



UNIVERSITÀ DEGLI STUDI DI FIRENZE
SCUOLA DI INGEGNERIA - DIPARTIMENTO DI INGEGNERIA
DELL'INFORMAZIONE

Tesi di Laurea Triennale in Ingegneria Informatica

**EXPLAINABLE ARTIFICIAL INTELLIGENCE PER
LA VALUTAZIONE DI AGENTI ADDESTRATI
TRAMITE REINFORCEMENT LEARNING IN GIOCHI
STRATEGICI TURN-BASED**

Candidato

Manuel Cecere Palazzo

Relatori

Prof. Andrew D. Bagdanov

Dott. Alessandro Sestini

Anno Accademico 2019/2020

Indice

Elenco delle figure	iii
Elenco delle tabelle	iii
Introduzione	i
1 Lavori Correlati	1
1.1 Machine Learning ed Explainable AI	1
1.1.1 Reinforcement Learning	3
1.1.2 Explainable AI	5
1.2 DeepCrawl	7
1.2.1 Gameplay	8
1.2.2 DRL in DeepCrawl	11
1.2.3 Unity ed Entity-Component-System	13
2 Sistema di Explainable AI	15
2.1 Progettazione	16
2.2 Game Over	16
2.3 Navigare l'episodio	18
2.3.1 Salvataggio e Caricamento	18
2.4 Informazioni sugli attori	21

2.4.1	Distribuzione di probabilità delle azioni	22
2.4.2	Entropia	24
2.4.3	Errore dell'agente	26
2.4.4	Reward	27
2.4.5	Azione del Player	29
3	Test di Usabilità	34
3.1	Fasi del test	35
3.2	Domande	36
3.3	Esito del test	38
3.3.1	Efficacia complessiva	38
3.3.2	Valutazione interfaccia	38
3.3.3	Look-and-feel	39
4	Conclusioni	41
4.1	Lavori Futuri	42
	Bibliografia	45

Elenco delle figure

1	Crescente popolarità della parola <i>machine learning</i>	i
1.1	Rappresentazione grafica di una <i>Deep Neural Network</i>	3
1.2	Schema Agent-Enviroment del RL	4
1.3	Interfaccia grafica di XAI realizzata per il gioco Amidar.	7
1.4	Esempio di turno in DeepCrawl	11
1.5	Esempio di ECS applicato in DeepCrawl	14
2.1	Mock-Up in fase di progettazione	16
2.2	Schermata di Game Over	17
2.3	Pulsanti di Undo e Redo	19
2.4	Highlight della distribuzione di azioni dell'agente	24
2.5	Barra dell'entropia	25
2.6	Pop-up in caso di brusca variazione di entropia	26
2.7	Errore dell'agente e x rossa generata	27
2.8	Pannello delle discounted rewards	29
2.9	Esempi di highlight dell'azione del player	30
2.10	Esempio delle conseguenze dell'azione del player	32
2.11	Immagine riassuntiva dei vari elementi della UI	33

Elenco delle tabelle

- 3.1 Tabella riassuntiva dei risultati del test SeQ effettuato per valutare la qualità generale del sistema di XAI. 37

Sommario

Negli ultimi anni l'intelligenza artificiale, ed in particolare il machine learning, sta riscuotendo sempre maggiore popolarità. Mentre i modelli utilizzati crescono in complessità tuttavia, tendono a perdere in trasparenza rendendo tali sistemi simili a delle *black-box*, il cui funzionamento interno è di difficile comprensione anche ai loro sviluppatori. Da queste necessità è nato il campo dell'*explainable AI*, che si pone come obiettivo la realizzazione di sistemi di AI più comprensibili e chiari agli esseri umani. Il contesto di lavoro di questa tesi è il gioco strategico a turni DeepCrawl, in cui tecniche di Deep Reinforcement Learning (DRL) sono state usate allo scopo di realizzare nemici con comportamenti *human-like*. Per facilitare la valutazione del comportamento dell'agente è stato realizzato un sistema di XAI, nei panni di una modalità sviluppatore attivabile dall'utente a fine partita. Essa permette di navigare liberamente un replay della partita svolta, forniti di dati aggiuntivi sull'agente, mostrati attraverso una nuova UI realizzata per l'occasione. Nel corso della tesi vengono spiegati aspetti teorici ad alto livello legati al DRL e ai sistemi XAI per comprendere al meglio il lavoro svolto; in seguito viene illustrato nel dettaglio il gioco DeepCrawl, dalle meccaniche di gameplay alle tecniche di DRL utilizzate. Si analizza poi il sistema di XAI realizzato, descrivendo da un punto di vista funzionale e strutturale i vari elementi della UI utili a mostrare le informazioni sull'agente. Infine sono descritti i test di usabilità realizzati per valutare la modalità sviluppatore e la sua efficacia. L'interpretazione di quest'ultimi mostra come il sistema di XAI realizzato faciliti grandemente la valutazione del comportamento dell'agente.

Introduzione

L'intelligenza artificiale sta acquisendo negli ultimi anni un ruolo sempre più importante all'interno della nostra società. In particolare nell'ultimo decennio un suo sottocampo, il *machine learning*, sta attirando sempre maggior attenzione sia dalla comunità scientifica che dai mass media per le sue numerose applicazioni in vari campi. Dal suo utilizzo nella computer vision, con grandi progressi nell'individuazione e tracking di oggetti, ai linguaggi naturali, fino alla creazione di agenti intelligenti per la guida autonoma.

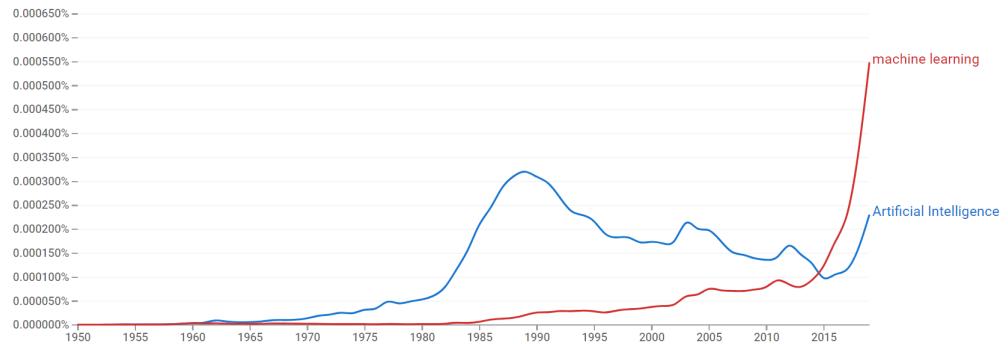


Figura 1: Grafico che illustra la crescente popolarità delle parole *machine learning* e *Artificial Intelligence* nell'ultimo decennio.

Mentre questo campo si espande sia in complessità che nel raggio di applicazione, diventa sempre più importante la trasparenza per rendere il processo decisionale e i risultati ottenuti dall'intelligenza artificiale il più comprensibili possibile. Tuttavia nel machine learning ci imbattiamo spesso in sistemi complessi che danno poche informazioni sul perché, in un dato momento, stanno producendo determinati risultati. Ciò ha portato questi sistemi a essere spesso paragonati a delle metaforiche *black box*: il loro funzionamento interno è sconosciuto persino ai loro sviluppatori, che di conseguenza non sono in grado di spiegare le motivazioni dietro una specifica scelta.

Da queste necessità è nato il campo dell'*Explainable AI* (XAI). Esso si pone come obiettivo la realizzazione di sistemi d'intelligenza artificiale il cui comportamento sia più comprensibile agli esseri umani, capaci di spiegare cosa ha fatto, cosa sta facendo in questo momento e cosa succederà in seguito, mostrando i processi decisionali su cui si basa [3]. Lo sviluppo di questo campo è fondamentale e possiamo presumere che acquisirà sempre maggior importanza in futuro, poiché necessario per l'incremento della fiducia dell'utente in questi sistemi, e non solo. Le tecniche sviluppate possono essere infatti utilizzate allo scopo di valutare la performance di un agente, compito spesso non semplice nei vari contesti in cui non si ha un obiettivo specifico quanto un generico comportamento intelligente da valutare.

È il caso dell'ambito videoludico, contesto nel quale è immersa questa tesi. Lo scopo dell'utilizzo di un sistema di intelligenza artificiale nei videogiochi è solitamente la creazione di comportamenti human-like per i personaggi non giocanti (NPC). In questo contesto comprendere le motivazioni dell'agente nell'eseguire una specifica azione è fondamentale per valutarne il suo sviluppo e capire se l'obiettivo è stato raggiunto [8]. Il gioco su cui questa tesi si concentra è un videogioco strategico a turni di tipo *RogueLike* chiamato

DeepCrawl [6]. Nel gioco il giocatore dovrà affrontare dei nemici in mappe composte da stanze generate proceduralmente, in una struttura a livelli di difficoltà crescente. In DeepCrawl vengono utilizzate tecniche di *Deep Reinforcement Learning* (DRL) per generare nemici credibili, ossia percepiti come intelligenti e che offrano una sfida all’utente, senza tuttavia risultare super-umani e dando sempre una possibilità di vittoria al giocatore. Tuttavia solamente osservare il comportamento dei nemici senza altre informazioni rendeva la valutazione dell’agente molto problematica. Torna qui utile la metafora della black box citata in precedenza: con il sistema originale era molto difficile, in fase di valutazione, sapere con certezza se una specifica azione dell’agente, percepita da un osservatore esterno come inutile, fosse un errore o invece una parte di una strategia più grande poi cambiata o abortita come conseguenza delle mosse del giocatore.

Allo scopo quindi di valutare al meglio il comportamento qualitativo dell’agente è stata introdotta una modalità sviluppatore attivabile su richiesta da un utente. In questa modalità si può assistere ad un replay di una partita appena svolta avendo la possibilità di scorrere avanti e indietro tra le varie mosse. Tramite una nuova interfaccia grafica implementata per l’occasione gli sviluppatori saranno forniti di maggior informazioni sul comportamento dell’agente, come la distribuzione di probabilità sulle possibili azioni disponibile ad ogni turno o un indice di sicurezza basato sull’entropia delle sue azioni e come questo varia come conseguenza delle mosse del giocatore. Questa nuova interfaccia è in grado di far valutare la qualità di un agente pre-addestrato in maniera semplice ed intuitiva, fornendo le informazioni necessarie in forma leggibile agli umani.

Per comprendere se l’obiettivo della tesi fosse stato effettivamente raggiunto sono stati realizzati dei test di usabilità. Quest’ultimi si sono di-

mostrati utili per raccogliere valutazioni qualitative e soggettive sul sistema realizzato. Composti da un sessione di gioco libera ed una guidata, seguito da questionario, le opinioni raccolte ci hanno permesso di valutare in modo efficace la modalità sviluppatore e la UI realizzata per l'occasione.

La tesi è così strutturata:

- nel capitolo 1 tratteremo i lavori correlati alla tesi. Daremo alcuni cenni di teoria riguardo al *machine learning*, in particolare *reinforcement learning*, e *explainable AI*. Entreremo poi nel dettaglio del gioco DeepCrawl, mostrando le meccaniche di gameplay, le tecniche di DRL utilizzate e il pattern architetturale *Entity-Component-System*;
- nel capitolo 2 descriveremo il sistema di XAI, entrando nel dettaglio della UI realizzata, dalla modalità di navigazione dei turni alle informazioni sull'agente e come queste vengono illustrate;
- nel capitolo 3 verranno mostrati i risultati ottenuti attraverso il test di usabilità svolto, interpretando le valutazioni raccolte per valutare l'efficacia del sistema realizzato;
- nelle conclusioni riassumeremo il lavoro svolto e parleremo di possibili sviluppi futuri emersi durante le fasi di realizzazione e test del sistema.

Capitolo 1

Lavori Correlati

In questo capitolo verranno introdotte conoscenze utili al fine di comprendere appieno il lavoro svolto per questa tesi. Verranno spiegati alcuni concetti fondamentali del *machine learning*, con maggiore attenzione sulla branca del *reinforcement learning* e dell'*explainable AI*.

Entreremo poi nel dettaglio del gioco DeepCrawl, ambito di lavoro di questa tesi. Illustreremo le caratteristiche del gioco sia dal punto di vista funzionale che strutturale, spiegandone le meccaniche di gameplay, le tecniche di DRL usate e il pattern architettonicale *Entity-Component-System* utilizzato nel codice.

1.1 Machine Learning ed Explainable AI

Il *machine learning* o apprendimento automatico è una branca dell'intelligenza artificiale dedicata allo sviluppo di algoritmi che portino alla creazione di sistemi intelligenti, capaci di formulare previsioni o decisioni sulla base di un set di dati senza seguire uno specifico insieme di regole. Il programmatore dunque non definisce esplicitamente il comportamento del sistema, ma lascia

che sia questo ad autoregolarsi attraverso i dati osservati, migliorando quindi con l'esperienza.

L'apprendimento automatico si può dividere in tre macroaree

- *Supervised learning*: in questo campo si lavora con dataset con una forte distinzione fra input e output. L'algoritmo si addestra quindi con dati di esempio già forniti di output allo scopo di risolvere problemi di classificazione o regressione sui successivi input;
- *Unsupervised learning*: in questa area il dataset non è fornito di informazioni sull'input, sarà il sistema a dover trovare nuovi pattern nei dati;
- *Reinforcement learning*: questo paradigma si occupa di problemi di decisioni sequenziali, con la supervisione fatta successivamente sotto forma di ricompense che l'agente vuole massimizzare.

Un campo trasversale a quelli citati è il *deep learning*. Questa tipologia di algoritmi cerca di replicare il funzionamento delle reti neurali biologiche, formulando modelli matematici chiamati *Deep Neural Network*, composti da più strati, o *layer*, dove ogni strato calcola i valori per quello successivo affinché l'informazione venga elaborata in maniera sempre più completa. In figura 1.1 un'immagine d'esempio. L'aggettivo deep nasce del gran numero di strati che queste reti possono possedere. Applicando il *deep learning* insieme ai precedenti approcci sono stati raggiunti numerosi miglioramenti nei problemi in cui sono presenti dataset con un alto numero di dimensioni, come immagini o video.

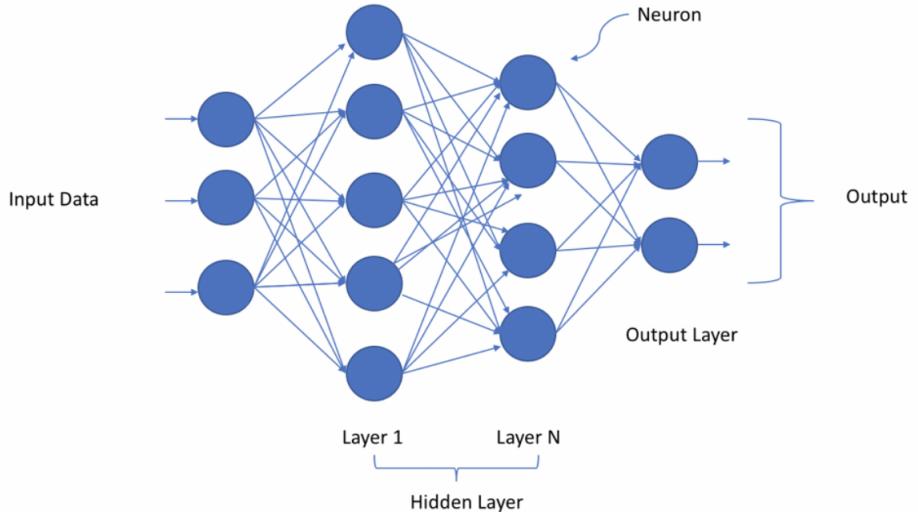


Figura 1.1: Rappresentazione grafica di una *Deep Neural Network*.

1.1.1 Reinforcement Learning

Illustreremo ora la teoria generale del *reinforcement learning* in modo da avere un immagine più chiara del problema affrontato in questa tesi, la valutazione degli agenti nel gioco DeepCrawl. Nel videogioco si fanno uso di tecniche di Deep Reinforcement Learning, una variante più complessa dell’approccio originale, ma che condivide con esso i punti che sono per noi di maggior interesse.

Il *reinforcement learning* è una tecnica di addestramento di agenti che hanno come obiettivo la massimizzazione delle ricompense che ricevono come conseguenza delle loro azioni. L’agente interagisce con l’ambiente in cui è immerso ricevendo le informazioni che lo descrivono raccolte in un insieme di dati, detto *stato*. L’attore si basa su questo stato per scegliere l’interazione capace di dargli la maggiore ricompensa, ripetendo più volte la scelta e migliorandosi in un meccanismo di *trial-and-error*. L’agente ha quindi modo di intuire quali sono le azioni più efficaci valutando la reward ottenuta

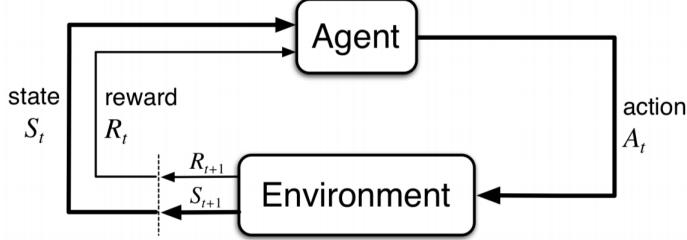


Figura 1.2: Schema che illustra le meccaniche di interazione fra agente e ambiente tramite azioni e reward.

cambiando nel tempo la sua *policy*, cioè la metodologia di scelta delle azioni che definisce il suo comportamento. Uno schema riassuntivo è fornito con la figura 1.2.

Future Discounted Reward

Il risultato di un’azione dell’agente al tempo t viene espresso sotto forma numerica e chiamato reward. Questo valore fa parte dell’input dell’agente al tempo $t + 1$ assieme allo stato s_{t+1} . Con *reward function* si intende la funzione all’interno dell’ambiente che restituisce un certo valore r_t dati lo stato s_t , l’azione a_t e lo stato successivo s_{t+1} . Per permettere all’agente di formulare strategie di più ampio respiro è importante che, ad ogni mossa, l’agente abbia come obiettivo non solo massimizzare la ricompensa immediata ma anche quelle successive. Si definisce quindi *future discounted reward* al tempo t la quantità:

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$

, dove T rappresenta la fine dell’episodio e R_t è la reward function che restituisce la ricompensa istantanea al tempo t . Spesso, G_t viene anche indicato come *ritorno* (o *return*). Allo scopo dunque di ottenere la ricompensa cumulativa più alta l’agente cercherà di massimizzare G_t in ogni turno. Il parame-

tro $\gamma \in [0, 1]$ rappresenta il fattore di sconto e definisce quanta importanza diamo ai reward immediati piuttosto che a quelli lontani nel tempo: se $\gamma = 0$, allora la strategia farà affidamento solo ai reward istantanei senza tenere conto di quello che potrebbe succedere nel futuro; se il nostro ambiente fosse deterministico, cioè se a ogni azione corrispondesse sempre lo stesso reward, allora potremmo usare un $\gamma = 1$.

Policy

La *policy* di un agente è il metodo attraverso cui esso prende le decisioni dopo aver osservato l'ambiente. Solitamente viene indicata con π e rappresenta una mappatura dagli stati alla distribuzione di probabilità sulle azioni:

$$\pi : S \longrightarrow p(A = a | S)$$

Una *policy* di questo tipo è detta stocastica, contrapposta a un possibile modello deterministico che mapperebbe stati a singole azioni in invece di distribuzioni di probabilità. In un ambiente episodico come un gioco a turni possiamo considerare una sequenza discreta di stati, azioni e ricompense ottenute. Questa sequenza è detta *traiettoria* o *rollout* della policy. Ogni rollout accumula i rewards dall'ambiente con i quali viene costruito G_t e obiettivo dell'addestramento è trovare una policy ottima π^* che lo massimizzi:

$$\pi^* = \max_{\pi} \mathbb{E}[G_t | \pi]$$

1.1.2 Explainable AI

Lo scopo di un sistema di *explainable AI* è rendere il comportamento di un agente addestrato quanto più comprensibile agli esseri umani attraverso spiegazioni di varia natura. Vi sono alcuni principi generali che un sistema

XAI dovrebbe seguire per raggiungere questo scopo: esso dovrebbe essere capace di spiegare ciò che può fare e ciò che comprende, dovrebbe cioè avere la capacità di mostrare all'utente le sue azioni in ogni momento e mostrare su quali informazioni si è basato per decidere tali azioni. [3]

Questo obiettivo non è facile da raggiungere sia per la grande varietà di modelli utilizzati sia per la natura estremamente soggettiva di una spiegazione, la cui efficacia cambia al variare delle conoscenze dell'utente. È stato inoltre osservato un trade-off tra performance di un modello e la sua facilità nell'essere spiegato. Spesso infatti modelli ad alte performance (i.e. reti neurali) sono opachi agli esseri umani, mentre modelli più comprensibili (i.e. alberi di decisione) sono meno performanti.

Ad oggi non ci sono metodi condivisi da tutti per misurare quanto un sistema AI sia comprensibile all'uomo. Alcune misure sono di natura più soggettiva come la soddisfazione dell'utente riguardo alla chiarezza e utilità della spiegazione fornita, mentre altre sono più oggettive nel determinare l'efficacia della spiegazione, come il calcolo di un miglioramento nelle capacità decisionali dell'utente mentre usa il sistema in questione.

Un ottimo esempio in questo ambito è il lavoro di Druce et al. [4] L'obiettivo posto era incrementare la fiducia dell'utente in sistemi di DRL. Il dominio d'interesse era un agente che giocava un gioco Atari, Amidar.

Per raggiungere il fine è stata realizzata un interfaccia utente, mostrata in figura 1.3, in grado di offrire una spiegazione del modello su tre livelli: una illustrazione grafica del agente e della sua performance nel gioco; quanto bene avrebbe giocato in ambienti simili a quello in cui era immerso e una spiegazione testuale di cosa implicassero le informazioni grafiche.

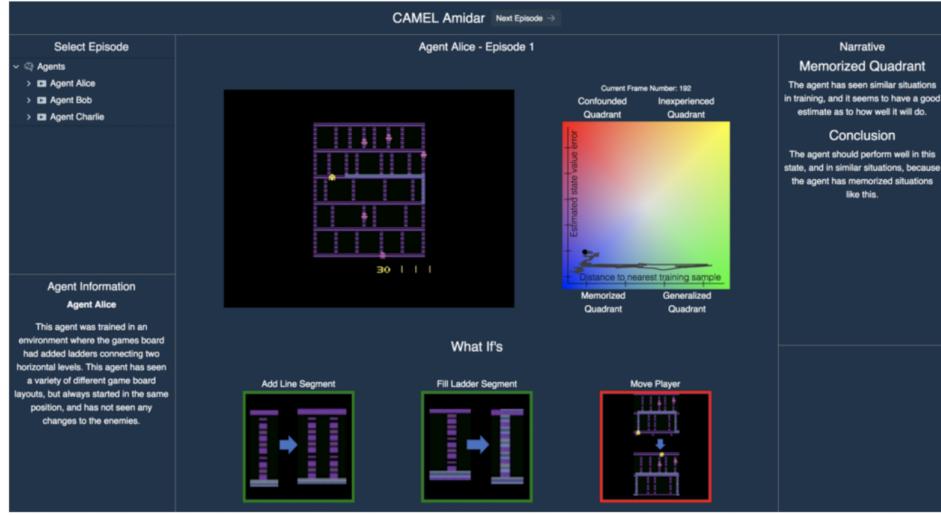


Figura 1.3: Interfaccia grafica di XAI realizzata per il gioco Amidar.

Di particolare interesse è stata l'analisi dell'efficacia dell'interfaccia tramite un test di usabilità, che ha utilizzato degli utenti come tester divisi in due gruppi: a uno di questi è stato mostrato il gameplay dell'agente con l'intefaccia utente di spiegazione, mentre nell'altro quest'ultima era assente. In seguito è stato richiesto di compilare questionari sulla fiducia degli utenti nell'agente. Vediamo quindi in questo caso una misurazione soggettiva dell'efficacia della spiegazione fornita.

Questo lavoro è stato preso come punto di riferimento nello sviluppo della tesi, per il terreno comune dell'ambito videoludico e l'ottimo utilizzo dell'interfaccia grafica per fornire informazioni sul comportamento dell'agente.

1.2 DeepCrawl

Il lavoro svolto in questa tesi si può vedere come un proseguo di Sestini et al. [6], il cui obiettivo era la creazione di un videogioco, un prodotto interamente ludico per Android e iOS, in cui i personaggi non giocanti

presentassero un comportamento intelligente definito attraverso tecniche di Deep Reinforcement Learning Lo scopo era dunque la creazione di agenti:

- *Credibili*, ossia percepiti dal giocatore come intelligenti e in modo da rappresentare una sfida per il giocatore;
- *Imperfetti*, perché la creazione di un super-agente avrebbe reso la sfida non proporzionata alle capacità dell'utente e non adatta ad un prodotto ludico;
- *Model-free*, capaci quindi di formulare la corretta strategia in diverse situazioni, senza imposizioni nella condotta da parte del programmatore;
- *Variabili* nelle strategie che possono formulare, in modo da definire diverse tipologie di nemici.

DeepCrawl è un gioco strategico di tipo *RogueLike*. Con questo termine solitamente si indica un gioco di ruolo a turni, in 2D con una visuale in terza persona, caratterizzato da elementi come mappe casuali e morte permanente.

In DeepCrawl il giocatore dovrà avanzare in ogni livello del gioco, ognuno di questi caratterizzato da un numero crescente di stanze generate casualmente, con all'interno nemici completamente controllate da policy addestrate con algoritmi di DRL che l'utente dovrà eliminare per considerare il livello completato.

1.2.1 Gameplay

Ambiente

Ogni livello del gioco è composto da un numero crescente di stanze. Una stanza è composta da una griglia di forma casuale, di dimensioni massime

10x10 caselle. In ogni casella è possibile trovare il giocatore, un nemico, un ostacolo che blocca il passaggio o oggetti raccoglibili. Questi oggetti possono essere armi da mischia, armi a distanza o pozioni, quest'ultime a loro volta divise in buff o pozioni di cura. La stanza è generata casualmente sia nella forma che nella presenza e disposizione degli oggetti.

Ogni personaggio ha a disposizione un inventario, contenente un'arma da mischia, una arma a distanza e una pozione. Non è possibile possedere più di un oggetto di ciascun tipo: quando si raccoglie un oggetto, andando sulla casella in cui risiede, il precedente salvato verrà eliminato.

Caratteristiche

Personaggio giocante e nemici hanno caratteristiche personali che influenzano il combattimento:

- *Punti vita*: definisce il numero dei danni che un personaggio può ricevere. Quando i punti vita scendono a 0, il personaggio muore;
- *Attacco*: influenza il calcolo dei danni quando viene effettuato un attacco in mischia;
- *Destrezza*: influenza il calcolo dei danni quando viene effettuato un attacco a distanza;
- *Difesa*: modifica i danni ricevuti quando il personaggio subisce un attacco, sia in mischia che a distanza.

Spazio delle azioni

È importante far notare ai fini degli obiettivi del progetto che in DeepCrawl il giocatore e i nemici hanno lo stesso spazio delle azioni, possono svolgere

cioè lo stesso tipo di mosse, dai modi di attaccare, alla raccolta degli oggetti fino all'uso delle pozioni. Inoltre, hanno a disposizione le stesse informazioni riguardante l'ambiente in cui interagiscono, vedendo cioè lo stesso stato. Questo li pone sullo stesso livello e sposta l'attenzione sul comportamento dell'agente, in quanto risulta più facile comprendere le sue mosse e cercare di capire la sua strategia. Ogni personaggio può effettuare potenzialmente 17 azioni diverse:

- *8 azioni di movimento o attacchi in mischia:* è possibile muoversi di una casella in una qualsiasi delle 8 direzioni della scacchiera. Se la casella in questione è occupata da un ostacolo come una colonna, il movimento non sarà eseguibile, se invece è presente un oggetto raccoglibile il movimento verrà eseguito e sarà raccolto l'oggetto. In alternativa se la casella è occupata da un nemico, sarà effettuato un attacco in mischia;
- *8 attacchi a distanza:* si può scegliere di usare la propria arma a distanza invece di muoversi o attaccare in mischia. L'attacco può essere effettuato in una delle 8 direzioni della scacchiera, ma sarà permesso solo se effettivamente presente un nemico in quella direzione, a una distanza massima definita dal raggio dell'arma;
- *Uso della pozione:* a differenza delle precedenti, è possibile usare questa azione senza perdere il turno, potendo fare dunque un'altra azione dopo questa. Gli effetti della pozione cambiano a seconda del tipo:
 - *Pozione di cura:* recupera 5 punti vita, senza però superare la vita massima;
 - *Buff:* aumenta Attacco e Difesa per 3 turni.

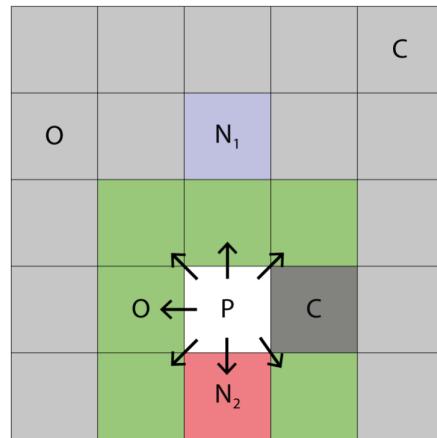


Figura 1.4: Esempio delle possibili azioni in un turno: il personaggio giocante (P), può muoversi in una qualsiasi delle caselle verdi e andando sulla casella a sinistra raccoglierà l'oggetto (O). Non può invece muoversi verso la colonna (C) e se seleziona la casella con il nemico (N2) effettuerà un attacco in mischia. Può in alternativa decidere di eseguire un attacco a distanza verso N1. Finita una di queste azioni avrà completato il proprio turno.

In figura 1.4 vediamo una rappresentazione del turno e delle possibili mosse che può fare un personaggio.

1.2.2 DRL in DeepCrawl

In questa sezione vedremo ad alto livello il sistema di DRL utilizzato nel gioco. Ci concentreremo sugli aspetti più interessanti per lo scopo della tesi, ossia quelli che ci sono più utili per valutare il comportamento dell'agente durante l'episodio.

Policy

Per definire la policy dell'agente, viene utilizzata un algoritmo chiamato *Proximal Policy Optimization* [5] che utilizza una rete neurale con numerosi strati. Questa riceve in ingresso tutte le informazioni d'interesse per l'agente, dalla mappa di gioco alle condizioni del giocatore, inclusive dell'inventario e

varie caratteristiche. In uscita viene prodotta una distribuzione di probabilità composta da 17 valori, ciascuna corrispondente alla probabilità dell’agente di eseguire una delle corrispondenti azioni viste nella sezione 1.2.1.

È importante notare che nel modello utilizzato non viene selezionata l’azione più probabile, ma viene semplicemente campionato il vettore delle probabilità a seconda dei valori calcolati. Questo fattore porta l’agente a non scegliere sempre ciò che considera la migliore azione, commettendo tecnicamente uno sbaglio. Ciò porta il suo comportamento a essere considerato più umano, oltre a introdurre maggiore varietà nelle situazioni, evitando la possibilità di cicli di azioni sempre uguali.

Reward Function

Con l’intenzione di far sì che l’agente riuscisse a estrapolare una strategia in modo autonomo senza essere eccessivamente influenzato dalle singole azioni, in DeepCrawl viene utilizzata una *reward function* molto sparsa:

$$R(t) = -0.01 + \begin{cases} -0.1 & \text{per un movimento impossibile} \\ +10 * hp & \text{in caso di vittoria} \end{cases}$$

dove hp è l’ammontare normalizzato di punti vita rimasti all’agente. Il valore iniziale -0.01 introduce un economia delle azioni nella strategia dell’agente, incoraggiandolo a evitare mosse inutili. Il valore di -0.1 velocizza l’addestramento dell’agente di fronte a mosse impossibili da eseguire, mentre il valore $+10 * hp$ alza l’asticella per l’obiettivo dell’attore: l’agente deve infatti non solo vincere, ma farlo con più HP possibili. Questo fattore aiuta ad addestrare più velocemente la capacità dell’agente nel riconoscere l’importanza degli HP in un gioco di questo tipo.

1.2.3 Unity ed Entity-Component-System

Il gioco è stato realizzato usando Unity [7], un motore grafico per la realizzazione di giochi 3D/2D e altri contenuti interattivi per varie piattaforme. È basato sul linguaggio di programmazione C#. La scelta di questo motore grafico è dovuta alla disponibilità del framework open-source Unity Machine Learning Agents Toolkit (ML-Agents), dedicato all'addestramento di agenti di DRL.

Nei riguardi invece dell'aspetto strutturale di DeepCrawl, attore protagonista in esso è il pattern Entity-Component-System(ECS). Molto usato nello sviluppo di giochi, questo pattern architettonico segue il principio della *composizione a favore dell'ereditarietà* e fornisce grande flessibilità oltre a rimuovere il problema delle grandi gerarchie di ereditarietà difficili da mantenere. Vediamo i principali elementi di questo pattern:

- *Component*: un componente è una struttura dati paragonabile ad una *struct* del linguaggio C. Non presenta metodi ed è responsabile d'immagazzinare dati. Ogni componente descrive un certo aspetto di un'entità;
- *Entity*: un'entità è un oggetto che esiste nel mondo di gioco (nemico, posizione, colonna, etc...). È una struttura molto semplice, poco più di una lista di componenti con un ID unico per identificarla. I componenti possono essere aggiunti e rimossi durante l'esecuzione del gioco, modificando quindi dinamicamente le caratteristiche dell'entità;
- *System*: il sistema è l'elemento che introduce la logica di gioco. Ciascuno di essi lavora su determinate entità che posseggono dei particolari gruppi di componenti. Ad ogni frame i sistemi vengono aggiornati in parallelo, identificando su quali entità dovranno lavorare. Ogni sistema

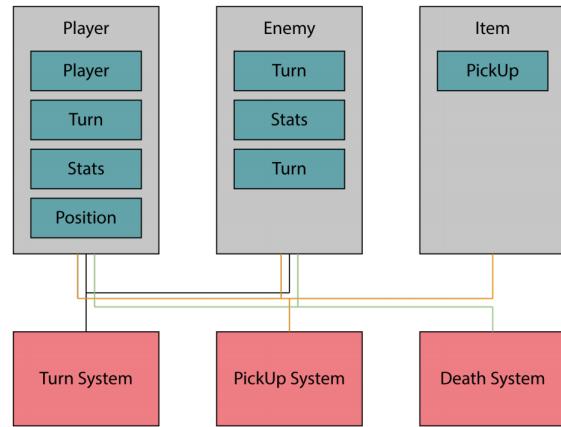


Figura 1.5: Esempio di ECS applicato in DeepCrawl: in rosso i sistemi, che agiscono sulle entità come Player, Enemy e Item sulla base dei componenti che esse possiedono, qui rappresentati in blu.

avrà delle responsabilità riguardo delle particolari meccaniche di gameplay del gioco. L’uso combinato di questi crea quindi tutta la logica di un videogioco.

Un esempio dell’ECS è illustrato con la figura 1.5. DeepCrawl presenta una struttura ibrida, che fa largo uso dell’ECS nelle meccaniche di gameplay, utilizzando un approccio Object Oriented nella la creazione e gestione della mappa.

Capitolo 2

Sistema di Explainable AI

Nel capitolo precedente abbiamo illustrato tutto ciò che era necessario per comprendere al meglio la natura del problema affrontato: la valutazione di un agente allenato tramite DRL in un contesto videoludico come quello di DeepCrawl.

In questo capitolo entreremo nel dettaglio della soluzione proposta. Come già accennato nell'introduzione, al fine di facilitare la valutazione del comportamento dell'agente è stata realizzata una modalità sviluppatore. Questa modalità dà la possibilità all'utente di rivedere la partita appena svolta, ricominciando dall'inizio e andando avanti e indietro tra i turni del giocatore e i turni del nemico, visualizzando informazioni non visibili durante la normale partita.

Nei prossimi paragrafi mostreremo nel dettaglio l'interfaccia utente (UI) realizzata, implementata tramite gli strumenti offerti dal pattern Entity-Component-System. Mostreremo quindi tutti gli elementi della UI, dalla modalità di navigazione dei turni alle informazioni rese disponibili sull'agente e come quest'ultime siano ottenute e gestite durante la partita. Per ogni elemento indicheremo le principali funzionalità e il motivo per cui esso è stato

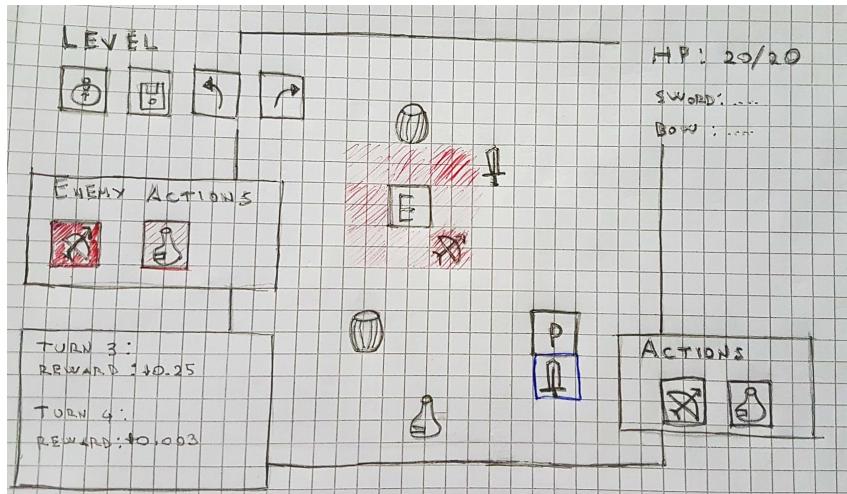


Figura 2.1: Un esempio di Mock-up prodotto durante la fase di progettazione

aggiunto.

2.1 Progettazione

Tutti gli elementi descritti di seguito sono stati implementati dopo un'attenta fase di progettazione e mock-up utile a definire le funzionalità mancanti e da aggiungere. Un esempio di artefatto prodotto durante questa fase è visibile in figura 2.1. Abbiamo individuato quali potevano essere le necessità e le difficoltà di un utente che ha come scopo quello di spiegare il comportamento di un agente pre-addestrato tramite algoritmi di DRL. Dopo questa prima fase astratta, abbiamo quindi definito gli elementi concreti che andremo a vedere nei prossimi paragrafi.

2.2 Game Over

La schermata di Game Over, in figura 2.2, è l'elemento della nuova interfaccia che fornisce la prima informazione sulla condotta dell'agente durante



Figura 2.2: Schermata di Game Over. Questa dà la possibilità di entrare nella modalità sviluppatore o cominciare un'altra partita. Nella sezione *Agent's Reward* è mostrata la somma delle discounted reward dell'agente.

la partita.

Oltre a dare la possibilità di entrare nella modalità sviluppatore o di cominciare una nuova partita nella modalità classica, nella sezione *Agent's Reward* viene mostrata la somma complessiva di tutte le discounted reward, evidenziando di rosso se la somma è negativa o nulla, di verde se positiva. Il valore mostrato è una misura quantitativa dell'andamento complessivo dell'agente, particolarmente utile allo sviluppatore che, conoscendo i valori che le discounted reward possono assumere (sezione 1.2.2), ha un'idea più chiara di quanto positiva o negativa sia stata la performance dell'agente nella partita.

La modalità sviluppatore dà la possibilità di rivedere la partita e le mosse svolte in una forma di replay dell'episodio. Premendo il tasto corrispondente nella schermata di Game Over, la partita ricomincia dal primo turno, ossia dalla prima mossa eseguita.

2.3 Navigare l'episodio

Per rivedere le varie mosse della partita era necessario che l'interfaccia fornisse all'utente la possibilità di muoversi fra i vari turni, andando avanti e indietro liberamente. Per turno indichiamo tutte le volte in cui uno dei personaggi compie un'azione: se ad esempio il giocatore usa una pozione ciò conta come un turno, nonostante nel successivo la mossa spetterà sempre allo stesso, come spiegato nella sezione 1.2.1.

Abbiamo ritenuto importante fornire questo strumento per navigare liberamente l'episodio. Inserire un elemento interattivo nell'interfaccia dà maggior controllo all'utente, che non segue in maniera passiva il replay dell'episodio ma può concentrarsi sulle parti più d'interesse della partita. Dato il gran numero di dati presenti nell'interfaccia tornare indietro e nuovamente avanti su una singola azione può essere d'aiuto per non perdersi cambiamenti in queste informazioni, aiutandolo a comprendere come le mosse del player influenzino le strategie del nemico.

A questo scopo sono stati aggiunti i pulsanti di *undo* e *redo*, che appaiono nell'interfaccia posizionati in alto a sinistra. Premere uno dei pulsanti porterà l'episodio avanti o indietro di un turno, fino alla fine o all'inizio della partita. I pulsanti passano da un bianco a grigio scuro quando si è raggiunto uno dei due estremi dell'episodio e non è possibile andare oltre. Un esempio è mostrato in figura 2.3.

2.3.1 Salvataggio e Caricamento

Per avere a disposizione un replay della partita è stato necessario gestire il salvataggio di tutte le informazioni utili durante l'episodio e il caricamento



Figura 2.3: Frame del gioco; in alto a sinistra (1) i pulsanti di Undo e Redo. Il secondo da sinistra, in grigio scuro, ci avvisa che non possiamo andare più avanti.

di queste.

I dati di ogni turno sono organizzati in una struct, quest’ultima salvata in un oggetto della classe *SerializableDictionary* [2]. Questa classe è risultata particolarmente utile: oltre alla comodità nell’accesso dei dati usando come chiave il turno corrispondente, questo dizionario risulta infatti serializzabile; è dunque possibile trasformare l’istanza in un file JSON, molto semplice da consultare, accelerando le fasi di debug. I dati salvati sono di vario tipo e includono quelli necessari per ricostruire la partita in ogni turno, come la posizione degli attori, le loro caratteristiche, la disposizione degli oggetti raccoglibili sulla mappa e altro ancora. In aggiunta a queste, sono state salvate delle informazioni aggiuntive legate a delle statistiche dell’agente (i.e. la distribuzione di probabilità sulle azioni a ogni turno), con alcune eccezioni riguardanti dei dati sul player. Tutte queste informazioni saranno di vitale importanza per la nostra UI, come vedremo in seguito.

Utilizzando il pattern ECS sono stati dunque realizzati il *Save System* e il *Load System*, dalle omonime responsabilità.

Save System

Questo sistema ha il compito di operare ad ogni azione di un attore. Per fare questo è stato inizialmente utilizzato un nuovo componente *toSave*, associato all'entità che aveva svolto l'azione. Questo però portava alcune complicazioni di vario genere, ed è risultato più efficiente utilizzare un booleano come flag per far partire il sistema. Al momento dell'attivazione si specifica anche il tipo di salvataggio, ad esempio se un'azione dell'agente o dell'utente, tramite un enum *typeOfSave*.

Al suo avvio il sistema provvede a incrementare il contatore del turno e a salvare i dati del player, del nemico e della mappa di gioco nel dizionario. Tuttavia, non tutte le informazioni che verranno mostrate nella modalità sviluppatore sono immediatamente accessibili per essere salvate al turno corrispondente; alcune di queste, come ad esempio la futura mossa del player, devono essere caricate al momento del salvataggio del turno successivo.

Il *Save System* gestisce anche la responsabilità di salvare su disco il file JSON contenente le informazioni di tutta la partita. Questa procedura si avvia automaticamente a fine partita ma è anche invocabile con un apposito tasto della UI. Questa funzionalità era stata inserita nelle fasi iniziali del progetto allo scopo di salvare e successivamente caricare la partita, dando la possibilità di chiudere il gioco e continuare in un secondo momento. L'uso del *SerializableDictionary* e una definizione più precisa della modalità sviluppatore hanno reso poi non necessaria questa feature per l'esecuzione del replay dell'episodio. Tuttavia si è dimostrata particolarmente utile in fase di Debug per studiare la correttezza dei dati nella struct di salvataggio ed è

stato deciso di mantenerla per sviluppi futuri, come la possibilità di caricare interamente una partita su richiesta dell’utente.

Load System

Il *Load System* ha il compito di caricare le informazioni contenute nel dizionario quando uno dei pulsanti di *undo* o *redo* viene premuto. Come nel *Save System* anche qui vengono utilizzate delle variabili booleane come flag per far partire il sistema, ma anche per distinguere se caricare il turno precedente o quello successivo a seconda del pulsante selezionato.

Scelto quale turno caricare, il sistema invoca i metodi *playerLoad()*, *enemyLoad()* e *roomLoad()* che accedono ai dati del giocatore, del nemico e della mappa presenti nel dizionario, utilizzando come chiave il contatore del turno. I dati menzionati includono anche le informazioni non visibili durante la partita che il *Load System* userà nei metodi citati per modificare gli elementi grafici della UI, aggiornandola turno per turno.

Il sistema ha anche la responsabilità di inizializzare la modalità sviluppatore quando il tasto corrispondente nella schermata di Game Over viene premuto, attivando un flag chiamato *startReplay*. Vengono dunque caricati i dati del primo turno della partita e attivati i componenti della nuova UI.

2.4 Informazioni sugli attori

Per illustrare le informazioni aggiuntive della modalità sviluppatore sono stati utilizzati il più possibile gli elementi già presenti nella mappa e nell’interfaccia grafica originale. L’intento era mostrare in modo semplice ed intuitivo perché l’agente avesse fatto una determinata mossa, senza costringere l’utente a leggere spiegazioni eccessivamente verbose: il DRL infatti

è ricco d'informazioni quantitative, come statistiche e numeri, che possiamo utilizzare per dare un'effettiva spiegazione del comportamento (i.e. i plot dell'evoluzione del reward e dell'entropia visibili nel lavoro originale [6]). Tuttavia, far leggere dei semplici numeri a un utente umano non è altrettanto efficace quanto mostrare attraverso degli artefatti visivi e altamente intuitivi le stesse informazioni. Per raggiungere la massima spiegabilità di queste informazioni abbiamo deciso infatti di aggiungere degli elementi grafici che riassumessero queste statistiche e che fossero in perfetta simbiosi con il *look-and-feel* del gioco stesso, senza snaturare l'interfaccia già presente dando un senso di familiarità ai nuovi elementi. Dobbiamo infatti ricordare che questo sistema XAI è destinato a chi ha perfetta conoscenza di DeepCrawl.

Il concetto di rendere intuitive le informazioni quantitative fornite dagli elementi di cui è composta una black-box è lo scopo fondamentale di ogni sistema di XAI: la corretta valutazione della “*leggibilità*” di questi elementi è causa infatti della buona riuscita del sistema. Vediamo dunque quali sono le strategie di comunicazione che abbiamo adottato.

2.4.1 Distribuzione di probabilità delle azioni

Nella sezione 1.2.2 abbiamo spiegato il processo decisionale dell'agente. Fondamentale per questo meccanismo è la distribuzione di probabilità delle varie azioni, calcolata di turno in turno. La distribuzione offre un ottimo scorcio sui meccanismi interni dell'agente, perché fornisce in maniera chiara e completa l'output della rete neurale realizzata che rappresenta la sua *policy*, il sistema fondamentale che permette all'agente di prendere le sue decisioni. Per visualizzare la distribuzione in modo intuitivo si è scelto di evidenziare le varie azioni con un gradiente dal giallo al rosso a seconda del valore di probabilità per ogni azione: un colore giallo acceso indica bassa probabilità mentre un

colore rosso acceso indica alta probabilità. Plausibilmente, l’agente sarà più propenso a effettuare l’azione che risulta avere il valore di rosso più acceso. Come approfondiremo nelle sezioni successive però, questo non è sempre vero.

Per quanto riguarda le azioni di movimento e attacco in mischia, vengono evidenziate le caselle della mappa di gioco corrispondenti alle varie mosse. Per quanto riguarda invece gli attacchi a distanza e l’utilizzo della pozione, è stato aggiunto un nuovo pannello in cui viene evidenziato il colore di queste azioni. Ricordiamo che, come spiegato in precedenza, il numero delle azioni di attacco a distanza è 8, dato che è possibile lanciare una freccia in una qualsiasi delle direzioni della scacchiera. Tuttavia abbiamo deciso di mostrare solo se è stata effettivamente utilizzata una di queste: abbiamo infatti sommato la probabilità delle singole azioni e usato questo valore per calcolare il gradiente con cui colorare il simbolo dell’attacco a distanza. Il motivo di questa scelta è legato all’impatto visivo che l’alternativa avrebbe avuto: colorare infatti tutte le caselle in cui è possibile attaccare a distanza avrebbe riempito eccessivamente la mappa di colori, perdendo l’immediatezza e intuitività di questi elementi grafici. Un esempio del risultato finale è visibile in figura 2.4.

Un vettore di 17 variabili di tipo float viene utilizzato nella struct di salvataggio per gestire la distribuzione di probabilità. Questo vettore viene poi letto dal *Load System* per calcolare il gradiente ed evidenziare le informazioni descritte in precedenza.



Figura 2.4: Un esempio di highlight della distribuzione di azioni dell’agente: le caselle in cui esso può muoversi e le azioni di attacco a distanza e uso della pozione nel pannello *Enemy Actions* sono colorate con un gradiente dal giallo al rosso a seconda della loro probabilità di essere eseguite. In questo caso muoversi per prendere la spada è la mossa estremamente più probabile.

2.4.2 Entropia

Un dato d’appoggio che abbiamo voluto inserire è quello dell’entropia della distribuzione di probabilità della scelta delle azioni, calcolabile come:

$$H = - \sum_x p(x) \log(p(x)),$$

dove, in questo caso, $p(x)$ rappresenta la probabilità di eseguire di una particolare azione. Questo dato fornisce informazioni interessanti sull’agente: valori alti di entropia corrispondono infatti a una maggiore incertezza per lui, in quanto significa che molte azioni hanno un simile probabilità di essere effettuate e dunque una maggiore predisposizione a non scegliere la mossa con probabilità più alta, che nel nostro sistema è considerato un errore. Un valore basso di entropia significa invece un alto livello di sicurezza dell’agente:



Figura 2.5: Un esempio della barra dell’entropia: la barra si riempie a seconda dell’entropia della distribuzione di probabilità nel turno, usando un gradiente dal giallo al rosso.

questo infatti avrà solamente pochi valori di probabilità molto alti, rendendo la scelta dell’azione giusta molto più semplice.

Per integrare questo dato nell’interfaccia grafica abbiamo usato uno *slider*, un elemento grafico di Unity, normalmente usato in ambito videoludico per rappresentare quantità variabili, come una barra della vita. La barra viene aggiornata turno per turno, mostrando l’entropia dell’agente, come mostrato in figura 2.5.

Inoltre la variazione dell’entropia turno per turno evidenzia la reazione dell’agente alle mosse del giocatore. Per evidenziare questo aspetto, nel metodo *enemyLoad()* del *Load System* viene fatto un confronto tra l’entropia del turno corrente e quella del precedente: se la differenza supera una certa soglia viene generato un pop-up sopra il nemico, per attirare l’attenzione su questo cambiamento. Questo elemento è utile per notare se le mosse del giocatore causano cambiamenti nella strategia dell’agente, portandolo da una condizione di sicurezza nelle sue decisioni a una condizione d’incertezza o



Figura 2.6: Un esempio del pop-up che viene generato quando c'è una differenza superiore a una certa soglia tra l'entropia del turno precedente e di quello attuale.

viceversa. È stato notato, ad esempio, che se il giocatore è nelle immediate vicinanze del nemico e usa un buff l'agente mostrerà un cambio brusco di entropia e una variazione della strategia. Il dettaglio è mostrato in figura 2.6.

2.4.3 Errore dell'agente

Come visto nel precedente capitolo, uno degli obiettivi per il gioco Deep-Crawl era la creazione di agenti imperfetti: un nemico super-umano, infatti, potrebbe alla lunga creare frustrazione nel giocatore, in quanto difficile da battere e di conseguenza poco divertente. Per evitare ciò invece di scegliere l'azione più probabile secondo la distribuzione calcolata in quel turno, viene campionato il vettore delle probabilità a seconda dei valori calcolati. In questo modo tutte le volte che l'agente non sceglie l'azione più probabile, vuol dire che sta compiendo un errore, mostrando un comportamento più umano.

Abbiamo scelto di evidenziare quando avviene un errore, utilizzando una X rossa posta sotto la barra dell'entropia, come mostrato in figura 2.7. Questa icona è un elemento della UI sempre attivo, ma solitamente nascosto

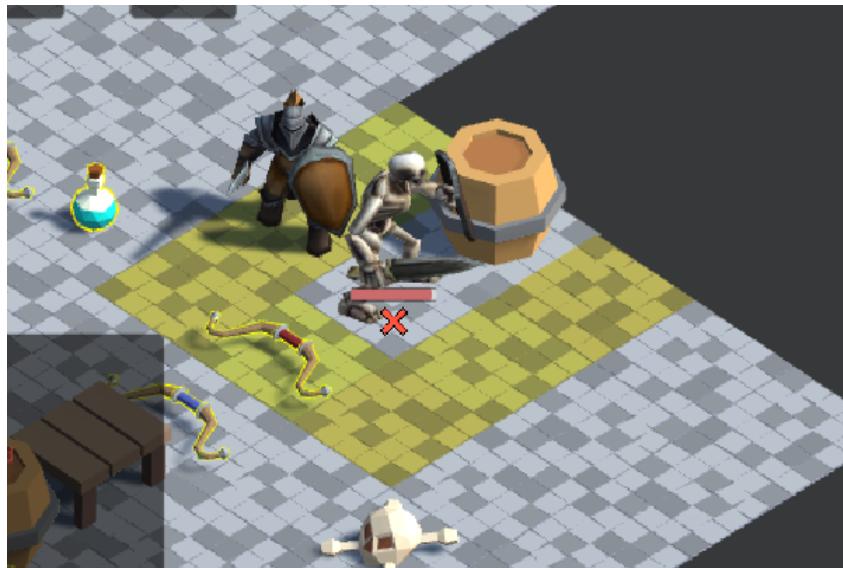


Figura 2.7: Frame in cui è segnalato un errore dell’agente: la x rossa sotto il nemico indica che nel turno precedente non è stata scelta la mossa più probabile.

portando la trasparenza dell’immagine al massimo. Il booleano *enemyError* presente nella struct di salvataggio segnala se in quel turno l’agente ha commesso un errore, e in caso positivo l’immagine viene resa visibile.

Tramite questo stratagemma l’osservatore sarà quindi informato, attraverso l’interfaccia, dei momenti in cui l’agente commette un errore. In questo modo cerchiamo di dare una spiegazione più corretta del perché l’agente effettui determinate mosse in particolari momenti: in presenza di una mossa contorta, senza questo artefatto non era possibile dire se l’agente avesse eseguito questa perché addestrato non correttamente o semplicemente perché ha commesso un errore.

2.4.4 Reward

Nella sezione 1.2.2 abbiamo visto come le reward istantanee e discounted vengono calcolate. Per non influenzare le strategie dell’agente è stato scelto

di non assegnare ricompense per azioni considerate utili come prendere una posizione da terra o attaccare l’agente, ma calcolare le reward principalmente guardando se l’agente vince o perde, e in caso di vittoria normalizzare questi valori secondo i punti vita rimasti.

Nonostante in DeepCrawl le singole discounted reward non forniscano informazioni interessanti sulle mosse del nemico, in molti altri sistemi di DRL queste possono rappresentare un ottimo strumento per valutare il comportamento dell’agente. Per questo motivo abbiamo sperimentato vari modi di mostrarle nella modalità sviluppatore. Abbiamo inizialmente utilizzato dei pop-up con i valori delle reward, generati a ogni turno del nemico. Per non riempire troppo lo spazio attorno all’agente abbiamo poi scelto di utilizzare un pannello nella UI in cui stampare le discounted reward, evidenziando in grassetto la ricompensa corrispondente al turno osservato. Questa versione presentava tuttavia alcune problematiche: il sistema di salvataggio non tiene infatti conto delle azioni illegali dell’agente, che non influenzano il gioco ma che vengono considerate decisioni e dunque ne viene calcolata la reward. È il caso di un attacco a distanza in una direzione in cui non è presente l’avversario; in questo caso il turno rimane dell’agente, perché l’azione non è eseguibile e il *Save System* non è attivato. Questo portava a volte a generare una lista delle reward di lunghezza maggiore del numero dei turni e rendeva impossibile associare il turno alla sua ricompensa. Nella versione finale è stato dunque rimosso il grassetto per evidenziare la reward corrispondente.

Un’immagine d’esempio del pannello è disponibile in figura 2.8, riassuntiva anche dei vari elementi dell’interfaccia.



Figura 2.8: Un esempio di lista delle discounted rewards visibili nel pannello posizionato nella parte destra dell’interfaccia. Da questo pannello è possibile osservare l’evoluzione del future discounted reward al passare dei turni.

2.4.5 Azione del Player

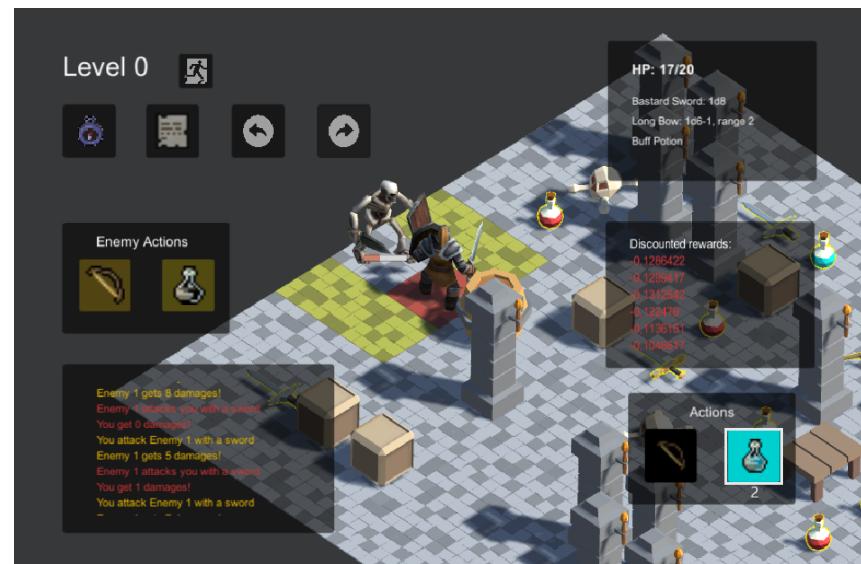
A differenza di tutti gli altri dati mostrati, come si può dedurre dal nome questa è l’unica informazione aggiuntiva che non riguarda l’agente, ma il giocatore. Si va infatti ad evidenziare la mossa che l’utente ha fatto nel turno successivo: in questo particolare gioco, in caso di movimento o attacco in mischia viene colorata la casella corrispondente sulla scacchiera; se invece l’azione scelta è un attacco a distanza o l’uso di una pozione il bottone nel pannello azioni viene evidenziato. Un esempio in figura 2.9.

La scelta dell’azione che l’utente vuole effettuare viene gestita dall’*Action System*, uno dei sistemi del gioco. È utilizzato un intero che può assumere un valore tra 0 e 16, ciascuno rappresentante una mossa nello spazio delle azioni (per maggiori informazioni vedere la sezione 1.2.1). Questo valore viene salvato di turno in turno e utilizzato nel metodo *nextActionHighlight()* del *Load System* per aggiornare l’interfaccia grafica.

Mostrare la mossa successiva dell’utente facilita quest’ultimo a ricordarsi come si stava svolgendo la partita ed evidenzia come le sue mosse influenzino il comportamento dell’agente, aiutandolo nella valutazione di esso. Come



(a)

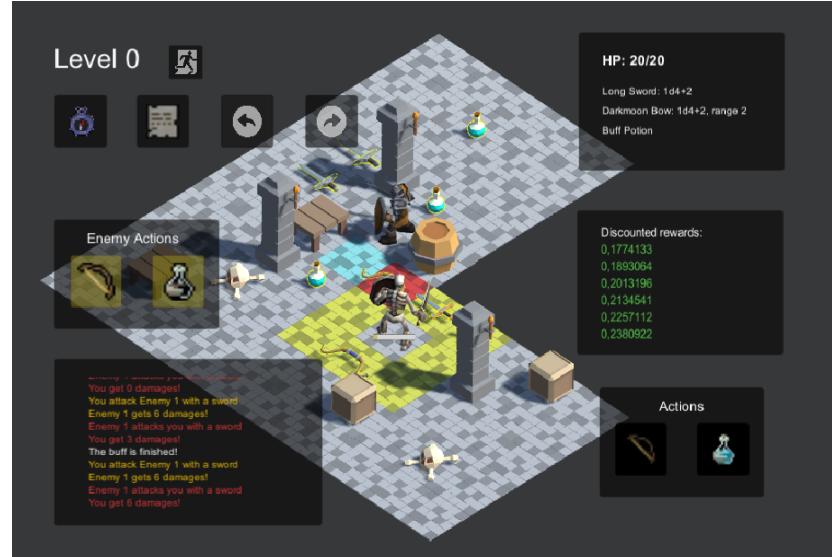


(b)

Figura 2.9: Esempi di highlight dell'azione del player: in (a), al centro dell'immagine (1) la casella in cui si muoverà il giocatore è evidenziata di azzurro; in (b), in basso a destra (2) la mossa che farà il giocatore nel prossimo turno è evidenziata colorando il corrispondente pulsante nel pannello delle azioni.

mostrato in figura 2.10 infatti, mosse del player possono portare a cambiamenti nella strategia dell’agente, che la distribuzione di probabilità e la barra della variazione dell’entropia contribuiscono a mostrare.

Con questa descrizione si chiude la disanima degli elementi caratterizzanti l’interfaccia grafica che definisce il sistema di XAI sviluppato. In figura 2.11 è possibile vedere uno screen della UI che riassume tutti gli elementi nella loro interezza, evidenziando come questi si mescolino bene al *look-and-feel* originale. Per ogni dettaglio è stata data una descrizione approfondita della loro funzionalità e del motivo per cui sono stati progettati e implementati. Nel prossimo capitolo vedremo il risultato di alcuni test di usabilità effettuati per l’occasione: questo procedimento è infatti risultato fondamentale per valutare la qualità del sistema sviluppato e per capire se l’obiettivo di rendere più semplice e intuitiva l’analisi del comportamento di agenti addestrati tramite DRL è stato raggiunto.



(a)



(b)

Figura 2.10: Esempio delle conseguenze dell'azione del player: spostarsi in una casella più vicina all'agente ha portato questo a rivalutare la sua strategia, prendendo in considerazione altre mosse possibili, aumentando complessivamente l'entropia.

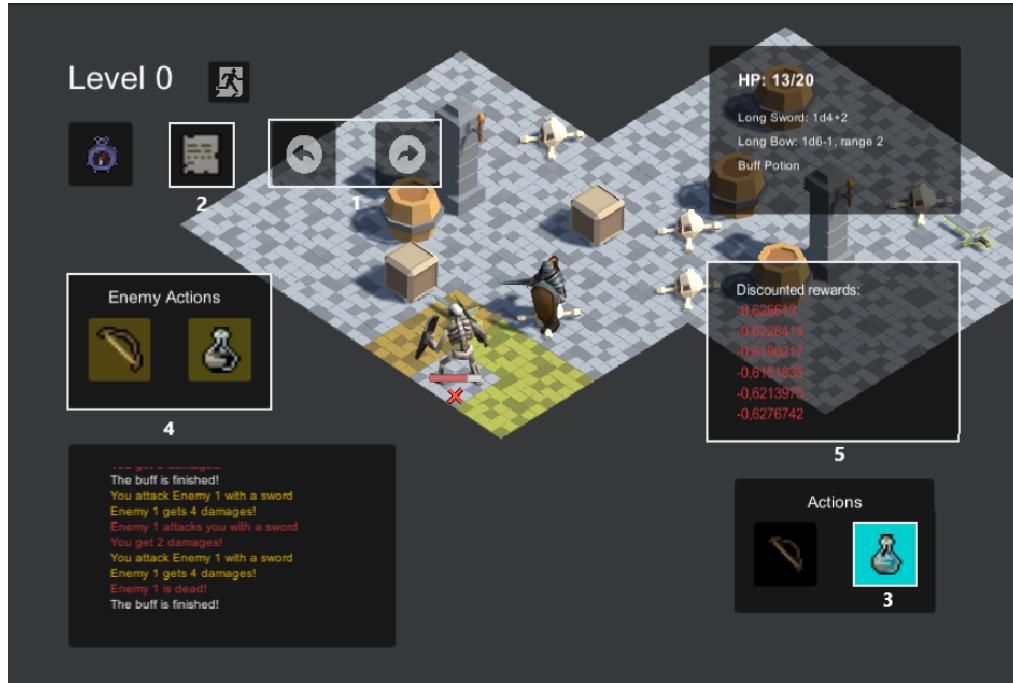


Figura 2.11: Immagine riassuntiva dei vari elementi della UI: in alto a sinistra (1) i tasti di *Undo* e *Redo* per navigare l'episodio; accanto a questi il pulsante (2) per salvare su disco i salvataggi della partita; l'azione successiva del player è evidenziata nel pannello azioni (3) mentre la distribuzione di probabilità delle mosse dell'agente è evidenziata sulla scacchiera e nel pannello *Enemy Actions* (4). Sotto il nemico sono visibili la barra dell'entropia e la x di errore e infine la lista delle discounted rewards è visibile nel pannello a destra (5).

Capitolo 3

Test di Usabilità

Nel capitolo precedente siamo entrati nel dettaglio del sistema di explainable AI proposto. Abbiamo analizzato la modalità sviluppatore, concentrando i nostri sforzi sull’interfaccia grafica realizzata, dato il suo ruolo chiave nel mostrare le informazioni utili in modo intuitivo attraverso artefatti visivi.

In questo capitolo mostreremo i test di usabilità effettuati. Quest’ultimi si sono dimostrati molto efficaci per comprendere se l’obiettivo posto all’inizio del lavoro di tesi sia stato effettivamente raggiunto: volevamo capire infatti se la modalità sviluppatore rappresenti un utile strumento per l’analisi del comportamento dell’agente. Lo scopo dei test era raccogliere valutazioni qualitative e soggettive sul sistema realizzato, attraverso una sessione di gioco libera e una guidata seguita da un questionario con domande sull’efficacia complessiva della UI, l’utilità delle sue singole parti e il generale *look-and-feel* di questa.

È giusto soffermarsi un momento sul campione di persone utilizzato. La fascia di utenti ideale sarebbe infatti quella dei game designers, ossia la figura professionale che si occupa della realizzazione del design di vari elementi del videogioco, dai nemici alla mappa di gioco, oltre alle varie meccaniche di

gameplay presenti (da qui il nome “*modalità sviluppatore*”). Queste figure normalmente presentano varie competenze tecniche in campo informatico compreso quello dell’intelligenza artificiale e dunque possono fare un pieno uso della modalità sviluppatore per comprendere le azioni dell’agente senza essere necessariamente esperti nel DRL. Queste tipo di lavori non sono però molto diffusi in Italia. Oltre a questo i test sono stati realizzati in un periodo particolare come quello della diffusione della pandemia COVID-19, che ha reso ulteriormente difficile trovare dei tester adatti – per maggior precisione i test sono stati effettuati a Novembre 2020. È stato dunque scelto un campione di 5 persone, appassionati di videogiochi e d’informatica, alcuni di questi studenti di Ingegneria Informatica dell’Università di Firenze. Gli interessi e il campo di studi dei tester li avvicina molto alla figura dei game designers, rendendoli adatti a valutare il sistema realizzato.

3.1 Fasi del test

A causa della pandemia il test è stato svolto a distanza, tramite videochiamata. Questo si è sviluppato in una prima fase di spiegazione verbale, in cui veniva illustrato il gioco DeepCrawl e le sue meccaniche; dopo di che sono state spiegate le tecniche di DRL utilizzate mantenendo un alto livello di astrazione, allo scopo di spiegare come l’agente prendesse le sue decisioni per comprendere meglio le informazioni di supporto utilizzate nella modalità sviluppatore. A queste spiegazioni è seguita una fase di gioco libero in cui i tester hanno svolto due partite per prendere confidenza con il sistema di gioco nella modalità classica.

Dopo questa fase è seguita un’altra partita, questa volta con un obiettivo più preciso: commentare ad alta voce le mosse del nemico, cercando di

capire la motivazione dietro queste. Finito l'episodio è stata attivata per la prima volta la modalità sviluppatore, che ripercorreva la stessa sequenza stato-azione effettuata in precedenza. Dopo aver spiegato i vari elementi dell'interfaccia grafica è stato chiesto ai tester di rivedere nella nuova modalità la partita appena conclusa, riprovando a spiegare le motivazioni delle mosse dell'agente. Lo scopo di questa task era permettere ai partecipanti di formulare una propria opinione sull'utilità della modalità sviluppatore e dei suoi componenti, paragonando la sua performance alla modalità utente. In generale, volevamo capire se la fiducia verso l'agente addestrato fosse aumentata rispetto all'utilizzo della sola modalità classica.

Dopo questa partita è seguita la fase finale del test, caratterizzata da un questionario realizzato su Google Form.

3.2 Domande

Il questionario era composto da 13 domande di tipo Single Ease Question (SeQ) con una scala che va da 1 a 7, dove 1 indica forte disaccordo e 7 fortemente d'accordo. Questa tipologia di domande è tipica dei test di usabilità di software.

Le domande possono essere divise in tre macroargomenti. Il primo riguardante l'efficacia complessiva della modalità sviluppatore, ossia se questa rappresenti uno strumento utile per valutare il comportamento dell'agente, paragonata alla modalità classica. Il secondo macroargomento riguardava la valutazione delle singole parti dell'interfaccia grafica, per comprendere quali fossero state le più apprezzate e utili nell'illustrare le informazioni sull'agente. Infine sono state formulate domande sul *look-and-feel* della UI. I risultati sono riassunti nella tabella 3.2.

N°	Domande	Media	σ
1	Durante la modalità classica percepisci che il nemico seguia una strategia.	6.2	0.83
2	Durante la modalità classica comprendevi spesso le motivazioni dietro le mosse del nemico.	5.0	1.22
3	L'interfaccia grafica in modalità sviluppatore è semplice e chiara.	5.0	1.87
4	L'interfaccia grafica in modalità sviluppatore è percepita come un'estensione naturale dell'interfaccia grafica in modalità classica.	5.0	1.22
5	Le informazioni aggiuntive della modalità sviluppatore occupano troppo spazio e confondono.	2.0	0.00
6	In modalità sviluppatore percepisci che il nemico seguia una strategia.	6.2	0.83
7	In modalità sviluppatore comprendevi spesso le motivazioni dietro le mosse del nemico.	6.2	0.44
8	Andare avanti e indietro nell'episodio era utile per valutare il comportamento del nemico.	6.6	0.89
9	L'highlight delle probabilità era utile per comprendere le motivazioni dietro le mosse del nemico.	6.6	0.54
10	Gli elementi dell'interfaccia grafica che seguono l'agente risultavano di difficile lettura (non chiari, troppo piccoli etc...).	4.8	1.64
11	Agent's rewards e la lista delle reward erano utili per valutare il comportamento nemico.	4.2	1.48
12	La barra dell'entropia non era utile per comprendere le motivazioni dietro le mosse del nemico.	2.2	1.64
13	La X di errore era utile per comprendere le motivazioni dietro le mosse del nemico.	5.8	1.09

Tabella 3.1: Tabella riassuntiva dei risultati del test SeQ effettuato per valutare la qualità generale del sistema di XAI.

3.3 Esito del test

In questa sezione daremo un'interpretazione ai risultati ottenuti tramite il questionario SeQ visibili in tabella 3.2. La discussione è suddivisa secondo gli stessi macroargomenti definiti nella sezione precedente.

3.3.1 Efficacia complessiva

Gli esiti del questionario mostrato in tabella confermano come la modalità sviluppatore abbia riscosso un buon successo nel suo obiettivo generale. Nelle risposte alle domande legate a questo argomento infatti traspare dai tester come questa modalità sia molto utile per comprendere le motivazioni dietro le mosse dell'agente, specie se a paragone con la modalità classica. È interessante notare come in entrambe le modalità l'agente è chiaramente percepito come intelligente e capace di formulare strategie, infatti le domande corrispondenti hanno ricevuto gli stessi voti. Questa è stata per noi una conferma della natura del problema: non una mancata percezione della strategia del nemico quanto un'assenza di strumenti per comprenderne il processo decisionale e di conseguenza anche una conferma dell'efficacia della sua soluzione. Tuttavia, il nostro sistema di XAI ha aumentato la fiducia nel sistema. Possiamo notare infatti dalle coppie di domande (1, 2) e (5, 6) come i tester, una volta assodato che i nemici fossero intelligenti, si siano sentiti molto più sicuri del perché di determinate scelte con la modalità sviluppatore, come dimostra la differenza della deviazione tra le domande 2 e 6.

3.3.2 Valutazione interfaccia

Le domande sui singoli elementi dell'interfaccia ci hanno permesso di comprendere quali strategie di comunicazione sono state più apprezzate per

mostrare le informazioni sull’agente e quali hanno maggiormente influenzato l’aumento di fiducia. In generale tutti gli elementi della nuova UI hanno ricevuto, con vari gradi, valutazioni positive. In particolare, la possibilità di muoversi avanti e indietro nell’episodio liberamente ha ricevuto alte valutazioni: probabilmente introdurre un elemento interattivo nell’interfaccia e dare la possibilità di rivedere più volte alcune azioni ha aiutato molto nell’obiettivo. Un altro artefatto grafico che ha ricevuto buoni voti è stato l’highlight della distribuzione di probabilità, grazie alla sua caratteristica di fornire informazioni chiare e molto esplicative sulle azioni dell’agente. Un elemento che ha ricevuto una valutazione più bassa è stato l’uso delle reward nella UI. Questo esito non era inaspettato data la natura sparsa della *reward function* di DeepCrawl che impedisce di usare singole reward come metro di valutazione delle azioni dell’agente, rendendo la lista delle discounted reward non utile in questo contesto. Tuttavia la somma di queste, mostrata nella schermata di Game Over, fornisce uno strumento quantitativo per la performance dell’agente e ciò gli ha meritato una media dei voti comunque positiva.

3.3.3 Look-and-feel

L’interpretazione delle risposte date alle domande sul *look-and-feel* della UI si è dimostrata più complessa a causa delle condizioni particolari in cui è stato svolto il test. Essendo in videochiamata infatti il gioco è stato trasmesso tramite condivisione schermo, diminuendone la qualità visiva complessiva. Nelle domande riguardo l’estetica o la chiarezza di elementi della UI più piccoli alcuni tester infatti hanno ammesso di aver preso in considerazione anche questo elemento, dando voti più alti, mentre altri hanno preferito non tenerlo in considerazione. Questo ha creato una disparità tra i voti, visibile nella alta deviazione standard delle risposte a queste domande,

ad esempio nelle domande 10 e 12. Tuttavia possiamo comunque fare alcune osservazioni. La nuova UI realizzata è stata percepita come un'estensione dell'interfaccia nella modalità classica, rispettandone il *look-and-feel*; inoltre gli utenti hanno giudicato le informazioni aggiuntive come non invasive, non riempiendo eccessivamente la schermata. Queste due valutazioni ci danno delle conferme sulla leggibilità delle informazioni e sulla capacità della UI di mostrarle apparentemente gradevole agli occhi, senza presentarsi come snaturata dall'interfaccia originale.

Abbiamo dunque concluso la spiegazione sul test di usabilità effettuato. Nel prossimo capitolo trarremo le conclusioni del lavoro complessivo svolto, riassumendo i traguardi raggiunti e definendo, visto il buon successo di questa tesi, alcuni lavori futuri che non è stato possibile portare a termine ma che potrebbero migliorare la qualità generale del sistema.

Capitolo 4

Conclusioni

In questa tesi è stato realizzato un sistema di Explainable AI allo scopo di facilitare la valutazione del comportamento di un agente intelligente, formato con tecniche di Deep Reinforcement Learning. Il contesto della tesi è il videogioco strategico a turni DeepCrawl [6] in cui tutti i personaggi non giocanti sono stati addestrati tramite tecniche di DRL. Il sistema di XAI proposto è una modalità sviluppatore, attivabile alla conclusione di una partita. Questa modalità prevede un replay della partita appena svolta, con un interfaccia grafica realizzata *ad-hoc* per mostrare informazioni aggiuntive sul comportamento dell'agente, facendo uso di vari artefatti grafici all'interno del gioco per mostrare questi dati nel modo più semplice e intuitivo possibile.

Abbiamo iniziato con qualche breve cenno teorico per comprendere appieno il lavoro svolto in questa tesi. Abbiamo spiegato i concetti di *machine learning*, in particolare riguardo al *reinforcement learning*, e di *explainable AI*. Siamo poi entrati nel dettaglio del gioco DeepCrawl, illustrandone le meccaniche di gameplay, le tecniche di DRL utilizzate e il pattern architettonurale *Entity-Component-System*.

Successivamente siamo entrati nel dettaglio del sistema di XAI, descriven-

do la UI realizzata nei suoi vari elementi, dalla modalità di navigazione dei turni alle informazioni rese disponibili sull'agente e come quest'ultime siano ottenute e gestite durante la partita. Di ognuno di questi elementi abbiamo mostrato le principali funzionalità e le loro motivazioni e responsabilità, approfondendo come sono stati realizzati utilizzando gli strumenti offerti del pattern ECS.

Abbiamo quindi effettuato un test di usabilità che ci ha permesso di capire la qualità finale del sistema proposto. I risultati del test hanno dimostrato che lo scopo della tesi è stato soddisfatto, in quanto è emerso che il sistema di XAI facilita molto la valutazione del comportamento dell'agente, grazie all'aumento di fiducia verso quest'ultimo che è stato mostrato durante il test. La modalità sviluppatore realizzata rappresenta un utile strumento per comprendere le motivazioni dietro le mosse dell'agente, attraverso una modalità replay liberamente navigabile e un'interfaccia grafica che presenta le varie informazioni su di esso e sui suoi processi decisionali in modo intuitivo, facilitandone la comprensione. Il sistema di DRL utilizzato, prima paragonabile a una *black box*, è ora più chiaro e semplice da interpretare nei suoi vari dati.

4.1 Lavori Futuri

Durante la fase di realizzazione del sistema e durante il test di usabilità sono emerse numerose idee e progetti legati alla modalità sviluppatore allo scopo di migliorare ancora la nostra comprensione dell'agente. Ne citiamo alcune:

- l'attuale sistema è molto efficace nel mostrare dati legati a una singola azione svolta, ma potrebbe dare maggiori informazioni su sequenze di mosse che l'agente compie; ciò potrebbe essere utile per darci maggiore

comprendere sulle strategie a lungo termine che è capace di formulare. Un modo per ottenere ciò sarebbe far simulare all’agente una sequenza di azioni indipendenti dalle scelte effettuate dal giocatore ed evidenziarle con le stesse strategie di comunicazione testate in questa interfaccia, come l’highlight delle caselle della scacchiera di gioco;

- l’entropia della distribuzione di probabilità delle azioni è risultato molto efficace come metro di valutazione dell’indecisione dell’agente, inoltre è il dato più soggetto a variazioni immediatamente dopo una mossa del player. Attualmente questo dato viene utilizzato nella barra presente sotto il nemico e in caso di grandi variazioni anche attraverso un pop-up, ma ci siamo chiesti, anche grazie ai commenti raccolti durante il test di usabilità, se questo non possa essere usato anche in altri modi. In particolare, registrando un numero sufficientemente grande di partite sarebbe possibile realizzare una heat map sulla scacchiera, che evidenzia quali azioni del player inneschino un’entropia più alta nell’agente, una soluzione simile a quella adottata in altri sistemi di XAI [1];
- l’interfaccia grafica realizzata mostra informazioni sull’agente in modo intuitivo, affidandosi più ad artefatti grafici che a rappresentazioni numeriche. Questo permette di valutare ogni mossa dell’agente a colpo d’occhio, rendendo l’analisi della partita più rapida. Tuttavia alcuni tester avrebbero apprezzato dati numerici in alcune occasioni. Potrebbero dunque essere realizzati strumenti per mostrare alcune informazioni anche in forma numerica, senza però appesantire o snaturare la UI. Ad esempio attraverso l’azione di *long press*, già presente nelle meccaniche di gioco per mostrare i dati di un giocatore o di un oggetto. Dati più verbosi potrebbero quindi apparire solo dopo questa azione, rimanendo

normalmente non visibili;

- questo intero sistema è basato totalmente sul videogioco DeepCrawl, tuttavia crediamo che l'utilizzo di questa modalità sviluppatore possa essere generalizzato ed utilizzato senza problemi anche in altri giochi strategici a turni. Ciò nonostante, attualmente non esistono altri giochi del genere open-source che utilizzino il DRL come strumento attivo di game design, per cui non è possibile testare il sistema su altri domini. Lasciamo quindi questa analisi ad un prossimo futuro.

Bibliografia

- [1] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klau-schen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLOS ONE*, 10(7):1–46, 07 2015.
- [2] Mathieu Le Ber. Serializabledictionary. <https://github.com/azixMcAze/Unity-SerializableDictionary>, 2018.
- [3] David Gunning, Mark Stefk, Jaesik Choi, Timothy Miller, Simone Stum-pf, and Guang-Zhong Yang. Xai—explainable artificial intelligence. *Science Robotics*, 4(37), 2019.
- [4] James Tittle Jeff Druce, Michael Harradon. Explainable artificial intel-ligence (xai) for increasing user trust in deep reinforcement learning dri-ven autonomous systems. In *Advances in Neural Information Processing Systems*, 2019.
- [5] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.
- [6] Alessandro Sestini, Alexander Kuhnle, and Andrew D. Bagdanov. Dee-pcrawl: Deep reinforcement learning for turn based strategy games. In *Proceedings of AIIDE Workshop on Experimental AI in Games*, 2019.

- [7] Unity technologies. Unity, 2020. Consultato: 29-Ottobre-2020.
- [8] J. Zhu, A. Liapis, S. Risi, R. Bidarra, and G. M. Youngblood. Explainable ai for designers: A human-centered perspective on mixed-initiative co-creation. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8, 2018.