Relazione finale progetto miniLaska

Sale Riccardo Nicoletti Gabriele

Indice:

1. Struttura del progetto
    a. Funzioni implementate
    b. Le strutture dati
    c. Implementazione gioco player vs player
    d. Implementazione gioco player vs CPU
        i. Strategie
        ii. Funzione ricorsiva
2. Suddivisione del lavoro
3. Difficoltà riscontrate

1. Struttura del progetto

a. Struttura del codice sorgente

Vista la complessità del codice generato è stata decisa una suddivisione dello stesso in macro aree per facilitare lo sviluppatore nella ricerca di specifiche funzioni.
Di seguito la suddivisione operata:

Shift -> Funzioni che operano uno scambio;
Board -> Funzioni che eseguono operazioni sul campo da gioco board;
Recursive -> Funzione ricorsiva che si occupa della valutazione del punteggio dei rami;
Find -> Funzioni che operano una ricerca;
Game_engine -> Funzione relative al ciclo di gioco;
Vector -> Funzioni operanti sui vettori.

b. Le strutture dati

La struttura utilizzata per rappresentare il campo da gioco è una matrice di dimensione 7 x 7 popolata da torri. Queste torri possono contenere un numero di pedine variabile da 0 a 3.
Per semplificare la gestione generale delle torri è stata creata una struct torre_t contenente un array statico di pedine.

```
typedef struct {
    char symbol;
    int is_enhanced;
} pedina_t;
```

La pedina è costituita da due campi, symbol e is_enhanced. Il primo di tipo char identifica il possessore (player_1 o player_2) della stessa oppure il carattere # che identifica l'eventuale casella "vuota" del campo. Il secondo di tipo int ha valore 1 se la pedina è un ufficiale (potenziata).

```
typedef struct {
    pedina_t pa[3];
} torre_t;
```

La torre è costituita da un singolo campo, un array di tipo pedina_t con dimensione 3.

c. Implementazione gioco player vs player

La seguente modalità di gioco viene eseguita tramite la lettura ad ogni turno delle coordinate da tastiera della mossa che il giocatore (1 o 2 a seconda del turno) vuole effettuare. Una volta letti i dati si esegue un controllo per segnalare tempestivamente eventuali errori in inserimento. Terminata la fase di interazione con l'utente il programma esegue la mossa seguendo le regole del gioco effettuando ove necessario, mangiate oppure eliminazioni di pedine nemiche. Concluse le operazioni di scambio tra le varie torri si effettua una verifica riguardante lo stato della partita per decretare l'eventuale vincita del giocatore valutando l'assenza di mosse effettuabili dall'avversario.

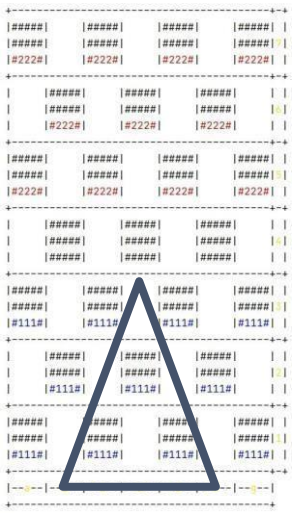
d. Implementazione gioco player vs CPU

i. Strategie

La tattica su cui si fonda la nostra proposta per l'implementazione del gioco player vs cpu è una MiniMax, basata su alcune delle strategie più utilizzate dagli esperti di Laska.

I tre punti cardine individuati sono:

1- Il mantenimento del triangolo centrale (c1, d2, e1):



Così facendo evitiamo che il giocatore possa ottenere delle torri potenziate, infatti mantenere il triangolo centrale per il maggior tempo possibile blocca le diagonali d'accesso alla nostra base.

2- Favorire mosse che spostino il fronte verso il centro della scacchiera:
Cerchiamo di far avanzare omogeneamente verso il fronte della scacchiera le pedine controllate dalla CPU in modo tale da mantenere il controllo della scacchiera, in caso di mangiata da parte del giocatore avremo subito disponibile una contromossa grazie alla linea difensiva del triangolo centrale.

3- Favorire le mangiate nei confronti delle torri deboli:

Si cerca sempre di effettuare mangiate verso torri deboli, in questo modo non solo liberiamo delle nostre pedine ma conquistiamo anche una pedina avversaria. Inoltre è sempre preferibile effettuare mosse che portino alla creazione di torri forti.

Torre debole giocatore 1

```
|#####|    |#####|
|#111#|    |#####|
|#222#|    |#111#|
|#222#|    |#222#|
|#####|    |#####|
```

Torre forte giocatore 1

```
|#####|    |#####|
|#111#|    |#####|
|#111#|    |#111#|
|#222#|    |#111#|
|#####|    |#####|
```

Dalla definizione di questi tre punti cardine è stato possibile determinare un punteggio specifico per ogni combinazione di torre mangiante e torre mangiata (torri differenti in base alla composizione), questi punteggi saranno fondamentali nella valutazione da parte della funzione ricorsiva del miglior ramo generato da una specifica mossa.

ii. Funzione ricorsiva

Al fine di poter scegliere la miglior mossa effettuabile dalla CPU per un generico stato del campo, abbiamo implementato una funzione ricorsiva che prende ispirazione dalla MiniMax.

Quest'ultima trova il punteggio di ogni ramo, di profondità scelta, permettendo alla game_engine_cpu di determinare la mossa che genera il ramo di punteggio maggiore. I rami rappresentano tutti i possibili percorsi di gioco che possono scaturire da una mossa.

Il valore del ramo esamina sia le mosse dell'avversario, valutate con un peso negativo, sia le mosse della CPU, valutate con un peso positivo. Il valore della singola mossa è ottenuto tramite una chiamata alla funzione find_score che assegna i punteggi seguendo la tattica descritta poco fa.

Alla funzione viene sempre passata una copia del campo temporanea, quest'ultima sarà ridefinita ogni qual volta sia necessario, perché a causa delle mosse effettuate il campo si modificherà e dovremmo quindi ritornare allo stato precedente per la successiva chiamata ricorsiva.

2. Suddivisione del lavoro

Una volta valutata la complessità del progetto abbiamo suddiviso il lavoro in vari sottoproblemi di diversa difficoltà scegliendo di affrontare quelli più semplici singolarmente e quelli più complessi in team, andando quindi a convogliare tutte le idee per raggiungere la soluzione.

3. Difficoltà riscontrate

Le principali problematiche sono sorte nell'implementazione della funzione ricorsiva dal momento che la maggior parte delle funzioni implementate fino a quel momento non erano ottimizzate per un utilizzo ricorsivo. Un'altra grande fonte di problemi è stata la gestione della memoria heap.