CreateInstance<AND_Logic>());
groupDecision2
    .SetLogicPort(ScriptableObject.CreateInstance<AND_Logic>());

groupDecision.AddChild(conditionIntEquals);
groupDecision1.AddChild(conditionIntEquals1);
groupDecision2.AddChild(conditionIntEquals2);
Assert.AreEqual(1, groupDecision.GetChildrenCount());
Assert.AreEqual(1, groupDecision1.GetChildrenCount());
Assert.AreEqual(1, groupDecision2.GetChildrenCount());

groupDecision.AddTrueAction(trueAction);
groupDecision1.AddTrueAction(trueAction);
groupDecision2.AddTrueAction(trueAction);

groupDecision.AddFalseAction(falseAction);
groupDecision1.AddFalseAction(falseAction);
groupDecision2.AddFalseAction(falseAction);

groupDecision.AddFalseDecision(groupDecision1);
groupDecision.AddTrueDecision(groupDecision2);

// Root
var root = ScriptableObject.CreateInstance<DecisionRootNode>();
root.SetDecision(groupDecision);
// Tree
var tree = ScriptableObject.CreateInstance<DecisionTree>();
tree.SetRootNode(root);
var connectList = tree.GetMakeDecisionConnectList()
    .OfType<DecisionGroupVerifiableNode>().ToList();
Assert.AreEqual(3, connectList.Count);
// Test
conditionIntEquals.SetValue(1, 1);
```

```
    conditionIntEquals1.SetValue(1, 1); // false
    conditionIntEquals2.SetValue(1, 1); // true
    connectList.ForEach(node => node.ResetEvaluationData());
    tree.MakeDecision();
    Assert.AreEqual(EBoolEvaluation.True,
        groupDecision.GetEvaluationData());
    Assert.AreEqual(EBoolEvaluation.None,
        groupDecision1.GetEvaluationData());
    Assert.AreEqual(EBoolEvaluation.True,
        groupDecision2.GetEvaluationData());

    conditionIntEquals.SetValue(1, 0);
    conditionIntEquals1.SetValue(1, 1); // false
    conditionIntEquals2.SetValue(1, 1); // true
    connectList.ForEach(node => node.ResetEvaluationData());
    tree.MakeDecision();
    Assert.AreEqual(EBoolEvaluation.False,
        groupDecision.GetEvaluationData());
    Assert.AreEqual(EBoolEvaluation.True,
        groupDecision1.GetEvaluationData());
    Assert.AreEqual(EBoolEvaluation.None,
        groupDecision2.GetEvaluationData());

    conditionIntEquals.SetValue(1, 0);
    conditionIntEquals1.SetValue(1, 0); // false
    conditionIntEquals2.SetValue(1, 0); // true
    connectList.ForEach(node => node.ResetEvaluationData());
    tree.MakeDecision();
    Assert.AreEqual(EBoolEvaluation.False,
        groupDecision.GetEvaluationData());
    Assert.AreEqual(EBoolEvaluation.False,
        groupDecision1.GetEvaluationData());
    Assert.AreEqual(EBoolEvaluation.None,
        groupDecision2.GetEvaluationData());
}
```

Listato 4.7: Codice sorgente dei test dell'albero decisionale

Mediante la verifica dello stato di controllo dei nodi di verifica, si effettua un esame attento per identificare quali nodi vengono inclusi nel processo decisionale. Questo approccio consente di valutare l'evoluzione del percorso seguito all'interno del processo decisionale.

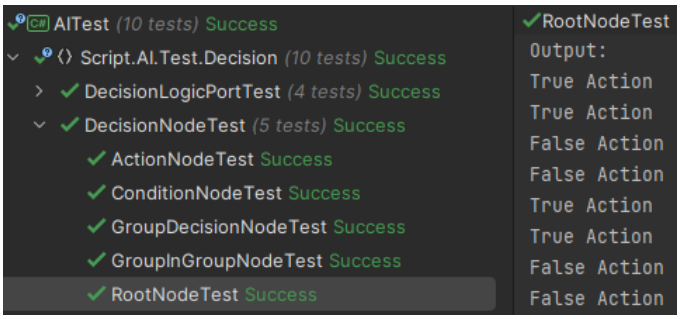

Figura 4.13: Controllo dei test dell'albero decisionale

Attraverso la fase di sviluppo e validazione dei test, è possibile inferire che gli elementi che saranno utilizzati nella composizione dell'editor non manifestano alcun errore. Questo processo di verifica e convalida ci fornisce una chiara indicazione riguardo alla robustezza e all'integrità degli alberi decisionali che saranno creati utilizzando l'editor.

4.1.3 Custom Nodes

Per conferire al sistema decisionale dell'albero la capacità di generare una vasta gamma di nodi decisionali o azioni all'interno dell'ambito di un gioco o di un combattimento a turni, è necessario sviluppare nodi che incorporino elementi specifici. Per i nodi di condizione, è imprescindibile la creazione di classi in grado di restituire valori booleani, i quali orienteranno la logica decisionale. Parallelamente, per i nodi d'azione, si dovranno concepire classi atte all'esecuzione di funzioni particolari, contribuendo così alla dinamica operativa del sistema.

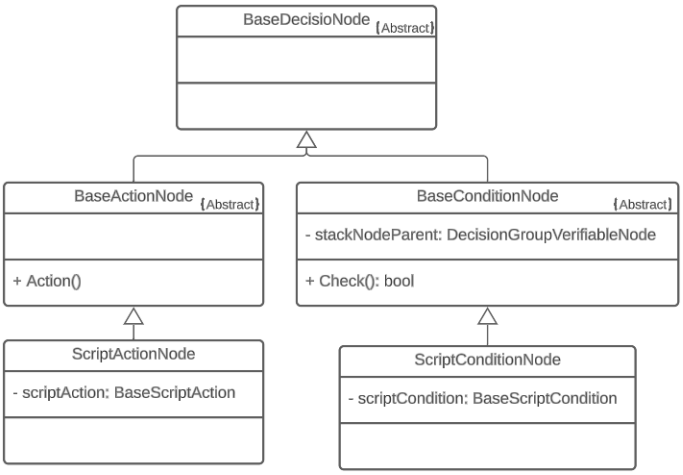

Figura 4.14: Schema UML delle classi per i nodi custom

Custom Script

Attraverso questa strategia, si mira a creare un nodo unificato denominato "ScriptConditionNode" per le condizioni e un altro denominato "ScriptActionNode" per le azioni. Questi nodi consentiranno l'associazione di varie classi al nodo corrispondente, consentendo uno scambio fluido e versatile di classi all'interno dei rispettivi nodi. Questa flessibilità intrinseca nella struttura dei nodi garantirà la possibilità di creare e espandere in modo agile, assicurando un approccio altamente adattivo e scalabile nel sistema complessivo.

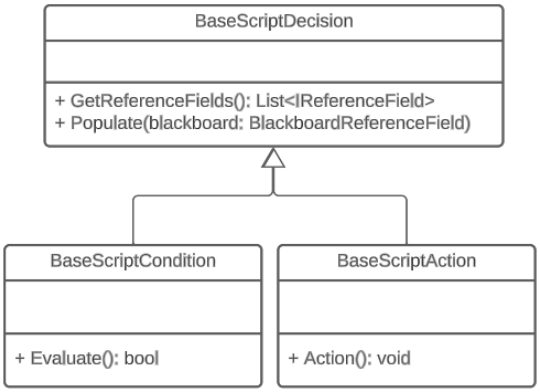

Figura 4.15: Schema UML dei nodi custom

Al fine di agevolare lo scambio e la conservazione delle classi personalizzate all'interno dei nodi pertinenti, si fa ricorso a uno strumento di Unity noto come "SerializeReference". Questo strumento consente di memorizzare una vasta gamma di classi, ognuna associata alla classe di base serializzata, consentendo così una gestione versatile e trasparente della struttura.

```
public class ScriptConditionNode : BaseConditionNode,
    IScriptDecisionNode
{
    [SerializeField,
        ButtonFindReferences(typeof(BaseScriptCondition))] private
        ScriptableObject btnCondition;
    [SerializeReference] private BaseScriptCondition scriptCondition;

    public void SetScript(BaseScriptCondition instance) =>
        this.scriptCondition = instance;

    public List<IReferenceField> GetReferenceFields() =>
        this.scriptCondition.GetReferenceFields();
```

```
    public void Populate(BlackboardReferenceField blackboard) =>
        this.scriptCondition.Populate(blackboard);

    public override bool Evaluate() =>
        this.scriptCondition.Evaluate();
}
```

Listato 4.8: Codice sorgente del nodo di condizione custom

All'interno di questo breve frammento di codice, viene presentato il nodo denominato "ScriptConditionNode". Conformemente alla spiegazione fornita in precedenza, tale nodo comprende un campo denominato "scriptCondition", dove viene effettuata la serializzazione tramite referenza della classe "BaseScriptCondition". Accanto a ciò, è presente un ulteriore campo denominato "btnCondition".

Questa specifica proprietà personalizzata è stata concepita per agevolare la gestione della creazione di finestre nell'editor per la "SerializeReference". Tale finestra richiede due parametri: il primo indica il tipo di classe, mentre il secondo stabilisce il campo al quale si desidera associare. Utilizzando il primo parametro e sfruttando il concetto di "reflection", la finestra è in grado di elencare tutti i tipi derivati dal suddetto parametro. Ciascuna voce nell'elenco ha la funzione di istanziare il tipo scelto e assegnarlo al campo corrispondente. In questa situazione, il campo "btnCondition" avrà la capacità di mostrare nell'editor tutte le classi che ereditano dalla classe di base "BaseScriptCondition".

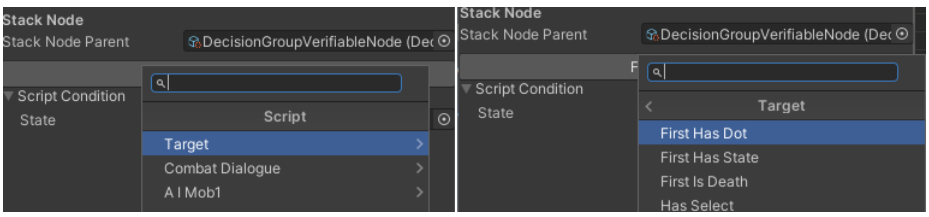

Figura 4.16: Finestra di ricerca dei custom script

Ricerca delle referenze dinamiche

Per acquisire i riferimenti agli oggetti a run-time, sia all'interno di una scena che in un prefab, è necessario disporre di un meccanismo che, durante il processo di clonazione dell'albero decisionale, assicuri il popolamento di tutti i campi con i rispettivi valori provenienti dalla scena o dal prefab. Questo processo consente di lavorare in modo fluido con le referenze clonate all'interno dell'ambiente di Unity. È stato elaborato un sistema basato su associazioni

chiave-valore, in cui nell'ambiente dell'editor è resa accessibile soltanto la chiave del riferimento desiderato. In alternativa, all'interno di una scena o nel contesto di un prefab, viene mostrata la chiave con il valore corrispondente da associare.

Questa procedura richiede l'esistenza di una classe incaricata sia della duplicazione dell'albero decisionale che del riempimento di tutti i campi che necessitano di riferimenti esterni. A tale scopo, è stata introdotta la classe "DecisionTreeRunner", ideata per gestire tutte le dinamiche precedentemente descritte. All'interno di questa classe, verranno messe a disposizione tutte le associazioni chiave-valore pertinenti all'albero decisionale associato.

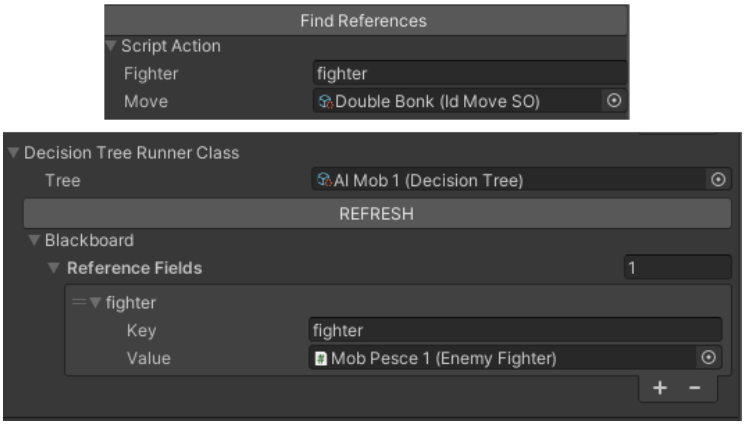

Figura 4.17: Componente runner per gli alberi decisionale

Inoltre, senza entrare nei dettagli, sono stati creati, come si può notare dalle immagini, attributi personalizzati di Unity. Questi attributi sono stati progettati per occultare il riferimento nell'editor dell'albero decisionale, rendendolo visibile esclusivamente all'interno della classe "Runner".

4.1.4 Editor

Per consentire la rappresentazione attraverso l'editor di tutti i meccanismi delineati dell'albero decisionale, Unity mette a disposizione il linguaggio "UXML" per le componenti grafiche e il linguaggio "USS" per la manipolazione dell'aspetto visivo dei componenti, con regole analoghe a quelle del "CSS". Questi elementi sono poi collegati a una classe C# per garantire un'integrazione completa.

Al fine di agevolare il processo di creazione, si fa uso di un'utilissima risorsa messa a disposizione da Unity: lo "UI Builder". Senza entrare nei dettagli specifici, questo strumento consente di interagire con una piattaforma grafica

per manipolare i file "UXML" e "USS". Tra le sue caratteristiche, è incluso anche un'anteprima della struttura complessa.

Naturalmente, per la generazione di ciascun componente, è stato sviluppato un sistema graduale che parte dai componenti più generici per poi specializzarli progressivamente, al fine di avere un riuso delle componenti e di ottenere l'editor dell'albero decisionale desiderato.

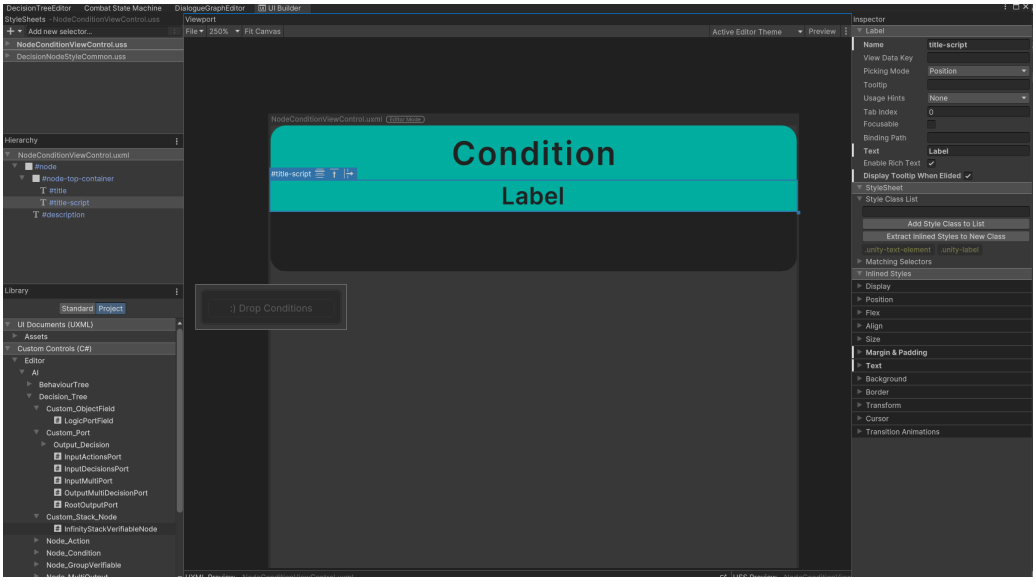

Figura 4.18: Ambiente di sviluppo "Builder" per elementi grafici

Come evidente nell'illustrazione, è possibile notare che nella parte inferiore sinistra dell'immagine sono presenti tutti i componenti personalizzati, utili per la creazione di ciascun nodo nell'albero decisionale.

Nodi dell'albero decisionale

Per ogni tipo di nodo è stata sviluppata una rappresentazione grafica apposita, in cui sono state integrate tutte le funzionalità necessarie per interagire con ciascun nodo.

Per il nodo radice, è stata integrata una porta per la creazione di archi, mirando a connettere nodi di verifica. Nei nodi d'azione, è presente una porta destinata a collegare l'arco proveniente dai nodi di verifica. Per quanto riguarda i nodi di condizione, essi costituiscono nodi semplici privi di connessioni, ma presentano la possibilità di essere inseriti all'interno dello stack dei nodi di verifica.

Infine, per i nodi di verifica, viene fornito un contenitore di nodi verificabili, tra cui nodi di condizione e nodi di verifica stessi. In aggiunta, sono presenti

diverse porte che consentono di collegarsi alle azioni e/o ad altri nodi di verifica. Inoltre comprende anche un campo per modificare la porta logica corrente.

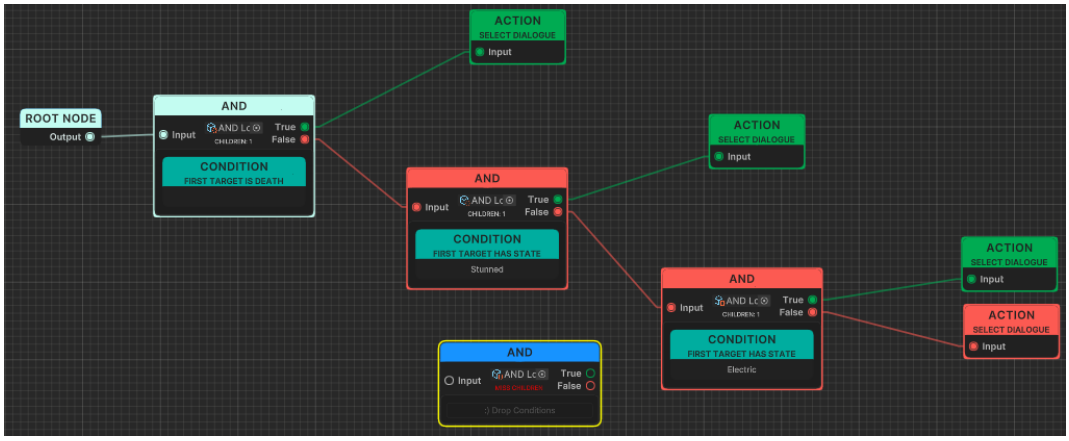

Figura 4.19: Editor degli alberi decisionale

Si è inoltre ideato un sistema cromatico appositamente per agevolare l'intuitività nell'interpretazione dei nodi decisionali. Questo sistema fonde i colori degli archi connessi al nodo, contribuendo a una rappresentazione visuale agevole e intuitiva. Nel contesto della fase di generazione dell'albero decisionale, sono state implementate delle scorciatoie progettate appositamente per semplificare il procedimento di creazione dell'albero stesso e l'accesso all'editor. Queste scorciatoie mirano a ottimizzare l'esperienza dell'utente, consentendo un accesso rapido alle funzionalità chiave durante la creazione.

In aggiunta, è stato sviluppato un sistema di visualizzazione che riveste un ruolo fondamentale. Questo sistema è stato concepito per mostrare in modo esaustivo ed efficiente solamente l'albero decisionale selezionato, eliminando il rumore visivo derivante da altri elementi o componenti. Questo approccio mirato è particolarmente rilevante all'interno del contesto del progetto di Unity, in quanto ottimizza l'efficienza e la chiarezza nell'analisi e nella modifica dell'albero decisionale.

Infine, è stato sviluppato un sistema di colori altamente reattivo durante l'esecuzione, il cui obiettivo è visualizzare graficamente il percorso seguito dalle decisioni all'interno dell'albero decisionale. Questo strumento si dimostra utile per migliorare la costruzione stessa dell'albero decisionale e per esaminare i percorsi generati nel corso del processo decisionale. I valori booleani derivanti dai nodi di condizione e verifica sono rappresentati visivamente, enfatizzati attraverso l'uso dei colori corrispondenti, mentre i nodi che effettuano una decisione sono distintamente evidenziati mediante la colorazione.

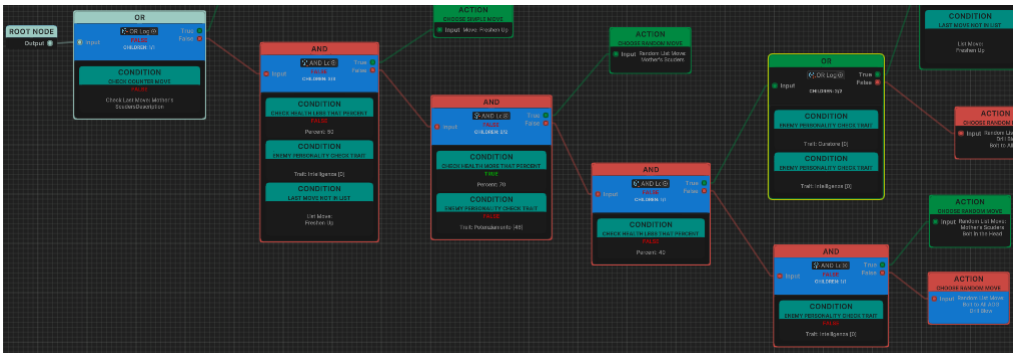

Figura 4.20: Sistema dei colori a run-time

4.2 Fighter

Nella presente sezione, verrà esaminata l'applicazione dei modelli basati su alberi decisionali per la gestione delle strategie dei nemici, inclusa la progettazione del componente responsabile della creazione iniziale di un insieme di alberi decisionali, necessari per l'intelligenza artificiale dei nemici durante il combattimento.

Il componente incaricato dell'inizializzazione degli alberi decisionali svolge un processo complesso che inizia con la duplicazione degli alberi decisionali originali, seguito da un'attenta assegnazione dei parametri dinamici del gioco a ciascun nodo dell'albero in base alle specifiche necessità. Questa procedura garantisce che gli alberi decisionali risultanti siano completamente pronti per l'utilizzo, incorporando tutte le informazioni rilevanti e le configurazioni necessarie per guidare in modo efficace il comportamento dei nemici nel contesto del gioco.

```
public class DecisionMultiTreeRunner : MonoBehaviour
{
    [Header("Editor")]
    [SerializeField] private DecisionTree editorDecisionTree;
    [Header("Multi Tree")]
    [SerializeField] private List<DecisionTree> treeList = new();
    [SerializeField] private BlackboardReferenceField blackboard;
    private Dictionary<DecisionTree, DecisionTree> treeDictionary;
    private bool isInit = false;

    public void Init()
    {
        if (this.isInit) return;
        this.isInit = true;
```

```
        this.treeDictionary = this.treeList.ToDictionary(tree =>
            tree, tree => tree.Clone());
        this.treeDictionary.Values.ToList().ForEach(tree =>
            tree.Init());
        this.treeDictionary.Values.ToList().ForEach(pair =>
            pair.Populate(this.blackboard));
    }
}
```

Listato 4.9: Codice sorgente del componente Runner per più alberi decisionale

Grazie a questo componente, il modulo del fighter sarà in grado di ospitare un ampio numero di alberi decisionali, consentendo la creazione di diverse configurazioni di processi decisionali per generare nuovi comportamenti. Questa caratteristica conferisce un alto grado di versatilità e personalizzazione al processo decisionale relativo ai nemici, permettendone l'espansione e la modifica in modo agevole per adattarlo alle esigenze specifiche del contesto di gioco. In questo caso di studio sono richiesti due alberi decisionali per i nemici: uno per determinare la strategia di scelta per la mossa e l'altro per decidere il bersaglio su cui eseguire la mossa.

```
public abstract class BaseAIRuleSets : MonoBehaviour
{
    [SerializeField, ReadOnly] private BaseFighter myFighter;
    [Header("Last Mov1e")]
    [SerializeField, ReadOnly] private IdMoveSO lastMove;
    [Header("Tree")]
    [SerializeField] private DecisionTree moveTree;
    [SerializeField] private DecisionTree targetTree;
    [Header("Multi Tree")]
    [SerializeField] private DecisionMultiTreeRunnerClass multiTree;

    public void MakeDecision()
    {
        this.multiTree.TryMakeDecision(this.moveTree);
        this.multiTree.TryMakeDecision(this.targetTree);
        this.lastMove = TargetManager.Instance.GetMove()?.GetId();
        this.ActionOnMakeDecision();
    }
}
```

Listato 4.10: Codice sorgente dedicata all'AI dei nemici

4.2.1 Personalità

Oltre all'impiego degli alberi decisionali come strumento chiave, si è inoltre sviluppato un sofisticato meccanismo denominato "sistema delle personalità". Questo sistema è progettato con l'obiettivo di modulare l'architettura del processo decisionale in base alle caratteristiche distintive e alle peculiarità intrinseche dei personaggi all'interno del contesto di gioco. La sua funzione principale è quella di conferire una profonda e articolata personalizzazione al comportamento dei personaggi, consentendo loro di manifestare reazioni e scelte coerenti con la loro individualità, contribuendo così a una rappresentazione più realistica e coinvolgente dei personaggi nel contesto del gioco.

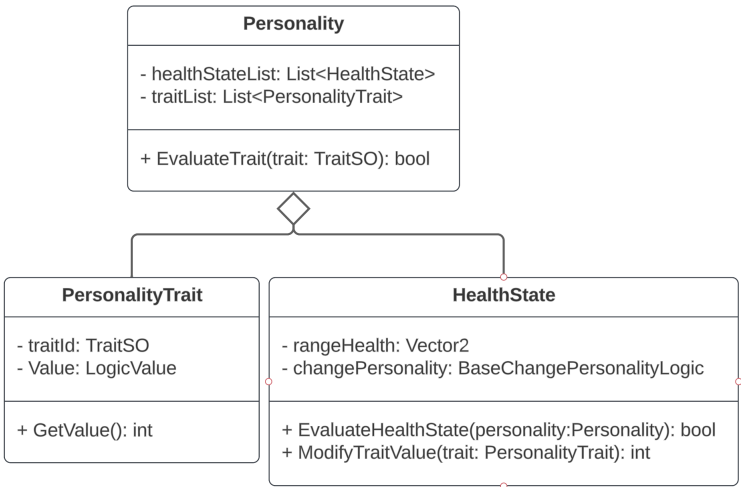

Figura 4.21: Schema UML delle personalità dei nemici

Come illustrato nell'UML dell'architettura relativa alla personalità, la classe "Personality" presenta due componenti fondamentali. La prima di queste strutture è dedicata all'archiviazione di tutti i tratti caratteristici della personalità, mentre la seconda struttura è progettata per ospitare i vari stati di una state machine, che consente la modifica dinamica dei valori associati a tali tratti, in base al livello di vita. La parte successiva, sarà dedicata ad approfondire in modo dettagliato di entrambi questi componenti.

4.2.2 Tratti

Nel quadro di questa metodologia di personalità, è essenziale comprendere che i tratti di personalità sono i pilastri sui quali si basa l'identità e il comportamento dei personaggi. Ciascuno di questi tratti sarà determinato da logiche specifiche, progettate con l'obiettivo di regolare in maniera dettagliata

il loro valore all'interno del sistema. Inoltre, al fine di garantire un'identificazione inequivocabile e agevolarne la gestione, a ciascun tratto sarà associato un identificativo unico. Questo approccio mira a sottolineare l'importanza fondamentale dei tratti di personalità nella struttura decisionale dei personaggi all'interno dell'ambiente di gioco.

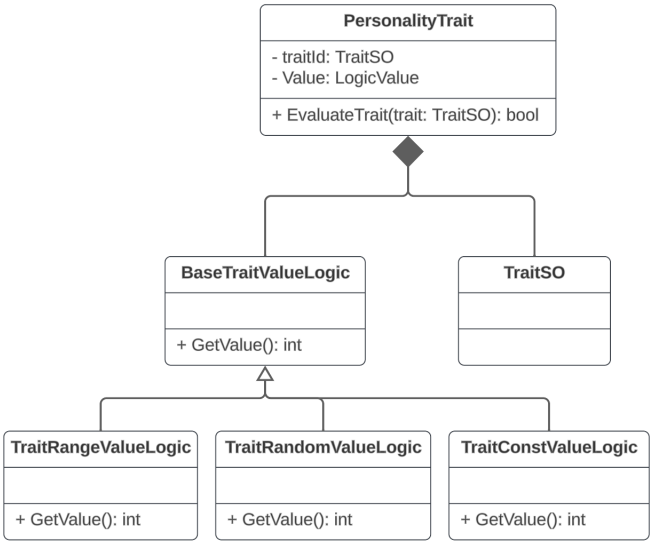

Figura 4.22: Schema UML dei tratti della personalità

All'interno di questo schema UML, è evidente la presenza di tre logiche distintive, ciascuna delle quali è stata progettata per acquisire e determinare il valore associato a un tratto specifico della personalità. Per gestire in modo organizzato e flessibile queste diverse logiche, si è adottato un approccio basato sul pattern comportamentale "strategy". L'obiettivo fondamentale di questo pattern è creare una struttura che consenta di definire una famiglia di algoritmi, di incapsularli individualmente e di renderli intercambiabili all'interno del sistema. Tale approccio offre la possibilità di adattare in modo agevole il comportamento del sistema a diverse esigenze, fornendo una gestione dinamica e modularità alle logiche utilizzate per determinare i valori dei tratti di personalità.

In aggiunta, è stata implementata una serie di soluzioni progettuali al fine di agevolare l'interscambiabilità delle logiche per il valore, una caratteristica chiave offerta dal pattern "strategy", all'interno dell'ambiente di sviluppo Unity. Grazie a questa tecnologia, è possibile apportare modifiche alla logica di calcolo dei valori dei tratti di personalità in modo rapido e semplice, direttamente attraverso l'interfaccia utente del pannello "Inspector" di Unity, richiedendo solamente un singolo clic.

Nodo custom per la personalità

Dopo aver definito in modo esaustivo l'architettura delle personalità, diventa di fondamentale importanza condurre un'analisi approfondita circa l'integrazione di questa complessa componente all'interno delle strutture altamente strutturate degli alberi decisionali. Di conseguenza, si rende necessario istituire con attenzione e precisione delle componenti specializzate nell'ambito dell'editor, allo scopo di facilitare la gestione delle dinamiche delle personalità associate ai personaggi. Questo elemento all'interno dell'editor è basato su una funzionalità avanzata che permette di creare nodi di condizione altamente personalizzati. In particolare, si procede con la creazione di un nodo di condizione specializzato che, attraverso l'uso della probabilità, effettua una valutazione del valore associato a un tratto specifico della personalità del personaggio.

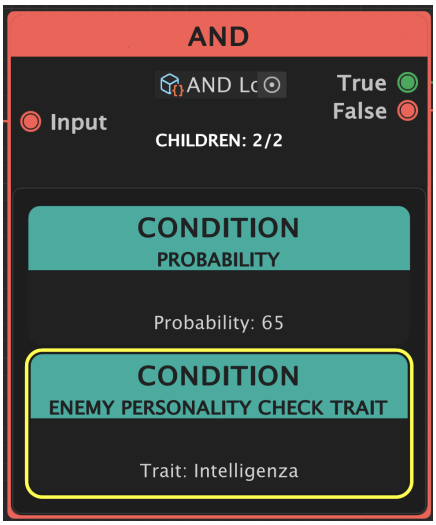

Figura 4.23: Nodi di condizione custom per la personalità

Nella figura è illustrato un nodo personalizzato che ha il compito di effettuare una valutazione basata sulla probabilità in relazione al tratto di intelligenza. Questa rappresentazione visiva sottolinea in modo esplicito il ruolo attivo e cruciale svolto da questo nodo all'interno della complessa architettura dell'albero decisionale, evidenziando come la sua presenza e le sue azioni influenzino in modo determinante il flusso decisionale globale all'interno del sistema. La notevole adattabilità del nodo personalizzato si traduce nella possibilità di effettuare modifiche al tratto sottoposto a valutazione con un alto grado di flessibilità. Questa operazione diventa agevole grazie all'utilizzo dell'interfaccia dell'editor, nota come "Inspector," che consente un'interazione intuitiva e semplificata con il nodo, permettendo così una personalizzazione precisa e dinamica del comportamento del sistema.

Figura 4.24: Inspector del nodi di condizione della personalità

La natura intrinsecamente flessibile dell'editor si manifesta nella sua abilità di consentire la creazione di nodi altamente personalizzati. Questa caratteristica fondamentale apre la porta a un vasto ventaglio di possibilità, in quanto consente agli sviluppatori di ideare e implementare una varietà di nodi che traggono vantaggio dai dati relativi alle personalità dei personaggi. Questo processo non solo agevola la creazione di nuove logiche di comportamento basate su tali dati di personalità, ma anche la semplice integrazione di nuovi nodi all'interno dell'editor stesso, ampliando così in modo significativo le opzioni disponibili per la personalizzazione e la diversificazione del comportamento all'interno del sistema.

4.2.3 Cambiamento in base al livello di vita

Il componente responsabile della modifica dinamica dei tratti della personalità di un personaggio è una sofisticata state machine. Questa state machine opera in modo reattivo, adattandosi in tempo reale alle fluttuazioni del livello di vita correlato al personaggio. La state machine in questione è costituita da una serie di stati distinti, ciascuno dei quali, una volta attivato, esercita un'influenza specifica e significativa sulla percezione e sull'interpretazione del valore associato al tratto di personalità specificamente scelto.

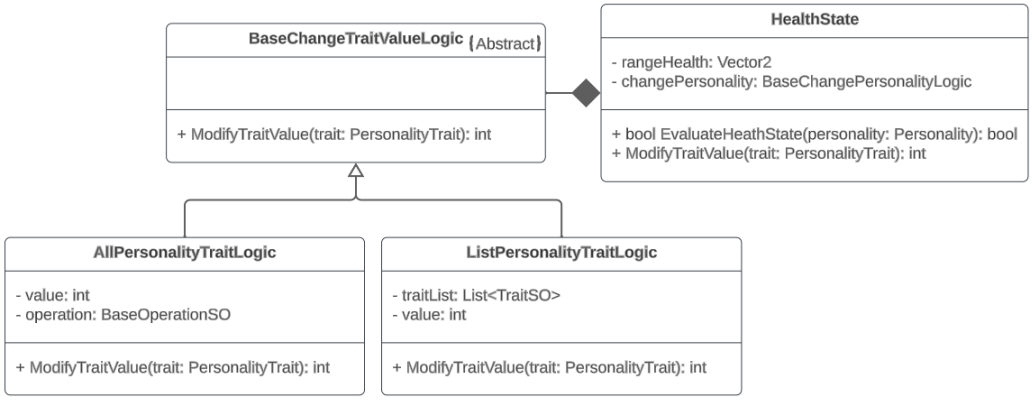

Figura 4.25: Schema UML della macchina a stati per la personalità

Nel contesto dello schema UML, emerge chiaramente una netta distinzione tra la logica intrinseca relativa allo stato e la logica specificamente dedicata alla gestione delle dinamiche di modifica dei tratti di personalità. La metodologia usata per la creazione dei valori dei tratti della personalità, è stata usata anche per regolare in modo fluido le variazioni nei valori dei tratti di tale personalità, sfruttando in modo particolarmente efficace il noto pattern comportamentale, denominato "Strategy".

Inoltre, nel corso del progetto, è stata sviluppata e integrata con successo una componente specifica che svolge un ruolo fondamentale all'interno del sistema dello stato. Questa componente è progettata per incapsulare una vasta gamma di operazioni matematiche di base, tra cui, ma non limitandosi a, somma e sottrazione. Tale implementazione è stata attentamente concepita al fine di garantire la massima flessibilità e interscambiabilità delle operazioni coinvolte durante la modifica dei valori dei tratti di personalità, contribuendo in modo sostanziale a rendere il sistema dello stato altamente adattabile e versatile. Nello specifico, il componente denominato "BaseOperationSO" è stato progettato con l'obiettivo di incarnare il comportamento precedentemente delineato. Grazie alla sua struttura gerarchica, esso offre un elevato grado di scalabilità, sia in termini di architettura complessiva che di logica sottostante, contribuendo così in modo significativo alla flessibilità del sistema dello stato.

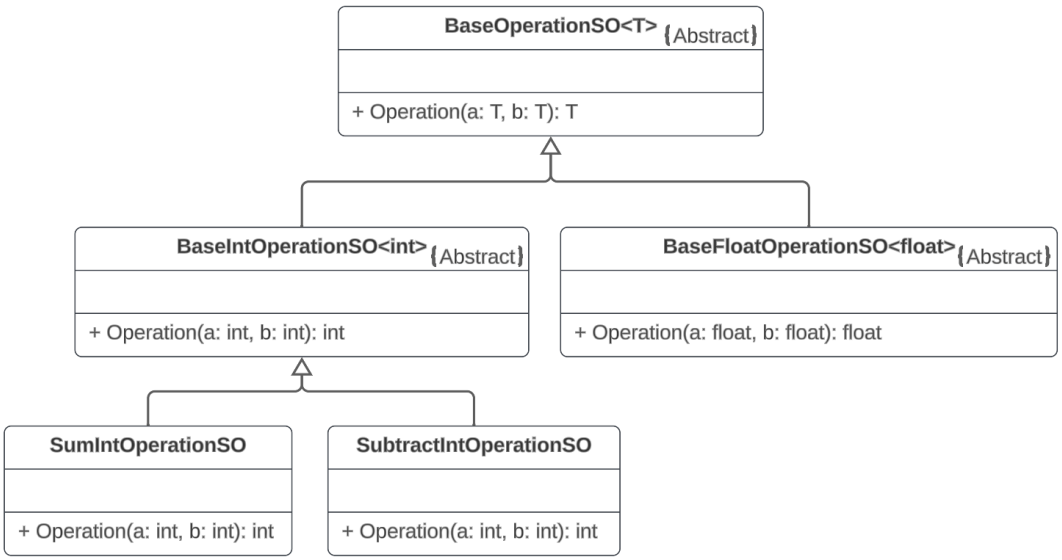

Figura 4.26: Schema UML delle operazioni per la logica dalla macchina a stati

4.3 Elementi del combattimento

In questa sezione, verrà condotto un esame approfondito dell'implementazione delle logiche necessarie per consentire all'editor di acquisire una chiara comprensione degli eventi del gioco. L'obiettivo principale è quello di stabilire un quadro che permetta l'interpretazione degli eventi di gioco e la conseguente reattività all'interno dell'albero decisionale, fornendo così all'editore tutti gli strumenti indispensabili per una gestione ottimale del sistema. Pertanto, è necessario condurre un'analisi dettagliata degli aspetti fondamentali del combattimento, tra i quali figurano il fighter, la mossa, e gli elementi come gli stati, i dot e gli hot.

4.3.1 Fighter

I personaggi, indiscutibilmente centrali nell'ambito delle interazioni durante il combattimento, richiedono un approccio altamente personalizzato attraverso la creazione di nodi specifici, sia di tipo condizionale che azionale. Questa pratica fornisce all'editore un potente strumentario per interpretare e analizzare in dettaglio gli eventi del gioco che coinvolgono direttamente il personaggio, conferendogli la capacità di reagire in modo altamente strategico e adattabile alle molteplici dinamiche che emergono durante le fasi di combattimento.

Al fine di analizzare le diverse logiche potenzialmente applicabili ai nodi di condizione personalizzati, è possibile prendere in considerazione due esemplificazioni specifiche. Questi due esempi rappresentano concretamente strategie operative che operano in modo diretto e incisivo sul livello di vita del personaggio, dimostrando così l'ampiezza delle possibilità di personalizzazione e adattamento all'interno dell'editor.

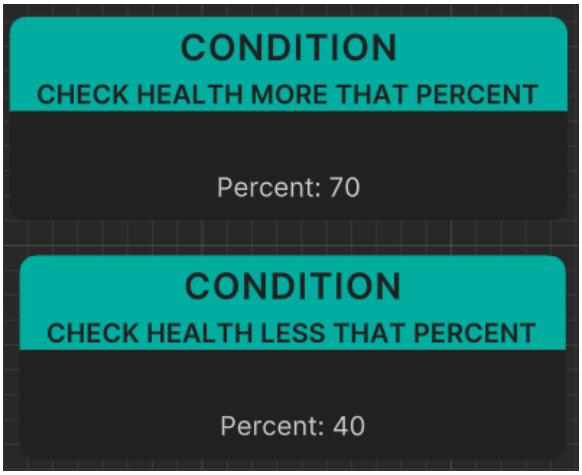

Figura 4.27: Esempio di nodi di condizione per i fighters

Nelle rappresentazioni grafiche dei nodi presenti nella figura, è possibile distinguere due differenti logiche operative. La prima di queste logiche calcola accuratamente la percentuale di vita del personaggio e successivamente effettua un controllo per verificarne il valore attuale rispetto a una soglia prestabilita, in particolare, se il livello di vita supera la percentuale stabilita. La seconda logica, invece, esamina attentamente il livello di vita del personaggio e verifica se esso è al di sotto della percentuale desiderata, offrendo in questo modo due diverse prospettive di gestione dell'aspetto relativo alla vita del personaggio all'interno del sistema.

I nodi precedentemente esaminati sono stati progettati per attivare la loro logica in base all'appartenenza al personaggio associato. Tuttavia, in determinate circostanze, può sorgere la necessità di sviluppare logiche altamente personalizzate relative ai bersagli delle mosse avversarie, ovvero i target. Per ulteriori approfondimenti, verrà analizzata l'implementazione di nodi d'azione altamente specifici per la selezione dei bersagli, al fine di mettere a disposizione dell'editor gli strumenti necessari per il processo decisionale riguardante la scelta dei bersagli per ciascun personaggio coinvolto.

Figura 4.28: Esempio di nodi di azione per i fighters

Nella rappresentazione grafica fornita, sono illustrate tre diverse strategie logiche per la selezione del bersaglio. La prima di queste logiche opta per la scelta del bersaglio in base al valore di vita più elevato, la seconda seleziona casualmente un personaggio come bersaglio, mentre la terza logica predilige il personaggio con il valore di vita più basso come bersaglio designato.

4.3.2 Mosse

È importante sottolineare che le mosse non si limitano a essere soltanto eventi centrali all'interno del contesto del combattimento, ma sono anche intrinsecamente legate alle reazioni a tali eventi. Di conseguenza, i dati associati alle mosse rivestono un ruolo cruciale nell'elaborazione della logica che sottende ai nodi personalizzati all'interno dell'editor. Questa logica è destinata ad essere impiegata sia per la definizione dei nodi d'azione che per quelli di condizione, garantendo un approccio completo e integrato alla gestione delle dinamiche di combattimento all'interno del sistema dell'editor.

A titolo di esempio, si può considerare un nodo di condizione di particolare rilevanza, quale ad esempio la valutazione dell'ultima mossa eseguita dal personaggio. Questo presupposto conduce alla concezione di soluzioni di design che integrano e implementano tale logica all'interno del sistema.

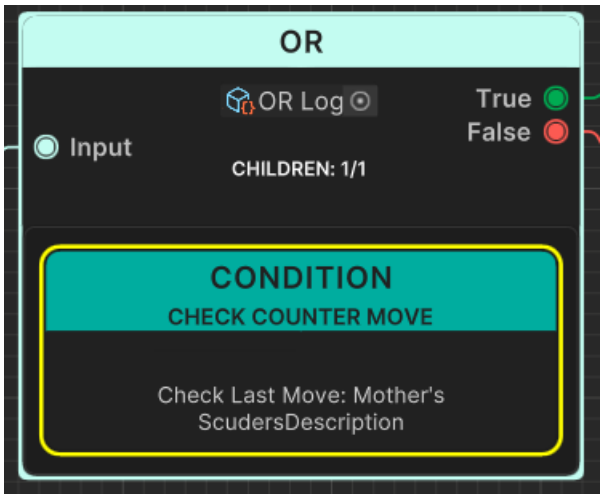

Figura 4.29: Esempio di nodo di condizione per le mosse

Questo particolare nodo di condizione personalizzato è stato progettato per svolgere una specifica funzione: prendere in ingresso l'identificativo di una mossa come parametro e condurre una verifica approfondita per determinare se la mossa specificata corrisponda all'ultima mossa eseguita in precedenza dal personaggio. La caratteristica chiave di questa implementazione è la sua notevole flessibilità nel consentire il passaggio di qualsiasi identificativo di mossa come parametro di input. Tale flessibilità, a sua volta, apre le porte a una vasta gamma di potenziali logiche e scenari di utilizzo che possono essere attuati con successo all'interno del sistema.

Dentro l'ampio panorama dei nodi reattivi, della famiglia dei nodi d'azione, emerge un nodo particolarmente rilevante, rappresentato dall'azione di

selezione della mossa da parte del personaggio durante il complesso contesto del combattimento. Attraverso l'impiego di varie metodologie logiche tese a ottenere in maniera efficiente e coerente la mossa più appropriata in un determinato momento, si apre ampiamente la porta alla creazione di nodi d'azione personalizzati. Questi nodi personalizzati assumono la responsabilità di condurre una scelta della mossa desiderata, adattandosi alle specifiche esigenze e dinamiche del sistema in cui operano.

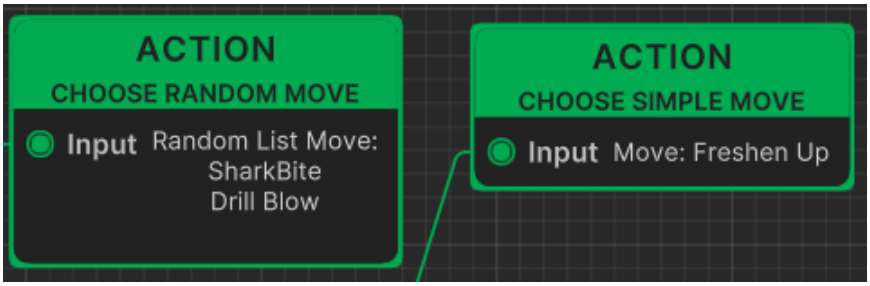

Figura 4.30: Esempio di nodo di azione per le mosse

Nell'illustrazione fornita, sono presentate due alternative logiche per effettuare la selezione della mossa. La prima di queste logiche implica l'estrarre casualmente una mossa dalla lista disponibile, mentre la seconda logica si concentra sulla scelta deliberata di una mossa specifica in base a determinati criteri o strategie.

4.3.3 Componenti: Stati, Dot e Hot

Nell'ambito dell'analisi dei componenti, si procede all'esame degli elementi conclusivi, ovvero gli stati, i dot e gli hot. Per l'implementazione di tali componenti, si adotta la stessa metodologia precedentemente applicata con successo per la creazione di nodi personalizzati, utilizzata sia per le mosse che per i personaggi, al fine di garantire tutti gli strumenti necessari per la lettura degli eventi durante il combattimento e la reazione a tali eventi.

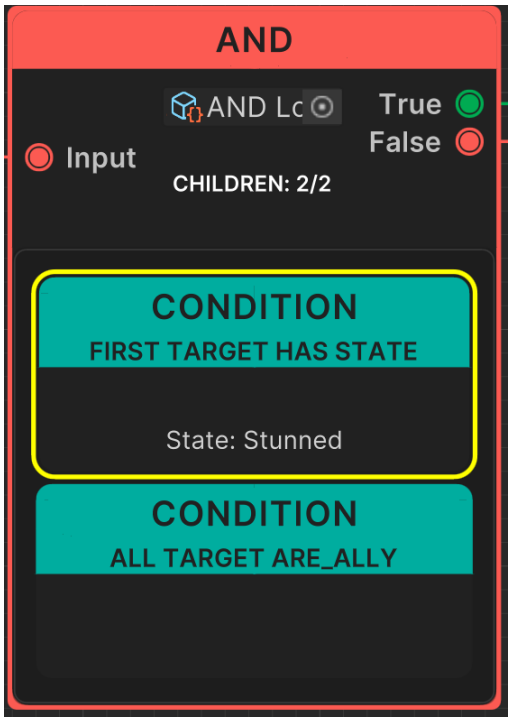

Figura 4.31: Esempio di nodi di condizione per i componenti

Nell'illustrazione presente, è evidenziata una delle potenziali strategie da adottare per la progettazione di un nodo personalizzato, in questo caso, un nodo appartenente alla categoria delle condizioni. I dati associati a ciascun componente (stati, dot e hot) possono essere elaborati attraverso varie logiche, producendo un valore booleano come risultato. Nel contesto di questo esempio, viene messa in luce una verifica specifica riguardante la presenza o l'assenza di uno stato associato a un personaggio. Questo schema logico può essere replicato in modo analogo sia per gli oggetti Dot che per gli effetti degli hot.

Per concludere questa sezione, è stato effettuato un processo completo di implementazione, includente tutti gli strumenti fondamentali e necessari all'editor. L'obiettivo primario di questo sforzo è fornire all'editor la versatilità e la capacità necessaria per creare alberi decisionali specificamente destinati ad essere utilizzati nei confronti dei nemici presenti all'interno del contesto di combattimento del gioco. È importante sottolineare che l'editor è stato progettato con un'ottica altamente modulare, consentendo l'integrazione agevole e l'ampliamento dei requisiti di strumenti futuri necessari per il gioco. Questo approccio mette in evidenza la possibilità di ottenere una completa e totale scalabilità nella creazione di alberi decisionali sempre più distinti e sofisticati, consentendo una piena adattabilità alle esigenze future del gioco.

Capitolo 5

Dati analitici degli alberi creati

In questo capitolo, viene condotto un approfondito studio dei dati generati dagli alberi decisionali, concentrando l'attenzione in modo specifico su quelli progettati per gestire l'intelligenza artificiale dei nemici durante il combattimento. Tali alberi decisionali sono creati mediante l'utilizzo dell'editor, con l'obiettivo di esaminare dettagliatamente gli aspetti precedentemente esposti nel terzo capitolo della tesi. All'interno di questa analisi, uno dei temi centrali è costituito dal processo decisionale che guida la selezione delle azioni intraprese dai nemici e la scelta dei bersagli a cui indirizzarle. Questo processo riveste una rilevanza critica nella dinamica dei combattimenti, influenzando direttamente la sfida proposta ai giocatori.

Inoltre, vengono considerati i dettagli relativi al processo di definizione dei tratti di personalità dei personaggi nemici. Questi attributi svolgono un ruolo determinante nel modellare il comportamento degli avversari, contribuendo a caratterizzarli in termini di aggressività, intelligenza, o altre caratteristiche che incidono sulle loro scelte e strategie di gioco. Infine, si esamina il funzionamento del sistema della state machine, una componente che opera in tempo reale per regolare i valori dei tratti di personalità associati ai personaggi nemici, in risposta alle variazioni nel loro livello di salute. Questo meccanismo aggiunge un livello di adattabilità al comportamento dei nemici, creando un'esperienza di gioco dinamica e stimolante.

In sintesi, questo capitolo ci offre l'opportunità di esplorare dettagliatamente il complesso mondo degli alberi decisionali utilizzati per gestire le azioni dei nemici durante i combattimenti nel gioco, analizzando in profondità i processi decisionali, la definizione dei tratti di personalità e il funzionamento della state machine, senza dimenticare il loro impatto sulla sfida proposta ai giocatori.

Al fine di esaminare gli aspetti precedentemente delineati, si procede all'analisi dei processi decisionali connessi al personaggio del boss all'interno del gioco.

5.1 Scelta della Mossa

	A	B	C	D	E	F
1		NODE NAME	LOGIC PORT	RESULT	DESCRIPTION	DESCRIPTION RESULT
2	MAIN NODE	Verifiable Group	OR	False		
3		Condition		False	CHECK COUNTER MOVE	Check Last Move: Mother's Scuders [0 / 2]
4	MAIN NODE	Verifiable Group	AND	False		
5		Condition		False	CHECK HEALTH LESS THAT PERCENT	
6		Condition		None	ENEMY PERSONALITY CHECK TRAIT	Trait: Intelligenza [0][0]
7		Condition		None	LAST MOVE NOT IN LIST	
8	MAIN NODE	Verifiable Group	AND	False		
9		Condition		True	CHECK HEALTH MORE THAT PERCENT	
10		Condition		False	ENEMY PERSONALITY CHECK TRAIT	Trait: Potenziamento [26][26]
11	MAIN NODE	Verifiable Group	AND	False		
12		Condition		False	CHECK HEALTH LESS THAT PERCENT	
13	MAIN NODE	Verifiable Group	AND	True		
14		Condition		True	ENEMY PERSONALITY CHECK TRAIT	Trait: Intelligenza [70][70]
15		Action			CHOOSE RANDOM MOVE	Mother's Scuders
16						
17	MAIN NODE	Verifiable Group	OR	True		
18		Condition		True	CHECK COUNTER MOVE	Check Last Move: Mother's Scuders [1 / 2]
19	MAIN NODE	Verifiable Group	OR	True		
20		Condition		True	ENEMY PERSONALITY CHECK TRAIT	Trait: Intelligenza [76][76]
21		Condition		None	ENEMY PERSONALITY CHECK TRAIT	Trait: Aggressivita [0][0]
22		Action			CHOOSE RANDOM MOVE	Bolt to All AOG
23						
24	MAIN NODE	Verifiable Group	OR	True		
25		Condition		True	CHECK COUNTER MOVE	Check Last Move: Mother's Scuders [2 / 2]
26	MAIN NODE	Verifiable Group	OR	True		
27		Condition		True	ENEMY PERSONALITY CHECK TRAIT	Trait: Intelligenza [74][74]
28		Condition		None	ENEMY PERSONALITY CHECK TRAIT	Trait: Aggressivita [0][0]
29		Action			CHOOSE RANDOM MOVE	Bolt to All AOG

Figura 5.1: Dati analitici del processo decisionale della mossa del Boss

Come evidenziato nell'illustrazione soprastante, i riquadri colorati in blu rappresentano un singolo processo decisionale associato alla selezione delle mosse del boss durante il combattimento. È possibile notare che il boss dimostra una preferenza marcata per le mosse caratterizzate da tratti legati all'intelligenza rispetto agli altri attributi della sua personalità.
Inoltre, è interessante notare che le decisioni prese nei processi decisionali successivi mostrano una variazione significativa, introducendo un elemento di pseudo-casualità all'interno della personalità del boss. Questa variabilità nelle scelte è altamente influenzata anche dal design specifico delle mosse disponibili, il quale gioca un ruolo fondamentale nella definizione della natura delle decisioni adottate dal boss.

Nella fase iniziale del processo decisionale, il boss prende la decisione di utilizzare la mossa denominata "Mother's Scuders". Questa mossa si distingue per il suo design, il quale include un meccanismo che permette di sbloccare un'ulteriore mossa per il personaggio. Sfruttando questa nuova opportunità, il processo decisionale incorpora sia gli aspetti legati alla personalità distintiva del boss che quelli relativi al design specifico della mossa per determinare la selezione successiva. In questo esempio particolare, si nota la scelta di

una mossa con connotazioni marcatamente aggressive, identificabile dal nome: "Bolt to All AOG".

Proseguendo nella trattazione, sarà condotta un'analisi approfondita della state machine, componente di primaria rilevanza nell'ambito della definizione delle personalità dei personaggi. In questa fase, verrà esaminato in dettaglio il funzionamento dinamico e attivo di tale algoritmo, il quale modifica in tempo reale i valori associati ai tratti di personalità, esercitando così un'influenza diretta sulla direzione intrapresa nel processo decisionale. Inoltre, verranno esaminate le componenti personalizzate, rilevanti per la modellazione accurata del processo decisionale. Si potrà osservare come queste componenti, nel contesto del gioco, svolgano un ruolo di fondamentale importanza nell'ottimizzazione e nell'arricchimento dell'esperienza di gioco, contribuendo a definire le scelte e le azioni dei personaggi in modo sempre più dettagliato e coinvolgente.

56	MAIN NODE	Verifiable Group	OR	False		
57		Condition		False	CHECK COUNTER MOVE	Check Last Move: Mother's Scuders [0 / 2]
58	MAIN NODE	Verifiable Group	AND	True		
59		Condition		True	CHECK HEALTH LESS THAT PERCENT	
60		Condition		True	ENEMY PERSONALITY CHECK TRAIT	Trait: Intelligenza [64][64]
61		Condition		True	LAST MOVE NOT IN LIST	
62		Action			CHOOSE SIMPLE MOVE	Freshen Up
63						
64	MAIN NODE	Verifiable Group	OR	False		
65		Condition		False	CHECK COUNTER MOVE	Check Last Move: Mother's Scuders [0 / 2]
66	MAIN NODE	Verifiable Group	AND	False		
67		Condition		True	CHECK HEALTH LESS THAT PERCENT	
68		Condition		True	ENEMY PERSONALITY CHECK TRAIT	Trait: Intelligenza [10][10]
69		Condition		False	LAST MOVE NOT IN LIST	
70	MAIN NODE	Verifiable Group	AND	False		
71		Condition		False	CHECK HEALTH MORE THAT PERCENT	
72		Condition		True	ENEMY PERSONALITY CHECK TRAIT	Trait: Potenziamento [37][37]
73	MAIN NODE	Verifiable Group	AND	True		
74		Condition		True	CHECK HEALTH LESS THAT PERCENT	
75	MAIN NODE	Verifiable Group	OR	True		
76		Condition		True	ENEMY PERSONALITY CHECK TRAIT	Trait: Curatore [70][70]
77		Condition		None	ENEMY PERSONALITY CHECK TRAIT	Trait: Intelligenza [0][0]
78		Action			CHOOSE RANDOM MOVE	Mother's Scuders
79						
80	MAIN NODE	Verifiable Group	OR	True		
81		Condition		True	CHECK COUNTER MOVE	Check Last Move: Mother's Scuders [1 / 2]
82	MAIN NODE	Verifiable Group	OR	True		
83		Condition		False	ENEMY PERSONALITY CHECK TRAIT	Trait: Intelligenza [16][16]
84		Condition		True	ENEMY PERSONALITY CHECK TRAIT	Trait: Aggressivita [91][91]
85		Action			CHOOSE RANDOM MOVE	Bolt to All AOG

Figura 5.2: Dati analitici del processo decisionale della mossa del Boss

Nell'immagine presentata, vengono esaminati i processi decisionali adottati dal boss quando il suo livello di salute si attesta al 50%. Dall'analisi dei dati esposti emerge un impatto determinante sulla scelta effettuata nel primo processo decisionale: viene selezionata la mossa denominata "Freshen Up". Questa mossa, caratterizzata dal suo specifico design, conferisce al personaggio la capacità di beneficiare di un meccanismo di guarigione graduale noto come "HOT" (Healing Over Time). La decisione del boss di adottare questa strategia rispecchia chiaramente l'intenzione di preservare la propria vitalità

dopo aver subito un considerevole decremento nella salute. L'influenza di questa scelta si riflette in modo significativo sull'esperienza di gioco offerta dal boss, introducendo una dimensione di sfida e coinvolgimento più profonda e gratificante per l'utente.

Inoltre, è possibile osservare le componenti personalizzate di condizione, tra cui "CheckHealthMoreThanPercent". Queste specifiche condizioni, insieme alle loro controparti analoghe, vengono controllate durante il corso dei combattimenti al fine di monitorare il livello di salute del personaggio. La loro funzione principale consiste nel selezionare un insieme specifico di mosse tra cui il personaggio può effettivamente scegliere, contribuendo in modo significativo ad ampliare ulteriormente e dettagliare il design complessivo del processo decisionale.

142	MAIN NODE	Verifiable Group	OR	True		
143		Condition		True	CHECK COUNTER MOVE	Check Last Move: Mother's Scuders [1 / 2]
144	MAIN NODE	Verifiable Group	OR	True		
145		Condition		True	ENEMY PERSONALITY CHECK TRAIT	Trait: Intelligenza [10][10]
146		Condition		True	ENEMY PERSONALITY CHECK TRAIT	Trait: Aggressivita [71][71]
147		Action			CHOOSE RANDOM MOVE	Bolt to All AOG
148						
149	MAIN NODE	Verifiable Group	OR	True		
150		Condition		True	CHECK COUNTER MOVE	Check Last Move: Mother's Scuders [2 / 2]
151	MAIN NODE	Verifiable Group	OR	True		
152		Condition		False	ENEMY PERSONALITY CHECK TRAIT	Trait: Intelligenza [10][10]
153		Condition		True	ENEMY PERSONALITY CHECK TRAIT	Trait: Aggressivita [84][84]
154		Action			CHOOSE RANDOM MOVE	SharkBite

Figura 5.3: Dati analitici del processo decisionale della mossa del Boss

Nell'ultima figura presentata, si evidenzia un significativo cambiamento nei valori dei tratti di personalità del boss durante il proseguimento del combattimento. È osservabile un notevole decremento del tratto relativo all'intelligenza rispetto alla fase iniziale, accompagnato da un costante aumento del tratto dell'aggressività ad ogni successivo passo all'interno del processo decisionale. Tali variazioni sono strettamente riconducibili al design specifico della state machine associata al boss. Questo algoritmo è stato attentamente progettato per accentuare gli attributi aggressivi della personalità del boss, mentre contemporaneamente riduce quelli associati all'intelligenza. Questi cambiamenti emergono in modo più evidente nelle fasi avanzate del combattimento, ovvero quando il boss si avvicina alla sua sconfitta inevitabile e adotta una strategia basata principalmente sull'aggressività.

Nel contesto dell'esempio illustrato, emerge la selezione della mossa denominata "SharkBite". Questa mossa, oltre a infliggere notevoli danni al bersaglio designato, attiva anche un effetto di danno continuo noto come "Damage over Time" (DoT). È da notare che questa mossa rientra tra le opzioni più aggressive a disposizione del boss. La scelta di utilizzare questa mossa contribuisce a confermare in modo inequivocabile il quadro precedentemente delineato, evi-

denziando l'orientamento aggressivo adottato dal boss nel cercare di arrecare danni significativi e protratti al suo avversario.

5.2 Scelta del Target

In questa sezione, sarà condotta un'analisi dettagliata del processo decisionale adottato dal boss del gioco per selezionare il bersaglio in cui utilizzare la mossa scelta. Attraverso questa indagine, sarà possibile osservare l'evoluzione dinamica della personalità del personaggio e la sua interazione con il contesto di combattimento. Il caso di studio preso in considerazione coinvolgerà un combattimento in cui il boss emerge vittorioso.

	A	B	C	D	E	F
1		NODE NAME	LOGIC PORT	RESULT	DESCRIPTION	DESCRIPTION RESULT
2	MAIN NODE	Verifiable Group	AND	False		
3		Condition		False	SELECTED MOVE SIMPLE CHECK TYPE TARGET	
4		Action			CHOOSE DEFAULT TARGET	AUTOMATIC
5						
6	MAIN NODE	Verifiable Group	AND	True		
7		Condition		True	SELECTED MOVE SIMPLE CHECK TYPE TARGET	
8	MAIN NODE	Verifiable Group	AND	True		
9		Condition		True	ENEMY PERSONALITY CHECK TRAIT	Trait: Aggressivita [34]
10	MAIN NODE	Verifiable Group	AND	False		
11		Condition		False	ENEMY PERSONALITY CHECK TRAIT	Trait: Intelligenza [85]
12		Action			SINGLE RANDOM ALLY	Rhaadi
13						
14	MAIN NODE	Verifiable Group	AND	False		
15		Condition		False	SELECTED MOVE SIMPLE CHECK TYPE TARGET	
16		Action			CHOOSE DEFAULT TARGET	AUTOMATIC
17						
18	MAIN NODE	Verifiable Group	AND	False		
19		Condition		False	SELECTED MOVE SIMPLE CHECK TYPE TARGET	
20		Action			CHOOSE DEFAULT TARGET	AUTOMATIC
21						
22	MAIN NODE	Verifiable Group	AND	True		
23		Condition		True	SELECTED MOVE SIMPLE CHECK TYPE TARGET	
24	MAIN NODE	Verifiable Group	AND	True		
25		Condition		True	ENEMY PERSONALITY CHECK TRAIT	Trait: Aggressivita [56]
26	MAIN NODE	Verifiable Group	AND	True		
27		Condition		True	ENEMY PERSONALITY CHECK TRAIT	Trait: Intelligenza [86]
28		Action			SINGLE ALLY WITH MORE HEALTH	Deezel

Figura 5.4: Dati analitici del processo decisionale della scelta del Target del Boss

In questa figura, viene rappresentato l'inizio del combattimento e le prime decisioni adottate dal boss. È possibile notare, in grassetto e in colore nero, l'indicazione del bersaglio selezionato. La dicitura "AUTOMATIC" è utilizzata per indicare che il bersaglio è determinato in modo automatico attraverso il design della mossa stessa. A titolo di esempio, questa situazione potrebbe verificarsi quando una mossa è progettata per colpire tutti i personaggi presenti senza richiedere una selezione specifica.

In aggiunta, si può notare un progressivo incremento dei tratti di personalità correlati all'aggressività del boss, il quale seleziona come bersaglio il

personaggio con la massima quantità di salute. Questa decisione è in parte modulata dall'influenza esercitata dal tratto di intelligenza del personaggio stesso. Il design concepito per questo processo decisionale conduce a una rimozione sistematica dei nemici del boss, seguendo una strategia che riflette appieno i tratti distintivi della sua personalità.

56	MAIN NODE	Verifiable Group	AND	True		
57		Condition		True	SELECTED MOVE SIMPLE CHECK TYPE TARGET	
58	MAIN NODE	Verifiable Group	AND	True		
59		Condition		True	ENEMY PERSONALITY CHECK TRAIT	Trait: Aggressivita [73]
60	MAIN NODE	Verifiable Group	AND	False		
61		Condition		False	ENEMY PERSONALITY CHECK TRAIT	Trait: Intelligenza [75]
62		Action			SINGLE RANDOM ALLY	**Rhaadi**
63						
64	MAIN NODE	Verifiable Group	AND	True		
65		Condition		True	SELECTED MOVE SIMPLE CHECK TYPE TARGET	
66	MAIN NODE	Verifiable Group	AND	True		
67		Condition		True	ENEMY PERSONALITY CHECK TRAIT	Trait: Aggressivita [51]
68	MAIN NODE	Verifiable Group	AND	False		
69		Condition		False	ENEMY PERSONALITY CHECK TRAIT	Trait: Intelligenza [58]
70		Action			SINGLE RANDOM ALLY	**Rhaadi**
71						
72	MAIN NODE	Verifiable Group	AND	True		
73		Condition		True	SELECTED MOVE SIMPLE CHECK TYPE TARGET	
74	MAIN NODE	Verifiable Group	AND	True		
75		Condition		True	ENEMY PERSONALITY CHECK TRAIT	Trait: Aggressivita [47]
76	MAIN NODE	Verifiable Group	AND	True		
77		Condition		True	ENEMY PERSONALITY CHECK TRAIT	Trait: Intelligenza [63]
78		Action			SINGLE ALLY WITH MORE HEALTH	**Deezel**
79						
80	MAIN NODE	Verifiable Group	AND	True		
81		Condition		True	SELECTED MOVE SIMPLE CHECK TYPE TARGET	
82	MAIN NODE	Verifiable Group	AND	True		
83		Condition		True	ENEMY PERSONALITY CHECK TRAIT	Trait: Aggressivita [68]
84	MAIN NODE	Verifiable Group	AND	True		
85		Condition		True	ENEMY PERSONALITY CHECK TRAIT	Trait: Intelligenza [62]
86		Action			SINGLE ALLY WITH MORE HEALTH	**Deezel**

Figura 5.5: Dati analitici del processo decisionale della scelta del Target del Boss

Come si può osservare dall'illustrazione, nella fase conclusiva del combattimento, si registrano oscillazioni significative nei tratti di personalità del boss, accompagnate dalla successiva eliminazione dei personaggi "Suggo," "Rhaadi," e "Deezel." Queste fluttuazioni nei valori dei tratti di personalità del boss sono il risultato diretto dell'operato della state machine, che regola in modo dinamico tali valori in base al livello di salute del personaggio. Questo algoritmo genera variazioni dinamiche durante il processo di selezione del bersaglio, conferendo così una notevole dinamicità all'esperienza di gioco. Tale dinamicità non è influenzata solamente dalle meccaniche di gioco, ma è anche fortemente plasmata dalla personalità in continua evoluzione del personaggio, la quale subisce trasformazioni attive lungo l'intero arco del combattimento.

In conclusione, in seguito all'approfondita analisi dei due processi decisionali che dirigono il comportamento del boss all'interno del gioco, emerge un'esperienza di gioco straordinariamente reattiva alle sfumature delle dina-

miche di combattimento. Questa elevata reattività è resa possibile attraverso l'implementazione accurata del sistema di personalità e l'impiego efficace della state machine. Entrambi questi elementi conferiscono un'unicità distintiva ad ogni personaggio all'interno del contesto ludico. L'editor creato si dimostra un potente strumento, permettendo la modellazione di ogni dettaglio in modo unico e altamente flessibile. Questo approccio complessivo offre al giocatore una sfida in continua evoluzione, mettendo costantemente alla prova le sue capacità e coinvolgendolo in modo dinamico in ciascun combattimento.

Conclusioni

Nel presente capitolo conclusivo, si procederà all'approfondimento delle tematiche relative al funzionamento dell'ambiente di sviluppo creato, con particolare attenzione all'analisi dei vantaggi intrinseci offerti dall'editor, nonché alla valutazione delle possibili estensioni e all'identificazione dei punti critici che possono emergere in relazione all'architettura proposta per l'editor.

In conclusione, verrà esaminata la fattibilità di ampliare l'algoritmo degli alberi decisionali attraverso l'implementazione di un algoritmo precedentemente trattato nel secondo capitolo, il Decision Tree Learning. Tale analisi si concentrerà sulle prospettive e la potenziale efficacia che quest'ultimo algoritmo può apportare al contesto in esame.

5.3 Editor

All'interno di questa sezione, sarà condotta un'analisi dettagliata dei molteplici percorsi di sviluppo potenziali per l'editor, partendo dalla valutazione dei suoi vantaggi, procedendo poi con l'esame della sua scalabilità e delle possibili estensioni al fine di arricchirne e ampliarne il campo di applicazione. In conclusione, si procederà con un'analisi approfondita dei punti critici che possono emergere a causa dell'architettura proposta per l'editor.

5.3.1 Scalabilità e Vantaggi

All'interno degli strumenti operativi forniti dall'editor, è opportuno sottolineare la presenza dei nodi di condizione e di azione come componenti di rilievo. Questi nodi si distinguono per una struttura intrinsecamente scalabile e altamente pragmatica, il che implica che possono essere adattati in modo flessibile alle esigenze specifiche dell'utente. Una caratteristica di notevole rilevanza è la possibilità di associare script personalizzati a tali nodi, consentendo così una notevole versatilità nell'implementazione di funzionalità specializzate. Inoltre, va sottolineata la loro elevata interscambiabilità, che facilita l'integrazione e la cooperazione sinergica con altri componenti dell'editor. Questi attributi si inseriscono in un sistema di creazione avanzato, che amplifica ulteriormente le capacità e le potenzialità dell'editor nell'ambito dell'ambiente di sviluppo.

Nodo di Azione

Per quanto concerne l'analisi dei nodi di azione, è fondamentale sottolineare la complessità dell'architettura implementata, la quale si caratterizza per la presenza di una distinta suddivisione in due responsabilità primarie, isolate tra di loro. Queste due responsabilità sono fondamentali poiché contribuiscono sinergicamente alla realizzazione delle funzioni assegnate ai singoli nodi. Innanzitutto, la prima responsabilità riguarda il comportamento specifico che ciascun nodo di azione adotta all'interno del processo decisionale. Questo aspetto è di vitale importanza, in quanto definisce in modo dettagliato e preciso le azioni e le reazioni del nodo in risposta agli stimoli ambientali e ai dati di input. Tale comportamento rappresenta il cuore del funzionamento del nodo, guidandone l'azione e determinandone il contributo al processo decisionale complessivo. La seconda responsabilità cruciale si concentra sul funzionamento interno del nodo all'interno della struttura più ampia dell'albero decisionale. Questo aspetto è essenziale per garantire che il nodo possa interagire in modo efficiente e coerente all'interno del contesto più ampio dell'albero decisionale. Ciò implica che il nodo debba essere in grado di comunicare in modo efficace

con gli altri nodi e di contribuire in modo sinergico alla presa di decisioni e all'esecuzione delle azioni previste nell'ambito del processo decisionale globale.

Questi due aspetti fondamentali del nodo di azione trovano manifestazione attraverso due elementi distinti: il primo è rappresentato dallo script associato al nodo, il quale assume la responsabilità di definire e orchestrare il comportamento specifico del nodo. Il secondo elemento, a sua volta, incapsula questo comportamento in modo che possa interagire sinergicamente con l'intera struttura dell'albero decisionale. La sinergia tra questi due elementi apre le porte a prospettive virtualmente illimitate per la creazione di nodi d'azione altamente personalizzati, creando un terreno fertile per un'ampia gamma di scenari e funzionalità all'interno dell'ambiente di sviluppo in questione.

Nodo di Condizione

All'interno dell'analisi dettagliata del nodo di condizione, si evidenziano due aspetti fondamentali che si sviluppano parallelamente, in una maniera simile a quanto già osservato per il nodo di azione. In primo luogo, c'è da considerare il complesso processo che il nodo di condizione deve intraprendere al fine di giungere a una valutazione booleana, la quale rappresenta il fulcro della sua funzionalità nell'interazione con l'ambiente di sviluppo. Questo processo implica una serie di operazioni e valutazioni che, compiute dal nodo di condizione, determinano il valore booleano risultante, che a sua volta incide direttamente sulle decisioni e le azioni intraprese nell'editor. In tal senso, il nodo di condizione svolge un ruolo critico nell'analisi e nella risposta alle condizioni ambientali e ai dati di input, contribuendo in modo determinante alla dinamica complessiva dell'editor. In secondo luogo, altrettanto significativa è la responsabilità delle funzionalità che il nodo di condizione deve assolvere in relazione agli altri elementi dell'editor. Questa responsabilità comprende la gestione delle interazioni e delle connessioni con gli altri componenti dell'editor, garantendo un processo efficace nell'ambito dell'albero decisionale.

Questi due aspetti essenziali trovano una manifestazione tangibile all'interno del nodo stesso, attraverso la presenza di due elementi concomitanti. Il primo di questi elementi è costituito dallo script associato al nodo, il quale assume la responsabilità di definire e guidare il processo necessario per ottenere il valore booleano richiesto, fungendo da motore trainante del nodo. Il secondo elemento di rilievo è il nodo in sé, il quale si erge come l'entità operativa responsabile per le interazioni esterne con gli altri elementi dell'editor. La struttura intrinseca del nodo di condizione presenta un potenziale di creazione virtualmente infinito, consentendo così una scalabilità senza limiti nella generazione di nuovi componenti all'interno dell'editor.

Vantaggi

La natura intrinseca degli strumenti offerti dall'ambiente di sviluppo rivela una serie di vantaggi di notevole rilevanza. Questi vantaggi sono orientati verso l'obiettivo di integrare alberi decisionali in un'ampia gamma di contesti, destinati a qualsiasi sistema che richieda un sofisticato processo decisionale. Questo approccio conferisce all'editor un ruolo di portata globale e universale nel contesto dello sviluppo, consentendo la creazione di processi decisionali complessi e adattabili all'interno di una vasta gamma di applicazioni, compresi sistemi di gioco e altre soluzioni software.

Un esempio eloquente dell'ampia portata di utilizzo dell'editor emerge nella sua capacità di agevolare la creazione di alberi decisionali per il processo di selezione dei dialoghi durante il combattimento. Questo processo è orchestrato in modo tale da riflettere comportamenti specifici registrati durante gli scontri, con l'obiettivo di attivare dialoghi appropriati per il contesto di gioco. Questa integrazione di alberi decisionali nel contesto dei dialoghi di combattimento sottolinea in modo inequivocabile il potenziale vantaggio offerto dall'ambiente di sviluppo.

5.3.2 Punti Critici

L'intrinseca complessità dei nodi di condizione e azione personalizzati può dare luogo alla possibile occorrenza di errori nel corso del processo decisionale. Questa complessità sottolinea la necessità di un'analisi attenta e approfondita per garantire che tali nodi siano progettati e implementati in modo coerente e affidabile, al fine di evitare potenziali inesattezze o comportamenti indesiderati nel processo decisionale complessivo.

Per quanto attiene ai nodi di condizione, si evidenzia un'importante questione relativa alla possibilità di errori significativi nel processo di acquisizione del valore booleano necessario. Questi errori possono sorgere in particolare a causa della presenza di loop di programmazione, situazioni in cui il nodo di condizione potrebbe rimanere intrappolato in un ciclo senza fine, senza riuscire a raggiungere una decisione definitiva. Questo scenario, se non gestito in modo adeguato, potrebbe causare il blocco dell'intero albero decisionale poiché gli altri componenti dell'albero sono costretti ad attendere indefinitamente il completamento del processo del nodo di condizione problematico. L'effetto risultante è un rallentamento significativo o addirittura l'arresto completo dell'intero flusso decisionale, un risultato che va chiaramente contro l'obiettivo di mantenere una reattività tempestiva nelle decisioni, elemento di fondamentale importanza in molti contesti operativi.

In modo analogo, i nodi di azione possono essere soggetti all'emergere di loop di programmazione, situazioni in cui, durante l'esecuzione della loro azione, non viene raggiunto un punto di terminazione definitivo. Di conseguenza, ciò impedisce il progresso nell'esecuzione del nodo successivo nell'albero decisionale, generando una situazione di stallo nell'evoluzione del flusso decisionale complessivo. In prospettiva di sviluppi futuri, è possibile considerare l'implementazione di componenti specializzate in grado di rilevare e risolvere potenziali loop di programmazione all'interno del processo decisionale. Questa futura evoluzione consentirebbe di garantire una conclusione definitiva per il processo, garantendo che non si verifichino situazioni di stallo prolungate.

5.3.3 Estensioni

L'architettura dell'editor si contraddistingue per la sua notevole versatilità, che va oltre la semplice agevolazione nella creazione di nuove famiglie di componenti. Essa consente anche la configurazione di nuovi alberi decisionali, aggiungendo un ulteriore livello di adattabilità e personalizzazione. Questa intrinseca flessibilità apre ampie opportunità per l'innovazione nell'ambito della progettazione e dello sviluppo di strumenti avanzati dedicati alla concezione e all'evoluzione degli alberi decisionali. In sostanza, l'architettura dell'editor non solo semplifica il processo di sviluppo attuale, ma offre anche una base dinamica per l'espansione continua e l'arricchimento dell'ambiente di sviluppo, garantendo che sia sempre all'avanguardia dalle esigenze e dalle sfide emergenti.

Nodo Radice

Nell'ambito delle estensioni potenziali, emerge l'interessante possibilità di concepire e implementare alberi decisionali che presentino più nodi radice. Questa prospettiva apre le porte a una serie di scenari operativi complessi e ricchi di sfumature. Innanzitutto, tale sviluppo consentirebbe l'esecuzione simultanea di diversi processi decisionali all'interno dello stesso ambiente di sviluppo, fornendo un livello di flessibilità senza precedenti. Inoltre, questa nuova architettura offrirebbe la capacità di sfruttare in modo condiviso elementi comuni all'interno dell'ambiente di sviluppo, promuovendo l'efficienza e la condivisione di risorse e componenti. Questo approccio favorirebbe una maggiore coerenza e coesione tra i vari processi decisionali in corso.

Da un punto di vista più dettagliato, l'introduzione di nodi radice multipli consentirebbe un grado di precisione senza precedenti nell'esecuzione dei processi decisionali, poiché ciascun nodo radice potrebbe essere configurato in modo specifico per rispondere a determinati contesti o scenari. Ciò amplia notevolmente le possibilità di adattare l'editor alle esigenze specifiche del

progetto, promuovendo al contempo una gestione più efficace dei processi decisionali complessivi. In sintesi, l'introduzione di alberi decisionali con più nodi radice rappresenta un passo significativo verso una maggiore flessibilità, condivisione e precisione nell'ambito dell'ambiente di sviluppo.

Nell'ambito dell'espansione delle funzionalità dell'editor, che ora include la possibilità di avere più nodi radice, emergono sfide significative nella gestione dei nodi all'interno dell'editor. Queste sfide diventano evidenti quando si utilizzano nodi comuni in diversi percorsi degli alberi decisionali, poiché ciò richiede la salvaguardia dell'integrità dei dati contenuti in tali nodi. L'attraversamento ripetuto dei nodi condivisi può portare alla sovrascrittura dei dati e, di conseguenza, alla perdita di coerenza delle informazioni. Una delle soluzioni possibili per affrontare questa problematica potrebbe consistere nella clonazione multipla dei nodi condivisi durante il processo di clonazione dell'albero. In alternativa, potrebbe essere considerata la creazione di un sistema che impedisca l'utilizzo simultaneo dei dati nei nodi condivisi. Tuttavia, l'implementazione di quest'ultimo approccio potrebbe richiedere la creazione di ulteriori nodi all'interno dell'editor per gestire comportamenti simili in modo separato, al fine di garantire l'integrità e la consistenza dei dati all'interno dell'editor.

Multi Connessioni per il nodo di verifica

Una delle possibili estensioni meritevoli di approfondimento riguarda la creazione di nodi capaci di generare una molteplicità di diramazioni all'interno della struttura degli alberi decisionali. Questo concetto supera la semplice logica di avere un singolo valore booleano per determinare un percorso, introducendo invece la capacità di definire una serie di condizioni distinte, ciascuna in grado di innescare una ramificazione diversa nel processo decisionale.

In questa prospettiva, si verifica un incremento esponenziale delle possibilità di configurare il flusso delle decisioni all'interno di un albero decisionale. Questo significa che, a seconda delle combinazioni specifiche di condizioni soddisfatte o non soddisfatte, possono emergere una vasta gamma di percorsi decisionali alternativi. Questo approccio offre un elevato grado di flessibilità e adattabilità nell'ambito della definizione dei processi decisionali, consentendo di modellare in modo più preciso e dettagliato il comportamento dell'editor in risposta a condizioni e contesti complessi e mutevoli. In sintesi, l'implementazione di nodi con questa capacità di creare più percorsi rappresenta una chiara espansione delle possibilità di progettazione e personalizzazione nell'ambito degli alberi decisionali.

L'integrazione di un nuovo nodo nell'editor porta con sé una serie di sfide di progettazione legate alla struttura dell'albero decisionale che coinvolge

questo nodo aggiunto. Tra queste sfide, emerge un scenario critico in cui si possono creare processi decisionali che giungono a una conclusione senza che venga eseguita alcuna azione. Questo scenario, sebbene rappresenti una sfida, può essere relativamente gestibile, in quanto il flusso decisionale si conclude senza ulteriori complessità quando nessuna condizione associata al nuovo nodo viene soddisfatta. Tuttavia, la complessità aumenta notevolmente quando più percorsi del nuovo nodo vengono attivati simultaneamente. In questa situazione, emerge una significativa mancanza di controllo sulla direzione e sugli esiti del processo decisionale, generando un comportamento imprevedibile. Questo comporta notevoli difficoltà nella gestione e nel controllo del design dell'albero decisionale nel suo complesso.

Per affrontare questa complessa problematica, una possibile soluzione consiste nell'implementare un sistema in cui viene attivata solo la prima condizione che risulta soddisfatta tra i percorsi possibili del nodo. Questo approccio mira a migliorare il controllo sul design dell'albero decisionale. Tuttavia, l'implementazione di un tale algoritmo introduce ulteriori sfide, poiché richiede una comprensione dettagliata e una gestione attenta delle condizioni che determinano l'attivazione dei percorsi, rendendo necessaria un'analisi approfondita per garantire il corretto funzionamento del processo decisionale.

5.4 Estensione al Learning

Nel secondo capitolo della tesi, viene presentato uno degli algoritmi fondamentali utilizzati nell'ambito degli alberi decisionali, noto come Decision Tree Learning. Questo algoritmo è caratterizzato dall'abilità di generare dinamicamente la struttura di un albero decisionale attraverso l'apprendimento da dati nominali, all'interno di un contesto specifico di sviluppo. L'approccio intrinseco a tale algoritmo si basa sulla manipolazione di un insieme di dati nominali, al fine di creare progressivamente tutti i nodi dell'albero decisionale. Il processo di costruzione dell'albero segue un approccio 'greedy', il che significa che l'algoritmo prende decisioni istantanee per massimizzare il guadagno d'informazione durante il processo di creazione dei nodi dell'albero.

In pratica, l'algoritmo Decision Tree Learning inizia con un set di dati contenente variabili nominali. Esso utilizza un'analisi approfondita dei dati per individuare iterativamente le variabili più rilevanti e i punti di divisione ottimali. Man mano che prosegue, il processo prevede la creazione di nodi dell'albero che rappresentano decisioni basate su queste variabili e la successiva suddivisione del set di dati in sottoinsiemi più piccoli. Questo iter continua fino a quando vengono soddisfatte specifiche condizioni di arresto, come ad esempio il raggiungimento della purezza dei nodi o la profondità massima dell'albero

decisionale. A questo punto, si giunge alle foglie dell'albero decisionale. In conclusione, l'algoritmo Decision Tree Learning si configura come una potente tecnica di apprendimento automatico, in grado di creare dinamicamente la struttura di un albero decisionale mediante l'analisi di dati nominali. Tale processo permette di prendere decisioni basate sui dati in modo efficace e contestualizzato all'interno del processo di sviluppo in esame.

L'implementazione di tale algoritmo implica la creazione di un ambiente ottimizzato per l'acquisizione di data-set rilevanti, al fine di guidare in modo accurato la costruzione dell'albero decisionale. Nel contesto di studio della tesi, del gioco "FarAfter", ciò richiede l'esecuzione di diversi studi mirati al fine di ottenere dati nominali appropriati. Questi dati possono includere, ad esempio, informazioni sulle mosse, gli stati, i dati dei "DOT" (Damage Over Time) e "HOT" (Healing Over Time). Un passo cruciale in questo processo è l'analisi dettagliata di ogni aspetto rilevante che potrebbe influire sul processo di creazione dell'albero decisionale. Questa analisi si propone di identificare e valutare attentamente ogni dettaglio pertinente che potrebbe avere un impatto sull'efficacia e l'accuratezza dell'albero decisionale durante la fase di sviluppo.

Questa prospettiva di sviluppo potrebbe tradursi nell'espansione dell'editor, dando luogo alla creazione di un ambiente specificamente progettato per l'analisi e la manipolazione dei dati nominali associati a ciascun componente del gioco. Tale approccio comporterebbe inevitabilmente la generazione di un considerevole volume di dati per ogni nuovo elemento introdotto nel contesto del gioco. Questa acquisizione di dati mirata risulterebbe essenziale in quanto essi influenzano attivamente il comportamento di gioco. La gestione di un vasto e crescente corpus di dati presenterebbe, nell'implementazione dell'algoritmo Decision Tree Learning, una sfida di notevole complessità. Tuttavia, potrebbe anche stimolare ulteriori sforzi mirati verso l'ottimizzazione e l'automazione dei processi connessi alla creazione degli alberi decisionali.

Ringraziamenti

In primo luogo, vorrei ringraziare il mio relatore, Alessandro Ricci, per i suoi preziosi consigli e la sua costante disponibilità. Grazie per avermi fornito spunti importanti nella stesura di questo lavoro e per avermi indirizzato nei momenti di indecisione.

Non posso fare a meno di esprimere la mia profonda gratitudine alle due persone che hanno avuto un immenso impatto sulla mia crescita ed educazione: i miei genitori. Mamma e papà, il vostro costante sostegno, la vostra guida e il vostro amore hanno reso possibile superare le sfide più ardue in questo percorso. Senza di voi, non avrei mai potuto raggiungere questo fondamentale traguardo nella mia vita accademica. I vostri insegnamenti, la vostra dedizione e la la vostra filosofia sono stati i pilastri su cui ho costruito il mio percorso di vita. Vi sono eternamente grato per avermi accompagnato in questo viaggio e per aver contribuito in modo così significativo al mio sviluppo personale e professionale. Grazie, di cuore, per tutto ciò che avete fatto.

In particolare, desidero manifestare la mia profonda gratitudine a Luana Mennuti, la mia fidanzata, il tuo carattere solare e dinamico, la tua forza interiore e il tuo coraggio sono stati una fonte di ispirazione e crescita personale per me. Ti ringrazio di cuore per il tempo prezioso che hai dedicato a sostenermi e per essere stata sempre al mio fianco, senza esitazioni. Le tue parole di incoraggiamento, i momenti condivisi di sostegno e gioia, insieme alla tua presenza costante, hanno reso questo percorso non solo straordinario ma anche profondamente significativo.
Grazie, di tutto cuore, per essere sempre al mio fianco.

Desidero esprimere la mia gratitudine agli amici che hanno condiviso con me progetti impegnativi e gioie durante il mio percorso accademico. In particolare, vorrei ringraziare Mattia Matteini e Alberto Paganelli, che hanno affrontato con me esami e fatiche, contribuendo a raggiungere insieme il traguardo. Nonostante il ritardo, sono felice di annunciare che anch'io ho portato a termine questa sfida!

Desidero rivolgere un sentito ringraziamento a tutti gli amici straordinari che ho avuto il privilegio di conoscere lungo il mio percorso accademico. La loro preziosa presenza ha illuminato il mio cammino con gioia, condivisione di momenti felici e instancabile impegno nello studio, contribuendo in modo significativo alla mia crescita e al raggiungimento di questo traguardo.

Bibliografia

[1] Ralph Johnson John Vlissider Erich Gamma, Richard Helm. *Design Patterns*. Pearson, Milano, 2022.

[2] IBM. Alberi decisionali.

[3] Robert C. Martin. *Clean Architecture*. APOGEO, Milano, 2020.

[4] MathWorks. State machine.

[5] Medium. Decision trees: Id3 algorithm explained.

[6] Ian Millington. *AI for Games*. Taylor Francis Group, LLC, 2019.

[7] Sunil Ray. Naive bayes classifier explained: Applications and practice problems of naive bayes classifier.

[8] Sebastian Taylor. Nominal data.

[9] Unity. Forum.

[10] Unity. Unity documentation.

[11] Wikipedia. Albero di decisione.

[12] Wikipedia. Finite-state machine.

[13] Wikipedia. Naive bayes classifier.