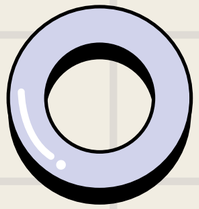
CORSO DI:
INTELLIGENZA ARTIFICIALE
FACOLTA' DI INGEGNERIA
INFORMATICA E DELL'AUTOMAZIONE
PROF: ALDO FRANCO DRAGONI

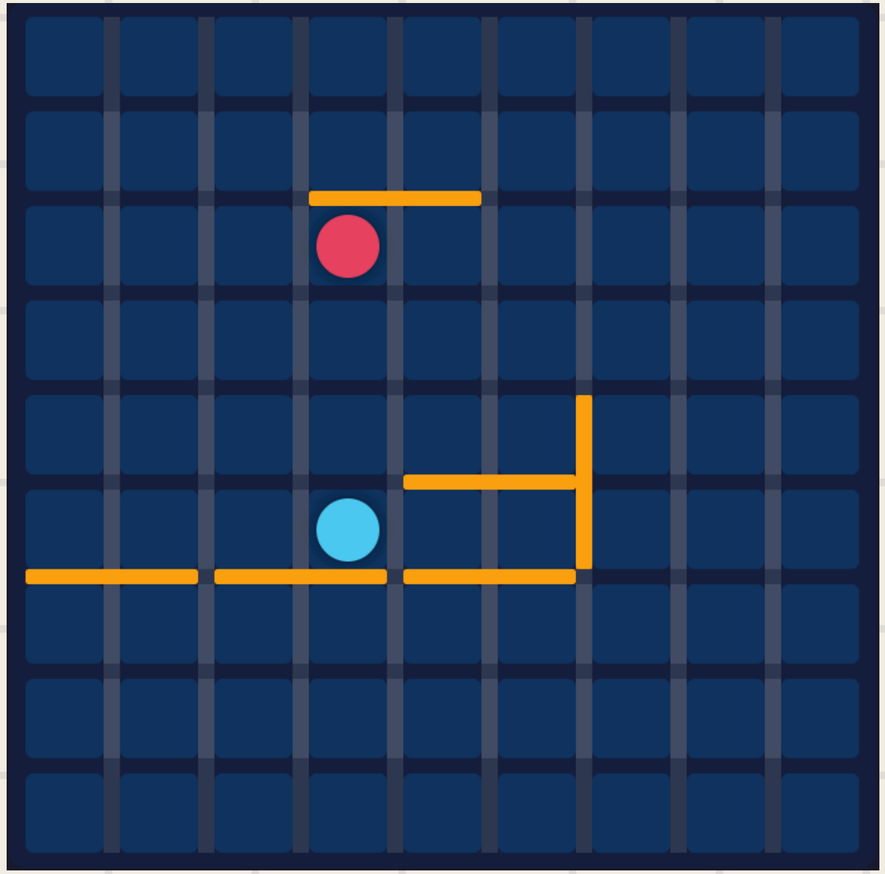
QUORIDOR



VIDEOGIOCO

ANDREA ALTIERI 1128865
ANDREA CONTURSI 1126917
ANDREA FLAIANI 1126928

Cos'è Quoridor?



Un duello tattico dove devi decidere se avanzare verso la meta o costruire un muro per sbarrare la strada al tuo avversario.

Regola d'oro: non puoi mai chiudere completamente l'avversario.

Regole del gioco

- **Turni: ogni giocatore compie una sola azione per volta.**
- **Scelta: puoi muovere la tua pedina o piazzare un muro.**
- **Movimento: ci si sposta di una casella in orizzontale o verticale.**
- **Sbarramento: i muri servono a rallentare l’avversario.**
- **Il Salto: se due pedine si scontrano, puoi scavalcare l'avversario.**
- **Vittoria: vince il primo che raggiunge il lato opposto della scacchiera.**

Livelli di difficoltà

- Livello **Easy** (Random): L'IA sceglie una mossa casuale tra tutte quelle valide. Imprevedibile ma senza strategia.
- Livello **Medium** (Greedy): Algoritmo "goloso". Punta sempre al percorso più breve verso la meta, ignorando l'avversario.
- Livello **Hard** (Minimax + Alpha-Beta): Analisi tattica a profondità 2. Usa muri difensivi se l'avversario è troppo vicino.
- Livello **Extreme** (Heuristic Minimax): Analisi profonda (depth 5). Filtra le mosse migliori e calcola ogni possibile contromossa.

Livello Medium

- **Concetto:** L'IA agisce come un "corridore puro". Non si cura dell'avversario, ma pensa solo a se stessa.
- **Algoritmo Greedy:** Ad ogni turno, analizza tutte le mosse legali e sceglie quella che minimizza istantaneamente la propria distanza dalla meta.
- **Funzione di Valutazione Locale:**

Score = Distanza IA

- **Il Limite:** Non avendo memoria o capacità di previsione, il livello Medium cade facilmente in trappole banali o loop infiniti se bloccato dai muri.

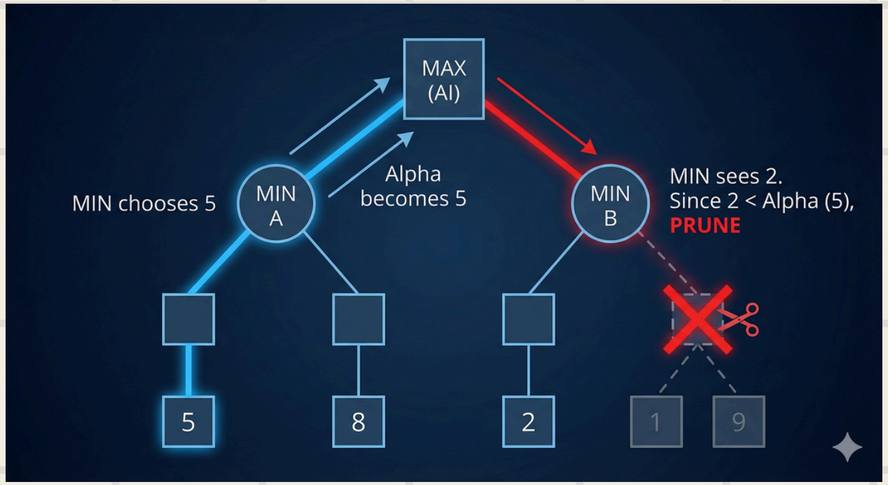
Livello Hard

- **Pathfinding Dinamico (BFS):** Utilizzo dell'algoritmo Breadth-First Search per il calcolo del cammino minimo reale, ricalcolato ad ogni variazione della topologia (posizionamento muri).
- **Funzione Euristica (f(s)):** Valutazione numerica dello stato della scacchiera:

$$\textbf{Score} = \textbf{Dist\_Avversario} - \textbf{Dist\_IA}$$

- **Trigger Strategico:** Controllo della "Soglia di Pericolo". Se l'avversario è a dist < 4, il sistema forza una Priorità Difensiva (muro), ignorando rami di movimento meno urgenti.

Livello Hard



Algoritmo Minimax

- **Maximizer (IA) vs Minimizer (Umano):** Ricerca della mossa ottima assumendo un avversario razionale.
- **Depth 2:** Esplorazione dell'albero degli stati fino al secondo livello di profondità.
- **Ottimizzazione Alpha-Beta:**
- **α e β:** Monitoraggio dei bound inferiori e superiori dei punteggi garantiti.
- **Pruning (Potatura):** Taglio immediato dei rami che non possono migliorare la soluzione corrente.
- **Efficienza:** Riduzione della complessità temporale senza perdita di ottimalità.

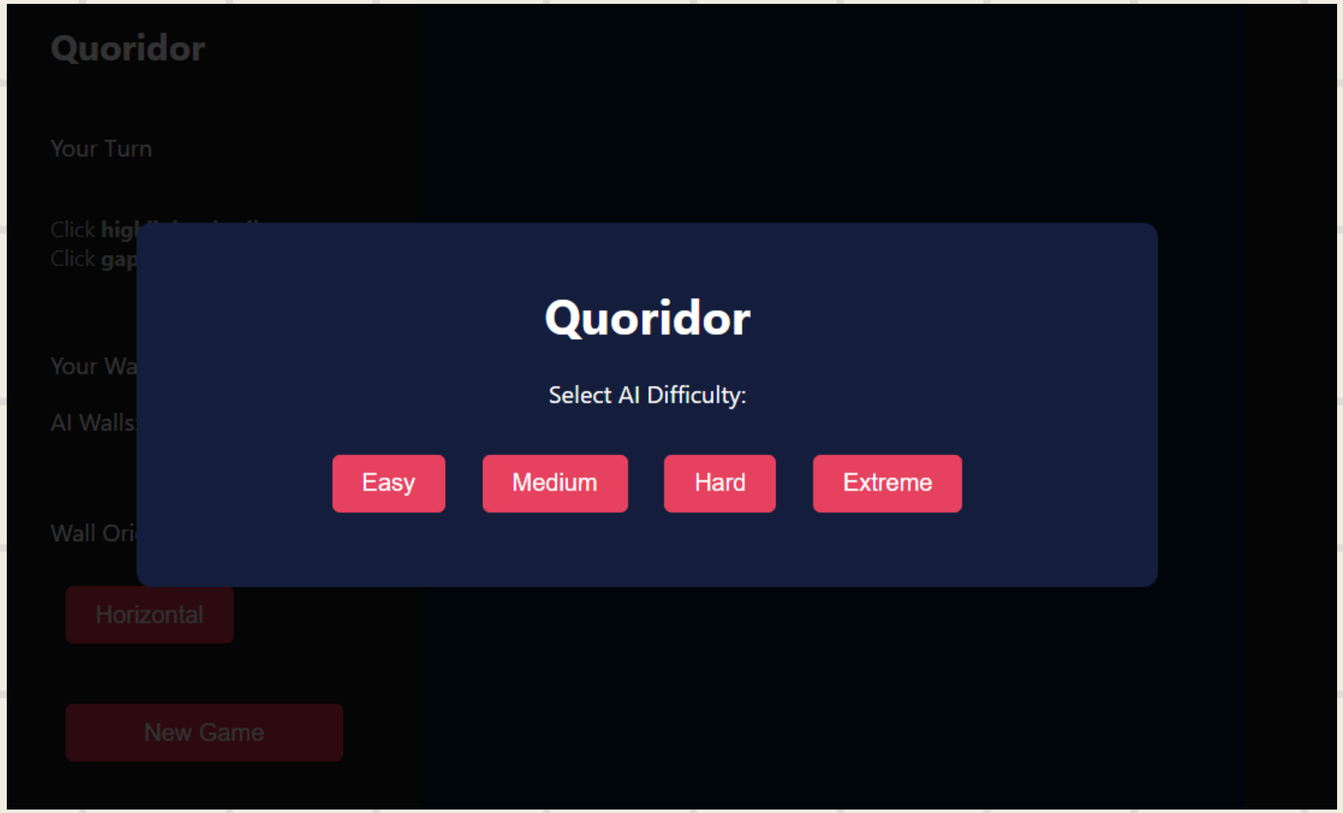
Livello Extreme


```
% --- Selezione delle migliori mosse Pedina ---
findall(Score-M, (
    member(M, AllPawnMoves),
    make_move(State, M, NextState),
    evaluate(NextState, Score)
), ScoredPawns),
sort(ScoredPawns, SortedPawns),
reverse(SortedPawns, TopPawns),
take_n(5, TopPawns, BestPawns), % Seleziona le top 5

% --- Selezione dei migliori 3 Muri ---
take_n(3, TopWalls, BestWalls),

% --- Minimax profondo solo sulle mosse filtrate ---
minimax_limited(State, CombinedMoves, 5, -1000, 1000, Move, _)
```

Problema della Complessità:

- A profondità 5, il numero di stati possibili diventa ingestibile per un calcolo in tempo reale.
- **Filtro Euristico Preliminare:** L'IA non esplora l'intero spazio delle mosse.
- **Step 1 (Ranking):** Valutazione rapida di tutte le mosse legali.
- **Step 2 (Smart Selection):** Vengono mantenute solo le Top 5 mosse pedina e le Top 3 muri (tramite funzione *take_n*).
- **Step 3 :** Esecuzione del Minimax (Depth 5) solo sul sottoinsieme filtrato.
- **Vantaggio:** Visione strategica a lungo termine con tempi di risposta ottimizzati.

Scelta della difficoltà

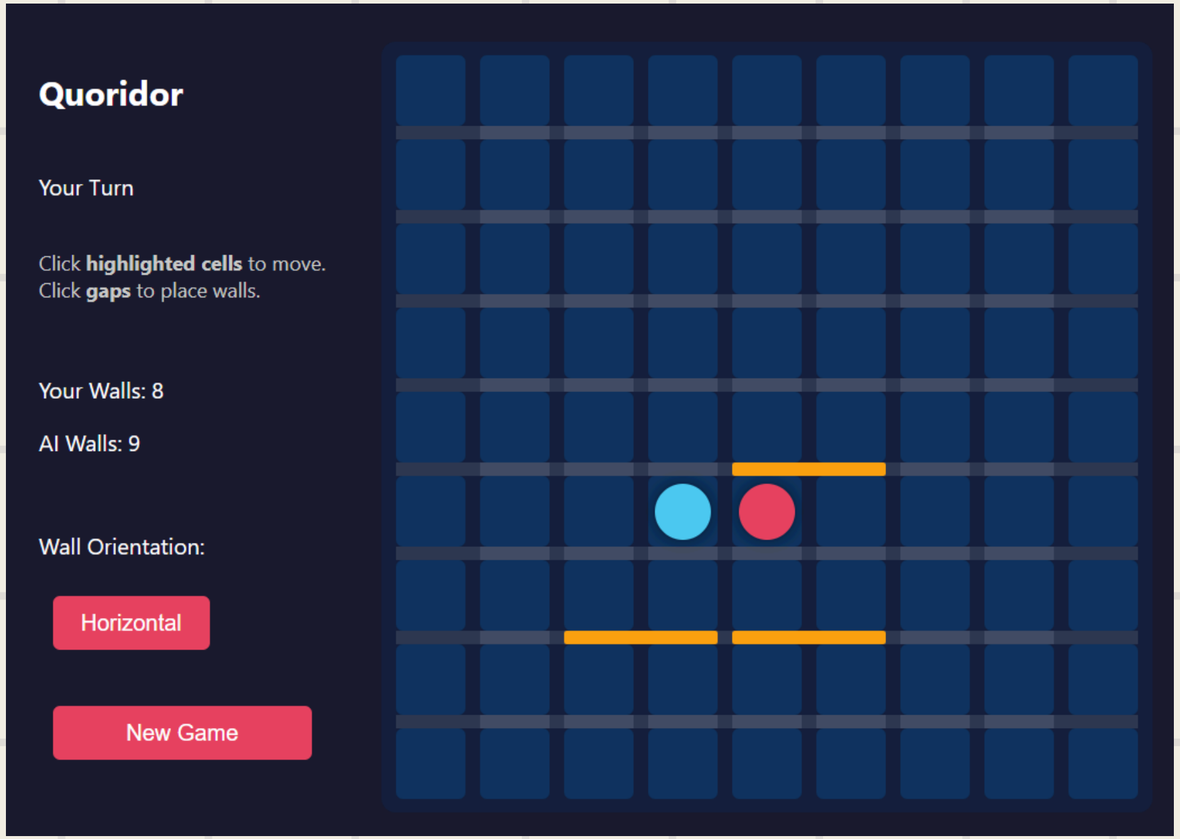


Non appena si fa partire il gioco viene mostrata la seguente schermata che consente di selezionare il livello di difficoltà della partita

Inizio partita

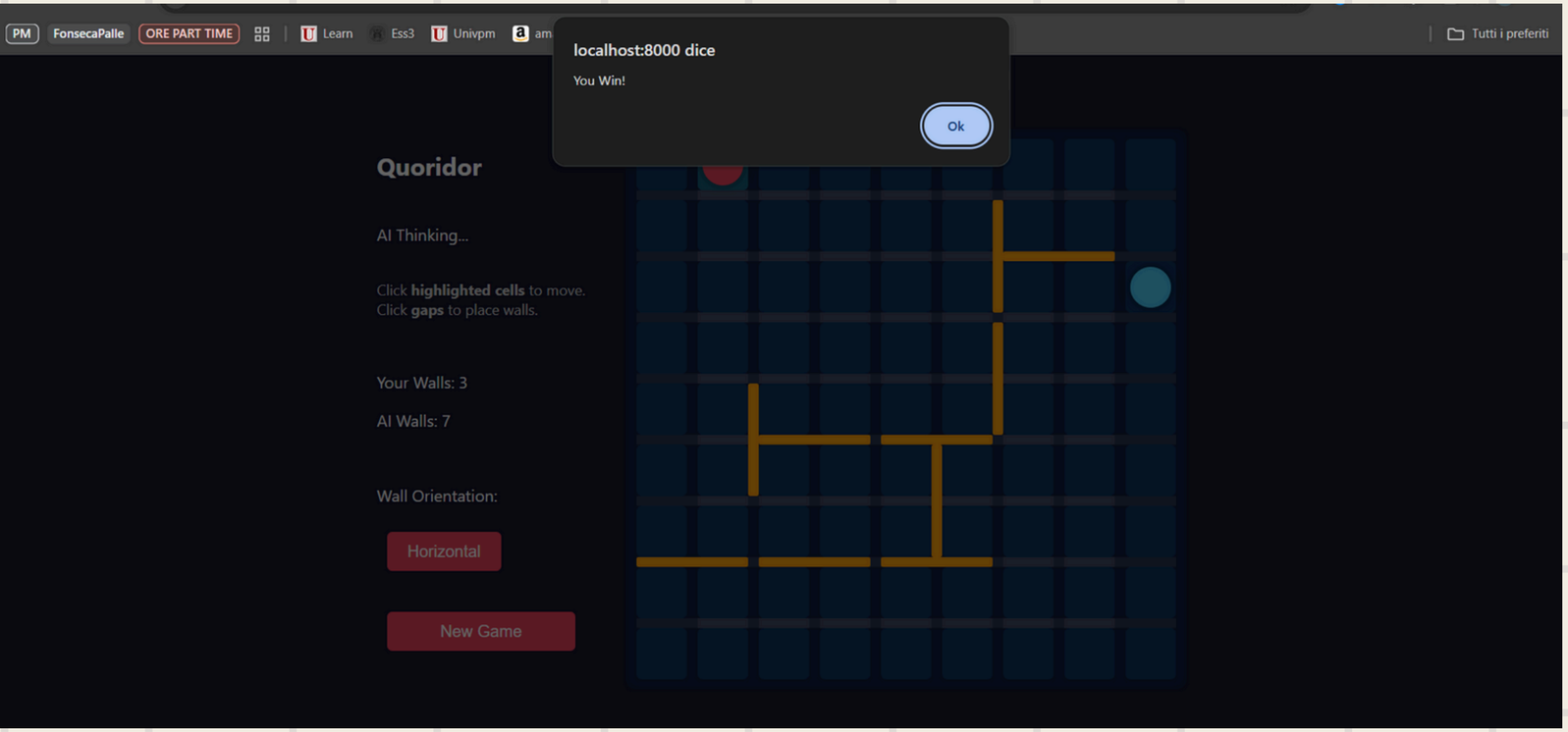
Una volta scelta la difficoltà desiderata, apparirà la griglia 9x9 all'interno del quale si trovano posizionati i due pedoni (quello che potrà muovere il giocatore è quello rosso, quello azzurro verrà guidato dall'IA). Sulla sinistra è presente un pulsante che consente di scegliere l'orientamento dei muri da posizionare

Turni iniziali



Quest'immagine mostra lo stato della partita dopo 7 turni. Si nota il posizionamento dei primi muri da parte sia del giocatore sia dell'IA; inoltre, sulla sinistra ci sono dei valori che indicano quanti muri ha ancora a disposizione ogni giocatore.

Verifica della vittoria



Dopo svariati turni, si nota come la pedina rossa sia arrivata sul suo lato opposto della griglia prima della pedina azzurra, di conseguenza, la vittoria viene assegnata al giocatore.

Stato del gioco


```
% State: game(P1, P2, Walls, Turn)
% P1, P2 = player(X, Y, WallsLeft)
% Walls = [wall(X, Y, Orientation), ...]
% Turn = p1 or p2

initial_state(game(player(5, 9, 10),
                   player(5, 1, 10),
                   [],
                   p1)).
```

Lo stato del gioco è un pacchetto chiamato game che contiene:

- I due giocatori P1 e P2
- Walls: la lista di tutte le barriere già messe sulla scacchiera.
- Turn: di chi è il turno.

- player(X, Y, WallsLeft) significa: il pedone del giocatore sta nella casella di coordinate (X,Y) e gli restano WallsLeft barriere.

Validazione di una mossa: pedone o barriera


```
% Controllo generale: sceglie il giocatore corrente e delega
is_valid_move(game(P1, P2, Walls, Turn), Move) :-
    ( Turn == p1 -> CurrentPlayer = P1, OtherPlayer = P2
    ;               CurrentPlayer = P2, OtherPlayer = P1 ),
    is_valid_action(CurrentPlayer, OtherPlayer, Walls, Move).

% ----- Movimento del pedone -----
% Mossa del tipo: move(NX, NY)
is_valid_action(player(X, Y, _), _, Walls, move(NX, NY)) :-
    valid_pos(NX, NY),                % dentro la scacchiera
    adjacent(X, Y, NX, NY),           % casella adiacente (su/giù/sx/dx)
    \+ wall_blocks(X, Y, NX, NY, Walls). % nessuna barriera blocca il passaggio

% ----- Posizionamento di una barriera -----
% Mossa del tipo: place_wall(X, Y, O)
is_valid_action(player(PX, PY, W), player(OX, OY, _), Walls,
                place_wall(X, Y, O)) :-
    W > 0,                            % il giocatore ha ancora muri
    valid_wall_pos(X, Y),             % la posizione è valida per i muri
    \+ wall_overlaps(X, Y, O, Walls), % non si sovrappone/interseca illegalmente
    \+ blocks_path(player(PX, PY, W), % non blocca il cammino di P1
                   player(OX, OY, _), % né quello di P2
                   [wall(X, Y, O)|Walls]).
```

is_valid_move/2 decide chi è il giocatore di turno e passa tutto a is_valid_action/4, che gestisce entrambi i tipi di mossa nella stessa interfaccia.

Funzione di valutazione euristica


```
% SCORE = Distanza_P1 - Distanza_P2
% + = buono per IA (P2 più vicina)
% - = buono per umano (P1 più vicina)

evaluate(game(P1, P2, Walls, _), Score) :-
    shortest_path_len(P1, Walls, p1, D1),    % Distanza minima P1 → riga 1
    shortest_path_len(P2, Walls, p2, D2),    % Distanza minima P2 → riga 9
    Score is D1 - D2.                        % Differenza di distanza

shortest_path_len(player(X, Y, _), Walls, Player, Dist) :-
    goal_row(Player, GoalY),                 % Riga di arrivo
    bfs_dist([[X, Y, 0]], GoalY, Walls, [], Dist). % BFS con distanza
```

shortest_path_len/4 restituisce il numero esatto di mosse per il percorso più breve, usando BFS, aggirando i muri.

La funzione evaluate calcola prima D1 e D2, ovvero la distanza minima rispettivamente di P1 e P2 dalla riga di arrivo.
Successivamente, viene calcolato lo score = D1 - D2:
Se questo valore è positivo, l'IA è messa meglio (P2 è più vicina), viceversa, l'umano è più vicino a vincere.

Applicazione del Minimax - caso base


```
minimax(State, 0, _, _, nil, Score) :-
    evaluate(State, Score), !.
```

Questa parte di codice rappresenta il caso base della ricorsione. Quando la profondità raggiunge 0, l'algoritmo smette di esplorare l'albero. La funzione valuta lo stato attuale utilizzando l'euristica e restituisce lo score ottenuto, terminando così la ricerca.

L'albero di minimax ha una profondità massima (2 per hard, 5 per extreme).
Quando l'IA arriva a quel limite:

- Non esplora altre mosse
- Chiama evaluate → calcola Distanza_P1 - Distanza_P2
- Restituisce lo score e torna indietro all'albero

Decisione - "Chi gioca?"


```
minimax(State, Depth, Alpha, Beta, BestMove, BestScore) :-
    findall_pawn_moves(State, Moves),
    State = game(_, _, _, Turn),
    (Turn == p2 -> maximize(...) ; minimize(...)).
```

Questa sezione genera tutte le mosse di pedone possibili nello stato attuale. Successivamente, l'algoritmo verifica di chi è il turno: se è il turno dell'IA (P2), invoca la funzione maximize per cercare il miglior score possibile; diversamente, invoca minimize perché l'avversario cercherà di minimizzare il vantaggio dell'IA.

Massimizzazione - "Io scelgo il meglio"


```
maximize([Move|Rest], State, Depth, Alpha, Beta,
         CurrentBest, CurrentScore, FinalMove, FinalScore) :-
    make_move(State, Move, NextState),
    minimax(NextState, Depth-1, Alpha, Beta, _, Score),
    (Score > CurrentScore -> NewMove = Move, NewScore = Score
    ;   NewMove = CurrentBest, NewScore = CurrentScore),
    (NewScore >= Beta -> FinalMove = NewMove
    ;   NewAlpha is max(Alpha, NewScore),
        maximize(Rest, ...)).
```

Quando è il turno dell'IA, l'algoritmo entra in modalità massimizzazione. In questo caso, l'IA valuta tutte le mosse disponibili e seleziona quella che produce il punteggio più alto. L'IA cerca di avvicinarsi il più possibile alla vittoria, massimizzando il valore della posizione secondo la funzione di valutazione euristica. Per esempio, se le mosse disponibili producono punteggi di 3, 1 e 5, l'IA sceglie la mossa che dà 5, il valore massimo.

Minimizzazione - "L'avversario mi frega"


```
minimize([Move|Rest], State, Depth, Alpha, Beta,
         CurrentBest, CurrentScore, FinalMove, FinalScore) :-
    make_move(State, Move, NextState),
    minimax(NextState, Depth-1, Alpha, Beta, _, Score),
    (Score < CurrentScore -> NewMove = Move, NewScore = Score
    ;   NewMove = CurrentBest, NewScore = CurrentScore),
    (NewScore =< Alpha -> FinalMove = NewMove
    ;   NewBeta is min(Beta, NewScore),
        minimize(Rest, ...)).
```

Quando è il turno dell'avversario umano, l'algoritmo passa in modalità minimizzazione. In questo caso, l'algoritmo assume che l'avversario giocherà nel modo più intelligente possibile, scegliendo la mossa che produce il punteggio più basso (dal punto di vista dell'IA). Questo rappresenta il fatto che l'avversario farà tutto il possibile per dannegggiare le prospettive dell'IA. Usando lo stesso esempio, se le mosse dell'avversario producono punteggi di 3, 1 e 5, l'algoritmo assume che l'avversario sceglierà 1, il valore minimo.

Grazie per l’attenzione!

ANDREA ALTIERI 1128865
ANDREA CONTURSI 1126917
ANDREA FLAIANI 1126928