Marco Mihai Condrache (896018) – 2/01/2023

Relazione progetto Snake

Introduzione del progetto

```
####################
0      #             $  #
#      #             $  #
#      #!        #   $  #
#      #         #   $  #
#      #         #      _
#                #      #
#      $$$$      #      #
####################
```

La figura sopra rappresenta un labirinto dove il nostro protagonista Snake ‘o’ deve trovare l’uscita ‘_’. Snake percorre il labirinto raccogliendo o evitando gli oggetti nel piano di gioco finché non raggiunge l’uscita.

Per la specifica completa si veda il file relativo.

Analisi del problema

La realizzazione del progetto si concentra sulla realizzazione delle due modalità principali: interattiva e automatica. Nella modalità interattiva il giocatore decide come muovere l’eroe, decidendo il percorso e gli oggetti da prendere.
Nella modalità automatica, il programma decide il percorso che realizza il punteggio migliore.

Entrambe presentano difficoltà matematico-logiche, progettistiche e relative ai limiti del linguaggio di programmazione C.

Modalità interattiva:
- Gestione della lunghezza dell’eroe
- Gestione degli imprevisti e trapano.

Modalità automatica:
- Algoritmo per il percorso più breve
- Algoritmo per il percorso che colleziona più monete
- Utilizzo del trapano
- Aggirare gli imprevisti
- Impedire la perdita di monete a causa di incontri con il proprio corpo

Progettazione dell’algoritmo

Per soddisfare le specifiche del problema bisogna memorizzare informazioni come la posizione del giocatore, la lunghezza del corpo, quante mosse ha fatto ecc...

Per fare ciò si sono definite alcune strutture dati necessarie per il funzionamento del programma:

- Una struttura per il salvataggio del labirinto, le sue dimensioni, e l'indirizzo di memoria dove risiede la matrice con tutti i caratteri.
- Una struttura che mantiene le statistiche del giocatore, la sua posizione, le monete raccolte, i passi effettuati e il vettore contenente il suo corpo.
- Una struttura che rappresenta una posizione all'interno del labirinto insieme a metadati come il costo per muoversi in quella posizione, il costo accumulato fino a quella posizione, la provenienza, il numero di monete raccolte e i trapani a disposizione.
- Una struttura che permetta di definire un percorso, realizzato come un vettore dinamico di posizioni.
- Una struttura che permetta di definire un insieme di percorsi, realizzata come un vettore dinamico di percorsi.

Implementazione

Definite le informazioni di output e di input, possiamo procedere all'implementazione degli algoritmi principali.

Modalità interattiva:

Il corpo del giocatore viene gestito attraverso un vettore lineare (gestito dinamicamente attraverso funzioni di memoria). Ogni volta che l'eroe incontra una moneta ($), un pezzo del corpo viene inserito dopo l'ultimo elemento del vettore. Nel caso in cui la testa dell'eroe incontri una parte del suo corpo, viene persa la parte dal punto di incontro alla fine (si effettua una traslazione a sinistra degli elementi).

Modalità automatica:

L'algoritmo principale di risoluzione è una versione modificata dell'algoritmo A* (riferimento). Sono stati implementati gli algoritmi base di A* e DFS necessari per la funzione di stima e il funzionamento della versione modificata.

L'algoritmo principale si concentra sul numero di monete che è necessario raccogliere (determinato dalla funzione di stima), espandendo ogni volta il percorso che ha il minor numero di passi. Al raggiungimento del numero di monete necessario, si esegue un A* normale dall'ultima moneta raccolta fino all'uscita del labirinto.
Il componente principale è dunque un ciclo che, dato un vettore chiamato "open", seleziona il percorso con il minor numero di passi, lo estende in tutte le direzioni e, se ha raggiunto la fine, lo inserisce in un altro vettore "ended".
Allo scadere del time-out preimpostato o al numero di monete raccolte, viene selezionato il percorso che produce il percorso migliore.
La funzione di stima seleziona ogni moneta all'interno del labirinto, ricerca attraverso il DFS se la moneta è raggiungibile (altrimenti si esegue A* per eventuali trapani) e, attraverso A*, crea due percorsi dal punto iniziale alla moneta e dalla fine alla moneta. Se questi due percorsi possiedono almeno un punto in comune, uno dei due ha preso un imprevisto oppure non esiste un percorso, la moneta viene scartata perché non ritenuta "necessaria". La funzione di stima il 90% dei casi è corretta. Nel caso in cui fosse

sbagliata, l'algoritmo viene eseguito fino a time-out e poi viene selezionato il percorso migliore.
Nota: nel caso in cui la stima non fosse corretta, un alto time-out aiuta a trovare il migliore percorso. 60s e 180/360s per alcuni casi sono valori sufficienti nel 95% dei casi.

Benchmark

La tabella con i risultati del benchmark è disponibile qui.

I benchmark sono stati eseguiti eseguendo la versione finale dell'algoritmo sui labirinti delle challenge su due sistemi diversi: Mac con Chip M1 e Server Linux con processore Intel Xeon E3-1245v5.

Come si può notare il processore Intel ottiene un punteggio leggermente più basso, in quanto su alcuni labirinti ha raggiunto il time-out prima di trovare la soluzione migliore. In questo caso un time-out leggermente più alto avrebbe permesso il raggiungimento dello stesso punteggio ARM.

L'algoritmo è inoltre stato messo a confronto con i risultati di un competitor della challenge, i punteggi sono simili ma le bottle-neck dell'algoritmo gli impediscono di poco di raggiungere quel punteggio (imprecisione della funzione di stima).

Nota: l'algoritmo consegnato durante le challenge è diverso da quello finale.

L'imprecisione della funzione di stima, in questo caso ha sbagliato 3 stime su 18 (dove il tempo è > 20s), allungando il processo di ricerca del percorso. Per rendere l'algoritmo corretto al 100%, dunque coprire ogni caso, bisognerebbe migliorare la funzione di stima ed eseguire un'ottimizzazione profonda (più di quanto sia già stato ottimizzato).

Il problema principale è dato dal fatto che in un problema del genere esistono percorsi in maniera esponenziale, dove oggetti come le monete, i pericoli e i trapani possono modificare il flow, aggiungendo o riducendo il numero dei percorsi.

Labirinti generati

Il programma è anche in grado di generare labirinti, con oggetti random e percorsi random. La dimensione massima consentita, per specifica, dei labirinti è 255 x 255.

L'algoritmo che genera il labirinto è il recursive backtracking dove si parte dalla fine del labirinto e si generano sotto-percorsi in stile DFS.

L'algoritmo è veramente efficiente e in grado di generare labirinti di dimensioni enormi in qualche secondo.