RELAZIONE PROGETTO INTRODUZIONE ALLA PROGRAMMAZIONE

NOME PROGETTO: X-Tetris **Enrico Zanotto 887566**

Descrizione generale del progetto

Questo progetto è stato sviluppato in più file **.c** e **.h** dove è stato suddiviso il codice sorgente.
Per la compilazione del programma è stato creato un **Makefile** contenete le regole che si occupano di assemblare compilare e linkare i vari file sorgente. Per compilare il programma ed ottere l'eseguibile è necessario quindi inserire il comando "make".

IMPORTANTE: per lo sviluppo del programma si è fatto uso di metodi appartenenti alla libreria termios.h per rendere l'input utente fluido e più veloce. Ciò però **preclude** al programma la possibilità di essere compilato ed eseguito su sitemi **non** conformi allo standard **POSIX** come Windows.

All'avvio del programma, per prima cosa, viene chiesto all'utente in che **modalità** vuole giocare.
Le scelta può essere fatta tra **SinglePlayer**, **Multiplayer**, **CPU.**
Una volta che l'utente ha scelto, viene generato il menù di gioco e il campo (o i campi) da gioco, con l'ausilio delle funzioni grafiche presenti nel file grafica.c.
All'utente viene quindi chiesto di scegliere nel menù di gioco il **pezzo** da posizionare per primo.
La scelta può essere fatta utilizzando le freccette direzionali e confermando con il tasto invio.
Il pezzo viene quindi generato all'interno del campo da gioco e viene data la possibilità all'utente di **spostarlo** e ruotarlo nella posizione desiderata. Una volta scelta la locazione del pezzo è necessario premere invio per confermare il posizionamento. A questo punto nel caso della modalità multiplayer il turno viene dato all'altro giocatore e il tutto si ripete.

Suddivisone progetto

Il progetto è stato suddiviso in più file in modo da raggruppare insieme le funzioni che gestiscono operazioni simili:

- "eventiPartita.c"→ in questo file sono racchiuse le funzioni che rilevano l'input utente e agiscono sul piano di gioco di conseguenza;
- "grafica.c" → contiene tutte le funzioni grafiche e di stampa che permettono l'interazione con l'utente e la visualizzazione dello stato della partita;
- "funzioniCpu.c" → file contenente l'algoritmo che permette al computer di sfidarsi contro i giocatori;
- "headers.h" → file dove sono presenti tutte le firme di funzioni, le structs usate con la relativa descrizione;
- "varGlobali.c" → contiene le variabili globali usate nel programma come il centro schermo e i pattern di default dei vari pezzi

Definizione delle strutture

- **"Partita"**→ questa è la struct più importante e contiene informazioni vitali per il funzionamento della partita. Informazioni come la modalità di gioco scelta, il turno attuale di gioco, punteggi di giocatori e pezzi disponibili rimasti.
  La struct partita contiene inoltre due matrici di interi da 15x10 che rappresentano i campi da gioco in ogni istante della partita, ogni intero presente nelle matrici può assumere 3 valori in base a cosa si trova in quella posizione:
  0 - implica che in quella posizione il campo è libero
  1 - implica che in quella posizione il campo è occupato da un pezzo che il giocatore ha inserito in precedenza
  2 - implica che in quella posizione il campo è temporaneamente occupato da un pezzo che il giocatore sta muovendo

- **“Pezzo”** → questa struct ci permette di tenere traccia di un pezzo dal momento della sua creazione fino al suo posizionamento finale sulla mappa. La struct possiede informazioni come le **coordinate** in cui il pezzo si trova e la sua **disposizione**.
  Quest'ultima è una matrice di dimensione dipendente dal tipo di pezzo (2x2 3x3 o 4x4) che rappresenta l'effettiva disposizione dei “blocchi” che lo compongno, essa viene infatti utilizzata come **maschera** per “stampare” il pezzo sul campo di gioco.
  Il contenuto di questa matrice può variare durante il posizionamento del pezzo poiché l'utente può decidere in qualsiasi momento di effettuare una rotazione.

Regole del gioco

Il numero di pezzi disponibili è fisso a **20** per tipo ed entrambi i giocatori selezionano i pezzi dalla stessa pila. Il numero di pezzi restanti è visibile dal menù di gioco.
Il punteggio dei giocatore incrementa in proporzione alle righe rimosse: 1 riga 1 punto, 2 righe 3 punti, 3 righe 6 punti, 4 righe 12 punti. I punti sono sempre visibili nel menù di gioco.
Nel momento in cui il numero di righe rimosse sia di 3 o 4, il corrispondente numero di righe nel campo dell'avversario viene **invertito**.
Una partita termina solo nel caso tutti i **pezzi finiscano** o nel caso uno dei giocatori posizioni un pezzo nell'**area di gioco vietata** corrispondente alle righe 0 e 1.

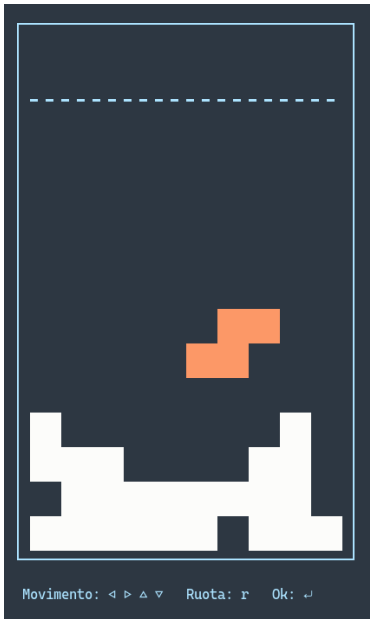
Algortimo per giocatore vs CPU

Per permettere all’ utente di giocare contro il computer è stato implementato un algoritmo di ricerca mossa **ottimale** che analizza la mappa e i blocchi a disposizione ed esegue il posizionamento di un pezzo in modo da occupare le righe della mappa il più velocemente possibile.

Le funzioni che si occupano di questo si trovano nel file funzioniCpu.c e sono le seguenti:

eseguiMossaCPU:	funzione da chiamare per far eseguire la mossa al pc
miglioreMossa:	esegue effettivamente il calcolo della mossa migliore da fare
calcolaPuntaggioScenario:	funzione per la valutazione dello stato della mappa

Il calcolo della migliore mossa viene in pratica fatto testando ogni possibile mossa eseguibile dal computer e dando un “**punteggio**” ad ogni nuovo scenario che viene a crearsi come conseguenza.
Testare ogni mossa significa usare ogni pezzo in ogni rotazione e in ogni posizione legale (ovvero non “fluttuante” nel vuoto)



Per ogni mossa fatta la funzione per il calcolo del punteggio esegue una stima partendo da **100 punti** iniziali e decrementando il punteggio dello scenario in base a **2 fattori** che influenzano negativamente lo scenario:

1. Per ogni riga che la nostra pila di blocchi occupa sulla mappa viene rimosso dal punteggio **un punto** evitando cosi di costruire troppo in altezza rischiando la sconfitta
2. Per ogni “buco” (ovvero un blocco 1x1 non occupato) lasciato in una riga già parzialmente occupata vengono rimossi **3 punti** + **altri 2** se quel buco non è raggiungibile (ovvero è circondato da altri blocchi)

Ogni volta che si trova uno scenario con un punteggio maggiore vengono salvati i dati riguardanti la mossa fatta come rotazione, tipo di pezzo e posizione per poter poi eseguire effettivamente la mossa al termine della ricerca.

Ulteriori considerazioni

L'algoritmo utilizzato non è quindi un algoritmo ricorsivo tuttavia, perchè esso diventi tale basterebbe (oltre ad altri piccoli cambiamenti) eseguire una nuova chiamata ricorsiva per ogni scenario di gioco che creiamo. In questo caso il punteggio andrebbe calcolato unicamente al termine della ricorsione ovvero nel momento in cui si raggiunge un dato livello **n** di **profondità.**

Il fondamentale problema da me incontrato è stato proprio quello di trovare una strategia ricorsiva con una **complessità temporale** ragionevole che possa permettere alla macchina di esaminare gli scenari con un livello di profondità che porti dei vantaggi concreti.

Tuttavia la quantità di mosse possibili a ogni turno rende questo molto difficile. Per ogni livello analizzato, infatti, il numero di scenari creati si moltiplica per un fattore di circa 200 rendendo impraticabile la simulazione di scenari che si trovano a piu di 2 o 3 mosse nel futuro.