Progetto Lasca

Gruppo 52 - Paul Toffoloni

Introduzione

Lasca è una variante della Dama, ideato dallo scacchista tedesco Emanuel Lasker nel 1911; è noto anche come Laska o semplicemente come Lasker, o anche Laskers. Si gioca con le stesse regole della Dama, ma su un campo di sole 49 caselle (7 per lato). I pezzi catturati non vengono rimossi dal campo, ma trascinati e sormontati dal pezzo catturante.

Struttura

> All'interno della documentazione di Doxygen si possono trovare degli schemi che mostrano la struttura del progetto a livello grafico in modo da facilitare la lettura.

Il programma è strutturato in diversi Abstract Data Types (ADT).

struct

- **Board**: Il campo da gioco, contenente una matrice fissa di Cell;
- **Cell**: Una singola cella del campo da gioco, contenente delle coordinate x e y (int) ed eventualmente una Tower;
- **Piece**: Un singolo pezzo di gioco, caratterizzato da colore e tipo (soldier o officer);
- **Tower**: Una torre, ovvero un array di Piece, ciò che viene usato in realtà giocando, caratterizato da un'altezza, un Color e un Type, questi ultimi due indicati dal colore e dal tipo della prima pedina della torre, ovvero quella che controlla la torre;
- **Move**: Una singola mossa, contenente le Cell di origine e destinazione ed eventualmente una Cell da mangiare;
- **Moves**: Una raccolta di Move che serve nel momento in cui dobbiamo calcolarci le possibili mosse.

enum

- **Type** `{SOLDIER = 0, OFFICER = 1}` : rappresenta il tipo di pedina, quindi se è una dama o no;
- **Color** `{WHITE = 0, BLACK = 1}` : rappresenta il colore di una pedina;
- **bool** `{false = 0, true = 1}` : siccome in ANSI C non esiste bool e non si può importare `<stdbool.h>` ho creato un enum per simulare il tipo booleano.

Ogni struttura è a conoscenza esclusivamente dei suoi parametri e non conosce dati di altre strutture, seguendo il principio dell'incapsulamento per accederci utilizza metodi come getter e setter, creati all'interno di ogni struttura. Purtroppo C non è un linguaggio ad oggetti quindi non si può seguire un approccio ad oggetti "tradizionale" ma bisogna applicare dei metodi alternativi, per esempio al posto delle classi verranno usate delle struct.

Ogni funzione è il più possibile astratta con l'obiettivo del riuso del codice, di conseguenza la struttura proposta è anche estremamente estendibile ed è molto semplice implementare nuove funzionalità senza dover né modificare né conoscere per filo e per segno ogni riga del codice già scritto.

Funzionalità

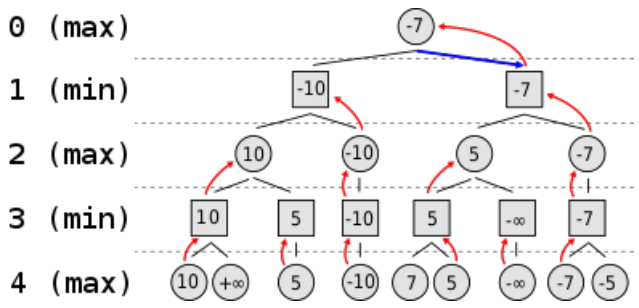
Il gioco, funzionante su terminale, presenta un menù con il quale l'utente può scegliere se giocare contro un altro giocatore, contro il computer o uscire. Dopodichè, il gioco comincerà.

Per giocare, il codice è estremamente user friendly e allo stesso tempo limitato: non permette all'utente di fare scelte sbagliate che potrebbero andare a creare problemi, come per esempio immissione di coordinate sbagliate: sarà il programma che proporrà all'utente quali mosse sono disponibili e gliene farà scegliere una. Ovviamente l'inserimento della scelta da parte dell'utente è controllata e non permette l'inserimento errato di mosse. Una volta scelta la mossa, sarà il sistema a preoccuparsi del resto, muovendo la pedina ed eventualmente mangiando, se è una mossa di conquista. Anche per quanto riguarda queste ultime, sarà il sistema ad accorgersi se è possibile mangiare, e presenterà all'utente solo tali mosse. Infine, il gioco continuerà e cambierà turno, a me no che non ci siano più mosse disponibili per colui che deve giocare il turno, in quel caso la partita finirà.

Per il giocatore singolo, è stato implementato l'algoritmo Minimax in modo che la CPU possa fare delle scelte con un certo criterio.

MiniMax

1. Crea un albero sulla base delle mosse disponibili, ogni livello di questo albero conterrà le mosse che un giocatore (sia CPU sia utente) può svolgere.
2. Attraversa ricorsivamente l'albero creato, assegnando ad ogni mossa un valore, valutato sulla base del vantaggio che può portare alla CPU. Il valore è calcolato nella funzione `evaluate(Board *board)` e si basa sul numero di pezzi presenti sul campo da gioco dopo aver simulato una mossa, quindi indica se ci sono state delle mangiate o no e a favore di chi. Inoltre, tarato con meno importanza, c'è anche la quantità di pedine `Officer` presenti, quindi anche la possibilità di promuovere una pedina ha ruolo nella valutazione. Più il numero è alto, più e vantaggioso per il giocatore.
3. Sceglie, per ogni livello, la mossa più conveniente comparando i valori delle varie mosse, a seconda del turno che si sta controllando si sceglierà il numero più alto (se il turno è lo stesso della CPU) oppure il numero più basso se il turno controllato è dell'utente.



Più la depth della AI è alta, quindi il numero di turni e di conseguenza il numero di mosse controllati, più l'AI ci metterà a trovare un risultato. La complessità

computazionale è **O(b^m)**, dove b è il numero di mosse possibili per ogni turno ed m è il numero massimo di profondità (depth).

Possibili miglioramenti

- **Multi-mangiata**: Non è ancora possibile svolgere più mangiate in una mossa sola. Per implementare ciò servirebbe semplicemente aggiungere un ciclo o addirittura una ricorsione nella funzione che si occupa di controllare la mossa e di svolgere la mangiata;
- **GUI**: attraverso librerie grafiche come GTK oppure SDL si può aggiungere una interfaccia grafica molto più piacevole per l'utente.
- **Alpha-Beta Pruning**: Tecnica che permette di diminuire la complessità computazionale dell'algoritmo Minimax.

Difficoltà incontrate

Personalmente la difficoltà più grande è stata quella di abituarsi ad un linguaggio di programmazione di basso livello come C, essendo normalmente abituato ad usare Java, Swift e altri linguaggi che non necessitano di gestione della memoria manuale. In più, C non è neanche un linguaggio ad oggetti, di conseguenza strutturare un progetto del genere senza usare gli oggetti, ha allungato di molto i tempi, siccome ho cercato comunque di seguire le best practices di incapsulamento, astrazione ed information hiding. La cosa più interessante è stata senza dubbio l'implementazione dell'algoritmo Minimax per giocare contro la CPU.