ForSA 4

Annealing Simulato nel gioco Forza 4

Davide Petturiti

Ricerca Operativa
Prof. Loris Faina – A.A. 2007/2008

—

Corso di Laurea Specialistica in Informatica
Facoltà di Scienze Matematiche, Fisiche e Naturali
Università degli Studi di Perugia

Indice

1 Introduzione

Canonicamente, l'algoritmo di Annealing Simulato viene presentato contestualmente ai problemi di ottimizzazione combinatoriale e, più precisamente, nella categoria degli algoritmi di ricerca locale, come particolare variante dell'algoritmo hill-climbing stocastico. D'altro canto i problemi di ricerca con avversari, ovvero i giochi, sono affrontati in letteratura con tecniche quali algoritmi MIN-MAX e potature alfa-beta.

Infatti, nei giochi per due giocatori si è soliti costruire un albero di ricerca in cui ogni nodo rappresenta un possibile stato del gioco ed in particolare, le foglie corrispondono ad uno stato terminale di vittoria, pareggio o sconfitta. La Figura 1 mostra una porzione dell'albero di ricerca per il gioco del Tris.

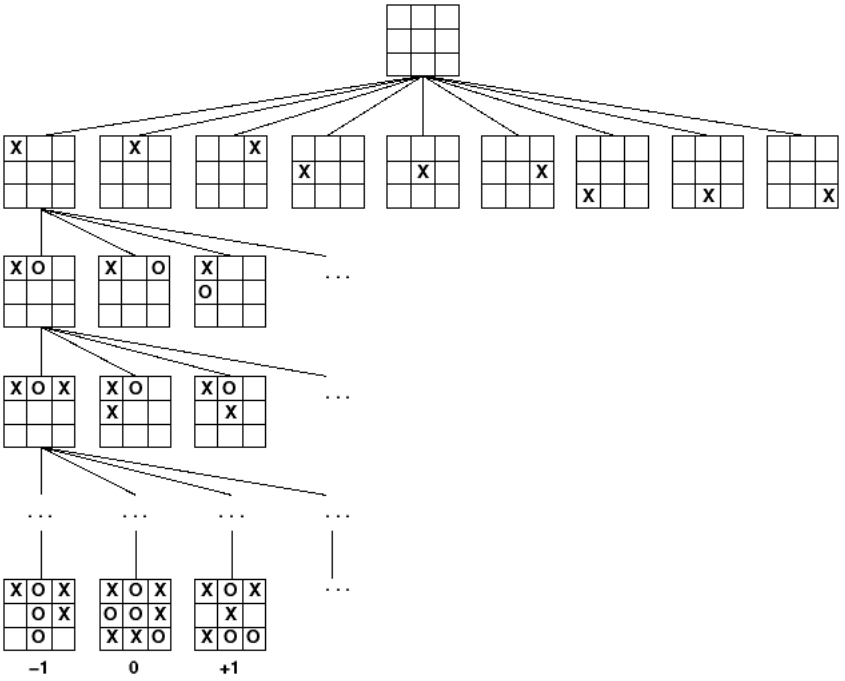

Figura 1: Porzione dell'albero di ricerca per il gioco del Tris

In genere il programma di gioco viene chiamato MAX, in quanto tende a massimizzare la sua funzione utilità, mentre il giocatore "reale" è chiamato MIN, in quanto tende a minimizzare la funzione utilità del programma.

Ad ogni foglia è associato un numero che indica il valore di utilità di quello stato terminale: i valori alti sono buoni per MAX e cattivi per MIN. E' compito di MAX, ovvero del programma di gioco, analizzare l'albero di ricerca in modo da determinare la mossa migliore, cioè quella che porta la massima utilità.

Nei consueti problemi di ottimizzazione, la soluzione ottima è costituita da una sequenza di mosse che portano ad uno stato obbiettivo; in un gioco, d'altra parte, il giocatore reale può dire la sua, quindi il programma di gioco deve trovare una strategia ottima, cioè il cui risultato sia almeno pari a qualsiasi altra strategia.

Come si vedrà in seguito, anche un gioco apparentemente semplice come Forza 4 è troppo complesso per riuscire a rappresentare interamente l'albero di ricerca.

Gli algoritmi di MIN-MAX si basano sul calcolo ricorsivo dell'utilità di una mossa eseguendo una visita completa in profondità dell'albero di ricerca in modo da "portare su" l'utilità delle foglie, fino al nodo corrente.

Chiaramente, da un punto di vista computazionale un approccio del genere risulta talvolta impraticabile a causa della sua complessità temporale, di ordine esponenziale. A questo proposito interviene la potatura alfa-beta che permette di "potare" i rami o i sottolaberi che non possono influenzare la decisione.

Tuttavia, anche la ricerca con potatura necessita di percorrere l'albero fino alle foglie, almeno per una porzione dello spazio di ricerca.

Quindi, per avere una diminuzione effettiva della complessità temporale, ci si concentra sul tagliare la ricerca prima di arrivare alle foglie, applicando una funzione di valutazione euristica agli stati, cioè, di fatto, trasfromando i nodi non terminali dell'albero di ricerca, in foglie.

Nel presente lavoro verrà affrontato il problema del gioco Forza 4 costruendo una opportuna **funzione di valutazione** degli stati e provvedendo a massimizzare questa funzione tramite l'algoritmo di Annealing Simulato, in modo da scegliere la mossa migliore. Tale funzione è sostanzialmente costituita da due parti che si occupano di rappresentare la bontà di una mossa sia per la situazione attuale che per l'immediato futuro.

E' ovvio che la costruzione della funzione di valutazione riveste un ruolo cruciale per tutto il lavoro, infatti, se inaccurata, tale funzione potrebbe portare il programma di gioco verso mosse perdenti.

Nello specifico, il lavoro si è concluso con la scrittura di un programma nel linguaggio C++, per la cui parte grafica sono state usate le librerie, di libera distribuzione, Qt. Il programma in questione è stato chiamato **ForSA4** per indicare l'uso del Simulated Annealing nella strategia di gioco.

2 Il gioco Forza 4

Forza 4, conosciuto anche con i nomi di Connect 4, Vertical Tic-Tac-Toe, Gravitational Tic-Tac-Toe e 4-in-a-row, è un gioco da tavolo astratto inventato da Milton Bradley nel 1974.

Si tratta di un gioco di allineamento che richiama i giochi del Filetto e del Tris,

il conosciutissimo gioco di carta e matita che si fa su carta quadrettata cercando di costruire una serie continua di simboli in una matrice 3×3.

Forza 4 è un gioco per due persone, in cui entrambi i giocatori hanno 21 gettoni rispettivamente di colore diverso (solitamente giallo e rosso), da disporre in una matrice 6×7. In questa sede, supporremo che i gettoni del programma di gioco siano gialli mentre quelli del giocatore siano rossi. La Figura 2 riporta un esempio di griglia per il gioco Forza 4.

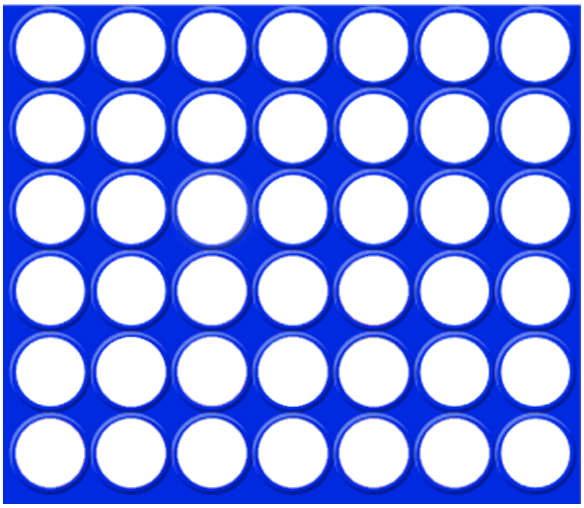

Figura 2: Esempio di griglia per il gioco Forza 4

Analogamente al Tris, l'obiettivo è mettere in fila 4 pedine del proprio colore, ma l'elemento fondamentale del gioco, che lo rende del tutto originale, è la forza di gravità: la griglia è infatti posta in verticale fra i giocatori, ed i gettoni vengono fatti cadere lungo le colonne, in modo tale che un gettone inserito in una certa colonna vada sempre ad occupare la posizione libera situata più in basso nella colonna stessa.

I giocatori eseguono le loro mosse a turno e non ci sono regole per chi debba cominciare. Entrambi i giocatori devono cercare di allineare 4 gettoni del loro colore orizzontalmente, verticalmente o diagonalmente ed il primo che riesce nell'intento, vince il gioco. La Figura 3 mostra un esempio dei tre possibili casi di allineamento.

Se tutti i 42 gettoni sono stati giocati e nessuno dei due giocatori ne ha allineati 4 del suo colore, la partita finisce in parità.

Nel gioco ricopre una importanza strategica, il cosiddetto **zugzwang** che consiste nel forzare l'avversario a fare una mossa che altrimenti non farebbe. E' bene, quindi, che l'algortimo di gioco abbia controllo, il più possibile dello zugzwang.

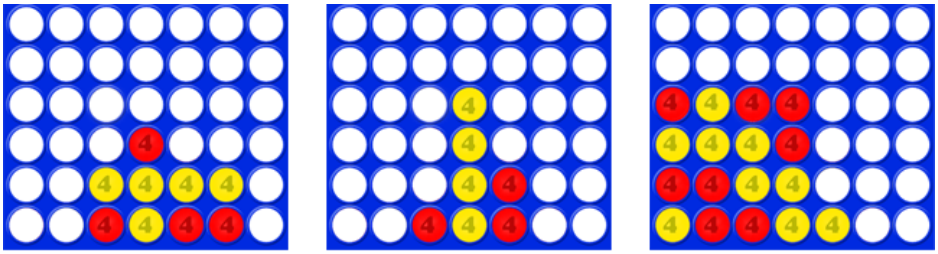

Figura 3: Esempi di allineamento orizzontale, verticale e diagonale

3 Complessità del gioco

Intuitivamente, viste le dimensioni ridotte della matrice di gioco si potrebbe pensare di affrontare il problema con un algoritmo brute-force.

Siccome ogni cella della matrice può essere in tre stati (vuota, gettone rosso o gettone giallo), il numero di possibili configurazioni della griglia è 3^{42}, tuttavia tale upper bound è abbastanza "grezzo" in quanto prende in considerazione anche configurazioni illegali e partite continuate nonostante una vittoria. Escludendo gli stati non consentiti, è stato trovato un upper bound di 7.1×10^{13} che rende, comunque, infattibile un approccio brute-force.

Ciò nonostante, Forza 4 è stato risolto nel 1988, indipendentemente da Victor Allis e James D. Allen. La soluzione mostra che un giocatore può sempre forzare una propria vittoria cominciando dalla colonna centrale, cioè la 4, e giocando perfettamente in seguito. Se, invece, un giocatore comincia nelle colonne 2, 3, 5 o 6, l'altro può arrivare ad un pareggio giocando perfettamente. In fine, se un giocatore comincia nelle colonne 1 o 7, il secondo può forzare una propria vittoria attraverso un gioco perfetto.

Nello specifico, Victor Allis ha scritto un programma per il gioco Forza 4 basato su queste osservazioni. Il programma, chiamato Victor, usa un approccio knowledge-based, piuttosto che la classica tree search e gioca seguendo un insieme di regole strategiche la cui correttezza è stata provata matematicamente.

Esistono altri programmi "forti" per il gioco del Forza 4, i quali impiegano, diversamente da Victor, delle combinazioni di ricerca nell'albero degli stati, euristiche di valutazione della situazione nella griglia di gioco e database di mosse pre-caricate.

Detto ciò, nel presente lavoro, abbiamo deciso di non seguire la strategia ottimale ma di intraprendere una strada ancora mai percorsa, applicando l'algoritmo del Simulated Annealing per massimizzare la funzione di valutazione della bontà di una mossa, la quale incorpora anche un'euristica di predizione della situazione futura conseguente alla mossa.

4 Massimizzazione con Annealing Simulato

E' noto che il Simulated Annealing viene impiegato generalmente per problemi di ottimizzazione combinatoriale: esempi classici sono la sua applicazione nel problema del commesso viaggiatore e in altri problemi di scheduling.

Nel nostro caso, invece, si potrebbe pensare che la tecnica di Annealing non sia adatta in quanto l'ottimizzazione non avviene una-tantum, ma deve essere eseguita regolarmente durante tutta la partita, in corrispondenza di ogni mossa del programma di gioco.

Di fatto però, proprio la natura combinatoriale dell'Annealing Simulato si adatta molto bene ad una investigazione, sebbene stocastica, dell'albero di ricerca.

L'idea è, infatti, di simulare un avanzamento del gioco per un certo numero di mosse e di eseguire l'ottimizzazione della funzione di valutazione calcolata sulla sequenza generata. In effetti, proprio la simulazione dell'avanzamento del gioco consente di creare una funzione euristica di predizione della bontà di una mossa, sia per il presente che per l'immediato futuro.

Più concretamente, si è pensato di generare una sequenza di 8 mosse casuali (4 per giocatore), eventualmente meno se si raggiunge un 4 o le caselle disponibili sono inferiori a 8, e su questa configurazione calcolare il valore della funzione di valutazione. E' bene puntualizzare che la configurazione generata comincia con una mossa del programma di gioco.

Per quanto riguarda la perturbazione della configurazione, invece, si è pensato di disfare ad ogni passo dell'algortimo di Annealing parte della configurazione generata (eventualmente tutta) provvedendo poi a rigenerare le mosse disfatte.

Ovviamente, di questa progressione nel futuro, la prima mossa risulta essere la più importante in quanto corrisponde alla mossa che effettivamente verrà eseguita dal programma di gioco, in caso di scelta.

Per il Forza 4 su matrice 6×7, i calcoli relativi alla funzione di valutazione richiedono poco tempo per cui si ha la libertà di impostare valori alti di temperatura e fattori di decremento vicini a 1, in modo da ottenere risultati molto prossimi all'ottimo in tempi che non compromettono la giocabilità.

Nello specifico, la temperatura iniziale viene impostata ad un valore 1000 mentre il fattore di decremento è 0.95.

Terminiamo la sezione ricordando che l'algoritmo di Annealing Simulato viene solitamente utilizzato nella minimizzazione di funzioni, quindi per il nostro lavoro, dovendo massimizzare la funzione di valutazione, considereremo una funzione a valori negativi.

5 Funzione di valutazione degli stati

E' stato già sottolineato che la nostra strategia per il gioco consiste nel massimizzare la funzione di valutazione, quindi è di fondamentale importanza che tale funzione sia la più accurata possibile: infatti, si è già avuto modo di dire che una funzione grossolana potrebbe indurre nell'algoritmo delle scelte che potrebbero portarlo al fallimento.

Una **funzione di valutazione** è definita come una funzione che restituisce una *stima* del guadagno atteso in una determinata posizione e per essere considerata **buona**:

- dovrebe ordinare gli stati terminali nello stesso modo della vera funzione di utilità (quella che può essere valutata percorrendo tutto l'albero di ricerca);
- il suo calcolo non dovrebbe richiedere troppo tempo;
- dovrebbe avere una forte correlazione con la probabilità reale di vincere la partita.

La maggior parte delle funzioni di valutazione, ragiona in base alle **caratteristiche** di uno stato del gioco: questo comporta che gli stati risultino raggruppati in classi di equivalenza, sulla base dei valori delle caratteristiche considerate. In genere, le funzioni di valutazione calcolano separatamente i valori di ogni caratteristica di uno stato combinandoli poi insieme per formare il valore finale.

Da un punto di vista matematico, una funzione del genere è detta **funzione lineare pesata**, ed è espressa come

$$Eval(s) = \sum_{i=1}^{n} w_i f_i(s)$$

con w_i peso della caratteristica i-esima, f_i caratteristica i-esima e s stato attuale.

Per il nostro scopo, la funzione di valutazione è leggermente più complessa in quanto si articola di due parti: una che stima la bontà *immediata* della mossa attuale e una che invece stima quella *futura*, in modo prettamente euristico. Le capacità di predizione della funzione si basano proprio sulla simulazione dell'avanzamento del gioco, di cui si è parlato nella sezione precedente.

La parte di funzione di valutazione che si occupa della stima della bontà immediata della mossa, si concentra solo sulla prima delle mosse generate: infatti, proprio questa sarà la mossa che verrà eventualmente eseguita.

Occorre tenere presente che la parte di stima della bontà immediata deve avere un peso maggiore nella funzione di valutazione poiché è inutile fare degli ottimi piani per il futuro se ci togliamo la possibilità di vincere già nel presente.

La seconda parte della funzione, invece, si concentra sulla situazione complessiva nella griglia di gioco al termine della simulazione dell'avanzamento del gioco.

Ricapitolando, la funzione di valutazione può essere così espressa

$$Eval(m, s) = present(first(m), s) + future(m, s)$$

dove m rappresenta la sequenza di mosse generate casualmente, s è il contenuto attuale della griglia, *present* è la funzione di valutazione della bontà immediata, $first$ è una funzione che restituisce la prima mossa della sequenza m e $future$ è la funzione euristica di valutazione della bontà futura della prima mossa della sequenza.

6 Bontà attuale di una mossa

Per quanto rigurda la valutazione della bontà attuale di una mossa, occorre stimare il pericolo ed il vantaggio della prima mossa nella sequenza generata dall'algoritmo di Annealing. A questo scopo, si è pensato di tenere traccia della situazione nella griglia tramite un insieme di vettori che quantificano il vantaggio ed il pericolo del piazzamento di un gettone in ciascuna colonna.

I vettori citati hanno, ovviamente, dimensione pari al numero di colonne della matrice di gioco e sono:

- `danger`;
- `advantage`;
- `next_danger`;
- `next_advantage`.

Scendendo nei dettagli, ciascun vettore esprime, rispettivamente, il pericolo e il vantaggio che l'algoritmo ha nel posizionare un gettone in una colonna ed il pericolo ed il vantaggio che avrebbe dopo il posizionamento di un gettone in una colonna.

E' importante sottolineare che tali vettori devono essere aggiornati ad ogni inserimento effettivo di un gettone e non nelle sequenze di previsione generate dall'Annealing.

Il pericolo ed il vantaggio sono calcolati allo stesso modo considerando rispettivamente il colore dell'avversario e quello del computer. In entrambi i casi, per ogni cella di inserimento si conta il numero di gettoni del colore specificato che sono raggiungibili sulla stessa riga, sulla stessa colonna e sulle due diagonali.

Chiaramente, per affinare la funzione di valutazione si devono scartare i conteggi che non comportano un vantaggio o pericolo effettivo.

In altre parole, per le righe andremo a considerare solo se nelle immediate vicinanze della cella considerata ci sono dei gettoni del colore specificato e se esiste la possibilità di allineare 4 gettoni. La Figura 4 mostra che nella colonna 3 il programma di gioco (gettoni gialli) ha vantaggio 0 poichè non ha possibilità di espansione né a destra né a sinistra.

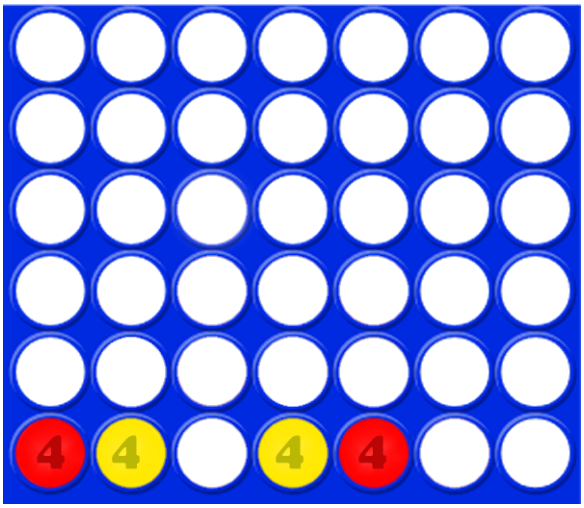

Figura 4: Situazione di vantaggio 0 nella riga

Analogamente, per le colonne ci preoccuperemo di considerare solo i casi in cui abbiamo dei gettoni del colore specificato nelle celle sottostanti ed esiste la possibilità di formare 4 sulla colonna. La Figura 5 mostra ancora una situazione di vantaggio 0 per il programma di gioco, sta volta nella colonna 4.

In fine, per le diagonali ci limiteremo a considerare solo quelle costituite da almeno 4 celle e per le quali non siano già state precluse le possibilità di fare 4. La Figura 6 mostra le diagonali che non vengono prese in considerazione dalla funzione di valutazione.

I vantaggi ed i pericoli di ogni colonna sono ottenuti come somma dei vantaggi e pericoli su riga, colonna e diagonali. In base a quanto detto appare ovvio che una situazione di tre gettoni dello stesso colore allineati o comunque prossimi tra loro deve avere un grosso risalto in quanto situazione di grande pericolo o vantaggio.

A tale scopo, se un conteggio su riga, colonna o diagonale genera un numero maggiore o uguale a 3, tale quantità verrà moltiplicata per 5 in modo da essere debitamente amplificata.

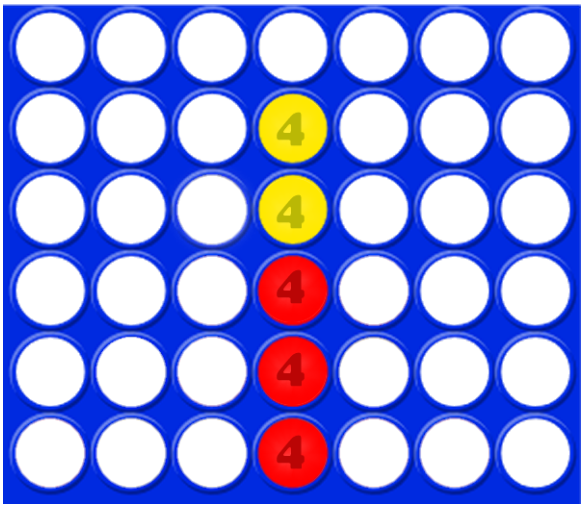

Figura 5: Situazione di vantaggio 0 nella colonna

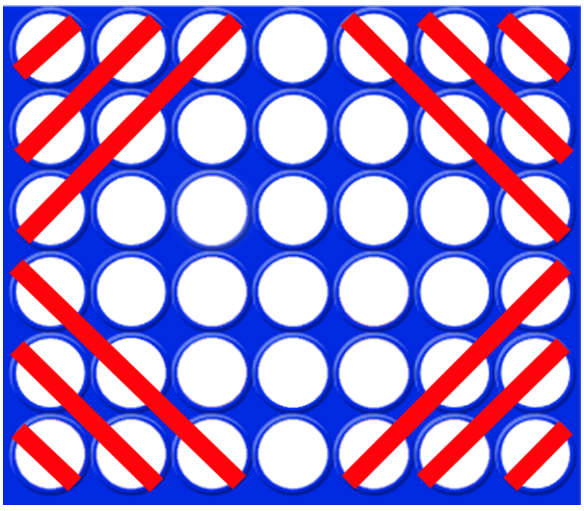

Figura 6: Diagonali non considerate dalla funzione di valutazione

Meritano particolare attenzione i casi critici, cioè quelli di tris sia per il programma di gioco che per il giocatore e quelli in cui il vantaggio o il pericolo futuro sono più alti degli attuali.

In questi casi, infatti, la funzione deve essere opportunamente "drogata" facendole assumere valori o molto bassi o molto alti in modo da essere sicuri che la mossa venga sicuramente accettata o sicuramente rifiutata.

Riassumendo, la funzione di valutazione della bontà immediata della mossa non fa altro che considerare la prima delle mosse simulate nel futuro e restituire la somma del vantaggio e del pericolo moltiplicati entrambi per una opportuna costante, cioè formalmente possiamo scrivere

$$present(first(m), s) = advantage[first(m)] \times c_a + danger[first(m)] \times c_d.$$

Sperimentalmente, è stata osservata una buona risposta dell'algoritmo per i seguenti valori delle costanti $c_a = -2200$ e $c_d = -2000$. Il valore delle costanti è leggermente diverso in modo da prediligere la difesa piuttosto che l'attacco.

I casi urgenti in cui la funzione deve essere "drogata" sono riportati di seguito

- $present(first(m), s) = -50000$ nel caso in cui
  - $advantage[first(m)] \geq 15$
- $present(first(m), s) = 50000$ nel caso in cui
  - $next\_danger[first(m)] \geq danger[first(m)]$ e
  - $next\_danger[first(m)] \geq advantage[first(m)]$ e
  - $next\_danger[first(m)] \geq 15$
- $present(first(m), s) = 20000$ nel caso in cui
  - $next\_advantage[first(m)] \geq advantage[first(m)]$ e
  - $next\_advantage[first(m)] \geq danger[first(m)]$ e
  - $next\_advantage[first(m)] \geq 15$ e
  - $danger[first(m)] < 15$.

Senza entrare nei dettagli, notiamo che la funzione restituisce un valore positivo molto grande in modulo nei casi in cui fare una mossa ci farebbe perdere un vantaggio o ci porterebbe in una situazione di estremo pericolo. In questi casi, infatti, il programma cercherà di evitare per quanto più possibile di fare la mossa aspettando di forzare l'avversario a farla al suo posto. In questo modo viene relizzato un controllo sullo zugzwang.

D'altro canto, la funzione restituisce un valore negativo molto grande in modulo nei casi in cui si hanno 3 gettoni allineati ed il programma deve essere forzato a chiudere la partita.

7 Bontà futura di una mossa

La parte della funzione di valutazione descritta nella precedente sezione non rende il programma di gioco in grado di prevedere quali saranno le conseguenze future di una sua scelta, se non per la mossa immediatamente successiva a quella attuale.

Tuttavia, risulta di prominente importanza che il programma sia in grado di vedere oltre la mossa successiva alla attuale affinché possa evitare che un piccolo vantaggio immediato comprometta l'intero esito della partita.

Al fine di dare questa sorta di "visione sul futuro", viene eseguita la simulazione dell'avanzamento del gioco. La funzione euristica che si occupa di calcolare la bontà nel futuro di una mossa si concentra sulla situazione nella matrice di gioco al termine dell'avanzamento simulato e opera contando la quantità di 1, 2, 3 e 4 (o eventualmente più di 4) gettoni allineati e dello stesso colore.

Ovviamente, anche in questo caso, sono considerati nel conteggio solo gli allineamenti che hanno una reale possibilità di diventare 4. La funzione euristica creata può essere formalmente espressa come

$$future(m, s) = \sum_{i=1}^{4} f_i(m, s)c_i$$

dove m è la configurazione generata, s è la situazione effettiva nella griglia, f_i restituisce il numero di allineamenti di i gettoni con possibilità di espansione e c_i è una costante di peso.

Sperimentalmente, le costanti di peso sono state fissate come $c_1 = -10$, $c_2 = -100$, $c_3 = -3000$ e $c_4 = -4000$.

8 Implementazione

L'algoritmo descritto in precedenza è stato implementato nel linguaggio C++ seguendo il paradigma ad oggetti, in particolare sono state scritte le seguenti classi:

- `Vector`;
- `Matrix`;
- `RandomGen`;
- `Forza4`.

Precisiamo che tutto il codice è stato scritto da scratch e, in particolare, le classi `Vector` e `Matrix` sono due classi template che realizzano dei "contenitori"

generici di tipo vettore e matrice, consentendo la numerazione delle componenti partendo dall'indice 1.

La classe `RandomGen` crea, invece, un oggetto generatore di numeri pseudo-casuali nell'intervallo $(0, 1)$.

In fine, la classe `Forza4` realizza l'algoritmo di gioco vero e proprio. La parte decisionale che utilizza l'Annealing Simulato, si nasconde dietro il metodo pubblico `genColSA()` che restituisce un numero corrispondente alla colonna scelta dal programma per l'inserimento.

L'inserimento vero e proprio di un gettone, sia per l'utente che per il programma, è demandato al metodo pubblico `insertToken(int token, int col)` che richiede in input il colore del gettone da inserire e la colonna di inserimento, la quale è ottenuta o tramite la specifica dell'utente o tramite il metodo `genColSA()`.

9 Interfaccia Grafica

L'interfaccia grafica del gioco è stata sviluppata con le librerie multipiattaforma e di libera distribuzione Qt. Essa è molto intuitiva e minimale e consta solamente della griglia di gioco e di un timer che indica il tempo della partita, come mostrato in Figura 7.

L'interfaccia grafica ha richiesto la scrittura di un'ulteriore classe in C++ per la realizzazione della finestra di gioco, cioè la classe `ForSA4`.

All'avvio, viene mostrato uno splashscreen con il nome del programma, dopo di che lo splashscreen lascia posto alla schermata di gioco.

Una volta nella schermata di gioco, l'utente può eseguire la sua mossa, semplicemente clickando con il mouse in corrispondenza della colonna dove inserire il gettone. Per default, la prima partita dopo l'avvio viene sempre cominciata dal giocatore e ad ogni sua mossa ne segue una del programma di gioco.

E' stato scelto il colore rosso per i gettoni del giocatore e giallo per quelli del programma.

La partita prosegue con l'alternarsi di mosse finché uno dei due giocatori allinea 4 gettoni o non ci sono più celle libere. Al termine di ogni partita viene mostrato un dialog con l'esito del gioco e due pulsanti con cui l'utente può scegliere se cominciare o no una nuova partita. La Figura 8 mostra il dialog di esito partita.

In caso di vittoria di uno dei due giocatori, la partita successiva verrà iniziata dal giocatore vincitore mentre, in caso di pareggio, inizierà l'ultimo ad aver giocato.

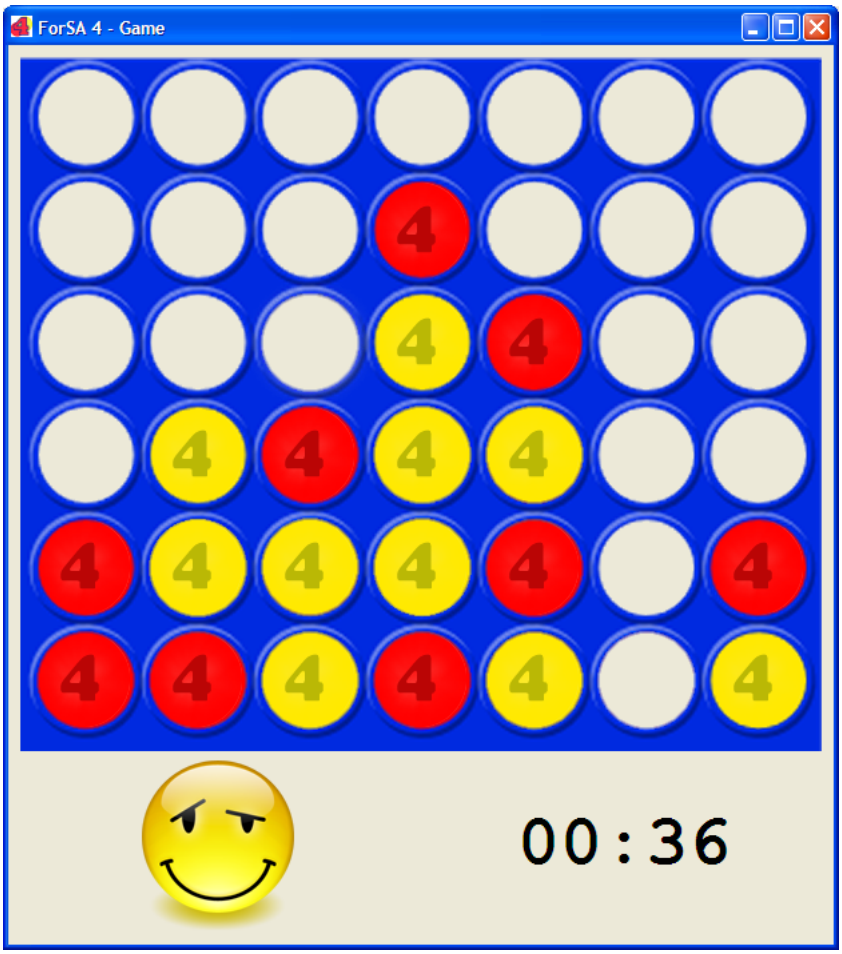

Figura 7: Interfaccia grafica del gioco ForSA4

Figura 8: Dialog di esito partita in caso di Game Over

10 Conclusioni

Il gioco creato offre una buona giocabilità ed è caratterizato da tempi di risposta ragionevoli. La strategia si comporta, in linea di massima, abbastanza bene anche se conserva una certa "cecità" nelle previsioni a lungo termine. In effetti, l'algoritmo ha una visibilità di sole 4 mosse avanti e sceglie la migliore mossa su questa conoscenza.

La decisione di limitare il lookahead a 4 mosse è stata presa perchè 4 è circa il numero di mosse che un giocatore discreto riesce a prevedere, ed inoltre ciò non ha un peso eccessivo sul tempo di calcolo, e quindi di risposta, del gioco.

Anche se, di fatto, è stata dimostrata la possibilità di creare un programma "perfetto" per il gioco del Forza 4, altri algoritmi, come quello discusso nel presente lavoro, offrono una interessante esperienza di gioco. Questa ultima osservazione non deve essere trascurata nello sviluppo di giochi in quanto, nella maggior parte dei casi, i giocatori vogliono vincere i giochi in cui si cimentano.

Quindi, se il programma di gioco si basasse su un algoritmo perfetto che lo facesse vincere sempre, il gioco soffrirebbe di uno scarso interesse da parte del pubblico.

Bibliografia e Sitografia

[1] **V. Allis**, *A Knowledge-based Approach of Connect-Four.*
Department of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, The Netherlands.

[2] **S. Russel**, **P. Norvig**, *Intelligenza Artificiale - Un approccio moderno*, Volume 1, Seconda Edizione.
Pearson - Prentice Hall.

[3] **H. Schildt**, *C++ - La guida completa*, Quarta Edizione.
Mc Graw Hill.