

# Progetto Algoritmi a.s. 2021/2022

## Team

Xuanqiang Huang # TODO: matricola

Giovanni Spadaccini # TODO: matricola

## Riassunto

**first take** La nostra implementazione di un AI per il gioco di MNKGame utilizza un algoritmo di Minimax euristico con alpha-beta pruning. L'algoritmo utilizza l'euristica di valutazione per scoprire l'ordine di esplorazione di un numero limitato di nodi, le esplora fino a un livello di profondità prefissato, dopo il quale ritorna un valore euristico

**second take** Abbiamo utilizzato di un algoritmo di Minimax euristico con alpha-beta pruning per la risoluzione di un gioco di tris generalizzato. L'algoritmo utilizza l'euristica di valutazione per scoprire l'ordine di esplorazione di un numero limitato di nodi, le esplora fino a un livello di profondità prefissato, dopo il quale ritorna un valore euristico. Dai test fatti in locale, l'algoritmo sembra avere capacità simili o superiori a quelle di un umane per le tavole di dimensioni umane (ossia da 10 in giù).

**note a caso** Il gioco è stato implementato in linguaggio Java, scelta che rendeva difficile la gestione della memoria a basso livello per la presenza di un garbage collector.

## Introduzione

Il gioco proposto è una versione generalizzata del gioco (nought and crosses) conosciuto più generalmente in occidente come tris, o gomoku in Giappone: un gioco a due giocatori su una tavola simile a quanto in immagine, a turni in cui bisogna allineare un certo numero di pedine del proprio giocatore al fine di vincere.

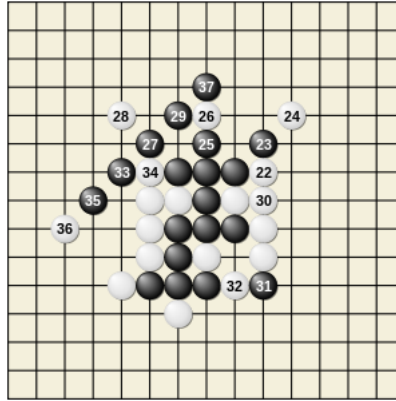


Figure 1: Esempio di tavola di gioco di gomoku in cui il giocatore nero ha vinto

Seguendo la caratterizzazione di un ambiente di gioco di Russel e Norvig<sup>1</sup> possiamo definire il gioco come un ambiente deterministico, multigiocatore, con informazioni complete, sequenziale, statico, conosciuto e discreto. Questa descrizione ci ha permesso di avere una prima idea di quali algoritmi potessero essere utilizzati per la risoluzione del gioco, in quanto in letteratura questo problema, e problemi simili, sono stati risolti con tecniche che hanno sopravvissuto allo scorrere del tempo.

Sotto questa logica abbiamo scelto l'implementazione di un minimax euristico.

## MarkCell e unmarkCell

Abbiamo ideato un sistema che è in grado di fare `markCell`, `unmarkCell` in tempo costante in caso ottimo, pessimo e medio, senza considerare i checks aggiuntivi per verificare lo stato del gioco e l'aggiornamento dell'euristica.

Al fine di raggiungere questa velocità, l'insieme delle mosse eseguite è tenuto alla fine di un array che contiene tutte le mosse, in modo simile a quanto fa una heap studiata nel corso al momento di rimozione. Queste mosse sono tenute come se fossero uno stack, e possono essere ripristinate con semplici assegnamenti. Questa implementazione migliora rispetto alla board provvista nel caso pessimo, in quanto non deve più necessitare di un hashtable, il cui caso pessimo è  $O(n)$  con  $n$  la grandezza della table.

**Nota:** tutte le celle contengono un *index* che indica la posizione dell'array in cui è contenuta la mossa se questa non è ancora stata rimossa, altrimenti, indica la posizione di ritorno.

---

**Algorithm 1:** markCell senza checks sulla board

---

**Result:** Cella richiesta della tavola è marcata**Input** : int freeCellsCount: numero di celle libere, è compreso fra 1 e allCells.length**Input** : int index: l'index della cella da marcare, compresa fra 0 e freeCellsCount**Input** : Cell[] allCells: array tutte le celle, in cui le prime freeCellsCount sono considerate libere**Output:** void: viene modificata allCells

```
1 allCells[freeCellsCount - 1].index = allCells[index].index
2 swap(allCells[freeCellsCount - 1], allCells[index])
  // marca la cella come occupata dal giocatore
3 mark(allCells[freeCellsCount - 1])
4 freeCellsCount = freeCellsCount - 1
```

---

---

**Algorithm 2:** markCell senza checks sulla board

---

**Result:** Viene rimosso l'ultima mossa della tavola**Input** : int freeCellsCount: numero di celle libere, è compreso fra 0 e allCells.length - 1**Input** : Cell[] allCells: array tutte le celle, in cui le prime freeCellsCount sono considerate libere

```
  // marca la cella come libera
1 markFree(allCells[freeCellsCount - 1])
2 swap(allCells[allCells[freeCellsCount].index], allCells[freeCellsCount])
3 allCells[freeCellsCount].index = freeCellsCount
4 freeCellsCount = freeCellsCount + 1
```

---

## TODOS

Alcune cose importanti che si dovrebbero fare?

- ☒ pseudocodice di markCell
- ☒ pseudocodice di unmarkCell
- ☒ spiegazione delle FreeCell
- ☐ Spiegazione dell'euristica
  - ☐ 1. Spiegazione dell'ordering delle mosse -> cenno a Late Move reduction (o citazione del paper di MICS)
  - ☐ 2. Spiegazione dell'eval della board
  - ☐ 3. Note: sul pruning utile grazie a questo ordering
  - ☐ 4.
    - [ ]
- ☐ algoritmo di sorting delle celle
- ☐ spiegazione del timer test (numero dei nodi cercati)
- ☐ spiegazione dei valori euristici per l'esplorazione in depth e in weight

## Algorithm 1

Just a sample algorithm

---

**Algorithm 3:** While loop with If/Else condition

---

**Result:** Write here the result

**Input :** Write here the input

**Output:** Write here the output

```
1 while While condition do  
2   | instructions  
3   | if condition then  
4   | | instructions1  
5   | | instructions2  
6   | else  
7   | | instructions3  
8   | end  
9 end
```

---

## Approcci fallimentari

1. Simulazione di Montecarlo (MCTS), dato il grande successo di AlphaGo
  1. Guardava celle che avevano poco valore per la vittoria
  2. Guardava tutti gli stati ad ogni livello, il che pesava molto anche sulla memoria (poichè si teneva tutto il game tree)
  3. Il limite di tempo era troppo basso per avere un numero di simulazioni sufficienti
  4. La capacità dell'hardware incideva molto sui risultati.
2. Puro algoritmo euristico (anche conosciuto come Greedy best first Search)
  1. Non riusciva ad andare in profondità, dato che selezionava ogni volta la cella con maggiori probabilità di vittoria al primo livello, questo non gli permetteva di pianificare le proprie mosse.
3. alpha beta pruning puro, per i test grandi impiegava troppo tempo d'esecuzione, non riuscendo a fare l'eval di neanche una mossa

## References

1. Russell, Stuart J., e Peter Norvig. Artificial Intelligence: A Modern Approach. Fourth edition, Global edition, Pearson, 2022. Chapt. 2