

Progetto X-Tetris

Gruppo 25: Alessandro Cecchin
Matricola: 859869

Corso di INTRODUZIONE ALLA
PROGRAMMAZIONE

Nome docente:
Alvise Spanò

Prefazione

Il mio progetto è composto da due file: `tetrix.c` e `tetramini.h`, non ho usato nessuna libreria esterna e ho creato tutto usando solamente la funzione `printf`, vettori e aritmetica dei puntatori.

Funzione main

La funzione `main` consiste nel chiedere all'utente di scegliere la modalità di gioco con cui vuole cominciare, a livello di codice questo consiste nello scegliere tra un tipo enumerato chiamato `giocatori_t` e rappresenta le varie modalità: `SINGLE_PLAYER` per giocare da soli, `MULTI_PLAYER` per la modalità a due giocatori e `CPU_PLAYER` per giocare contro la CPU.

Successivamente la funzione esegue una serie di azioni che rappresentano il menu di interazione, questo permette di uscire e terminare il programma, visualizzare i pezzi ancora disponibili oppure di selezionare il tetramino che si vuole utilizzare. Questo innesca poi tutte le funzioni successive. Continua fino a quando la variabile `fine_gioco` non diventa positiva e stampa messaggio di errore quando il codice inserito dall'utente è sbagliato. Per capire la struttura è importante conoscere la logica della costruzione del campo di gioco e del tetramino.

Cosa Succede quando scelgo di selezionare un tetramino?

Viene chiesto all'utente di selezionare la colonna e viene controllato se la colonna selezionata è corretta. il ciclo poi continua e viene chiamata la funzione `scelta` (riga 620 `tetrix.c`). Questa funzione ci chiede poi di selezionare uno dei tetramini e tramite un'altra funzione (riga 419 `tetrix.c`) la rotazione che vogliamo dare al nostro tetramino. In base a quanto abbiamo selezionato la funzione restituirà un puntatore che punterà al primo elemento del vettore da noi scelto. Questo tetramino poi verrà mandato alla funzione `salva_tetramino` (riga 338 `tetrix.c`) per essere “costruito”.

Metodo di costruzione del campo di gioco

Lo spazio rettangolare in cui si svolge la partita è stato pensato come una serie di riquadri che uno accanto all'altro, formano il campo da gioco.

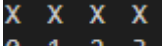
Ogni casella è stata pensata per avere solo due possibili valori `OCCUPATO` o `VUOTO`, per questo ho dichiarato un tipo enumerato chiamato `riquadro` (riga 32, file `tetrix.c`).

Dopo aver dichiarato il tipo “`riquadro_t`”, ho creato un puntatore chiamato “`*campo_di_gioco`” che punterà allo spazio rettangolare del giocatore 1 o giocatore 2 (riga 46, file `tetrix.c`). Per questo all'inizio della funzione `main`, ho dichiarato due campi di gioco utilizzando la funzione `malloc` per allocare lo spazio per un vettore. Ho avuto bisogno di utilizzare l'operatore `sizeof` per allocare la quantità necessaria per ogni elemento di questo vettore, cioè `RIGHE (15) * COLONNE (10) * sizeof(riquadro_t)` la grandezza di una singola casella.

Metodo di costruzione del tetramino

Nel gioco del tetris sono presenti sette tipi di tetramino I, J, L, O, S, T e Z. Il mio progetto rappresenta il tetramini con un vettore di quattro elementi e ogni tetramino può avere differenti vettori in base alla sua rotazione.

Per esempio, il tetramino I può essere posizionato in orizzontale o verticale, per questo all'interno del file tetramini.h (dove sono presenti tutti i vettori di tutti i tetramini), ho inserito il vettore $I_size = \{1,1,1,1\}$ che indica il tetramino in posizione neutra quella orizzontale e $I_90[size] = \{7,0,0,0\}$ che indica il tetramino ruotato di 90 gradi cioè verticale.

 $I_size = \{0, 1, 2, 3\}$ il primo elemento del vettore è 1, questo vuol dire che è presente una singola casella occupata, il secondo elemento del vettore è sempre 1 e si trova a destra del primo elemento ed è sempre una singola casella occupata.

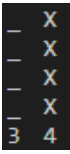


Cosa significano i numeri all'interno dei vettori?

0= indica che in quella posizione il tetramino non occupa la colonna, in questo caso tutta la logica di gioco non conta quella colonna e viene saltata. 1= una sola casella occupata; 2= sono due caselle occupate e sovrapposte; 3= tre caselle occupate e sovrapposte; 4= 1 casella vuota e sopra di essa una casella occupata; 5= due caselle vuota e sopra di esse una casella occupata; 6= una casella vuota e sopra di essa due caselle occupate e infine 7= 4 blocchi sovrapposti uno all'altro (usato solo da I_90).

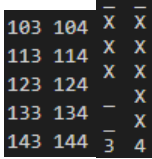
Esempio grafico: $J_180[4] = \{4 = \begin{smallmatrix} X \\ - \end{smallmatrix}, 4 = \begin{smallmatrix} X \\ - \end{smallmatrix}, 2 = \begin{smallmatrix} X & X \\ - & - \end{smallmatrix}, 0 = \begin{smallmatrix} - \\ - \end{smallmatrix}\} \rightarrow \begin{smallmatrix} X & X & X & X \\ - & - & - & - \end{smallmatrix}$

Una delle difficoltà riscontrate per questa mia scelta è stato calcolare il punto di appoggio di ogni tetramino infatti, per simulare una caduta dall'alto ho creato una funzione apposita chiamata contatto (riga 283, file tetrix.c) essa viene chiamata all'interno della funzione salva_tetramino (riga 338, file tetrix.c) che si occupa di gestire tutta la logica descritta qui sopra.

La funzione contatto analizza tutta la colonna scelta dal giocatore in verticale e poi con un ciclo for innestato controlla orizzontalmente le tre colonne successive, questo perché un tetramino al massimo occupa quattro colonne. Non appena il ciclo trova una singola casella occupata, il valore contatto viene restituito in base al numero contenuto all'interno del vettore tetramino.

Esempio grafico:  Selezione la colonna 3 e il tetramino J_90 , il risultato  dovrà essere questo.

Per questo la funzione contatto appena arriva alla posizione 113 (calcolata con $\text{campo_giocatore}_1[\text{colonna selezionata} + \text{size}]$ dove colonna selezionata aumenta di 10 ad ogni ciclo per passare alla riga sottostante), trova la prima casella occupata allora il punto di contatto è uguale a 123 cioè colonna selezionata - COLLONE perché il tetramino viene costruito sempre

partendo da sinistra. 

Logica della CPU

Ho implementato anche la possibilità di giocare contro la CPU. Non ho per motivi di tempo introdotto nessuna logica complessa ma una molto semplice.

Ho utilizzato la funzione rand per generare un numero pseudo casuale che genera un numero dispari o pari con uguale probabilità, per questo ho creato una funzione rand50 che restituisce 0 o 1 in ugual misura per simulare una probabilità del 50 per cento.

Partendo da questo ho creato una seconda funzione rand87 che confrontando in “and” 3 chiamate alla funzione rand50 genera una probabilità dell’87.5 per cento di ottenere un 1 e 0 al 12%. Questo perché $0.5 * 0.5 * 0.5 = 0.125$ la probabilità di ottenere uno zero.

La cpu quindi la maggior parte delle volte seguirà delle mosse predefinite atte a farle fare 2 punti (riga 1002, file tetrax.c) e qualche volta invece inserisce un tetramino casuale.

Modifiche rispetto a consegna originale e libere scelte

Ho seguito alla lettera la consegna, tranne per il fatto che anche l’uscita dalle colonne prevede una sconfitta. Questo perché l’implementazione pensata permette di scegliere colonna, rotazione e tetramino di volta in volta e quindi per esempio selezionare colonna 8 e inserire un tetramino I_ che occupa 4 colonne comporta una negligenza che si paga con la sconfitta.

Ho notato inoltre che nel caso in cui uno dei giocatori completi 3 o 4 righe (innesca la funzione inverti_campo_di_gioco riga 920) e il campo avversario è per lo più vuoto, questo genera un “autogol” perché le righe avversarie vuote diventeranno piene generando quindi dei punti. Ho lasciato la cosa così pensando fosse interessante.

Ho inserito una funzione stampa_anteprema (riga 78) che stampa una piccola anteprima del tetramino in posizione base, con al di sotto le colonne che andremo ad influenzare. Questo perché sono cosciente del fatto che l’implementazione è molto meccanica e non presenta colori o altre carinerie.

Vittoria e sconfitta

La vittoria o la sconfitta viene gestita dalla funzione Controlla_vittoria, che viene chiamata ad ogni ciclo del main. Questa controlla se il punteggio dei uno dei due giocatori supera i 50 punti o se uno dei due è uscito dal campo di gioco, controlla anche lo stato dei tetramini e se sono finiti incorona il giocatore con il punteggio più alto.

La verifica dell’uscita dal campo viene effettuata dalla funzione verifica_uscita (riga 157) e viene chiamata all’interno della funzione contatto, questa per quanto riguarda le colonne occupate è molto semplice perché se il vettore del tetramino è diverso da 0 questo occupa una colonna. Per quanto riguarda il numero di righe occupate invece la situazione è più complicata, questo perché in base a dove il tetramino poggia occupa un diverso numero di righe. Guardiamo l’esempio fatto nella pagina precedente, L_90 occupa 3 righe se poggia su una superficie piana, ma come nell’immagine questo poggia su un’altra colonna di tetramini e per questo occupa solamente una riga.

Quindi ho inserito una serie di if che verifica quante righe sottrarre a quelle già disponibili, nel caso le righe disponibili siano minori di 0 allora la variabile globale Perdita_uscita_campo sarà impostata a TRUE.

