ITTS Vito Volterra

Approccio teorico alla Rescue Maze

Simone Simonella

5^A

A.S. 2015/2016

RESCUE MAZE

La *Rescue Maze* è una delle competizioni organizzate della *Robot Rescue*[1] il cui scopo consiste nella ricerca e nel salvataggio di vittime identificate utilizzando fonti di calore in un labirinto.

Il labirinto non ha ingresso ed uscita e presenta alcuni foyer[2] posizionati casualmente. Vi possono essere presenti delle caselle nere che rappresentano dei burroni, degli ostacoli come bottiglie o mattoni e dei bumper[3] il cui scopo è quello di rendere più complessa la visita del labirinto.

SCOPO

Lo scopo di questo lavoro è la creazione di un simulatore che permetta il confronto tra vari programmi sviluppati facendo in modo di determinare quale tra questi sia in grado di ricavare il percorso più veloce.

Così facendo è stato reso possibile la dimostrazione del funzionamento del programma A7M [4] sviluppato per la partecipazione alla competizione *Rescue Maze*.

METODO DI VALUTAZIONE

I movimenti di avanzamento e di rotazione sono gli unici che richiedono una quantità di tempo considerevole si è deciso di misurare l'efficenza degli algoritmi come la somma degli avanzamenti e delle rotazioni.

ASPETTI TRASCURATI

Dopo un'attenta analisi sono state trascurate tutte le complicazioni risolvibili a livello fisico (detriti, imprecisioni nell'avanzamento e nella rotazione) o semplicemente risolvibili a livello logico (burroni, divisione in vari piani del labirinto). Così facendo è stato possibile rendere più intuitivo ed immediato il confronto tra i vari programmi.

UNITY

Per la realizzazione del simulatore è stato necessario in primo luogo lo studio approfondito dell'ambiente di sviluppo Unity[5] sia nella componente 3D che 2D.

Dato che la realizzazione di modelli 3D avrebbe richiesto una quantità di tempo maggiore (sarebbe stato necessario apprendere il funzionamento dell'ambiente di modellazione 3D Maya o simili) si è deciso di realizzare l'intero progetto in 2D.

---

[1] Robots identify victims within re-created disaster scenarios, varying in complexity from line-following on a flat surface to negotiating paths through obstacles on uneven terrain.

[2] Gruppi di caselle senza muri tra queste.

[3] Ostacoli generalmente di forma cilindrica.

[4] Nome originario del programma installato nel robot, usato per semplificare la comprensione del testo.

[5] Unity Technologies is revolutionizing the game industry with Unity, the breakthrough development platform for creating games and interactive 3D and 2D experiences like training simulations and medical and architectural visualizations, across mobile, desktop, web, console and other platforms. Unity was created with the vision to democratize game development and level the playing field for developers across the globe.

REALIZZAZIONE LABIRINTO

Una volta compreso il funzionamento dell'ambiente di sviluppo nelle sue parti essenziali è stata necessaria una seconda documentazione per quanto riguarda gli algoritmi esistenti usati nella realizzazione dei labirinti.

Dopo un'attenta analisi, l'algoritmo *Depth first search*[6] si è rilevato il più adatto.

L'algoritmo crea un labirinto nel quale sono presenti per ogni casella mura in tutte e quattro le direzioni (nord, sud, est, ovest).

Casualmente viene selezionata e segnata una casella. Fino a quando tutte le caselle del labirinto non sono contrassegnate si seleziona casualmente una delle quattro caselle vicine, se disponibili, la si segna, si cancellano le mura presenti tra le due caselle e questa diventa la nuova casella dalla quale ripetere l'operazione (dopo aver salvato la casella precedente).

Se non sono presenti caselle vicine disponibili si seleziona l'ultima casella salvata dalla quale sono accessibili delle caselle vicine non ancora segnate.

L'algoritmo *Depth first search* si adatta perfettamente allo scopo perché permette la memorizzazione della posizione delle mura e quindi la creazione di una mappa logica.

Da tale mappa è facile effettuare delle modifiche come l'inserimento dei foyer, prima di mostrare graficamente il labirinto (dato che è strutturata come un array bidimensionale x,y bastano due cicli for).

Nel sito web *Flat Tutorials* è disponibile lo pseudo-codice per la realizzazione di questo algoritmo[7].

Una volta compreso e studiato, tale algoritmo è stato riscritto partendo da zero in C#, con il solo aiuto del pseudo-codice riportato.

Tale algoritmo crea un labirinto perfetto[8] ed è quindi stato integrato con un secondo algoritmo, il quale seleziona casualmente delle caselle in cui creare dei foyer.

Da questo secondo algoritmo è possibile scegliere la frequenza di creazione dei foyer semplicemente modificando il valore di una variabile.

---

[6] This algorithm is a randomized version of the depth-first search algorithm. Frequently implemented with a stack, this approach is one of the simplest ways to generate a maze using a computer.

[7] create an array or list to hold a list of last visited cells and call it lastCell,we will use it as a stack.

```
int totalCells = total number of cells present.
int currentCell = 0;
int visitedCells = 0;
currentCell = choose a cell at random.
visitedCells++; increment the value of visitedCells
while visitedCells < totalCells
        find all neighbors of currentCell which haven't visited yet.
        if one or more found
                choose one at random destroy or remove the wall between it and currentCell
                add currentCell location on the lastCell.
                make that random neighbor cell currentCell
                visitedCells++
        else
                get the most recent entry off the lastCell and make it currentCell
        endIf
endWhile
```

[8] Mazes containing no loops are known as "simply connected", or "perfect" mazes

IMPLEMENTAZIONE ALGORITMO A7M

Con il labirinto realizzato è stato possibile dimostrare il funzionamento del programma A7M sviluppato per la competizione *Rescue Maze* di Bari.

Sono state fatte delle modifiche e delle migliorie al codice scritto in precedenza per la competizione.

Il programma era stato scritto in C ed Unity aveva la possibilità di interagire solamente con programmi scritti, in Javascript e C# si è resa necessaria una rivisitazione sostanziale del codice per quanto riguarda le parti di avanzamento e rotazione dato che queste non dovevano più controllare dei motori, ma delle variabili presenti in altri programmi.

FUNZIONAMENTO A7M

Il funzionamento del programma è semplice: fino a quando è possibile avanza, altrimenti svolta a destra o a sinistra, oppure torna all'ultima casella inserita nella lista delle caselle da visitare.

Ogni qualvolta si avanzi, vengono aggiunte alla lista di caselle da visitare le caselle a destra e sinistra (la loro posizione assoluta ricavata dalla posizione attuale e l'orientamento) se non sono ancora state visitate e se non sono presenti mura.

WAVEFRONT

La parte più complessa dell'algoritmo consiste nel tornare indietro, in quanto si potrebbe verificare che ci sia una strada più corta rispetto a quella fatta all'andata. Per trovare questa strada è stata necessaria una terza documentazione inerente agli algoritmi di shortest path finding come quello di *Dijkstra*[9] applicato alla robotica. Il più adatto alla situazione si è rilevato quello denominato *Wavefront*[10].

Dopo un'attenta analisi e comprensione è stata creata una variante la quale permette l'interazione con la mappa che il robot ha salvato in memoria centrale e che aggiorna dinamicamente durante il processo di visita del labirinto.

Il risultato di questo lavoro è stato un algoritmo, il quale conoscendo la posizione attuale e quella alla cui deve recarsi, effettua dei controlli sulle caselle circostanti (nord, sud, est e ovest) e se queste rispettano alcune caratteristiche (sono state visitate, non sono ancora state associate e non ci sono muri tra la casella attuale e queste) a loro volta l'algoritmo fa il controllo delle caselle adiacenti a quelle risultanti alla selezione con gli stessi criteri dopo averle associate. Il tutto fino a quando una delle caselle circostanti non è la casella di arrivo.

A questo punto viene estrapolata dalla lista delle associazioni tra le caselle, creata durante la selezione descritta, il percorso da seguire.

Così facendo il percorso ottenuto non può che essere il più breve possibile.

Questo fatto rende tale algoritmo perfetto nella valutazione degli spostamenti del robot ed è stato implementato nel programma in modo che venga richiamato ogni qualvolta sia necessario effettuare uno spostamento.

[9] L'algoritmo di Dijkstra è un algoritmo utilizzato per cercare i cammini minimi (Shortest Paths) in un grafo con o senza ordinamento, ciclico e con pesi non negativi sugli archi.

[10] Denominazione trovata nel sito in cui è stata compiuta la documentazione

MODIFICA ALL'ALGORITMO A7M

Osservando il funzionamento di tale algoritmo nel simulatore, in labirinti di dimensioni apprezzabili, risulta lampante una possibile miglioria.

Quando sono presenti dei muri sia di fronte che a destra e a sinistra del robot, è possibile che si verifichi il caso nel quale ci sia una casella ancora da visitare più vicina rispetto all'ultima inserita nella lista di caselle da visitare.

Per sopperire a tale carenza è stato creato un secondo programma il quale è identico nelle parti essenziali al primo, ed incorporato un secondo *Wavefront* modificato, il quale una volta avviato identifica la casella più vicina ancora da visitare (sempre conoscendo la posizione attuale) e crea il percorso più veloce per il raggiungimento di questa.

Dopo una serie di test nel simulatore è apparso lampante come quest'ottimizzazione rendesse la visita del labirinto assai più efficiente in proporzione all'aumentare delle dimensioni di questo.

UN'ULTERIORE MODIFICA

In seguito è stata creata una terza variante del programma la quale presenta una sola modifica sostanziale: quando è possibile l'algoritmo fa girare il robot a destra.

Questa modifica apparentemente insignificante cambia il comportamento del robot in maniera radicale. Questo impiega più tempo rispetto alla versione precedente per percorrere il labirinto nella sua completezza, ma presenta (in labirinti di ampie dimensioni) un comportamento peculiare.

Il programma porta il robot ad avere la tendenza di percorrere il perimetro del labirinto fino al raggiungimento del centro per poi tornare a visitare quelle caselle rimaste da visitare. Il fatto che sia più lento rispetto al precedente è determinato dal fatto che nonostante effettui meno avanzamenti, le volte che questo necessita di girarsi lo portano a sopperire alla differenza di avanzamenti ed a superarla.

SITOGRAFIA

- ➔ http://rcj.robocup.org/rescue.html
- ➔ https://unity3d.com/public-relations
- ➔ https://en.wikipedia.org/wiki/Maze_generation_algorithm
- ➔ http://flattutorials.blogspot.it/2015/02/lets-create-perfect-maze-generator.html
- ➔ http://www.societyofrobots.com/programming_wavefront.shtml