RELAZIONE PROGETTO DONALD GERA 892604

Suddivisione del progetto nei file
Il progetto si articola in 5 file
main.c : Questo file contiene la funzione main, che rappresenta l'entry point del programma. Si offre all'utente la possibilità di scegliere tra la modalità interattiva o l'esecuzione di algoritmi AI per risolvere un labirinto ricevuto in input. La scelta viene effettuata tramite l'input dell'utente e il programma eseguirà l'azione corrispondente alla modalità selezionata.

my_functions.h : Insieme di funzioni e di strutture dati che vengono utilizzate per gestire il labirinto, trovare percorsi ottimali e consentire l'interazione con il giocatore.

my_functions.c : Implementazioni delle funzioni definite in my_functions.h e costanti globali

heap.h : Questo file fornisce una struttura dati e delle funzioni per gestire un max heap binario che contiene oggetti maze_t (struttura dati utilizzata per modellare i labirinti). Questo tipo di heap è utilizzato per mantenere ordinati gli oggetti in base al loro punteggio, consentendo di estrarre rapidamente il labirinto con il punteggio più alto.

heap.c : Contiene l'implementazione delle funzioni definite in heap.h

Fondamenta del lavoro e sfide principali
Ho lavorato in modo individuale su questo progetto, affrontando diverse difficoltà e apprendendo da una varietà di fonti per portarlo a termine con successo. Una delle sfide che ho incontrato è stata quella di riuscire a passare dalla teoria alla pratica. Sebbene avessi una buona comprensione teorica di strutture dati e algoritmi avanzati, implementarli in un progetto concreto richiedeva attenzione e adattamento al problema specifico. La difficoltà principale che ho affrontato è stata quella di strutturare il progetto iniziale in modo accurato e ben ponderato. Ho dovuto considerare come organizzare il codice, definire le principali funzionalità, le strutture dati necessarie e pianificare come avrei gestito le dipendenze tra le varie componenti.
Una conseguenza naturale di questa fase iniziale è stata la riscrittura del codice su più occasioni soprattutto perché inizialmente mi ero concentrato subito sul definire una strategia risolutiva senza prima preoccuparmi di avere una infrastruttura su cui costruire l'algoritmo risolutivo. Ho trovato una grande gratificazione dopo l'aver realizzato lo "scheletro" del progetto, poiché questo mi ha permesso di costruire le varie componenti successive senza dover effettuare riscritture totali del codice. Mi sono concentrato su "porzioni" di codice più piccole, più semplici da testare e mantenere ed il risultato è stato un codice più pulito e un conseguente riutilizzo del codice.

Composizione del progetto
Organizzazione delle strutture dati:
Dopo una fase iniziale di sperimentazione con il progetto, ho compreso l'importanza di fondare le basi del progetto e sviluppare le strutture dati necessarie. In particolare le strutture dati sono necessarie per rappresentare il labirinto e le informazioni ad esse associate e la strategia risolutiva AI_Dijkstra.

Le strutture dati si articolano in:

- *maze_t*: progettata per contenere le informazioni essenziali del labirinto, come la matrice che rappresenta il labirinto, le dimensioni della matrice, le posizioni di partenza, arrivo e attuale dello snake, il punteggio, le monete, i trapani e il percorso svolto fino a quel punto nel labirinto
- *position_t*: contiene una coppia di interi (funzionamento simile a pair) creata per rappresentare le posizioni del labirinto in forma compatta
- *heap_t*: creata per la realizzazione della soluzione AI_Dijkstra che usa un max heap che memorizza e ordina elementi maze_t in base al loro score (spiegato nel dettaglio avanti)

Queste strutture sono state definite in modo da essere facilmente manipolabili attraverso le funzioni definite nel file "my_functions.c".

Strategie risolutive :

- *AI_Recursive*: Questa è la prima soluzione tentata e si tratta di una strategia ricorsiva che percorre tutte le strade possibili e ritorna quella relativa al punteggio maggiore. Ovviamente sono state apportate diverse ottimizzazioni per evitare ricorsioni infinite e per risparmiare tempo di computazione (pur sempre molto elevato) . In particolare si evita, mediante l'utilizzo di una matrice che tiene per ogni cella del labirinto il punteggio massimo raggiunto quando si passa sulla suddetta cella, di risvolgere determinati percorsi inutilmente (se con un altro percorso giungiamo in una posizione che ha un punteggio maggiore rispetto a quello con l'attuale labirinto si interrompe la ricorsione). Inoltre si impedisce a snake di passare più di due volte su una casella avendo constatato che il contrario risulta inutile nella maggioranza dei casi. Le soluzioni derivanti da questa strategia sono pressoché ottimali ma è necessario utilizzare l'algoritmo solamente su labirinti piccoli (max labirinti con 50 celle) altrimenti l'algoritmo risulta estremamente inefficiente.
- *AI_Dijkstra*: Si adotta una strategia iterativa che si basa sull'idea dell'algoritmo di Dijkstra. L'algoritmo di Dijkstra viene utilizzato per trovare il percorso più breve da un punto di partenza a un punto di destinazione in un grafo pesato, dove i pesi rappresentano la distanza tra i nodi. Nel nostro caso, l'obiettivo è trovare il percorso verso l'uscita del labirinto che ci permette di ottenere il punteggio massimo e non il percorso più breve come accade in Dijkstra. L'idea è quella di utilizzare una coda prioritaria implementata mediante un max heap binario contenente i labirinti fino a quel punto esplorati ordinati in base al loro score, che viene calcolato come risultato del numero di passi, monete raccolte e trapani, in particolare allo score iniziale 1000 viene sottratto 1 per ogni passo, si somma 10 per ogni moneta e si somma 2 per la presenza di trapani (valore arbitrario determinato in seguito a vari test), e ad ogni iterazione dell'algoritmo estrarre dalla coda prioritaria il labirinto a cui è associato il punteggio maggiore fino a quel momento. Si compie in questo senso ad ogni passo dell'algoritmo la mossa che sembra migliore in quel momento sperando di avvicinarsi alla soluzione ottimale. La strategia implementata si allontana parecchio dal classico algoritmo di Dijkstra e ne eredità semplicemente l'impostazione della coda di priorità, in particolare non si lavora con grafi e non termina non appena trovata una soluzione ma quando lo heap risulta essere vuoto. In sintesi, si cerca di trovare il percorso (memorizzato all'interno di una stringa ogni qualvolta si arrivi alla fine del labirinto con un punteggio maggiore di quello memorizzato fino a quel momento) che porta al

punteggio massimo all'interno di un labirinto e alla fine si riproduce il percorso trovato.

Conclusione

Sono estremamente soddisfatto dei risultati ottenuti in questo progetto. Nonostante le sfide incontrate, sono riuscito a padroneggiare concetti complessi e a migliorare notevolmente le mie competenze di programmazione.

Questo progetto è stato un'opportunità formativa unica che mi ha permesso di crescere come programmatore e di acquisire una comprensione più profonda di concetti teorici avanzati. Ha inoltre dimostrato la mia capacità di lavorare in modo indipendente su progetti complessi.