Relazione a cura di Andrea Giacomello, Carola Pessotto, Serena Zanon

RELAZIONE SUL PROGETTO DEL CORSO 'INTRODUZIONE ALLA PROGRAMMAZIONE'

STRUTTURA DEL PROGETTO

Lo scopo del progetto proposto è quello di ricreare una variante del gioco da tavolo '*Laska*'. Il linguaggio utilizzato per realizzarlo è ANSI C.
In testa al codice sono state definite delle costanti, per consentire una lettura migliore, e tre struct: due per implementare il campo da gioco e una per realizzare una lista di possibili mosse che un giocatore può eseguire.

Struct:

- **Struct pawn**: definisce il colore e lo stato di una pedina (ufficiale o soldato);
- **Struct col**: definisce lo spazio di una torre composta da un massimo di tre pedine, memorizzando il numero di pedine effettivamente presenti al suo interno;
- **Struct moves_list**: definisce le possibili mosse attuabili turno per turno, imponendo l'obbligatorietà alle mosse mangiata;

La scacchiera è definita da 49 struct col: le struct corrispondenti alle caselle bianche del campo da gioco avranno al loro interno un array di tre struct pawn, mentre le struct corrispondenti alle caselle nere avranno un puntatore a NULL.

Funzioni:

Qui di seguito sono elencate le funzioni di maggiore rilevanza nel codice.

- **void simple_shift (col_t* board, int x0, int y0, int xd, int yd);**

  Copia la torre situata nelle coordinate (x0, y0) nelle coordinate di destinazione (xd, yd), aumenta la size della torre nelle coordinate di arrivo, azzera la size della torre nelle coordinate di partenza e libera lo spazio occupato da quest'ultima, essendo ora vuota.

- **void capture (col_t* board, int x0, int y0, int x1, int y1, int xd, int yd);**

  Se la size è diversa da 3, copia la prima pedina della torre mangiata, situata in coordinate (x1, y1), nell'ultima posizione della torre mangiante in coordinate (x0, y0), aumenta la size della torre mangiante e diminuisce la size della torre mangiata.
  Se ci sono altre pedine nella torre mangiata, vengono spostate di una posizione avanti, altrimenti viene liberato lo spazio di tale torre.
  Si esegue lo spostamento semplice della torre mangiante verso la casella vuota in coordinate (xd, yd).

- **moves_list_t *insert_simple_shift (moves_list_t* l, col_t* board, int x0, int y0, int xd, int yd);**

  Crea una cella per inserire una mossa che comporta uno spostamento semplice e conseguente inizializzazione dei campi della struct moves_list..

- **moves_list_t *insert_capture (moves_list_t* l, col_t* board, int x0, int y0, int x1, int y1, int xd, int yd);**

  Crea una cella per inserire una mossa obbligatoria (mangiata) e conseguente inizializzazione dei campi della struct moves_list.

- **moves_list_t* moves_analysis (moves_list_t* l, col_t* board, int player);**

  Individua le possibili mosse eseguibili scorrendo il campo da gioco cella per cella. Nel caso in cui ci siano le condizioni per eseguire una determinata mossa essa verrà inserita nella lista di mosse di quel giocatore mediante l'ausilio delle funzioni 'insert_simple_shift' e 'insert_capture'.

- **moves_list_t* filter (moves_list_t* l);**

  Data una lista di mosse in input ricava solo quelle obbligatorie. In caso non ve ne siano, restituisce la lista originale.

- **void check_and_do_promotion (col_t* board, int to_print);**

  Controlla la prima e l'ultima riga della scacchiera per verificare se è necessario promuovere delle pedine da soldato a ufficiale. La promozione avviene con il cambio di stato della singola pedina.

- **void matrix (col_t* board) ;**

  Crea il campo iniziale composto da struct col e struct pawn.

- **int evaluation (col_t* board);**

  Esegue una valutazione della scacchiera in base alla presenza in cima alle torri di pedine nere o bianche. Al termine restituisce un valore che risulta positivo se il numero di pedine bianche supera quello di pedine nere, negativo altrimenti.

- **int minimax (col_t* board, int depth, int player);**

  Implementa l'algoritmo 'minimax' in modo ricorsivo. In questa modalità di gioco, le pedine bianche appartengono al computer. Lo scopo di tale funzione è di minimizzare la massima perdita possibile per l'intelligenza artificiale, analizzando tutte le possibili combinazioni di mosse. In input riceve un puntatore a una scacchiera (riferita alla mossa che si sta analizzando), la profondità (posta a 5 di default) e il giocatore (computer o utente).

- **int macro_ai (moves_list_t* l, int depth, int player);**

  Scorrendo la lista mosse del computer, individua quella più vantaggiosa invocando la funzione 'minimax'.

Spiegazione del 'main'

All'inizio del **'main'** si alloca in memoria dinamica lo spazio per il campo da gioco e si definisce una lista di mosse che viene posta a NULL. Viene invocata la funzione 'matrice' che inizializza la scacchiera con le struct col e le struct pawn. Tramite un ciclo while si chiede all'utente di scegliere una modalità di gioco: '1' se vuole giocare contro l'intelligenza artificiale (computer), '2' se vuole giocare contro un altro utente. La condizione di uscita del ciclo è che l'input sia valido (1 o 2).

Scelta la modalità, si procede all'esecuzione del gioco vero e proprio.

Se l'utente ha deciso di giocare la propria partita contro il computer si entrerà nel corpo dell'if dedicato a tale modalità. La partita si basa su un ciclo while che definisce l'alternarsi dei turni di gioco. A ogni turno viene stampata la scacchiera aggiornata con le mosse effettuate. Il primo a muovere è il computer la cui lista mosse è ricavata dalle funzioni

‘moves_analysis’ e ‘filter’ e successivamente verrà passata alla funzione ‘macro_ai’ che invocherà ‘minimax’. Al termine di ‘macro_ai’ viene restituito il numero della mossa la cui previsione avvantaggia le pedine bianche, ossia il computer. La scacchiera di tale mossa, dopo il controllo di eventuali promozioni e il cambio turno, viene stampata a schermo all’utente. Poi verrà visualizzata la lista di possibili mosse che quest’ultimo può eseguire, anch’essa passata precedentemente in input alle funzioni ‘moves_analysis’ e ‘filter’. Dopo aver selezionato la mossa desiderata, si procede al controllo delle promozioni delle pedine, al cambio turno e si continua in questo modo finché uno dei due giocatori non ha più mosse a disposizione e il programma termina automaticamente.

Se la scelta effettuata è ‘2’, il programma entra nel corpo dell’else che, come per la modalità ‘1’, ha un ciclo while le cui istruzioni sono simili a quelle sopra descritte con l’eccezione che non viene richiamata la funzione ‘macro_ai’ dato che si gioca contro un altro utente. Per questo motivo si ha un susseguirsi delle funzioni di ‘moves_analysis’ e ‘filter’ seguite dalla scelta della mossa e della stampa della scacchiera corrispondente. Il ciclo termina come sopra spiegato, ossia quando non ci sono più mosse disponibili.

Prima della fine di ogni iterazione dei cicli, oltre al controllo della promozione delle pedine e al cambio turno, vi è l’eliminazione dello spazio occupato dalla lista di mosse.

Funzioni ausiliarie utili al processo di esecuzione:

- **void copy_pawn (pawn_t* p_dst, pawn_t* p_src);**
- **void copy_board(col_t* src, col_t* dst);**
- **moves_list_t *destroy (moves_list_t* l);**
- **void fill_description1 (char* desc, int x0, int y0, int xd, int yd);**
- **void fill_description2 (char* desc, int x0, int y0, int x1, int y1, int xd, int yd) ;**
- **void print_moves(moves_list_t* l, int* n) ;**
- **col_t* estract (moves_list_t* l, int n);**
- **void field(col_t* board) ;**
- **int max(int n1, int n2) ;**
- **int min(int n1, int n2) ;**
- **char* estract_description (moves_list_t* l, int n);**

SUDDIVISIONE DEL LAVORO SVOLTO

Prima della suddivisione delle tasks si è discusso riguardo le strutture dati da utilizzare per realizzare il gioco. Per la modalità giocatore contro giocatore il lavoro è stato suddiviso fra i tre componenti del gruppo, mentre l’implementazione della modalità giocatore contro l’intelligenza artificiale si è codificata insieme, così come la stesura della relazione.

DIFFICOLTÀ RISCONTRATE

In primo luogo, un difficoltà incontrata è stata quella di comprendere la consegna e decidere quali strutture dati utilizzare per implementare il gioco. Successivamente a una prima stesura del codice è seguito un periodo di debugging impegnativo. In generale, aver lavorato a distanza, nel rispetto delle misure governative per il contenimento della diffusione dell’epidemia da COVID-19, ha reso più complicato lavorare in squadra.