Alma Mater Studiorum - Università degli Studi di Bologna
Sede di Cesena

SCUOLA DI SCIENZE

Corso di Studi in Scienze e Tecnologie Informatiche

STUDIO E IMPLEMENTAZIONE DI UN SISTEMA PER LA PROFILAZIONE BASATO SUL CONTESTO E SUL COMPORTAMENTO DELL'UTENTE

Tesi di Laurea in:
Sistemi operativi

Presentata da:
Alberto Fariselli

Relatore:
Chiar.ma Prof.ssa Paola Salomoni

Correlatore:
Dott.ssa Silvia Mirri

Sessione III
Anno Accademico 2012/2013

INDICE

Introduzione

- L'utilizzo di dispositivi elettronici come strumento per la fruizione di contenuti multimediali e´ in costante aumento. Parallelamente alla loro diffusione e alla loro integrazione nella quotidianita` delle persone, e´ in crescita anche la differenziazione di questi device. Se fino a qualche decennio fa la consultazione di contenuti digitali avveniva solo ed esclusivamente attraverso computer dotati di schermi di dimensioni considerevoli, al giorno d'oggi tale funzione viene offerta da una gamma di prodotti molto piu` ampia e dalle caratteristiche sempre piu` differenti. I calcolatori, grazie all'innovazione tecnologica, hanno assunto forme e dimensioni e caratteristiche sempre piu` varie, successivamente sono comparsi gli smartphone e i tablet i quali, a loro volta, con il passare del tempo hanno ampliato sensibilmente la varieta` delle loro dimensioni, delle loro caratteristiche e delle loro capacità e hanno rivoluzionato il concetto di informazione in mobilita`. Infine, le tecnologie indossabili come orologi e occhiali stanno per fare il loro ingresso nel nostro quotidiano. L'estrema velocita` con la quale questi dispositivi riescono a trasmettere informazione ha innescato una sorta di rivoluzione culturale all'interno della societa`. Esula dallo scopo di questa tesi analizzare i risvolti sociali derivati dalla sempre piu` consistente utilizzo e diffusione di questi dispositivi tra la popolazione, bisogna pero` tenere presente che se da un lato l'innovazione nell'ambito questi dispositivi e´ in costante, non si puo` dire lo stesso per quanto riguarda le opzioni di accessibilita` rivolte a quegli utenti che trovano in molti degli aspetti che costituiscono i contenuti multimediali delle vere e proprie "barriere" digitali. Sono vari, infatti, i casi in cui un utente può trovarsi in

difficolta` nel momento in cui si accinge a fruire di contenuti digitali. Le opzioni di accessibilita` tipicamente fornite dai sistemi operativi e piu` in particolare dai browser di navigazione Web sono molteplici, ma vanno a modificare l'intero contenuto digitale, agendo anche sulle parti che non rappresentano una effettiva barriera. Infatti, un browser per la navigazione Web tipicamente permette di impostare una dimensione minima per i caratteri delle pagine Web, ma questa modifica non andra` ad interessare solo i contenuti ai quali l'utente e´ realmente interessato, avra` anzi effetto globale sugli elementi della pagina. In generale, da un lato i browser non forniscono tutte le opzioni necessarie al soddisfacimento delle preferenze degli utenti, in particolare di quegli utenti che trovano difficolta` nella fruizione di contenuti testuali con determinate caratteristiche, dall'altro lato e´ complicato fornire tutte le opzioni necessarie al fine di rappresentare ogni forma di preferenza da parte dell'utente e altrettanto complicato sarebbe per l'utente definire tali preferenze attraverso le impostazioni se esse consentissero realmente di rappresentare ogni possibile necessità. Nasce, però, una riflessione a partire da quest'ultimo concetto, ovvero: potrebbe essere interessante un sistema che non costringa l'utente ad impostare in modo esplicito le proprie preferenze, ma che le apprenda man mano che l'utente naviga, osservando determinati comportamenti. La ricerca nel campo dell'intelligenza artificiale ha consentito la definizione di particolari algoritmi detti "algoritmi di apprendimento per rinforzo", in grado di comprendere ed adattarsi al contesto nel quale vengono eseguiti. L'applicazione di questi algoritmi potrebbe fornire una valida soluzione per il problema della profilazione utente, almeno nell'ambito delle tecnologie Web. Se l'utente riuscisse a trasmettere ad uno di questi algoritmi le proprie preferenze semplicemente permettendo a tale algoritmo di osservare il suo comportamento, l'algoritmo potrebbe definire al suo interno una propria rappresentazione di queste preferenze e contemporaneamente applicarle al suo posto. In questo modo non ci sarebbe più bisogno di determinare impostazioni necessarie per

l'auto- profilazione del browser e, allo stesso tempo, l'utente non dovrebbe fornire alcuna indicazione al browser. L'obiettivo di questa tesi consiste nella valutazione di un algoritmo di apprendimento per rinforzo utilizzato per la profilazione utente, basata unicamente sul suo comportamento. Per poter effettuare questo studio e successivamente trarre conclusioni concrete sull'efficacia di questi algoritmi di apprendimento, e` necessaria da un lato l'implementazione di un sistema in grado di simulare il comportamento di un utente durante la navigazione Web, dall'altro lato l'implementazione di un contesto all'interno del quale un algoritmo di apprendimento possa essere eseguito, in modo tale da poter osservare il comportamento dell'utente simulato e conseguentemente definire una propria rappresentazione delle sue esigenze. Una volta completate le implementazioni di entrambi i sistemi, il passo successivo consiste nella loro integrazione e nella valutazione di alcuni test. E' fondamentale che il meccanismo di simulazione utente consenta di rappresentare varie tipologie di utente, ovvero che sia in grado di simulare il comportamento di utenti con particolari difficolta` nella fruizione di contenuti testuali, come l'ipovisione o la dislessia per mettere alla prova l'algoritmo. In aggiunta, e´ oggetto di interesse per questa tesi analizzare l'approccio con il quale un algoritmo di apprendimento per rinforzo si pone di fronte a comportamenti particolari e non standard dell'utente. Questi comportamenti anomali non sono relativi ad una particolare configurazione di preferenze, ma fanno riferimento unicamente al modo di agire dell'utente. Ci si puo` infatti aspettare che le scelte effettuate da un utente durante la navigazione Web non siano sempre coerenti fra loro ma al contrario che a volte siano controproducenti ai fini dell'apprendimento. Questo perche´ per qualche ragione l'utente potrebbe non essere interessato ad un particolare tipo di adattamento nonostante egli l'abbia preferito costantemente fino a

quel momento. Allo stesso modo e´ possibile che le esigenze di un utente cambino col passare del tempo, di conseguenza anche le sue preferenze per quanto riguarda la navigazione Web e le caratteristiche degli elementi che compongono le pagine Web. Con questa tesi si intende fornire un quadro generale su come un algoritmo di apprendimento per rinforzo possa rappresentare una soluzione per il problema della profilazione utente basandosi unicamente sul suo comportamento. La tesi e´ suddivisa in tre capitoli principali come segue:

- Nel primo capitolo, “Tecnologie coinvolte”, si approfondisce il concetto di algoritmo di apprendimento per rinforzo attraverso la storia della sua nascita e la sua definizione formale, in seguito si descrive una framework pre-esistente atto alla modellazione di problemi ed esperimenti basati sul supporto fornito proprio da questo tipo di algoritmi ed esso rappresenta la base per l'implementazione del contesto oggetto di questa tesi.
- Nel secondo capitolo, “Integrazione del simulatore ExTraS nel framework PIQLE”, si effettua una descrizione di un simulatore di utenza Web scritto in Java, rivolto alla simulazione di un utenza caratterizzata dalla presenza di difficolta` legata a vari disturbi visivi. Nel capitolo verra` dedicata particolare attenzione al meccanismo di simulazione di utenza effettuata dal simulatore, in quanto essa consiste nella rappresentazione del comportamento tipico di un utente che durante la navigazione Internet effettua determinate modifiche agli elementi delle pagine Web e per questo motivo costituisce un modello perfetto per essere osservato da un algoritmo di apprendimento per rinforzo, al fine di studiarne il comportamento. Successivamente nello stesso capitolo sono descritti i passi seguiti durante la progettazione finalizzata all'implementazione dell'integrazione dei due sistemi. Come descritto, la progettazione e` stata suddivisa in due parti principali. La prima fase si concentra sullo studio del comportamento del simulatore durante la sua esecuzione e sull'individuazione dei momenti significativi di apprendimento per l'algoritmo. La seconda

parte della progettazione invece consiste nello studio finalizzato alla rappresentazione del contesto nel quale eseguire l'algoritmo di apprendimento per rinforzo all'interno del framework. Tale dualità nella progettazione e` rispecchiata anche nell'organizzazione di questo capitolo.

- Nel terzo capitolo infine si descrive l'implementazione effettivamente dell'integrazione tra i due sistemi. Tale descrizione e´ accompagnata dall'esposizione delle parti più importanti del codice, facendo particolare attenzione sui punti critici per l'apprendimento dell'algoritmo. Come e` possibile notare l'organizzazione con la quale e´ affrontata l'esposizione dell'implementazione rispecchia l'organizzazione seguita durante la progettazione, in modo da rendere piu` facilmente interpretabile il codice. Il terzo capitolo termina con la fase di test, nella quale vengono mostrati risultati ottenuti durante l'esecuzione dei due sistemi in comunicazione fra loro, facendo osservazioni relative alle varie configurazioni prese in esame.

Introduzione

1. Tecnologie Coinvolte

1.1 Stato dell'arte

L'utilizzo di dispositivi elettronici per la fruizione di contenuti testuali é in costante aumento ed in aumento sono anche le tipologie e funzionalità di questi dispositivi. Se fino a dieci anni fa tali contenuti potevano essere goduti solo tramite computer standard con schermi di certe dimensioni, al giorno d'oggi troviamo smartphone, tablet e orologi dotati di schermi le cui dimensioni spaziano dai due ai dieci pollici. E' innegabile che la distribuzione di questi dispositivi, unitamente alla diffusione di Internet, abbia innescato una rivoluzione culturale all'interno della società odierna: la ricerca di un informazione non consiste più nella lenta consultazione di tomi o dal passaparola, ma da una semplice Query diretta ad un sistema informatico dalla velocità pressoché istantanea. Tali dispositivi accorciano le distanze e rimpiccioliscono il pianeta nel quale viviamo, poiché la comunicazione viaggia alla velocità della luce e, soprattutto, la quantità di informazione trasmissibile contemporaneamente ha proporzioni enormi in confronto a quella consentita fino a venti anni fa. L'utilizzo massivo della carta che si è fatto fino a non molto tempo fa come veicolo informativo ha sempre posseduto limiti intriseci legati alla sua velocità di diffusione e alla sua capacità di aggiornamento, al contrario i dispositivi elettronici odierni permettono una comunicazione dalla velocità pressoché istantanea anche tra terminali molto distanti fra loro e permettono una rapida e agile revisione dei contenuti. E' dunque evidente come questi dispositivi svolgano una funzione importante per l'uomo e che con il passare del tempo e dei cambi generazionali tale importanza sarà profondamente radicata nella quotidianità delle persone. Esula dallo scopo di questa tesi analizzare gli aspetti socialmente positivi e negativi che una tale permeazione tecnologica possa apportare alla società, tuttavia é bene tenere presente un aspetto fondamentale: una grossa parte della popolazione presenta difficoltà più o meno gravi legati alla vista e/o alla comprensione del testo in generale che gli impediscono di fruire appieno dei contenuti testuali e dei servizi

offerti dai suddetti dispositivi elettronici e sistematicamente gli impediscono di partecipare a questa rivoluzione culturale. Al momento, le varie opzioni di accessibilità fornite dai sistemi operativi e più in particolare dai browser non consentono un'applicazione selettiva delle preferenze dell'utente, al contrario consentono modifiche generiche e insufficientemente elastiche di fronte alla varietà di preferenze degli utenti. Per quanto riguarda i browser, una funzionalità molto diffusa é quella del text-to-speech, modalità in cui il dispositivo elettronico si prende l'incarico di leggere ad alta voce i contenuti delle pagine, ma questa funzionalità non risulta comoda in mobilità, in presenza di altre persone e nei momenti in cui si ha fretta dal momento che la sintesi legge ad una velocità pre-impostata. Un'altra funzionalità è quella che permette di impostare una dimensione minima per i caratteri delle pagine, in questo modo non comparirà mai un testo di dimensione inferiore a quella scelta ma contemporaneamente ogni elemento presente verrà ingrandito. Lo stesso avviene per quanto riguarda i colori degli elementi, è possibile ad esempio far eseguire al browser una sostituzione di colore con un altro più facilmente fruibile, ma anche questa opzione avrebbe effetto su tutti gli elementi dei contenuti. Lo scopo di questa tesi è implementare un sistema software che impari osservando il comportamento dell'utente quali sono le sue preferenze e che con il passare del tempo proponga ed effettui modifiche alle pagine web in modo da fornire pagine esenti da imperfezioni dal punto di vista dell'utente. Un sistema che non abbia bisogno di parametri a priori per funzionare ma che sia in grado di capire da solo cosa debba essere modificato e cosa no. Allo scopo risulta necessario un meccanismo di apprendimento automatico, che sia in grado di apprendere autonomamente quali siano le necessità dell'utente attraverso l'analisi del suo comportamento e di essere capace, una volta ottenuta la giusta quantità di informazioni, di fornire all'utente una pagina pulita, ovvero scevra di ogni barriera digitale per l'utente. Questo meccanismo dovrà non aver bisogno di impostazioni iniziali di alcun tipo e dovrà adattarsi a quanti più casi possibili. Per progettarlo e realizzarlo, nell'ambito di questa tesi, è stato preso in considerazione il concetto di apprendimento per rinforzo.

1.2 L'apprendimento per rinforzo

Prima di definire l'apprendimento per rinforzo è bene tenere in considerazione la definizione generale di apprendimento:

"Acquisizione di un nuovo comportamento indotta dall'esperienza, al fine di un migliore adattamento con l'ambiente" (Nuovissimo Dardano, Dizionario della lingua Italiana, 1982)

Definizione di apprendimento per rinforzo:

L'apprendimento per rinforzo è una tecnica di apprendimento automatico che punta ad attuare sistemi in grado di apprendere ed adattarsi alle mutazioni dell'ambiente in cui sono immersi attraverso la distribuzione di una ricompensa che consiste nella valutazione delle loro prestazioni e che prende appunto il nome di rinforzo. (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998)

La storia dell'apprendimento è divisa in due branche principali, (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998) che si sono sviluppate indipendentemente prima di intrecciarsi nel concetto moderno di apprendimento per rinforzo (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998). Una branca considera l'apprendimento attraverso i concetti di "prova" ed "errore" e trova fondamento nella psicologia dell'apprendimento animale (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998). Questa branca si è sviluppata a partire dai primi progetti di intelligenza artificiale ed ha condotto alla rinascita dell'apprendimento per rinforzo nei primi anni ottanta (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998). L'altra branca considera il problema dell'"Optimal Control" e la sua soluzione sfruttando i valori di funzioni e la programmazione dinamica (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998). In gran parte, questa branca

non include il concetto di apprendimento. Sebbene le due correnti siano state in gran parte indipendenti, le eccezioni in questo senso hanno ruotato attorno ad una terza branca, meno distinta rispetto alle altre due, riguardante i metodi di apprendimento basati sulle differenze temporali. (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998)

1.2.1.1 Optimal Control

Il termine "Optimal Control" è entrato in uso alla fine degli anni cinquanta per descrivere il problema della progettazione di un controllore per minimizzare la misura del comportamento di un sistema dinamico con il passare del tempo (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998). Uno degli approcci a questo problema fu sviluppato nella metà degli anni cinquanta da Richard Bellman e i suoi colleghi (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998) estendendo una teoria del diciannovesimo secolo di Hamilton e Jacobi (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998). Questo approccio sfrutta il concetto di stato di un sistema dinamico e di valore di funzione, o meglio "funzione a risultato ottimale", per definire una equazione funzionale, ora spesso chiamata equazione di Bellman (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998). Questa classe di metodi per la soluzione di problemi di Optimal Control attraverso la soluzione di questa equazione si diffuse con il nome di programmazione dinamica. Bellman introdusse anche la versione stocasticamente discreta del problema di Optimal Control conosciuta come Processi di Decisione di Markov (MDPs) (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998), e Ron Howard nel 1960 progettò il metodo di politica di integrazione per questi ultimi (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998). Questi sono gli elementi essenziali che sottolineano la teoria e gli algoritmi di apprendimento per rinforzo. (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998)

La programmazione dinamica è largamente considerata l'unica maniera fattibile di risolvere problemi stocastici di Optimal Control generici (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998). Essa soffre di quella che Bellman definì "la maledizione della dimensionalità", intendendo la crescita esponenziale dei requisiti computazionali al crescere del numero di variabili di stato, ma è ancora più efficiente e più largamente applicabile di ogni altro metodo (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998). La programmazione dinamica è stata intensamente sviluppata negli ultimi quaranta anni, includendo estensioni ai Processi di Decisione di Markov parzialmente osservabili (1991), metodi di approssimazione (1996) e metodi sincroni (1983). (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998)

Il lavoro svolto sul problema dell'Optimal Control è considerato lavoro per l'apprendimento per rinforzo (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998). L'apprendimento per rinforzo è definito come una maniera effettiva di risolvere i problemi di apprendimento per rinforzo, e risulta chiaro come questi siano strettamente collegati con i problemi di Optimal Control, in particolare ai Processi di Decisione di Markov (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998). Di conseguenza bisogna considerare i metodi risolutivi del problema del Optimal Control, come la programmazione dinamica, anche come metodi di apprendimento per rinforzo (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998). Quasi tutti questi metodi richiedono una completa conoscenza del sistema che deve essere controllato, per questa ragione sembra innaturale considerarli parte dell'apprendimento per rinforzo. D'altra parte, molti metodi di programmazione dinamica sono incrementali e iterativi. Come i veri metodi di apprendimento, essi raggiungono gradualmente la risposta corretta attraverso successive approssimazioni. (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998)

1.2.1.2 Trial-and-Error

Questa branca trova fondamento nella psicologia, dove le teorie del rinforzo nell'apprendimento sono comuni. Probabilmente il primo che espresse brevemente l'essenza dell'apprendimento Trial-and-Error fu Edward Throndike, che nel 1991 descrisse l'idea che azioni seguite da un buono o cattivo risultato abbiano la tendenza ad essere ri-intraprese modificate di conseguenza:

"Tra le molte reazioni basate sulla stessa situazione, quelle che sono accompagnate o seguite molto da vicino da soddisfazione per l'animale, a parità di altre, avranno più probabilità di essere ripetute. Quelle che sono accompagnate o seguite molto da vicino da disagio per l'animale saranno, a parità di altre, connesse con quella situazione di indebolimento, per cui quando se si ripresentassero, avranno meno probabilità di verificarsi. Maggiore è la soddisfazione o il fastidio, maggiore è il rafforzamento o indebolimento del legame". (E. Throndike, 1991)

Thorndike chiamò questo concetto "Legge dell'Effetto" perché descrive l'effetto di rinforzare eventi sulla tendenza a selezionare le azioni. Sebbene a volte sia controversa la Legge dell'Effetto è largamente considerata come un principio base sottostante al comportamento. La Legge dell'Effetto include i due più importanti aspetti di quello che intendiamo per apprendimento per Trial-and-Error. Per prima cosa esso è a selezione invece di essere ad istruzioni, intendendo che comprende il provare alternative e selezionarle facendo confronti sulle loro conseguenze. In secondo luogo, è associativo, nel senso che le alternative trovate dalla selezione sono associate con particolari situazioni. La selezione naturale è un primo esempio di processo di selezione, anche se non è associativa. L'apprendimento per supervisione è associativo ma non è di selezione. Esso è la combinazione di questi due aspetti ed è essenziale alla Legge dell'Effetto e all'apprendimento Trial-and-Error. La Legge dell'Effetto è un modo elementare di combinare ricerca e memoria: la ricerca nella forma di provare e selezionare tra molte azioni in ogni situazione, e memoria nella forma di ricordare quali azioni hanno funzionato meglio, associandole con le situazioni nelle quali erano migliori. Combinare ricerca e memoria in questo modo è essenziale per l'apprendimento per rinforzo.

(Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998)

All'inizio dello sviluppo dell'intelligenza artificiale, prima che fosse distinta da altre branche dell'ingegneria, molti ricercatori cominciarono ad esplorare l'apprendimento Trial-and-Error come un principio di ingegneria. Le prime investigazioni computazionali dell'apprendimento Trial-and-Error furono probabilmente di Minsky, Farley e Clark, tutti nel 1954. Minsky presentò dei modelli computazionali di apprendimento per rinforzo e descrisse la costruzione di una macchina analoga, composta da un componenti che egli chiamò SNARC, (Calcolatori Stocastici a rinforzo neurale). Farley e Clark descrissero un altra macchina basata su rete neurale per l'apprendimento progettata per imparare attraverso il paradigma Trial-and-Error. Negli anni sessanta compaiono per la prima volta all'interno della letteratura ingegneristica i termini "rinforzo" e "apprendimento per rinforzo". (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998)

Particolarmente influente fu il documento presentato da Minsky "Steps Toward Artificial Intelligence", nel 1961, nel quale discusse molti problemi relativi all'apprendimento per rinforzo, includendo quello che chiamò "problema dell'assegnamento del credito": Come distribuire la ricompensa per il successo tra le tante decisioni che potrebbero essere state coinvolte nella sua produzione? Questo fu uno dei più grandi problemi affrontati durante lo sviluppo dell'apprendimento per rinforzo. (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998)

Gli interessi di Farley e Clark nella metà degli anni cinquanta si spostarono dall'apprendimento per Trial-and-Error alla generalizzazione e identificazione di modelli, dall'apprendimento per rinforzo all'apprendimento supervisionato. Questo portò confusione sulla relazione tra i due tipi di apprendimento. (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998)

Influente fu il lavoro di Donald Michie. Nel 1961 e nel 1963 descrisse un semplice sistema di apprendimento per Trial-and-Error per imparare a giocare a TicTacToe, il tris. (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998)

Nel 1975 John Holland delineò una teoria generale sui sistemi adattabili basati sui principi di selezione. Il suo precedente lavoro riguardava principalmente il concetto di Trial-and-Error nella sua forma non associativa. Nel 1986 introdusse i Sistemi Classificatori, sistemi di apprendimento per rinforzo comprendenti l'associazione e i valori di funzioni. Un componente chiave dei sistemi classificatori di Holland era sempre un algoritmo genetico, un metodo evolutivo il cui ruolo era quello di sviluppare rappresentazioni utili. I sistemi classificatori sono stati ampiamente sviluppati da molti ricercatori per formare una branca superiore nella ricerca degli algoritmi per rinforzo, ma gli algoritmi genetici, i quali di per se non sono sistemi di apprendimento per rinforzo, hanno ricevuto molta più attenzione. (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998)

Fondamentale per la rivalutazione del Trial-and-Error nell'apprendimento per rinforzo nell'ambito dell'intelligenza artificiale è Harry Klopf. (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998) Klopf riconobbe che gli aspetti essenziali dei comportamenti adattivi venivano persi dal momento che i ricercatori si concentrarono quasi esclusivamente sull'apprendimento supervisionato. (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998) Quello che mancava, secondo Klopf, (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998) erano gli aspetti edonistici del comportamento, la spinta a raggiungere qualche risultata dall'ambiente, per condurre l'ambiente verso estremità desiderate e lontano dalle estremità indesiderate. Questa è l'idea essenziale dell'apprendimento per rinforzo. Le idee di Klops furono in particolar modo influenti sugli autori perchè il loro apprezzamento li condusse alla

distinzione tra apprendimento per rinforzo e supervisionato, dal momento che gran parte del lavoro dei ricercatori era impiegato nel dimostrare che i due tipi di apprendimento erano molto diversi tra loro (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998). Altri studi dimostrarono come l'apprendimento per rinforzo poteva affrontare importanti problemi nell'apprendimento con reti neurali, in particolare riguardo la produzione di algoritmi di apprendimento per reti multi-strato. (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998)

1.2.1.3 Temporal Difference

I metodi di apprendimento Temporal-Difference si distinguono dagli altri metodi essendo guidati dalle stime temporalmente successive della stessa quantità.(Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998) Questi metodi è più contenuta e meno distinta rispetto alle altre due, ma ha svolto un ruolo particolarmente importante nel campo, in parte perché sembrano essere nuovi e unici nel campo dell'apprendimento per rinforzo (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998).

Le origini dell'apprendimento Temporal-Difference sono in parte legate alla psicologia di apprendimento animale, in particolare nella notizione di "Rinforzi Secondari" (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998). Un rinforzo secondario è uno stimolo che viene accoppiato ad un rinforzo primario come il nutrimento o il dolore e di conseguenza ha assunto proprietà simili di rinforzo. Minsky nel 1954 potrebbe essere stato il primo ad accorgersi che questo principio psicologico sarebbe potuto essere importante per i sistemi di apprendimento artificiale. Arthur Samuel nel 1959 fu il primo a proporre e implementare un metodo di apprendimento che includeva le idee di Temporal-Difference, come parte del suo celebre programma sul gioco

della Dama (Richard S. Sutton, Barto, Andrew G. , “Reinforcement Learning, an introduction”, 1998). Samuel non fece riferimento al lavoro di Misky o a possibili connessioni con l'apprendimento animale (Richard S. Sutton, Barto, Andrew G. , “Reinforcement Learning, an introduction”, 1998). La sua ispirazione provenne apparentemente dal suggerimento di Claude Shannon negli anni cinquanta che un computer potesse essere programmato per giocare al gioco degli scacchi sfruttando una funzione di valutazione e che fosse capace di migliorare il suo gioco modificando la sua funzione in diretta (Richard S. Sutton, Barto, Andrew G. , “Reinforcement Learning, an introduction”, 1998). (E' possibile che questa idea di Shannon abbia influenzato anche Bellman, ma non c'è prova evidente di questo). Minsky nel 1961 discusse ampiamente il lavoro di Samuel nel documento “Steps”, suggerendo la connessione con le teorie di rinforzo secondario, animali e artificiali (Richard S. Sutton, Barto, Andrew G. , “Reinforcement Learning, an introduction”, 1998).

Nel 1972 Klopf unì all'apprendimento Trial-and-Error un importante componente dell'apprendimento Temporal-Difference (Richard S. Sutton, Barto, Andrew G. , “Reinforcement Learning, an introduction”, 1998). Klopf era interessato a principi di apprendimento per sistemi di proporzione molto elevata, e così si è interessato sulle nozioni di rinforzo locale, per le quali i sottocomponenti di un sistema di apprendimento globale potrebbero rafforzarsi reciprocamente. Egli sviluppò l'idea di “rinforzo generico”, per la quale ogni componente (nominalmente ogni neurone) del sistema vede i propri input in termini di apprendimento, input eccitatori come ricompense e input inibitori come punizioni (Richard S. Sutton, Barto, Andrew G. , “Reinforcement Learning, an introduction”, 1998). Questa non rispecchia l'idea odierna di apprendimento Temporal-Difference, in retrospettiva è distante dal lavoro di Samuel, tuttavia il lavoro di Klopf associò l'idea di apprendimento Trial-and-Error con l'enorme database empirico della psicologia di apprendimento animale (Richard S. Sutton, Barto, Andrew G. , “Reinforcement Learning, an introduction”, 1998).

Sutton nel 1978 sviluppò ulteriormente le idee di Klopf, in particolare i collegamenti con le teorie di apprendimento animale, descrivendo le regole di apprendimento guidate dai cambiamenti in predizioni temporalmente successive (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998). Sutton e Barto raffinarono queste idee e svilupparono un

modello psicologico classico basato sul condizionamento dell'apprendimento per rinforzo (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998). Seguirno molti altri influenti modelli psicologici tra la fine degli anni Ottanta e l'inizio degli anni Novanta, alcuni modelli di neuroscienza furono sviluppati in quel periodo da Hawkins e Kandel, da Byrne, Gelperin, Tesauro e Friston. sebbene in molti casi non fosse una connessione storica fra loro. Una recente sintesi dei legami tra l'apprendimento Temporal-Difference e i principi di Neuroscienza è stata fornita da Scultz, Dayan e Montague nel 1997 (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998).

Infine, le due branche di apprendimento Temporal-Difference e di Optimal-Control furono uniti assieme nel 1989 con lo sviluppo del Q-Learning da parte di Chris Watkins. Questo lavoro estese e integrò i precedenti lavori svolti nelle tre correnti principali della ricerca sull'apprendimento per rinforzo (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998). Paul Werbos contribuì anch'esso in questa integrazione sostenendo la convergenza dell'apprendimento Trial-and-Error e della programmazione dinamica sin dal 1977. Al tempo del lavoro di Watkins ci fu una grande espansione nella ricerca sull'apprendimento per rinforzo, principalmente nel sotto-campo di apprendimento automatico dell'intelligenza artificiale, ma anche nelle reti neurali e nell'intelligenza artificiale più in generale. Nel 1992 il notevole successo del programma sul gioco del Backgammon di Gerry Tesauro portò molta attenzione sul settore e con il passare del tempo sono stati forniti contributi sempre più numerosi e importanti (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998).

1.3 Il Q-Learning

Come si può evincere dalla storia della ricerca sull'apprendimento per rinforzo, l'algoritmo Q-Learning sviluppato da Chris Watkins nel 1989 rappresenta un punto di svolta nella ricerca del settore. (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998)Al giorno d'oggi il Q-Learning è uno dei più conosciuti algoritmi di apprendimento per rinforzo e uno dei suoi maggiori punti di rilievo consiste nell'abilità di comparare l'utilità aspettata dalle azioni disponibili senza richiedere un modello dell'ambiente.

Il suo obiettivo è quello di permettere ad un sistema di apprendimento automatico di adattarsi all'ambiente che lo circonda migliorando la scelta delle azioni da eseguire. Per giungere a questo obiettivo cerca di massimizzare il valore della successiva ricompensa.

1.3.1 Il modello

Il modello del problema può essere descritto da un Agente, un insieme di dati S e un insieme di azioni per stato A.
Effettuando un'azione appartenente all'insieme delle azioni possibili per stato l'agente si muove da uno stato ad un altro stato. Ogni stato fornisce all'agente una ricompensa (un numero reale o naturale). L'obiettivo dell'agente è quello di massimizzare la ricompensa totale. L'agente in questo modo apprende passo per passo quali sono le azioni ottimali associate ad ogni stato.

L'algoritmo è provvisto di una funzione per calcolare la Qualità di una certa coppia stato-azione da cui prende il nome l'algoritmo:

Prima che l'apprendimento inizi, la funzione Q restituisce un valore fisso, scelto dal progettista. Successivamente ogni volta che l'agente riceve una ricompensa vengono calcolati nuovi valori per ogni combinazione stato-azione. Il cuore dell'algoritmo fa uso di un processo iterativo di aggiornamento e correzione

basato sulla nuova informazione (Richard S. Sutton, Barto, Andrew G. , "Reinforcement Learning, an introduction", 1998).

1.3.2 Formula

La seguente formula descrive l'algoritmo Q-Learning:

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{vecchio valore}} + \underbrace{\alpha_t(s_t, a_t)}_{\text{learning rate}} \times \overbrace{\left[\underbrace{R_{t+1}}_{\text{ricompensa}} + \underbrace{\gamma}_{\text{fattore di sconto}} \underbrace{\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})}_{\text{valore futuro massimo}} - \underbrace{Q(s_t, a_t)}_{\text{vecchio valore}}\right]}^{\text{valore appreso}}$$

dove

R_{t+1} é una ricompensa ottenuta dopo avere eseguito l'azione a_t in s_t.

Il tasso di apprendimento o *learning rate* è identificato da $a_t(s,a)$ con a tale che

$0 < a < 1$.

Il valore del fattore di sconto γ è tale che $0 < \gamma < 1$.

Un episodio dell'algoritmo termina quando lo stato s_{t+1} è uno stato è detto "finale" o "di assorbimento".

In tutti gli stati finali il valore s_f, $Q(s_f, a)$ non viene mai aggiornato, di conseguenza conserva il suo valore iniziale.

Il tasso di apprendimento determina con quale estensione le nuove informazioni acquisite sovrascriveranno le vecchie informazioni. Un fattore con valore '0'

impedirebbe all'agente di apprendere, al contrario un fattore con valore '1' indirizzerebbe l'agente a considerare solo le informazioni recenti.

Il fattore di sconto determina l'importanza delle ricompense future. Un fattore di sconto pari a '0' rende l'agente "opportunista", indirizzerebbe cioè l'agente a considerare solo le ricompense attuali, mentre un fattore con valore tendente a '1' consente all'agente di considerare anche le ricompense che riceverà in futuro a lungo termine.

L'algoritmo Q-Learning può essere implementato normalmente attraverso tabelle di memorizzazione come le HashMap. Tuttavia questo approccio perde di consistenza al crescere del livello di complessità del sistema, per cui può essere implementato anche attraverso le reti neurali artificiali.

1.4 PIQLE

Il framework scelto per implementare l'algoritmo d'apprendimento Q-Learning nell'ambito di questo lavoro di tesi é PIQLE (“PIQLE, a platform for Implementation of Q-Learning Experiments”, 2006), proprio con lo scopo di implementare e testare algoritmi di Reinforcement Learning applicati a diversi contesti e problemi noti. L'apprendimento per rinforzo dunque é una tecnica di apprendimento indirizzata agli automi che in qualche modo si ispira alla tecnica di educazione puramente umana indirizzata alla propria prole che prevede un premio o una punizione a seconda dei casi. Il concetto di giusto e sbagliato dunque deve essere codificato, e questo è l'obiettivo del framework PIQLE.

1.4.1 Organizzazione generale

Il framework si basa su tre entità:

Ambiente: rappresenta l'universo del problema, come la fisica o le regole di un gioco, i differenti metodi di descrivere stati e azioni.

Algoritmo: elemento software in grado di scegliere la successiva azione da eseguire. All'interno del framework, gli algoritmi sono chiamati Selettori (selector). Gli algoritmi interessati sono quelli capaci di imparare.
Agente: Costituisce l'interfaccia (non nel senso di Java) tra l'ambiente e l'algoritmo.

Al di sopra di queste entità, gli Arbitri schedulano la successione delle fasi come segue:

- L'ambiente dice all'agente in quale stato si trova in quel momento e fornisce la lista di possibili azioni (Figura 1.1)
- L'agente comunica questa informazione al suo algoritmo (Figura 1.2)
- L'algoritmo sceglie l'azione da eseguire, e da la sua risposta all'agente. (Figura 1.3)

- L'agente dice all'ambiente quale azione vuole che sia eseguita (Figura 1.4)

- L'ambiente esegue il nuovo stato, la ricompensa, e la nuova lista di possibili azioni, e manda queste informazioni all'agente (Figura 1.4)

- L'agente trasmette tutto all'algoritmo, che ora può imparare dall'esperienza, siccome ha ricevuto una ricompensa per la sua scelta precedente, sceglie la successiva azione da eseguire, e il ciclo ricomincia. (Figura 1.5).

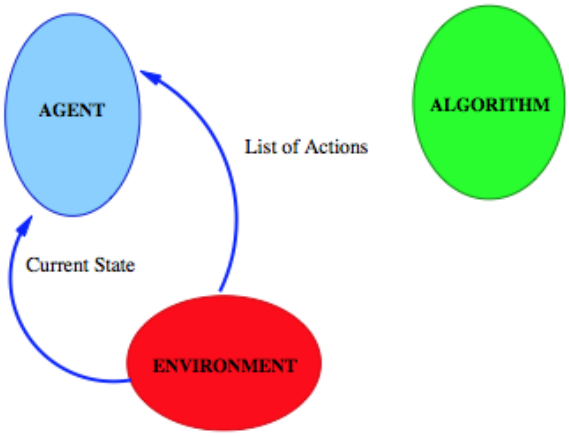


Figura 1.1, L'agente impara il suo stato

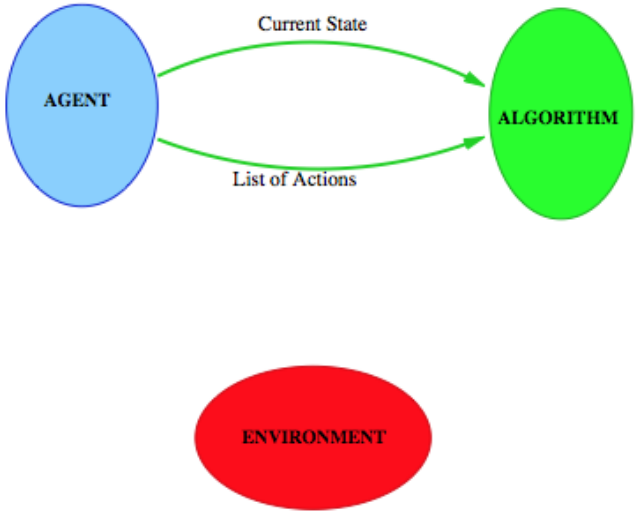


Figura 1.2, L'algoritmo conosce il suo stato corrente

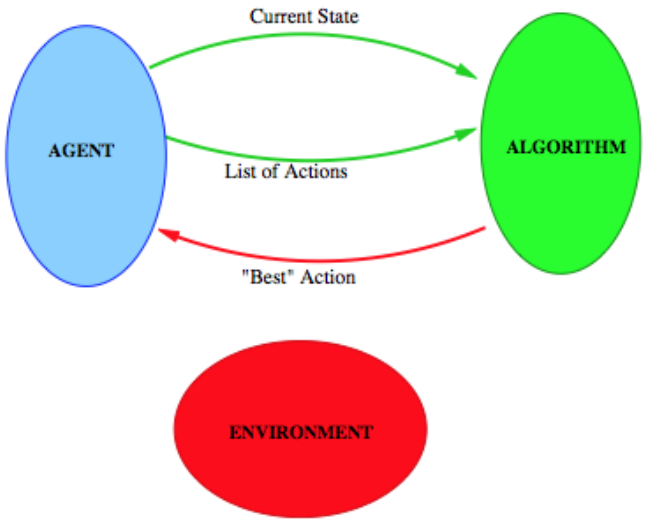


Figura 1.3, L'algoritmo sceglie l'azione

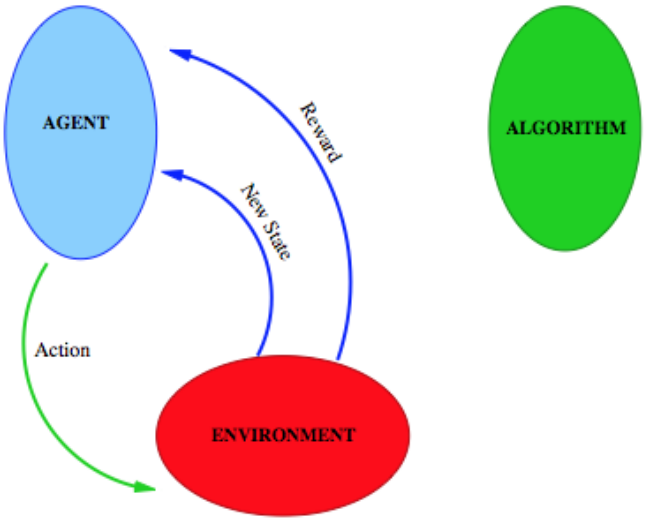


Figura 1.4, l'ambiente genera la ricompensa e il nuovo stato

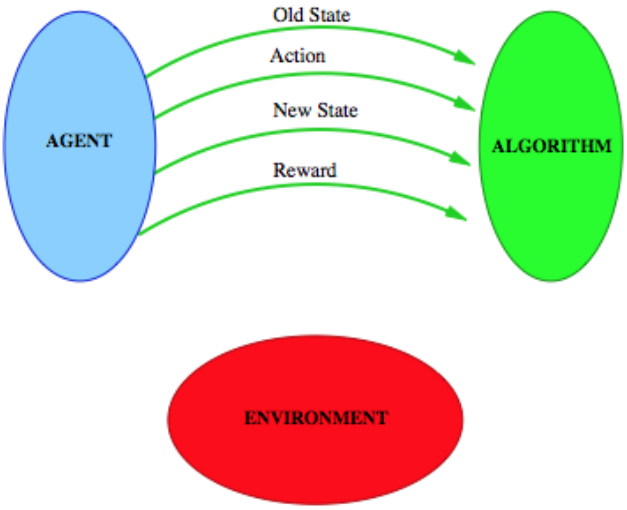


Figura 1.5, l'algoritmo può imparare

Gli agenti sono solo un'interfaccia tra gli ambienti e gli algoritmi: ricevono e trasmettono soltanto informazioni: non hanno bisogno di sapere esattamente in quale universo si stanno muovendo.

Agli algoritmi è richiesto solo di scegliere tra una lista di possibili azioni e ricevono una ricompensa per la loro ultima scelta. La cosa che devono ricordarsi o imparare è grosso modo: “Quale sarà la mia prossima ricompensa se scelgo questa azione?”

Questo schema di ragionamento può essere fatto indipendentemente dall'ambiente, siccome l'algoritmo principalmente deve immagazzinare e restituire la sua passata esperienza come tripletta (stato, azione, ricompensa). Gli ambienti possono essere descritti come entità generiche e astratte. Perciò l'unico luogo in cui si dovrà realmente descrivere il problema che si vuole risolvere è l'istanziazione delle classi generiche del package dell'ambiente.

Queste osservazioni conducono direttamente all'organizzazione Java di PIQLE:

3 principali `package`, indipendenti dal problema da risolvere: `agents`, `referees`, `algorithms`.
1 pacchetto "`environment`" generico, per il comportamento di qualsiasi ambiente rappresentabile. Per ogni problema, un nuovo `package` per istanziare il `package` "`environment`".

Attorno a questi `package` principali, alcune utility saranno raccolte in `package` secondari. Alcune future ristrutturazioni potrebbero spostare questi `package` in posti più accurati, rendendoli sotto- `package` di quelli principali.

Nel seguito di questo capitolo, saranno descritti questi package in modo dettagliato.

1.4.2 Agenti

Gli agenti hanno due ruoli strettamente correlati:

- Permettere all'ambiente e all'algoritmo di comunicare tra loro.
- Comunicare esso stesso con l'arbitro, in modo da schedare ogni episodio.

Un osservazione generale della gerarchia del `package` "`agents`" è mostrata nella Figura 6.

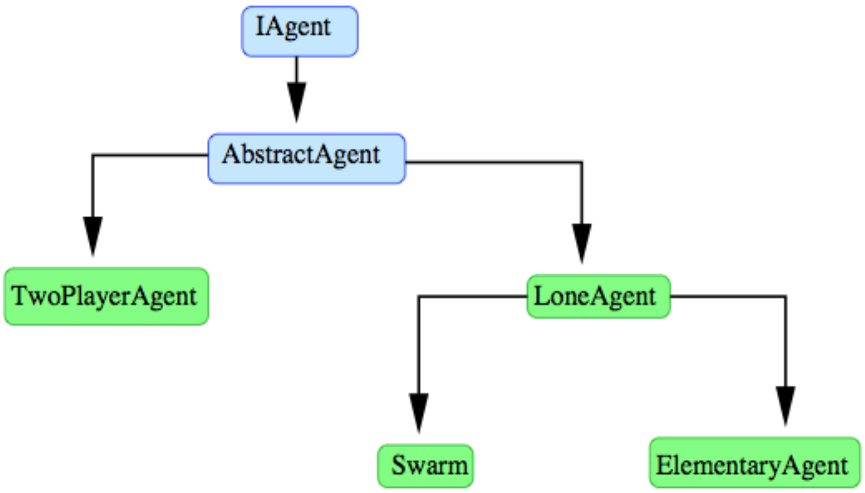


Figura 1.6, gerarchia del package `agents`

L'interfaccia di base `IAgent` è riportata nel Codice 1:

```
public interface IAgent {
public IStrategy getAlgorithm();
public IEnvironment getEnvironment();

public void enableLearning();
public void freezeLearning();

public IState getCurrentState();
public void setInitialState(IState s);

public double getLastReward();

public IAction act();

public void explainValues();

public Dataset extractDataset();

public void saveAgent();
public void saveAgent(String fileName);

public void newEpisode();
}
```

Codice 1.1, L'interfaccia `IAgent`

Le funzionalità principali richieste dall'interfaccia sono:

Nel momento in cui l'agente fornisce un'interfaccia tra un algoritmo e un ambiente, deve contenere due campi per quelle due entità.
I successivi due metodi, `enableLearning()` e `freezeLearning()`, controllano il comportamento di apprendimento dell'agente: uno può chiedere all'agente di smettere di imparare in una certa fase, e vedere come questo agente si comporta abbastanza intelligentemente.

- Il metodo `getLastReward()` chiede all'algoritmo (in cascata, attraverso lo stato attuale) la ricompensa corrispondente all'ultima azione dell'agente. Questo è usato dall'arbitro per processare la ricompensa totale di un episodio.
- Il metodo `act()` è il nucleo del comportamento dell'agente: qui l'agente chiederà al suo algoritmo l'azione migliore da eseguire, e restituire l'azione scelta all'arbitro (il quale poi comunicherà questa scelta all'ambiente).
- Il metodo `explainValues()` emette i valori `Q(s,a)` imparati: è compito dello programmatore scoprire, da questi dati, quale è l'agente che ha realmente imparato.
- Il metodo `extractDataset()` è molto simile a `explainValues()`: emette i valori `Q(s,a)` in un formato adatto per l'analisi da parte della rete neurale di PIQLE.
- Nel caso ci siano alcune impostazioni da essere fatte all'inizio di un episodio (come ad esempio resettare la ricompensa, resettare l'algoritmo,..) l'interfaccia fornisce il metodo `newEpisode()`.

Direttamente sopra l'interfaccia `IAgent`, la classe astratta `AbstractAgent` definisce il codice di quasi tutti il metodi definiti nell'interfaccia. I metodi statici `readAgent()`

devono essere scritti in questa sezione, siccome è necessario sapere se si sta parlando di un `LoneAgent` o di un `TwoPlayerAgent`.

La classe di base per un agente di gioco a giocatore singolo è `LoneAgent`. Esso è semplicemente un `AbstractAgent` con l'unico vincolo di avere un ambiente associato `IEnvironmentSingle`. Esso così manipola stati di classe `AbstractState`.

1.4.3 Ambienti

All'interno del `package` environment ci sono 3 principali entità:

- *Ambiente*: Tutto ciò che è necessario per processare il nuovo stato e la ricompensa, indicando quando il gioco è finito e eventualmente, chi ha vinto.
- *Azione*: E' in grado di generare ogni azione per un dato problema, strumenti per la comparazione, azioni per copia e codifica.
- *Stato*: stati di codice, comparazione, copia. Metodi ausiliari per accedere l'ambiente associato.

1.4.3.1 Ambiente

Tutti gli ambienti devono implementare l'interfaccia IEnvironment mostrata nel Codice 2, questa interfaccia definisce il comportamento atteso di ogni ambiente istanziato.

```
public interface IEnvironment extends Serializable{
public ActionList getActionList(IState s);
public IState successorState(IState s,IAction a);
public double getReward(IState s1,IState s2,IAction a);
public boolean isFinal(IState s);
public int whoWins(IState s);
```

Codice 1.2, L'interfaccia `IEnvironment`

Le funzionalità principali richieste dall'interfaccia sono:
dato uno stato dell'ambiente, elencare quali azioni sono possibili.
dato uno stato e un'azione, processare il prossimo stato.
processare una ricompensa dal precedente stato, lo stato corrente, l'azione presa (come è usuale nell'apprendimento per rinforzo)

Direttamente sopra questa interfaccia ne sono definite altre due:

- `IEnvironmentSingle` per gli ambienti dei giochi a un giocatore.
- `IEnvironmentTwoPlayers` per i giochi a due giocatori.

Il bisogno di differenziare questi due casi deriva dal fatto che vogliamo definire un metodo che restituisca lo stato iniziale dell'ambiente, e gli stati sono classi differenti se implementiamo il problema a uno o due giocatori.

1.4.3.2 Azione

Le azioni sono probabilmente le classi più semplici in PIQLE. Le uniche operazioni che dobbiamo essere in grado di eseguire sono le seguenti: creazione, comparazione per uguaglianza, codifica per l'apprendimento, memorizzazione da parte degli algoritmi.
La definizione di base dell'interfaccia `IAction` è descritta nel Codice 3.

```
public interface IAction extends Cloneable,Serializable{
public Object copy();
public int nnCodingSize();
public double[] nnCoding();
public boolean isNullAction();
public int hashCode();
public boolean equals(Object o);
```

Codice 1.3, L'interfaccia di `Iaction`

Le funzionalità principali richieste dall'interfaccia sono:

- `copy()`, crea una nuova Azione che è la copia della precedente: usata quando si immagazzina le coppie (stato, azione).
- `nnCoding()`, `nnCodingSize()`, per definire una codifica di un azione come un vettore a valori reali, da usare in algoritmi basati su reti neurali.
- `hashCode()`, `equals()`. E' compito del programmatore definire quando due azioni sono dichiarate uguali. Per evitare comportamenti sperimentali inaspettati, bisogna assicurarsi di ridefinire sempre entrambi i metodi quando si istanzia un nuovo problema.
- `isNullAction()`, per alcuni problemi, un giocatore può non avere alcuna azione possibile, e deve passare il turno, mentre il gioco non è ancora finito. La `NullAction()` deve essere un'azione speciale, o una proprietà di un azione.

`ActionList` è una classe ausiliaria per immagazzinare la lista delle possibili azioni da un dato stato. La lista è data dall'algoritmo il quale fa la scelta fra gli elementi (azioni) della lista.

1.4.3.3 Stato

Uno stato deve definire accuratamente una data configurazione dell'ambiente: in particolare, bisogna essere in grado di decidere se due stati di un ambiente sono uguali, o equivalenti.

L'interfaccia `IState` è descritta nel Codice 4.

```
public interface IState {
public void setEnvironment(IEnvironment c);
public IEnvironment getEnvironment();

public ActionList getActionList();
public IState modify(IAction a);
public double getReward(IState old, IAction a);
public boolean isFinal();

public IState copy();
public int nnCodingSize();
public double[] nnCoding();
public int hashCode();
public boolean equals(Object o);
}
```

Codice 1.4, L'interfaccia `Istate`

Le funzionalità principali richieste dall'interfaccia sono:

- `setEnvironment()`, `getEnvironment()`: connette lo stato con l'ambiente al quale appartiene.
- `getActionList()`, `modify()`, `getReward()`, `isFinal()`, semplicemente chiamano i corrispondenti metodi nella classe Ambiente.
- `copy()`, per clonare uno stato, principalmente usata quando si immagazzina lo stato dentro una HashMap.
- `nnCodingSize()`, `nnCoding()` definiscono il formato della rappresentazione dello stato per l'uso in algoritmi basati su reti neurali.
- `hashCode()`, `equals()` lo stesso come descritto sopra per `IAction`.

La classe `AbstractState` definisce solo il codice dei primi 4 metodi visti sopra, quelli che connettono lo stato con il suo ambiente. Esso setta anche il campo corrispondente a quell'ambiente. Gli agenti, e così gli algoritmi, potrebbero non essere in grado di percepire tutti i dettagli di uno stato. Questo è riflesso nel fatto

che gli stati potrebbero contenere campi che sono usati dall'ambiente per processare un nuovo stato, mentre quei campi non sono usati ne per l'esecuzione dei metodi hashCode() e equal(), ne per la definizione di nnCoding().

1.4.4 Algoritmi

Un algoritmo è definito per ogni `IAgent`: il suo ruolo primario è di scegliere un'azione da eseguire, data una lista di possibili azioni.
Buoni algoritmi sceglieranno l'azione migliore. Gli algoritmi di apprendimento useranno la loro esperienza passata per migliorare la loro scelta, sperando di scegliere più spesso le azioni migliori.
Perciò gli algoritmi in PIQLE implementeranno principalmente:

- Un metodo per scegliere un'azione all'interno di una lista di azioni.
- Un metodo per imparare, dato uno stato iniziale e l'azione scelta, il nuovo stato dopo aver applicato l'azione e il premio dato dall'ambiente per questa scelta di azione: questo è il paradigma standard dell'apprendimento per rinforzo.

Alcuni algoritmi potrebbero non imparare, altri potrebbero scegliere le loro azioni a caso, dipendentemente dalle implementazioni di questi due metodi. Gli algoritmi hanno molti modi per immagazzinare la loro esperienza passata, e anche differenti modi per usare questa esperienza.
Gli algoritmi sono contenuti in `package` e sono organizzati secondo una gerarchia a livelli multipli. Questa gerarchia può aiutare a definire variazioni di algoritmi esistenti come nuove classi, o trovare la giusta collocazione ad un nuovo algoritmo.

Quello che un algoritmo deve fare é:

- Scegliere un azione: questa parte del suo comportamento è contenuta nell'interfaccia di base `IStrategy`.
- Imparare: questo è il metodo principale dell'interfaccia `IStrategyLearner`. Il metodo `newEpisode()` è stato collocato in questo punto e dovrebbe eseguire tutte le inizializzazioni necessarie all'inizio di un episodio.

Questi tre metodi sono chiamati dall'`IAgent` al quale l'algoritmo appartiene. Potrebbe essere utile capire o analizzare cosa un algoritmo ha imparato fino a quel momento. Per questi algoritmi che immagazzinano le loro esperienze, questo significa devono essere in grado di restituire e analizzare questi valori. Due metodi possono aiutare:

- Personalizzando il metodo toString().
- Estraendo un dataset preformattato da sfruttare da una rete neurale: questo è il perché l'interfaccia `ISelector` definisce il metodo `extractDataset()`.

Possiamo ora descrivere facilmente la cima della gerarchia degli algoritmi:

- Un'interfaccia di base che definisce il metodo `getChoice`: `Istrategy`.
- Un'altra interfaccia di base che definisce `learn()` e `newEpisode()`: `IstrategyLearner`.
- Una terza interfaccia che raggruppa le precedenti due, aggiungendo il metodo `extractDataset`: `ISelector`.

Tutti gli algoritmi definiti in PIQLE sono implementazioni dell'interfaccia `ISelector`, alcuni attraverso una cascata di classi intermedie, formando la sotto-gerarchia di algoritmi di apprendimento per rinforzo. Una visione completa degli algoritmi di PIQLE è mostrata nella Figura 7:

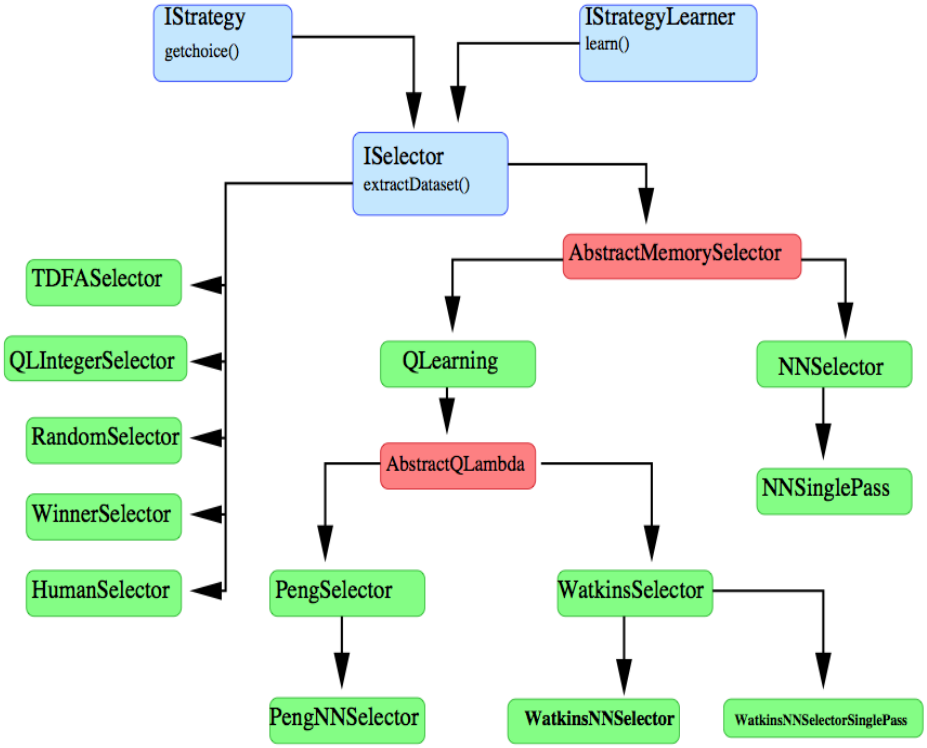


Figura 1.7, gerarchia degli algoritmi

1.4.5 Descrizione di alcuni algoritmi

Nel seguito di questo paragrafo saranno descritti alcuni algoritmi già presenti in PIQLE.

1.4.5.1 Random Selector

Questo algoritmo restituisce un'azione prelevata a caso nella lista di azioni disponibili. Questo algoritmo non impara, non ha bisogno di re-inizializzare niente all'inizio di un episodio, e non è in grado di restituire cosa potrebbe aver imparato.

RandomSelector è tuttavia utile nei seguenti casi:

Per testare nuovi algoritmi, problemi appena codificati: Infatti i bug possono venire solo da algoritmi di apprendimento o codifica di problemi.

Per abbassare il margine di riferimento: un algoritmo che si comporta peggio di un RandomSelector é un problema rilevante.

Per agire come un compagno per allenarsi in un gioco a due giocatori.

1.4.5.2 WinnerSelector

Molto simile al RandomSelector, a parte che se c'è un'azione che rende l'agente il vincitore del problema, l'azione è presa. Si tratta di un algoritmo semplice, ma più impegnativo come compagno di allenamento.

1.4.5.3 HumanSelector

Avendo un agente che presumibilmente ha imparato a giocare a un certo gioco, potrebbe essere interessante testarlo personalmente, giocando contro se stessi. Questo é l'obiettivo di HumanSelector: Questo oggetto è semplice come il RandomSelector e il WinnerSelector: non impara, e ha solo un metodo `getChoice()`. Questo metodo g`etChoice()` stampa una lista di azioni disponibili sullo schermo e chiede al giocatore umano di scegliere il suo movimento nella lista. Questo algoritmo non efficace ed efficiente per l'apprendimento, siccome rallenta tutto il processo.

1.4.6 Gerarchia dell'apprendimento per rinforzo

L'apprendimento per rinforzo, nell'ambito del framework PIQLE, può essere descritto come segue:

1. L'algoritmo riceve dall'agente uno stato e una lista di possibili azioni.
2. L'algoritmo stima il valore di ogni possibile azione, in accordo con il valore `Q(s, a)` che ha immagazzinato o processato.
3. L'algoritmo restituisce l'azione che egli crede produrrà la ricompensa migliore.
4. Dopo che l'azione è stata intrapresa, e lo stato dell'ambiente è stato aggiornato, l'algoritmo aggiorna la sua stima `Q(s, a)`.

Questo conduce alle seguenti osservazioni preliminari:

A. E' necessaria una classe che possa immagazzinare e restituire i valori indicizzati con una coppia (stato, azione).
B. E' necessario sapere quando due stati (e due azioni) sono uguali (per poter applicare il punto 1).
C. Non é necessario stimare valori per le coppie `(state,action)` che non abbiamo mai incontrato.

Tutto quello di cui si ha bisogno per rendere possibile il punto B è già stato fatto durante la definizione dei metodi `equals()` e `hashCode()` di `IState` e `IAction`. Il punto C dice che eviteremo di usare grandi array vuoti per immagazzinare valori `Q(s,a)`.
Nel seguito saranno descritti gli algoritmi di apprendimento per rinforzo della gerarchia di classe, cominciando con la classe astratta che raggruppa la parte più importante della struttura e del codice.

Abstract Memory Selector

Questa classe astratta definisce:
Una memoria: il posto in cui il valore `Q(s,a)` sarà memorizzato.
Un metodo per scegliere la successiva azione da eseguire. Attualmente 3 metodi, dipendendo dalla strategia usata per mantenere l'idea di casualità nella scelta.
Parametri per calibrare l'algoritmo.

Nel seguito sono descritti questi elementi.

In questa classe, il tipo esatto di memoria che usiamo è lasciato astratto, e sara istanziato differentemente in ogni sua implementazione. Questa memoria deve implementare l'interfaccia `IRewardStore`, dal pacchetto "`qlearning`", il che significa che gli è richiesto di essere in grado di:
Restituire un valore associato con una coppia (stato, azione): metodo `put()`.
Immagazzinare un valore associato con un valore (spazio, azione): metodo `get()`.

ActionStatePair

Quando immagazziniamo valori `Q(s,a)` nelle HashMap, bisogna definire una chiave. Nelle impostazioni Q-Learning, la chiave naturale è la coppia (stato, azione) associata con questo valore `Q(s,a)`. Perciò definiamo un nuovo oggetto, `ActionStatePair` che raggruppa in un singolo oggetto un `IAction` e un `IState`.
I metodi `hashCode()` e `equals()` sono ridefiniti per questa classe, dalla definizione di metodi similari nel corrispondente stato e azione. Niente deve essere fatto in questa classe, ma è importante definire accuratamente `hashCode()` e `equals()` quando si istanzia un problema: la nozione di uguaglianza che si definisce per stati e azioni sarà quella che verrà usata per immagazzinare e restituire i valori `Q(s,a)`.

Ricordare le ricompense

Ogni classe designata a immagazzinare o processare valori `Q(s,a)` deve implementare l'interfaccia

`IRewardStore`, che definisce tre metodi:

- `get()`: quando un valore è già immagazzinato, restituisce il suo valore. Altrimenti, ritorna il valore di default.
- `put()`: immette una ricompensa nel meccanismo di immagazzinamento, indicizzato con stato e azione.
- `extractDataset()`: trasforma i valori `Q(s,a)` in un `dataset` adatto per essere usato con le reti neurali di PIQLE. Questo metodo non è sempre implementato, e può tornare il valore `NULL`.

Ci sono 4 modi di immagazzinare valori di ricompense attesi in PIQLE in questo momento:

- Usando le HashMap (per numeri reali o interi)

- Usando reti neurali (con o senza `dataset` di apprendimento)

Usare HashMap

Il modo più naturale di immagazzinare valori `Q(s,a)` è in array indicizzati da stati e azioni, o da qualche codifica derivata da questi due oggetti. Il problema è che potrebbe necessitare molto spazio, e molte celle potrebbero risultare vuote e inutili. Le HashMap di Java rendono possibile di immagazzinare solo valori per coppie (stato,azione) attualmente incontrate durante un episodio: i valori sono immagazzinati nell'HashMap e la coppia (stato, azione) serve come chiave per otternerli.

Due classi implementano `IRewardStore` in termini delle HashMap Java:

- `RewardMemorizer` per valori `Q(s,a)` a valori reali
- `RewardMemorizerInteger` per un approssimazione di valore intero(byte) di valori `Q(s,a)`

Usare reti neurali

Quando lo spazio di uno stato e/o di un azione diventa troppo grande, anche le HashMap sono incapaci di immagazzinare tutte le esperienze che un agente incontra eseguendo episodi successivi. L'idea è allora di provare a imparare i valore Q(s,a) come una funzione della coppia (stato, azione). Ogni algoritmo di apprendimento macchina può essere usato, ma allo stesso tempo, solo reti neurali con *feed-forward e* retro-propagazione sono usate. Lo schema generale di apprendimento è come segue:
L'algoritmo memorizza un numero fissato di triplette (stato, azione,ricompensa). Durante la fase della memorizzazione, l'algoritmo sceglie un azione a caso.
Quando questo numero di triplette è raggiunto per la prima volta, una rete neurale viene costruita, e le successive decisioni saranno prese ritornando molto spesso (c'è ancora un po' di esplorazione) l'azione per la quale l'output della rete è massimo. Restituendo l'azione migliore a ogni passo, l'algoritmo continua a immagazzinare le nuove triplette che l'agente gli invia. Dopo che un nuovo blocco di triplette viene memorizzato, la rete aggiorna il suo peso lanciando una fase di apprendimento. Questo comportamento è implementato nella classe

RewardMemorizerNN: una variante, dove la rete neurale è costruita inizialmente e impara a ogni passo, senza memorizzare alcun dataset, è implementata nella classe `RewardMemorizerNNSinglePass`. Queste reti neurali possono essere usate negli algoritmi di Q-Learning senza cambiare quegli algoritmi, dicendo semplicemente a Java che l'`IRewardStore` sarà di classe `RewardMemorizerNN`. Anche le classi `PengNNSelector` e `WatkinsNNSelector` illustrano questa semplice definizione.

Codificare stati e azioni per reti neurali

Le reti neurali di PIQLE necessitano come input un vettori a valori reali, e danno come output un singolo numero reale. L'input è costruito dalla coppia (stato, azione), che significa che dobbiamo fornire il modo per trasformare una coppia (stato, azione) in un vettore a numeri reali. Questo è fatto in tre passi, situati in differenti posti nella gerarchia di classi di PIQLE.

- Codifica uno stato come un vettore di numeri reali: questo è definito nell'interfaccia `IState` dal `package` environment dal metodo `nnCoding()`. Per ogni problema, è compito del programmatore trovare un buon schema di codifica per gli stati, in termini di un vettore di numeri reali.
- Codifica di un azione come vettore di numeri reali: definito nel metodo `nnCoding()`, dall'interfaccia `IAction`, nel `package` environment. Anche qui, il programmatore deve trovare, per un dato problema da codificare, il modo migliore di codificare un azione in un vettore di numeri reali, in modo da catturare tutte le caratteristiche rilevanti che permetteranno alla rete di apprendere.
- Raggruppare questi due vettori in un singolo vettore, che verrà inserito nella rete: questo è fatto nei metodi `get()` e `put()` dalla classe `RewardMemorizerNN`, nel `package` "`qlearning`".

Un modo standard per codificare stati o azioni in un vettore di numeri reali, che sembra dare risultati ragionevolmente buoni risulta essere:

- Per ogni caratteristica di stato o azione che può essere espressa come numero reale: usare direttamente questo valore, magari dopo averlo normalizzato.
- Per ogni campo discreto, che può assumere k valori distinti (con k non troppo grande): codificalo con k celle di input, dalle quali solo una sarà uguale a 1, le altre a 0. (una fra k)

Scegliere un valore di default

La prima volta che un algoritmo guarda sia in una HashMap che in una rete neurale in cerca di un valore `Q(s,a)`, non troverà ovviamente nulla, il problema è che deve restituire per forza un valore. In PIQLE, il valore di ritorno può essere scelto dal programmatore, definendo l'algoritmo. L'interfaccia che definisce il valore di che deve essere ritornato nel caso non ci sia attualmente un valore disponibile è chiamata `IDefaultValueChooser` ed è parte del pacchetto "`qlearning`". Questa interfaccia contiene solo il metodo `getValue()`;

Le classi che implementano questa interfaccia sono:

- `NullValueChooser`: che restituisce sempre zero.
- `ConstantValueChooser` che restituisce un valore costante, passato come parametro.
- `IntervalValueChooser` che ritorna un valore uniformemente distribuito tra due limiti (parametri).

Ovviamente, ogni altra strategia di scelta di un valore di default può essere aggiunta a questa lista. Lo sceglitore di valore di default è il `NullValueChooser`. Si può cambiare questa impostazione di default usando l'algoritmo del costruttore chiedendo uno sceglitore di valore come parametro: questa informazione è rilasciata dall'algoritmo nel `RewardMemorizer` incluso nell'algoritmo.

La scelta dell'azione successiva

Grossomodo, il metodo `getChoice()` restituisce l'azione migliore, finchè le `Q(s,a)` sono interessate. Ma, nell'apprendimento per rinforzo, gli algoritmi devono

preservare un bilancio tra esplorazione (visitare nuovi stati) e sfruttamento (scegliere l'azione più ricompensante). Questo è fatto introducendo un pizzico di casualità nella scelta della successiva azione da eseguire. Molti molti per implementare e controllare la casualità sono stati proposti. PIQLE definisce tre implementazioni:

- *E-Greedy*: Un azione casuale è scelta con una certa probabilità invece dell'azione migliore.
- *Roulette Wheel*: Ogni azione può essere scelta con una probabilità proporzionale al suo valore Q(s,a) associato.
- *Boltzmann Selection*: Come nella procedura Roulette Wheel, ogni azione può essere scelta con una probabilità relativa al suo valore Q(s,a) associato, ma usando una differente funzione da quella lineare.

Il comportamento di default è l'*E-Greedy*.

Alcuni parametri sono necessari per controllare e calibrare il comportamento degli algoritmi di apprendimento per rinforzo, altri per definire precisamente le funzioni di casualità. Questi valori sono dati con i loro metodi setter e getter. Alcuni di loro necessitano anche di cambiare durante il tempo, e la classe indica come e quanto devono cambiare tra due successivi episodi.
I parametri sono:

- α (alfa) : Parametro dimensione del passo. Deve teoricamente diminuire per assicurare convergenza. Due metodi di decremento sono stati proposti, l'esponenziale e il geometrico. Solo il primo assicura la convergenza, ma entrambi danno buoni risultati pratici.
- γ (gamma): Parametro tasso di sconto - τ (tau): Temperatura per l'implementazione della selezione di Boltzmann della casualità.
- η (epsilon): La probabilità di scegliere un azione a caso nel caso *E-Greedy*. Si suppone che epsilon decresca durante il tempo: è infatti stato scelto di far controllare questo al programma principale, invece di implementare

schemi decrescenti come per α. Così, quando si implementa un esperimento usando l'apprendimento per rinforzo *E-Greedy*, non bisogna dimenticarsi di decrementare esplicitamente e impostare (con il metodo setEpsilon()) il valore di epsilon.

Il package di utilità Q-Learning

Questo pacchetto raggruppa alcune utili classi per implementare gli algoritmi di apprendimento per rinforzo:

Raggruppando uno stato e una azione nello stesso oggetto, lo usa come chiave in una HashMap.

- Definendo differenti tipi di oggetti per immagazzinare `Q(s, a)`.
- Definendo Iterazione di valori.
- Definendo un oggetto per gestire le tracce di ammissibilità.
- Definendo modi diversi per ritornare un `Q(s, a)` di default quando nessuno è disponibile .

Trovare i parametri per un buon apprendimento é complicato dal momento che l'algoritmo è molto sensibile a variazioni di parametri, è necessario dunque eseguire varie prove prima di individuare la configurazione adatta al proprio problema

2. Integrazione del simulatore ExTraS nel framework PIQLE

Obiettivo di questo lavoro di tesi é analizzare l'efficacia degli algoritmi di apprendimento per rinforzo nell'ambito della profilazione utente. L'obiettivo è, quindi, quello di determinare la qualità del supporto che questi algoritmi possono garantire e capire in quanto tempo essi ad apprendere le preferenze degli utenti. Il comportamento atteso da parte del simulatore deve rispecchiare il comportamento di un utente che prima di poter fruire dei contenuti della pagina Web decide di modificare l'aspetto degli elementi che la compongono. Inoltre, deve consentire di testare il comportamento di più tipi di utente, con l'obiettivo di individuare e porre rimedio a eventuali a casi in cui l'algoritmo non sia efficace ed efficiente. Per fare ciò, é necessario simulare alcuni utenti Web, con caratteristiche ed esigenze specifiche. In particolare, nell'ambito di questa tesi, un simulatore Java é stato integrato in PIQLE. Nel seguito di questo capitolo sarà descritto il simulatore ExTraS (EXperiential TRAnscoding System) e il progetto della sua integrazione in PIQLE.

2.1 Il Simulatore ExTraS

Scritto in Java, il simulatore ExTraS rappresenta una valida risposta alle richieste di questa tesi in quanto il suo funzionamento consiste nel simulare la fruizione di alcune tra le più visualizzate pagine Web da parte di un utente fittizio, il quale per ogni pagina Web aperta conta il numero di modifiche che apporterebbe agli elementi della stessa al fine di ottenere una pagina perfettamente consultabile relativamente alle sue necessità e preferenze. Il suo funzionamento consiste nel modellare il concetto di preferenze dell'utente sulla base dei principali parametri che compongono gli elementi della pagina, ovvero la dimensione e il tipo di font, l'interlinea e l'allineamento. Il simulatore permette la rapida rappresentazione di qualsiasi tipo di utente previa un'accurata analisi delle sue caratteristiche, in questo modo volendo ad esempio modellare un utente con ipovisione sono state analizzate diverse pubblicazioni e studi che riportano le preferenze di utenti di questo tipo rispetto ai contenuti testuali delle pagine Web (font di dimensione

almeno pari a 16, interlinea almeno uguale a 1,5, evitando allineamenti diversi da quello sinistro). Un altro tipo di utente al contrario potrebbe essere più interessato a un determinato tipo di font o contrasto tra colore di testo e colore di sfondo. E' quindi necessario prestare molta attenzione nel momento della modellazione dell'utente.

La simulazione da parte di ExTraS avviene attraverso la modellazione in classi di Utente, Pagina, Elemento della pagina. I seguenti sotto-paragrafi li descrivono nel dettaglio

2.1.1 La classe Element, tipo di elemento della pagina

Questa classe rappresenta un tipo di elemento testuale generico di una pagina Web, il quale può assumere innumerevoli forme in base a ciò che deve rappresentare, ad esempio un semplice campo di testo, una nota, un commento, un titolo, un link, un item di una lista e così via. Nonostante i tipi di elemento possano differenziarsi molto tra loro per il loro ruolo, tutti quanti risultano essere una particolare combinazione di valori dello stesso insieme di parametri, in base ai quali cambiano di aspetto e dunque di funzione. In particolare in questa tesi si prenderà in considerazione un sottoinsieme di questi parametri, quelli che maggiormente possono rappresentare una barriere digitale, condizionando la leggibilità degli elementi della pagina, da parte degli utenti:

- Font-family, ovvero il tipo di carattere che costituisce il testo.
- Font-size, ovvero la dimensione del carattere che costituisce il testo.
- Alignment, ovvero l'allineamento del testo.
- Line-height, ovvero l'interlinea, la distanza tra una riga di testo e l'altra.

Sono questi, infatti, gli aspetti sui quali principalmente bisogna focalizzarsi per adattare gli elementi in modo tale da rispondere alle esigenze della maggior parte delle tipologie di utente nell'ambito di questo tesi. La classe Element, dunque, rappresenta la collezione di questi valori e le funzioni che consentono il loro

ottenimento e la loro modifica. Dal momento che questa classe rappresenta un tipo di elemento e non un singolo elemento, essa contiene anche il numero di elementi di quel tipo presenti all'interno della pagina. In più, per rendere più realistica la simulazione ,è presente anche la distinzione tra il numero di elementi presenti nella pagina e il numero di elementi presenti nella pagina a cui l'utente è realmente interessato, questo perché aprendo una determinata pagina è possibile trovare centinaia di elementi di un certo tipo, ma è anche vero che l'utente non avrà necessità di leggerli effettivamente tutti poiché in ampia parte potrebbero essere pubblicità o contenuti non rilevanti. Il numero di elementi effettivamente ispezionati è stimato come una percentuale sulla totalità degli elementi presenti nella pagina. Questa percentuale varia tra sito e sito perché ci possono essere siti più densamente ricchi di informazioni utili per l'utente, e altri magari caratterizzati da una presenza più invasiva di pubblicità. Inoltre per simulare la casualità di accesso ai contenuti da parte dell'utente, a ogni pagina è correlata ad una legge di probabilità che descrive quanti elementi vengono fruiti mediamente per quel tipo di pagina. La distinzione tra elementi presenti e elementi effettivamente letti all'interno della pagina costituisce la base di partenza per rappresentare la selettività dell'intervento sulla pagina, al contrario delle modifiche generali e generiche proposte dai browser odierni. Infine il simulatore considera il contrasto di luminosità e cromatico tra il colore di background (ovvero il colore di sfondo dell'elemento), e il colore di foreground (ovvero il colore del testo). Questa caratteristica degli elementi non sarà però tenuta in considerazione in questo lavoro di tesi.

2.1.2 La classe Page, una generica pagina Web

La classe Page sé una collezione di elementi Element, ciascuno con determinati parametri attraverso i quali è possibile rappresentare le più comuni pagine Web. La collezione di elementi può variare molto da pagina a pagina o tra le pagine di uno stesso sito e questo aspetto è stato curato all'interno del simulatore. Così come per la classe che modella l'utente, anche per quanto riguarda le pagine Web è

necessario una analisi prima di procedere alla loro rappresentazione. Non si sta cercando di studiare il comportamento della simulazione su ogni pagina Web esistente, ma si applica l'indagine su quelle che sono tra le più frequentate pagine Web al mondo. L'indagine da parte del simulatore viene svolta sulla rappresentazione delle pagine Web delle più visitate sulla base di uno studio internazionale condotto nel 2013 (User Centered and Context Dependent Personalization Through Experiential Transcoding, 2013), il simulatore, infatti, considera il modello delle seguenti pagine:

- Home Page di Facebook
- Pagina di un utente di Facebook
- Home Page di Google
- Pagina con i risultati di una ricerca di Google
- Home Page di Yahoo
- Pagina con i risultati di una ricerca di Yahoo
- Home Page di Wikipedia
- Articolo di Wikipedia
- Home Page di YouTube
- Video di YouTube

Ad ogni tipo di pagina è associato un id e un tipo per poter far loro riferimento in modo più semplice.

2.1.2.1 La classe InfoPages, i tipi di pagina

Un ruolo fondamentale svolto durante la rappresentazione delle pagine Web è quello della classe InfoPages, ovvero la classe che contiene le informazioni necessarie al fine di creare una determinata pagina a partire dal suo id o dal nome del suo tipo. In ogni caso, il costruttore della classe page andrà a chiedere alla classe InfoPages quali elementi andare a caricare. In questo modo si separa il concetto di Pagina dal concetto di Tipo di Pagina e risulta comodo apportare modifiche a posteriori. E' compito di questa classe fornire la rappresentazione vera

e propria di una pagina Web, così al suo interno è possibile studiare come siano state modellate le pagine Web ai fini della simulazione. Nel seguito descriviamo brevemente come sono strutturate le pagine Web campione che abbiamo scelto.

Home Page di Facebook

La rappresentazione in ExTraS della home page di Facebook all'interno del simulatore è composta da 76 elementi, di cui 33 presentano come carattere TimesNewRoman di grandezza 18, allineato a sinistra con interlinea doppia. Gli altri 43 elementi presentano carattere TimesNewRoman di grandezza 18 e allineamento a sinistra con interlinea singola.

Pagina personale di Facebook

La bacheca di Facebook è stimata essere composta da 27 elementi, di cui 12 presentano come carattere Serif di grandezza 15, allineamento a sinistra con interlinea doppia. Altri 14 elementi hanno carattere Serif di grandezza 15 allineato a sinistra con interlinea singola. Infine un elemento con carattere Serif di grandezza 20, allineato a sinistra con interlinea doppia.

Home Page di Google

L'home page di Google è composta da 14 elementi, 1 dei quali ha il carattere di tipo Arial di grandezza 15 allineato a sinistra con interlinea doppia, mentre gli altri 13 elementi hanno carattere Arial di dimensione 14, allineato a sinistra con interlinea doppia.

Pagina risultati di ricerca di Google

La pagina di ricerca di Google come ci si aspetta è molto più corposa rispetto alla sua home page, infatti presenta in tutto 32 elementi, di cui 11 hanno carattere Arial di dimensione 16 allineato a sinistra con interlinea doppia. Altri 10 elementi con carattere Arial di dimensione 10 allineato a sinistra con interlinea singola,

infine altri 18 elementi con carattere Arial di dimensione 10 allineato a sinistra con interlinea singola.

Home Page di Yahoo

L'home page di Yahoo è costituita da un cospicuo numero di elementi, infatti nella rappresentazione del simulatore si trovano 47 elementi, di cui uno ha carattere Serif di dimensione 20, allineato a sinistra con interlinea doppia, altri sette elementi hanno carattere Serif di dimensione 18 allineato a sinistra e interlinea doppia. Tre elementi hanno carattere Serif di dimensione 16 allineato a sinistra e interlinea doppia. Tredici Elementi hanno carattere Serif di dimensione 15 allineato a sinistra con interlinea doppia, infine 23 elementi con carattere Serif di dimensione 15 allineato a sinistra con interlinea singola.

Pagina risultati di ricerca di Yahoo

La pagina di ricerca di Yahoo comprende 43 elementi in totale, di cui 18 con carattere Serif di dimensione 16 allineato a sinistra con interlinea doppia, nove elementi hanno carattere Serif di dimensione 15 allineato a sinistra con interlinea doppia, infine 16 elementi hanno carattere Serif di dimensione 15 allineato a sinistra con interlinea singola.

Home Page di Wikipedia

L'home page di Wikipedia comprende 38 elementi, di cui 4 con carattere Serif di dimensione 20 allineato a sinistra con interlinea doppia, altri 5 elementi con carattere Serif di dimensione 16 allineato a sinistra con interlinea doppia, infine 29 elementi con carattere Serif di dimensione 15 allineato a sinistra con interlinea doppia.

Articolo di Wikipedia

Un articolo di Wikipedia comprende 61 elementi, di cui 5 hanno carattere Serif di dimensione 20 allieneato a sinistra con interlinea doppia, altri 55 elementi

presentano i carattere Serif di dimensione 15 allineato a sinistra con interlinea doppia, infine un elemento con carattere Serif allineato a sinistra con interlinea doppia.

Home Page di YouTube

L'home page di YouTube comprende 34 elementi, di cui sette hanno carattere Arial di dimensione 15 allineato a sinistra con interlinea doppia, i restanti 27 invece hanno carattere Arial di dimensione 15 allineato a sinistra con con interlinea singola.

Video di YouTube

Un articolo di YouTube è rappresentato da 43 elementi, di cui 10 hanno carattere Arial di dimensione 15 allineato a sinistra con interlinea doppia. Altri 31 elementi hanno carattere Arial di dimensione 15 allineato a sinistra con interlinea singola, infine due elementi hanno carattere Arial di dimensione 20 allineato a sinistra con interlinea doppia.

2.1.3 La classe User, l'utente

Questa classe modella il comportamento di un utente Web generico e costituisce il cuore del simulatore. In particolare questa classe è focalizzata sulle azioni che compierebbe un utente nel momento in cui esso ha la necessità di modificare una pagina secondo le proprie esigenze, necessità che possono derivare da molte cause, tra cui eventuali disabilità visive o legate a difficoltà nella lettura dei contenuti testuali. La struttura del meccanismo di simulazione, come mostrato in Figura 2.1 prevede che l'utente apra una pagina, la consulti, e applichi una modifica per ogni elemento consultato, nel caso in cui le cui caratteristiche non soddisfino le sue necessità. L'utente in questo modo, applica una modifica selettiva alla pagina, e non generale come se fosse applicata dagli strumenti forniti dai browser. Il simulatore nasce come strumento per calcolare una stima del

numero di modifiche medio che un utente con determinate necessità dovrebbe applicare per ottenere una pagina facilmente consultabile, per questo motivo il simulatore originale si occupa semplicemente di contare il numero di queste modifiche, tenendo presente che presa una pagina Web, il numero di elementi consultati ed eventualmente modificati da un utente non è deterministico ma casuale.

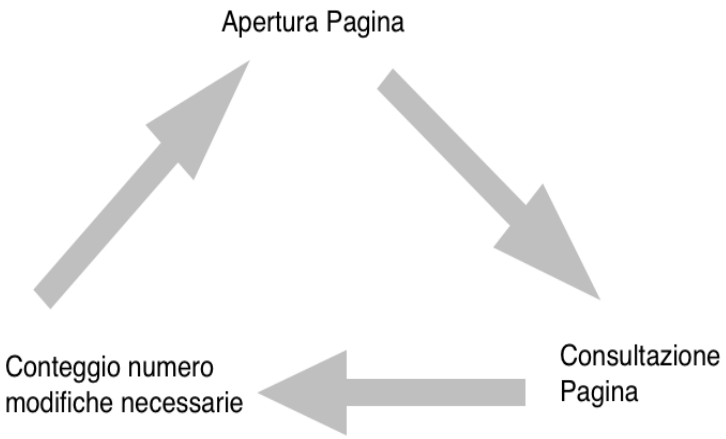


Figura 2.1, Fasi del comportamento dell'utente in ExTraS

Nonostante il simulatore non effettui alcuna modifica sulla pagina ma si limiti a contare il numero di modifiche che un utente effettuerebbe in una pagina considerando le proprie necessità, esso rappresenta un valido supporto per lo sviluppo e il test di un ambiente in PIQLE, in quanto fornisce tutti gli strumenti necessari per rappresentare le tipologie di utente e pe simulare una tipica navigazione Web. Inoltre permette di effettuare modifiche sostanziali al codice con agilità essendo esso ben studiato e strutturato.

2.1.3.1 Profilazione Utente

All'interno del simulatore ExTraS, la modellazione del comportamento di una determinata tipologia di utente avviene definendo le sue necessità riguardo le quattro principali caratteristiche degli elementi, ovvero dimensione del testo, famiglia del carattere, allineamento, interlinea. Una volta effettuato uno studio sul

tipo di soggetto da rappresentare, è possibile quantificare e definire le necessità e permettere la corretta esecuzione del simulatore. Attraverso i parametri forniti durante la rappresentazione, l'utente si comporterà in maniera consona e coerente e il suo compito sarà quello individuare per ogni pagina in numero di modifiche necessario per un'agevole fruizione dei contenuti della stessa. Alcune stime possono essere fatte a priopri, per esempio se si dovesse rappresentare il comportamento di un utente con problemi di ipovisione ci si aspetta che egli preferisca una determinata dimensione del testo, oppure che le righe siano ben separate tra loro, in questo esempio i parametri coinvolti sono font-size e lineheight. All'interno del simulatore sono state implementate le rappresentazioni di tre tipi di utenti ben precisi, ovvero la rappresentazione di un utente con dislessia, un utente con ipovisione e quella di un utente anziano. I test svolti ad implementazione ultimata sono stati condotti sfruttando queste tre tipologie di utenti le quali costituiscono tre esempi di utenti con specifiche esigenze legate alla fruizione dei contenuti testuali durante la navigazione Web. Oltre a queste prime tre categorie di utenza dal comportamento standard, è interessante studiare il comportamento del framework PIQLE di fronte a ulteriori tipi di situazioni, più generali e non indirizzate ad un particolare tipo di utente. Si vuole cioè studiare anche la reazione del framework di fronte al comportamento di un utente non sempre coerente, ovvero che non si comporta sempre nello stesso modo e a volte esegue azioni contradditorie rispetto a quelle fatte in precedenza. Inoltre è stata posta attenzione alla gestione da parte del framework di un eventuale cambio di comportamento, radicale o meno, da parte di un utente. Tale casistica andrebbe a rappresentare una possibile correzione da parte dell'utente rispetto alle proprie preferenze, eventualmente anche a fronte di un cambio delle proprie necessità. L'esempio classico viene fornito da un utente anziano, il quale potrebbe subire un calo della vista col passare del tempo, di conseguenza le sue necessità cambierebbero nel tempo. Se dapprima il framework si fosse adattato ad un profilo utente, sarebbe necessario che esso si accorgesse di un eventuale cambio di preferenze e che agisca di conseguenza.

L'utente con dislessia

La dislessia è un disturbo classificato tra i Disturbi Specifici di Apprendimento e la sua principale manifestazione consiste nella difficoltà che hanno i soggetti colpiti a leggere velocemente e correttamente ad alta voce ("Difficoltà di apprendimento e dislessia", 2004). La dislessia è una disabilità dell'apprendimento di origine neurobiologica. Essa è caratterizzata dalla difficoltà a effettuare una lettura accurata e/o fluente e da scarse abilità nella scrittura (ortografia) ("Difficoltà di apprendimento e dislessia", 2004). Queste difficoltà derivano tipicamente da un deficit nella componente fonologica del linguaggio, che è spesso inatteso in rapporto alle altre abilità cognitive e alla garanzia di un'adeguata istruzione scolastica ("Difficoltà di apprendimento e dislessia", 2004). Conseguenze secondarie possono includere i problemi di comprensione nella lettura e una ridotta pratica nella lettura che può impedire una crescita del vocabolario e della conoscenza generale ("Difficoltà di apprendimento e dislessia", 2004). All'interno del simulatore un utente dislessico è rappresentato da un utente che necessita caratteri di dimensione maggiore o uguale a 18, un carattere di tipo arial o serif, e un allineamento a sinistra. Non ha preferenze riguardo l'interlinea.

L'utente ipovedente

L'ipovisione è una condizione di acutezza visiva molto limitata che ha notevoli conseguenze sulla vita quotidiana ("Tesauro del Nuovo Soggettario", 2013). Può essere causata da vari fattori (siano essi congeniti o acquisiti). La vista si può ridurre fortemente in seguito a patologie che possono colpire diverse strutture oculari, che vanno dalla cornea alla retina, fino al nervo ottico ("Tesauro del Nuovo Soggettario", 2013). L'ipovisione può essere associata a malattie che provocano una riduzione del campo visivo. Ad esempio, nel caso del glaucoma avanzato, che danneggia il nervo ottico, è come se si guardasse attraverso un tubo; oppure si può essere colpiti da patologie della macula, la zona centrale della retina

(la più comune è la degenerazione maculare senile, che provoca la perdita della visione centrale). L'ipovisione grave può degenerare in cecità, che può essere parziale o totale ("Tesauro del Nuovo Soggettario", 2013). All'interno del simulatore un utente con ipovisione è rappresentato da un utente che necessita caratteri di dimensione maggiore o uguale a 16, caratteri di tipo roman o serif, allineamento a sinistra, interlinea pari a 1,5.

L'utente anziano

La popolazione italiana odierna è costituita al 20% da persone con un età maggiore o uguale ai 65, un dato che pone il paese al secondo posto per percentuale di anzianità, subito dietro alla Germania. Questo dato è destinato ad aumentare con il tempo e non si può fare a meno di tenerne conto. L'indagine sulle cause di questo aspetto non fanno parte dello scopo di questa tesi, al contrario il dato di fatto è decisamente rilevante, dal momento che solo una piccola percentuale di queste persone utilizza abitualmente dispositivi elettronici anche solo per i scopi più semplici. E' del tutto comprensibile che in molti non non riescano o non vogliano nemmeno stare costantemente al passo con l'evoluzione tecnologica, dato che si tratta di una realtà molto dinamica che richiede continui aggiornamenti. E' sicuramente vero, però, che se queste tecnologie fossero più semplici, intuitive e quindi alla portata di tutti gli utenti, questi potrebbero essere più incentivati sfruttarle. L'avanzare dell'età può comportare alcune difficoltà (legate alla vista, alla destrezza dei movimenti, ecc.) che potrebbero influenzare la fruizione di contenuti Web. Per questo motivo nell'ambito di questo lavoro è stata modellata anche una rappresentazione per questo tipo di utenti. All'interno del simulatore, un utente anziano è rappresentato da un utente che necessita di caratteri di dimensione maggiore o uguale a 14, caratteri di tipo SanSerif o Arial, allineamento a sinistra e interlinea maggiore o uguale a 1,5 righe.

L'utente incoerente

L'utente incoerente all'interno del simulatore è stato implementato a partire dalla rappresentazione di un utente dislessico, la scelta è stata puramente casuale e non inficia i risultati dei test. Preso un utente qualunque, l'utente incoerente è l'utente che effettua con una certa probabilità un'azione non consistente rispetto al suo comportamento usuale. La sua implementazione consiste nel far restituire al framework PIQLE una ricompensa completamente opposta come valutazione di una azione proposta, con una certa probabilità. La valutazione della proposta di una azione da parte dell'utente deve essere coerente con i suoi criteri di adattamento delle pagine. In questo modo, invece, il framework si accorgerà che l'utente effettua determinate modifiche, ma allo stesso tempo restituisce, con una determinata probabilità, un giudizio negativo quando è il framework stesso a proporre la determinata azione. I parametri di adattamento per l'utente incoerente consistono, da un lato, in quelli dell'utente con dislessia, mentre nel momento in cui restituisce un responso incoerente si comporta esattamente al contrario dell'utente dislessico classico. Ad esempio, se l'utente dislessico necessita di una dimensione dei caratteri maggiore o uguale a 18, quando si comporta incoerentemente assegna un giudizio negativo se il framework PIQLE gli propone una correzione della dimensione del carattere di valore maggiore o uguale a 18, mentre normalmente restituirebbe un giudizio positivo.

L'utente dinamico

Le preferenze di un utente possono cambiare nel tempo, anzi è del tutto probabile che ciò accada. Dopo un certo periodo l'utente potrebbe decidere di non desiderare più una determinata impostazione o aggiungerne delle nuove, quindi sistematicamente smetterebbe di modificare determinati parametri degli elementi, oppure al contrario potrebbe aumentare il numero di modifiche per ogni elemento. Nel caso di un utente anziano è probabile che con il passare del tempo possa aver bisogno di aumentare ulteriormente la dimensione dei caratteri del testo a fronte di

un calo della vista. In generale, questa dinamica caratterizza la maggior parte degli utenti, non solo quelli con particolari condizioni o disabilità. Dal momento che comprende la maggioranza degli utenti, è giusto studiare il comportamento degli algoritmi di apprendimento per rinforzo di fronte a questa eventualità. La differenza tra questo tipo di utente e quello incoerente consiste nel fatto che seppur l'utente cambi preferenze ,esso mantiene la propria coerenza, semplicemente rispetto ad altri parametri, mentre il secondo è sempre incoerente con una certa probabilità. Quello dinamico è un utente generico che dopo una certa quantità di tempo cambia politica di adattamento e, quindi, è necessario indagare quali effetti questo comportamento produce sull'algoritmo di apprendimento per rinforzo.

2.3 ExTraS e PIQLE

2.3.1 Struttura del sistema

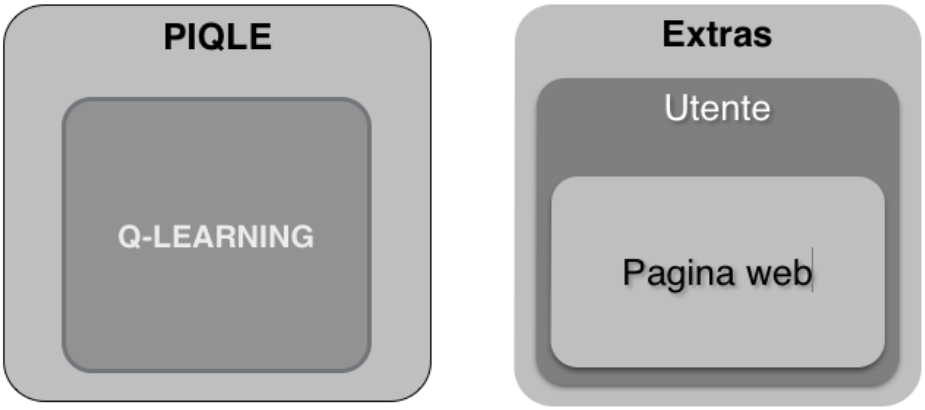


Figura 2.2, I due sistemi separati

L'obiettivo di questa tesi, dunque, consiste nell'integrazione di due macro-sistemi, ExTraS e PIQLE, in modo da studiare la qualità del supporto fornito da un algoritmo di apprendimento per rinforzo come soluzione per l'implementazione di un meccanismo automatico di profilazione utente. Per fare in modo che il framework sia in grado di fornire supporto nella maggioranza dei casi, esso deve essere in grado di adattarsi a quante più realtà possibili. Allo stesso modo, l'algoritmo non deve essere interessato a quale particolare tipo di utente si trova davanti, ma semplicemente studiare il suo comportamento e in base a quello, apprenderne le preferenze. In questo modo si mantengono indipendenti i due sistemi e si ottiene un modello in grado di adattarsi nella maggioranza delle situazioni. Dal momento che il compito del framework PIQLE è quello di apprendere il comportamento di un utente, l'unica porzione di realtà alla quale deve essere interessato è la pagina Web aperta dall'utente e le modiche che esso apporta ad essa, ovvero il suo modo di agire. Per fare questo è necessario che il framework sia attento alle modifiche effettuate dall'utente, in più deve poter essere in grado di apportare modifiche alla pagina stessa una volta raggiunta la conoscenza necessaria per proporre automaticamente azioni efficaci di supporto per l'utente. E', dunque, necessario che entrambi i due sistemi abbiano un riferimento l'uno dell'altro per fare in modo che l'ambiente sia correttamente informato delle modifiche effettuate dall'utente e contemporaneamente l'ambiente

possa proporre ed effettuare le modifiche necessarie per l'adattamento della pagina.

La classe Universo

Per fare in modo che i due sistemi interagiscano fra loro è necessario che entrambi risiedano nello stesso spazio o, meglio, nello stesso contesto; questo al fine di fornire a entrambe le entità in gioco un riferimento reciproco e garantire la condivisione delle medesime risorse. L'ambiente di PIQLE deve essere in grado di comunicare con l'utente per poterlo osservare ed apprendere le sue preferenze attraverso il suo comportamento ma, soprattutto, deve avere accesso alla pagina Web che l'utente sta consultando per poter applicare le dovute modifiche. L'utente, invece, deve poter interagire con l'ambiente di PIQLE per valutare le azioni da esso proposte e accettarle o meno, fornendo all'algoritmo preziosi dati da analizzare. Nasce per questo motivo il concetto di una terza entità il cui scopo è proprio quello di fare da tramite tra i due macro-sistemi e rappresentarne il contesto, per questo motivo prende il nome di classe Universo, ovvero l'ambiente in cui entrambi i sistemi in gioco possono muoversi e interagire fra loro. In questo modo risulta sufficiente fornire ai due sistemi il riferimento allo stesso Universo, e, nel momento in cui necessitino di comunicare con l'altra entità, non dovranno fare altro che passare le informazioni a tale Universo, il quale si occuperà di fornire correttamente le informazioni. La classe Universo ricopre anche il ruolo della gestione dell'ordine delle interazioni e fornisce alcuni utili strumenti durante la fase di test, permettendo di eseguire confronti tra gli stati delle due entità principali e di trarne le dovute conclusioni.

La Figura 2.3 riassume la struttura del progetto:

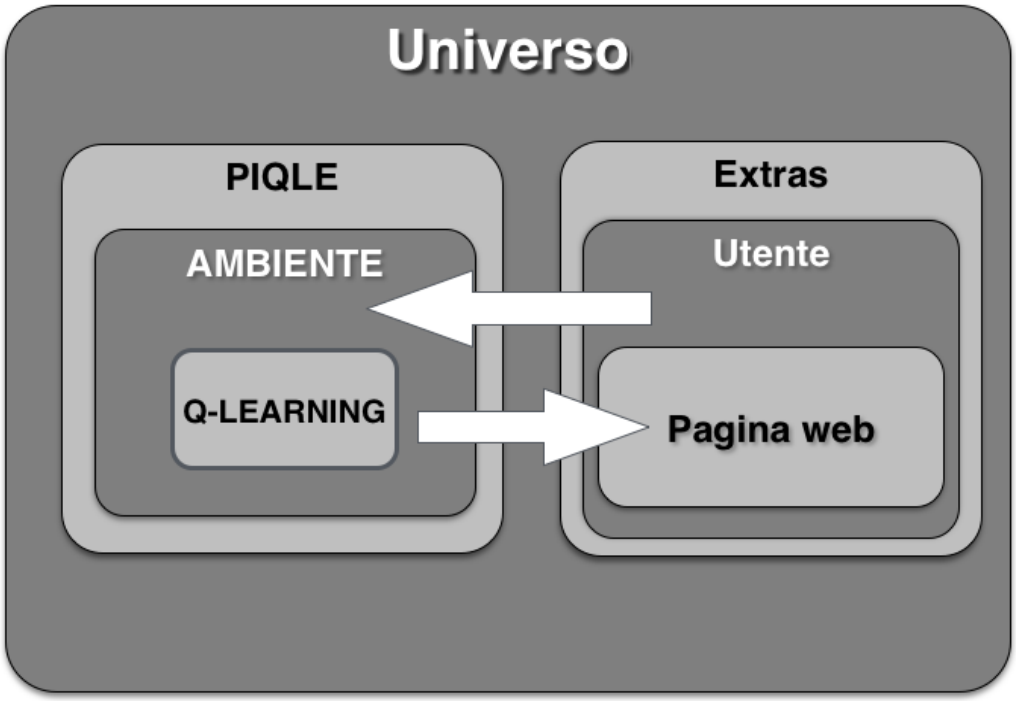


Figura 2.3, Struttura di progetto

2.3.2 Interazione

Il meccanismo di funzionamento dei due sistemi prevede che la loro interazione sia suddivisa in due momenti principali. Un primo momento è quello in cui l'utente scorre la pagina, individua gli elementi dei quali è interessato a fruire del contenuto e applica le modifiche necessarie per una lettura agevole. In questo momento l'algoritmo di apprendimento si limita ad osservare e ad apprendere in maniera passiva immagazzinando informazioni utili, man mano che l'utente effettua le modifiche. Il secondo momento è quello in cui l'algoritmo forte dei dati ottenuti fino a quel momento, effettua tutti i calcoli necessari e propone all'utente varie modifiche alla pagina e attende il parere dell'utente. Se la modifica proposta dall'algoritmo risulta ininfluente, o addirittura deleteria, l'utente scarta la proposta, al contrario, se la proposta risulta proficua per le necessità dell'utente, verrà accettata ed applicata effettivamente sulla pagina. L'algoritmo apprende quali azioni siano state prese in considerazione o meno dall'utente e sfrutterà i nuovi dati per generare le proposte successive.

Fase iniziale

I due momenti di interazione precedentemente descritti si susseguono con un ordine che viene mantenuto per tutta la sessione della simulazione. Dal momento che l'obiettivo finale dell'algoritmo è quello di presentare all'utente pagine già adattate sulla base delle sue necessità, è chiaro che il momento in cui l'algoritmo interviene sulla pagina debba precedere il momento in cui l'utente interviene sulla pagina. Il momento in cui l'utente apre una pagina in cui l'algoritmo è già intervenuto e scorrendo la pagina non trova la necessità di apportare alcuna modifica coincide con il momento in cui l'algoritmo ha appreso appieno le preferenze dell'utente. Nasce però un dilemma che riguarda la fase iniziale, ovvero la fase in cui l'utente apre la pagina per la prima volta. In questo momento l'algoritmo è completamente all'oscuro delle preferenze dell'utente dal momento che non ha ancora avuto modo di apprendere nulla. In questa fase, dunque, l'algoritmo deve limitarsi ad apprendere passivamente mentre già dalla pagina successiva può cominciare ad elaborare i dati ricevuti. Lo schema di interazione, dunque, prevede che l'utente apra una prima pagina, l'algoritmo si limiti ad osservare e l'utente applichi le sue modifiche. Dal turno successivo, nel momento in cui l'utente apre una nuova pagina, sarà per primo l'algoritmo ad intervenire e generare proposte, dopodiché spetterà all'utente scorrere la pagina ed eventualmente applicare ulteriori modifiche nel caso in cui l'algoritmo non sia riuscito a generare tutte modifiche necessarie.

Dalle considerazioni fatte in precedente è possibile delineare uno schema di esecuzione ben definito tra i due macro-sistemi, come mostrato in Figura 2.4

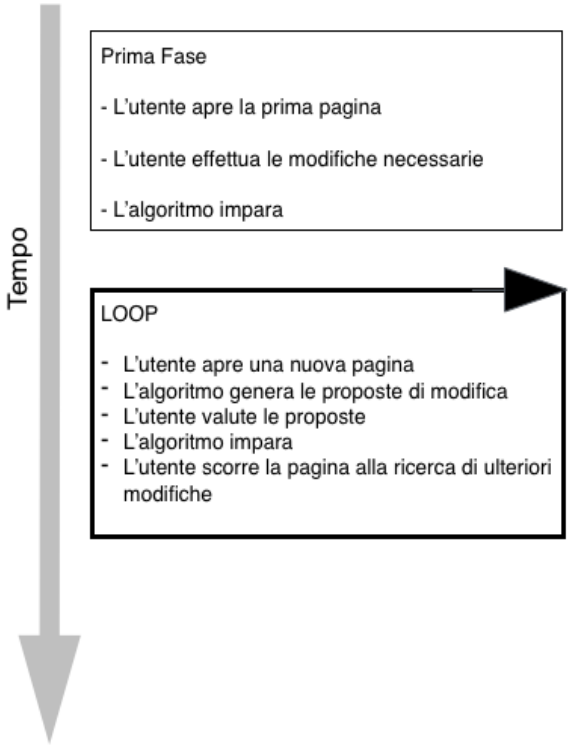


Figura 2.4, Schema di esecuzione

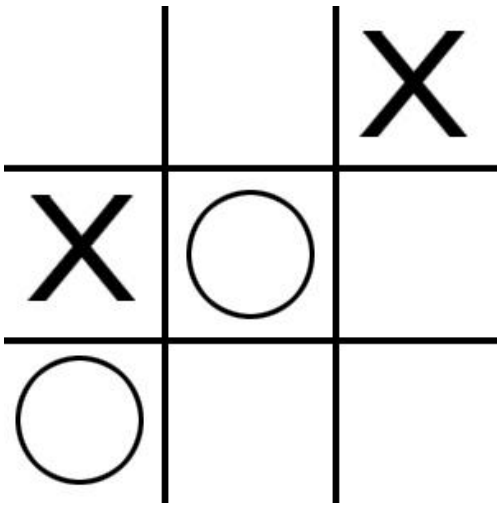
2.3.3 Modellazione di PIQLE

Affinché i due sistemi possano interagire e per fare in modo che l'algoritmo possa fornire effettivamente un supporto per l'utente, è necessario che il framework PIQLE sia correttamente impostato al fine di modellare il problema che si vuole affrontare. La rappresentazione di un problema in PIQLE consiste nella definizione di tre entità separate, che sono l'ambiente, l'azione, lo stato. E' sufficiente definire opportunamente queste tre componenti perché l'algoritmo abbia tutto il necessario per essere eseguito e fornire soluzioni. PIQLE, infatti, è strutturato in modo da consentire una rapida modellazione dei problemi, previa una loro analisi accurata.

Esempio del TicTacToe

L'ambiente rappresenta il contesto del problema e deve contenere tutti gli elementi che lo costituiscono. Nella documentazione di PIQLE sono presenti vari esempi di

problemi implementati, tra cui quello del gioco del TicTacToe, in Italia conosciuto come Tris. Si tratta di un gioco a turni molto semplice che coinvolge due giocatori. Ogni giocatore possiede un proprio simbolo e l'obiettivo del gioco consiste nell'inserire prima dell'avversario tre simboli della propria squadra in modo che occupino un intera riga, un intera colonna o una diagonale del quadrante di gioco, costituito da una tabella tre per tre.



2.5, Esempio tabella del gioco del tris

Questo esempio è di grande utilità, dal momento che il gioco è molto semplice e studiare la sua implementazione in PIQLE permette di capire velocemente la filosofia del framework. Dallo studio fatto sulla documentazione di PIQLE sappiamo che i compiti essenziali dell'ambiente sono quello di stabilire quali sono le azioni possibili in presenza di un determinato stato dell'ambiente, generare un nuovo stato a partire da un determinato stato e una azione, infine, generare una ricompensa su un insieme costituito dallo stato attuale e dallo stato precedente dell'ambiente associati ad una azione. Nel caso del Tris, ad ogni turno ogni casella rimasta vuota rappresenta una possibile azione per il giocatore, mentre una singola azione rappresenta posizionare il proprio simbolo in una casella rimasta vuota. Uno stato, invece, rappresenta una determinata configurazione raggiunta nel corso della partita e generare una ricompensa sull'azione eseguita consiste nel

determinare se la mossa appena eseguita ha portato benefici per il giocatore o meno e se lo ha avvicinato alla vittoria.

Ambiente

Nell'ambito di questo lavoro di tesi il numero di azioni possibili per ogni pagina è di gran lunga superiore a quello delle azioni possibili nell'esempio del TicTacToe. Dovendo generare tutte le possibili azioni dato uno stato, l'ambiente andrà a considerare ogni elemento della pagina e per ognuno elencherà tutte le azioni effettuabili sull'elemento. Questo vuol dire che il sistema genererà un azione per ogni valore assumibile da ogni parametro dell'elemento. Preso un elemento con valore di allineamento “Left”, ogni possibile azione per quanto riguarda l'allineamento sarà la sostituzione del valore attuale con “Right” , “Center” o “Justified”. Lo stesso discorso vale per quanto riguarda la grandezza del carattere, la famiglia del carattere e l'interlinea. Chiaramente il sistema non deve generare proposte inutili, come la sostituzione di un valore con il medesimo valore. Dovendo invece processare un nuovo stato a partire da uno stato e un azione, non dovrà fare altro che applicare la modifica rappresentata dall'azione su ogni altro elemento visualizzato con le stesse caratteristiche di quello che ha generato l'azione. Se l'azione consiste nell'impostare la grandezza del carattere a 18 per un elemento con dimensione di carattere minore di 18, la modifica si ripercuote su tutti gli elementi visualizzati che non soddisfano tale caratteristica. La differenza tra una modifica generale e una mirata sulla pagina consiste proprio nel fatto che le azioni, seppur agiscano in cascata sugli altri elementi, esse, in generale trattano solo gli elementi effettivamente consultati dall'utente, e non la loro totalità. Per quanto riguarda il generare la ricompensa, invece, l'ambiente deve comportarsi in maniera differente rispetto all'esempio del TicTacToe e, in generale, rispetto a tutti gli esempi forniti con il framework. Il compito di stabilire il premio o punizione su una data azione e un dato stato in questo caso non spetta all'ambiente, ma all'utente. Volendo implementare un ambiente capace di adattarsi a qualsiasi tipo di utente, è necessario che esso non integri alcuna politica di ricompensa ma che la riceva dall'esterno. Per ogni azione generata relativamente

ad uno stato l'utente esegue una valutazione, il valore della quale determinerà l'effettiva applicazione della modifica o il suo rifiuto. E' il risultato di questa valutazione di cui l'ambiente terrà da conto e, in questo modo, avrà un primo strumento per apprendere il comportamento e le preferenze dell'utente.

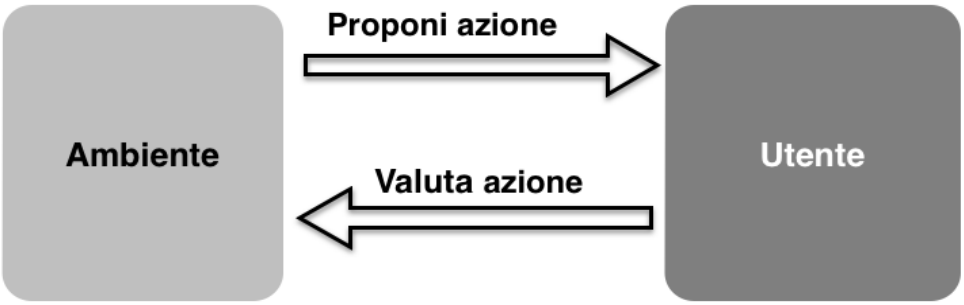


Figura 2.6, Comunicazione Ambiente/Utente

Stato

Anche il concetto di stato nel nostro lavoro di tesi cambia leggermente rispetto all'esempio visto. Come già accennato, lo stato rappresenta una particolare configurazione dell'ambiente in un dato momento e nell'esempio del TicTacToe è facilmente riconducibile alla configurazione della tabella in un determinato turno. Nel nostro caso ogni pagina rappresenta un nuovo episodio per l'ambiente e ogni episodio può raggiungere vari stati. L'ambiente, nel nostro caso, non cambia stato solo nel momento in cui le modifiche vengono man mano applicate ad una determinata pagina, ma cambia stato anche nel momento in cui l'utente cambia pagina. All'ambiente non deve interessare nemmeno il tipo di pagina che sta visitando l'utente sul momento, la sua unica preoccupazione deve essere l'apprendimento dell'utente nei confronti degli elementi, non importa da quali pagine provengano.

Azione

Per quanto riguarda le azioni non ci sono sostanziali differenze rispetto ai casi di studio offerti dal framework PIQLE. L'azione deve rappresentare una modifica, una modifica viene generata nel momento in cui un parametro assume un determinato valore o insieme di valori che non risultano adeguati e si vuole che invece assuma un preciso valore. Ci sono due tipi di azioni possibili, che dipendono dal tipo di parametro che si vuole modificare. Per quanto riguarda i parametri di presentazione a valori non numerici come il tipo di famiglia o l'allineamento, le uniche azioni possibili consistono in "Se il parametro non ha valore X, il parametro assuma il valore X". Prendendo in considerazione invece i parametri numerici come la dimensione del carattere e la dimensione dell'interlinea, oltre alle azioni sopra descritte, è possibile effettuare azioni come "se il parametro assume valori minori/uguali/maggiori del valore X, il parametro assuma il valore X. Per questo motivo è necessario che in ogni azione siano mantenute le informazioni riguardo al parametro che si intende modificare, il valore che il parametro assume (e per il quale é stata generata l'azione) e il valore che deve assumere il parametro in seguito all'applicazione dell'azione. L'azione è così rappresentata dalla terna (parametro, vecchioValore, nuovoValore).

Sessioni singole

Come già accennato, l'algoritmo di apprendimento per rinforzo deve essere impostato in modo tale da disinteressarsi alla tipologia dell'utente di cui deve apprendere il comportamento e ottenere un sistema capace di adattarsi alle preferenze di qualsiasi utente. Il sistema, inoltre, deve essere concepito per apprendere il comportamento di un singolo utente alla volta, non è infatti contemplato il caso multiutente sulla stessa istanza dell'ambiente, dal momento che ogni sessione di navigazione è comunque svolta da un singolo utente, per cui l'algoritmo si deve concentrare su un utente alla volta. E' comunque possibile in futuro implementare la rappresentazione del caso multiutente e studiare il comportamento dell'algoritmo assunto di conseguenza.

2.3.2 Modellazione dell'utente

Il simulatore ExTraS costituisce una base di partenza per eseguire vari test sull'efficacia del supporto fornito dagli algoritmi di apprendimento per profilare un utente sulla base del suo comportamento. Nonostante ciò è necessario apportare alcune modifiche alla struttura interna della classe utente per fare in modo che i due sistemi collaborino.

Meccanismo di comunicazione

Durante l'analisi del problema è emerso un chiaro schema di interazione tra il simulatore e l'ambiente in PIQLE, ovvero a parte il primo episodio in cui l'utente apre la prima pagina e l'algoritmo osserva passivamente, per tutto il resto dell'esecuzione l'interazione è costituita dall'alternarsi delle azioni da parte delle due entità in gioco. Per primo entra in azione l'algoritmo che avanza varie proposte di modifica, successivamente entra in azione l'utente per occuparsi della sistemazione di eventuali altre fonti di disturbo all'interno della pagina. Si delinea così una sorta di schema a turni, dove due entità collaborano per il raggiungimento dello stesso scopo, come mostrato nello schema di Figura 2.7

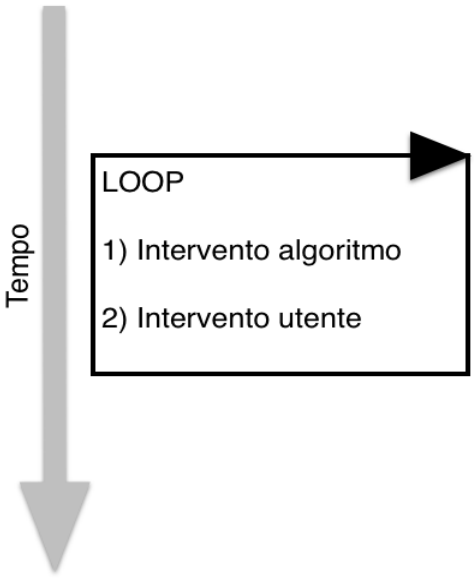


Figura 2.7, Alternanza degli interventi

Nel momento in cui l'utente entra in azione, l'algoritmo deve porsi in stato di osservazione e apprendere passivamente il comportamento dell'utente. Per fare ciò è necessario che l'utente comunichi direttamente all'algoritmo quali azioni compie, in particolare quali modifiche desidera apportare le pagine. In questo modo l'algoritmo riceve molte informazioni rilevanti dal momento che le azioni eseguite dall'utente sono sicuramente corrette e costituiscono un esempio da seguire. In definitiva ogni volta che l'utente rileva la necessità di effettuare una modifica, deve comunicarlo direttamente all'algoritmo ed esso deve memorizzare tutte le informazioni ricevute, in modo tale da migliorare il proprio criterio di scelta delle azioni al suo prossimo turno.

Meccanismo di valutazione

E' necessario dotare l'utente di un meccanismo di valutazione in base al quale soppesare le varie proposte di modifica fornite dall'ambiente durante la sua esecuzione, in quanto, come già detto, l'ambiente deve ottenere una ricompensa o punizione per ogni azione proposta per riuscire a ottenere un profilo preciso dell'utente. Questo meccanismo deve rispecchiare la politica che l'utente segue nei

confronti degli elementi delle pagine da esso visitate perché questa valutazione rappresenta una delle due principali fonti di apprendimento per l'algoritmo. Se i criteri di valutazione della ricompensa dell'utente fossero incoerenti con i criteri di valutazione che l'utente segue quando esamina la pagina alla ricerca di eventuali elementi da modificare, allora si genererebbe incoerenza e, in questo modo, l'algoritmo potrebbe non riuscire a convergere verso un profilo ben delineato dell'utente.

Elasticità della valutazione

Una azione proposta dall'algoritmo di apprendimento può risultare negativa o positiva a seconda delle preferenze dell'utente. E' anche vero, però, che un'azione potrebbe risultare più deleteria di un altra, avere cioè un fattore di negatività maggiore rispetto ad un altra azione, e allo stesso modo potrebbero esserci azioni che portano benefici e azioni che portano più benefici rispetto ad altre. Un esempio potrebbe essere legato alla dimensione del carattere. Consideriamo un utente che desidera avere gli elementi della pagina con dimensione del carattere maggiore o uguale a 20. L'algoritmo potrebbe generare un azione che preveda l'assunzione del valore 19 per tutti gli elementi che hanno dimensione del carattere uguale a 15. In questo caso il giudizio dell'utente sarà negativo, dal momento che la modifica non ha soddisfatto appieno le sue esigenze, d'altro canto per correttezza non può ricevere una valutazione uguale o peggiore di una azione che propone una dimensione inferiore di 19 o addirittura inferiore allo stesso 15. Nasce così l'idea che l'utente abbia una scala di valori ben precisa per valutare le azioni proposte dall'algoritmo, dove prendendo il valore '0' come elemento neutro, maggiore sarà il beneficio apportato dalla modifica, maggiore sarà il valore della ricompensa, al contrario sarà un numero negativo di valore sempre più piccolo al diminuire dell'efficacia dell'azione proposta.

2.3.5 Significatività dei test

L'obiettivo di questa tesi è verificare la qualità del supporto fornito da un algoritmo di apprendimento per rinforzo di fronte al problema della profilazione basata sul comportamento dell'utente. Per poter trarre le dovute conclusioni è necessario effettuare determinati test ed esaminarne i risultati. Una volta realizzata l'integrazione vera e propria tra il simulatore e il framework PIQLE sarà dunque possibile mettere alla prova le capacità dell'algoritmo. Ci si aspetta che l'algoritmo riesca a determinare le preferenze dell'utente dopo un certo numero di passi e quindi che converga ad una soluzione. Se questo avvenisse sarebbe interessante successivamente studiare in quanto tempo o dopo quanti episodi esso converga, e allo stesso tempo studiare le sue reazioni di fronte a casi anomali come quello proposto dall'utente incoerente o dall'utente dinamico. Per eseguire questi test si considereranno due possibili scenari:

- l'algoritmo termina il proprio turno dopo aver effettuato una serie di proposte di modifica, l'utente interviene sulla pagina e la pagina è completamente ordinata
- l'algoritmo termina il proprio turno dopo aver effettuato una serie di proposte di modifica, l'utente interviene sulla pagina e la pagina è parzialmente ordinata

Essi costituiscono un primo caso di studio, e permetteranno di osservare con quale frequenza e dopo quanto tempo l'algoritmo è in grado di produrre pagine adattate alle esigenze dell'utente, il quale non avrebbe più la necessità di effettuare modifiche.

In secondo luogo sarà interessante studiare la curva di apprendimento dell'algoritmo, in termini di numero di errori commessi prima di apprendere appieno le preferenze dell'utente e proporre sempre e solo proposte valide. Ci si aspetta infatti che in un primo momento l'algoritmo non fornisca sempre proposte corrette, ma che ne effettui varie che l'utente scarterà. Se dopo un certo lasso di tempo l'algoritmo non produce più errori ma genera sempre proposte giuste, significa che l'algoritmo ha imparato appieno le preferenze dell'utente. Questo perché per come è stato progettato l'ambiente, il ventaglio di scelte a disposizione dell'algoritmo è molto vasto. La probabilità di non effettuare mai errori eseguendo

azioni casuali tende a zero, per cui se sarà il numero di errori a tendere a zero sarà indice del fatto che l'algoritmo ha realmente appreso le preferenze dell'utente. I risultati dei test saranno mostrati nel capitolo seguente di questa tesi.

3. Implementazione e test

In questo capitolo è presente la descrizione dei principali passaggi che hanno costituito l'implementazione dell'integrazione dei due sistemi, attraverso l'esposizione di alcune porzioni di codice. Infine sono mostrati i risultati di alcuni test effettuati ad implementazione completata.

3.1 Implementazione

L'implementazione dell'integrazione del simulatore con il framework PIQLE é divisa in due parti principali, la prima consiste nella rappresentazione del contesto all'interno del framework, in particolare l'implementazione dell'ambiente, dello stato e dell'azione in PIQLE. La seconda parte consiste nella modifica del simulatore in modo da consentire l'interazione tra i due sistemi. Essendo entrambi i sistemi implementati in linguaggio Java, di conseguenza anche la loro integrazione sarà scritta in tale linguaggio. Non é assolutamente necessario eseguire un porting di entrambi i sistemi su un'altra piattaforma, dal momento che il linguaggio Java è Object Oriented e tale paradigma è particolarmente adatto per scopi di questo tipo. Ogni classe definita durante l'implementazione avrà il suffisso "EP", come simbolo di unione tra ExTraS e PIQLE. Di seguito un analisi nel dettaglio delle parti implementative più importanti.

3.1.1 La classe EPPresentation

Dal momento che la comunicazione tra i due sistemi coinvolti si basa sullo scambio di informazioni riguardanti i parametri degli elementi delle pagine, per prima cosa è stata definita una codifica comune sulla quale basare tale comunicazione. Per questo motivo nasce la classe EPPresentation. Questa classe ha lo scopo di fungere da interfaccia tra i due sistemi, definendo le collezioni di valori disponibili per ciascun parametro oggetto della comunicazione, ovvero la dimensione del carattere, la famiglia del carattere, l'allineamento, l'interlinea. La parte principale della classe è, dunque, quella costituita dalla definizione di questi valori, ed è mostrata in Codice 3.1:

```
private static final String[] users = new String[]{"dyslexic","visuallyimpared","elderlyuser","other"};
private static final String[] parameters = new String[]{"fontsize","fontfamily","alignment","lineheight"};
private static final int[] fontSizes = new int[]{10,11,12,13,14,15,16,17,18,19,20};
private static final String[] fontFamilies = new String[]{"arial","serif","timesnewroman","sanserif"};
private static final char[] alignments = new char[]{'l','c','r','j'};
private static final double[] lineHeights = new double[]{1.0, 1.5, 2.0, 2.5};
private static final String[] rulesNames = new String[]{"minore","uguale","maggiore"};
```

Codice 3.1, Campi della classe EPPresentation

I valori disponibili per ogni tipo di parametro sono memorizzati all'interno di vettori, questo per garantire ordine e per renderli facilmente reperibili. Infatti, attraverso metodi Getter relativi a ciascun campo, e attraverso funzioni che permettono di ottenere l'indice dei valori all'interno del proprio vettore a partire dal valore stesso, questa classe costituisce il modello di rappresentazione comune dei parametri tra i due sistemi. Per garantire assoluta coerenza tra i valori durante l'esecuzione del programma finale, ciascun campo è stato etichettato come "`final`", per impedire che in alcun modo questi valori possano essere modificati a programma avviato. In aggiunta, confinare la rappresentazione di tutti i parametri e di tutti i relativi valori in un unico punto, permette interventi più rapidi e a effetto globale nel caso di eventuali modifiche. I valori disponibili per ciascuno parametro sono stati scelti analizzando i parametri costituenti gli elementi delle pagine del simulatore ExTraS. Per quanto riguarda i possibili valori assumibili dal parametro "`font-size`", ovvero la dimensione dei caratteri, si è deciso di fornire un'ampia varietà di opzioni, per fornire compatibilità con quanti più casi possibili. La dimensione del carattere è, infatti, il parametro i cui valori possono oscillare più frequentemente di grandezza. Se in una pagina è possibile trovare qualche variazione di carattere, è ancora più probabile trovare differenti dimensioni di caratteri.

3.1.2 Implementazione del problema in PIQLE

La rappresentazione di qualsiasi problema o esperimento legato a PIQLE consiste nell'implementazione del concetto di ambiente, stato e azione seguendo le linee guida e le interfacce fornite nella documentazione. Per questo motivo sono state definite tre classi molto importanti a partire dalle specifiche del framework, ovvero le classi EPEnvironment, EPState, EPAction. Durante l'implementazione delle classi è stata anche fatta la scelta di usufruire di una HashMap come struttura dati in cui memorizzare le coppie Q(stato,azione), in quanto risulta molto più semplice da gestire rispetto alle reti neurali e comunque adatta a contenere le numerose occorrenze di coppie Q(stato,azione) generate dal problema.

3.1.2.1 La classe EPAction

Come già accennato nel capitolo precedente, questa classe ha il compito di rappresentare una specifica azione all'interno di PIQLE. Deve, quindi, essere in grado di trasmettere il concetto di modifica di un parametro e di sostituzione di un valore al posto di un altro. Dovrà allora contenere le informazioni relative al parametro che si intende modificare, il valore del parametro che ha generato la necessità di modifica, il parametro che andrà a sostituire quello presente. In più dev'essere possibile rappresentare un'azione del tipo "Dato un certo parametro, sostituisci in esso il valore X se esso assume valore maggiore/minore/uguale a Y", ovvero un azione che interessi una fascia di valori, e non solo uno in particolare. Questo tipo di azione coinvolge i parametri a valori numerici come la grandezza del carattere o l'interlinea, essendo essi rispettivamente di tipo `int` e `double`. Per rappresentare questo concetto è stato inserito un campo aggiuntivo chiamato "`rule`", ovvero "regola". Esso è un campo di tipo intero che può assumere i soli valori "1", "2" o "3", il cui compito sarà quello di rappresentare rispettivamente il concetto di "minore", "uguale", "maggiore". Il concetto di "regola" viene preso in considerazione solo dai parametri a valori numerici, i parametri a valori non numerici non ne tengono conto in realtà, ma per convenzione ogni azione che li riguarda avrà valore "2" nel rispettivo campo "regola", per indicare comunque

l'intenzione di modificare il parametro con un valore X se questo assume valore uguale a un valore Y.

Di particolare rilievo è, dunque, l'implementazione del costruttore di EPAction, mostrata in Codice 3.2:

```
private int parameter;
private int oldValue;
private int rule;
private int newValue;


public EPAction(int parameter, int oldValue, int rule, int newValue){

    this.parameter = parameter;
    this.oldValue = oldValue;
    this.rule = rule;
    this.newValue = newValue;

}
```

Codice 3.2, Il costruttore della classe EPAction

Nel codice 3.2 sono stati implementati tutti i concetti sopracitati. La classe rispetta completamente l'interfaccia `IAction` di PIQLE, tra cui anche la questione della codifica per la memorizzazione delle coppie Q(s,a) in HashMap. La funzione `hashCode`() infatti è la funzione incaricata di codificare una determinata coppia Q(s,a) in un unico valore di tipo intero, che la differenzi da tutte le altre coppie che contengono valori diversi.

La strategia implementata per la codifica è descritta in Codice 3.3.

```
public int hashCode(){

    return parameter*10+oldValue*5+3*newValue+2*rule;

}
```

Codice 3.3, Implementazione codifica della classe EPAction

3.1.2.2 La classe EPState

La classe EPState ha il compito di rappresentare un determinato stato dell'ambiente in PIQLE. Il concetto di stato nell'ambito del problema considerato consiste nella particolare pagina consultata dall'utente e dalla configurazione degli elementi che la compongono. Per questo motivo l'implementazione della classe EPState, la classe incaricata di modellare lo stato dell'ambiente, deve avere la possibilità di memorizzare interamente la configurazione di una pagina, così come è rappresentata dal simulatore. Dal momento che il simulatore rappresenta le pagine come una collezione di oggetti di tipo Element (in particolare attraverso un Java ArrayList<Element>), anche lo stato dell'ambiente rappresenterà le pagine allo stesso modo. Durante la sua inizializzazione lo stato non farà altro che eseguire una copia della pagina correntemente visualizzata dall'utente e memorizzarla al suo interno. Per ottenere questa copia dovrà fare richiesta alla classe Universo citata nel capitolo precedente della pagina correntemente visualizzata dall'utente, la quale garantisce l'ottenimento della stessa versione. L'implementazione del costruttore di EPState svolge dunque un ruolo fondamentale e la sua implementazione è mostrata in Codice 3.4.

```
public EPState(IEnvironment env){

    super(env);

    EPEnvironment epenv = (EPEnvironment)env;

    ArrayList<Element> lista = epenv.getUniverse().getUser().getCurrentPage();

    elements = new ArrayList<Element>();

    for(Element e: lista){

        elements.add(new Element(e.getSize(),e.getFont()
                ,e.getAlignment(),e.getBackgroundColor(),
                e.getForegroundColor(),e.getLineHeight(),
                e.getNum(),e.getNumRead()));

    }


}
```

Codice 3.4, Il costruttore della classe EPState

La classe EPState rispecchia pienamente le specifiche dell'interfaccia IState, sia per quanto riguarda i concetti di uguaglianza tra stati che di copia di stati. Anche questa classe come EPAction, necessita di una strategia per poter essere codificata in un singolo valore di tipo intero, per poter essere memorizzata all'interno dell'HashMap come coppia Q(stato,azione). La strategia per la sua codifica è implementata nella rispettiva funzione `hashCode()` ed è mostrata in Codice 3.5.

```
public int hashCode(){

    int result = 0;

    for(Element e : elements){


        double alignmentInt = e.getLineHeight();

        int size = EPPresentation.getFontSizeIndex(e.getSize());
        int fontfamily = EPPresentation.getFontFamilyIndex(e.getFont(), 1);
        int alignment = EPPresentation.getAlignmentIndex(e.getAlignment(), 1);
        int lineHeight = EPPresentation.getLineHeightIndex(e.getLineHeight(), 1);

        result = size + 2 *fontfamily + 4 * alignment + 8*lineHeight;

    }

    return result;

}
```

Codice 3.5, Implementazione codifica della classe EPState

3.1.2.3 La classe EPEnvironment

L'ambiente è l'entità il cui compito è quello di descrivere il contesto del problema vero e proprio. Esso definisce le regole sulle quali l'intero framework deve basarsi essendo in esso definite le istruzioni per determinare la lista della azioni possibili a partire da un determinato stato, generare gli stati successivi al primo, generare le ricompense per una data azione, stabilire quando il proprio turno è terminato. La classe EPEnvironment è la classe incaricata di implementare tutti i concetti sopracitati. Come sopra citato questa classe deve poter interagire con l'utente, per questo motivo deve poterne avere accesso e di conseguenza è necessario che abbia un modo per riferirsi ad esso. Questo collegamento è consentito dalla classe EPUniverse, classe che implementa il concetto di classe Universo descritto nel capitolo precedente e che verrà descritta in dettaglio in seguito. Essa compare per la prima volta nel costruttore della classe EPEnvironment e il suo compito è proprio quello di fungere da pontefice tra l'ambiente e l'utente consentendone la comunicazione e garantendo l'unicità di entrambe le istanze. L'implementazione del costruttore della classe EPEnvironment prevede dunque l'ottenimento di un riferimento alla classe EPUniverse e essa è mostrata in Codice 3.6.

```
public EPEnvironment(EPUniverse universe){

    EPEnvironment.universe = universe;

    defaultInitialState = new EPState(this);

    System.out.println("Ambiente inizializzato\n");

}
```

Codice 3.6, Il costruttore della classe EPEnvironment

Dal momento che l'ambiente rappresenta il contesto del problema, verranno mostrate le parti più importanti.

Per quanto riguarda la generazione della lista di tutte le azioni effettuabili all'interno di un determinato stato, l'ambiente si affida alla funzione

`getActionList()`, la quale, prendendo in ingresso uno stato, (e quindi una particolare configurazione di una pagina) analizza tutti gli elementi presenti e per ognuno genera tutte le azioni normalmente effettuabili su di esso. Quindi includerà tutte le possibili configurazioni frutto della combinazione di tutti i valori assumibili dai suoi parametri. Se, per esempio, capita un elemento con allineamento sinistro, la funzione genererà le azioni necessarie per modificare quell'allineamento in tutte le altre possibili, quindi in allineamento centrato, a destra, giustificato. In questo modo, ogni volta l'ambiente genera realmente tutte le azioni possibili su una data pagina e, anche in presenza di un numero esiguo di elementi, è facile immaginare come la quantità delle azioni possibili aumenti velocemente. Le azioni generate comprendono, ovviamente, anche le azioni deleterie, ovvero le azioni che l'utente non accetterebbe ed è proprio grazie all'intervento dell'algoritmo di apprendimento per rinforzo che queste azioni non preferibili dall'utente non vengono più prese in considerazione con il passare del tempo, ed è proprio in questo modo che è possibile determinare l'efficacia dell'algoritmo stesso. La parte fondamentale di questa funzione è quella in cui si generano le azioni a partire dal un singolo elemento e la sua implementazione è mostrata in Codice 3.7.

```
 //FONT SIZE
for(int newSize = 0; newSize<11;newSize++){
     if(newSize>=EPPresentation.getFontSizeIndex(fontSize)){
         actionList.add(new EPAction(0,EPPresentation.getFontSizeIndex(fontSize),1,newSize));
     }
 }
 for(int j = 0; j<4; j++){

     //LINE HEIGHT
     if( j != EPPresentation.getLineHeightIndex(lineheight, 1)){

         actionList.add(new EPAction(3,EPPresentation.getLineHeightIndex(lineheight, 1),2,j));

     }
     //ALIGNMENT
     if(j != EPPresentation.getAlignmentIndex(alignment, 1)){

          actionList.add(new EPAction(2,EPPresentation.getAlignmentIndex(alignment, 1),2,j));

     }
     //FONT FAMILY
     if(j != EPPresentation.getFontFamilyIndex(fontFamily, 1)){

         actionList.add(new EPAction(1,EPPresentation.getFontFamilyIndex(fontFamily, 1),2,j));
     }

 }
```

Codice 3.7, Implementazione della creazione delle azioni possibili

Da notare come la funzione faccia affidamento sulla rappresentazione dei parametri fornita dalla classe EPPresentation.

Un'altra funzione fondamentale per l'ambiente, in quanto si occupa di effettivamente di fargli cambiare stato, è quella che riguarda la generazione di un nuovo stato a partire da uno stato e una azione. Questa funzione prende il nome di `successorState()` ed è sempre una funzione presente nell'interfaccia `IEnvironment`. Il compito di base di questa funzione è quello di prendere in ingresso uno stato e un azione e restituire un nuovo stato frutto dell'applicazione dell'azione sullo stato ricevuti in ingresso. Dal momento che questa funzione viene richiamata nel momento in cui l'ambiente intende applicare effettivamente un'azione precedentemente scelta dall'algoritmo, è proprio in questo momento che la modifica dell'ambiente sulla pagina dell'utente viene proposta. Per questo motivo prima di generare effettivamente un nuovo stato da codificare e memorizzare, l'ambiente chiede il parere dell'utente sull'effettiva utilità dell'azione. In base alla valutazione dell'utente, la funzione applica o meno la modifica. In seguito, l'ambiente assocerà quella stessa ricompensa a quella azione ed apprenderà. In Codice 3.8 è mostrato il primo approccio della funzione di fronte alla richiesta di

generazione di un nuovo stato, nella quale chiede una valutazione da parte dell'utente.

```
public IState successorState(IState s, IAction a) {


    if(EPConfiguration.actionVerbosity){
        printAction(a);
    }


    EPState oldState = (EPState) s;
    EPState newState=(EPState) s.copy();

    EPState eps = (EPState) s;
    EPAction epa = (EPAction) a;

    double userEvaluation = universe.getUser().evaluateProposal(epa);


    if(userEvaluation >= 0.0){

        //System.out.println("Ambiente: Agisco");

        int actionParameter = epa.getParameter();
        int actionOldValue = epa.getOldValue();
        int actionNewValue = epa.getNewValue();
        int actionRule = epa.getRule();
```

Codice 3.8, la funzione successorState()

In Codice 3.9, invece, viene mostrata l'implementazione della funzione nel momento in cui modifica lo stato ricevuto in ingresso per quanto riguarda modifiche indirizzate ai parametri di famiglia di carattere e interlinea.

```
else if(actionParameter == EPPresentation.getParameterIndex("alignment")){

    if(EPPresentation.getAlignmentIndex(e.getAlignment(),1) == actionOldValue){

        e.setAlignment(EPPresentation.getAlignment(actionNewValue));

    }

}

else if(actionParameter == EPPresentation.getParameterIndex("lineheight")){

    if(EPPresentation.getLineHeightIndex(e.getLineHeight(),1) == actionOldValue){

         e.setLineHeight(EPPresentation.getLineHeight(actionNewValue));

    }

}
```

Codice 3.9, Prima parte dell'applicazione della modifica

Una volta terminata l'applicazione della modifica sullo stato preso in ingresso, la funzione andrà a sostituire nella pagina effettivamente consultata dall'utente gli elementi che ha appena modificato, rendendo così disponibile all'utente la pagina correttamente adattata. Questo passaggio è mostrato in Codice 3.10.

```
universe.getUser().setPage(newState.getElements());
```

Codice 3.10, Applicazione vera e propria della modifica

Come già accennato nel capitolo precedente, la generazione della ricompensa per una determinata azione non dipende dall'ambiente, come canonicamente è fatto nella documentazione di PIQLE, ma dipende dall'utente stesso. La funzione di interfaccia `getReward()`, dunque, non farà altro che chiedere il parere dell'utente per poi associarlo all'azione e agli stati interessati dall'azione e tutto questo è permesso sempre attraverso la classe EPUniverse come mostrato in Codice 3.11.

```
double reward = 0.0;
reward = universe.getUser().evaluateProposal((EPAction)a);
return reward;
```

Codice 3.11, ottenimento della ricompensa da parte dell'utente

Un’altra funzione fondamentale svolta dall'ambiente è quella di determinare se uno stato è lo stato finale oppure no. Il concetto di stato finale può avere varie interpretazioni. Per stato finale, in generale, si intende lo stato che l'ambiente considera conclusivo, che sancisce la fine della sua esecuzione e il raggiungimento di una soluzione del problema. Nell'ambito di questo problema, lo stato finale può essere determinato da varie situazioni. Si potrebbe infatti definire stato finale lo stato in cui l'algoritmo ha proposto e correttamente effettuato tutte le modifiche necessarie al fine di ottenere una pagina perfettamente adattata sulla base delle preferenze dell'utente, oppure si potrebbe definire come stato finale lo

stato in cui l'algoritmo ha proposto e correttamente effettuato una determinata percentuale di modifiche sulla pagina secondo le preferenze dell'utente. Si può determinare come stato finale lo stato in cui l'algoritmo ha proposto anche solo una singola modifica coerente con le preferenze dell'utente oppure si potrebbe determinare come stato finale lo stato in cui le modifiche che l'utente dovrebbe effettuare sulla relativa pagina sono minori di quelle che avrebbe dovuto effettuare se l'algoritmo non fosse intervenuto. Quest'ultimo caso è quello che verrà preso in considerazione durante i test. Per come è stato implementato il simulatore, non è possibile tenere traccia di un numero preciso di modifiche effettuate dal momento che le pagine sono rappresentate come collezioni di tipi di elementi, e non come collezioni di singoli elementi. In questo modo una modifica fatta ad un tipo di elemento interessa tutti gli elementi di quel tipo, di conseguenza le modifiche sono eseguite in blocchi. Non avrebbe quindi senso definire lo stato finale, in questo contesto, come lo stato in cui è stata eseguita una determinata percentuale di modifiche sulle modifiche totali da eseguire. Si possono, però, fare confronti sulla base del numero di modifiche che l'utente dovrebbe effettuare sulla pagina se l'algoritmo non entrasse in azione e il numero di modifiche effettivamente richieste dall'utente dopo l'intervento dell'algoritmo. Per questo motivo l'implementazione della funzione `isFinal()`, funzione definita dall'interfaccia IEnvironment incaricata di determinare se l'ambiente è giunto ad uno stato finale, considererà uno stato come definitivo nel caso in cui l'algoritmo abbia generato almeno una modifica corretta.

L'implementazione della valutazione sull'effettivo raggiungimento dello stato finale è mostrata in Codice 3.12.

```
int modifichetotalidafare = universe.getUser().modifications();

int modificheancoradafare = universe.getUser().getModificationsFromState(s1);


if(modificheancoradafare<modifichetotalidafare){

    isFinal = true;

}
```

Codice 3.12, Strategia di determinazione dello stato finale

La funzione, mostrata in codice 3.12, prima di tutto interroga l'utente e ottiene il numero di modifiche totali che esso dovrebbe effettuare se l'algoritmo non intervenisse mai, in seguito interroga nuovamente l'utente chiedendogli quante modifiche dovrebbe applicare sullo stato corrente della pagina. Se il numero di modifiche da effettuare sul momento sulla pagina è minore delle modifiche che l'utente avrebbe dovuto applicare se l'algoritmo non fosse intervenuto, allora lo stato preso in esame è considerato finale.

3.1.3.4 La classe User

La classe User all'interno del simulatore è la classe astratta che modella il comportamento di un utente generico. In essa sono definite le principali azioni comuni alla rappresentazione di qualsiasi utente come, ad esempio, l'apertura di una nuova pagina. La modellazione di ogni utente all'interno del simulatore avviene attraverso l'estensione di questa classe, in particolare nell'implementazione della funzione adaptPage(), la funzione che rappresenta la ricerca e il conteggio del numero di modifiche che l'utente effettuerebbe per ottenere una pagina perfettamente formattata secondo le sue esigenze. All'interno di questa funzione sono rappresentate le preferenze dell'utente, in quanto è attraverso di essa che si manifestano. La funzione ha l'incarico di scorrere ogni tipo di elemento presente nella pagina consultata dall'utente e per ognuno di essi

verificare che rispetti determinate caratteristiche. L'implementazione di questa funzione è generica ed è mostrata in Codice 3.13.

```
public void adaptPage() {

        int modificheTotali = 0;

        ArrayList<Element> lista = loadPage();

        for (int i=0; i<lista.size(); i++) {

                Element e = lista.get(i);


                int modifiche = analizzaElemento(universe,e);

                modificheTotali += modifiche;

        }

        modifiedElements = modificheTotali;

}
```

Codice 3.13, La funzione adaptPage()

Ogniqualvolta un tipo di elemento non rispetti i parametri desiderati dall'utente, essa incrementa un contatore pari al numero di elementi di quel tipo oggetto di

```
if (e.getAlignment() != 'l' ){

    EPAction action = new EPAction(2,
            EPPresentation.getAlignmentIndex(e.getAlignment(),1)
            , 2 , EPPresentation.getAlignmentIndex('l',1)  );

        for(int i = 0; i<e.getNumRead(); i++){
            universe.sendToSelector(action, BEST);
        }

        modifiche += e.getNumRead();

}
if (e.getLineHeight() < 1.5 ){

    EPAction action = new EPAction(3,
            EPPresentation.getLineHeightIndex(e.getLineHeight(),1) ,
            2 , EPPresentation.getLineHeightIndex(2.0,1)  );

    for(int i = 0; i<e.getNumRead(); i++){
            universe.sendToSelector(action, BEST);
    }

    modifiche += e.getNumRead();

}
```

interesse per l'utente. Il compito di analizzare il singolo tipo di elemento è affidato ad una seconda funzione, chiamata di volta in volta dalla funzione `adaptPage()` il cui nome é `analizzaElemento()`. Per cui una volta definiti i parametri di preferenza per un determinato utente, occorre inserirli correttamente all'interno di quest'ultima funzione. Presa, ad esempio, la modellazione di un utente con difficoltà visive come l'ipovisione, le cui preferenze sono state individuate in dimensione di carattere pari a 16 o superiore, famiglia di carattere Times New Roman o Serif, allineamento a sinistra e interlinea di 1,5 o superiore, la sua funzione caratteristica `analizzaElemento()` andrà a verificare che ogni elemento rispetti tali parametri. In Codice 3.14 è mostrata la porzione di codice della funzione associata a questo utente, in particolare nel momento in cui verifica i valori assunti da un elemento nei parametri di allineamento e interlinea.

```
if (e.getAlignment() != 'l' ){

    EPAction action = new EPAction(2,
            EPPresentation.getAlignmentIndex(e.getAlignment(),1)
            , 2 , EPPresentation.getAlignmentIndex('l',1)  );

        for(int i = 0; i<e.getNumRead(); i++){
            universe.sendToSelector(action, BEST);
        }

        modifiche += e.getNumRead();

}
if (e.getLineHeight() < 1.5 ){

    EPAction action = new EPAction(3,
            EPPresentation.getLineHeightIndex(e.getLineHeight(),1) ,
            2 , EPPresentation.getLineHeightIndex(2.0,1)  );

    for(int i = 0; i<e.getNumRead(); i++){
            universe.sendToSelector(action, BEST);
    }

    modifiche += e.getNumRead();

}
```

Codice 3.14, la funzione `analizzaElemento()`

L'esecuzione della funzione `analizzaElemento()` rappresenta l'azione dell'utente di individuazione e modifica degli elementi di una pagina, ovvero rappresenta il comportamento dell'utente. Dal momento che l'obiettivo dell'integrazione tra i due sistemi è quello di far apprendere all'algoritmo le preferenze dell'utente in base al suo comportamento, l'algoritmo deve essere in grado di percepire le azioni svolte all'interno di questa funzione. Come mostrato in Codice 3.14, la funzione, oltre ad individuare gli elementi che non soddisfano le sue preferenze, ha anche l'incarico di informare l'algoritmo delle azioni che ha svolto. Per fare questo va a richiamare direttamente l'algoritmo di apprendimento attraverso la funzione `sendToSelector()` e gli invia la rappresentazione delle azioni svolte, trasmettendo anche la ricompensa che assocerebbe a tali azioni se fosse l'algoritmo a proporle, ed ovviamente quest'ultimo valore è quello massimo consentito, dal momento che è proprio l'utente a compiere quelle azioni.

La funzione originale di conteggio è stata mantenuta cambiandole semplicemente nome in quanto svolge il fondamentale ruolo di generare il numero di modifiche totale da effettuare sulla pagina senza andare a trametterе nulla all'algoritmo. La funzione si chiama `getNModifications()` e sarà sfruttata dall'ambiente in PIQLE per determinare lo stato finale, assieme alla funzione `getNModificationsFromState()`, la quale invece esegue lo stesso conteggio disinteressandosi dell'algoritmo, ma a partire da uno stato EPState del framework e non da un pagina del simulatore.

Il compito di generare una ricompensa per l'ambiente da parte dell'utente a fronte di una proposta di modifica, è affidato alla funzione `evaluateProposal()`, la quale prendendo in ingresso una EPAction, valuta quanto essa sia coerente con le caratteristiche dell'utente. La ricompensa può assumere cinque valori, ovvero WORST, BAD, NEUTRAL, GOOD, BEST, per implementare l'elasticità della valutazione. L'esecuzione di questa funzione rappresenta un'altra fonte fondamentale di informazioni per l'algoritmo di apprendimento, e la sua implementazione è simile a quella di `analizzaElemento()`, la differenza consiste nel fatto che deve generare un valore e non contare il numero di modifiche.

La classe EPUniverse

Come descritto in precedenza, questa classe rappresenta il contesto generale all'interno del quale il simulatore e l'algoritmo possono interagire. Questa classe ha il compito di inizializzare ogni componente necessario all'esecuzione del framework PIQLE, inizializzare l'utente e di fungere da tramite tra i due macro-sistemi. La creazione di un particolare tipo di utente avviene attraverso un parametro, come mostrato in Codice 3.15, per facilitare lo sviluppo dei test. Ogni parametro necessario alla calibrazione dei due sistemi é stato definito in una classe esterna chiamata EPConfiguration e descritta in seguito. In Codice 3.16 è

mostrata l'inizializzazione e calibrazione del framework PIQLE, dove viene appunto istanziata la classe dell'algoritmo, la classe dell'ambiente, la classe dell'agente, la classe dell'arbitro.

```
int userID = EPConfiguration.userID;

if(userID == 0){

    user = new DyslexicUser(this);

}else if(userID == 1){

    user = new VisuallyImparedUser(this);

}else if(userID == 2){

    user = new ElderlyUser(this);

}else if(userID == 3){

    user = new InconsistentUser(this);

}else if(userID == 4){

    user = new DynamicUser(this);

}
```

Codice 3.15, Inizializzazione

```
//CREAZIONE AMBIENTE
environment = new EPEnvironment(this);

//INIZIALIZZAZIONE ALGORITMO

sql=new WatkinsSelector(0.95);

sql.setExponentialAlphaDecay();



//CREAZIONE AGENTE
zero07=new LoneAgent(environment,sql);

//CREAZIONE ARBITRO
arbitre=new OnePlayerReferee(zero07);
arbitre.setMaxIter(maxIter);
```

Codice 3.16, Inizializzazione del framework PIQLE

La classe EPUniverse ha inoltre il compito di coordinare le azioni dei due sistemi, che è effettuato durante l'esecuzione della funzione `execute()`, la cui implementazione è mostrata in Codice 3.17.

```
public void execute() throws IOException{


        EPState ep = new EPState(environment);


        //L'AMBIENTE INTERVIENE SULLA PAGINA
        arbitre.episode(ep);


        //L'UTENTE INTERVIENE SULLA PAGINA
        user.adaptPage();


        //UTENTE CAMBIA PAGINA

        user.changePage();

        //SETTINGS
        step++;

        epsi*=factor;
        sql.setEpsilon(epsi);



}
```

Codice 3.17, la funzione execute()

La classe EPConfiguration, configurazione

Come precedentemente accennato ogni parametro di configurazione è disponibile in una classe dedicata chiamata EPConfiguration, in modo da rendere facilmente accessibile ogni valore e velocizzare la calibrazione del sistema finale. Ogni campo presente all'interno della classe è stato dichiarato "`final`" per impedire in qualsiasi modo modifiche durante l'esecuzione della simulazione. La configurazione prevede l'utilizzo dell'algoritmo WatkinsSelector, ovvero l'algoritmo di apprendimento per rinforzo definito da Watkins nel 1989 chiamato Q-Learning, la cui memorizzazione delle coppie $Q(s, a)$ è basata sulle HashMap. Per quanto riguarda i parametri dell'algoritmo è stato scelto il decadimento esponenziale, in quanto secondo la documentazione garantisce convergenza, un valore Epsilon di partenza uguale a 1, e un fattore di diminuzione ad episodio per Epsilon di 0,95, attraverso i quali è garantito un apprendimento da parte del WatkinsSelector dopo circa 500 passi. Il numero di pagine visualizzate dall'utente durante la simulazione è variabile e dipende dalla dimensione della curva di apprendimento mostrata dall'algoritmo.

3.2 Test

A implementazione ultimata sono stati eseguiti alcuni test per verificare l'efficacia dell'algoritmo di apprendimento di fronte al problema oggetto della tesi, ovvero la profilazione utente basata sul contesto e il suo comportamento. I test riguardano il comportamento dell'algoritmo di fronte alle cinque tipologie di utente descritte nel primo capitolo, ovvero la rappresentazione di un utente con difficoltà visive legate alla dislessia, un utente con difficoltà visive legate all'ipovisione, un utente con difficoltà visive legate all'età, un utente dal comportamento incoerente, un utente che cambia preferenze con il passare del tempo. I risultati dei test sono proposti attraverso diagrammi cartesiani, sull'asse delle ascisse è presente il numero delle pagine aperte, sull'asse delle ordinate è presente il numero di errori commessi

dall'algoritmo, ovvero il numero di proposte non accettate dall'utente in quanto non coerenti con le sue preferenze. Nel seguito di questo capitolo mostreremo i grafici relativi ad alcuni tipi di test che sono stati effettuati.

In Figura 3.1 è mostrato il risultato del test effettuato sulla rappresentazione di un utente con disturbi visivi legati alla dislessia, è possibile notare chiaramente la curva di apprendimento dell'algoritmo, che intorno al passo numero 500 effettivamente cala e si stabilizza intorno allo zero. L'algoritmo, dunque, risulta efficace di fronte a questo tipo di preferenze.

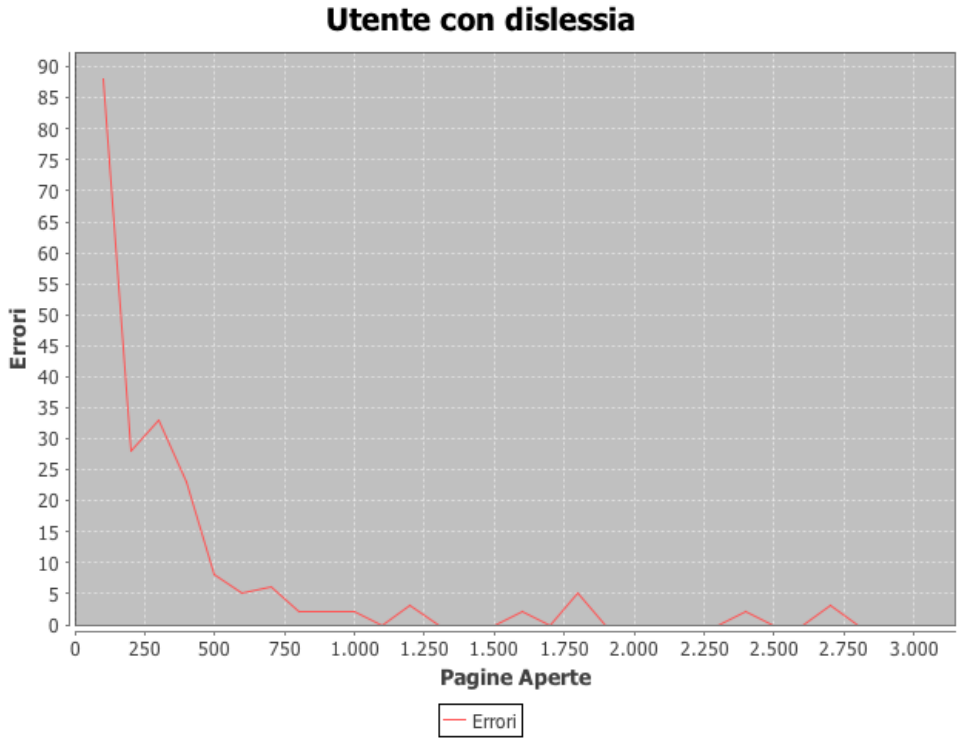


Figura 3.1, test effettuati su un utente con dislessia

In Figura 3.2 è mostrato il risultato di un test effettuato sulla rappresentazione di un utente con difficoltà visive legate all'ipovisione, anche in questo caso è possibile notare la curva di apprendimento dell'algoritmo che intorno al passo 500 tende a zero e si stabilizza su di esso. L'algoritmo, dunque, é efficace anche di fronte a queste esigenze.

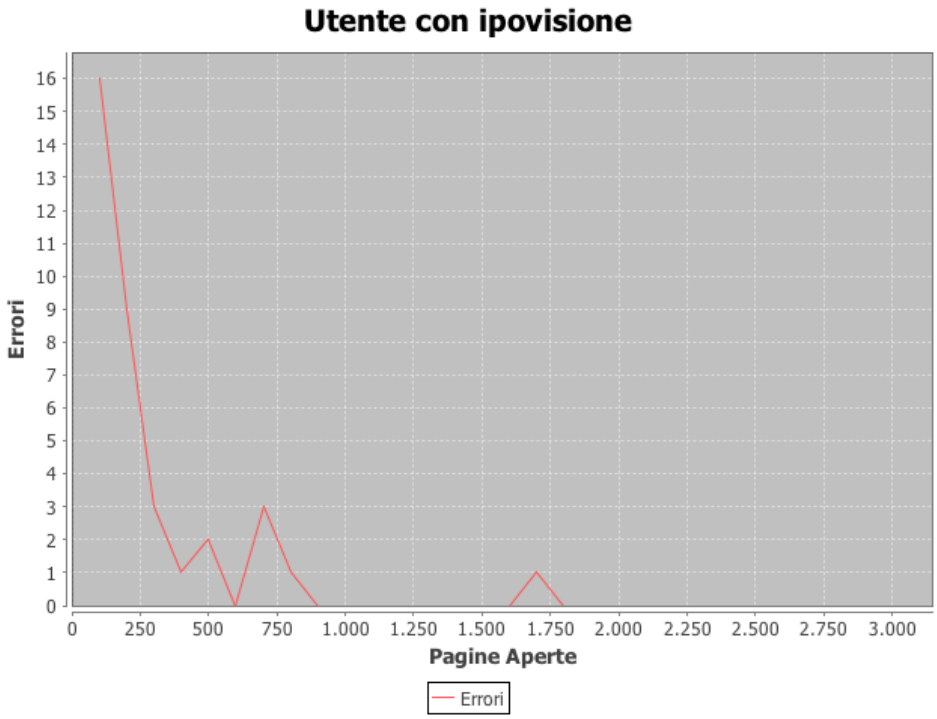


Figura 3.2, Test effettuato su un utente con ipovisione

In Figura 3.4 e in Figura 3.5, è mostrato il risultato di un test effettuato sulla rappresentazione di un utente incoerente, ovvero che con una probabilità del 5% restituisce all'algoritmo valutazioni incoerenti tra loro rispetto alle sue preferenze. Come è possibile notare dal grafico non è possibile individuare una curva di apprendimento, ne per un numero relativamente basso ne per un numero relativamente grande di passi. L'algoritmo, dunque, dimostra di non essere in grado di apprendere nel caso di un utente non coerente con le proprie preferenze.

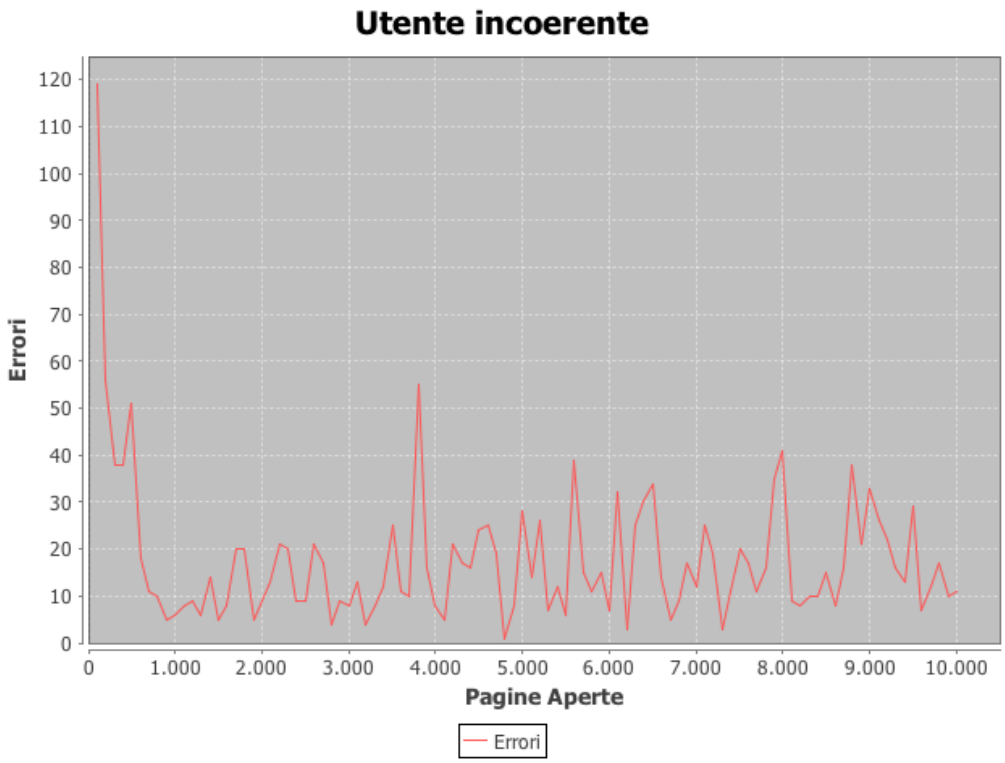


Figura 3.4, Primo test effettuato su un utente incoerente

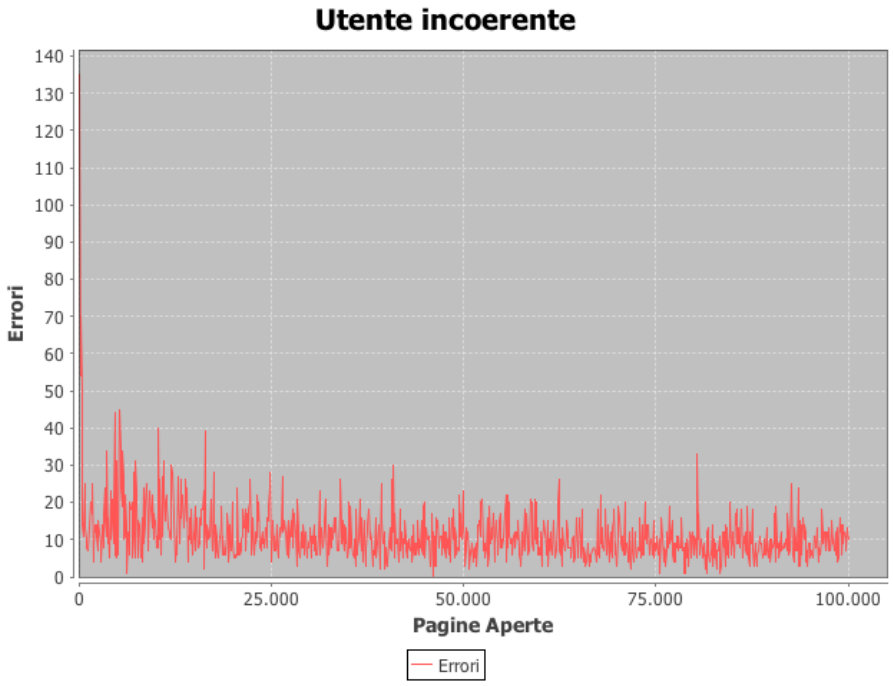


Figura 3.5, Secondo test effettuato su un utente incoerente

Infine, in Figura 3.6 e in Figura 3.7 sono mostrati i risultati di alcuni test effettuati su un utente che in un determinato momento effettua un cambio di preferenze

quindi comportamento. Dai grafici emergono due curve di apprendimento, una è quella iniziale comune agli altri test, la seconda è quella che si delinea al cambio di preferenze ma ha un picco molto alto. Questo significa che l'algoritmo è in grado di ricominciare ad apprendere a seguito di un cambio di preferenze, purché l'utente mantenga coerenza, questo a discapito di un elevato numero di errori iniziali al cambio. In Figura 3.6, inoltre, è mostrato il fatto che maggiore è il numero di pagine aperte prima del cambio, maggiore sarà il numero di errori effettuati dall'algoritmo al cambio.

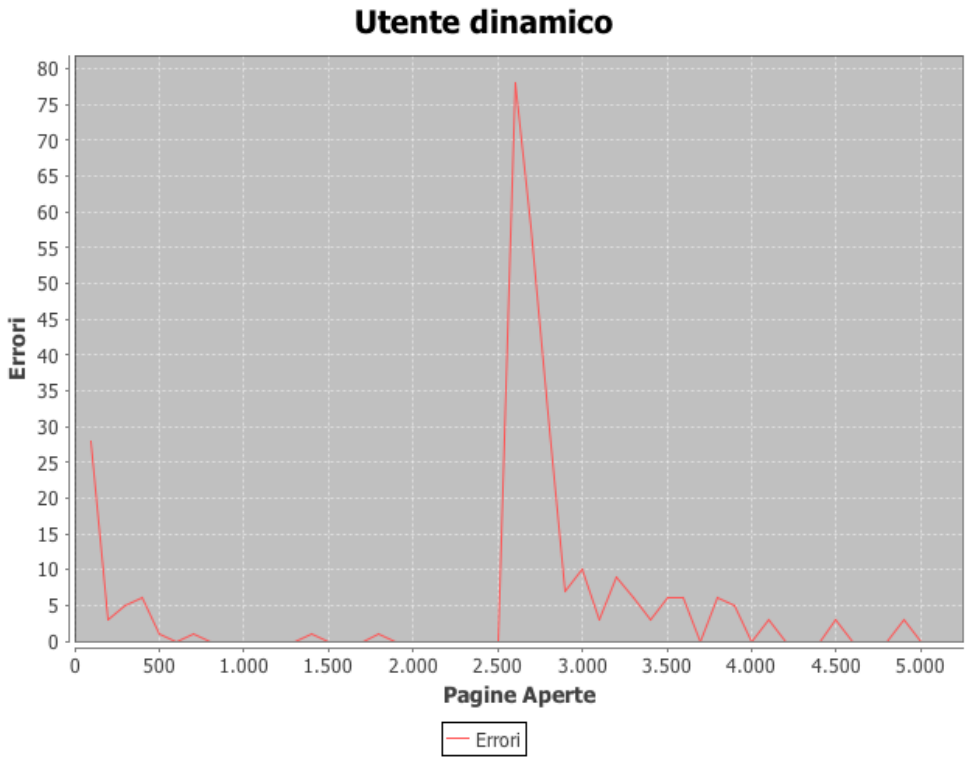


Figura 3.6, Test n.1 effettuato su un utente che cambia dinamicamente preferenze

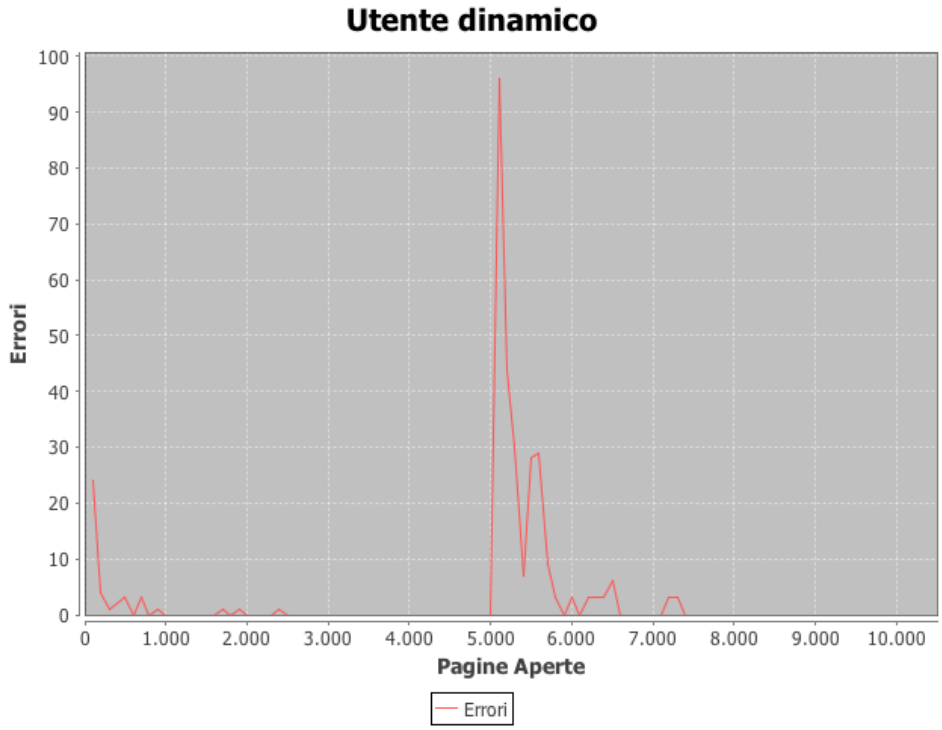


Figura 3.7, Test n.2 effettuato su un utente che cambia dinamicamente preferenze

Conclusione

Nell'ambito di questa tesi é stata studiata l'efficacia dell'applicazione di un algoritmo di apprendimento per rinforzo, in particolare il Q-Learning, come soluzione al problema della profilazione utente. E' stato dimostrato che se l'utente mantiene una determinata coerenza con le proprie preferenze, l'algoritmo é in grado di percepire quali siano le preferenze dell'utente osservando il suo comportamento. Per trarre queste conclusioni è stato necessario rappresentare il contesto all'interno del quale l'algoritmo potesse muoversi, è stato necessario c'è implementare un ambiente che descrivesse il problema della profilazione utente. Questo é stato reso possibile grazie al framework PIQLE, il quale nasce come strumento indirizzato all'implementazione di problemi o giochi nei quali é necessario l'intervento di un algoritmo di apprendimento per rinforzo. L'implementazione di tale contesto all'interno di PIQLE é stata progettata per fare in modo che l'algoritmo potesse essere in grado di osservare quanti più casi possibili, più precisamente quante più tipologie di utente possibili, in modo da rendere universale la significatività dei test. Per rappresentare questo concetto non é stata inserita l'idea di tipologia di utente all'interno del framework, ma é stato progettato per essere in grado di delineare le preferenze di qualsiasi tipo di utente. Per eseguire i test sull'idoneità dell'algoritmo di apprendimento lo scopo preposto, é stato necessario simulare in qualche modo un utenza Web e mettere alla prova l'algoritmo di fronte a varie tipologie di utenza. Per questo scopo si è rivelato decisamente efficace l'utilizzo di un simulatore di utenza Web chiamato ExTraS, un sistema software progettato e sviluppato come progetto interno di alcuni professori dell'Università di Bologna. Tale simulatore rende possibile la riproduzione del comportamento di un utente durante una classica navigazione Web, durante la quale l'utente analizza le pagine Web visitate ed conta il numero di modifiche che esso apporterebbe agli elementi all'interno della pagine stesse al fine di ottenere un pagina perfettamente formattata secondo le sue esigenze.

Grazie ad ExTraS é stato possibile simulare il comportamento di alcune tra le più diffuse tipologie di utente caratterizzate dalla difficoltà nella fruizione di contenuti testuali durante la navigazione Web. E' stato così possibile rappresentare il comportamento di un utente con dislessia, un utente con ipovisione e un utente anziano, considerate appunto come le casistiche maggiormente diffuse di utenza caratterizzata dalla difficoltà di lettura determinate da alcuni aspetti di presentazione relativi agli elementi delle pagine Web. L'integrazione dei due sistemi ha reso possibile l'esecuzione di alcuni test che hanno portato alla luce l'effettiva validità degli algoritmi di apprendimento per rinforzo di fronte al problema della profilazione utente, in quanto tale integrazione permette all'algoritmo di osservare il comportamento dell'utente simulato e quindi di effettuarne la profilazione. I risultati dei test riflettono il fatto che l'algoritmo di apprendimento non presenta problemi per quanto riguarda l'apprendimento del comportamento di utenti dal comportamento coerente e costante. Infatti laddove l'utente mantenga costante la propria coerenza in termini comportamento e valutazioni delle proposte di modifiche, è possibile osservare chiaramente dai grafici ottenuti dai test la curva di apprendimento dell'algoritmo. Al contrario laddove l'utente non sia strettamente coerente con le sue preferenze nei confronti dell'algoritmo nel momento in cui deve fornire una valutazione sulle modifiche da lui proposte, é stato osservato che l'algoritmo non riesce mai a convergere ad una soluzione. E' inoltre emerso che l'algoritmo non presenta problemi dal punto di vista dell'apprendimento di fronte ad un cambio di preferenze dell'utente, purché esso mantenga la propria coerenza dell'utente. In generale si é dimostrato che un algoritmo di apprendimento per rinforzo può rappresentare un valido supporto per la profilazione utente basata unicamente sul suo comportamento. Il modello sviluppato nell'ambito di questa tesi può rappresentare un punto di partenza per vari progetti futuri. Si possono ad esempio sviluppare e studiare casistiche più complesse del stesso problema, coinvolgendo più aspetti legati agli elementi.

L'implementazione sviluppata durante questa tesi infatti, non tiene conto di tutti i parametri che costituiscono gli elementi di una pagina Web, ma solo di quelli principali ovvero la dimensione del testo, la famiglia del carattere del testo, l'allineamento e l'interlinea. In realtà i parametri che possono rappresentare criticità durante la navigazione Web sono molti altri, come ad esempio i colori di

sfondo, i colori dei caratteri, il contrasto degli elementi. Effettuare uno studio completo riguardo a tutte le casistiche fonti di disagio per l'utente consentirebbe in futuro lo sviluppo di browser capaci di imparare il comportamento dell'utente e consentire ad esso un utilizzo sempre più agevole, lo stesso discorso si può applicare ai sistemi operativi e alle loro interfacce grafiche. In generale lo studio e implementazione di sistemi atti a facilitare la fruizione di contenuti digitali e in generale dei dispositivi elettronici che ne permettono l'ottenimento, é fondamentale perché consentirebbe un coinvolgimento maggiore nella rivoluzione culturale a cui stiamo assistendo da parte di quelle persone che al giorno d'oggi faticano a inserirvisi.

Conclusione

Bibliografia

Comitè F. "PIQLE, a platform for Implementation of Q-Learning Experiments", 2006

Ferretti S., Mirri S., Prandi C., Salomoni P., "User Centered and Context Dependent Personalization Through Experimental Transcoding" – 2013

Pratelli M. , "Difficoltà di apprendimento e dislessia", Junior Editore, Bergamo, 2004

Sutton R. S., Barto A. G. , "Reinforcement Learning, An Introduction" , 1998

"Nuovissimo Dardano Dizionario della lingua italiana" , Armando Curcio Editore, Roma, 1982

"Tesauro del Nuovo Soggettario", Editrice Bibliografica, Milano, 2006