POLITECNICO DI MILANO
Corso di Laurea Magistrale in Ingegneria Informatica
Facoltà di Ingegneria Industriale e dell'Informazione

Algoritmi di Ricerca ad Albero Monte Carlo Applicati all'Intelligenza Artificiale nel Gioco della Briscola a Cinque

Relatore: Prof. Pier Luca Lanzi
Correlatore: Spartaco Albertarelli

Tesi di Laurea di:
Andrea Villa, Matr. 780690

Anno Accademico 2012-2013

Alla mia famiglia e alla mia Marta...

Prefazione

Questa tesi si inquadra nell'ambito dell'intelligenza artificiale nei videogiochi, in particolare nei giochi da tavolo e di carte.

Lo scopo della tesi è quello di sviluppare un algoritmo di intelligenza artificiale competitivo per il gioco della Briscola a 5. Altro scopo della tesi è quello di valutare le performance dell'algoritmo di Ricerca ad Albero Monte Carlo applicato alla Briscola a 5. Abbiamo creato quattro diversi algoritmi basati sulla Ricerca ad Albero Monte Carlo e poi messi a confronto con un algoritmo di intelligenza artificiale a regole che abbiamo sviluppato con l'aiuto di esperti del settore e libri specializzati.

I risultati sperimentali hanno mostrato che gli algoritmi basati sulla Ricerca ad Albero Monte Carlo hanno performance superiori rispetto all'algoritmo a regole. Inoltre abbiamo quantificato l'influenza di una fase di chiamata gestita in maniera adeguata sulla percentuale di vittoria delle due squadre.

I risultati ottenuti con questo lavoro ci hanno permesso di sviluppare un prototipo dell'applicazione del gioco della Briscola a 5 per iOS e Android.

Ringraziamenti

Il mio primo ringraziamento è rivolto al Prof. Lanzi e a Spartaco Albertarelli per avermi offerto l'opportunità di realizzare questa tesi e per l'aiuto, i consigli e il tempo dedicatomi durante tutto il lavoro.

Un grande grazie alla mia famiglia, a mio padre, a mia madre e a mio fratello Alessandro, che non mi hanno mai fatto mancare l'affetto e il supporto durante tutti questi anni.

Infine, un ringraziamento alla mia ragazza, Marta, che mi ha sempre sostenuto in tutto questo tempo e ha sempre saputo regalarmi un sorriso.

Indice

Elenco delle figure

Elenco delle tabelle

Elenco degli algoritmi

Capitolo 1

Introduzione

Questa tesi si inquadra nell'ambito dell'intelligenza artificiale nei videogiochi, in particolare nei giochi da tavola e di carte. Questo settore è nato negli anni '50 e i primi algoritmi di intelligenza artificiale, sviluppati per giochi da tavolo a due giocatori (come la dama e gli scacchi), erano in grado di eseguire solo le mosse finali di una partita o, al massimo, di competere con giocatori principianti. Negli anni la ricerca in questo campo è riuscita non solo a creare algoritmi in grado di competere con giocatori umani professionisti, ma anche di risolvere giochi, ovvero di predire, partendo da qualsiasi posizione, il risultato finale di una partita in cui i giocatori giocano in maniera perfetta.

Lo scopo di questa tesi è quello di sviluppare un algoritmo di intelligenza artificiale competitivo per la Briscola a 5, una variante della popolare Briscola, uno dei tre giochi di carte più giocati in Italia. Altro scopo della tesi è quello di valutare le performance del metodo chiamato Ricerca ad Albero Monte Carlo (o Monte Carlo Tree Search, MCTS) applicato alla Briscola a 5. L'algoritmo MCTS è stato introdotto nel 2006 da Kocsis e Szepesvari [5]. A differenza dei metodi classici (come il Minimax), che sviluppano interamente l'albero di ricerca, questo algoritmo lo costruisce progressivamente e lo esplora in profondità. L'esplorazione può essere fermata in qualsiasi

momento, ottenendo come risultato la stima calcolata fino a quell'istante. Questa proprietà rende la Ricerca ad Albero Monte Carlo molto efficiente in termini di tempo e memoria. Kocsis e Szepesvari [5] hanno dimostrato che il metodo MCTS, avendo a disposizione abbastanza iterazioni, converge al risultato ottimo.

Per valutare l'efficacia del metodo basato sulla Ricerca ad Albero Monte Carlo abbiamo deciso di confrontarlo con un algoritmo di intelligenza artificiale a regole, sviluppato con l'aiuto di esperti del settore e testi specializzati. I risultati ci hanno permesso di valutare le performance del metodo MCTS, mostrando che l'approccio è competitivo.

Abbiamo sviluppato due versioni dell'applicazione una per scopi sperimentali e una per scopi ludici, grazie al motore di gioco Unity3D.

1.1 Contributi Originali

Questa tesi contiene i seguenti contributi originali:

- Lo sviluppo di una intelligenza artificiale a regole per il gioco della Briscola a 5.
- Lo sviluppo di quattro intelligenze artificiali basate sulla Ricerca ad Albero Monte Carlo per il gioco della Briscola a 5.
- Il confronto sperimentale tra gli algoritmi di intelligenza artificiale creati.
- Lo sviluppo di un framework di ricerca sul gioco della Briscola a 5.
- Lo sviluppo dell'applicazione del gioco della Briscola a 5 per Android e iOS.

1.2 Struttura della Tesi

La tesi è strutturata nel modo seguente.

Nel Capitolo 2 presentiamo una panoramica sullo sviluppo di algoritmi di intelligenza artificiale nei giochi da tavolo e di carte; in seguito presentiamo in dettaglio l'algoritmo Ricerca ad Albero Monte Carlo.

Nel Capitolo 3 descriviamo le regole della Briscola a 5.

Nel Capitolo 4 presentiamo le versioni di intelligenza artificiale a regole utilizzate nel nostro gioco.

Nel Capitolo 5 illustriamo gli algoritmi di intelligenza artificiale basati sulla Ricerca ad Albero Monte Carlo che abbiamo sviluppato per il nostro gioco.

Nel Capitolo 6 discutiamo i risultati ottenuti dagli esperimenti condotti sugli algoritmi da noi creati.

Nel Capitolo 7 effettuiamo alcune valutazioni sul lavoro svolto e gli sviluppi futuri.

Nell'Appendice A descriviamo in breve lo sviluppo e la struttura dell'applicazione .

Capitolo 2

L'Intelligenza Artificiale nei Giochi da Tavolo e di Carte

In questo capitolo illustriamo l'applicazione dell'intelligenza artificiale nei giochi da tavolo e di carte. Successivamente ci focalizziamo sul metodo di Ricerca ad Albero Monte Carlo, confrontandolo con il più classico algoritmo di intelligenza artificiale, il Minimax, discutendone i vantaggi e gli svantaggi.

2.1 Intelligenza Artificiale e Giochi da Tavolo

L'intelligenza artificiale nei giochi da tavolo nasce con lo scopo di creare un avversario in grado di simulare un comportamento razionale. Le prime applicazioni di intelligenza artificiale vengono presentate negli anni '50, quando Christopher Strachey inventa un programma per il gioco della dama, mentre Dietrich Prinz ne presenta uno per gli scacchi [13]. L'algoritmo di Prinz è in grado di trovare l'azione migliore a due mosse dallo scacco matto [10], cioè risolve i *problemi matto-in-due-mosse*, non è però in grado di giocare un'intera partita a causa dei limiti computazionali della macchina utilizzata, il Ferranti Mark I [24]. Successivamente Arthur Samuel implementa un algoritmo di intelligenza artificiale per la dama in grado di

competere con giocatori principianti [29]. Samuel introduce così l'algoritmo di ricerca chiamato Minimax con potatura alfa-beta. Il Minimax è un metodo per minimizzare la massima perdita possibile (o per massimizzare il minimo guadagno). Questo algoritmo, a partire dallo stato corrente, costruisce un albero completo degli stati del gioco e computa la decisione migliore facendo un calcolo ricorsivo [18]. A ogni passo l'algoritmo assume che, dei due giocatori, il primo cerchi di massimizzare le proprie probabilità di vincere e viceversa il secondo cerchi di minimizzarle. Il Minimax risulta essere molto dispendioso computazionalmente, soprattutto in domini in cui lo spazio degli stati è ampio, dato che l'albero deve essere completamente espanso e visitato. Per limitare il numero di nodi esaminati Samuel utilizza la potatura alfa-beta, che consiste nel terminare la valutazione di una possibile mossa appena viene dimostrato che è peggiore di una esaminata in precedenza. Samuel, con l'intento di migliorare il proprio programma, introduce il metodo da lui chiamato *apprendimento meccanico* [28]. Questa tecnica consiste nell'addestrare la sua intelligenza artificiale facendola giocare migliaia di volte contro se stessa [18]. Verso la fine degli anni '80 Jonathan Schaeffer et al. [18] iniziano a lavorare su Chinook, un programma per la dama sviluppato per personal computer. Basato sulla ricerca alfa-beta e su un database pre-calcolato con più di 400 miliardi di posizioni con 8 pezzi in gioco o meno, il programma di Schaeffer diventa campione del mondo nel '94. Lo stesso gruppo di ricerca riesce a risolvere nel 2007 il gioco della dama (con scacchiera classica 8 x 8), dimostrando che la partita giocata senza errori finisce in parità [19].

Gli scacchi, rispetto alla dama, sono un gioco molto più diffuso ma anche molto più complesso, il primo computer in grado di battere un umano a scacchi in condizioni regolamentari è Deep Thought nel 1989 [26]. La macchina, costruita dall'informatico dell'IBM Feng-hsiung Hsu, sconfigge David Levy scacchista di livello Grande Maestro, in una sfida ufficiale lanciata da que-

st'ultimo. Hsu entra poi nel progetto Deep Blue, un calcolatore progettato da IBM espressamente per giocare a scacchi. La forza di Deep Blue è dovuta alla sua elevata potenza computazionale, è infatti costituito da un calcolatore a parallelismo massivo con 480 processori. L'algoritmo per il gioco degli scacchi è scritto in C ed è capace di calcolare 100 milioni di posizioni al secondo. Le sue funzioni di valutazione sono state scritte con parametri determinati dal sistema stesso, analizzando migliaia di partite di campioni. La conoscenza degli scacchi del programma era stata finemente migliorata dal gran maestro di scacchi Joel Benjamin. La lista delle aperture fu fornita dai campioni Miguel Illescas, John Fedorowicz e Nick De Firmian [25]. Nel 1996 Deep Blue sconfigge il campione del mondo in carica Garry Kasparov nella prima partita, subendo però tre sconfitte e due pareggi in quelle seguenti, perdendo quindi la sfida. Nel 1997 l'ultima versione di Deep Blue sconfigge 3.5 a 2.5 Kasparov, diventando la prima macchina vincitrice in una sfida contro un campione del mondo. Le prestazioni dei software di scacchi sono in continuo miglioramento. Nel 2009 il software Pocket Fritz 4, installato su uno smartphone, vince un torneo di categoria 6, riuscendo a valutare circa 20000 posizioni al secondo [23].

Un altro gioco ampiamente studiato nel campo dell'intelligenza artificiale è il Go, il gioco da tavolo più popolare in Asia. La scacchiera del Go è un quadrato con 19 caselle per lato, il che rende il fattore di ramificazione in alberi di ricerca elevatissimo (361 per il primo livello), la conseguenza è l'inutilizzabilità dei metodi tradizionali (come il Minimax). Il primo programma sul Go viene fatto risalire agli anni 60', quando D. Lefkovitz [4] implementa un algoritmo basato sul riconoscimento di pattern. Successivamente Zobrist [4] scrive il primo programma in grado di battere un umano principiante. Il programma di Zobrist è basato principalmente sul calcolo di una funzione del potenziale che approssima l'influenza delle pietre sulla scacchiera. Negli anni '70 Bruce Wilcox [4] scrive il primo programma in grado di giocare

meglio di un principiante assoluto. Il suo algoritmo utilizza una rappresentazione astratta della scacchiera, dividendola in zone e sfruttando la teoria delle linee di settore. Qualche passo in avanti avviene alla fine degli anni 90, combinando la ricerca limitata a zone circoscritte di Wilcox e sfruttando il riconoscimento di pattern [18]. Queste tecniche ottengono risultati discreti, riuscendo a competere a livelli superiori rispetto a principianti, tuttavia i risultati migliori si hanno con l'applicazione dell'algoritmo di Ricerca ad Albero Monte Carlo, di cui trattiamo nella Sezione 2.3.

Visti gli ottimi risultati con la dama e gli scacchi e il Go, l'interesse per l'intelligenza artificiale si è estesa ad altri giochi, come per esempio il Backgammon. La maggiore difficoltà nel creare una IA adeguata in questo gioco è il fattore di incertezza legata al tiro coi dadi. L'aleatorietà presente in questo gioco rende infatti impossibile eseguire una ricerca ad albero profonda. Nel 1992 Gerry Tesauro [18] unendo il metodo di apprendimento di Samuel con tecniche di reti neurali riesce a creare un accurato valutatore di posizioni. Grazie a centinaia di migliaia di partite di addestramento, il suo programma, TD-Gammon, è considerato tuttora uno dei tre giocatori più forti del mondo.

Negli anni '90 nascono anche programmi in grado di giocare a Othello (o Reversi). Le principali applicazioni di intelligenza artificiale per questo gioco si basano su Minimax con potatura alpha-beta. Nel 1997 il programma Logistello, creato da Micheal Buro batte 6-0 il campione del mondo Takaeshi Murakami. Attualmente il gioco dell'Othello è risolto nelle versioni con scacchiera 4 x 4 e 6 x 6. Nella versione 8 x 8 (quella standard), sebbene non sia stato dimostrato matematicamente, l'analisi computazionale mostra un probabile pareggio, invece per le scacchiere 10 x 10 o più grandi non esiste nessuna stima, se non una forte probabilità di vittoria per il primo giocatore (il nero) [32].

Con la diffusione dei personal computer la ricerca nel campo dell'intelli-

genza artificiale si è estesa anche a giochi da tavolo più moderni. A partire dal 1983 Brian Sheppard [20] inizia a lavorare su un programma in grado di giocare al gioco da tavolo Scarabeo. Il suo programma, chiamato Maven è tuttora considerato la migliore intelligenza artificiale per lo Scarabeo [27] e gareggia a livello dei Campionati Mondiali. Maven combina un generatore selettivo di mosse, la simulazione di scenari di gioco verosimili e l'algoritmo di ricerca B*.

Gli algoritmi di intelligenza artificiale sono stati applicati a moltissimi altri giochi. Grazie a questi algoritmi è stato possibile risolvere giochi come il Tris, Forza 4, Pentamino, Gomoku, Nim, ecc. [34].

2.2 Intelligenza Artificiale e Giochi di Carte

I giochi di carte costituiscono una sfida per le intelligenze artificiali, in quanto sono l'esempio classico di giochi a informazione imperfetta. Oltre all'informazione imperfetta, la difficoltà nel costruire un'intelligenza artificiale è l'elemento multi-giocatore. L'incertezza sulla mano degli avversari rende il fattore di ramificazione dei metodi tradizionali di ricerca ad albero molto elevato, di conseguenza i metodi tradizionali si rivelano spesso inadatti.

Un gioco che ha fornito molti spunti per la ricerca in questo campo è il Bridge [30]. Questo gioco richiede quattro partecipanti, divisi in due coppie, e la sinergia nelle squadre è fondamentale. Agli inizi degli anni '80 Throop et al. [2] sviluppano la prima versione di BridgeBaron, un'intelligenza artificiale per il gioco del Bridge che, al contrario dei giochi a informazione perfetta, non utilizza il Minimax, ma una tecnica di pianificazione chiamata *hierarchical task network* (HTN) [21]. La tecnica HTN è una metodologia di pianificazione che si basa sulla scomposizione delle mansioni, dette *task*. Il sistema di pianificazione scompone ricorsivamente i task, creando sotto-task sempre più piccoli, fino a raggiungere task primitivi che riesce a eseguire direttamente. Nonostante le ottime performance nel Baron Barclay World

Bridge Computer Challenge, BridgeBaron non è in grado di competere con giocatori umani esperti [21]. Nel 1999 Ginsberg et al. [18] sviluppano un programma chiamato GIB (Ginsberg's Intelligent Bridgeplayer), con il quale, nel 2000, vincono il campionato mondiale di Bridge per computer. L'algoritmo di Ginsberg, invece di scegliere la mossa per ogni possibile configurazione di carte in gioco, basa la propria valutazione su un campione casuale di 100 ordinamenti diversi (campionamento Monte Carlo). GIB utilizza anche una tecnica chiamata *generalizzazione basata sulla spiegazione* che calcola e memorizza regole di giocata ottima in diverse situazioni standard della partita [18]. La ricerca sull'intelligenza artificiale nel gioco del Bridge è tuttora aperta perché nonostante gli ottimi livelli raggiunti da programmi come GIB, non si è ancora riuscito a raggiungere il livello esperto [2].

Un altro gioco ampiamente studiato in questo ambito è il Poker. Oltre all'informazione imperfetta e al multi-giocatore non cooperativo, l'elemento centrale in questo gioco è il fattore psicologico del bluff. Nei primi anni 2000 Darse Billings et al. [3] creano il programma Poki, in grado di giocare con ragionevole accuratezza a Poker, nella sua variante Texas Hold'em. La strategia di Poki è suddivisa in tre passi: calcolare la forza effettiva della mano, utilizzare questa informazione unita a una modellazione del comportamento avversario per creare una distribuzione di probabilità per le tre azioni possibili (lasciare, vedere, rilanciare), generare un numero casuale per scegliere l'azione. La modellazione del comportamento di ogni avversario avviene tramite una rete neurale di tipo feed-forward, addestrata con dati collezionati in partite online con giocatori umani, migliorandone l'accuratezza tramite una cross-validazione con dati raccolti in partite precedenti di ogni giocatore.

Nel 2009 Wan Jing Loh sviluppa un'intelligenza artificiale per il gioco Mahjong. Questo programma valuta la forza della mano, considerandolo come un problema di soddisfacimento di vincoli, e indica le possibilità di vittoria per ogni scelta sulla mano da comporre. Il metodo utilizzato costruisce

in un istogramma sul quale si basa la decisione [15].

2.3 Ricerca ad Albero Montecarlo

Gli algoritmi tradizionali di intelligenza artificiale sono molto potenti, tuttavia richiedono molta memoria nella loro esecuzione. Nel 2006 Kocsis e Szepesvari, con l'intento di migliorare gli algoritmi classici, introducono la *Ricerca ad Albero Monte Carlo* (o *Monte Carlo Tree Search*, *MCTS*). Rispetto al Minimax la differenza principale del metodo MCTS è che l'espansione l'albero degli stati non avviene completamente. Le *Figure 2.1* e *2.2* mostrano un esempio del gioco del Tris risolto utilizzando rispettivamente il Minimax e la Ricerca ad Albero Monte Carlo. Mentre il Minimax espande tutto l'albero e poi sceglie la mossa migliore, la Ricerca ad Albero Monte Carlo effettua una ricerca in profondità, fino a raggiungere uno stato finale. A questo punto valuta lo stato corrente, assegna un valore a quel nodo e lo propaga a tutti quelli percorsi fino alla radice. Questo procedimento viene ripetuto più volte costruendo un albero in maniera incrementale e asimmetrica, scegliendo a ogni iterazione il nodo più urgente da espandere. Tale scelta è fatta in base a una politica mirata a bilanciare l'esplorazione delle parti non ancora visitate e lo sfruttamento dei nodi che sembrano promettenti. La costruzione dell'albero si arresta arbitrariamente a seconda del tempo o della memoria a disposizione. Un ulteriore vantaggio è di non dovere calcolare i valori degli stati intermedi. Considerare unicamente la situazione negli stati finali richiede una conoscenza minore del dominio. Kocsis e Szepesvari hanno dimostrato che la Ricerca ad Albero Monte Carlo, avendo a disposizione abbastanza tempo e memoria, converge al risultato ottimo del Minimax [5].

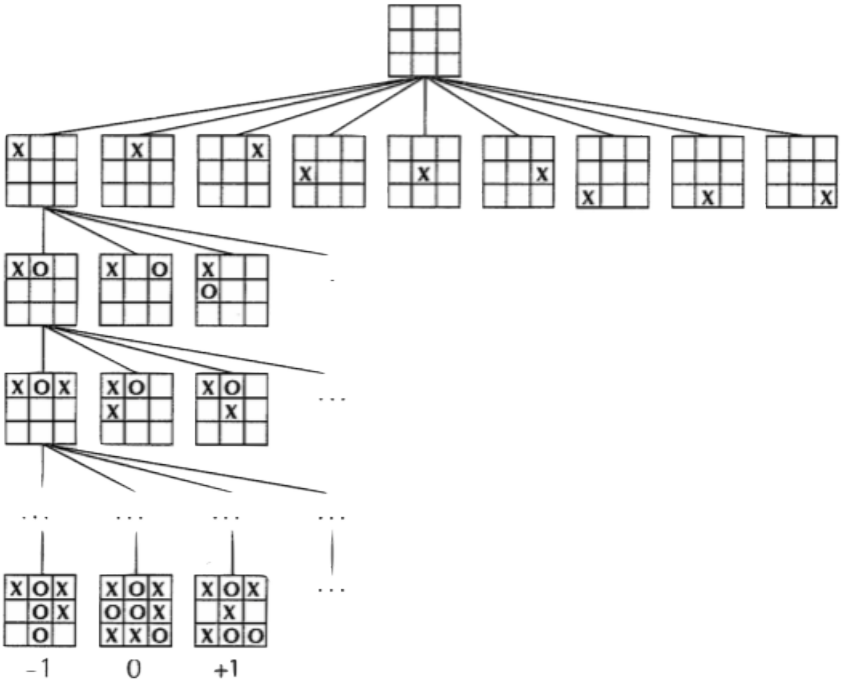

Figura 2.1: L'albero di ricerca Minimax del gioco del Tris espanso parzialmente.

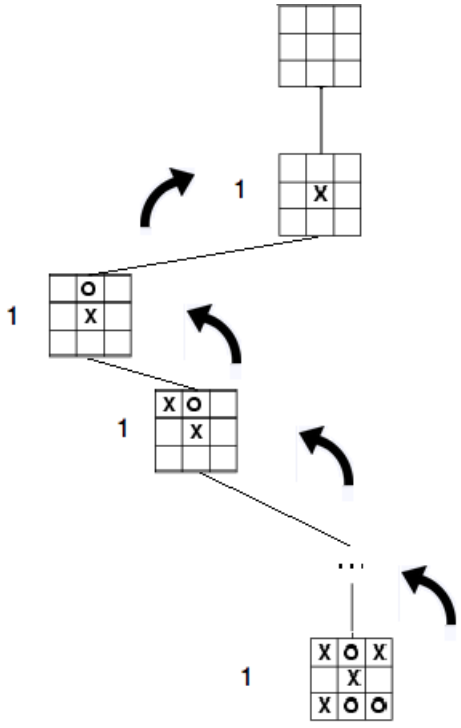

Figura 2.2: Una iterazione della Ricerca ad Albero Monte Carlo nel gioco del Tris, in maniera casuale viene scelto una mossa. Giunti in uno stato finale, il risultato viene propagato all'indietro nell'albero.

2.3.1 Stato dell'Arte

La Ricerca ad Albero Monte Carlo attira l'interesse dei ricercatori grazie ai risultati che ottiene col Go [11]. Il Go è un gioco con un altissimo fattore di ramificazione, a cui non è possibile applicare le tecniche tradizionali. Inoltre non esistono euristiche affidabili per situazioni di gioco non-terminali [5]. Il metodo MCTS, applicando una ricerca in profondità, raggiunge risultati che gli algoritmi classici non hanno mai ottenuto.

L'Hex è un gioco da tavolo inventato negli anni '40, viene giocato su scacchiera romboidale con caselle esagonali di dimensioni comprese tra gli 11 x 11 e i 19 x 19. A differenza del Go, l'Hex ha una robusta funzione di valutazione per gli stati intermedi, per questo motivo tecniche con potatura alfa-beta riescono a creare valide intelligenze artificiali [1]. A partire dal 2007 Arneson et al. [1] sviluppano un programma basato sulla Ricerca ad Albero Monte Carlo, in grado di giocare al gioco da tavolo Hex. Il loro programma, chiamato MoHex, vince l'argento nel 2008 e l'oro nel 2009 alle Computer Olympiads. Arneson et al. dimostrano che il loro programma basato sulla Ricerca ad Albero Monte Carlo è in grado di competere alla pari con le intelligenze artificiali basate sulla potatura alfa-beta.

Il metodo MCTS è particolarmente adatto a giochi con *informazione perfetta*, è invece più limitato in giochi con *informazione imperfetta*, dove, per raggiungere buoni risultati, necessita l'integrazione con altre tecniche. Una versione del metodo di Ricerca ad Albero Monte Carlo in questa tipologia di giochi viene implementata nel 2010 da M. Ponsen et al. [17], che lo applicano al Poker Texas Hold'em. L'algoritmo MCTS viene integrato con un classificatore bayesiano, utilizzato per modellare il comportamento degli avversari. Il classificatore bayesiano è in grado di predire sia le carte, sia le azioni degli altri giocatori. Il programma di Ponsen si rivela più forte di intelligenze artificiali basate su regole, ma di livello inferiore rispetto al programma Poki.

Nel 2011 Nijssen e Winands [16] utilizzano il metodo della Ricerca ad Albero Monte Carlo nell'intelligenza artificiale di un gioco da tavolo, Scotland Yard. In questo gioco i partecipanti devono raggiungere con la propria pedina quella di un giocatore che si nasconde su una mappa fatta a grafo. Il giocatore che scappa mostra la sua posizione ad intervalli prefissati, l'unica informazione che hanno a disposizione gli altri giocatori è il tipo di casella (detta *stazione*) in cui si può trovare il giocatore che si nasconde. In questo caso, al metodo MCTS viene integrata la *categorizzazione della locazione*, una tecnica che fornisce una predizione migliore sulla posizione del giocatore che si nasconde. Nijssen e Winands mostrano che il loro programma gioca su livelli più alti dell'intelligenza artificiale del gioco Scotland Yard per Nintendo DS, considerato essere uno dei giocatori più forti esistenti.

P. Cowling et al. [6] nel 2012, utilizzano l'algoritmo di Ricerca ad Albero Monte Carlo in una variante semplificata del gioco Magic: The Gathering. Come la maggior parte dei giochi da carte Magic ha una forte componente di indeterminazione, dovuta al vasto assortimento di carte presente nel mazzo. Nel loro programma, l'algoritmo MCTS è integrato con metodi di determinizzazione. Con questa tecnica, durante la costruzione dell'albero, le informazioni nascoste o imperfette sono considerate conosciute da tutti i giocatori.

Lo stesso P. Cowling sviluppa all'inizio del 2013 un'intelligenza artificiale per Spades [22], un gioco di carte a quattro giocatori. Cowling et al., utilizzando ancora strategie di determinizzazione, dimostrando ottime prestazioni in termini di tempo di calcolo. Il programma è scritto per essere montato su un telefono Android e per trovare una soluzione ottima con sole 2500 iterazioni in un quarto di secondo.

2.3.2 L'Algoritmo

L'algoritmo base del metodo MCTS consiste nel costruire iterativamente un albero di ricerca fino al raggiungimento della quantità prestabilita di tempo o memoria. Ogni nodo rappresenta uno stato nel dominio e ogni suo figlio è un possibile stato successivo. Il processo può essere suddiviso in quattro macro-fasi [12], come mostrato in *Figura 2.3*:

- *Selezione*: Partendo dal nodo-radice R, ricorsivamente si sceglie un nodo-figlio ottimo fino al raggiungimento di un nodo-foglia L.
- *Espansione*: Se L non è un nodo terminale (cioè il gioco non si conclude), si generano uno o più nodi figli e se ne sceglie uno C.
- *Simulazione*: Si simula una esecuzione sul nuovo nodo C per produrre un risultato.
- *Retro-propagazione*: Il risultato della simulazione è propagato all'indietro (fino alla radice) attraverso i nodi selezionati, aggiornandone le statistiche.

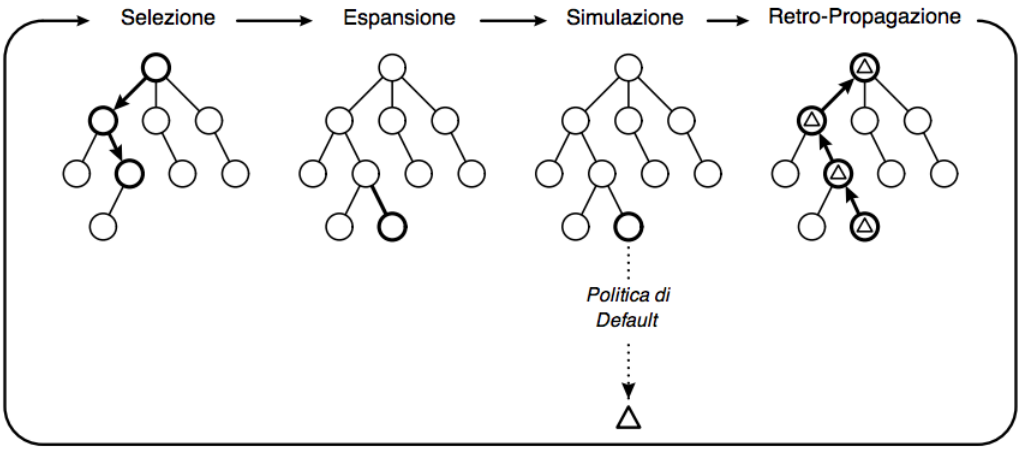

Figura 2.3: Una iterazione dell'algoritmo MCTS.

Per la selezione e l'espansione viene utilizzata la *Tree Policy* (cioè la politica dell'albero) che decide quale nodo prendere in considerazione. Invece per la simulazione viene usata la *Politica di Default* che mette in atto la conoscenza del dominio e produce una stima del valore dello stato corrispondente alla foglia finale. Queste fasi del metodo MCTS possono essere riassunte nell'*Algoritmo 1.*

1: **function** MCTS(n_0)
2: crea una radice n_0 con stato s_0
3: **while** entro i limiti di tempo/memoria **do**
4: $n_l \leftarrow$ TreePolicy(n_0)
5: $\Delta \leftarrow$ PoliticaDefault($s(n_l)$)
6: Propaga(n_l, Δ)
7: **end while**
8: **return** a(MigliorFiglio(n_0))
9: **end function**

Algoritmo 1: Ricerca ad Albero Monte Carlo

In questo algoritmo n_0 è il nodo radice a cui è associato lo stato s_0, mentre il nodo n_l è l'ultimo nodo raggiunto applicando la Politica Albero con corrispondente stato s_l. Il valore Δ è la ricompensa ottenuta eseguendo la Politica di Default sullo stato s_l. Infine $a(MigliorFiglio(n_0))$, il risultato finale della ricerca, è l'azione a che conduce al miglior figlio del nodo radice n_0.

2.3.3 UCT

Nella fase di selezione (*Figura 2.3*) viene applicato l'UCT (Upper Confidence Bound for Trees), per scegliere il nodo successivo da espandere. In

particolare, viene selezionato il nodo j che massimizza:

$$UCT = \overline{X_j} + C\sqrt{\frac{\log N}{n_j}} \tag{2.1}$$

dove N è il numero di volte in cui il nodo attuale (ossia il padre) è stato scelto, n_j è il numero di volte in cui il figlio j è stato scelto, C è un parametro positivo da settare e $\overline{X_j}$ è la media del valore totalizzato dal nodo j. I pareggi sono rotti con numeri casuali.

Questo algoritmo bilancia l'utilizzo delle ricompense raccolte (lo sfruttamento) con l'esplorazione dei nodi relativamente poco visitati. Dato che nel campionamento iniziale pesa molto la componente casuale, i nodi devo essere visitati un certo numero di volte prima che questi valori siano efficacemente stimati. L'*Algoritmo 2* mostra lo pseudo-codice dell'UCT implementato nella Ricerca ad Albero Monte Carlo. Ogni nodo n contiene quattro informazioni: lo stato associato $s(n)$, l'azione entrante $a(n)$, la ricompensa totale della simulazione $Q(n)$, e un intero non negativo che indica il numero di visite $N(n)$. Il termine $\Delta(n, p)$ rappresenta la funzione di ricompensa per il giocatore corrente p nel nodo n. Il risultato finale della ricerca è l'azione a che porta al figlio con la ricompensa maggiore.

```
1: function MCTS(n_0)
2:     crea una radice n_0 con stato s_0
3:     while entro i limiti di tempo/memoria do
4:         n_l ← TreePolicy(n_0)
5:         Δ ← PoliticaDefault(s(n_l))
6:         Propaga(n_l, Δ)
7:     end while
8:     return a(MigliorFiglio(n_0))
9: end function
10:
11: function TreePolicy(n)
12:     while n non è terminale do
13:         if n non è totalmente espanso then
14:             return Espandi(n)
15:         else
16:             n ← MigliorFiglio(n, C_p)
17:         end if
18:     end while
19:     return n
20: end function
21:
```

Algoritmo 2: Ricerca ad Albero Monte Carlo con UCT

22: **function** MigliorFiglio(n, c)

23: **return** $\arg\max_{n' \in \, figli \; di \; n} \frac{Q(n')}{N(n')} + c\sqrt{\frac{\log N(n)}{N(n')}}$

24: **end function**

25:

26: **function** PoliticaDefault(s)

27: **while** s non è terminale **do**

28: sceglie $a \in \mathrm{A}(s)$ in maniera casuale

29: $s \leftarrow f(s, a)$

30: **end while**

31: **return** ricompensa per lo stato s

32: **end function**

33:

34: **function** Propaga(n, Δ)

35: **while** n non è nullo **do**

36: $N(n) \leftarrow N(n) + 1$

37: $Q(n) \leftarrow Q(n) + \Delta$

38: $n \leftarrow$ padre di n

39: **end while**

40: **end function**

2.3.4 Punti di Forza

Il metodo MCTS offre molti vantaggi rispetto ai tradizionali algoritmi con alberi di ricerca [12]. Innanzitutto non richiede nessuna conoscenza strategica o tattica del dominio, può funzionare benissimo conoscendo solamente le mosse legali e le condizioni di fine della partita. Questo costituisce un vantaggio per molti giochi (come abbiamo visto nel caso del Go), dato che non è necessaria una funzione di valutazione per le situazioni non terminali della partita. Inoltre questo metodo ha il grosso vantaggio di riuscire a gestire bene alberi con un alto fattore di ramificazione. La ricerca si concentra

soprattutto su i nodi più promettenti, tralasciando l'esplorazione di quelli che conducono a ricompense inferiori. Tuttavia la caratteristica principale che lo contraddistingue è la possibilità di fermarlo in qualsiasi momento, ottenendo come risultato la stima calcolata fino a quell'istante. Naturalmente più è il tempo lasciato all'esplorazione, più sarà preciso il risultato. Questa proprietà lo rende perfetto in casi di scarsità di risorse, in termini di memoria e tempo.

2.3.5 Punti Deboli

In alcune applicazioni (per esempio in domini discreti) la Ricerca ad Albero Monte Carlo risulta inutilizzabile, ciò è dovuto principalmente a una esagerata vastità del dominio, la conseguenza è che i nodi chiave non vengano visitati un numero sufficiente di volte. In altri casi per convergere a una soluzione ottima l'algoritmo necessita di troppo tempo, questo può essere un problema per molte applicazioni che richiedono un tempo di risposta piuttosto breve. Tuttavia è possibile ottimizzare le performance dell'algoritmo, implementando potature sull'albero o integrandolo con altri tipi di tecniche per ridurre lo spazio/tempo di ricerca [12].

2.4 Sommario

In questo capitolo abbiamo trattato l'intelligenza artificiale nei giochi da tavolo e di carte. Nella Sezione 2.1 abbiamo presentato una breve panoramica riguardante i primi algoritmi utilizzati per simulare un avversario intelligente nei giochi da tavolo. Successivamente nella Sezione 2.2 abbiamo preso in analisi dei programmi in grado di giocare a giochi di carte, soffermandoci sui problemi e le soluzioni trovate. Nell'ultima sezione, la 2.3, abbiamo spostato la nostra attenzione sull'algoritmo utilizzato nel nostro lavoro, la Ricerca ad Albero Monte Carlo, confrontandolo con l'algoritmo Minimax. Dopo averne

presentato lo stato dell'arte (2.3.1), nella Sezione 2.3.2 abbiamo descritto l'algoritmo nel dettaglio, per poi passare, nella Sezione 2.3.3, all'UCT, ossia l'euristica utilizzata per la selezione del nodo. Infine abbiamo preso in esame i vantaggi e gli svantaggi del metodo MCTS (Sezioni 2.3.4 e 2.3.5).

Capitolo 3

La Briscola a 5

In questo capitolo introduciamo le regole del gioco della Briscola a 5. Partendo dal regolamento, dalle carte e dai punteggi della Briscola classica, spieghiamo le variazioni e la struttura della partita della Briscola a 5.

3.1 La Briscola Classica

La Briscola è il gioco di carte italiano più diffuso, insieme alla *Scopa* e al *Tressette* [9]. La sua origine viene fatta risalire all'Olanda del XVI secolo, dalla quale diviene famosa prima in Francia, dove nascono diverse famiglie di giochi (*Brisque*, *Bazzica*, *Brusquembille*, ecc.), poi in Spagna (*Briscas*) e infine in Italia, dove si diffonde nelle versioni attuali [8]. È molto giocata anche in Slovenia e Croazia, sotto il nome di *Briškula* [14]. Comunemente viene giocata in due o in quattro giocatori, ma esistono le varianti con tre giocatori o, nel caso della Briscola a Chiamata, in 5. Ci sono inoltre altre varianti come la Briscola Scoperta, la Briscola a 31, il Briscolone, ecc. [31].

Nella Briscola lo scopo di ogni giocatore è di aggiudicarsi durante la partita il maggior numero di punti. Essendo il totale dei punti pari a 120, per vincere la partita il giocatore deve totalizzare almeno 61 punti, mentre

con 60 punti a testa si termina con un pareggio. In genere si gioca a vincere 2 partite su 3 oppure 3 su 5 [8] [14] [7].

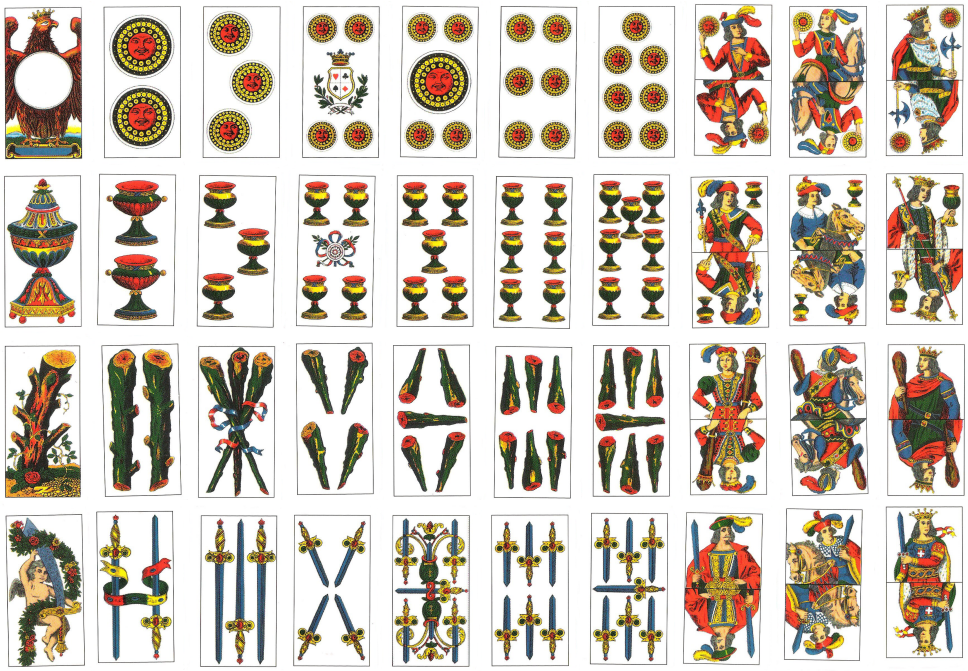

Figura 3.1: Il mazzo di carte piacentine utilizzate per la Briscola.

3.1.1 Le Carte

Per la Briscola si utilizza il classico mazzo italiano da 40 (*Figura 3.1*), oppure quello internazionale da 52 carte da cui vengono rimossi 8, 9, 10 e Jolly (*Figura 3.2*). L'ordine di valore delle carte prevede, dalla più alta alla più bassa: Asso, Tre, Re, Regina (nelle regionali Cavallo), Fante, 7, 6, 5, 4 e 2.

Come viene riassunto nulla *Tabella 3.1*, a ogni carta è associato un valore. Asso e Tre sono detti *Carichi* [9] e valgono rispettivamente 11 e 10 punti; Re (4 punti), Regina (3 punti) e Fante (2 punti) sono detti *Carte Vestite* [9], mentre le carte rimanenti, dette *Scartine* [9], non hanno valore. Sommando i punteggi delle singole carte, il totale del mazzo è di 120 punti.

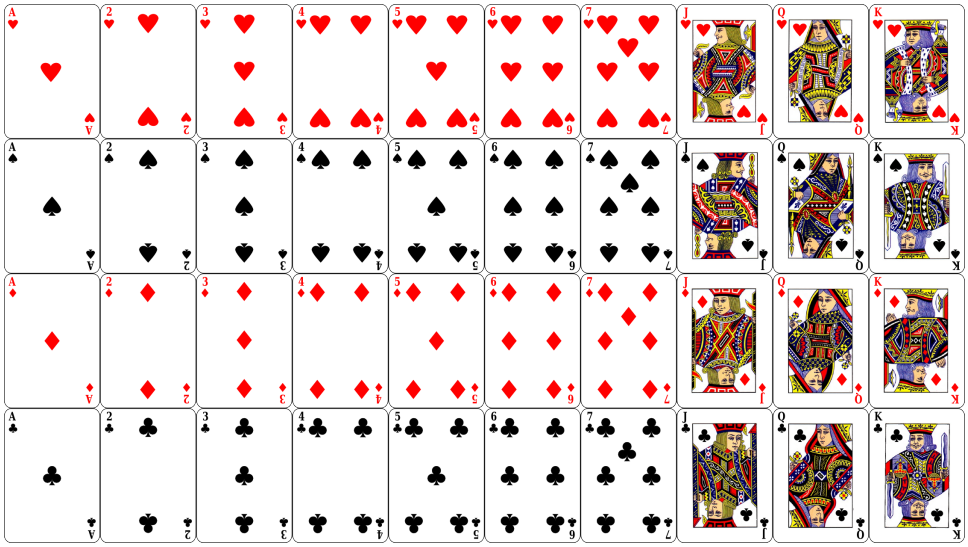

Figura 3.2: Il mazzo di carte internazionali utilizzate per la Briscola.

Carta	A	3	K	Q	J	7	6	5	4	2
Punti	11	10	4	3	2	0	0	0	0	0

Tabella 3.1: Valori delle carte

3.1.2 Svolgimento della Partita a 2 Giocatori

Uno dei due giocatori, scelto a sorte come mazziere, mischia le carte e ne distribuisce tre a testa, dopodiché prende una carta e la mette scoperta sul tavolo, sotto la pila di carte coperte (come mostrato in *Figura 3.3*), detto *tallone* [7]. Il seme della carta scoperta è detto seme di Briscola. A questo punto la partita ha inizio, il primo di turno è del giocatore che non ha distribuito le carte. Questo sceglie una carta tra quelle che ha in mano, disponendola a faccia in su sul tavolo. L'avversario quindi gioca a sua volta una carta. Si aggiudica la presa chi ha giocato la Briscola più alta oppure la carta più alta del primo seme giocato in quella mano (esempi di giocata nelle *Figure 3.4, 3.5, 3.6*). Prendere con una carta di valore più alto e dello stesso

Figura 3.3: La carta posta al di sotto del mazzo.

seme della prima giocata è detto *strozzare*. Il giocatore che si è aggiudicato la presa pesca la prima carta in cima al tallone, seguito dal perdente. Il vincente della mano è anche il primo a giocare in quella successiva. Si prosegue così fino a esaurimento delle carte; l'ultimo a pescare dal tallone prende la carta di Briscola a faccia in su e il gioco continua senza pescare. Quando tutte le carte sono state giocate, ogni giocatore somma i punti delle carte prese (in accordo coi valori della *Tabella 3.1*) e si aggiudica la partita chi totalizza il punteggio maggiore.

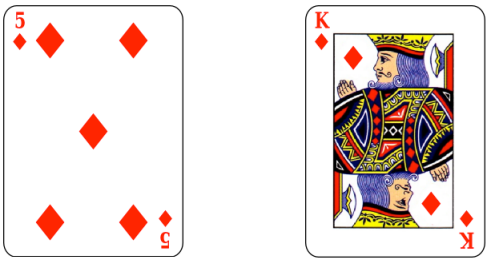

Figura 3.4: Esempio 1. La prima carta giocata è il 5 di Quadri. Il secondo giocatore si aggiudica la mano strozzando *con una carta dello stesso seme, ma con valore maggiore.*

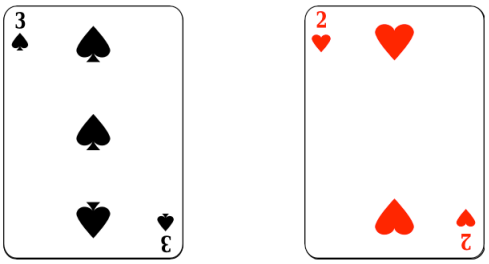

Figura 3.5: Esempio 2. La prima carta giocata è il Tre di Picche, il seme di Briscola è Cuori. Il secondo giocatore vince la mano giocando un Due di Cuori, perchè, anche se di valore inferiore, la Briscola vince sempre contro una carta di un altro seme.

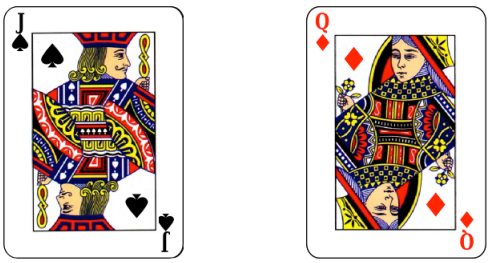

Figura 3.6: Esempio 3. La prima carta giocata è un Fante di Picche, il seme di Briscola è cuori. Il secondo giocatore gioca una Regina di Quadri, il primo giocatore si aggiudica la mano. Nonostante la Regina abbia valore maggiore, il seme dominante in questa mano è Picche.

3.1.3 Svolgimento della Partita a 4 Giocatori

Nelle partite a 4, i giocatori si dividono in due squadre e ognuno si siede di fronte al proprio compagno, come mostrato in *Figura 3.7*. Le regole rimangono le stesse della Briscola a 2. Il primo a iniziare è il giocatore alla destra di chi ha distribuito le carte, a ogni turno il primo a giocare è chi prende nella mano precedente. Nella Briscola è permesso parlare (a eccezione della prima mano) e segnalare al compagno con gesti convenzionali le carte possedute o quella da giocare. I gesti più comuni utilizzati sono: Asso, si stringono le labbra; Tre, si storce la bocca; Re, si alzano gli occhi al cielo; Donna, mostrando la punta della lingua; Fante, alzando una spalla. Una volta che sono state pescate le ultime 4 carte, i compagni si mostrano le carte passandosele. Come nella briscola a 2 vince la coppia che ha fatto più punti [9].

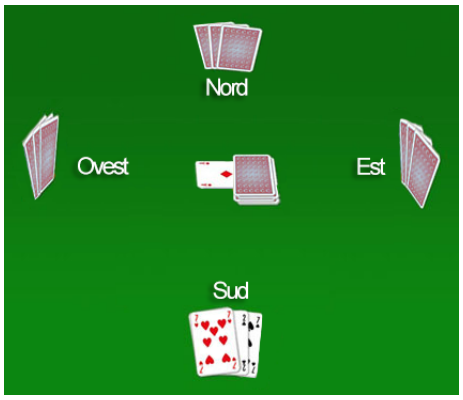

Figura 3.7: La disposizione dei giocatori intorno al tavolo. Le coppie sono Nord-Sud contro Est-Ovest.

3.2 La Briscola a 5

La Briscola a 5 (detta anche Briscola a Chiamata o Briscola a Chiamare [9] [8] [7]) è una variante della Briscola classica che viene giocata in cinque

giocatori. Condivide con la Briscola classica le regole di presa e i valori delle carte [9].

3.2.1 Struttura della Partita

A differenza della Briscola classica si comincia distribuendo tutte le carte del mazzo, per un totale di otto a testa. Le due squadre sono asimmetriche, una composta da tre giocatori e l'altra da due, in alcuni casi si gioca quattro contro uno. La partita a Briscola a 5 si divide in due fasi: l'asta (detta anche Licitazione [9]) e il gioco vero e proprio.

3.2.2 L'Asta

Il primo di turno (il giocatore alla destra del mazziere) decide se *chiamare* una carta, cioè dichiarare il valore (ma non il seme) della carta che gli manca, oppure di *passare*, cioè di abbandonare l'asta. Procedendo in senso antiorario tutti i giocatori devono decidere se rilanciare o passare. Nel primo caso devono dichiarare una carta di valore inferiore a quella chiamata in precedenza. L'asta si arresta dopo un giro completo di *passo* oppure quando viene chiamato il 2. A questo punto il vincitore (il *chiamante*) deve dichiarare il seme della carta chiamata; il seme di questa carta costituisce la Briscola e chi la possiede (il *chiamato*) diventa il compagno segreto del chiamante. Terminata l'asta le due squadre sono formate da chiamante e chiamato contro gli altri tre giocatori. In una variante dell'asta, se un giocatore chiama il 2, un altro giocatore può rilanciare chiamando a sua volta il 2 e dichiarando il punteggio minimo che si impegna a raggiungere a fine partita (maggiore di 61). Vince questa nuova asta chi dichiara il punteggio maggiore [9].

3.2.3 La Chiamata in Mano

Durante l'Asta, a un giocatore è consentito *chiamarsi in mano*, ossia chiamare una carta che lui stesso possiede. In caso di vittoria dell'asta, deve affrontare la partita da solo contro gli altri quattro, ovviamente a loro insaputa [9] [8] [7]. In alcune regioni esiste una regola che impone al giocatore che chiama una propria carta di dichiararlo alla fine dell'asta [8] [7].

3.2.4 La Fase di Gioco

La fase di gioco conserva le regole della Briscola classica. Ogni partita è costituita da 8 mani, nella prima mano il primo di turno è il giocatore alla destra del mazziere, mentre nelle altre il primo a giocare è chi si aggiudica la mano precedente. L'obiettivo della partita è di far accumulare alla propria squadra 61 punti (o di più, nel caso di chiamata al 2 con punteggio). La particolarità del gioco è che solo il chiamato conosce la composizione delle squadre e si scoprirà solamente giocando la carta chiamata. A differenza della Briscola classica è proibito qualsiasi tipo di segnale [8]. Finché il chiamato non si svela, gli altri giocatori possono solo intuire le composizioni delle squadre in base ai comportamenti di gioco [9].

3.2.5 Punteggi e Vittoria Finale

Alla fine di ogni partita, i giocatori vincenti ricevono un punto, mentre quelli perdenti ricevono un malus, sempre di un punto, tranne il chiamante che riceve un bonus/malus doppio. Nel caso di chiamata in mano il punteggio del chiamante è quadruplicato anziché raddoppiato [9] [7]. Secondo una variante, in caso di vittoria con *cappotto* (cioè 120 a 0) tutti i punteggi sono raddoppiati [9]. Vince il giocatore che raggiunge un punteggio precedentemente deciso (di solito 11 [9]), oppure chi ha guadagnato più punti dopo un numero concordato di mani.

Partita	Punti Chiamante	Punti Chiamato	Punti Avversari
Normale	+2/-2	+1/-1	+1/-1
Chiamata in Mano	+4/-4	+1/-1	+1/-1

Tabella 3.2: Punteggi dopo ogni partita

3.3 Sommario

In questo capitolo abbiamo descritto innanzitutto le regole della Briscola classica, partendo delle carte utilizzate (3.1.1) e dei loro valori, passando poi al regolamento della partita a 2 giocatori (3.1.2) e a 4 giocatori (3.1.3). Dopodiché siamo passati alle regole della Briscola a 5, descrivendo prima la fase di asta (3.2.2) e poi la fase di gioco (3.2.4). Nella Sezione 3.2.5 abbiamo spiegato i punteggi di ogni partita e le modalità di vittoria finale.

Capitolo 4

L'Intelligenza Artificiale a Regole

In questo capitolo descriviamo le versioni di intelligenza artificiale a regole utilizzate nel nostro gioco della Briscola a 5. Le regole codificano la conoscenza di dominio fornita da alcuni esperti e libri di strategie dei giochi di carte. Lo scopo è quello di sviluppare degli algoritmi di intelligenza artificiale con cui effettuare un confronto con quelli basati sul metodo MCTS e verificarne la validità. In particolare in questo capitolo illustriamo gli algoritmi di intelligenza artificiale che abbiamo creato per l'asta e per la fase di gioco, analizzando i tre differenti ruoli, ovvero chiamante, avversario e chiamato.

4.1 La Scelta del Modello

I giochi di carte sono caratterizzati dalla presenza di informazione imperfetta, in quanto ogni giocatore non è a conoscenza delle carte in mano agli altri giocatori. La Briscola a 5, oltre all'informazione imperfetta, ha delle caratteristiche che rendono una sfida lo sviluppo di algoritmi di intelligenza artificiale competitivi. La fase d'asta iniziale, che determina le squadre, ha

una struttura e un insieme di regole completamente diversa dal resto della partita e richiede strategie specifiche. Secondariamente la composizione delle squadre, che rimane sconosciuta durante gran parte della partita, costituisce un'ulteriore sorgente di informazione imperfetta, in aggiunta alle carte nascoste. Questa informazione da imperfetta diventa perfetta solo quando il chiamato si scopre (giocando la carta dichiarata), da quel momento in poi le squadre sono note e le strategie di ogni giocatore sono differenti. Per questi fattori abbiamo deciso di creare per il nostro gioco due algoritmi separati di intelligenza artificiale a regole (una per l'asta e una per la fase di gioco), che gestiscono le due fasi della partita in modo da avvicinarsi il più possibile alla conoscenza umana del dominio.

4.2 Gestione dell'Asta

Nella Briscola a 5 l'asta è il momento in cui si decidono i ruoli dei giocatori. Per ogni giocatore è fondamentale valutare se si hanno le carte adatte per essere il chiamante, allo stesso tempo è sconveniente lasciare a un avversario una chiamata a una carta alta, aumentandone le possibilità di vittoria. Abbiamo deciso di semplificare l'algoritmo che gestisce l'asta eliminando la regola della chiamata in mano, che permette al giocatore di chiamare una carta che possiede. Per decidere quale chiamata effettuare, l'intelligenza artificiale dei nostri giocatori valuta due fattori principali: la distribuzione dei semi e i punti totali in mano. Se consideriamo tutte le combinazioni possibili dei 4 semi nelle 8 carte in mano a un giocatore, possiamo contare 15 distribuzioni differenti, 8-0-0-0 nel caso in cui tutte le carte in mano siano dello stesso seme, 7-1-0-0 nel caso in cui le carte siano tutte dello stesso seme tranne una di seme differente, e così via per tutte le altre distribuzioni, ovvero 6-2-0-0, 6-1-1-0, 5-3-0-0, 5-2-1-0, 5-1-1-1, 4-4-0-0, 4-3-1-0, 4-2-2-0, 4-2-1-1, 3-3-2-0, 3-3-1-1, 3-2-2-1, 2-2-2-2. Più la distribuzione dei semi è squilibrata, più è conveniente per un giocatore chiamare, in quanto il nume-

ro elevato di briscole permetterà più possibilità di presa. Viceversa, la mano con distribuzione omogenea dei semi è quella meno adatta per affrontare la partita perché si possiedono un numero minore di briscole. Il secondo fattore valutato dall'algoritmo sono i punti in mano, che si ottengono sommando i valori delle carte possedute. Abbiamo suddiviso questi punti in nove fasce: 0-14, 15-20, 21-25, 26-30, 31-35, 36-40, 41-45, 46-50, più di 50. Un maggior punteggio in mano significa briscole alte e/o carichi da dare al compagno ed è dunque un fattore favorevole per la chiamata. Una volta valutati questi due fattori il nostro algoritmo determina per ogni distribuzione dei semi e fascia di punteggio quali carte bisogna possedere per effettuare una chiamata. Il criterio utilizzato dall'algoritmo di intelligenza artificiale è illustrato in *Tabella 4.1.* Tale tabella indica quali carte deve possedere il giocatore per chiamare (l'etichetta *Sempre* significa con qualsiasi carta in mano), e a quale valore fermarsi (il valore della carta indicata dopo l'etichetta *Lim:*). Il seme su cui ricade la scelta è sempre il seme *lungo*, cioè quello con almeno tre carte, mentre il valore della carta chiamata è quello della più alta mancante nel seme scelto. La prima fascia di punteggio non è stata inserita in tabella, in quanto è uso comune che con meno di 15 punti in mano si chiama solamente se si possiedono almeno cinque carte in un seme e se tutti i punti sono distribuiti in quel seme; il nostro algoritmo segue questo criterio.

Esempio 1: supponiamo che la mano del giocatore sia composta da A ♠, 3 ♠, 7 ♠, 4 ♠, 2 ♠, K ♢, J ♡, 5 ♡; la distribuzione è quindi 5-2-1-0, mentre il punteggio è 27 (11+10+0+0+0+4+2+0). Il seme lungo è Picche, dunque il giocatore valuta la chiamata in base alla *Tabella 4.1*: la casella corrispondente (evidenziata in azzurro) indica che la dichiarazione avviene se si possiede uno tra Asso, Tre, Re e il limite è il 6. Il giocatore durante l'asta chiama quindi la più alta delle carte mancanti di Picche (K, Q, J, 6), rilanciando fino al 6 compreso; se l'asta prosegue oltre, dichiara il passo.

In caso di due semi lunghi, a parità di carte il giocatore sceglie quello con

più punti, a parità di punti sceglie quello con più carte. Nel caso in cui non sia possibile dichiarare una carta del seme lungo, il giocatore chiama una carta del secondo seme lungo (se presente), altrimenti una di un seme con 2 carte, seguendo come regola quella della *Tabella 4.1*, incrociando la riga 2-2-2-2 e la colonna della distribuzione della propria mano.

Esempio 2: supponiamo che la mano sia composta da K ♢, Q ♢, J ♢, 7 ♣, 4 ♣, 2 ♣, A ♡, 3 ♡, e che l'ultimo rilancio sia stato al Re. La distribuzione delle carte è 3-3-2-0, mentre il punteggio è 30. In questo caso ci sono due semi lunghi, Quadri e Fiori. Il primo seme scelto per la chiamata è Quadri, in quanto la somma dei punti è maggiore. La casella corrispondente (evidenziata in verde) nella *Tabella 4.1* indica che la chiamata è da effettuare in presenza di uno tra Asso, Tre e Re e di limitarsi a chiamare fino al Fante. La prima condizione è soddisfatta, tuttavia, possedendo sia la Regina che il Fante, il limite impone di non dichiarare. A questo punto l'algoritmo prende in considerazione il secondo seme lungo. Questo seme non ha i requisiti per la chiamata perché non si possiede una delle tre carte richieste. Infine il giocatore valuta il seme composto da due carte, cioè Cuori. La casella corrispondente alla riga 2-2-2-2 (evidenziata in giallo) indica di chiamare in caso di possesso di Asso e Tre. Essendo il requisito soddisfatto, il giocatore chiama la prima carta mancante dopo il Re, cioè la Regina, rilanciando fino al 7 in caso di proseguimento dell'asta.

4.3 Gestione della Fase di Gioco

L'algoritmo di intelligenza artificiale adotta per la gestione della fase di gioco strategie differenti in base a tre fattori fondamentali, ovvero il ruolo, il posizionamento durante la mano e se il chiamato si è rivelato, giocando la carta chiamata. Come è noto dal Capitolo 3, durante una partita di Briscola a 5, un giocatore può ricoprire tre ruoli diversi: il chiamante, l'avversario e

Punti / Distrib.	15-20	21-25	26-30	31-35	36-40	41-45	46-50	> 50
8-0-0-0	Sempre; Lim:4	Sempre; Lim:2	Sempre; Lim:2	Imposs.	Imposs.	Imposs.	Imposs.	Imposs.
7-1-0-0	Sempre; Lim:6	Sempre; Lim:5	Sempre; Lim:4	Sempre; Lim:2	Sempre; Lim:2	Sempre; Lim:2	Imposs.	Imposs.
6-2-0-0 6-1-1-0	Sempre; Lim:7	Sempre; Lim:6	Sempre; Lim:5	Sempre; Lim:4	Sempre; Lim:2	Sempre; Lim:2	Sempre; Lim:2	Sempre; Lim:2
5-3-0-0 5-2-1-0 5-1-1-1	A o 3; Lim:J	A o 3; Lim:7	A o 3 o K; Lim:6	A o 3 o K; Lim:5	A o 3 o K; Lim:4	A o 3 o K; Lim:2	A o 3 o K; Lim:2	Sempre; Lim:2
4-4-0-0 4-3-1-0 4-2-2-0 4-2-1-1	A o 3; Lim:Q	A o 3; Lim:J	A o 3 o K; Lim:7	A o 3 o K; Lim:6	A o 3 o K; Lim:5	A o 3 o K; Lim:4	A o 3 o K; Lim:2	A o 3 o K; Lim:2
3-3-2-0 3-3-1-1 3-2-2-1	A o 3; Lim:K	A o 3; Lim:Q	A o 3 o K; Lim:J	A o 3 o K; Lim:7	A o 3 o K; Lim:6	A o 3 o K; Lim:5	A o 3 o K; Lim:4	A o 3 o K; Lim:2
2-2-2-2	Passo	A + 3; Lim:J	A + 3; Lim:7	A + 3; Lim:6	A + 3; Lim:5	Due tra: A,3,K; Lim:4	Due tra: A,3,K; Lim:4	Due tra: A,3,K; Lim:2

Tabella 4.1: Carte che il giocatore deve possedere per chiamare (l'etichetta Sempre *significa con qualsiasi carta in mano), e a quale valore fermarsi (il valore della carta indicata dopo l'etichetta* Lim:*). Le righe corrispondono rispettivamente alle possibili distribuzioni delle carte in mano e dei punti (l'etichetta* Imposs. *indica che quella combinazione non può avvenire). Evidenziate le celle corrispondenti alle chiamate negli Esempi 1 e 2 della Sezione 4.2.*

il chiamato. Ogni ruolo adotta una strategia differente. Inoltre l'algoritmo prende decisioni diverse a seconda della posizione in cui si trova nella mano, abbiamo distinto tre casi: primo di mano (il giocatore decide quale sarà la carta dominante e attende le mosse degli altri), posizione di mezzo (la posizione in cui è più difficile giocare in quanto il giocatore deve adattarsi alle mosse di chi lo ha preceduto e deve stare attento a quelle dei giocatori che lo seguono) e ultimo di mano (il giocatore ha la posizione più vantaggiosa perché conosce le mosse di tutti gli altri e può giocare in maniera più spregiudicata). Tutti i giocatori all'inizio della partita adottano la strategia detta *pre-rivelazione*, nel momento in cui il compagno del dichiarante si palesa, giocando la carta chiamata, l'intelligenza artificiale adotta la strategia chiamata *post-rivelazione*. La regola che vale praticamente per qualsiasi giocatore nella scelta della carta è quella che abbiamo chiamato *valorizzazione delle briscole*. La regola dice che se si possiede una briscola e il numero di briscole in gioco che la dominano è inferiore di uno al numero di turni mancanti, bisogna giocare quella briscola. Ad esempio, supponiamo che un giocatore abbia in mano un Re di briscola e che sia la terzultima mano della partita. Il giocatore è a conoscenza che Asso e Tre non sono ancora stati giocati. Questi deve giocare il Re dato che le ultime due mani saranno dominate da Tre e Asso.

4.3.1 Il Chiamante

Il chiamante, se ha adottato la strategia giusta nel momento della chiamata, in linea teorica dovrebbe avere la sua squadra con le briscole più forti. Il suo scopo è quello di costringere gli avversari a tenere in mano il più a lungo possibile le carte più alte, in modo da dominarle negli ultimi turni con le proprie briscole. L'algoritmo di intelligenza artificiale che interpreta il ruolo di chiamante ed è primo di mano, gioca la carta con il valore minore, sia che che adotti la strategia pre-rivelazione, che quella post-rivelazione. Se il

giocatore pre-rivelazione si trova nella posizione di mezzo nella mano, decide la carta da giocare in base ai punti sul tavolo e se la carta dominante è una briscola o meno. Gli stessi fattori sono valutati se il giocatore è nella fase post-rivelazione, con l'aggiunta del ruolo del giocatore che sta vincendo e della posizione del compagno. Nel caso sia ultimo di mano e in fase pre-rivelazione prende solamente se può *strozzare* (vedi Sezione 3.1.2) oppure se ci sono molti punti sul tavolo. Lo stesso vale dopo la rivelazione e sta vincendo un avversario. Se invece si sta aggiudicando la mano il compagno, il giocatore lo *carica* (ossia gioca la carta con il valore più alto, non di briscola).

4.3.2 L'Avversario

Lo scopo dell'avversario è identificare il chiamato e per questo deve tenere conto dei punti dati al chiamante da parte di tutti gli altri giocatori. Se i punti dati al dichiarante superano una soglia minima, il giocatore che ha dato questi punti viene classificato come potenziale chiamato. Quando l'avversario è primo di mano, gioca o una briscola bassa (minore del Fante) oppure la carta dal valore più basso, sia nella strategia pre-rivelazione che in quella post-rivelazione. Quando l'avversario che adotta la strategia pre-rivelazione non è primo di mano, si comporta come il chiamante, facendo le stesse valutazione nel caso valga la pena di prendere; se invece non ha la possibilità di prendere, concede qualche punto a chi sta prendendo se questo non è né il dichiarante, né il potenziale chiamato. L'avversario nella fase post-rivelazione ha gli stessi parametri di scelta del chiamante, con la differenza che deve valutare entrambe le posizioni dei compagni.

4.3.3 Il Chiamato

Il chiamato è l'unico giocatore che conosce le composizione delle squadre sin dalla prima mano. Il suo scopo è quello prendere e far guadagnare al com-

pagno più punti possibile, tentando allo stesso tempo di non farsi scoprire dagli avversari e di ritardare la propria rivelazione. Mantiene i due comportamenti diversi (pre/post-rivelazione) perché l'atteggiamento di gioco è differente tra prima e dopo che la carta chiamata viene giocata. Quando il chiamato che adotta la strategia pre-rivelazione è primo di turno gioca una briscola bassa (ovviamente non la carta chiamata) o una carta da pochi punti (al massimo una figura). Nelle altre posizioni di mano nella strategia pre-rivelazione, tenta sempre di prendere quando può strozzare o quando ci sono molti punti in palio, altrimenti gioca qualche punto al compagno. Se supera la soglia di punti con cui viene scoperto, inizia a caricare il chiamante con carte alte. Nel caso in cui stia vincendo un avversario, lascia il minor numero di punti possibile. Non gioca mai la carta chiamata nei primi tre turni, a meno che i punti in palio non siano elevati e non abbia altro modo di prendere. Nella fase post-rivelazione la sua strategia equivale a quella del chiamante post-rivelazione.

4.4 Sommario

In questo capitolo abbiamo descritto le versioni di intelligenza artificiale a regole presenti nel nostro gioco della Briscola a 5. L'approccio a regole ci ha permesso di codificare la conoscenza del dominio simile a quella che può avere un essere umano, in modo da avere dei giocatori di buon livello per mettere alla prova gli algoritmi basati sulla Ricerca ad Albero Monte Carlo. Il nostro algoritmo cerca di imitare le strategie utilizzate dagli esseri umani, per questo adotta strategie differenti a seconda del ruolo, della posizione durante la mano e della rivelazione delle squadre. Nella Sezione 4.1 abbiamo descritto le difficoltà che si possono incontrare nello sviluppo di algoritmi di intelligenza artificiale competitivi per la Briscola a 5. Nella Sezione 4.2 abbiamo presentato le diverse scelte effettuate dai giocatori durante l'asta. Valutando la distribuzioni dei semi e il numero di punti in mano, abbiamo

creato una tabella sulla quale il giocatore si basa per scegliere se chiamare e quando fermarsi. Nella stessa sezione sono presenti degli esempi di situazioni specifiche. Successivamente nella Sezione 4.3 abbiamo spiegato i vari comportamenti dell'intelligenza artificiale per la fase di gioco. Si distinguono i tre ruoli dei giocatori, ossia chiamante, avversario e chiamato. Per ognuno dei quali abbiamo elencato i comportamenti nelle varie situazioni di gioco, pre/post-rivelazione e primo/in mezzo/ultimo di mano.

Capitolo 5

Il Metodo MCTS utilizzato nella Briscola a 5

In questo capitolo illustriamo gli algoritmi di intelligenza artificiale basati sulla Ricerca ad Albero Monte Carlo che abbiamo sviluppato per il nostro gioco. In particolare abbiamo creato quattro varianti del metodo MCTS che utilizzano diverse euristiche nella fase di espansione dell'albero. Il primo algoritmo che abbiamo studiato si basa sull'applicazione del metodo MCTS standard in cui, nella fase di espansione, i nodi sono scelti in modo casuale. Successivamente abbiamo aggiunto euristiche sempre più complesse per la scelta del nodo da espandere.

5.1 MCTS Semplice

La prima versione della Ricerca ad Albero Monte Carlo che abbiamo studiato per la Briscola a 5 è stata la versione di base. Ogni nodo dell'albero descrive un'istanza dello stato corrente del gioco (con tutte le informazioni come il giocatore di turno, la carta dominante, il giocatore vincente, l'ultima carta giocata, ecc.). Oltre allo stato di gioco tutti i nodi memorizzano il numero di visite e il valore totale ottenuto in quelle visite. Nella fase di selezione

della costruzione dell'albero (si veda Sezione 2.3.2), il nodo selezionato viene determinato dall'algoritmo UCT (si veda *Algoritmo 3*). Mentre nella fase di espansione (2.3.2), il nodo non ancora espanso è scelto in maniera casuale. Una volta raggiunto un nodo foglia si termina la fase di espansione e si passa alla fase di retro-propagazione (2.3.2).

Per quanto riguarda il valore da propagare all'indietro abbiamo creato due funzioni di valutazione. La prima è la valutazione degli algoritmi di intelligenza artificiale classici come il Minimax, ossia restituisce il valore 1 se il giocatore vince, -1 se la vittoria va all'altra squadra, 0 se finisce in parità; la seconda funzione restituisce la differenza tra i punti della squadra del giocatore e 60. Alla fine dell'algoritmo l'azione scelta dall'intelligenza artificiale è quella rappresentata dall'azione corrispondente al nodo figlio della radice che ha il più alto rapporto valore su visite. Ad esempio, come si vede nella *Figura 5.1* (rappresentante l'albero costruito dopo due iterazioni dell'algoritmo), se l'esecuzione si arrestasse in quel momento, l'intelligenza artificiale del Giocatore 1 sceglierebbe la carta J ♢ perché ha valore 1/1, mentre l'altro figlio espanso (7 ♡) ha valore 0/1.

La profondità massima dell'albero della Briscola a 5 è di 40, questo valore corrisponde al numero di carte che non sono ancora state giocate; la profondità massima si ottiene all'inizio della partita quando tutte le 40 carte sono ancora in mano ai giocatori. Il fattore di ramificazione massimo è di 8, ossia il numero massimo di carte in mano ad un giocatore. Il numero totale di configurazioni finali (o nodi foglia) dell'albero, nella sua espansione completa, sarebbe di $(8!)^5$, cioè circa 10^{23} situazioni possibili.

5.2 MCTS con IA a Regole

La seconda variante della Ricerca ad Albero Monte Carlo è stata pensata per ridurre sensibilmente i nodi dell'albero. Assunto che l'algoritmo di intelligenza artificiale a regole costituisce un giocatore di buon livello, abbiamo

1: **function** UCT
2: **if** è il turno della mia squadra **then**
3: $turno = 1$
4: **else**
5: $turno = -1$
6: **end if**
7: $migliorValore \leftarrow -\infty * turno$
8: **for** ogni nodo figlio f **do**
9: $punteggioFiglio = \frac{f.valore}{f.viste+\epsilon}$
10: $bias = c\sqrt{\frac{visite}{f.visite+\epsilon}}$
11: $UCT = punteggioFiglio + bias * turno + rand$
12: **if** $UCT * turno > migliorValore * turno$ **then**
13: $figlioSelezionato \leftarrow f$
14: $migliorValore \leftarrow UCT$
15: **end if**
16: **end for**
17: **return** $figlioSelezionato$
18: **end function**

Algoritmo 3: Lo pseudo-codice dell'UCT da noi utilizzato. L'algoritmo si basa sul classico UCT, aggiungendo una costante piccolissima ϵ per evitare divisioni per 0 e una variabile casuale $rand$, anch'essa piccolissima, per rompere i pareggi. I valori $f.valore$ e $f.visite$ sono i corrispettivi campi nel nodo figlio di $valore$ (il punteggio totalizzato retro-propagando il valore finale) e $visite$ (il totale delle visite) per il nodo padre. Infine c è una costante positiva da settare. L'algoritmo restituisce il nodo $figlioSelezionato$ che viene poi espanso.

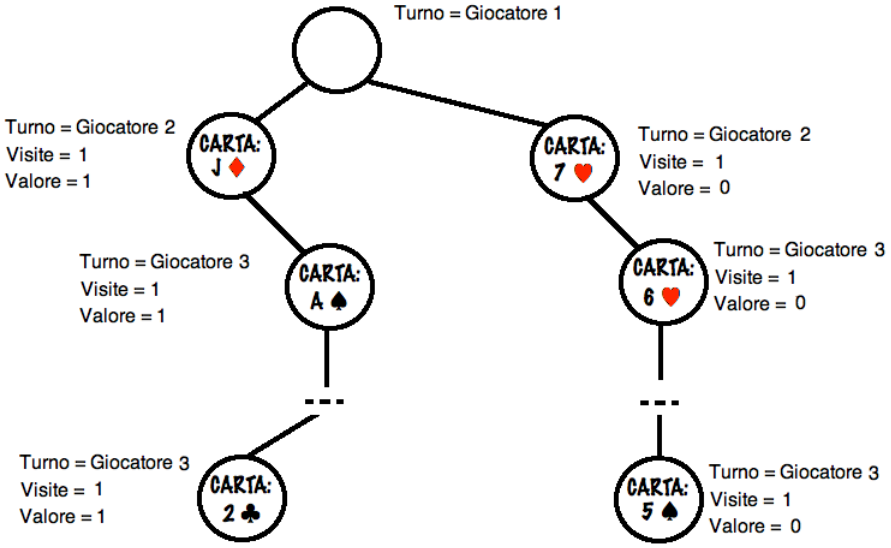

Figura 5.1: L'albero di ricerca dopo due iterazioni del metodo MCTS semplice con valutazione classica.

deciso di integrarlo come euristica per la riduzione degli stati. La fase di selezione rimane tale e quale alla versione base, cioè la scelta è fatta con l'UCT (*Algoritmo 3*). Invece nella fase di espansione, i nodi in cui è di turno il giocatore che implementa il metodo MCTS sono selezionati in maniera casuale, gli altri nodi sono selezionati secondo le regole dell'intelligenza artificiale descritta nel Capitolo 4. In questo modo il numero di stati finali sono ridotti da $(8!)^5$ a 8!, cioè 40320.

5.3 MCTS con Categorizzazione delle Carte

La terza variante della Ricerca ad Albero Monte Carlo ha anch'essa lo scopo di ridurre l'albero di ricerca della partita. Per ridurre la scelta sulla carta da giocare, durante la fase di espansione utilizziamo la tecnica della *cate-*

gorizzazione delle carte. Questo stratagemma consiste nel raggruppare in categorie le carte che hanno valore identico (o molto simile) durante un turno e considerarne una, eliminando di fatto dalla scelta le altre ridondanti. Le categorie in cui abbiamo diviso le carte sono:

- *briscole scartine vincenti* (tutte le briscole da 0 punti che superano la carta dominante, o qualsiasi briscola scartina se è la prima carta giocata nella mano).
- *briscole scartine non vincenti* (tutte le briscole da 0 punti che non superano la carta dominante)
- *scartine vincenti* (tutte le carte da 0 punti, non briscole, che strozzano la carta dominante, o giocate come prima carta della mano)
- *scartine* (tutte le carte da 0 punti che non possono prendere)
- *figure vincenti* (tutte le figure, non briscole, che strozzano la carta dominante, o giocate come prima carta della mano)
- *figure* (tutte le figure che non possono prendere)
- *carichi vincenti* (tutti i carichi, non briscole, che strozzano la carta dominante, o giocate come prima carta della mano)
- *carichi* (tutti i carichi che non possono prendere)

Esempio 1: supponiamo che il giocatore di turno abbia una mano composta da A ♠, J ♠, 7 ♢, 5 ♢, 2 ♢, 4 ♣, 2 ♣, J ♡ e che la carta dominante sia un 5 di Fiori, che è anche seme di briscola. Le sue carte sono categorizzate come *carichi* (A ♠), *figure* (J ♠, J ♡), *scartine* (7 ♢, 5 ♢, 2 ♢) e *briscole scartine non vincenti* (4 ♣, 2 ♣). In questo caso l'algoritmo tiene una sola carta per categoria, eliminando le altre, e la scelta si riduce da 8 carte a 4.

5.4 MCTS con IA a Regole e Categorizzazione delle Carte

Nell'ultima versione della Ricerca ad Albero Monte Carlo, abbiamo deciso di combinare le modifiche introdotte nelle due varianti precedenti. Nei nodi in cui il giocatore di turno è lo stesso su cui è implementato il metodo MCTS, l'espansione avviene in maniera casuale, scegliendo tra le carte rimaste dopo avere applicato la categorizzazione. Mentre agli altri nodi viene applicata l'euristica per l'espansione, ossia gli i giocatori scelgono secondo l'intelligenza artificiale a regole. Questa ultima versione del metodo MCTS riduce nettamente lo spazio di ricerca, più di tutte le altre varianti.

5.5 Sommario

In questo capitolo abbiamo presentato le versioni di intelligenza artificiale basate sulla Ricerca ad Albero Monte Carlo che abbiamo sviluppato per il nostro gioco. Nella Sezione 5.1 abbiamo descritto l'implementazione del metodo MCTS di base, analizzando in dettaglio l'algoritmo dell'UCT per la selezione del nodo. Nella Sezione 5.2 abbiamo presentato la variante del nostro algoritmo in cui il metodo MCTS è integrato con l'algoritmo di intelligenza artificiale a regole spiegato nel Capitolo 4. Nella Sezione 5.3 abbiamo descritto la tecnica della categorizzazione, cioè un raggruppamento delle carte giocabili in modo da ridurre la scelta nella fase di espansione. Nell'ultima sezione (5.4), abbiamo descritto la quarta versione del metodo MCTS in cui vengono combinate le tecniche utilizzate nelle due versione viste in precedenza.

Capitolo 6

Esperimenti

In questo capitolo presentiamo i risultati degli esperimenti che abbiamo condotto allo scopo di valutare la performance dei quattro algoritmi di intelligenza artificiale basati sul metodo MCTS al variare del numero di iterazioni dell'algoritmo e del ruolo assunto durante la partita.

6.1 Design Sperimentale

Gli esperimenti che abbiamo condotto consistono nella simulazione di mille partite di Briscola a 5. Per ogni esperimento abbiamo assegnato strategie differente a seconda del ruolo del giocatore, per poi confrontare la percentuale di vittoria delle due diverse intelligenze artificiali. Abbiamo messo a confronto cinque versioni di intelligenza artificiale: quella a regole (che d'ora in poi chiameremo *strategia a regole* o *esperto*), la Ricerca ad Albero Monte Carlo senza euristiche (che chiameremo *MCTS1*), la Ricerca ad Albero Monte Carlo con integrazione dell'algoritmo a regole (*MCTS2*), la Ricerca ad Albero Monte Carlo con categorizzazione delle carte (*MCTS3*) e la Ricerca ad Albero Monte Carlo con entrambe le tecniche (*MCTS4*); infine abbiamo aggiunto un giocatore che seleziona le carte in maniera casuale (che nelle tabelle e nei grafici è definito *Random*). Allo scopo di avere un

comportamento equo, in tutti gli esperimenti è stato utilizzato per la fase d'asta l'algoritmo descritto nella Sezione 4.2. Facciamo notare che tutti i giocatori non conoscono le carte degli avversari. I metodi MCTS1 e MCTS3 utilizzano avversari casuali, mentre i metodi MCTS2 e MCTS usano un esperto che è all'oscuro delle carte degli avversari.

6.2 Esperimento 1: Strategia Casuale Contro Strategia Casuale

Nel primo esperimento abbiamo assegnato a tutti i giocatori la strategia casuale. I risultati, riassunti nella *Tabella 6.1*, mostrano uno squilibrio nelle vittorie a favore del chiamante e del chiamato. Più del 71 % delle partite sono vinte dalla squadra del chiamante, mentre la squadra degli avversari vince poco meno del 27 % delle partite. In questo modo siamo riusciti a quantificare un dato risaputo anche se mai confermato da esperimenti, ossia che il chiamante, effettuando una chiamata in maniera adeguata, ha un vantaggio competitivo che lo porta a vincere la maggior parte delle partite.

Random VS Random		
Vittorie Squadra Chiamante	Vittorie Squadra Avversari	Pareggi
71.52 %	26.94 %	1.54 %

Tabella 6.1: Percentuale di vittorie delle due squadre nelle quali tutti i giocatori adottano la strategia casuale.

6.3 Esperimento 2: Strategia a Regole Contro Strategia a Regole

Nel secondo esperimento abbiamo assegnato a tutti i giocatori la strategia a regole. La *Tabella 6.2* mostra che, come nell'esperimento precedente,

chiamante e chiamato hanno un vantaggio durante la partita. Il divario aumenta rispetto all'esperimento con giocatori che adottano strategia casuale in quanto l'algoritmo a regole sfrutta leggermente meglio la conoscenza del dominio di chiamante e chiamato rispetto a quella degli avversari. Il motivo di questa differenza è la difficoltà di tradurre in regole l'aspetto psicologico che porta un giocatore a fidarsi o meno di un altro giocatore.

Esperto VS Esperto		
Vittorie Squadra Chiamante	Vittorie Squadra Avversari	Pareggi
74.12 %	24.58 %	1.3 %

Tabella 6.2: Percentuale di vittorie delle due squadre nelle quali tutti i giocatori adottano la stretegia a regole

6.4 Esperimento 3: MCTS Contro MCTS

Nel terzo esperimento abbiamo assegnato a tutti i giocatori la strategia MCTS, abbiamo poi misurato il variare della performance all'aumentare del numero di iterazioni dell'algoritmo. Come mostrano la *Figura 6.1* e la *Tabella 6.3*, la curva che rappresenta la percentuale di vittoria della squadra del chiamante si appiattisce dopo pochissime iterazioni, circa un centinaio. Questo esperimento conferma ancora una volta che la squadra del chiamante, con una chiamata opportuna, parte avvantaggiata, tuttavia non avendo il metodo MCTS conoscenza del dominio, questo divario non è così grande come nei casi precedenti. Un altro modo di leggere i risultati è che la Ricerca ad Albero Monte Carlo gioca in maniera migliore nel ruolo degli avversari. Il grafico inoltre mostra che, con un numero basso di iterazioni, la percentuale di vittorie delle due squadre tende a quella della partita composta da tutti i giocatori con strategia casuale (si veda *Tabella 6.4*).

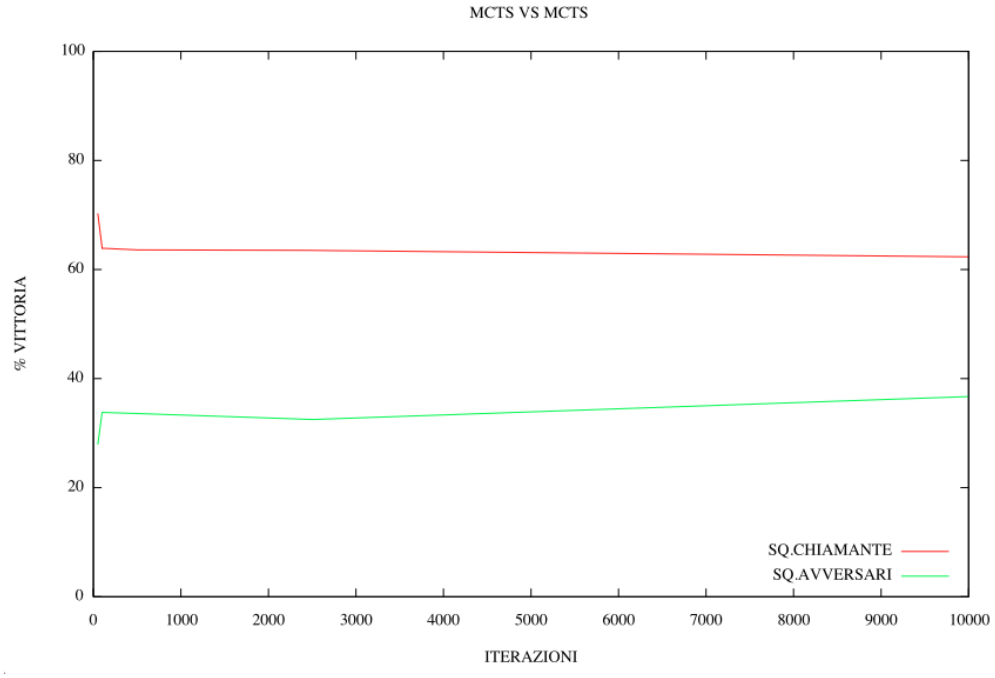

Figura 6.1: Percentuale di vittorie delle due squadre nelle quali tutti i giocatori adottano la strategia MCTS1.

MCTS VS MCTS con più di 100 iterazioni		
Media Vitt. Sq. Chiamante	Media Vitt. Sq. Avversari	Media Pareggi
63.33 %	34.15 %	2.52 %

Tabella 6.3: Percentuale media delle vittorie delle due squadre nelle quali tutti i giocatori adottano la stretegia MCTS con più di 100 iterazioni

MCTS VS MCTS con meno di 100 iterazioni		
Media Vitt. Sq. Chiamante	Media Vitt. Sq. Avversari	Media Pareggi
70.3 %	27.9 %	1.8 %

Tabella 6.4: Percentuale media delle vittorie delle due squadre nelle quali tutti i giocatori adottano la stretegia MCTS con meno di 100 iterazioni

6.5 Esperimento 4: Strategia a Regole Contro Strategia Casuale

Nel quarto esperimento abbiamo messo a confronto la strategia a regole con quella casuale. Inizialmente abbiamo simulato una serie di partite nelle quali chiamante e chiamato utilizzavano l'algoritmo a regole, mentre gli avversari la scelta casuale. Successivamente abbiamo condotto lo stesso esperimento invertendo però le strategie, ossia casuale per chiamante e chiamato, a regole per gli avversari. La *Tabella 6.5* mostra che la percentuale di vittoria della squadra del chiamante che adotta la strategia a regole è più del 92 % e quella degli avversari che giocano in maniera casuale supera il 6 %. Questo significa che, nonostante il grande vantaggio della squadra del chiamante, c'è una piccola probabilità che gli avversari riescano anche giocando in maniera non ottimale a vincere la partita. Mentre la *Tabella 6.6* conferma il vantaggio competitivo di chiamante e chiamato che, effettuando una chiamata adeguata, possono vincere più di un terzo delle partite scegliendo in maniera casuale la mossa contro avversari esperti.

Esperto VS Random		
Vittorie Squadra Chiamante	Vittorie Squadra Avversari	Pareggi
92.58 %	6.75 %	0.67 %

Tabella 6.5: Percentuale di vittorie delle due squadre nelle quali chiamante e chiamato utilizzano la strategia a regole, mentre gli avversari adottano la strategia casuale.

Random VS Esperto		
Vittorie Squadra Chiamante	Vittorie Squadra Avversari	Pareggi
34.43 %	63.94 %	1.63 %

Tabella 6.6: Percentuale di vittorie delle due squadre nelle quali chiamante e chiamato utilizzano la strategia casuale, mentre gli avversari adottano la strategia a regole.

6.6 Esperimento 5: MCTS Contro Strategia Casuale

Lo scopo del quinto esperimento è quello di esaminare il cambio nella performance dell'algoritmo MCTS al variare del numero di iterazioni per la costruzione dell'albero. Abbiamo eseguito una serie di simulazioni di partite in cui chiamante e chiamato utilizzavano il metodo MCTS con 50 iterazioni, mentre gli avversari adottavano la strategia casuale. Successivamente abbiamo aumentato progressivamente il numero di iterazioni del metodo MCTS fino ad arrivare a 10000 e ripetuto l'esperimento invertendo i ruoli.

Come si può vedere nelle *Figure 6.2 e 6.3*, la percentuale di vittorie di tutti i metodi MCTS aumenta col numero di iterazioni raggiungendo un risultato stabile al di sopra dalle 2500 iterazioni circa. Il risultato migliore è ottenuto dal metodo MCTS1, seguito dai metodi MCTS3, MCTS2 e MCTS4. La performance del metodo MCTS2 è inferiore a quella dell'MCTS1 a causa dell'overfitting, cioè un eccessivo adattamento al modello e all'errore commesso dalla strategia a regole usata come euristica. Il metodo MCTS3 soffre invece degli effetti negativi causati dalla riduzione della scelta (la perdita di potenziali stati ottimi), che risultano essere superiori ai benefici (la riduzione degli stati dell'albero). La performance del metodo MCTS4 è la peggiore tra le quattro. La causa di questi risultati inferiori è la combinazione delle cause che rendono peggiori dell'MCTS1 le performance dell'MCTS2 e dell'MCTS3.

Le (*Tabelle 6.7 e 6.8*) mostrano il confronto tra i risultati dei metodi MCTS e quelli della strategia a regole contro avversari che adottano la strategia casuale.

Come si può vedere, i metodi MCTS1, MCTS2 e MCTS3 nel ruolo di chiamante e chiamato al di sopra delle 2500 iterazioni risultano essere leggermente migliori della strategia a regole; mentre nel ruolo degli avversari i

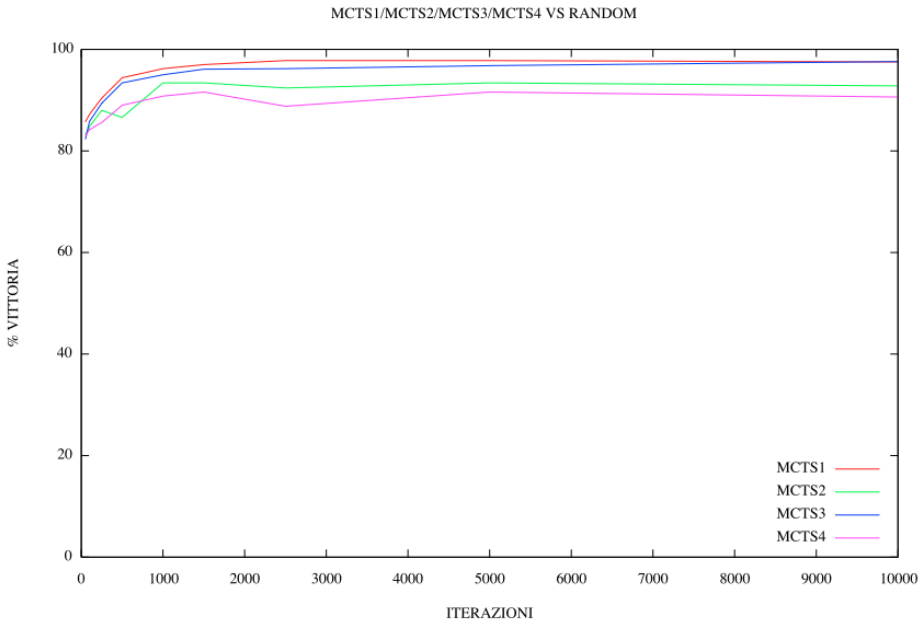

Figura 6.2: Confronto tra la percentuale di vittorie dei metodi MCTS che giocano nel ruolo di chiamante e chiamato contro avversari che adottano la strategia casuale.

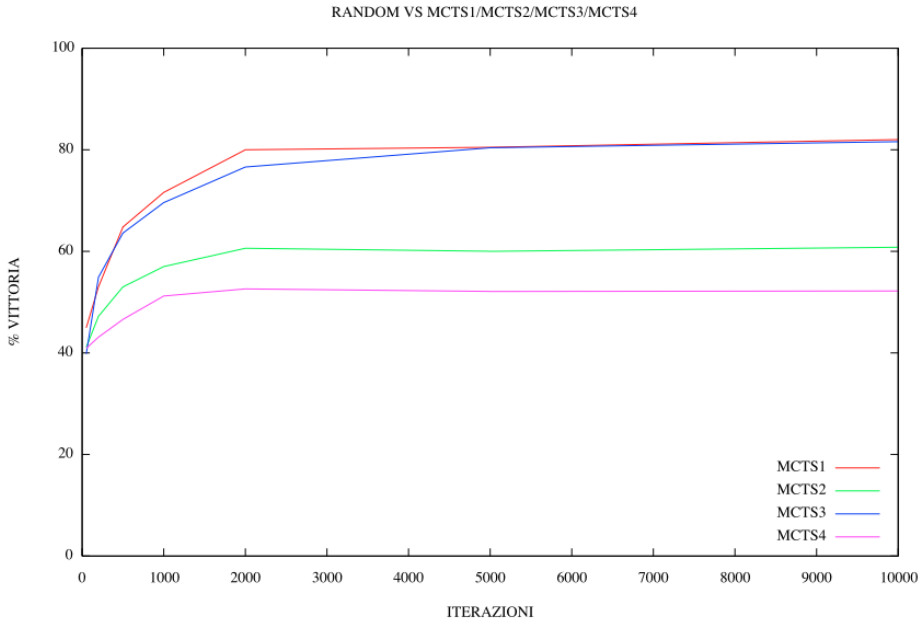

Figura 6.3: Confronto tra la percentuale di vittorie dei metodi MCTS che giocano nel ruolo degli avversari contro chiamante e chiamato che adottano la strategia casuale.

metodi MCTS1 e MCTS3 ottengono risultati nettamente superiori a quelli ottenuti dalla strategia a regole.

IA VS Avversari Random			
IA	Vitt. Sq, Chiamante	Vitt. Sq. Avversari	Pareggi
Esperto	92.58 %	6.75 %	0.67 %
MCTS1	97.7 %	2.2 %	0.1 %
MCTS2	93 %	6.63 %	0.37 %
MCTS3	96.87 %	2.8 %	0.33 %
MCTS4	90.68 %	8.64 %	0.68 %

Tabella 6.7: Confronto tra la percentuale media di vittorie dei quattro metodi MCTS e della strategia a regole che giocano nel ruolo di chiamante e chiamato contro gli avversari che adottano la strategia casuale.

Chiamante e Chiamato Random VS IA			
IA	Vitt. Sq, Chiamante	Vitt. Sq. Avversari	Pareggi
Esperto	34.43 %	63.94 %	1.63 %
MCTS1	16.7 %	81.4 %	1.9 %
MCTS2	39.07 %	60.47 %	0.46 %
MCTS3	18.53 %	79.53 %	1.94 %
MCTS4	46.23 %	52.3 %	1.47 %

Tabella 6.8: Confronto tra la percentuale media di vittorie dei quattro metodi MCTS e della strategia a regole che giocano nel ruolo degli avversari contro chiamante e chiamato che adottano la strategia casuale.

6.7 Esperimento 6: MCTS Contro Strategia a Regole

L'ultimo esperimento segue le modalità di quello precedente, mettendo però a confronto i metodi MCTS contro quello a regole.

Come si può vedere nelle *Figure 6.4 e 6.5*, in maniera simile all'esperimento precedente, la percentuale di vittorie di tutti i metodi MCTS aumenta col numero di iterazioni raggiungendo un risultato stabile al di sopra dalle 2500 iterazioni circa. Anche in questo caso il risultato migliore è ottenuto dal metodo MCTS1, seguito dai metodi MCTS3, MCTS2 e MCTS4. Tuttavia si può notare che, con poche iterazioni, il metodo con performance superiore è l'MCTS2 grazie all'integrazione della strategia a regole, che permette di ridurre gli stati da visitare prevedendo le mosse utilizzate dalla squadra avversaria. Questo dato non si riscontra con un numero di iterazioni superiori a causa dell'overfitting.

Le *Tabelle 6.9 e 6.10* mostrano il confronto tra i risultati dei metodi MCTS e quelli della strategia a regole contro avversari che adottano la strategia a regole.

Oltre alle 2500 iterazioni i metodi MCTS1 e MCTS3 sono migliori della strategia a regole nel ruolo di chiamante e chiamato. La grossa differenza rispetto agli esperimenti precedenti è che tutti i metodi MCTS nel ruolo degli avversari sono migliori della strategia a regole. Questo dimostra ancora una volta che i metodi basati sulla Ricerca ad Albero Monte Carlo sono molto più competitivi nel ruolo degli avversari rispetto all'algoritmo a regole, che non riesce a sfruttare la conoscenza del dominio in maniera efficace.

6.8 Sommario

In questo capitolo abbiamo presentato i risultati degli esperimenti condotti sugli algoritmi di intelligenza artificiali da noi sviluppati per il gioco della

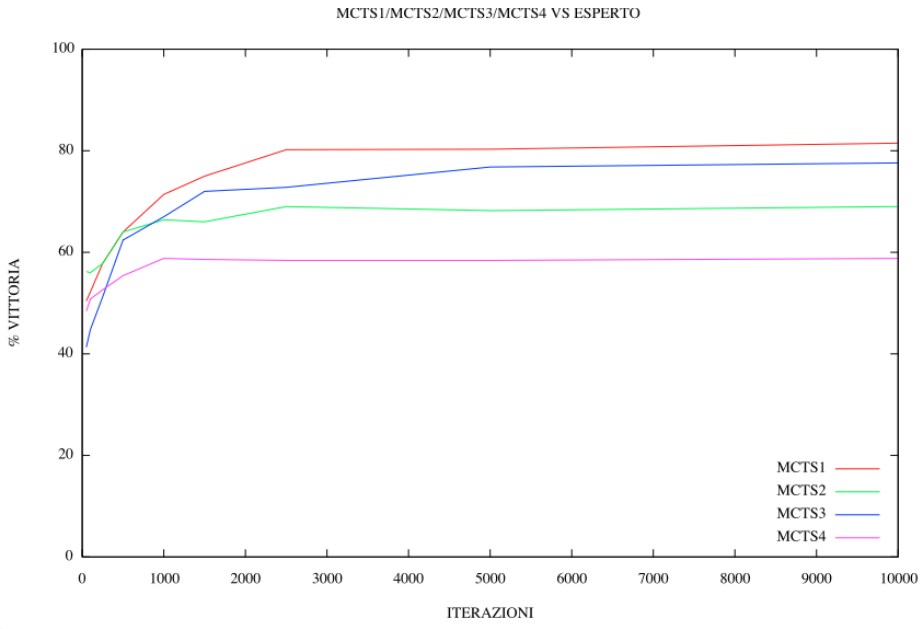

Figura 6.4: Confronto tra la percentuale di vittorie dei metodi MCTS che giocano nel ruolo di chiamante e chiamato contro avversari che adottano la strategia a regole.

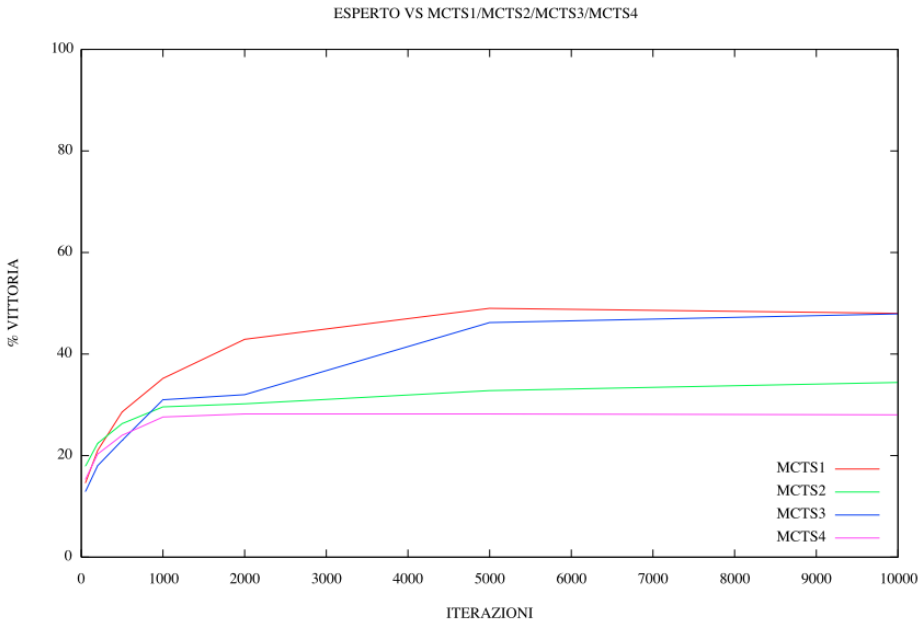

Figura 6.5: Confronto tra la percentuale di vittorie dei metodi MCTS che giocano nel ruolo degli avversari contro chiamante e chiamato che adottano la strategia a regole.

IA VS Avversari Esperti			
IA	Vitt. Sq, Chiamante	Vitt. Sq. Avversari	Pareggi
Esperto	74.12 %	24.58 %	1.3 %
MCTS1	80.7 %	17.86 %	1.54 %
MCTS2	68.7 %	30.6 %	0.7 %
MCTS3	77.2 %	21.4 %	1.4 %
MCTS4	58.6 %	40.6 %	0.8 %

Tabella 6.9: Confronto tra la percentuale media di vittorie dei quattro metodi MCTS e della strategia a regole che giocano nel ruolo di chiamante e chiamato contro gli avversari che adottano la strategia a regole.

Chiamante e Chiamato Esperti VS IA			
IA	Vitt. Sq, Chiamante	Vitt. Sq. Avversari	Pareggi
Esperto	74.12 %	24.58 %	1.3 %
MCTS1	49.16 %	48.6 %	0.93 %
MCTS2	66.6 %	32.47 %	0.93 %
MCTS3	56.37 %	42.03 %	1.6 %
MCTS4	70.67 %	28.13 %	1.2 %

Tabella 6.10: Confronto tra la percentuale media di vittorie dei quattro metodi MCTS e della strategia a regole che giocano nel ruolo degli avversari contro chiamante e chiamato che adottano la strategia a regole.

Briscola a 5. I risultati mostrano che, adottando una strategia di chiamata opportuna nella fase d'asta, il chiamante e il chiamato hanno un vantaggio competitivo sugli avversari, un dato risaputo anche se mai confermato da esperimenti. Questo vantaggio porta la squadra del chiamante a vincere circa due terzi delle partite. Abbiamo poi riscontrato che il metodo MCTS (in particolare la sua versione senza euristiche) gioca ad un livello superiore all'algoritmo di intelligenza artificiale che si basa sulle regole fornite da giocatori esperti e libri di strategie. I dati mostrano che, mentre le regole che codificano la conoscenza risultano competitive nel ruolo di chiamante e chiamato, non lo sono altrettanto per quanto riguarda gli avversari, a differenza del metodo MCTS che ottiene risultati nettamente superiori.

Capitolo 7

Conclusioni

L'obiettivo di questa tesi era quello di creare un algoritmo di intelligenza artificiale competitivo per il gioco della Briscola a 5. Per fare questo, con la collaborazione di esperti del settore e testi specializzati, abbiamo sviluppato un algoritmo a regole che codifica la conoscenza del dominio di un giocatore umano. In seguito abbiamo creato quattro algoritmi di intelligenza artificiale basati sulla Ricerca ad Albero Monte Carlo. Nella prima versione abbiamo applicato il metodo base, con esplorazione casuale dei nodi. Nella seconda versione abbiamo integrato l'algoritmo a regole come euristica di espansione dei nodi. Nella terza versione abbiamo ridotto la scelta sulla mossa raggruppando in categorie le carte che hanno valore simile, in modo da ridurre lo spazio di ricerca. Nella quarta versione abbiamo applicato entrambe tecniche delle due precedenti versioni. L'analisi sperimentale mostra che le prestazioni migliori sono raggiunte dalla prima versione, seguito dalla terza, dalla seconda e dalla quarta. Al di sopra di un certo numero di iterazioni le performance dell'algoritmo basato sul metodo MCTS sono superiori a quelle dell'algoritmo a regole, soprattutto nel ruolo degli avversari, per i quali la conoscenza del dominio non è sfruttata quanto per chiamante e chiamato. Inoltre abbiamo valutato quantitativamente l'influenza di una chiamata adeguata durante la fase d'asta. I risultati ottenuti suggeri-

scono che, effettuando una chiamata opportuna durante la fase d'asta, la squadra del chiamante ha un vantaggio competitivo rispetto a quella degli avversari, che lo porta a vincere intorno ai due terzi delle partite. Questo dato è in contrasto con l'assegnazione alla fine della partita, infatti, vincendo, il chiamante riceve il doppio dei punti nonostante sia il giocatore più avvantaggiato.

Grazie agli ottimi risultati ottenuti in questa tesi, abbiamo sviluppato un prototipo dell'applicazione del gioco della Briscola a 5 per Android e iOS, che pianifichiamo di mettere al più presto nei market.

7.1 Sviluppi Futuri

Dato che auspichiamo di pubblicare il gioco, prevediamo di effettuare degli esperimenti con avversari umani, dal momento che il numero di quelli tuttora effettuati non è sufficiente per trarre conclusioni. Questi esperimenti hanno l'obiettivo di testare il grado di difficoltà e la ragionevolezza del comportamento umano dell'intelligenza artificiale. Inoltre un ulteriore miglioramento potrebbe essere l'integrazione di altre regole non implementate nella versione attuale come la chiamata in mano e la chiamata ai punti. Il gioco attualmente è utilizzabile da giocatore singolo. Un possibile miglioramento da apportare all'applicazione sarebbe l'aggiunta di una modalità multi-giocatore.

Appendice A

L'Applicazione

A.1 Sviluppo dell'Applicazione

Abbiamo sviluppato la prima versione del gioco della Briscola a 5 in Java. Questa versione di ricerca ci ha permesso di effettuare una serie di esperimenti, grazie ai quali abbiamo valutato la performance dei vari algoritmi di intelligenza artificiale. Per la valutazione del comportamento dell'intelligenza artificiale basata sul metodo MCTS questa versione ci ha consentito di visualizzare la rappresentazione dell'albero di ricerca costruito dall'algoritmo (*Figura A.1*).

Successivamente abbiamo sviluppato il prototipo della nostra applicazione, utilizzando Unity3D e il linguaggio di programmazione C# per lo scripting. Unity3D [35] è un ambiente di sviluppo per videogiochi o animazioni in 3D e rende disponibili un insieme di librerie in C#, Javascript e Boo per gestire la grafica, la fisica, gli effetti sonori e gli input dell'utente. Integra inoltre l'IDE open source MonoDevelop [33] per gestire il sistema di scripting. Il grosso vantaggio di Unity è che supporta lo sviluppo su più piattaforme, comprese Windows, OS X, Linux, iOS, Android, BlackBerry 10, web browser e console di gioco. Per la nostra applicazione abbiamo utilizzato la versione 4 di Unity3D. Per l'interfaccia utente e i menu di gioco

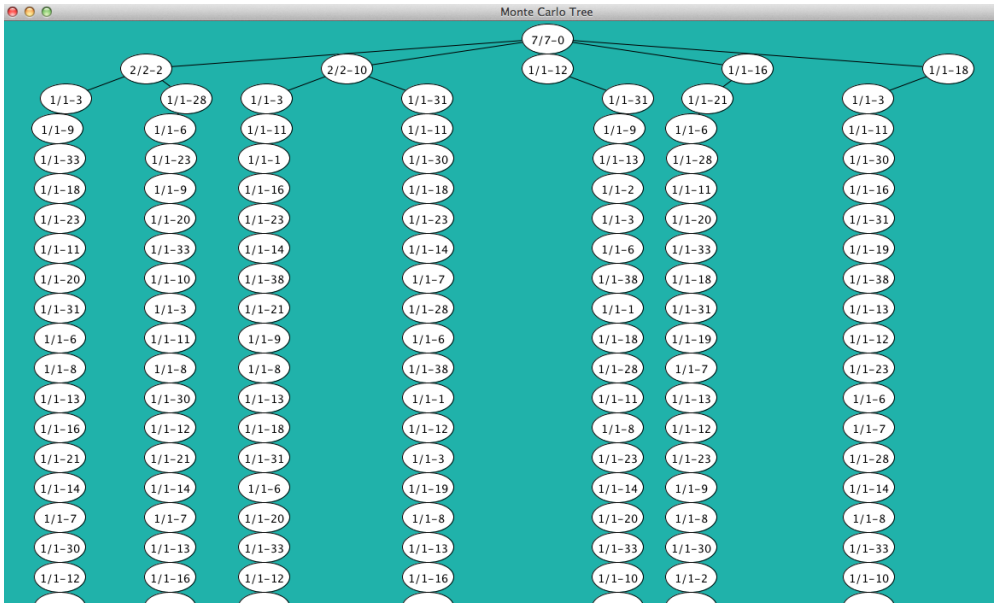

Figura A.1: Una rappresentazione dell'albero di ricerca del metodo MCTS creato dalla versione di ricerca in Java. I nodi contengono il punteggio ottenuto, il numero di visite e un numero che codifica la carta giocata

abbiamo utilizzato il pacchetto NGUI.

A.2 Struttura dell'Applicazione

Dato che una partita di Briscola a 5 può risultare molto lunga, abbiamo deciso di consentire all'utente di selezionare se giocare una partita singola oppure una partita classica a punteggio. A secondo del livello di difficoltà viene selezionata una differente intelligenza artificiale. Dopodiché la partita inizia, un giocatore scelto in maniera casuale inizia l'asta e il gioco prosegue secondo le regole descritte nel Capitolo 3. Il giocatore non ha limiti di tempo per pensare alla propria mossa e decide in che momento far giocare la carta agli avversari. La schermata di gioco mostra il proprio ruolo e la posizione del chiamante, mentre i ruoli degli altri vengono mostrati solamente dopo che il chiamato gioca la carta chiamata (come si può vedere nella *Figura A.2*).

Alla fine di ogni partita una schermata mostra il punteggio delle squadre per quella partita e la classifica aggiornata coi punteggi dei giocatori.

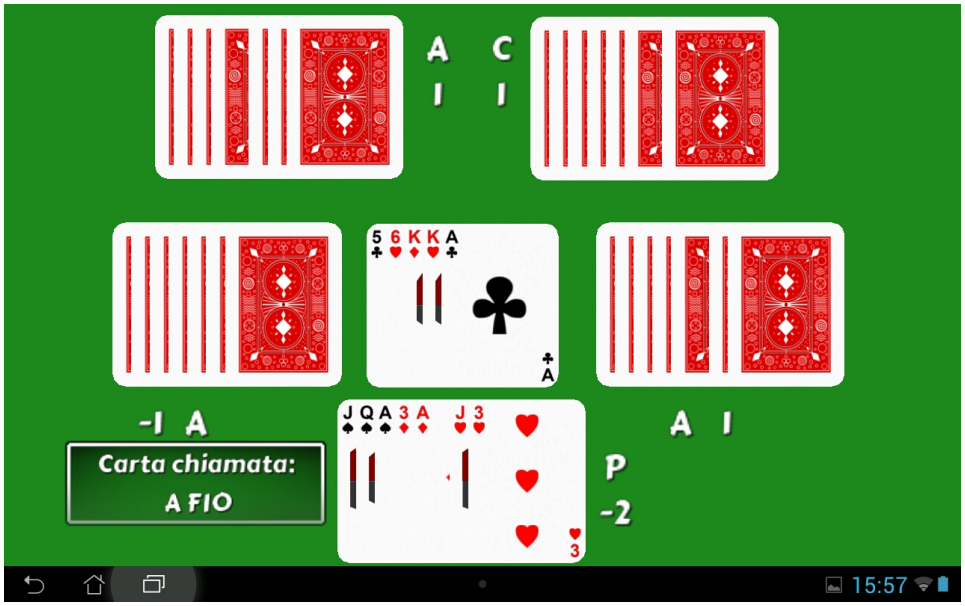

Figura A.2: Uno screenshot dell'applicazione installata su un tablet Android.

Bibliografia

[1] Broderick Arneson, Ryan B. Hayward, and Philip Henderson. Monte carlo tree search in hex. *IEEE Trans. on Comput'l Intel. and AI in Games*, 2(4):251–257, Dicembre 2010.

[2] Paul M Bethe. The state of automated bridge play. Gennaio 2010.

[3] Darse Billings, Aaron Davidson, Jonathan Schaeffer, and Duane Szafron. The challenge of poker. *Artificial Intelligence*, (134):201–240, 2002.

[4] Bruno Bouzy and Tristan Cazenave. Computer go: An ai oriented survey. *Artificial Intelligence*, (132):39–103, 2001.

[5] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES*, 4(1):49, Marzo 2012.

[6] Peter I. Cowling, Colin D. Ward, and Edward J. Powley. Ensemble determinization in monte carlo tree search for the imperfect information card game magic: The gathering. *IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES*, X(X), 2012.

[7] E. Fantini. *Il maxi libro dei giochi di carte.* De Vecchi, 2010.

[8] G. Farina and A. Lamberto. *Enciclopedia delle carte. La teoria e la pratica di oltre 1000 giochi.* Hoepli, 2006.

[9] P. Gorini. *Impara l'arte... delle carte. 27 giochi di carte italiani e internazionali, più 9 giochi originali.* Quattro e quattrotto. L'Airone, 2004.

[10] http://chessprogramming.wikispaces.com. Chess programming.

[11] http://fuego.sourceforge.net/. Computer go group at the university of alberta, fuego.

[12] http://mcts.ai/about/index.html. Monte carlo tree search. computational creativity group (ccg) of the department of computing, imperial college london, uk.

[13] http://www.alanturing.net. History of ai.

[14] http://www.pagat.com/aceten/briscola.html. Briscola.

[15] Wan Jing Loh. Ai mahjong. December 2009.

[16] A.M. Nijssen and Mark H.M. Winands. Monte-carlo tree search for the game of scotland yard.

[17] Marc Ponsen, Geert Gerritsen, and Guillaume Chaslot. Integrating opponent models with monte-carlo tree search in poker.

[18] S.J. Russell and P. Norvig. *Intelligenza artificiale. Un approccio moderno.* Number v. 1 in Intelligenza artificiale. Un approccio moderno. Pearson Education Italia, 2005.

[19] Jonathan Schaeffer, Neil Burch, Yngvi Björnsson, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen. Checkers is solved. *Science*, 317(5844):1518–1522, Settembre 2007.

[20] Brian Sheppard. World-championship-caliber scrabble. *Artificial Intelligence*, (134):241–275, 2002.

[21] Stephen J. J. Smith, Dana Nau, and Tom Throop. Computer bridge, a big win for ai planning. *AI Magazine*, 19(2), 1998.

[22] Daniel Whitehouse, Peter I. Cowling, and Edward J. Powley. Integrating monte carlo tree search with knowledge-based methods to create engaging play in a commercial mobile game. 2013.

[23] Wikipedia. Computer chess — wikipedia, the free encyclopedia.

[24] Wikipedia. Ferranti mark 1 — wikipedia, l'enciclopedia libera.

[25] Wikipedia. Ibm deep blue — wikipedia, l'enciclopedia libera.

[26] Wikipedia. Storia dei software scacchistici — wikipedia, l'enciclopedia libera.

[27] Wikipedia. Maven (scrabble) — wikipedia, the free encyclopedia, 2011.

[28] Wikipedia. Arthur samuel — wikipedia, the free encyclopedia, 2013.

[29] Wikipedia. Artificial intelligence (video games) — wikipedia, the free encyclopedia, 2013.

[30] Wikipedia. Bridge (gioco) — wikipedia, l'enciclopedia libera, 2013.

[31] Wikipedia. Briscola — wikipedia, l'enciclopedia libera, 2013.

[32] Wikipedia. Computer othello — wikipedia, the free encyclopedia, 2013.

[33] Wikipedia. Monodevelop — wikipedia, the free encyclopedia, 2013.

[34] Wikipedia. Solved game — wikipedia, the free encyclopedia, 2013.

[35] Wikipedia. Unity (motore di gioco) — wikipedia, l'enciclopedia libera, 2013.