ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Informatica

Curriculum: Sistemi e Reti

Training cognitivo adattativo mediante Reinforcement Learning

Relatore:
Prof
Mauro Gaspari

Presentata da:
Fabio Le Piane

Correlatore:
Dott.
Floriano Zini

II Sessione - Secondo appello
Anno Accademico 2017/18

Abstract

La sclerosi multipla (SM) è una malattia autoimmune che colpisce il sistema nervoso centrale causando un vasto spettro di alterazioni organiche e funzionali. In particolare, una rilevante percentuale di pazienti sviluppa deficit in differenti domini cognitivi. Essi possono essere causa di un grave degradamento della qualità della vita dei soggetti colpiti dalla malattia.
Per cercare di limitare la progressione di tali deficit, team di specialisti hanno ideato dei protocolli per la riabilitazione cognitiva atti a contenerli. Per effettuare le sedute di riabilitazione i pazienti devono recarsi nelle cliniche specializzate, necessitando dell'assistenza di personale qualificato quali psicologi, logopedisti, e altre figure sanitarie specializzate. Le sedute possono essere anche collettive, limitandosi però ad un massimo di una decina di pazienti trattati contemporaneamente, e gli esercizi si basano solitamente sulla scrittura su carta. A partire dagli inizi degli anni '90, però, si è iniziato un percorso verso la digitalizzazione di questo genere di esperienze e molti esercizi cartacei sono stati trasferiti su computer, permettendo dunque una partecipazione più vasta dei pazienti e la possibilità di svolgere la riabilitazione anche a domicilio.
Per procedere ulteriormente verso la digitalizzazione, un team multidisciplinare composto da ricercatori del DISI - Università di Bologna e da specialisti di vari centri italiani nel campo della riabilitazione per pazienti affetti da SM ha progettato un software, denominato MS-Rehab, il cui scopo è fornire alle strutture sanitarie un sistema completo e di facile utilizzo specifico per la riabilitazione dei deficit cognitivi tipici della malattia. MS-Rehab si presenta come una applicazione web che permette agli operatori clinici di gestire il percorso di riabilitazione dei pazienti; in particolare, permette di gestire il profilo cognitivo del paziente, di configurare i suoi esercizi riabilitativi e di visualizzare

una panoramica sulla sua situazione sanitaria/riabilitativa e le statistiche nel tempo delle sue performance negli esercizi svolti. Dal lato paziente, invece, il software permette lo svolgimento di numerosi esercizi per la riabilitazione nei tre domini cognitivi principali: attenzione, memoria e funzioni esecutive.
Questo lavoro di tesi si è concentrato sull'utilizzazione di metodi di Reinforcement Learning (RL) all'interno di MS-Rehab, allo scopo di implementare un meccanismo per permettere l'automatizzazione adattiva della difficoltà degli esercizi. Tale soluzione è inedita nell'ambito della riabilitazione cognitiva, e si prospetta come un notevole passo avanti nella direzione della personalizzazione del percorso riabilitativo. La peculiare caratteristica degli algoritmi di RL di apprendere direttamente dall'esperienza e non da esempi pre classificati permette una notevole personalizzazione, basandosi sul percorso del singolo paziente e non su schemi basati sui risultati generali collettivi, e in un ambito variabile come i deficit cognitivi nella SM. Tale personalizzazione è decisamente importante in quanto ogni tipologia di esercizio può essere trattata singolarmente, facendo variare i singoli parametri dell'esercizio in modo indipendente e non tramite schedulazioni predefinite come avviene attualmente nel sistema MS-rehab e in tutti i principali competitori. Allo scopo di verificare se la soluzione proposta permettesse un'esperienza riabilitativa pari o superiore a quella fornita dall'attuale versione di MS-rehab, questa è stata valutata tramite la somministrazione ad individui selezionati di test preliminari atti a valutare il loro livello nelle funzioni cognitive di attenzione e memoria, seguito poi da un periodo di allenamento su due esercizi (uno per la categoria attenzione, uno per la categoria memoria), per poi infine sostenere una nuova istanza del test iniziale al fine di evidenziare gli eventuali progressi ottenuti. Prima di tali test il software è stato allenato da altri utenti per ottenere una specie di entry level per i nuovi utenti.
A seguito di tale esperimento, si sono ottenuti dei risultati incoraggianti: la performance generale degli esercizi, infatti, è stata molto simile per entrambe le versioni, ma le prestazioni del test neuro-psicologico hanno evidenziato punteggi sensibilmente più alti per il gruppo che ha utilizzato la versione con RL; di tutti i risultati ottenuti si è confermata la rilevanza statistica tramite l'ausilio del t-test.

Il resto della tesi è strutturato come segue. Nel capitolo 1 viene analizzata l'importan-

za del training cognitivo e sono introdotte le potenzialità di un sistema computerizzato adattivo. Nel capitolo 2 sono analizzati alcuni sistemi per la riabilitazione cognitiva attualmente sul mercato ed è descritta più nello specifico la struttura di MS-Rehab; inoltre sono evidenziati i limiti di tutti questi sistemi e quali di questi limiti vengono superati dall'approccio basato su RL. Nel terzo capitolo viene spiegato più dettagliatamente e formalmente cosa si intende per RL, come funziona in generale e l'algoritmo di RL implementato in MS-Rehab. Nel capitolo 4 sono fornite le specifiche tecniche della soluzione, dalla progettazione alla versione finale implementata. Nel capitolo 5 viene descritto l'esperimento progettato per valutare il meccanismo adattativo di progressione delle difficoltà degli esercizi basato su RL e confrontarlo con quello attualmente utilizzato in MS-rehab. Il sesto capitolo, infine, presenta le conclusioni dell'autore e i possibili sviluppi futuri.

Indice

Capitolo 1

Introduzione

Per permettere una migliore comprensione dei temi trattati e del valore della soluzione proposta, questo capitolo sarà dedicato all'introduzione di alcuni concetti relativi alla sclerosi multipla (SM), alla riabilitazione cognitiva e la sua digitalizzazione, e al machine learning, con particolare attenzione alla branca del Reinforcement Learning (RL).

1.1 La sclerosi multipla

La SM è una malattia autoimmune cronica demielinizzante, che colpisce il sistema nervoso centrale causando un ampio spettro di segni e sintomi. Le cellule nervose trasmettono i segnali elettrici, definiti potenziali d'azione, attraverso lunghe fibre chiamate assoni, i quali sono ricoperti da una sostanza isolante: la guaina mielinica. Nella malattia, le difese immunitarie del paziente attaccano e danneggiano questa guaina e, quando ciò accade, gli assoni non sono più in grado di trasmettere efficacemente i segnali, con un crollo nella velocità di trasmissione dai 100 m/s in un individuo sano ai 5 m/s dei pazienti più colpiti. La malattia può manifestarsi con una vastissima gamma di sintomi neurologici e può progredire fino alla disabilità fisica e cognitiva. Al giorno d'oggi non esiste ancora una cura, e la terapia si limita a trattamenti farmacologici atti ad evitare nuovi attacchi e prevenire le disabilità. La prognosi è difficile da prevedere e dipende da molti fattori, mentre la speranza di vita è da 5 a 10 anni inferiore a quella della po-

polazione sana.
Poiché la malattia non colpisce una regione circoscritta del sistema nervoso e le aree colpite non hanno nessuna correlazione nota, il quadro clinico risulta estremamente variabile; inoltre, alcuni casi presentano uno sviluppo della malattia non lineare ma con periodi di maggiore intensità e di parziale miglioramento. In più, anche la sintomatologia è caratterizzata da una marcata variabilità, con sintomi quali affaticamento, problemi di coordinazione motoria, parestesie, dolore diffuso, depressione, e disturbi della vista, della parola, del controllo vescicale e intestinale, sessuali, e della sfera cognitiva. Tutto questo comporta un decorso della malattia difficilmente prevedibile, rendendo estremamente complicato lo sviluppo di iter terapeutici standardizzati. Nei soggetti affetti dalla patologia è alta la percentuale (oscilla fra il 40% e il 70%) di coloro che sviluppano deficit cognitivi in almeno uno dei tre domini: attenzione, memoria e funzioni esecutive [10].

1.2 Training cognitivo computerizzato

Come detto, la SM può causare una vasta serie di disturbi cognitivi. Nello specifico i disturbi coinvolgono più frequentemente la memoria, l'attenzione, la capacità di pianificazione e la velocità dei processi di elaborazione delle informazioni. Spesso i problemi si limitano a disturbi isolati di una di queste funzioni. Ad esempio, la maggior parte delle persone con SM è in grado di ricordare o di accumulare informazioni in maniera efficace, ma può trovare difficile ricordarle velocemente e in maniera adeguata. Data la variabilità dei sintomi riscontrabili caso per caso, si rivela necessario attuare uno studio preventivo e una pianificazione individuale di un processo riabilitativo efficace per limitare l'incidenza di questi deficit sulla qualità della vita dei malati.
L'approccio tradizionale alla riabilitazione cognitiva consiste in sedute di terapia ospedaliere ad opera di team composti in genere da logopedisti e psicologi, dove esercizi riabilitativi sono somministrati al paziente in forma cartacea, e il loro svolgimento è seguito personalmente dagli operatori sanitari, i quali possono eventualmente intervenire per variarne le caratteristiche, per chiedere al paziente di motivare le proprie scelte di risoluzione o per aiutare il paziente in caso non riesca in nessun modo a progredire nel-

lo svolgimento. A fronte di alcuni punti di forza (in primo luogo il totale controllo da parte del personale medico del percorso riabilitativo del paziente), tali soluzioni necessitano la presenza costante di personale specializzato e, di conseguenza, hanno grosse limitazioni quali l'esiguo numero di pazienti che possono essere seguiti in una singola sessione e/o in un dato lasso di tempo, la sporadicità dell'esperienza riabilitativa e il costo necessario per la formazione e il sostenimento del team specialistico.

Per questo motivo, negli anni, le equipe mediche hanno iniziato ad usare strumenti riabilitativi computerizzati, dove il paziente svolge le esercitazioni in ambiente ospedaliero sfruttando strumenti tecnologici quali computer e periferiche dedicate. In tal modo il trattamento medico diviene maggiormente interattivo ed è possibile l'automatizzazione dell'incremento della difficoltà degli esercizi, anche se, in definitiva, la sessione rimane guidata dall'operatore che può modificare manualmente la difficoltà suggerita dal software. Le più recenti ricerche in materia hanno rivelato che gli approcci computerizzati forniscono potenti strumenti al personale medico, permettendo di migliorare il processo riabilitativo [1] [15]. Un altro vantaggio della riabilitazione computerizzata è la possibilità di automatizzare la memorizzazione delle performance dei pazienti, velocizzando il processo e permettendo di automatizzare anche la creazione di grafici e tabelle, in modo che una personale medico abbia una visualizzazione chiara dell'andamento del processo riabilitativo; per questi motivi, la riabilitazione computerizzata è in rapida evoluzione e il suo impiego ha visto una crescita notevole [2] [3] [15].
Una componente critica dei sistemi riabilitativi computerizzati, che necessita quindi di notevoli studi e migliorie, è la loro capacità di adattarsi automaticamente allo stato cognitivo del singolo paziente, personalizzando completamente la sua esperienza; tale personalizzazione, in virtù di quanto detto in precedenza per la SM, è chiaramente funzionale ad una riabilitazione più efficace. Questo lavoro di tesi si è concentrato proprio sull'implementazione di un sistema che contribuisca a risolvere questa sfida.

1.3 Introduzione al Machine Learnin e al Reinforcement Learning

Con il termine *machine learning* si indica un insieme di metodi sviluppati in diversi ambiti (quali ad esempio la statistica computazionale, il riconoscimento di pattern, le reti neurali artificiali, il data mining, l'elaborazione di immagini, gli algoritmi adattivi) allo scopo di fornire ai calcolatori l'abilità di apprendere a svolgere determinati compiti senza essere stati esplicitamente programmati per la risoluzione degli stessi. Il machine learning è dunque strettamente legato alla teoria computazionale dell'apprendimento, e si concentra sullo studio di algoritmi il cui scopo è proprio l'apprendimento da un insieme di dati, costruendo induttivamente un modello basato sui campioni forniti. Al momento tale disciplina viene sfruttata in vari campi dell'informatica quando la progettazioni di algoritmi task-specific risulta impraticabile (ad esempio il filtraggio antispam delle email, il riconoscimento ottico dei caratteri, i motori di ricerca e la visione artificiale). A livello operativo, si potrebbe semplificare dicendo che si dice che un programma apprende se si apprezza un miglioramento delle prestazioni dopo lo aver risolto un'istanza (o più) di un problema. Una branca del machine learning denominata **Reinforcement Learning** (RL) punta alla realizzazione di sistemi capaci di apprendere direttamente dall'esperienza, ovvero **durante** l'interazione con l'ambiente in cui operano e non tramite l'analisi di database di esempi selezionati dagli sviluppatori. Questa è la tecnica utilizzata in questo lavoro di tesi, e verrà dunque approfondita maggiormente in seguito.

Capitolo 2

Stato dell'arte

In questo capitolo ci si concentrerà sull'adattività dei sistemi per la riabilitazione cognitiva attualmente disponibili e su come la soluzione implementata all'interno di MS-rehab si proponga di migliorare rispetto a tali sistemi. In particolare, gli ultimi paragrafi di questo capitolo saranno dedicati all'introduzione di una definizione di adattabilità e ad una discussione su come i software presi in esame non rispondano pienamente a una o più delle proprietà introdotte, evidenziando come, al contrario, la soluzione proposta punti invece a rispettarle tutte.

2.1 Panoramica sui sistemi esistenti

Alcuni dei più conosciuti software dedicati alla riabilitazione cognitiva sono [4]:

- Brainer [16]: software non specializzato per la riabilitazione di una malattia specifica, basato su una piattaforma web, sviluppato da Brainer srl. E' stato usato da diversi team e cliniche ospedaliere, soprattutto con pazienti affetti da schizofrenia. La difficoltà degli esercizi varia su tre livelli predefiniti (facile, medio, difficile) e viene impostata manualmente dal personale sanitario. Inoltre, per tutti gli esercizi è suggerita la presenza di una figura di supporto al malato, limitando fortemente l'autonomia del percorso riabilitativo.
- RehaCom [17]: software general disease avanzato, distribuito da Hasomed tramite appositi device USB per permettere l'installazione casalinga. Il software è

dunque utilizzabile su un normale personal computer, richiede però uno schermo da almeno 19 pollici e può essere integrato con una tastiera specifica. Propone 25 esercizi differenti, la cui difficoltà viene modulata tramite parametri propri di ogni esercizio. Il software propone dei suggerimenti per il setup della difficoltà basati sui risultati ottenuti dal paziente, ma è il terapista che deve manualmente confermare la configurazione o modificarla manualmente.

- Erica [18]: software general disease per la riabilitazione cognitiva sviluppato da Giunti O.S.; a differenza degli altri software, la sua principale limitazione consiste nell'impossibilità di svolgere sessioni casalinghe, poiché il software è disponibile solo nelle strutture cliniche. Inoltre, le sessioni devono essere configurate dal personale medico e non è prevista un'automazione dell'incremento di difficoltà.

- CogniPlus [19] [20]: strumento sviluppato in ambito scientifico per il trattamento dei disturbi delle funzioni cognitive sviluppato da Schuhfried. Offre 16 tipologie di esercizi, ognuna delle quali ha numerosi livelli di difficoltà, e l'avanzamento da un livello al successivo è basato sul raggiungimento di performance minime da parte dell'utente. Come per RehaCom, il software è distribuito tramite supporti USB.

- Neurab [21]: è un'applicazione non specifica per nessuna malattia, contenente esercizi mirati per riabilitare deficit specifici. In particolare l'architettura consiste in un sistema distribuito, accessibile al paziente tramite tablet appositamente configurati e forniti dall'azienda, tramite i quali il paziente ha accesso a 30 esercizi per vari domini cognitivi (attenzione, memoria, percezione, funzioni esecutive e linguaggio) le cui combinazioni di difficoltà possibili sono più di 10.000. È possibile scegliere la progressione automatica o seguire un percorso impostato dal personale sanitario. La difficoltà degli esercizi è ottenuta tramite la variazione (sia manuale che automatica) dei parametri che definiscono gli esercizi stessi.

- Padua Rehabilitation Tool [22] software è stato sviluppato con un'interfaccia e stimoli appositamente semplificati in modo da consentire un utilizzo più immediato. Grazie alla varietà degli esercizi è possibile sottoporre ad ogni paziente training

diversi. Il software è quindi utilizzabile con una varietà di pazienti a prescindere dalla patologia neurologica acquisita: è possibile definire programmi di intervento specificatamente focalizzati sulle necessità di ogni paziente. La difficoltà dei vari compiti è strutturata secondo livelli di crescente complessità, in modo che il paziente abbia la possibilità di iniziare con compiti semplici sperimentando un senso di adeguatezza, riducendo il senso di frustrazione e supportando la motivazione.

- HappyNeuron Pro [23]: è una suite che comprende vari tools per il trattamento e la diagnosi dei disturbi cognitivi di varia natura. Per quanto riguarda il training cognitivo offre 38 differenti task, divisi fra i vari domini cognitivi, ognuno dei quali può essere affrontato a vari livelli di difficoltà tramite la configurazione di vari parametri propri dell'esercizio. La progressione automatica segue pattern predefiniti ma può essere settata manualmente dal personale medico.

2.2 MS-rehab

MS-rehab è un software specializzato per la riabilitazione cognitiva dei malati di sclerosi multipla. Basato su una architettura web che utilizza Apache Tomcat e tecnologie quali Java Servlet e Java Servlet Pages (JSP), le funzionalità principali del sistema (ovvero le sedute riabilitative) sono utilizzabili solamente tramite computer o tablet, mentre alcuni servizi minori quali il monitoraggio dei pazienti sono accessibili anche tramite smartphone. Le interfacce grafiche sono realizzate tramite il noto framework Bootstrap, e quindi tramite l'utilizzo di tecnologie quali HTML, CSS e Javascript. Il server si appoggia ad una macchina virtuale con sistema operativo Ubuntu 16.04, e la persistenza dei dati è affidata ad un database SQLite, dove vengono conservate informazioni riguardanti gli utenti registrati e le performance da essi conseguite. Tale struttura permette a più centri di riabilitazione di accedere contemporaneamente all'applicazione, e in caso di necessità permettono la migrazione o la replicazione del sistema su macchine supplementari. L'architettura appena descritta è illustrata nella figura 2.1
L'applicazione presenta due sezioni separate: una per il personale sanitario (utilizzata per visualizzare le informazioni personali dei pazienti e per assegnare gli esercizi di riabilitazione) e una per i pazienti, dove gli stessi possono accedere alla sessione riabi-

litativa assegnata.
In particolare, ci si concentrerà sulla seconda poiché è quella coinvolta dalle modifiche apportate durante questo lavoro di tesi.

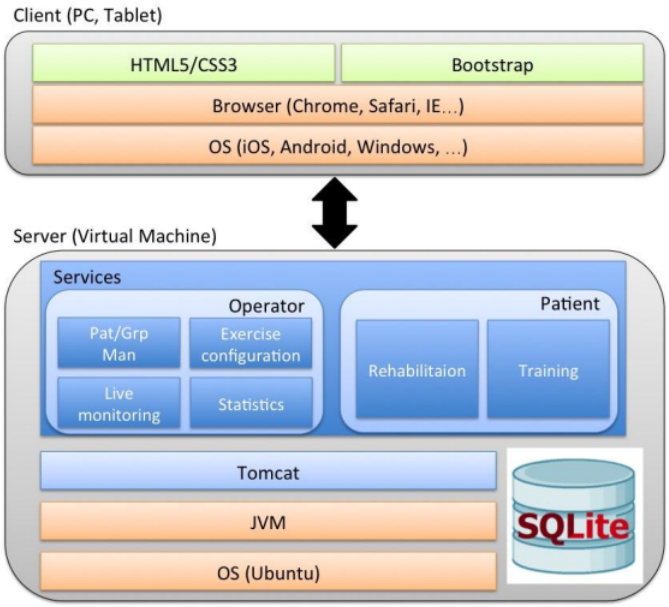

Figura 2.1: Architettura di MS-rehab.

2.2.1 Sezione pazienti ed esercizi disponibili

La sezione dedicata ai pazienti presenta due sezioni: una per affrontare gli esercizi di riabilitazione assegnati e una per cimentarsi con degli esercizi di prova su cui fare pratica prima di affrontare quelli effettivamente assegnati.
Le tipologie di esercizi presenti sono più di venti, suddivisi in esercizi peculiari per ogni funzione cognitiva:

- Attenzione: è definita come l'abilità di compiere un'analisi selettiva degli input, ovvero un'analisi delle informazioni che derivano dal mondo esterno. Sono presenti esercizi per la riabilitazione dei tre tipi di attenzione: attenzione selettiva, attenzione alternata e attenzione divisa. Tutti questi esercizi possono essere svolti

con vari tipi di stimoli: immagini (frutta, verdura, animali, scacchi), volti e simboli di orientamento (frecce, punti cardinali).

- Memoria: è la funzione che permette la codifica, la conservazione e il richiamo dell informazioni, anche a distanza di tempo dal momento del loro ottenimento. All'interno di MS-rehab sono presenti vari esercizi per allenare la memoria, quali esercizi di riconoscimento di immagini, volti e simboli di orientamento, esercizi per la memoria visuo-spaziale (anch'essi con immagini, volti o simboli di orientamento), esercizi per la memoria di lavoro (con le stesse varietà di stimoli) e un esercizio per l'associazione volti-nomi.

- Funzioni esecutive: sono un processo cognitivo di livello più alto che permettono il controllo degli altri processi per permettere il raggiungimento di un obbiettivo. MS-rehab fornisce due tipologie di esercizi dedicati alle funzioni esecutive: uno per allenare la funzione di controllo-inibizione, uno per allenare la funzione di pianificazione.

Ogni esercizio possiede dei parametri peculiari che lo definiscono e che ne regolano la difficoltà. Ad esempio, tali parametri possono essere il numero di elementi che appariranno durante l'esercizio, o il numero di figure diverse che andranno memorizzate oppure ancora il tempo di permanenza di uno stimolo a schermo prima di essere sostituito dal successivo e così via. Tutti questi parametri hanno tre livelli di difficoltà, dove ad ogni livello corrispondono dei differenti valori specifici per ogni parametro (ad esempio, il parametro "colore" di molti esercizi può assumere il valore "color" al livello 1, il valore "omogeneous" al livello 2 e il valore "black and white" al livello 3). La difficoltà degli esercizi è ottenuta tramite il valore cumulativo di tali parametri; ad esempio, un esercizio con 4 parametri, di cui due a livello 2 e due a livello 3, avrà un livello di difficoltà totale pari a 10. Inoltre, per agevolare la regolazione delle sessioni da parte del personale medico, è stata inserita una suddivisione fra esercizi facili, medi e difficili.
La progressione della difficoltà può essere settata manualmente dal personale sanitario oppure affidata ad un incremento automatico. Tale sistema di automazione funziona

nel modo seguente: ogni qual volta il paziente supera due esercizi consecutivamente[1], il software sceglierà casualmente un parametro fra quelli propri dell'esercizio in questione e lo porterà al successivo valore possibile (ad esempio: diminuirà il tempo di permanenza di uno stimolo[2]). Quando verranno completati consecutivamente due esercizi di difficoltà massima, il paziente verrà riportato alla home e quel dato esercizio sarà considerato completato; spetterà al personale sanitario decidere se il percorso riabilitativo è stato completato con successo o se al paziente dovranno essere assegnati ulteriori esercizi.

2.3 Limiti dei sistemi esistenti

In via preliminare, si propone una definizione di sistema adattivo; si definisce adattivo un software riabilitativo che rispetta le seguenti proprietà:

1. Il percorso riabilitativo proposto è personalizzato per ogni paziente.

2. Il percorso riabilitativo non viene stabilito a priori ma è continuamente regolato e raffinato basandosi sulle effettive performance del paziente ottenute durante la riabilitazione stessa.

3. Il percorso riabilitativo non prevede solo l'aumento ma anche il mantenimento o la riduzione della difficoltà durante l'intero iter.

4. Fatto salvo per un'eventuale fase di assestamento, il livello di difficoltà di ogni esercizio riabilitativo deve rivelarsi allenante per il paziente (ovvero né troppo alto né troppo basso) durante l'intero percorso riabilitativo.

Nonostante ognuno dei sistemi sopracitati abbia i propri punti di forza e di debolezza, ciò che va chiarito è che nessuno di essi può definirsi puramente adattivo rispetto alla definizione appena introdotta. Per quanto la proprietà **1** sia rispettata da molti dei software

[1] Un esercizio è considerato superato se terminato con performance uguale o superiore all'80% del punteggio massimo

[2] Da qui in avanti, la versione di MS-rehab con questo sistema di progressione della difficoltà verrà denominata "versione baseline"

sovramenzionati (il percorso è infatti personalizzabile per ogni paziente, al quale potrà essere assegnato un livello d'ingresso congruo alla sua condizione neuro-psicologica che aumenterà solo in base alle prestazioni ottenute) la progressione automatica, anche nei casi con il maggior numero di gradi di difficoltà, funziona tramite un incremento basato su pattern predefiniti studiati dagli sviluppatori oppure tramite l'intervento del personale sanitario. Inoltre, tutti gli schemi si riconducono in definitiva al superamento da parte del paziente di una certa soglia di prestazione o al più del mantenimento di tale soglia per un certo numero di esercizi (come, per esempio, MS-rehab) per poi incrementare in maniera predefinita la difficoltà generale dell'esercizio (ad esempio aumentando il livello di un certo numero di parametri dell'esercizio scelti casualmente fra quelli possibili). L'avanzamento, dunque, non avviene basandosi sull'effettiva abilità del paziente rispetto ai singoli parametri; tale approccio limita fortemente le possibili progressioni effettivamente realizzabili, creando di fatto delle progressioni parzialmente predefinite e contraddicendo dunque la proprietà **2**. Facendo l'esempio dell'attuale sistema di progressione di MS-rehab, questo si basa sul livello generale dell'esercizio, dove tale livello comporta solo la presenza di un certo numero di parametri di un determinato livello e di un certo numero di parametri con un altro livello, senza tenere conto di quali di questi abbiano effettivamente arrecato maggiore difficoltà al paziente o viceversa (per una trattazione più approfondita di tale sistema si rimanda al capitolo sull'implementazione della soluzione proposta). In particolare, il primo esercizio di una nuova seduta viene generato in maniera casuale basandosi sul livello complessivo raggiunto nell'ultimo esercizio svolto prima di interrompere l'ultima seduta e non sul valore dei singoli parametri. In tal modo, ad esempio, il paziente potrebbe aver concluso la precedente esperienza con un esercizio in cui il parametro 1 era di livello 3 e il parametro 2 era di livello 2 e ritrovarsi nella nuova sessione a dover affrontare un esercizio in cui il parametro 1 ha livello 2 e il parametro 2 ha livello 3, falsando potenzialmente la performance ottenuta durante l'esercizio stesso.

Inoltre nessuno dei software menzionati prevede la possibilità di diminuire automaticamente la difficoltà degli esercizi, delegando tale possibilità alla sola azione del personale sanitario, vanificando di conseguenza banalmente la proprietà **3**.

Infine, basando la progressione degli esercizi sul superamento di una certa soglia di per-

formance per un certo numero di esercizi consecutivi, tutti questi software rispettano solo parzialmente la proprietà **4**, poiché un esercizio potrebbe rivelarsi non allenante ma tale sistema di progressione obbligherebbe il paziente a svolgerne almeno un secondo alla medesima difficoltà prima di poter accedere al livello successivo (o, se necessario, a quello precedente), rallentando dunque, anche in maniera considerevole, il processo di adattamento.

2.4 Reinforcement Learning e applicazioni mediche

Il Reinforcement Learning è una branca del Machine Learning che negli ultimi anni ha visto un notevole sviluppo tecnico e che ha raggiunto risultati sorprendenti, in particolare nelle applicazioni del cosiddetto Deep Reinforcement Learning (basti pensare al caso specifico di AlphaGo [5], assurto agli onori della cronaca tra il 2015 e il 2016, il cui apprendimento era basato principalmente su algoritmi di Reinforcement Learning).
Particolare attenzione meritano le applicazioni del Reinforcement Learning in campo medico. Un traguardo ottenuto in questo campo sono, ad esempio, è la realizzazione di un agente capace di assistere i medici nell'iter diagnostico [9] (senza specializzazione in una particolare malattia). Un altro esempio degno di nota è lo sviluppo di sistemi capaci di assistere i medici nell'interpretazione di immagini, permettendo di ridurre il rischio di errori di interpretazione [8].
I notevoli risultati ottenuti da questo genere di software indicano chiaramente come l'applicazione di tecnologie di RL al campo dell'informatica medica sarà un campo al centro dell'attenzione nei prossimi anni; inoltre, le particolari sfide tecnologiche offerte da questo campo (è facile immaginare come, ad esempio, sia cruciale poter assicurare la sicurezza nell'utilizzo di queste applicazioni al fine di evitare potenziali conseguenze dannose a carico dei pazienti) potranno potenzialmente permettere un avanzamento del reinforcement learning tout court [7].
Sebbene al momento non ci siamo notizie di studi che applicano il RL alla riabilitazione cognitiva, va sottolineato come, invece, altre branche del machine learning abbiano ottenuto risultati promettenti sia nel campo della prognosi delle condizioni neuropsicologiche di pazienti affetti da degenerazione cognitiva [24] sia nel campo della

riabilitazione cognitiva [25].

2.4.1 Adattività basata su Reinforcement Learning

Tramite l'utilizzo di tecniche di RL è possibile ottenere un sistema in grado di regolare la progressione della difficoltà nella maniera più fine possibile, aggirando il concetto di difficoltà tipico dei sistemi illustrati in precedenza e realizzando un concetto di difficoltà più elastico, in cui ogni parametro viene considerato nella sua singolarità e gestito in maniera indipendente dagli altri. In questo modo è facile intuire come l'esperienza riabilitativa risulterebbe notevolmente più personalizzata (proprietà **1**), grazie ad una progressione della difficoltà basata sulla performance ottenuta passo passo dall'utente. Tale progressione è realizzabile considerando ogni volta come l'aumento di un singolo parametro influenza il rendimento (proprietà **2**), fornendo al sistema anche la possibilità di decrementare la difficoltà dell'esercizio (proprietà **3**) o di aumentare più rapidamente la difficoltà degli esercizi, raggiungendo più velocemente e mantenendo un livello di difficoltà allenante per il paziente (proprietà **4**).
Con questa tecnica si punta a far sì che la difficoltà degli esercizi si avvicini il più possibile alla soglia massima allenante dell'utente, aumentando i parametri meno problematici e rallentando invece la progressione di quelli che risultano più negativi sui risultati.

Capitolo 3

Reinforcement learning

Si procederà ora ad introdurre più nello specifico il Reinforcement Learning[1]. Innanzitutto verranno fornite le definizioni e le nozioni teoriche di base; in seguito saranno discusse alcune caratteristiche che permettono a tali algoritmi di adattarsi a vari tipi di contesto. Infine, verranno mostrate le due principali tipologie di algoritmo e in particolare verrà illustrato l'algoritmo effettivamente integrato all'interno di MS-rehab.

3.1 Definizioni e formalizzazione

La base di un sistema di Reinforcement Learning è composta di due elementi: l'**agente** e l'**ambiente**. Per **agente** si intende una qualunque entità in grado di percepire, tramite *sensori*, l'ambiente circostante e di eseguire delle azioni su di esso, tramite *attuatori*; l'**ambiente** comprende tutto ciò che è *al di fuori* dell'agente, ed è rappresentato tramite una struttura detta **stato**, che raccoglie tutti i dati necessari a descrivere all'agente il mondo in cui è immerso. Questi due elementi interagiscono continuamente in un ciclo infinito, dove l'agente compie delle azioni e l'ambiente risponde fornendo il suo nuovo stato. Lo scopo dell'agente è quello di imparare ad eseguire un determinato compito, e l'ambiente, oltre alla stato, restituisce all'agente un feedback sul livello di abilità raggiunto nel compito.

[1]Le nozioni trattate in questo capitolo sono tratte dal libro *Reinforcement Learning: An Introduction* [6].

Più nello specifico, è possibile individuare quattro ulteriori elementi fondamentali: la **Policy**, la **Reward**, la **Value Function** e il **Modello**.

- La **Policy**, il nucleo dell'agente, consiste in una mappatura dagli stati percepiti alle azioni da eseguire quando l'agente si trova nei suddetti stati; fondamentalmente, la policy descrive il comportamento dell'agente.

- La **Reward** consiste in un segnale numerico inviato dall'ambiente all'agente, tramite il quale viene comunicata la qualità dell'azione appena compiuta rispetto all'obiettivo da raggiungere. L'agente punterà alla massimizzazione nel lungo termine della reward totale ricevuta. Essa dunque indica all'agente quali sono gli eventi positivi e quelli negativi, e una sua accurata definizione da parte degli sviluppatori del sistema di RL e' fondamentale per ottenere un agente in grado di imparare proficuamenete il compito assegnatogli.
  Gli elementi introdotti finora e il rapporto fra essi sono rappresentati in figura 3.1.

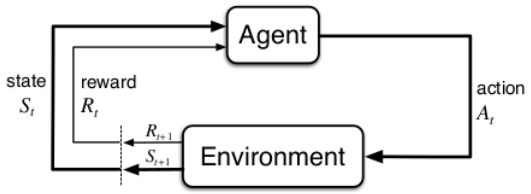

Figura 3.1: Schema del ciclo del Reinforcement Learning: ogni volta che l'agente compie un'azione, l'ambiente risponde fornendo il nuovo stato e la reward ottenuta

- La **Value Function**: è una funzione che associa ad ogni stato un *valore*, dove per *valore di uno stato* si intende la reward totale che l'agente dovrebbe aspettarsi di ricevere dopo la transizione che lo ha condotto allo stato stesso.

- Il **Modello**, infine, serve a simulare il comportamento dell'ambiente, ad esempio ricevendo in input uno stato e un'azione e restituendo in output lo stato successivo e la reward ricevuta. Il modello non è sempre necessario, esistono infatti i cosiddetti metodi *model-based*, basati sul planning, e i metodi *model-free*, più

esplicitamente trial-and-error; esistono inoltre sistemi che contemporaneamente apprendono tramite trial-and-error e apprendono un modello dell'ambiente da utilizzare per il planning.

La particolarità del Reinforcement Learning rispetto agli altri due paradigmi più famosi del machine learning, ovvero l'apprendimento supervisionato e l'apprendimento non supervisionato, consiste nella capacità dell'agente di apprendere direttamente dall'esperienza diretta (sia essa reale o simulata) e dunque *durante* l'interazione con l'ambiente stesso e non tramite l'estrapolazione di strutture generali da raccolte di dati preesistenti. Tale caratteristica permette di apprendere e agire contemporaneamente e in real time. Altra peculiarità del Reinforcement Learning è quella di permettere lo sviluppo di agenti goal-oriented che si concentrano sulla risoluzione di task direttamente nella loro interezza, senza la necessità di doverli scomporre in sotto-task più semplici come avviene nel caso dell'apprendimento supervisionato o non-supervisionato.
La parte fondamentale della teoria alla base del RL è ereditata dalla teoria della risoluzione dei **problemi decisionali di Markov (MDP, dall'inglese Markov Decision Process)**; questi nascono proprio allo scopo di offrire un framework per la modellizzazione del processo decisionale in ambienti parzialmente casuali. Più precisamente, un processo decisionale di Markov è un processo di controllo stocastico a tempo discreto. Formalmente, un MDP è definito da:

- Uno spazio degli stati S.
- Uno spazio delle azioni A che possono essere intraprese; si definisce poi A_s come l'insieme delle azioni che è possibile intraprendere nello stato s.
- Le probabilità di transizione $p(s'|s,a)$ che definiscono le dinamiche one-step dell'ambiente, ovvero, dato uno stato s e un'azione a, al tempo t, definiscono la probabilità di raggiungere ogni possibile stato successivo s' : $P(s'|s,a) = Pr(s_{t+1} = s'|s_t = s, a_t = a)$.
- Il valore atteso della reward $r(s,a,s') = \mathbb{E}(r_{t+1}|s_t = s, a_t = a, s_{t+1} = s')$, dove $\mathbb{E}$ rappresenta il valore atteso o previsione, r_{t+1} è il valore della reward $t+1$-esima sapendo che la transizione avvenuta è quella dallo stato s_t allo stato s_{t+1} tramite l'azione a_t.

- Il fattore di sconto (discount) $\gamma \in [0, 1]$, che cattura l'importante differenza tra le ricompense future (future rewards) e le ricompense presenti (present rewards). In pratica tale parametro serve a specificare all'agente quanta importanza dare alle reward future rispetto a quella immediatamente successiva: se γ è uguale a 0 l'agente è "miope", ovvero si concentra solo sulla massimizzazione della reward immediata; più γ si avvicina a 1 e più l'agente si concentra sulla massimizzazione del ritorno cumulativo, tenendo in maggiore conto anche le reward ottenute in istanti molto lontani nella computazione.

È facile dunque capire come una run di un problema di RL si possa riassumere come una sequenza (potenzialmente infinita) del tipo

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

Quindi, ponendo S_t ed R_t rispettivamente lo stato e la reward ottenuta al tempo t, queste hanno una ben precisa distribuzione di probabilità dipendente solo dalla precedente coppia stato-azione (S_{t-1}, A_{t-1}) definita dalla seguente formula:

$$p(s', r|s, a) = Pr[S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a]$$

Per ogni $s, s' \in S, r \in R, a \in A(s)$. È facile verificare come questa sia una distribuzione di probabilità. Da questa è dunque possibile ottenere la probabilità di transizione di stato (ovvero la probabilità di passare dallo stato s allo stato s' compiendo l'azione a), che con abuso di notazione risulta essere

$$p(s'|s, a) = Pr[S_t = s'|S_{t-1} = s, A_{t-1} = a] = \sum_{r \in R} p(s', r|s, a)$$

Ed estendendo il medesimo ragionamento è possibile ottenere la funzione per calcolare la reward prevista per la coppia stato-azione

$$r(s, a) = \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] = \sum_{r \in R} r \sum_{s' \in S} p(s', r|s, a)$$

Per trattare efficacemente il problema dell'apprendimento tramite interazione, è necessaria poi un'ultima definizione, quella dell'*expected return* (denotato G_t). Concettualmente, questo indica la reward cumulativa ottenuta durante l'intero esperimento, e nel

caso più semplice si limita ad essere la somma delle reward ottenute ad ogni passo della computazione. È facile capire come tale approccio possa funzionare solo in quei casi dove l'esperimento abbia un concetto di fine ben definito e che prima o poi avvenga (ad esempio una partita a scacchi), mentre la sua applicabilità in contesti senza termine risulti minato dal continuo accumularsi di reward senza poter mai avere la certezza di star intraprendendo il cammino migliore. Per tale motivo, si utilizza il fattore di sconto γ introdotto in precedenza, dando la seguente definizione di expected return:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \\ &= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \end{aligned}$$

Concettualmente, questa definizione indica il fatto che il peso da dare alle reward attese cala al crescere della distanza dallo stato attuale, ovvero che le reward più rilevanti sono quelli ottenibili in pochi passi, mentre le reward più distanti devono essere tenute in minore considerazione. Grazie alle definizioni finora introdotte è possibile definire la value function della policy π, ovvero quella funzione che assegna ad ogni stato un valore che ne quantifica la bontà assumendo che l'agente compia azioni basandosi sulla policy π; formalmente, questa è definita come segue:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s], \forall s \in S$$

A grandi linee, lo scopo finale di un algoritmo di RL è quello di trovare quella policy π^* tale da garantire la massima reward totale sul lungo periodo. Tale policy è detta ottimale, e formalmente è quella policy tale che $v_*(s) = \max v_\pi(s) \forall s \in S$. Tornando all'equazione (3.2), possiamo proseguire la computazione come segue:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma E_\pi[G_{t+1}|S_{t+1} = s']] = \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')], \forall s \in S \end{aligned}$$

Applicando dunque la definizione di v^* alla precedente, si ottiene che

$$v^*(s) = \max \mathbb{E}[R_{t+1} + \gamma v^*(S_{t+1}) | S_t = s, A_t = a] = \tag{3.1}$$

$$= \max_a \sum_{s',r} p(s', r|s, a) r + \gamma v^*(s') \tag{3.2}$$

Le due forme (3.1) e (3.2) sono due forme delle cosiddette **equazioni di ottimalità di Bellman**. Una volta ottenute le equazioni di Bellman per v^* è facile ottenere la policy π^*: essa, infatti, è semplicemente la policy che, per ogni stato s, assegnerà probabilità diversa da zero solo a quelle azioni che massimizzano l'equazione di Bellman. Una formulazione equivalente ma che rende meglio il concetto, è che la policy π^* è quella policy che ha un comportamento *greedy*[2] rispetto alla value function v^*. Si può dunque concludere che lo scopo di un algoritmo di RL è proprio quello di determinare v^*.
Ovviamente, tralasciando gli esempi più semplici, calcolare effettivamente v^* e π^* tramite le equazioni di Bellman avrebbe un costo computazionale semplicemente insostenibile, il che porta a dover trovare dei metodi di approssimazione tali da poter calcolare delle policy sub-ottime ma che allo stesso tempo raggiungano performance sufficientemente vicine a quelle ottime. L'idea principale su cui si basano tutti gli algoritmi che implementano queste metodologie, anch'essa mutuata dalla teoria della programmazione dinamica, è quella del loop denominato **policy iteration**. Tale loop è suddiviso in due sotto-componenti che si alternano: la **policy evaluation** (anche detta **prediction problem**) e la **policy improvement**. La prima consiste nel calcolare la value function v_π corrispondente alla attuale policy π: questa si basa sull'idea che, avendo una sequenza di policy approssimate $v_0, v_1, v_2, \ldots$ (dove v_0 è scelta arbitrariamente) allora è possibile determinare il passo $k+1$-esimo di questa sequenza partendo dal passo k-esimo come segue (richiamando le definizioni precedenti di v_π)

$$v_{k+1}(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s] =$$

$$= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) r + \gamma v_k(s')$$

ed è facile mostrare come questa sequenza converga asintoticamente a v_π.
La seconda parte, invece, consiste nel chiedersi se, scegliendo a un certo punto della

[2] Dove per policy greedy si intende una policy che sceglie sempre l'azione con valore più alto rispetto alla value function.

computazione un'azione a diversa da quella che verrebbe scelta seguendo la policy π, la reward totale ottenuta in questo modo sia superiore a quella che si sarebbe ottenuta seguendo π, ovvero se $v_{\pi'}(s) \geq v_{\pi}(s)$, dove π' è quella policy identica alla precedente policy π tranne che $\pi'(s) \neq \pi(s)$; possiamo dunque notare come un passo di policy improvement restituisca sempre una policy migliore o equivalente alla policy precedente. Applicando queste proprietà a quelle proprie delle equazioni di ottimalità di Bellman è possibile concludere che se $v_{\pi} = v_{\pi'}$ allora sia π che π' devono essere delle policy ottime.
È dunque facile concludere come sia possibile tendere asintoticamente alla policy e alla value function ottime alternando un passo di policy evaluation e uno di policy improvement, generando di volta in volta una policy migliore e una value function migliore sfruttando l'una per generare l'altra e viceversa. Tale schema è riassunto dalla figura 3.2.

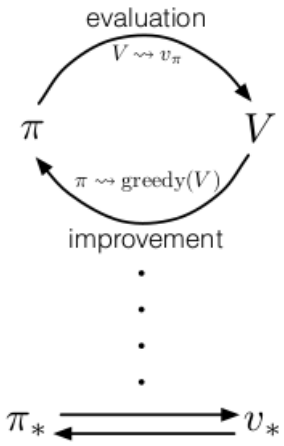

Figura 3.2: Schema della policy improvement

Per assicurare che tale meccanismo funzioni, però, bisogna garantire che l'agente continui a esplorare stati in cui non si è mai (o quasi) venuto a trovare, ovvero valori per cui la policy non è mai (o quasi) stata valutata; allo stesso tempo, però, l'esplorazione non deve mai essere così frequente da rischiare di degradare eccessivamente la performance, assicurandosi dunque di sfruttare (*exploite*) sufficientemente la policy ot-

tenuta. Il bilanciamento fra queste due componenti, definito appunto come **exploration-exploitation dilemma**, è una delle problematiche principali durante l'implementazione di sistemi basati su RL e non esistono soluzioni esatte alla sua risoluzione, la quale è affidata alle capacità degli sviluppatori.
Tutti i risultati introdotti fino a questo punto possono essere estesi anche per le cosiddette action-value functions; le action-value functions differiscono dalla value functions per il fatto che associano un valore non al solo stato ma alla coppia ordinata (stato, azione). Di solito, le action-value function sono denotate con $Q(S_t, A_t)$.
Come detto, questo framework è l'idea di base sottostante a tutti i sistemi di RL, anche quelli più evoluti. Ciò che differenzia i sistemi basilari da quelli più complessi sono le tecniche attuate per approssimare questo ciclo fondamentale nel modo più funzionale alla risoluzione degli specifici problemi che dovranno affrontare; alcune delle idee principali sono descritte brevemente nel paragrafo successivo.

3.2 Le tipologie di Reinforcement Learning

Quando si intende realizzare un sistema di RL è necessario capire in via preliminare quali sono le proprietà del proprio problema specifico in modo da poter restringere il campo degli algoritmi di RL esistenti, scegliendo quello più adatto ad apprendere una policy efficiente per risolvere il task.
In primo luogo è necessario domandarsi se lo spazio degli stati e quello delle azioni sono abbastanza piccoli da permettere di rappresentare la value function in forma di tavole (o matrici) o se la loro dimensione intrinseca è tale da necessitare di introdurre meccanismi di approssimazione. Nel primo caso si parla di metodi di risoluzione tabulare (dove appunto la value function appresa può essere rappresentata come una matrice, dove le righe corrispondono alle azioni che l'agente può compiere, le colonne agli stati possibili e nelle celle è contenuto l'expected return ottenibile compiendo l'azione rappresentata dalla riga quando ci si trova nello stato rappresentato dalla colonna) mentre nel secondo si parla di metodi di risoluzione approssimata. La differenza fondamentale fra i due casi sta nel fatto che a livello generale i primi consentono, a patto di disporre di un tempo sufficiente per addestrare l'agente, di ottenere una soluzione esatta del problema, ovvero

di ottenere la policy che permette di compiere l'azione migliore in ogni stato possibile, ma per fare questo l'agente deve avere la possibilità di trovarsi in ogni stato possibile almeno una volta nel corso della propria vita (ma, al crescere del numero di combinazioni possibili, non è detto che tale proprietà sia garantibile); al contrario, i secondi si basano sull'introduzione di funzioni per permettere all'agente di prendere una decisione anche quando questo viene a trovarsi in stati a lui sconosciuti basandosi sulle conoscenze da lui accumulate venendosi a trovare in stati simili a quello attuale, rendendo però di fatto impossibile raggiungere la policy migliore possibile. È dunque facile capire come sia preferibile scegliere i metodi tabulari , ma è altrettanto ovvio concludere che, nella maggior parte dei casi reali, questi non siano praticabili, e che ci si debba quindi concentrare sull'implementazione di metodi di approssimazione sempre più efficienti ed affidabili.
La seconda caratteristica fondamentale da considerare è la natura episodica o meno del problema che si sta cercando di risolvere tramite RL. Un sistema si definisce episodico quando l'interazione fra agente e ambiente può essere suddivisa naturalmente in intervalli di tempo, detti appunto episodi, per i quali è possibile individuare un tempo di inizio e uno di fine; al contrario, quando tale divisione non è possibile, il task è definito non-episodico. È facile capire come l'apprendimento risulti differente nei due casi e come sia necessario dunque attuare strategie differenti. Un tipico esempio di task episodico è quello degli scacchi, mentre un esempio di task non-episodico è un qualunque tipo di process-control.
Una scelta implementativa che merita una menzione è quella fra metodi on-policy e off-policy. Per on-policy si intendono quei metodi in cui l'agente apprende una policy sulla quale si basa anche per scegliere quale azione compiere, ovvero in cui effettivamente i processi di azione e apprendimento sono completamente collegati e si influenzano mutualmente; per metodi off-policy, invece, si intendono quei metodi in cui l'agente dispone di due policy differenti: una utilizzata per decidere quale azione compiere (tipicamente statica e non modificata durante la run) e una modificata in base alle reward ottenute. Se il primo archetipo si può considerare come il RL "puro", il secondo può rivelarsi utile in particolar modo durante le prime fasi dell'apprendimento, dove gli sviluppatori potrebbero voler fornire all'agente una policy di base da cui partire (ad esempio incentivando in maniera particolare determinate rotte ritenute più promettenti

o privilegiando in maniera particolare l'esplorazione) e tramite la quale apprendere la policy che verrà utilizzata in seguito per proseguire nell'apprendimento in maniera più efficiente o direttamente come policy finale.

3.3 Due archetipi fondamentali: Montecarlo e TD-control

Per esemplificare quelle che sono le due forme basilari opposte da cui poi prendono corpo tutte le forme più avanzate di RL, verranno illustrate qui di seguito le caratteristiche principali e peculiari che definiscono i metodi Montecarlo e i metodi TD-control.

3.3.1 Metodi Montecarlo

I metodi Montecarlo sono metodi basati sul calcolo della media dell'expected return. Per questo motivo, i metodi Montecarlo sono applicabili solo ai task episodici, poiché è solo in questi casi che è possibile ottenere dei return completi; più precisamente, sono adatti a trattare tutti quei casi in cui l'interazione è chiaramente suddivisibile in episodi i quali, prima o poi, raggiungeranno una fine a prescindere dalle azioni intraprese. Il punto di forza dei metodi Montecarlo è sicuramente quello di non necessitare di un modello dell'ambiente, ovvero di una conoscenza a priori delle transizioni di stato causate da una data azione, e di poter quindi imparare dall'interazione stessa con l'esterno nel senso più puro del termine.
Il funzionamento di base è quello del loop di policy improvement precedentemente descritto ma con una differenza fondamentale: in questo caso, infatti, la value function non viene effettivamente calcolata bensì appresa direttamente tramite la reward totale ricevuta al termine di un episodio, ovvero non è necessario conoscere a priori le probabilità delle transizioni da stato a stato ma queste sono apprese dall'agente stesso durante il processo.
Per quanto riguarda il problema dell'esplorazione, questa è assicurata principalmente tramite il meccanismo denominato *exploring starts*, che consiste nell'assumere che ogni episodio inizi in uno stato ben preciso e che ognuno degli stati possibili abbia una probabilità diversa da zero di essere selezionato. In tal modo, assicurandosi di somministrare

all'agente un numero sufficientemente grande di episodi, si può essere ragionevolmente certi che ogni stato venga visitato un numero di volte sufficiente. Utilizzando le notazioni introdotte in precedenza, lo pseudocodice per un generico metodo Montecarlo è riportato nel box sottostante.

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:
  $\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
  $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
  $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):
  Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ such that all pairs have probability > 0 (exploring starts)
  Generate an episode starting from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow G + R_{t+1}$
    Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
      Append G to $Returns(S_t, A_t)$
      $Q(S_t) \leftarrow \text{average}(Returns(S_t, A_t))$
      $\pi(S_t) \leftarrow \text{argmax}_a\, Q(S_t, a)$

Figura 3.3: Pseudocodice di base di un metodo Montecarlo

3.3.2 Metodi TD-control

I metodi Temporal Difference (TD) sono il secondo modello principale di algoritmo per sistemi di RL, prendendo elementi dalla teoria della programmazione dinamica classica e mutuandone altri dai metodi Montecarlo. I metodi TD, infatti, possono apprendere dalla pura esperienza come i metodi Montecarlo senza bisogno di un modello dell'ambiente, ma come i metodi di programmazione dinamica sono in grado di apprendere da expected returns parziali e non hanno quindi bisogno di attendere la fine di un episodio (e in generale possono essere applicati anche ai task non-episodici). In pratica, mentre i metodi Montecarlo devono aspettare il termine di un episodio per poter ricavare G_t per poter calcolare l'update della value function, i metodi TD devono compiere solo un passo ed effettuano quindi l'update utilizzando la reward ottenuta. Fatta salva questa differenza, anche in questo caso il funzionamento di base ricalca quello della policy improvement come detto in precedenza per i metodi Montecarlo.
Un ulteriore passo avanti rispetto a questo approccio è quello dei cosiddetti metodi n-step TD, che possono essere visti come un incrocio fra i metodi Montecarlo e i metodi

TD; tali metodi, infatti, non imparano né al termine di un episodio né ad ogni singolo passo, ma apprendono (da cui il nome) ogni n passi, utilizzando l'expected return incompleto ottenuto dopo gli stessi n passi. È facile immaginare come questi metodi possiedano le migliori peculiarità di entrambi i paradigmi.

3.4 L'algoritmo Tabular Sarsa

Sarsa è un metodo on-policy basato su TD control. Il sistema apprende una action-value function; in particolare, vale la seguente:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right]$$

La formula precedente indica che il valore associato alla coppia (S_t, A_t) viene aggiornato a seguito di una interazione nel seguente modo: al suo valore precedente viene aggiunta la somma fra la reward ottenuta, il valore associato alla coppia S_{t+1}, A_{t+1} moltiplicato per il fattore di sconto e poi viene sottratto nuovamente il valore attuale di $Q(S_t, A_t)$.
Questo aggiornamento dei valori viene compiuto dopo ogni transizione partita da uno stato non-terminale, mentre se S_{t+1} è uno stato terminale, allora $Q(S_{t+1}, A_{t+1})$ è per definizione impostato a zero. Si può notare come questa regola di update utilizza ogni elemento della quintupla $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, da cui deriva il nome **Sarsa**. Benché sia possibile realizzare una versione off-policy di Sarsa, viene spontaneo applicarlo in versione on-policy, applicando il metodo del value iteration come descritto precedentemente.
Il parametro α è denominato *step size*, e indica il tasso di apprendimento ad ogni passo; è un valore compreso fra 0 (incluso) e 1 (escluso), dove 0 indica l'agente che non modifica la propria action-value function in base alle nuove reward ottenute (apprendimento nullo), mentre 1 corrisponde all'agente che sostituisce completamente il vecchio valore di $Q(S_t, A_t)$ con quello di $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$ (apprendimento massimo). Esso può calare nel corso della computazione (in tal modo la computazione converge e permette di ottenere la policy ottimale per il problema a patto che esso sia stazionario[3]), mentre

[3] Ad esempio: un giocatore di scacchi che non cambia mai il proprio modo di giocare a prescindere dal comportamento del suo avversario.

se viene mantenuto costante non si può assicurare l'ottenimento della policy ottimale ma è possibile continuare ad apprendere anche in presenza di problemi non stazionari.
Per assicurare che il metodo abbia sempre una possibilità di esplorare nuove combinazione di stati e azioni, l'azione viene scelta tramite una modalità definita *epsilon-greedy*, ovvero l'azione viene scelta con probabilità $1 - \epsilon$ come l'azione più vantaggiosa secondo la policy, ma con probabilità ϵ in maniera casuale, dove ϵ è un numero reale positivo tendente a 0. Una forma generale dell'algoritmo è quella presentata nel box sottostante.

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Figura 3.4: Pseudocodice di Sarsa

I principali vantaggi di questo algoritmo risiedono nella sua semplicità concettuale unita ad un livello di performance comunque elevato. Inoltre, la sua natura di metodo TD-control lo rende flessibile e facilmente applicabile anche in casi reali.
Per queste ragioni, la soluzione implementata all'interno di MS-rehab utilizza questo algoritmo.

Capitolo 4

Progettazione e implementazione della soluzione

In questo capitolo si descriveranno le caratteristiche della soluzione proposta, partendo dalle caratteristiche degli esercizi presi in esame e analizzando poi le scelte implementative e la loro integrazione con l'architettura pre-esistente.

4.1 Descrizione degli esercizi

Le funzioni cognitive prese in analisi sono quelle dell'attenzione e della memoria; tale scelta è stata fatta innanzitutto per l'affinità strutturale fra gli esercizi di queste due categorie (contrariamente a quanto avviene per gli esercizi sulle funzioni esecutive).
Sono stati selezionati due esercizi, uno di attenzione e uno di memoria. Il primo è un esercizio di attenzione alternata, e consiste nel fornire al paziente una serie di immagini che si alterneranno sullo schermo una alla volta, e il paziente dovrà cliccare su tutte le occorrenze di una data immagine fino a quando udirà un suono; a quel punto, il paziente dovrà cliccare su tutte le occorrenze di un'altra immagine. La performance viene valutata in base al numero di risposte giuste, il numero di errori (ovvero il numero di occorrenze di altre immagini selezionate) e il numero di risposte omesse (ovvero le risposte corrette che il paziente non riconosce come tali). Più nello specifico, la performance è calcolata tramite una F-measure scontata in base al tempo impiegato

per completare l'esercizio; in prima battuta definiamo dunque precision e recall come segue:

$$p = \frac{pC}{pC + pW} \qquad r = \frac{pC}{pC + pM} \qquad p \in [0,1], \quad r \in [0,1]$$

Dove pC, pW e pM sono, rispettivamente, le risposte corrette, quelle sbagliate e quelle omesse. In seguito per semplificare la formula finale introduciamo il concetto di threshold, definita come segue:

$$t' = (1 - thr) * t + thr, \qquad thr \in [0,1], \quad t' \in [0,1]$$

Dove a sua volta t è la frazione dell'intervallo di tempo massimo disponibile impiegata dal paziente, ovvero

$$t = \frac{maxT - pT}{maxT}, \qquad maxT > 0, \quad t \in [0,1]$$

Definendo pT come la frazione di tempo massimo impiegata effettivamente impiegata dal paziente per completare il task.
Con tali elementi, possiamo infine introdurre la effettiva formula per calcolare la performance:

$$perf = (1 + \beta^2) * (\frac{p * r}{\beta^2 * p + r}) * t', \qquad \beta > 0, \quad perf \in [0,1]$$

La formula è costruita in modo tale che un paziente che completa un esercizio senza commettere errori entro il tempo massimo consentito, ottiene una performance di almeno thr (il minimo per superare l'esercizio). Se il paziente completa un esercizio senza commettere errori impiegando un tempo nullo, ottiene una performance pari a 1. Un esercizio completato velocemente può permettere al paziente di sopperire a qualche errore che ha commesso nell'esecuzione; i due parametri thr e β possono scelti, rispettivamente, per rendere più o meno semplice il superamento di un esercizio e pesare in maniera differente precision e recall nel calcolo della performance. Un valore tipico per thr è 0,8, mentre un valore tipico per β è 1, che dà la stessa importanza a precision e recall. Se $\beta < 1$, allora alla precision viene dato un peso maggiore che alla recall, viceversa se $\beta > 1$. Gli esercizi in cui il tempo non è rilevante mantengono una formula analoga in cui il valore di t' è assente.

Il primo esercizio preso in esame è un esercizio di attenzione alternata: al paziente verranno mostrate due immagini e gli verrà chiesto di selezionare le occorrenze della prima fino a quando verrà riprodotto un suono; a questo punto, dovrà selezionare le occorrenze della seconda immagine. Se il suono verrà nuovamente riprodotto dovrà tornare a selezionare la prima immagine e viceversa. Quando l'esercizio vero e proprio ha inizio, una serie di immagini apparirà a schermo, dove ogni immagine entrerà da destra, si fermerà al centro dello schermo per un certo intervallo di tempo e quindi uscirà sulla sinistra per lasciare il posto all'immagine successiva, e questo ciclo continuerà fino al termine dell'esercizio.
La difficoltà dell'esercizio varia in base a cinque parametri, i quali possono assumere tre valori diversi che corrispondono a tre livelli di difficoltà crescente:

- **#iterazioni**: numero di volte in cui il paziente dovrà cambiare obiettivo fra le due immagini fornite;
- **#stimuli**: numero totale di occorrenze per ogni target durante l'esercizio;
- **Time**: tempo di permanenza di una figura a schermo prima di essere sostituita dalla successiva
- **Distractors**: la presenza di elementi di distrazione più o meno complessi sullo sfondo della pagina dell'esercizio;
- **Color**: quanto marcata è la differenza di colore fra le varie icone.

La figura 4.2 mostra l'aspetto dell'esercizio appena descritto, mentre la figura 4.1 mostra i parametri dell'esercizio e i rispettivi valori:

Livello	# iterazioni	# stimoli per target	Tempo	Distrattori	Colore
1	2	20	4	no	diversi
2	3	15	3	semplici	omogenei
3	4	10	2	complessi	bianco/nero
NOTE			secondi a stimolo		

Figura 4.1: Parametri e rispettivi valori per l'esercizio di attenzione

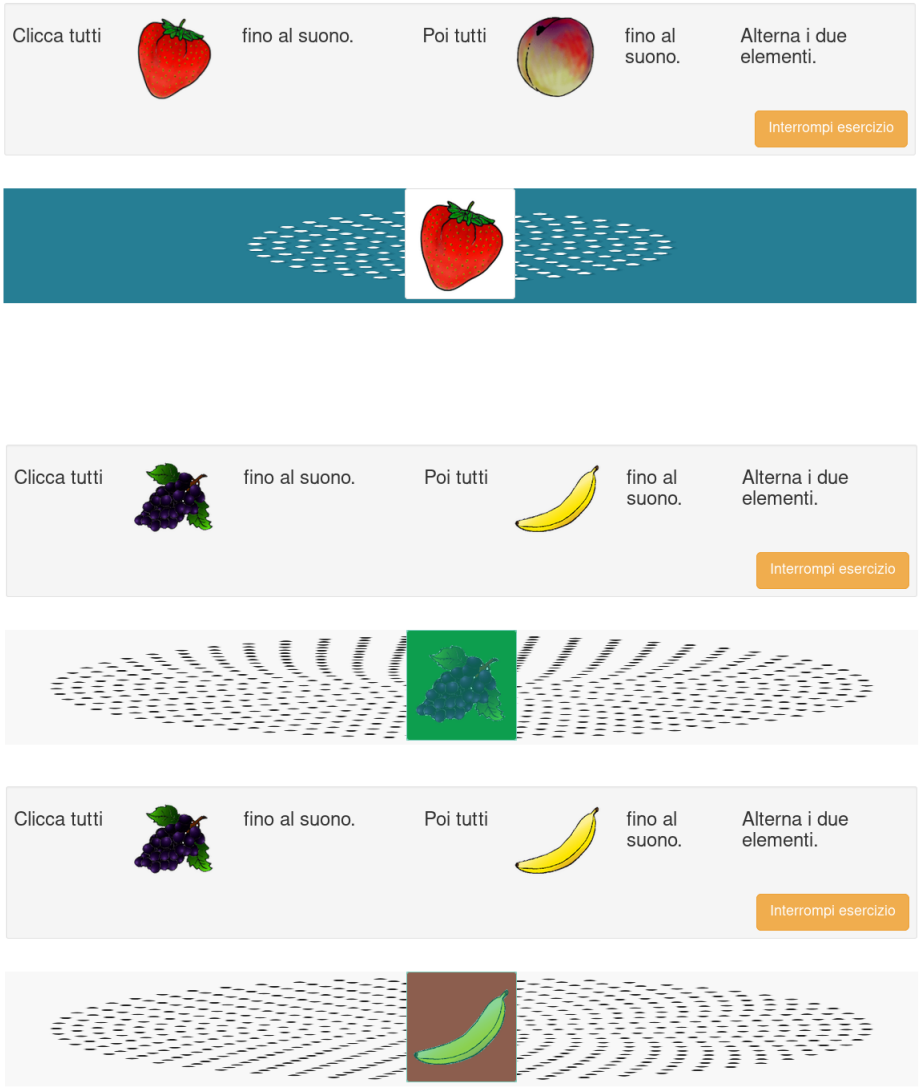

Figura 4.2: Aspetto dell'esercizio di attenzione

Il secondo è un esercizio di memoria (memoria nback o di lavoro) e consiste nel mostrare al paziente una alla volta una sequenza di immagini: il paziente dovrà premere il pulsante verde se l'immagine attualmente a schermo è uguale a quella che era presente a schermo N immagini prima (dove N viene comunicato al paziente all'inizio dell'eser-

cizio), altrimenti dovrà premere il pulsante rosso. La performance è valutata in maniera analoga alla precedente, mentre i parametri che regolano la difficoltà (anch'essi con tre diversi livelli possibili a difficoltà crescente) sono:

- **N**: stabilisce il numero di figure che il paziente deve ricordare, dovendo confrontare l'immagine attualmente a schermo con quella apparsa N immagini prima;
- **#stimuli**: numero totale degli elementi mostrati a schermo durante l'esercizio;
- **Time**: tempo di permanenza di uno stimolo sullo schermo prima di essere sostituito;
- **Color**: quanto marcata è la differenza di colore fra le varie icone

Di seguito è mostrato l'aspetto dell'esercizio (figura 4.4) e la tabella dei parametri (figura 4.3

Livello	N	Tempo	Colori	# stimoli
1	1	5	diversi	20
2	2	4	omogenei	40
3	3	3	bianco/nero	80
NOTE		secondi a stimolo		

Figura 4.3: Parametri e rispettivi valori per l'esercizio di memoria

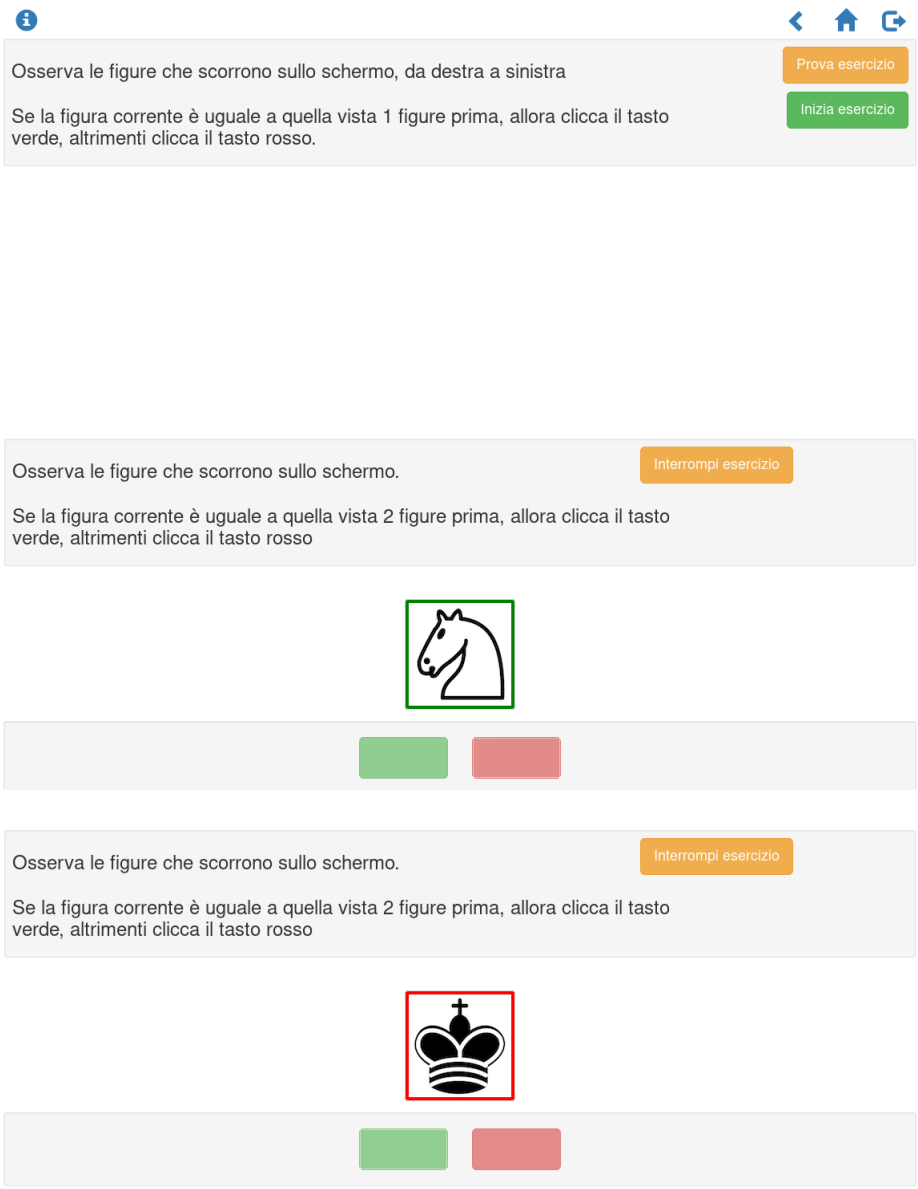

Figura 4.4: Aspetto dell'esercizio di memoria

4.2 Modellazione del problema

La regolazione automatica della progressione della difficoltà degli esercizi è stata modellata per essere formulato sotto forma di problema di RL. Lo stato dell'ambiente

consiste nel tipo di esercizio che si sta affrontando, nei parametri che determinano la difficoltà dell'esercizio in questione e nel livello raggiunto da ognuno di essei; con queste informazioni è possibile ricavare il livello e la difficoltà raggiunti, dove il livello è pari alla somma dei livelli dei singoli parametri, mentre la difficoltà, comunque basata sul livello totale, dipende da esercizio a esercizio. Tramite tale livello cumulativo, il sistema di MS-rehab sarà in grado di generare il nuovo esercizio con i parametri richiesti. In questo modo ogni esercizio può essere trattato in maniera completamente autonoma rispetto agli altri, garantendo la possibilità di apprendere una policy specifica per ognuno di essi.

L'agente si occupa di modificare il valore dei parametri in accordo con le informazioni ricevute dall'ambiente; in tal modo viene creato un nuovo esercizio basandosi sull'effettiva prestazione del paziente, permettendogli dunque di progredire in base ai risultati raggiunti esercizio per esercizio; per compiere la scelta, l'agente riceve come reward la performance ottenuta dal paziente nell'ultimo esercizio, ovvero quanto la prestazione nella risoluzione di tale compito sia stata buona o meno. In base a tale risultato l'agente potrà decidere di aumentare il livello di un parametro (a differenza della versione attuale di MS-rehab, l'agente può anche decidere di aumentare molto un parametro lasciando indietro gli altri[1], mentre nella versione attuale i parametri vengono aumentati allo stesso ritmo), di diminuirlo o eventualmente decidere che il paziente ha bisogno di effettuare un altro esercizio con la medesima difficoltà. In ogni caso, l'agente può modificare al più il livello di un solo parametro. Quest'ultima scelta è stata fatta sia per non alterare la struttura generale di generazione degli esercizi di MS-rehab (la quale appunto gestisce solo il caso in cui un solo parametro al massimo sia modificato ad ogni istanza successiva di un esercizio) sia per permettere l'utilizzo di algoritmi di RL senza funzioni di approssimazione; tali algoritmi, infatti, sono applicabili in contesti in cui la dimensione dell'albero delle scelte è arbitrariamente grande ma, come suggerisce il nome stesso, prendono decisioni in base ad una approssimazione su quanto appreso tramite le visite precedenti a stati simili. Essendo possibile si è dunque preferito modellare il problema in modo tale da poter utilizzare una policy esatta. Limitare ad uno il massimo di parametri modificabili in un singolo passo ha limitato fortemente il nume-

[1] Potenzialmente tale proprietà rappresenta una delle novità principali dell'approccio basato su RL.

ro massimo di coppie stato-azione possibili, permettendo dunque una implementazione che non necessitasse di una funzione di approssimazione.
Lo schema appena descritto è riassunto nella figura 4.5: MS_Agent ed MS_Environment compiono continuamente il loro ciclo, e il secondo, dopo ogni passo di interazione, notifica ad MS-rehab (in particolare, ciò accade all'interno della classe ExerciseService) il nuovo stato raggiunto.

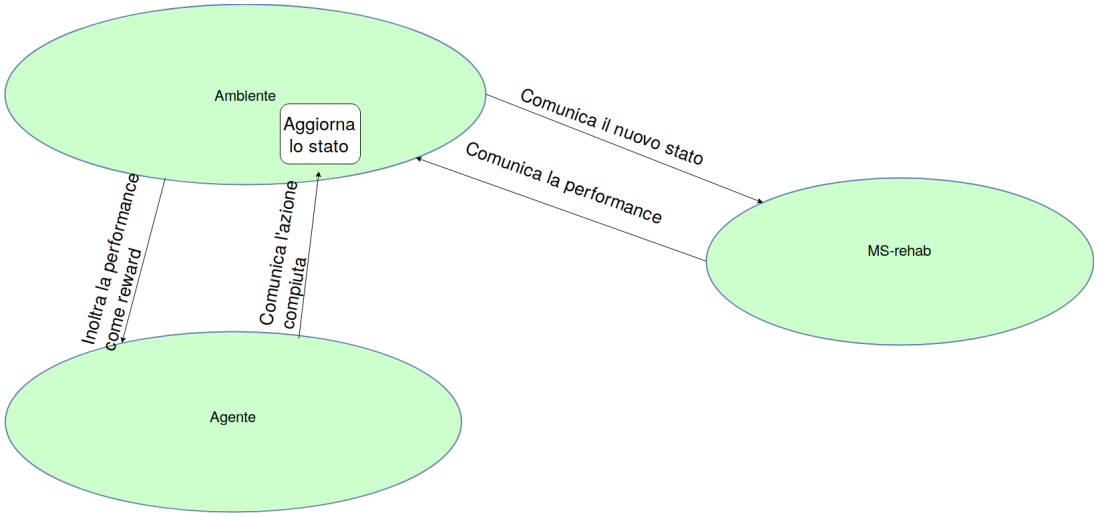

Figura 4.5: Interazione fra agente, ambiente e MS-rehab

Per fare un esempio pratico, supponiamo che un paziente abbia svolto un esercizio. Se la performance ottenuta è molto buona o ottima allora l'agente, in base alle sue conoscenze precedentemente acquisite tramite interazione con l'utente (ovvero in base alla policy attualmente conosciuta), potrebbe decidere di aumentare, nel prossimo esercizio, il livello di un parametro che in precedenza ha dato più difficoltà al paziente (ovvero il cui aumento aveva portato ad una reward cumulativa inferiore); se invece la performance fosse buona potrebbe per contro decidere di aumentare il livello di un parametro il cui incremento di difficoltà non ha dato problemi al paziente quando era stato incrementato in precedenza; infine, se la performance dovesse risultare scarsa, l'agente deciderà probabilmente di non aumentare il livello di un parametro o addirittura di decrementarlo, in particolare se il paziente avesse ottenuto risultati insufficienti in più prove consecutive.

4.3 Implementazione

Di seguito verranno descritte nel dettaglio le specifiche del codice implementato all'interno di MS-rehab per inserire le feature descritte in precedenza. In particolare verranno descritte le librerie utilizzate, le classi implementate e in che modo queste ultime interagiscono fra di loro.

4.3.1 Le librerie RL-Glue e RL-Library

RL-Glue consiste in una libreria C per facilitare la realizzazione di sistemi basati su RL, creando un framework standard per la comunicazione fra agente e ambiente senza la necessità di implementare ogni volta il codice di comunicazione fra le due parti. Le due entità vengono poi istruite su come comportarsi tramite la realizzazione di una classe che descrive l'esperimento, specificando il numero di episodi, il massimo numero di passi per episodio e altri dati rilevanti. Tramite le tre classi (l'agente, l'ambiente e l'esperimento) si definisce dunque il problema di reinforcement learning d'interesse. Tramite tale struttura standard si ottiene non solo un framework facilmente utilizzabile per la realizzazione di esperimenti ma anche una serie di linee guida da seguire per facilitare lo scambio e la verifica dei risultati.

RL-Library nasce con l'intento di creare un luogo centrale per la comunità scientifica dedita alla ricerca nel campo del RL per lo scambio di codice e informazioni. Allo stato attuale, la libreria si compone di codec per l'implementazione di agenti, ambienti ed esperimenti in linguaggi differenti. I codec sono compatibili con il core di RL-Glue, e in particolare sono presenti codec Java, Python, MatLab e Lisp. La libreria fornisce anche alcuni esempi di base per agenti e una collezione di ambienti celebri per il test di algoritmi di RL.

4.3.2 MS_Agent

È una classe implementata dall'autore in java, e racchiude al suo interno tutti i metodi necessari alla formalizzazione dell'agente, in particolare quelli per la sua inizializza-

zione, per l'avviamento, per la gestione dell'interazione con l'ambiente "ad un passo" (ovvero un ciclo "azione-reward") e quelle per l'interruzione dell'esecuzione. MS_-Agent è un agente studiato specificatamente per il caso di MS-rehab, ma è capace di interagire con qualunque tipo di ambiente.
Come detto in precedenza, nel paragrafo 3.4, MS_Agent è basato su tabular SARSA. Tale scelta è stata fatta principalmente perché si è reputato impossibile suddividere efficacemente il problema[2] in episodi separati facilmente riconoscibili e si è dunque deciso di apprendere ad ogni passo dell'interazione con l'ambiente (ovvero: dopo il completamento di ogni singolo esercizio) in maniera continuativa. Per quanto detto finora risulterà spontaneo al lettore capire perché si è dunque scelto di implementare un metodo TD-control per risolvere il problema (come discusso nella sezione 3.4). Inoltre, la versione tabulare è stata scelta in virtù del numero limitato di stati possibili. Negli esercizi più complicati, infatti, sono presenti sei parametri, e ognuno di questi può variare fra tre valori diversi (che all'interno di MS-rehab hanno valori differenti ma che all'interno del framework di RL sono stati tutti rimappati sui valori 0, 1 e 2), per un totale di 3^6 stati possibili. Tale numero è stato reputato affrontabile tramite una soluzione senza approssimazioni, e di conseguenza lo stesso vale a maggior ragione per gli esercizi con un numero di parametri inferiore.
Gli attributi principali della classe, oltre ai campi necessari all'implementazione dell'algoritmo SARSA (sarsa_stepsize, sarsa_epsilon, sarsa_gamma)[3] e la value function sono:

- **policyFrozen**: indica se l'apprendimento è attivo o meno; in caso sia inattivo l'agente continuerà a selezionare le azioni in base alla policy appresa fino a quel momento, senza modificarla ulteriormente;

- **exploringFrozen**: indica se l'esplorazione degli stati sconosciuti è attiva o meno; in caso sia inattiva (frozen) l'agente sceglierà sempre, per lo stato attuale, l'azione con associato il valore maggiore nella propria policy;

[2] Apprendere la policy migliore per guidare il processo di riabilitazione cognitiva di un malato di sclerosi multipla.

[3] Questi sono gli omonimi parametri stepsize ed epsilon (propri dei metodi TD-control e quindi di SARSA) e gamma (il discount factor).

- **valuePath**: indica il path in cui trovare il file su cui è salvata la policy appresa al termine dell'ultima interazione; in tal modo è possibile riprendere l'apprendimento da dove era stato interrotto;

- **exerciseType**: indica iltipo di esercizio attualmente in esecuzione; in tal modo l'agente ha modo di settare adeguatamente alcuni parametri specifici quali il numero di azioni disponibili (pari al numero di parametri) e la value function corrispondente all'esercizio selezionato[4].

Per quanto riguarda i metodi, invece, la classe contiene tutti i metodi necessari al funzionamento tramite la libreria RL_Glue:

- **agent_init**: inizializza tutte le variabili necessarie alla comunicazione con l'ambiente; tali informazioni sono contenute in un oggetto denominato taskSpecification che, come il nome stesso suggerisce, contiene le specificazioni proprie dell'ambiente in cui si agisce e del compito da apprendere (tipo e numero di azioni, numero degli stati possibili, range della reward);

- **agent_start**: inizia l'esperimento di RL ottenendo dall'ambiente lo stato di partenza e compiendo la prima azione;

- **agent_step**: sostanzialmente analoga ad agent_start, gestisce tutti i passi successivi al primo; la differenza sostanziale è che tutte le azioni successive alla prima vengono compiute non solo in base allo stato attuale dell'ambiente ma anche della reward ottenuta in seguito al passo precedente;

- **agent_end**: gestisce il momento dell'apprendimento al termine dell'episodio se il task è episodico o dopo un determinato numero di passi se il task è non-episodico; in questa specifica implementazione si è scelto di apprendere ad ogni passo, e tale impostazione è stata realizzata per semplicità modellando il problema come una serie di episodi composti da un solo step, dove ogni step coincide alla risoluzione di un'istanza dell'esercizio da parte del paziente;

[4] Ad ogni esercizio, infatti, corrisponde una diversa value function. Ciò è reso necessario dal diverso numero di parametri che regolano i vari esercizi, ma allo stesso tempo permette anche di migliorare la personalizzazione del percorso riabilitativo.

- **agent_cleanup**: permette di liberare la memoria occupata da dati non più utilizzati al termine dell'episodio;

- **agent_message**: permette di gestire l'utility di RL_Glue che consente di inviare messaggi all'agente per attivare determinate funzioni durante il learning; in particolare sono stati implementati messaggi per congelare e scongelare apprendimento ed esplorazione, per settare il tipo di esercizio, per ottenere l'indirizzo di salvataggio del file della policy, per salvare la policy attuale e per caricare la policy da file.

Oltre a quelli sopra elencati, sono stati implementati i seguenti metodi:

- **egreedy**: utilizzato ad ogni passo (quindi in agent_start e agen_step), permette di ottenere l'azione da intraprendere secondo il criterio epsilon-greedy tipico dell'algoritmo SARSA;

- **getAction**: permette di ottenere il valore dell'azione appena intrapresa (tale metodo viene invocato all'interno della classe ExerciseService per poter determinare quale parametro incrementare);

- **SaveValueFunction**: scrive su un file la value function calcolata fino a quel momento;

- **LoadValueFunction**: riempie il vettore contente la value function con i dati presenti su un file.

In figura 4.6 è riportato il class diagram di MS_Agent.

4.3.3 MS_Environment

È una classe che fornisce la struttura generale dell'ambiente necessario per la modellazione degli esercizi di MS-rehab come problemi di RL. In particolare, la classe implementa i metodi per l'inizializzazione, l'avviamento, l'interazione "ad un passo" con l'agente e l'interruzione dell'esecuzione di MS_Environment (in accordo con le

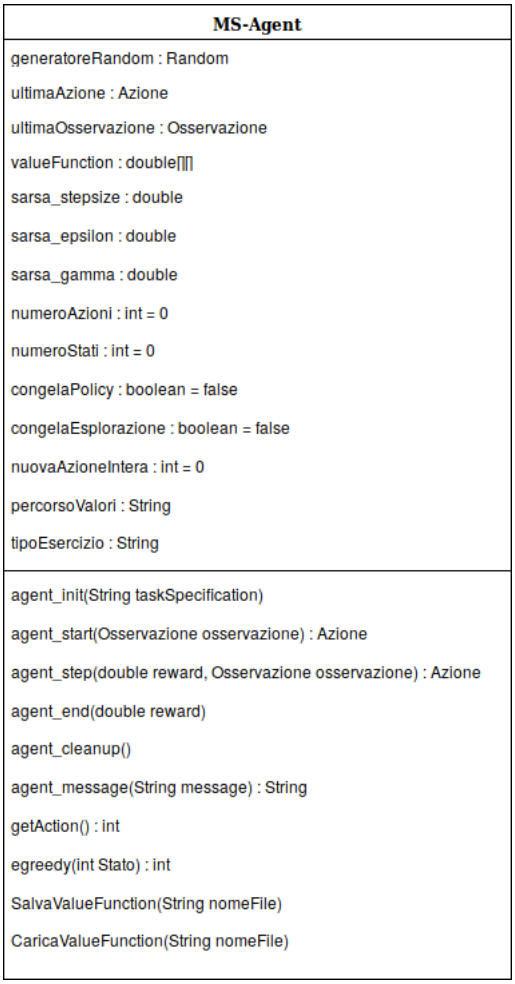

Figura 4.6: Diagramma di classe di MS_Agent

specifiche di RLGlue). Per permettere a MS_Environment di adattarsi a seconda dall'esercizio preso in esame viene utilizzata la classe ExerciseDescription, la quale contiene al suo interno il mapping necessario fra gli esercizi di MS-rehab e i dati necessari ad MS_Environment per cooperare con un'istanza di MS_Agent[5].
Gli attributi di MS_Environment sono analoghi ai medesimi attributi omonimi presenti in MS_Agent (valuePath, exerciseType) più l'attributo numParam (che indica il numero di parametri dell'esercizio), mentre i campi presenti in ExerciseDescription sono quelli

[5]Tali dati consistono nel nome dell'esercizio (utilizzato per salvare in maniera riconoscibile i rispettivi file degli stati e delle value function) e nel numero di parametri dello stesso.

necessari per descrivere lo stato attuale dell'ambiente, come il tipo di esercizio in esecuzione, il vettore dei parametri, la difficoltà raggiunta, un intero rappresentante lo stato attuale, la performance e il path dove è possibile trovare il file contenente le informazioni sullo stato raggiunto nell'ultima sessione.
Anche all'interno di MS_Environment sono state implementate le funzioni base[6] per garantire la compatibilità con RL_Glue:

- **env_init**: inizializza tutte le componenti necessarie per definire il problema (numero di parametri, nome dell'esercizio, range che definisce il valore massimo e minimo della reward) che verranno utilizzate da MS_Agent;
- **env_start**: analoga ad agent_start;
- **env_step**: analoga ad agent_step;
- **env_cleanup**: analoga ad agent_cleanup;
- **env_message**: analoga ad agent_message; sono stati implementati messaggi per ottenere il percorso in cui trovare/salvare il file dello stato, per salvare e caricare lo stato, per impostare il tipo di esercizio in esecuzione, per settare il numero di parametri necessari per modellare l'esercizio attuale e per stampare lo stato attuale.

Oltre a queste sono state implementate due funzioni: una per ottenere il livello di difficoltà raggiunto e una per impostare la performance al valore ottenuto dal paziente.
All'interno di ExerciseDescription sono presenti i metodi necessari per gestire l'effettivo aggiornamento dello stato:

- **ExerciseDescription**: nel costruttore vengono settati il tipo di esercizio e il numero di parametri;
- **calcolaNumeroStati**: permette di sapere in quanti stati è possibile trovarsi nell'esercizio attuale;

[6] Tali funzioni hanno nomi analoghi a quelle necessarie per MS_Agent, ed anche la funzione è analoga; il loro scopo è permettere la comunicazione fra agente e ambiente all'interno del framework di RLGlue.

- **ottieniStato**: ritorna il valore dello stato attuale;
- **ottieniLivello**: ritorna il valore del livello raggiunto;
- **impostaStatoIniziale**: imposta lo stato attuale a quello raggiunto durante l'ultimo esercizio della sessione precedente;
- **èValido**: serve per testare se l'azione intrapresa non porterebbe un parametro oltre il valore massimo;
- **ottieniReward**: ritorna il valore della reward (ovvero la performance del paziente);
- **conversioneTernaria**: converte lo stato dalla forma ternaria a quella decimale da salvare[7];
- **impostaStato**: aggiorna lo stato dopo l'azione comunicata dall'agente;
- **aggiornaPosizione**: aggiorna il valore della variabile corrispondente all'azione compiuta dall'agente;
- **stampa_stato**: permette di stampare lo stato attuale;
- **salvaStato**: salva lo stato attuale su file;
- **caricaStato**: carica lo stato raggiunto durante l'ultimo esercizio dal file in cui è salvato;

Anche in questo caso, viene riportato di seguito il diagramma di classe.

[7]Questo metodo serve per convertire lo stato visto come concatenazione del valore di ogni parametro (ad esempio 10122) nel valore decimale corrispondente; il valore così ottenuto sarà poi la colonna corrispondente a quello stato nella matrice rappresentante la value function. Dunque, lo stato 10122 viene convertito in 98 e viene salvato nella cella corrispondente.

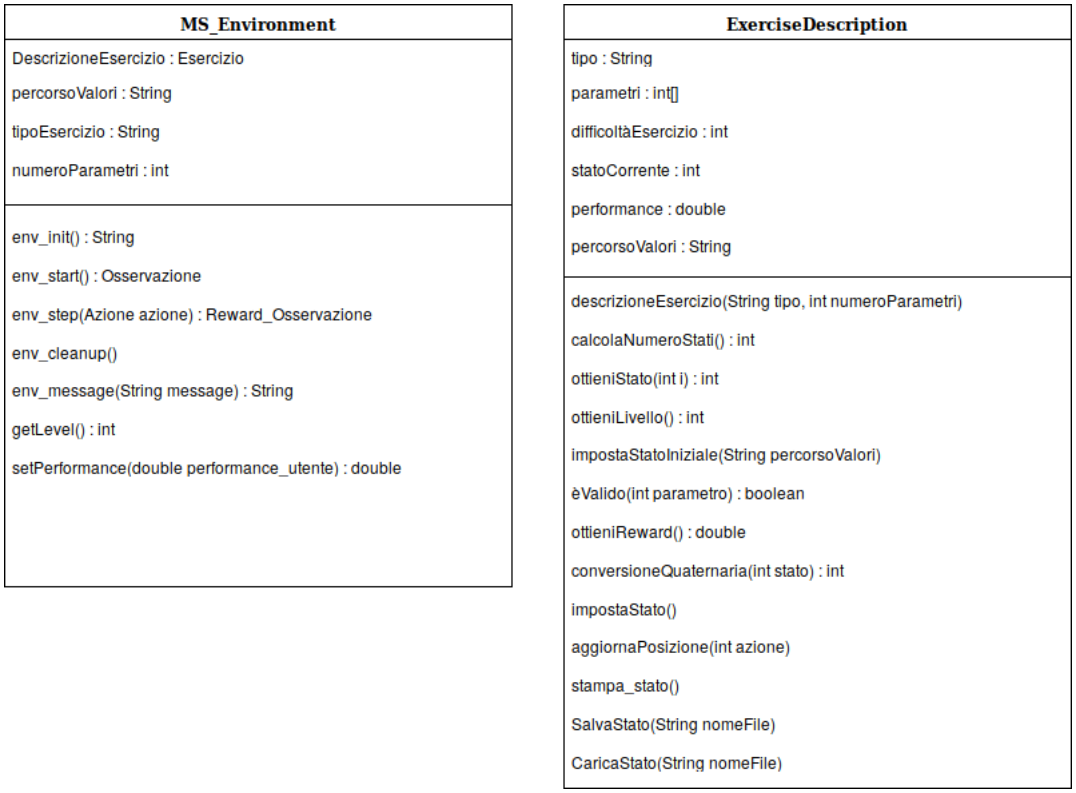

Figura 4.7: Diagramma di classe di MS_Environment

4.3.4 Descrizione della soluzione

Le due classi, MS_Agent e MS_Environment, rappresentano il framework generale applicato ai vari esercizi allo scopo di apprendere come regolare la difficoltà degli esercizi di attenzione e memoria all'interno di MS-rehab. In particolare, MS_Agent sceglie come azione da compiere quale parametro dell'esercizio aumentare o diminuire di livello, mentre MS_Environment fornisce all'agente, come relativa reward, la performance ottenuta dal paziente nell'esercizio così ottenuto. In questo modo l'agente può apprendere quali sono le strategie di progressione della difficoltà migliori per il paziente in base ai risultati da esso ottenuti negli esercizi propostigli. Ogni tipologia di esercizio necessiterà della propria istanza di MS_Agent e di MS_Environment, e anche le policy apprese saranno una differente per ogni classe di esercizi. Ai fini dei test condotti durante questa tesi, si è mirato all'apprendimento di un'unica policy basata sui risultati di tutti i test-user; tale scelta è stata attuata sia per motivi di ristrettezza di risorse ma anche

per ottenere una policy di iniziale. Tale base sarà poi utilizzata come policy di partenza per l'apprendimento personalizzato quando il software sarà utilizzato da pazienti reali; in questo modo si punta a fornire una policy iniziale funzionale che verrà poi raffinata in maniera personalizzata al crescere dell'esperienza riabilitativa del singolo individuo.

4.4 Integrazione con MS-rehab

All'interno di MS-rehab, tutte le chiamate alle funzioni necessarie per l'utilizzo di RL-GLue avvengono all'interno della classe ExerciseService. Prima della creazione di ogni esercizio vengono istanziate una variabile di classe MS_Agent e una di classe MS_Environment a cui, tramite opportune chiamate ai metodi agent_message e env_-message, vengono forniti i dettagli riguardanti l'esercizio in questione, in particolare il nome dell'esercizio (utilizzato per salvare lo stato raggiunto e la policy calcolata in file distinti per ogni esercizio) e il suo numero di parametri. In seguito, all'interno della fase dell'esercizio in cui viene calcolata la performance ottenuta dal paziente dopo il suo svolgimento, tale performance viene immediatamente fornita alla corrispondente istanza di MS_Environment, che potrà dunque terminare la costruzione dello stato e la comunicherà ad MS_Agent. L'agente potrà quindi selezionare l'azione da compiere e il parametro di cui modificare il livello; tale scelta viene comunicata ad ExerciseService sotto forma di un numero intero compreso fra 0 e il numero di parametri dell'esercizio: tale numero indica quale parametro aumentare, dove 0 indica che nessun parametro deve essere modificato. A questo punto viene effettuata l'effettiva variazione nel valore del parametro selezionato e infine viene creato l'esercizio da assegnare al paziente. Tale ciclo si ripete all'infinito, fornendo esercizi sempre nuovi al paziente, calibrati in base ai suoi personali risultati.
La scelta implementativa appena descritta è stata fatta al fine di mantenere più separate possibili la parte di codice che implementa il meccanismo di RL e la struttura preesistente di MS-rehab, che non ha subito modifiche sostanziali. Attraverso una variabile nel file di properties di sistema viene indicato se attivare il learning o meno. In caso affermativo, i parametri dell'esercizio vengono variati nella maniera sopra indicata invece che col sistema baseline precedente. Tale trasparenza permette la massima flessibilità

possibile nel passaggio fra un sistema di incremento della difficoltà all'altro senza alterare lo schema di funzionamento di base dell'applicazione; in tal modo un paziente che ha iniziato il proprio percorso riabilitativo con il vecchio sistema potrà passare facilmente a quello nuovo (e viceversa) solamente con un minimo intervento di configurazione. Oltre ad una evidente utilità in fase di testing, tale feature permetterà uno sviluppo parallelo delle due versioni del software in futuro senza doversi preoccupare di effettuare upgrade separati per le due versioni di MS-rehab.

Capitolo 5

Valutazione della soluzione

Per valutare se la soluzione proposta risultasse effettivamente superiore alla versione di baseline del sistema si è realizzato uno studio atto a misurare il miglioramento delle funzioni cognitive attenzione e memoria nei soggetti coinvolti nell'esperimento.
Il test è stato strutturato come segue[1]:

- In via preliminare, dieci utenti hanno addestrato il software, eseguendo venti[2] esercizi per entrambe le tipologie assegnate[3];
- Quindi sono stati creati altri due gruppi, entrambi composti da dieci utenti diversi da quelli che hanno addestrato il software: il gruppo A e il gruppo B;
- Tutti gli utenti dei gruppi A e B sono stati sottoposti al PASAT test (spiegato in dettaglio in seguito), un test psicologico atto a misurare la funzionalità cognitiva dei partecipanti all'esperimento;
- Tutti gli utenti dei due gruppi hanno svolto una sessione di due esercizi (uno per l'attenzione alternata e uno per la memoria di lavoro), dove per sessione si intende

[1]Tale struttura è stata creata in collaborazione con un team di ricerca del dipartimento di psicologia dell'Università di Padova.

[2]Tale numero è stato selezionato per permettere agli utenti di raggiungere, potenzialmente, il livello massimo di difficoltà dell'esercizio.

[3]Gli esercizi sono stati scelti sempre in collaborazione con il team dell'università di Padova guidato dalla professoressa Franca Stablum, in modo tale da poter risultare il più sfidanti possibili anche per dei pazienti sani.

l'esecuzione di varie istanze degli esercizi partendo dalla difficoltà minima fino al raggiungimento di un livello prefissato (corrispondente al livello massimo per l'esercizio di attenzione, mentre per l'esercizio di memoria è stato scelto il livello 15, ovvero il primo livello raggiungibile nella versione baseline in cui è richiesto di ricordare tre figure[4]). Il gruppo A ha utilizzato la versione di MS-rehab con RL, mentre il gruppo B ha utilizzato la versione di MS-rehab senza RL (già precedentemente denominata "versione baseline");

- A questo punto gli utenti sono stati nuovamente sottoposti al PASAT test in modo da misurare eventuali miglioramenti rispetto alla somministrazione precedente, cercando di evidenziare se uno dei due gruppi ottenesse un punteggio maggiore;

- Infine, a distanza di una settimana, a tutti gli utenti è stato chiesto di eseguire una singola istanza di entrambi gli esercizi alla massima difficoltà. Lo scopo di questo ulteriore controllo è quello di cercare di evidenziare se uno dei due gruppi mostrasse un livello di performance più alto e duraturo nel tempo[5].

Oltre ai risultati del PASAT test si sono raccolti i dati riguardanti le medie del numero di esercizi falliti[6], del numero di errori commessi e delle risposte omesse in media da entrambi i gruppi, cercando ancora una volta di evidenziare eventuali differenze fra i risultati dei due gruppi.
Ognuna delle fasi del test illustrate sopra verrà analizzata nello specifico nel seguito di questo capitolo.

[4] Si è scelto tale livello di difficoltà invece del livello massimo (21) per evitare che i soggetti del gruppo B dovessero eseguire il task per troppo tempo; infatti, con tale versione, raggiungere il livello massimo di 21 avrebbe richiesto il completamento di 42 istanze dell'esercizio (assumendo che non ne ne venisse fallita nemmeno una)

[5] Tale evenienza evidenzierebbe che uno dei due metodi permette di avere un percorso riabilitativo effettivamente efficace nel tempo e non solo provvisoriamente.

[6] Tale concetto viene ereditato dalla precedente versione di MS-rehab, dove per "fallito" si intende un esercizio concluso con una performance inferiore all'80%

5.1 Demografia

Di seguito è riportata la demografica degli utenti di che hanno partecipato allo studio; tale demografica è basata principalmente sulle caratteristiche di correlazione proprie del PASAT test (tali correlazioni saranno introdotte successivamente nel paragrafo dedicato al test stesso).
Di seguito sono riportate tre tabelle contenenti, rispettivamente, i dati demografici degli utenti selezionati per la fase di addestramento, per la fase di testing con la versione baseline di MS-rehab e per la fase di testing con la versione con RL.
In particolare sono riportate le voci legate all'età, al sesso, al titolo di studio e all'abilità coi videogames, specialmente con i videogiochi basati su piattaforme web. Quest'ultima voce demografica è basata su una scala di valori compresa fra 0 e 3, dove:

0 : mai utilizzato o quasi videogames

1 : utilizzati saltuariamente

2 : utilizzati di frequente

3 : utilizzati quotidianamente

I soggetti che hanno dato come risposta 0 sono stati esclusi dallo studio per evitare che la loro scarsa confidenza con la forma videogame potesse inficiare i risultati dell'esperimento.

	Età	Sesso	Titolo di studio	Abilità VG
Utente 1	26	F	Laurea Magistrale	2
Utente 2	22	F	Laurea	1
Utente 3	25	M	Laurea Magistrale	1
Utente 4	23	F	Laurea	2
Utente 5	26	M	Laurea Magistrale	2
Utente 6	24	M	Laurea	2
Utente 7	21	M	Superiori	1
Utente 8	25	F	Laurea	3
Utente 9	24	F	Laurea	2
Utente 10	21	M	Superiori	1

Tabella 1: demografica degli utenti scelti per la fase di addestramento

	Età	Sesso	Titolo di studio	Abilità VG
Utente 1	26	M	Laurea Magistrale	3
Utente 2	25	F	Laurea a Ciclo Unico	1
Utente 3	25	M	Laurea a Ciclo Unico	2
Utente 4	24	F	Laurea	1
Utente 5	26	F	Laurea Magistrale	2
Utente 6	22	M	Laurea	2
Utente 7	22	M	Superiori	2
Utente 8	23	F	Laurea	2
Utente 9	25	F	Laurea	1
Utente 10	20	M	Superiori	2

Tabella 2: demografica degli utenti scelti per la fase di testing della versione baseline

	Età	Sesso	Titolo di studio	Abilità VG
Utente 1	26	F	Laurea Magistrale	1
Utente 2	25	F	Laurea	1
Utente 3	26	F	Laurea Magistrale	2
Utente 4	28	M	Laurea Magistrale	3
Utente 5	28	F	Laurea Magistrale	2
Utente 6	23	M	Laurea	2
Utente 7	20	M	Superiori	2
Utente 8	20	F	Superiori	3
Utente 9	24	F	Laurea	2
Utente 10	23	M	Laurea	1

Tabella 3: demografica degli utenti scelti per la fase di testing con la versione con RL

Gli utenti sono stati distribuiti nella maniera più uniforme possibile, cercando in particolare di bilanciare le tre caratteristiche principali: età (media), titolo di studio (numero di laureati con titolo magistrale[7], laureati con titoli triennali, non laureati) e affinità coi VG (media). Tale bilanciamento è importante poiché la prestazione del PASAT test correla principalmente con gli anni di studio e con l'età, mentre l'abilità coi videogiochi è stata presa in esame per evitare che i risultati ottenuti durante l'interazione con MS-rehab risultassero influenzati da una minore affinità di alcuni soggetti con i videogiochi stessi.

5.2 Addestramento preliminare

Agli utenti sono stati assegnati esercizi partendo dal livello di difficoltà più basso, dando loro la consegna di effettuare venti esercizi per entrambe le tipologie; tutti gli utenti hanno avuto una progressione dello stato personalizzata, ma la policy appresa è stata unica e comune a tutti. In tal modo la policy inizialmente omogenea (ovvero dove ogni valore è settato a zero[8]) è stata aggiornata coi valori ottenuti tramite i risultati ot-

[7] Le lauree a ciclo unico sono state calcolate come lauree magistrali

[8] Tale scelta è stata attuata per non trasmettere dei bias all'agente, permettendogli di iniziare il proprio apprendimento da una policy uniforme rispetto a tutte le possibili azioni.

tenuti dagli utenti, partendo dagli esercizi più semplici e arrivando via via ai livelli più avanzati più di una volta, dando modo all'agente di visitare lo stesso stato o stati simili più volte e di vagliare i differenti risultati. Ogni utente ha iniziato la propria sessione di training partendo dalla policy ottenuta dall'utente precedente alla fine della sua interazione.

Per assicurarsi che l'agente esplorasse abbastanza spesso anche stati a prima vista meno promettenti il parametro di esplorazione è stato settato a 0.4 su un massimo di 1.0 (dove il parametro massimo corrisponde alla policy esplorativa pura, ovvero a un comportamento dell'agente tale per cui ogni azione è scelta casualmente, ovvero dove ogni azione ha la stessa probabilità di essere selezionata in ogni momento).

In figura 5.1 vengono mostrati i livelli di difficoltà massimi raggiunti rispettivamente dai vari utenti; si può notare come tali livelli siano inferiori al livello massimo (in particolare per quanto riguarda l'esercizio di memoria). Poiché ci si voleva accertare che l'agente

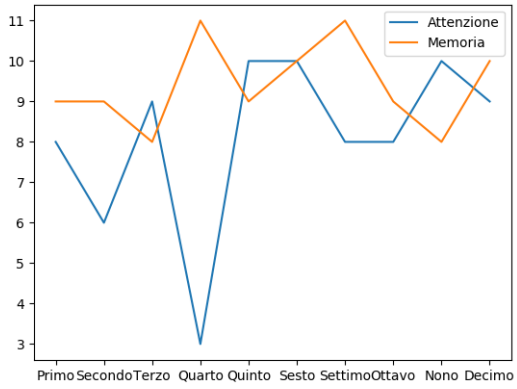

Figura 5.1: Livelli raggiunti durante la prima fase di addestramento

visitasse più stati possibile e che in particolare visitasse più traiettorie che portassero dal livello 1 al livello massimo, a tre utenti è stato chiesto di svolgere altri esercizi di attenzione, mentre a tutti gli utenti è stato chiesto di svolgere altri esercizi per la memoria. I risultati ottenuti a questo punto sono riassunti nei due grafici sottostanti, separati per le due funzioni cognitive (figure 5.2 e 5.3). Tramite tali grafici è possibile notare come dopo questa seconda sessione di training siano stati raggiunti livelli molto più alti, in

particolare nell'esercizio di memoria. L'elevato valore assegnato all'esplorazione ha

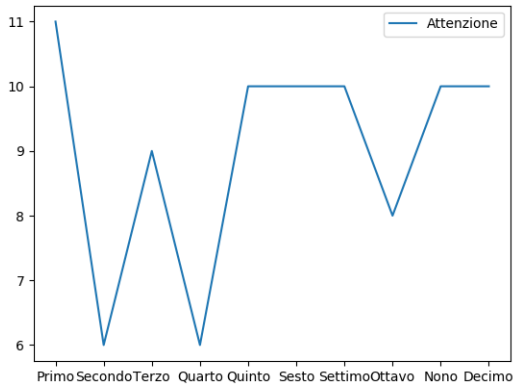

Figura 5.2: Livelli raggiunti durante la seconda fase di addestramento - Attenzione

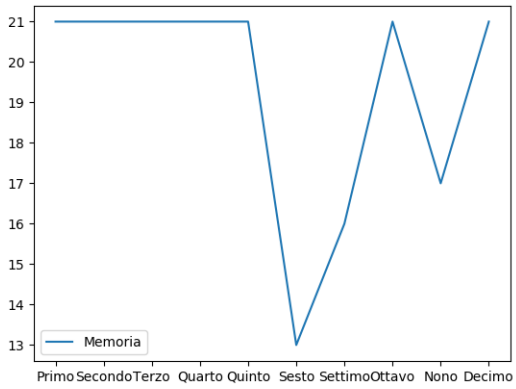

Figura 5.3: Livelli raggiunti durante la seconda fase di addestramento - Memoria

probabilmente causato le valli presenti nei precedenti grafici, ma tale elevato tasso di esplorazione era altresì necessario ad assicurarsi che l'agente visitasse il maggior numero di stati diversi possibile.
Avendo raggiunto più volte i livelli massimi ed avendo esplorato una notevole combinazione di stati possibili, si è ritenuto che l'agente avesse esplorato sufficientemente gli stati possibili e che dunque la policy così ottenuta potesse essere utilizzata per la fase successiva dello studio.

5.3 PASAT test - Introduzione e primo round

Ideato nel 1961 e successivamente sviluppato fra il 1974 e il 1981, questo test richiede di eseguire semplici addizioni allo scopo di misurare la velocità di elaborazione controllata dell'informazione, la memoria di lavoro e l'attenzione sostenuta [11]. La versione di PASAT utilizzata fornisce dei file audio contenenti una serie pre-registrata composta da 61 numeri (da 1 a 9) presentati in ordine casuale; tale serie può essere somministrata con cinque differenti ritmi di presentazione, dove l'intervallo fra gli stimoli può essere di 4000, 3000, 2600, 2200 o 1800 millisecondi, mentre la durata di pronuncia di ogni numero è invece costante a circa 600 ms. È facile intuire come la difficoltà aumenti al diminuire di tale intervallo temporale. Il soggetto è istruito a sommare l'ultimo numero fornito al numero immediatamente precedente, dunque il secondo numero è sommato al primo, il terzo al secondo, il quarto al terzo e così via. Ad esempio: se vengono presentati i numeri 1 e 9, la risposta corretta è 10; se il numero successivo è 4 la risposta sarà 13 e così via.
La somministrazione del test è preceduta, come suggerito dalle linee guida presenti nella lettura specializzata [11], da una fase di training divisa in quattro parti:

- Fase 0: spiegazione dettagliata delle regole;
- Fase A: simulazione scritta su carta di una breve istanza del test, prolungata fino alla piena comprensione del task da parte del soggetto;
- Fase B: simulazione del test nella sua versione finale, ovvero in forma orale;
- Fase C: ultima simulazione orale, questa volta utilizzando le registrazioni che saranno effettivamente utilizzate durante il test vero e proprio.

Terminata la fase di training si può dunque procedere con la somministrazione del test vero e proprio. Il test è stato eseguito nella versione con delay fra uno stimolo all'altro pari a 2600 millisecondi; tale scelta è stata effettuata anch'essa a seguito di un colloquio con il personale specializzato dell'università di Padova: le versioni da 4000 e 3000 ms sono state ritenute troppo semplici per una popolazione non affetta da disturbi cognitivi e dunque non sufficienti ad evidenziare un eventuale miglioramento a seguito dell'utilizzo del software, mentre la versione da 1800 ms è stata ritenuta troppo difficile per

essere somministrata nel corso di un test della durata prefissata.
La valutazione della prestazione del PASAT test viene fatta tenendo conto del numero di risposte esatte rispetto al totale (al fine di calcolare tale prestazione, dunque, non vi è differenza fra le risposte sbagliate e quelle omesse). Di questi risultati è stata poi calcolata la media per entrambi i gruppi e tali medie sono state poi paragonate tra loro per evidenziare eventuali divergenze.
In letteratura sono state dimostrate correlazioni fra la prestazione ottenuta nel PASAT test e l'età e gli anni di scolarizzazione [11]; la distribuzione degli utenti nei due gruppi è stata fatta in modo da assicurarsi che queste due variabili non influenzassero il risultato del test, poiché la popolazione è composta esclusivamente da studenti universitari fra i 20 e i 28 anni, la cui differenza di scolarizzazione risulta dunque estremamente ridotta[9].
Guardando ai risultati, la media della prestazione del gruppo A si è rivelata essere pari a 47,8, mentre quella del gruppo B 49,2; per accertarsi che tali valori non pregiudicassero i risultati successivi, sono stati calcolati tramite t-test i t-values e i p-values di entrambi, che sono risultati essere:

- t-value = 0,82
- p-value = 0,42

Questi risultati ci permettono di confermare l'ipotesi nulla, ovvero di asserire che i due gruppi appartengono alla stessa popolazione. Ciò permette dunque di concludere che le due medie non sono significativamente differenti.
Possiamo dunque concludere che tutte le differenze che rileveremo da ora in poi nei risultati ottenuti durante e dopo la fase di testing saranno effettivamente imputabili a differenze nella qualità della riabilitazione ottenuta interagendo con uno dei due sistemi.

5.4 Test del sistema

Una volta completato l'addestramento e la prima somministrazione del PASAT test, si è proceduto con la fase comparativa. Sia il sistema con RL che la versione baseline

[9] Si noti che anche i soggetti con titolo di studio Superiore sono attualmente iscritti ad un corso di laurea triennale o a ciclo unico.

di MS-rehab sono stati utilizzati da 10 soggetti; ad ognuno di essi è stata assegnata una sessione di esercizi a difficoltà minima e si è data consegna di procedere con gli esercizi fino al raggiungimento e al completamento con esito positivo di un esercizio a difficoltà massima.
Alla fine di questa fase, si sono controllati i dati menzionati precedentemente: il numero di esercizi falliti e il numero di errori commessi e di risposte omesse durante l'intera sessione.
In particolare, si sono ottenuti i seguenti risultati:

- Esercizi falliti: in media, il gruppo B ha fallito l'1,10% degli esercizi, mentre il gruppo A lo 0,56%.

- Numero di errori: in media, il gruppo B ha commesso 0,69 errori per esercizio, mentre il gruppo A 0,85.

- Numero di omissioni: il gruppo B ha omesso in media 0,18 risposte per esercizio, mentre il gruppo A solo 0,11.

Anche a questi risultati è stato applicato un t-test per verificare la rilevanza degli stessi rispetto all'ipotesi nulla "i due gruppi hanno ottenuto gli stessi risultati rispetto alla percentuale di esercizi falliti, al numero medio di errori e al numero medio di omissioni"; i t-value registrati sono stati, rispettivamente, -0,60, 0,60 e -1,03. I p-value, invece, sono stati rispettivamente pari 0,56, 0,56 e 0,31; questo significa, purtroppo, che per nessuna delle tre voci prese in esame è possibile rigettare l'ipotesi nulla.
Tali risultati evidenziano dunque un leggero miglioramento nella riduzione del numero di esercizi falliti e delle omissioni a favore del sistema basato su RL, mentre il sistema baseline ha permesso un risultato leggermente migliore rispetto al numero di errori; i risultati ottenuti tramite l'applicazione del t-test, però, non ci permettono di poterci basare su tali dati poiché gli elevati valori di tutti e tre i p-value suggeriscono che probabilmente la ripetizione dell'esperimento non restituirebbe lo stesso risultato.

5.5 PASAT test - Secondo round

A questo punto si è proceduto con la seconda somministrazione del PASAT test. Dopo un veloce riepilogo del compito da svolgere, gli utenti sono stati nuovamente sottoposti a due istanze del test con le medesime velocità del round precedente. I risultati sono stati analizzati in modo analogo a quelli riportati precedentemente per la prima somministrazione del test. Inoltre, a distanza di una settimana, agli utenti è stato richiesto di ripetere una istanza di entrambi gli esercizi alla massima difficoltà; questa ulteriore scelta sperimentale è stata fatta per cercare di evidenziare eventuali differenze nei progressi a lungo termine, che è un importante risultato che un programma di riabilitazione deve conseguire[10].
I risultati ottenuti durante questa fase sono stati quelli più rilevanti: infatti, il gruppo B ha ottenuto un incremento di prestazione medio di 2.8 punti, mentre il gruppo A ha incrementato in media la propria performance di ben 7.7 punti. Tali risultati sono inoltre convalidati dal t-test, che restituisce un t-value pari a 4.4 e un p-value pari a 0.0004, il che conferma che le due medie sono statisticamente differenti e che la probabilità che tali risultati siano frutto del caso è trascurabile. Inoltre, è stato applicato un t-test anche fra i risultati ottenuti durante la prima e la seconda istanza del PASAT test per ognuno dei due gruppi; questo ulteriore test ha restituito i seguenti risultati:

- Gruppo A: t-value = 8,27; p-value = 1,7 * 10^{-5}
- Gruppo B: t-value = 2,72; p-value = 0,02

In entrambi i casi è dunque possibile rifiutare l'ipotesi nulla, confermando la rilevanza dei dati ottenuti.
Tale risultato può essere dovuto al minor numero di esercizi necessari per completare il task assegnato: infatti, il gruppo B ha dovuto completare in totale 51 esercizi (22 per l'attenzione, 29 per la memoria) mentre il gruppo A in media ha completato 36 esercizi. Il minor numero di esercizi (e quindi di tempo) necessario per completare il task potrebbe aver comportato un minore livello di stress, permettendo dunque di attivare le

[10] Se i progressi si limitassero al solo breve termine è chiaro che il percorso riabilitativo non si può definire funzionale all'effettivo recupero delle funzioni cognitive del paziente.

funzioni cognitive degli utenti senza però comprometterle affaticandoli troppo. Questo potrebbe aver giocato un ruolo anche nel risultato legato al numero di errori commessi e di risposte omesse.

A questo punto, a distanza di una settimana, sono stati somministrati un esercizio di attenzione di livello 8 e uno di memoria di livello 15[11] a tutti gli utenti; ciò è stato fatto per controllare eventuali differenze di risultato a distanza di tempo dall'interazione, il che significherebbe che uno dei due sistemi garantisce dei risultati riabilitativi più duraturi nel tempo.

Per quanto riguarda l'esercizio di attenzione, la differenza fra le due performance è stata di 0,003 punti a favore del gruppo B per l'esercizio di attenzione, mentre per l'esercizio di memoria è stata 0,03 a favore del gruppo A. Applicando nuovamente un t-test, si ottiene per l'esercizio di attenzione un p-value pari a 0,33, mentre per l'esercizio di memoria tale valore si è rivelato essere pari a 0.06. In nessuno dei due casi è stato dunque possibile rigettare l'ipotesi nulla "i due gruppi hanno avuto performance equivalenti a distanza di una settimana", ma è comunque facile notare come i risultati ottenuti per l'esercizio di memoria siano al limite di tale condizione, indicando ancora una volta un leggero vantaggio del sistema basato su RL rispetto alla versione baseline di MS-rehab. Di seguito è riportata una tabella riassuntiva dei risultati ottenuti durante l'esperimento:

Fase dell'esperimento	Gruppo migliore	Rilevanza statistica del risultato
Primo PASAT	Pareggio	No
Esercizi falliti	A	No
Numero di errori	B	No
Numero di omissioni	A	Bassa
Secondo PASAT	A	Sì

Tabella 4: riassunto dei risultati ottenuti

[11] Tali difficoltà sono state scelte poiché sono i livelli minimi per cui un esercizio è "difficile", rispettivamente, per l'esercizio di attenzione e di memoria.

Capitolo 6

Conclusioni e futuri sviluppi

6.1 Riassunto dei risultati ottenuti

I risultati presentati nel capitolo 5 mostrano come l'applicazione di tecniche basate su RL si siano rivelate promettenti per il futuro della riabilitazione cognitiva computerizzata. In particolare, nonostante le performance degli esercizi non abbiano rilevato differenze sostanziali, i risultati ottenuti tramite le due somministrazioni del PASAT test dimostrano come il sistema con RL abbia permesso ai soggetti presi in esami di migliorare in maniera nettamente più marcata le proprie funzioni cognitive.
Tali risultati andranno confermati con ulteriori test eseguiti su soggetti in condizioni neuro-psicologiche non ottimali, ma confermano come tali tecnologie, seppure ancora ad uno stato embrionale, possano già competere con le tecnologie attualmente esistenti. Vale inoltre la pena sottolineare come tali risultati siano stati ottenuti nel corso di un esperimento le cui specifiche permettono una personalizzazione inferiore a quella che ci si propone di ottenere per i pazienti reali: questi, infatti, non interagiranno mai con il software con apprendimento bloccato, permettendo dunque di raggiungere un grado di personalizzazione dell'esperienza ancora maggiore. In questo modo l'esperienza dovrebbe risultare ancora più funzionale, aumentando di conseguenza la qualità della riabilitazione cognitiva esperita dai pazienti.

6.2 Sviluppi futuri

I risultati ottenuti finora, ovviamente, non possono essere considerati definitivi, in particolar modo poiché la popolazione utilizzata per la fase sperimentale è composta esclusivamente da soggetti giovani, sani e con un elevato tasso di scolarizzazione. La soluzione dovrà dunque essere messa alla prova con un esperimento appositamente realizzato per permettere l'interazione con una popolazione di soggetti con deficit cognitivi. Oltre a questo, il futuro del progetto verte principalmente sull'integrazione di metodi di **Safe** Reinforcement Learning [7], con la possibilità di realizzare un framework generale per l'implementazione di agenti di Reinforcement Learning all'interno di sistemi informatici per la riabilitazione cognitiva e non solo.
Inizialmente si potrebbe procedere all'implementazione di una meccanica di **Constrained Criterion**, ovvero tecniche atte alla realizzazione di sistemi nei quali si cerca contemporaneamente di massimizzare la reward ottenuta sul lungo periodo e di evitare che altri parametri selezionati non scendano (o salgano) oltre una certa soglia di tolleranza (ad esempio tramite l'inserimento un lower bound per la performance stessa, o per la varianza della performance).
Successivamente si potrebbe procedere all'implementazione di feature finalizzate a permettere la somministrazione all'agente di conoscenze pregresse. In particolare, si potrebbe permettere a esperti del settore di fornire all'agente la capacità di generalizzare le conoscenze in suo possesso, accelerando al tempo stesso il processo di apprendimento e riducendo ulteriormente la probabilità di incorrere in situazioni di fallimento, quali:

- **Transfer Learning**: consiste nel cercare di dotare l'agente di mezzi per generalizzare la conoscenza acquisita durante la risoluzione di un determinato task in modo da poterla sfruttare per risolvere altri task differenti ma analoghi (ad esempio conoscenze sulla forma di alcuni oggetti possono permettere di identificare oggetti nuovi) [12].

- **Teacher Advice**: per sopperire a quei task dove il Transfer Learning si riveli insufficiente, si procederà ad implementare un sistema di Teacher Advising, atto a permettere ad un teacher (nello specifico il personale medico specializzato) di fornire esempi e suggerimenti all'agente in modo da direzionare la policy inizia-

le dello stesso verso le aree dello state space che permetteranno di raggiungere le performance migliori, permettendo poi all'agente di iniziare da qui il suo apprendimento [13] [14].

Grazie a tali introduzioni, sarà possibile:

- Migliorare l'adattabilità e l'affidabilità del sistema MS-Rehab, in particolare per far lavorare i pazienti a livello soglia fin dal principio della riabilitazione.
- Testare l'estensione del sistema ad altre patologie tramite l'implementazione delle tecniche specifiche sopracitate.

Bibliografia

[1] S. Bonavita, R. Sacco, M. Della Corte, S. Esposito, M. Sparaco, A. d'Ambrosio, R. Docimo, A. Bisecco, L. Lavorgna, D. Corbo, S. Cirillo, A. Gallo, F. Esposito, G. Tedeschi, *Computer-aided cognitive rehabilitation improves cognitive performances and induces brain functional connectivity changes in relapsing remitting multiple sclerosis patients: an exploratory study*, J. Neurol., 262(1):91-100, Jan 2015.

[2] M. P. Amato, B. Goretti, R. G. Viterbo, E. Portaccio, C. Niccolai, B. Hakiki, P. Iaffaldano, M. Trojano, *Computer-assisted rehabilitation of attention in patients with multiple sclerosis: results of a randomized, double-blind trial*, Mult. Scler., 20(1):91-98, Jan 2014.

[3] A. Cerasa, M. C. Gioia, P. Valentino, R. Nistico, C. Chiriaco, D. Pirritano, F. Tomaiuolo, G. Mangone, M. Trotta, T. Talarico, G. Bilotti, A. Quattrone, *Computer-assisted cognitive rehabilitation of attention deficits for multiple sclerosis: a randomized trial with fMRI correlates*, Neurorehabil Neural Repair, 27(4):284-295, May 2013.

[4] M. Gaspari, F. Zini, D. Castellano, F. Pinardi, S. Stecchi, *An advanced system to support cognitive rehabilitation in multiple sclerosis*, In Proceeding of IEEE RTSI 2017, 3rd International Forum on Research and Technologies for Society and Industry, Modena, Italy, Sep 2017.

[5] J. X. Chen, *The Evolution of Computing: AlphaGo*, the IEEE CS and the AIP, 2016.

[6] R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction*, The MIT Press, 2018.

[7] Javier Garcìa, Fernando Fernàndez, *A Comprehensive Survey on Safe Reinforcement Learning*, Journal of Machine Learning Research 16, 2015.

[8] C. C. Bennett, K. Hauser, *Artificial Intelligence Framework for Simulating Clinical Decision-Making: A Markov Decision Process Approach*, Artificial Intelligence in Medicine, Volume 57, Pages 9-19, 2013.

[9] S. M. B. Netto, V. R. C. Leite, A. C. Silva, A. C. de Paiva, A. de Almeida Neto, *Application on Reinforcement Learning for Diagnosis Based on Medical Image, Reinforcement Learning: Theory and Applications*, I-Tech Education and Publishing, 2008

[10] R. H. Benedict, D. Cookfair, R. Gavett, M. Gunther, F. Munschauer, N. Garg, and B. Weinstock-Guttman, *Validity of the minimal assessment of cognitive function in multiple sclerosis (MACFIMS)*, J Int Neuropsychol Soc, 12(4):549-558, Jul 2006.

[11] F. Stablum, *PASAT: Paced Auditory Serial Addition Task*, Dipartimento di Psicologia Generale, Padova, 2005

[12] M. E. Taylor, P Stone, *Transfer learning for reinforcement learning domains: A survey*, Journal of Machine Learning Research, 10(1):1633-1685, 2009.

[13] A. L. Thomaz, C. Breazeal, *Teachable robots: Understanding human teaching behavior to build more effective robot learners*, Artificial Intelligence, 172(6-7):716-737, 2008.

[14] J. Garcìa, D. Acera, F. Fernàndez, *Safe reinforcement learning through probabilistic policy reuse*, In Proceedings of the 1st Multidisciplinary Conference on Reinforcement Learning and Decision Making, 2013.

[15] Y. Bogdanova, M. K. Yee, V. T. Ho, K. D. Cicerone, *Computerized Cognitive Rehabilitation of Attention and Executive Function in Acquired Brain Injury: A Systematic Review*, Journal of Head Trauma Rehabilitation, 2016.

[16] M. Cavallo, F. Trivelli, M. Adenzato, E. Bidoia, R. M. Giaretto, F. Oliva, L. Ostacoli, A. Sala, R. L. Picci, *Do neuropsychological and social cognition abilities in schizophrenia change after intensive cognitive training A pilot study*, Clinical Neuropsychiatry, vol. 10, no. 5, pp.202-212, 2013.

[17] F. Mattioli, C. Stampatori, F. Bellomi, M. Danni, L. Compagnucci, A. Uccelli, M. Pardini, G. Santuccio, G. Fregonese, M. Pattini, B. Allegri, R. Clerici, A. Lattuada, C. Montomoli, B. Corso, R. Capra, *A RCT Comparing Specific Intensive Cognitive Training to Aspecific Psychological Intervention in RRMS: The SMICT Study*, Front Neurol, vol. 5, p. 278, 2014.

[18] O. Orel, *Cognitive rehabilitation in patients with multiple sclerosis*, The European Charcot Foundation, 24th Annual Meeting, 2016.

[19] M. Hagovskà, P. Takàc, O. Dzvonìk, *Effect of a combining cognitive and balanced training on the cognitive, postural and functional status of seniors with a mild cognitive deficit in a randomized, controlled trial*, Eur J Phys Rehabil Med, 2016.

[20] E. D. de Bruin, E. van Het Reve, K. Murer, *A randomized controlled pilot study assessing the feasibility of combined motorcognitive training and its effect on gait characteristics in the elderly*, Clin Rehabil, 2013.

[21] Neurab srl, Rovereto, http://www.neurotablet.com/.

[22] S. Cardullo, L. Gamberini, S. Milan, D. Mapelli, *Rehabilitation Tool: A Pilot Study On A New Neuropsychological Interactive Training System*, Studies in Health Technology and Informatics, 2015.

[23] J. N. Motter, M. A. Pimontel, D. Rindskopf, D. P. Devanand, P. M. Doraiswamy, J. R. Sneed, *Computerized cognitive training and functional recovery in major depressive disorder: A metanalysis*, Journal of Affective Disorders, 2016.

[24] Joan Serrà, J. L. Arcos, A. Garcia-Rudolph, A. Garcìa-Molina, T. R. Rovira, J. M. Tormos, *Cognitive Prognosis of Acquired Brain Injury Patients Using Machine Learning Techniques*, The Fifth International Conference on Advanced Cognitive Technologies and Applications, 2013.

[25] M. Zhu, Z. Zhang, J. P. Hirdes, P. Stolee, *Using machine learning algorithms to guide rehabilitation planning for home care clients*, BMC Medical Informatics and Decision Making, 2007.