"AxelBrain", un progetto di Alex Rossi mat:0001089916;
CST course a.y. 22/23, Alma Mater Studiorum, University of Bologna, Italy.

-Informazioni generali:
"AxelBrain" è un'intelligenza artificiale sviluppata in JAVA per il gioco *Connect X*, una versione generalizzata di *Connect 4* comunemente chiamato in Italia "*Forza 4*", che ha come obiettivo quello di allineare per primo X pezzi al fine di vincere la partita.

-Informazioni riguardanti scelte progettuali:
Per lo sviluppo di AxelBrain è stato scelto l'utilizzo di un algoritmo MinMax con Alpha Beta pruning.
Sono presenti però degli accorgimenti, che verranno elencati in seguito, necessari all'ottimizzazione ed una migliore performance contro un avversario sia umano che non.
Generalmente il programma funziona richiamando il metodo *SelectColumn()* il quale inizia a scorrere ed analizzare ogni colonna data dalla lista di colonne disponibili, grazie al metodo *getAvaibleColumns()*, e fa partire per ognuna un'analisi specifica in un tempo prestabilito, attraverso il metodo *findBestMove()*, per trovare quella migliore da giocare alla prossima mossa al fine di vincere o cercare di non perdere.
FindBestMove() implementa come sopracitato l'algoritmo MinMax con ottimizzazione Alpha Beta pruning, ciò permette un'analisi fino ad una determinata profondità, in questa implementazione è data da *MAX_BRANCHING*, che indica il numero di possibili mosse che potranno essere compiute durante lo svolgimento della partita. Ad ogni ramo il branching factor viene diminuito di 1 e si controlla se la partita è ancora in corso, ovvero se non si è raggiunto uno stato di *Win*, *Lose* o *Draw*, oppure se si è raggiunto un nodo terminale, in tal caso si effettua una valutazione euristica della board attuale con il metodo *evaluation()*.
La valutazione della board avviene facendo una sottrazione tra il numero di pezzi consecutivi del player, ovvero l'AI, e quelli dell'avversario. Ciò viene effettuato per ogni riga, colonna, diagonale positiva e diagonale negativa della matrice; inoltre viene anche attribuito un punteggio maggiore per le colonne definite 'centrali' poiché permettono più facilmente di sviluppare strategie vincenti.
È presente inoltre una ricalibrazione delle colonne considerate centrali per le board più grandi derivanti da un *Hit-Rate* logico dato da quante volte una serie contigua di punti necessari per vincere 'passa' per quella colonna (per una comprensione migliore guardare la rappresentazione grafica nel file '*Connectx Hit-Rate Count'*).

-Informazioni riguardanti ottimizzazioni varie:
1) *LinkedHashMap*: l'utilizzo di questa struttura dati è dovuta dalla necessità di accorciare i tempi decisionali del programma.
   Il suo funzionamento è analogo a quello di una classica HashTable, che permette la ricerca di uno specifico elemento in un tempo medio di O(1), a differenza però di quest'ultima permette di avere come valori di base *NULL* e tenere traccia degli elementi memorizzati nell'ordine in cui vengono proposti.
   Grazie a quest'ultima feature si permette di controllare la grandezza della memoria impiegata per l'HashMap ed eventualmente eliminare gli elementi più vecchi poiché futili nella ricerca di soluzioni ottimali in stati di gioco avanzati.
2) *Try&catch*: l'utilizzo di questa funzionalità è dovuto dalla necessità di cercare una risposta senza eccedere limiti di tempo che altrimenti creerebbe errore dando partita

persa a tavolino. Con essa si permette quindi di restituire la risposta ottimale ottenuta fino al momento della fine del tempo prestabilito.
Il tempo a disposizione è gestito dal metodo *checktime()* e permette di 'lanciare il catch' quando è passato circa il 99% del tempo decisionale a disposizione.

3) *MAX_BRANCHING dinamico*: si calibra in base al tempo a disposizione per ogni mossa (in base ai test svolti si è stabilito che il *MAX_BRANCHING* ottimale per time limit di 1 sec. è 6, per 3 sec. è 7 e per 10 sec. è 8).

-Informazioni riguardanti test effettuati contro AI fornite:
(Tutti i test sono stati svolti imponendo come time-limit per ogni mossa di 1 sec. e svolgendo 100 partite, tranne per le *Large boards* che ne sono state svolte solo 5; W=win against L0 or L1 AI, L=loss …, D=draw …; I test sono stati svolti su un processore Intel® Core™ i7-8750H CPU @ 2.20GHz × 12; i valori …/… NON sono frazioni bensì rappresentano: AxelBrain Going 1/ AxelBrain Going 2).

Rows	**Col.**	**ToWin**	**W (L0)**	**L (L0)**	**D (L0)**	**W (L1)**	**L (L1)**	**D (L1)**
4	4	4	83/72	0/0	17/28	47/63	3/0	50/37
5	4	4	90/93	0/1	10/6	75/59	3/0	22/41
6	4	4	98/98	0/1	2/1	72/63	7/4	21/33
7	4	4	97/98	3/1	0/1	82/81	6/2	12/17
5	5	4	100/100	0/0	0/0	94/89	0/1	6/10
6	5	4	98/100	2/0	0/0	83/94	5/1	12/5
7	5	4	100/100	0/0	0/0	96/92	2/3	2/5
4	6	4	99/97	0/0	1/3	81/98	3/1	16/1
5	6	4	100/100	0/0	0/0	100/95	0/0	0/5
6	6	4	100/100	0/0	0/0	94/99	4/0	2/1
7	6	4	100/100	0/0	0/0	99/99	0/0	1/1
4	7	4	100/100	0/0	0/0	97/99	0/1	3/0
5	7	4	100/100	0/0	0/0	100/100	0/0	0/0
6	7	4	100/100	0/0	0/0	97/96	1/4	2/0
7	7	4	100/100	0/0	0/0	100/100	0/0	0/0
5	4	5	13/0	0/5	87/95	0/0	0/0	100/100
6	4	5	13/12	0/0	87/88	0/0	0/0	100/100
7	4	5	24/18	0/0	76/82	0/0	0/0	100/100
4	5	5	25/40	1/0	74/60	1/28	20/0	79/72

5	5	5	75/68	0/0	25/32	52/39	0/1	48/60
6	5	5	85/81	0/1	15/18	61/60	0/0	39/40
7	5	5	95/85	0/0	5/15	58/62	1/2	41/36
4	6	5	66/52	0/0	34/48	18/44	1/0	81/56
5	6	5	84/91	0/0	16/9	58/65	1/0	41/35
6	6	5	98/96	1/0	1/4	66/81	6/0	28/19
7	6	5	95/98	0/2	5/0	69/85	1/0	30/15
4	7	5	76/81	1/0	23/19	26/79	0/0	74/21
5	7	5	96/94	0/1	4/5	84/63	0/6	16/31
6	7	5	99/99	0/0	1/1	80/90	2/1	18/9
7	7	5	100/99	0/0	0/1	95/83	0/4	5/13
20	20	10	N/A	N/A	N/A	2/2	0/0	3/3
30	30	10	N/A	N/A	N/A	N/A	N/A	N/A
40	40	10	N/A	N/A	N/A	N/A	N/A	N/A
50	50	10	N/A	N/A	N/A	N/A	N/A	N/A

-Osservazioni in merito ai risultati ottenuti:

In base ai risultati delle partite giocate contro le AI fornite, L0 (mossa totalmente casuale) e L1 (mosse che bloccano l'imminente vittoria dell'avversario, mosse che danno vittoria nel breve futuro o mosse puramente casuali), si può notare come la generale percentuale di vittoria si aggiri intorno a circa il 90% avendo picchi fino al 100% (la percentuali di questi picchi aumenta specialmente se si inizializza per più volte consecutive partite da 100 round). Ci sono però alcuni casi particolari:

1) Le partite dove il ToWin corrisponde al numero di colonne e/o al numero di righe, come nel caso 5x4 5, e quindi per la conformazione stessa della *board* si ha un'alta probabilità di pareggio (in alcuni casi raggiunge il 100%!);
2) Le partite derivanti dalle *Large boards*, ovvero date da grigie maggiori o uguali a 20x20, dove, seppure non si siano riusciti a svolgere un numero sufficiente di partite per permettere un'analisi completa delle performance, si può notare come la percentuale di efficienza cali drasticamente andando perlopiù a cercare il *DRAW*.;
3) Le partite dove si ha più probabilità di vittoria se si parte per secondo, ad esempio in 4x7 5 la percentuale di vittoria (contro L1) andando per primo è 26% mentre 79% andando per secondo.

Generalmente però si può affermare con certezza che ha capacità sicuramente superiori nell'analisi della mossa migliore rispetto alla mente umana, specialmente per le *board* più contenute e standard come 6x7 4.

-Costi in termini di tempo e memoria:

Generalmente il costo in termini di memoria è dato da quanto spazio occupa l'immagazzinare le varie mosse da analizzare nella HashTable, che però nel caso pessimo è 2gb di memoria poiché è il limite fissato in fase di progettazione del programma.

Si può comunque fare un calcolo approssimativo del numero di mosse analizzate e memorizzate per scelta della colonna ottimale:

> Dato nCd = numero colonne disponibili, mBf = MAX_BRANCHING_FACTOR, N = numero colonne matrice;
>
> O(nCd ^ mbf x N).

Il costo invece in termini di tempo dipende in questa implementazione specialmente se la mossa ottimale è già stata calcolata e memorizzata nella HashTable (caso ottimale dove il tempo richiesto sarebbe O(1)) oppure se bisogna calcolare la mossa per la prima volta (caso pessimo).

Il calcolo dei costi è molto complicato e servirebbero studi e test più approfonditi per affermarlo con certezza quindi rimando al costo generale attribuito a questo tipo di algoritmo: O(n).

-Possibili esempi di miglioramenti per rendere l'AI più forte:

1) Utilizzare l'algoritmo *Iterative Deepening*: in quanto a parità di tempo impiegato analizza più situazioni di gioco (molte delle quali si ripetono);
2) Valutazione euristica più accurata;
3) Ottimizzazione migliore e più efficiente per le *Large boards*;
4) Gestione migliore del *MAX_BRANCHING dinamico* per quantità di tempo maggiori di 10 e minori di 1 secondi.

-References:

Pascal Pons, *Solving connect4: how to build a perfect ai*, http://blog.gamesolver.org/solving-connect-four/01-introduction/, 2016;

John Tromp, *John's Connect Four Playground,* https://tromp.github.io/c4/c4.html;

Chess Programming WIKI, Alpha-Beta, https://www.chessprogramming.org/Alpha-Beta;

Wikipedia, https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning.