



# Adaptive Bitboard Engine

## Documentazione di Progetto - Versione 1.1

### Informazioni Generali

<b>Corso di Laurea:</b>	Informatica
<b>Università:</b>	Università degli Studi di Salerno (UNISA)
<b>Docente:</b>	Chiar.mo Prof. PALOMBA
<b>Data:</b>	05/02/2026
<b>Anno Accademico:</b>	2025/2026

## Membri Del Gruppo

**MANFREDINI Umberto** Matricola 0512119797

**ROMANO Pino Fiorello** Matricola 0512120259

## Revision History

Data	Versione	Descrizione	Autore
30/01/2026	1.0	Capitolo 1 (Introduzione e Visione)	Gruppo
05/02/2026	1.1	Capitolo 2 (Architettura) e 3 (Modeling)	Gruppo

# Indice

<b>1 Introduzione e Visione Progettuale</b>	<b>4</b>
1.1 Abstract . . . . .	4
1.2 Limiti della Teoria dei Giochi Classica . . . . .	4
1.3 Obiettivo del Progetto . . . . .	5
1.4 Paradigma di Apprendimento . . . . .	5
<b>2 Architettura del Sistema e Scelte Tecnologiche</b>	<b>6</b>
2.1 Pattern Architettonico: MVC . . . . .	6
2.2 Core Tecnologico: Bitboard Engine . . . . .	6
2.3 Persistenza e Integrità dei Dati . . . . .	7
2.4 Modularità e Design Patterns . . . . .	7
<b>3 Modellazione Algoritmica e Ottimizzazione</b>	<b>8</b>
3.1 L'Algoritmo Base: Minimax e Alpha-Beta Pruning . . . . .	8
3.2 Ottimizzazioni Implementate . . . . .	8
3.2.1 Bit-Level Optimization (Hardware Counting) . . . . .	8
3.2.2 Move Ordering (Center-First Heuristic) . . . . .	9
3.2.3 Transposition Tables (Memoization) . . . . .	9
3.3 Euristica Avanzata e Anti-Blunder . . . . .	9
3.3.1 Riconoscimento "Fork" (Double Threat) . . . . .	9
3.3.2 Anti-Blunder Check (Zugzwang) . . . . .	10
3.4 Adattamento Dinamico (Profilazione) . . . . .	10
<b>Bibliografia</b>	<b>11</b>

# 1. Introduzione e Visione Progettuale

## 1.1. Abstract

Il gioco del Forza 4 (*Connect Four*) rappresenta, nel campo dell’Intelligenza Artificiale, un problema di ricerca spaziale a informazione perfetta risolto matematicamente. Sebbene esistano algoritmi in grado di giocare perfettamente, le implementazioni tradizionali basate esclusivamente sulla Teoria dei Giochi classica tendono a produrre agenti “statici”, incapaci di adattarsi alla variabilità dello stile umano.

Il presente progetto propone lo sviluppo di un motore decisionale ibrido, denominato **Adaptive Bitboard Engine**. Il sistema integra un algoritmo di ricerca *Minimax* con *Alpha-Beta Pruning* ad alte prestazioni — basato su rappresentazione *Bitboard* per massimizzare l’efficienza computazionale [2] — con un innovativo modulo di **Profilazione Comportamentale**. L’obiettivo è superare il concetto di “mossa ottimale assoluta” per approdare alla “mossa ottimale relativa”, massimizzando la probabilità di vittoria attraverso lo sfruttamento dei bias cognitivi dell’avversario.

## 1.2. Limiti della Teoria dei Giochi Classica

Nella sua formulazione classica, un agente basato su *Minimax* opera sotto l’assunto di razionalità perfetta: si presuppone che l’avversario (il giocatore *Minimizing*) esegua sempre la contromossa migliore possibile. Come formalizzato da Knuth e Moore [1], la funzione di valore  $V(s)$  per uno stato  $s$  è definita ricorsivamente come:

$$V(s) = \max_{a \in A} \left( \min_{b \in B} f(s') \right) \quad (1)$$

Sebbene l’ottimizzazione tramite *Alpha-Beta Pruning* permetta di ridurre significativamente lo spazio di ricerca [1], questo approccio presenta limiti intrinseci nell’interazione uomo-macchina:

- **Rigidità Strategica:** L’agente ignora le opportunità derivanti da errori non forzati dell’avversario se questi non rientrano nel suo orizzonte di calcolo (*Horizon Effect*), limitandosi a parare minacce che un giocatore umano potrebbe non aver visto.
- **Mancanza di Personalizzazione:** L’agente non distingue tra un esperto e un novizio. Come evidenziato da Wang et al. [3], l’efficacia di un’euristica dipende fortemente dal contesto della partita; un’euristica statica non può capitalizzare su errori sistematici (pattern ricorrenti) dell’avversario.

### 1.3. Obiettivo del Progetto

L'obiettivo primario è la realizzazione di un agente intelligente capace di evolvere la propria strategia in *Real-Time*. Il sistema non si limita a calcolare la mossa migliore sulla scacchiera, ma osserva il comportamento dell'utente per costruire un **Profilo Psicologico** dinamico, ispirandosi ai principi di modellazione dell'avversario teorizzati da Spronck et al. [4].

Se il sistema rileva che l'avversario soffre di “cecità diagonale” (incapacità di scorgere minacce su linee oblique), l'algoritmo di valutazione euristica aumenta dinamicamente il peso specifico dei pattern diagonali. Questa architettura sposta il focus dalla pura potenza di calcolo alla flessibilità cognitiva, creando un'esperienza di gioco più sfidante e organica.

### 1.4. Paradigma di Apprendimento

Il paradigma adottato si distacca dalle moderne tecniche di *Deep Reinforcement Learning* ("Black Box") per abbracciare un approccio definito **Apprendimento Euristic-Deterministico** ("White Box"). Invece di addestrare una rete neurale opaca su milioni di iterazioni generiche, il sistema è progettato per adattare esplicitamente i parametri della propria funzione di valutazione in risposta agli stimoli diretti dell'avversario corrente [5].

Questa scelta garantisce due vantaggi fondamentali per il contesto accademico e applicativo:

- **Trasparenza Decisionale:** Ogni scelta dell'IA è tracciabile e giustificabile. È possibile in ogni momento interrogare il sistema per comprendere *perché* ha preferito una mossa rischiosa (es. “Perché ho rilevato che l'avversario ignora le diagonali”).
- **Adattamento Immediato:** A differenza dei modelli statistici che richiedono grandi quantità di dati per convergere, l'approccio euristico permette all'agente di modificare la propria strategia già dopo poche mosse significative, offrendo un feedback immediato all'utente.

## 2. Architettura del Sistema e Scelte Tecnologiche

### 2.1. Pattern Architetturale: MVC

Per garantire la manutenibilità del codice e la separazione delle responsabilità, il sistema è stato progettato seguendo il pattern architetturale **Model-View-Controller (MVC)**. Questa struttura permette di isolare la logica computazionale dell'intelligenza artificiale dalla gestione dell'interfaccia utente, facilitando lo sviluppo parallelo e il testing dei singoli componenti.

- **Model (src/board/engine.py):** Rappresenta il nucleo del sistema. Include la rappresentazione dello stato del gioco tramite Bitboard, l'algoritmo Minimax e il modulo di profilazione. È responsabile di tutta la logica decisionale e delle regole di validazione.
- **View (src/board/interface.py):** Gestisce esclusivamente la rappresentazione grafica utilizzando la libreria Pygame. Si occupa del rendering della griglia, delle animazioni e della visualizzazione delle metriche di debug (es. bias correnti), senza possedere alcuna logica di gioco.
- **Controller (src/board/controller.py):** Agisce da intermediario, intercettando gli input dell'utente e orchestrando il flusso di gioco. Sincronizza lo stato del Model con l'aggiornamento della View e gestisce l'alternanza dei turni e il ciclo di vita della partita.

### 2.2. Core Tecnologico: Bitboard Engine

La scelta più critica per le prestazioni di un motore di ricerca in uno spazio degli stati vasto come quello del Forza 4 ( $4.5 \times 10^{12}$  posizioni possibili [6]) ricade sulla struttura dati. Abbiamo abbandonato la rappresentazione classica a matrice bidimensionale ( $6 \times 7$ ) in favore delle **Bitboard**.

Ogni stato del gioco è rappresentato da due interi a 64 bit (`uint64`):

- **position:** Una maschera che indica tutte le celle occupate (da entrambi i giocatori).
- **mask:** Una maschera che indica le celle occupate solo dal giocatore corrente.

Questa rappresentazione permette di verificare le condizioni di vittoria e le minacce non tramite costosi cicli iterativi, ma attraverso operazioni bitwise ( $O(1)$ ) supportate nativamente dalla CPU. Ad esempio, per rilevare un allineamento orizzontale, è sufficiente un'operazione di *shift* e *AND* logico:

```
m = position & (position >> 7)
if (m & (m >> 14)): return True // Vittoria rilevata
```

Questo approccio riduce i tempi di calcolo di ordini di grandezza rispetto all'uso di array, abilitando una profondità di ricerca maggiore per l'algoritmo Minimax.

## 2.3. Persistenza e Integrità dei Dati

Per supportare il modulo di apprendimento incrementale, è stato necessario implementare un sistema di persistenza robusto. A differenza di soluzioni basate su file JSON o CSV, che soffrono di problemi di concorrenza e integrità in caso di chiusura imprevista, è stato adottato **SQLite**.

L'utilizzo di un database relazionale embedded garantisce le proprietà necessarie per le sessioni di training massivo ("Headless"). Ogni partita conclusa viene salvata come una transazione atomica contenente:

- Metadati della sessione (Timestamp, ID Avversario).
- Esito della partita e numero di mosse.
- Snapshot dei bias cognitivi appresi dal Profiler.

Questo permette di interrompere e riprendere l'addestramento senza perdita di informazioni e facilita l'analisi statistica dei risultati tramite query SQL complesse.

## 2.4. Modularità e Design Patterns

Il sistema AI è stato progettato per essere estensibile senza modifiche al codice sorgente del motore di ricerca.

L'algoritmo Minimax è implementato come un motore generico che accetta in input un oggetto **Evaluator**. Questo ci ha permesso di implementare una strategia flessibile:

- **Training Bots:** Utilizzano evaluator statici con "difetti" programmati (es. `DiagonalBlinderEvaluator`).
- **Adaptive AI:** Utilizza l'`AdaptiveEvaluator`, che interroga il Profiler in tempo reale.

Tale architettura permette di creare nuovi scenari di test o nuove personalità di gioco semplicemente iniettando una diversa classe di valutazione, mantenendo inalterato il core algoritmico.

### 3. Modellazione Algoritmica e Ottimizzazione

In accordo con la fase di *Modeling* della metodologia CRISP-DM, questo capitolo dettaglia le tecniche algoritmiche adottate per costruire il motore decisionale. Partendo dalla struttura base del Minimax, vengono analizzate le ottimizzazioni specifiche implementate per ridurre la complessità computazionale e migliorare la qualità della strategia di gioco.

#### 3.1. L'Algoritmo Base: Minimax e Alpha-Beta Pruning

Il nucleo decisionale dell'agente è basato sull'algoritmo **Minimax**, una tecnica ricorsiva utilizzata nei giochi a somma zero per minimizzare la massima perdita possibile. Dato uno stato  $S$ , l'algoritmo esplora l'albero delle mosse future fino a una profondità  $d$ .

Tuttavia, la complessità esponenziale dell'albero di gioco del Forza 4 ( $7^d$ ) rende impraticabile una ricerca esaustiva. Per mitigare questo problema, è stata implementata la tecnica di **Alpha-Beta Pruning**. Questa ottimizzazione permette di "tagliare" (prune) i rami dell'albero che non influenzano la decisione finale, riducendo drasticamente i nodi visitati senza alterare il risultato matematico [1].

#### 3.2. Ottimizzazioni Implementate

Per rendere il motore capace di operare in tempo reale con profondità elevate, sono state introdotte tre ottimizzazioni critiche rispetto all'implementazione standard.

##### 3.2.1. Bit-Level Optimization (Hardware Counting)

La valutazione dello stato di gioco richiede il conteggio frequente dei pattern (es. "quante terzine ha il giocatore X?"). Le implementazioni tradizionali in Python utilizzano la conversione in stringa (`bin(x).count('1')`), un'operazione lenta e dispendiosa in termini di memoria.

Nel nostro *Adaptive Bitboard Engine*, abbiamo sfruttato le istruzioni native delle CPU moderne accessibili tramite Python 3.10+:

$$\text{PopCount}(x) \rightarrow \mathbf{x.bit\_count()} \quad (2)$$

Questa modifica sostituisce l'allocazione di stringhe con operazioni dirette sui registri del processore (istruzione hardware POPCNT), garantendo un incremento prestazionale significativo durante la valutazione euristica.

### **3.2.2. Move Ordering (Center-First Heuristic)**

L'efficacia dell'Alpha-Beta Pruning dipende fortemente dall'ordine in cui vengono esaminate le mosse. Se la mossa migliore viene trovata subito, l'algoritmo può tagliare immediatamente il resto dei rami. Poiché nel Forza 4 il controllo del centro è statisticamente determinante per la vittoria, l'ordine di esplorazione statico  $[0, 1, 2, 3, 4, 5, 6]$  è stato sostituito con un vettore di priorità dinamico:

$$\text{Order} = [3, 2, 4, 1, 5, 0, 6] \quad (3)$$

Esaminando prima la colonna centrale (3) e poi quelle adiacenti, l'algoritmo massimizza la probabilità di innescare *cutoff* (tagli) anticipati, accelerando la ricerca del 30-40% nelle fasi iniziali e medie della partita.

### **3.2.3. Transposition Tables (Memoization)**

Nel Forza 4, sequenze di mosse diverse possono portare alla stessa configurazione della scacchiera (trasposizioni). Per evitare di ricalcolare il valore Minimax per stati già visitati, è stata introdotta una **Transposition Table**. Questa cache utilizza una mappa hash dove la chiave è la coppia di Bitboard ( $P1, P2$ ) e il valore contiene:

- **Score:** Il punteggio calcolato.
- **Flag:** Indica se il valore è esatto o un limite (Upper/Lower bound).
- **Depth:** La profondità a cui è stata effettuata l'analisi.

Se l'algoritmo incontra uno stato già presente in tabella con una profondità sufficiente, restituisce immediatamente il valore salvato, trasformando la complessità temporale in complessità spaziale.

## **3.3. Euristica Avanzata e Anti-Blunder**

Mentre il Minimax gestisce la profondità di calcolo, la qualità della decisione dipende dalla funzione di valutazione statica (*Evaluator*). Oltre al conteggio standard di linee da 2 e 3 pezzi, sono stati introdotti moduli di ragionamento tattico avanzato.

### **3.3.1. Riconoscimento "Fork" (Double Threat)**

Una limitazione comune delle AI di base è l'incapacità di prevedere minacce multiple. Il nostro motore identifica esplicitamente le configurazioni di "Fork", ovvero stati in cui un giocatore ha due minacce di vittoria simultanee.

### Logica Fork

Se la bitmask delle minacce contiene  $\geq 2$  bit attivi, viene assegnato un bonus/malus massivo ( $\pm 5000$ ).

Questo costringe l'IA a bloccare preventivamente le mosse che porterebbero a tali configurazioni, invece di attendere che la minaccia diventi imparabile.

### 3.3.2. Anti-Blunder Check (Zugzwang)

È stato implementato un controllo di sicurezza ("Anti-Blunder") per evitare errori non forzati. Prima di considerare una mossa valida, l'Evaluator simula se l'inserimento della pedina nella colonna  $C$  sbloccherebbe una vittoria immediata per l'avversario nella cella  $C+1$  (quella immediatamente sopra). In tal caso, alla mossa viene applicata una penalità di  $-100.000$  ("Colonna Velenosa"), impedendo all'IA di "regalare" la partita pur di perseguire un vantaggio minore.

## 3.4. Adattamento Dinamico (Profilazione)

Infine, il motore supera l'approccio statico integrando i pesi dinamici forniti dal Profiler. La funzione di valutazione  $E(s)$  non è costante, ma è definita come combinazione lineare pesata:

$$E(s) = \sum_i (w_i \cdot \phi_i \cdot f_i(s)) \quad (4)$$

Dove  $w_i$  è il peso base della feature  $f_i$  (es. controllo diagonale) e  $\phi_i$  è il moltiplicatore adattivo derivato dall'analisi in tempo reale dell'avversario. Se il Profiler rileva una debolezza diagonale ( $\phi_{diag} > 1.0$ ), l'IA tenderà a preferire stati che massimizzano le connessioni oblique, sfruttando attivamente il bias cognitivo del giocatore umano.

## Riferimenti bibliografici

- [1] D. E. Knuth and R. W. Moore, *An analysis of alpha-beta pruning*, in *Artificial Intelligence*, vol. 6, no. 4, pp. 293-326, 1975.
- [2] J. Tromp, *The Fhourstones Benchmark*, <http://tromp.github.io/c4/fhour.html>. (Accesso: Gennaio 2026).
- [3] K. Wang et al., *Research on Different Heuristics for Minimax Algorithm: Insight from Connect-4 Game*, in *Proceedings of 2019 International Conference on Computing and Artificial Intelligence*, 2019.
- [4] P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma, *Opponent Modelling for Case-Based Adaptive Game AI*, in *Entertainment Computing - ICEC 2004*, Springer, 2004.
- [5] H. Silva et al., *Dynamic Difficulty Adjustment through an Adaptive AI*, in *ResearchGate Publications*, 2017.
- [6] L. V. Allis, *Searching for Solutions in Games and Artificial Intelligence*, Ph.D. Thesis, University of Limburg, Maastricht, 1994.