Soncin Tommaso Matricola: 890558

SVILUPPO PROGETTO

Il progetto è stato diviso in vari problemi di difficoltà minore:

- Definizione tipi e strutture
- Visualizzare il campo di gioco
- Allocazione e deallocazione
- Mossa del giocatore
- Aggiornamento campo di gioco e punteggi
- Creazione CPU e strategia associata
- Gestione partita

DEFINIZIONE TIPI E STRUTTURE

Il primo passo è stato definire le strutture dati necessarie per il giocatore:
Per il campo di gioco la scelta è ricaduta su un *array flattened* allocato dinamicamente per permettere all'utente di giocare con campi di dimensione non standard (maggiori di 15x10), durante lo sviluppo si è reso necessario un secondo *array flattened* di char, con le stesse dimensioni e tipo del campo di gioco, per implementare la visualizzazione a colori del campo di gioco.
I vari campi necessari per la board sono infine stati incapsulati in una *struct* per semplificare lo sviluppo delle altre sezioni.

Per quanto riguarda i tetramini, anche in questo caso, è stata utilizzata una *struct* per incapsulare i vari membri necessari per il corretto funzionamento.
Ogni tetramino ha dimensione variabile per evitare di allocare memoria non necessaria.
La forma (di default) dei tetramini è stata salvata anche essa come *array flattened* di *char*.
Inoltre, è presente un carattere utilizzato per decidere quale colore associare ad ogni tetramino.
La logica è stata salvata in un file a parte per semplificare la fase di test delle varie funzioni.

Durante lo sviluppo, si è resa necessaria una *struct* (cpuMove) per contenere la scelta della CPU su tetramino da usare, rotazione e colonna.

È presente infine un file "di configurazione" dove è possibile modificare il carattere utilizzato per impostare un blocco (del tetramino e del campo) come pieno oppure vuoto.
Nello stesso file inoltre sono presenti varie *ANSI Sequence* per permettere la visualizzazione dei colori da terminale.

VISUALIZZARE IL CAMPO DI GIOCO

Per permettere la visualizzazione del campo di gioco è stata creata la funzione *printGame(...)* dove i parametri sono i giocatori e una variabile per decidere o meno se stampare anche il campo del secondo giocatore (nel caso di partita single player sarebbe superfluo stampare il secondo campo).
È stata inoltre implementata la funzione *printPieceHint(...)* che mostra a terminale informazioni sui pezzi ancora utilizzabili e la loro quantità rimasta.

ALLOCAZIONE E DEALLOCAZIONE

Per rendere trasparente al chiamante l'allocazione e deallocazione di memoria sono state create due funzioni apposite:

- *startGame(...)*
  Questa funzione alloca la memoria necessaria per i campi di gioco e i tetramini.
- *endGame(...)*
  Dealloca la memoria utilizzata dai giocatori e dai tetramini, inoltre vengono stampati punteggi e eventuale vincitore (se multigiocatore).

Inoltre, durante la rotazione del tetramino, viene allocata (e successivamente deallocata) della memoria per facilitarne la rotazione.

MOSSA DEL GIOCATORE

All'inizio di ogni turno, se non si è selezionata la modalità CPU vs CPU, viene richiesto al giocatore corrispondente di selezionare pezzo, rotazione e colonna nella quale inserire il tetramino scelto. Tutto ciò viene effettuato nella funzione *getUserInput(...).* Successivamente, questi valori verranno utilizzati rispettivamente in una delle due funzioni sottostanti:

- *singlePlayerTurn(...)* se la modalità selezionata è giocatore singolo
- *multiPlayerTurn(...)* negli altri casi

Entrambe le funzioni sono *stub* di *playerTurn(...)* una funzione utilizzata per effettuare il turno sia del singolo giocatore, sia nel caso multiplayer.

L'inserimento del tetramino nel campo di gioco è regolato da tre funzioni:

- *findFree(...)*
- *isLegalMove(...)*
- *insertPiece(...)*

In ordine:

1. Inizialmente viene creata una *deep copy* del tetramino passato in input per poterla eventualmente rotare senza modificare il tetramino originale.
2. Ruotato il tetramino (opzionale)
3. Vengono controllati i limiti di altezza e larghezza per evitare di inserire pezzi non corretti / accedere a memoria non allocata.
4. Viene controllata ogni riga del campo di gioco per verificare se è possibile piazzare il tetramino partendo da quella riga. Alla prima riga non valida, la funzione modifica i parametri *freeRow* e *freeCol* con gli ultimi valori "legali" per inserire un tetramino.
5. Se tutti i controlli precedenti sono andati a buon fine, viene inserito il tetramino e successivamente viene decrementato il numero di tetramini disponibili per quella tipologia.

AGGIORNAMENTO CAMPO DI GIOCO E PUNTEGGI

Se l'inserimento del tetramino è andato a buon fine, allora vengono eseguite in ordine le seguenti funzioni:

- *decreaseQty(...)*
- *removeRows(...)*
- *updateGame(...)*
- *flipRows(...)* (solo se multiplayer)
- *updateScore(...)*

In ordine:

1. Viene decrementato il numero di tetramini disponibili per quella tipologia
2. Viene controllata ogni riga del campo di gioco e, se piena, viene eliminata.
3. Per ogni riga eliminata viene incrementata una variabile ausiliaria utilizzata per tre motivi:
   -Aggiornamento del punteggio giocatore.
   -Inverte le ultime righe del campo di gioco avversario (se rispettate condizioni iniziali).
   -Aggiornamento punteggio per numero di righe eliminate.
4. Se sono state eliminate delle righe, viene aggiornato il campo di gioco del giocatore che ha effettuato il turno sovrascrivendo il contenuto della riga attuale con quella precedente.
5. Se sono state rispettate le condizioni necessarie, vengono invertite le ultime righe del campo di gioco avversario (ovverosia se il campo di gioco in una casella era vuota, dal turno successivo quella casella conterrà un pezzo di tetramino).
6. Viene aggiornato il punteggio con valori definiti da progetto.

CPU E STRATEGIA

Per quanto riguarda la strategia della CPU, è stata implementata una strategia "semi-casuale" implementata nella funzione *cpuDecision(...)*

Viene scansionato il campo di gioco in cerca della "migliore" colonna, ovverosia la colonna che contiene il minor numero di caselle occupate da pezzi di tetramino. Successivamente, grazie alla funzione *rand()* vengono scelti in modo casuale tetramino e rotazione, viene controllato se sono presenti pezzi a disposizione per il tetramino dato e in caso contrario si sceglie nuovamente un tetramino casuale.

Le variabili precedenti vengono incapsulate nella *struct* cpuMove, nominata in precedenza.

GESTIONE PARTITA

Prima di tentare di effettuare una mossa, viene controllato se è possibile continuare la partita analizzando la quantità di tetramini rimasti. Se non sono più presenti tetramini, il gioco si conclude tramite la funzione *endGame(...)* decretando eventualmente un vincitore. Altrimenti, il gioco continua fino a quando una delle seguenti condizioni si verifica:

- Il giocatore tenta di inserire un pezzo (volontariamente) al di fuori dei limiti di altezza e/o larghezza (per esempio il giocatore prova ad inserire il tetramino 'I' con rotazione default nell'ultima colonna)
- Il giocatore tenta di inserire un pezzo senza averne più a disposizione
- Non sono più presenti pezzi a disposizione del giocatore

PRINCIPALI DIFFICOLTÀ

- Gestione *edge case* aggiornamento campo di gioco in *updateGame(...)*
  Nel caso in cui eliminando una o più righe il campo risulti completamente vuoto, la partita non andava avanti.
- Occasionale inversione di parametri nella formula per accedere ad un *array flattened* come se fosse un array 2D
  In questo modo le funzioni che hanno come parametri *array flattened* non producevano il risultato sperato, con alle volte inserimenti di pezzi non corretti oppure aggiornamento del campo di gioco parziale...

POSSIBILI MIGLIORIE

- Nascondere l'implementazione delle *struct*
- Implementare minimax per la mossa della CPU