NEMO

Mattia Campestri
Tommaso Liverani
Gioele Pisanelli

Scelte progettuali

- Gioco a due giocatori a somma zero → Algoritmo minmax con tagli alpha-beta ed iterative deepening
- Tempo limitato → Linguaggio C per efficienza
- 8 GB RAM → Transposition Table
- 2 Core → Multithreading con algoritmo Simplified ABDADA

- NemoTablut/board.h
- NemoTablut/entry.h
- NemoTablut/moves.h
- NemoTablut/abdada.h
- NemoTablut/client.h
- NemoTablut/minmax.h
- NemoTablut/search.h
- NemoTablut/move_type.h
- NemoTablut/evaluation.h
- NemoTablut/parameters.h
- NemoTablut/move_sorting.h
- NemoTablut/time_control.h
- NemoTablut/draws_control.h
- NemoTablut/trasformation.h
- NemoTablut/search_threads.h
- NemoTablut/traspostition_table.h

Rappresentazione dello stato

- 2 bit per ogni casella
- Un intero (32 bit) per ogni riga
- Gli ultimi 18 bit dell'intero codificano le 9 caselle della riga
- Struttura → allocazione statica per evitare malloc

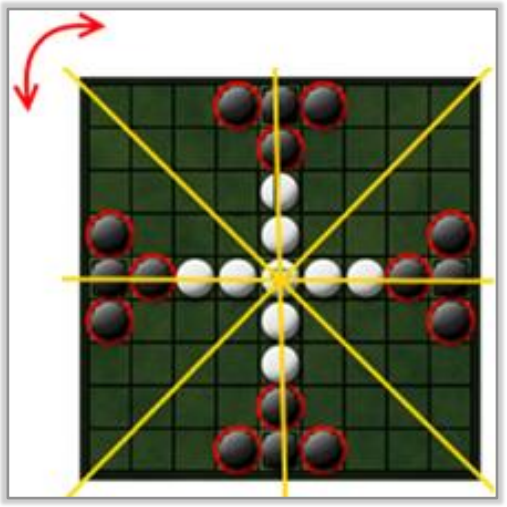
```
#define EMPTY 0U        // 00
#define BLACK 1U        // 01
#define WHITE 2U        // 10
#define WHITE_KING 3U   // 11

typedef struct {
        uint32_t Board[9];
} BoardState;
```

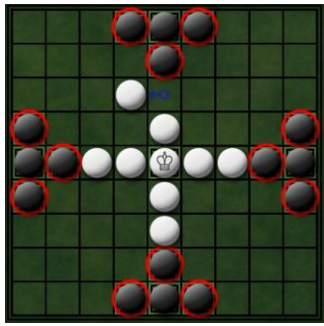
```
board->Board[0] = 0b00000001010100000U;
board->Board[1] = 0b00000000100000000U;
board->Board[2] = 0b00000001000000000U;
board->Board[3] = 0b01000001000000001U;
board->Board[4] = 0b01011010111010101U;
board->Board[5] = 0b01000001000000001U;
board->Board[6] = 0b00000001000000000U;
board->Board[7] = 0b00000000100000000U;
board->Board[8] = 0b00000001010100000U;
```

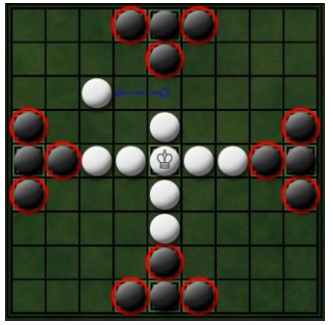
Transposition Table

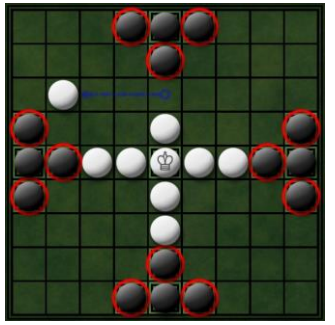
- Entry di 14 byte
- Riga della table implementata come array di entry
- Per ogni scacchiera controllo nella table la presenza di una scacchiera equivalente già incontrata. Considero come equivalenti la scacchiera stessa, le sue simmetrie e rotazioni
- Update: ad ogni aggiunta allo stesso indice l'array "scorre" e viene sovrascritta l'ultima entry → rimuovo le entry più vecchie

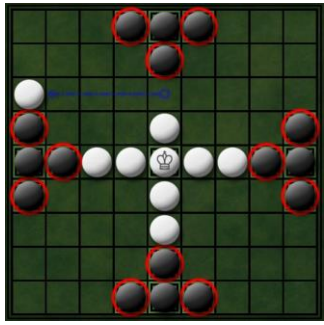
```
typedef struct {
    uint64_t hash;
    Move nextMove;
    ScoreType heuristicValue;
    uint8_t depthLeft;
    TypeNode type;
} Entry;
```

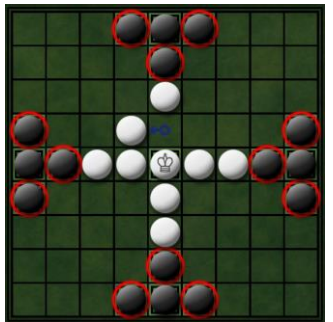


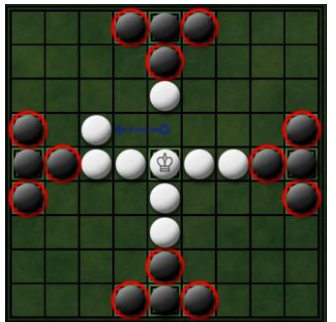


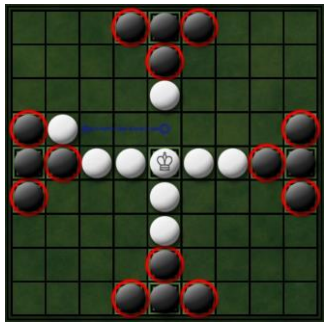




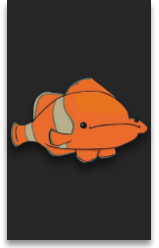








Euristiche e generazione mosse



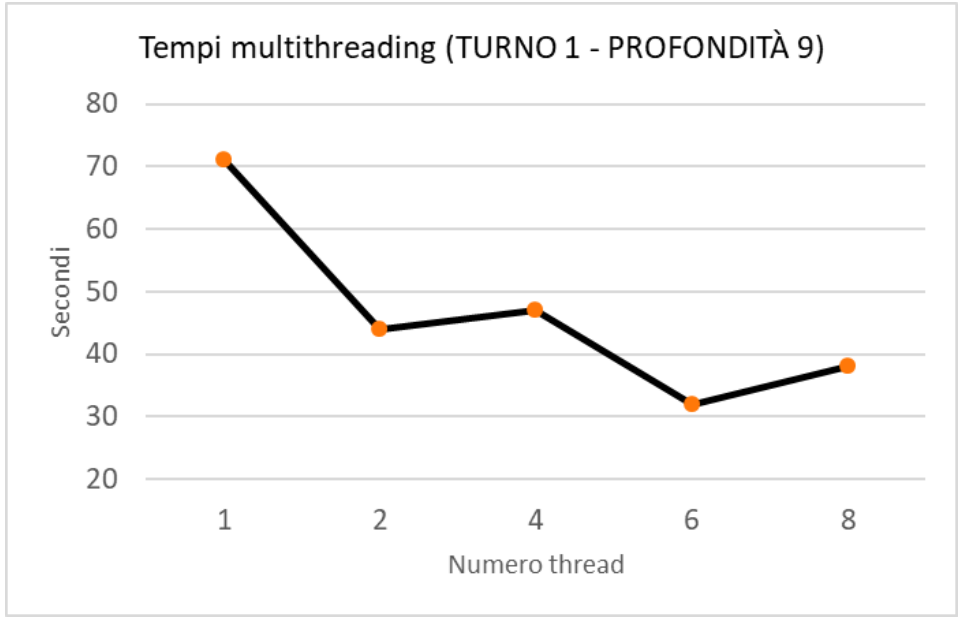
- Euristiche differenziate per bianco e nero
- Parametri principali considerati dalle euristiche:
  - Differenza tra i numeri dei pezzi
  - Bonus al nero per ogni casella di vittoria "bloccata"
  - Bonus al bianco proporzionale alla distanza delle pedine nere dal re
  - Controllo anticipato della vittoria del bianco
- Mossa = intero 16 bit
  `(startRow, startColumn, targetRow, targetColumn)`
- Mosse applicate allo stato mediante macro per manipolazione di bit

```
//chiamata da min. ha appena mosso il bianco
ScoreType evaluateForWhite(BoardState board) {

//chiamata da max, ha appena mosso il nero
ScoreType evaluateForBlack(BoardState board) {

typedef uint16_t Move;

#define APPLY_MOVE_WHITE(board, move) board[START_ROW(move)] ^=
#define APPLY_MOVE_BLACK(board, move) board[START_ROW(move)] ^=
#define APPLY_MOVE_KING(board, move) board[START_ROW(move)] ^= (

#define GET_CELL_STATE(board, row, column) ((board[row] >> (16
#define CREATE_MOVE(startRow, startColumn, targetRow, targetCo
#define GETSTART(board,move) GET_CELL_STATE(board,((move >> 12U)

#define DELETE_WHITE(board, row, column) board[row] ^= (1U << (
#define DELETE_BLACK(board, row, column) board[row] ^= (1U << (
#define DELETE_WHITE_KING(board, row, column) board[row] ^= (3U
```

Multithreading

- Algoritmo Simplified ABDADA
- Funzionamento:
  - Ogni thread inizia la ricerca dalla posizione attuale
  - Generazione mosse possibili
  - Ricerca della prima mossa in ogni caso
  - Sincronizzazione per esplorazione mosse successive:
    - Thread inizia esplorazione della mossa e lo segnala agli altri thread mediante una hash table, con chiave `hash(board, move)`
    - Mossa già in esplorazione da parte di un altro thread → spostata in coda all'array di mosse da esplorare (mossa differita)
    - Cooperazione fra thread mediante Transposition Table

```
typedef struct {
    uint64_t searchHashes[MOVE_TABLE_WIDTH];
    pthread_spinlock_t lock;    // Lock per mutua esclusione
} MovesTableEntry;
```



http://www.tckerrigan.com/Chess/Parallel_Search/Simplified_ABDADA/