



Università Ca' Foscari Venezia

Documento di progetto

Maze&Pacman F#

Corso: Introduzione alla programmazione

Anno Accademico: 2019/2020







Docente: Alvise Spanò

Studentessa: Cantoni Letizia

E-mail: 882713@stud.unive.it



INDICE

Requisiti minimi	3
File di progetto fornito dal professore	
Descrizione progetto	4
 BeforeMenu.fs	4
Animazione iniziale	
Descrizione	
 Menu.fs	5
Gestione del Menù	
Descrizione	
 Main.fs	5
Avvio del gioco	
Descrizione	
 Maze.fs	6
Generazione del Labirinto	
Algoritmo usato e descrizione	
Modalità di gioco Interattiva	
Descrizione ed Istruzioni	
 MazeAutoRes.fs	7
Risoluzione del Labirinto	
Algoritmo usato e descrizione	
 Pacman.fs	8
Spiegazione Animazioni	
Nemici / Movimento nemici	
Monete	
Modalità trapano	
 FinishScreen	9
Schermata finale	
Caso Vittoria/Caso Sconfitta	
Conclusioni	10

REQUISITI

I requisiti minimi per questo progetto sono elencati di seguito per comodità, ma si possono trovare anche nel documento fornito dal docente.

- Il labirinto: lo stesso generato con l'algoritmo previsto dal Task 1, disegnabile tramite uno sprite grande come tutto lo schermo o anche in altre maniere.
- Il giocatore: lo sprite del giocatore è controllabile tramite i tasti WASD allo stesso modo della modalità interattiva del Task 1.
- I nemici: un numero casuale (ma ragionevole) di nemici (equivalenti ai fantasmi di Pac-Man) implementabili tramite piccoli sprite devono muoversi in maniera molto semplice e casuale lungo i corridoi del labirinto. La collisione tra un nemico ed il giocatore provoca un Game Over, quindi la fine della partita.
- Le monete: un numero casuale (ma ragionevole) di monete (equivalenti alle ciliege in Pac-Man) implementabili sempre tramite sprite. Esse sono immobili e disseminate per
- il labirinto: se il giocatore ci finisce sopra la moneta viene raccolta ed 1 punto viene accreditato al giocatore.
- Il trapano: tramite il tasto SPAZIO il giocatore abilita la modalità trapano per 5 secondi. Quando la modalità trapano è attiva il giocatore può bucare i muri, quindi modifica il labirinto. Il trapano ha un tempo di ricarica (o cooldown) di 15 secondi, che devono essere contati dalla fine dei 5 secondi di durata.

Di seguito inserisco il link per consultare l'intero documento fornito dal docente, con la spiegazione del motore grafico ed altri metodi utilizzati per la realizzazione del progetto.

[DOCUMENTO DESCRIZIONE DEL PROGETTO](#)

DESCRIZIONE PROGETTO

BeforeMenu.fs

In questo sorgente viene creata l'animazione iniziale (**Figura 2**), l'idea di base è quella di ricreare lo screensaver utilizzato per i lettori dvd (**Figura 1**).

Per coloro che non avessero presente l'animazione di cui sto parlando, possono visualizzarla al seguente link: <http://sketchbrook.co.uk/dvdSim.html>



Figura 1



Figura 2

Per creare l'animazione ho rivisitato il codice già fornito nella Demo1, Ho utilizzato una lista di sprites che muovo randomicamente all'interno della funzione myupdate.

Menu.fs

In questo sorgente vengono creati 2 menù, uno per permettere all'utente di decidere quale modalità di gioco selezionare (**Figura 3**), e l'altro menù serve per far decidere all'utente la difficoltà del gioco(**Figura 4**).

La difficoltà del gioco va di pari passo con la grandezza del labirinto, dunque alla difficoltà maggiore, è associata la risoluzione della finestra più grande (90*45).

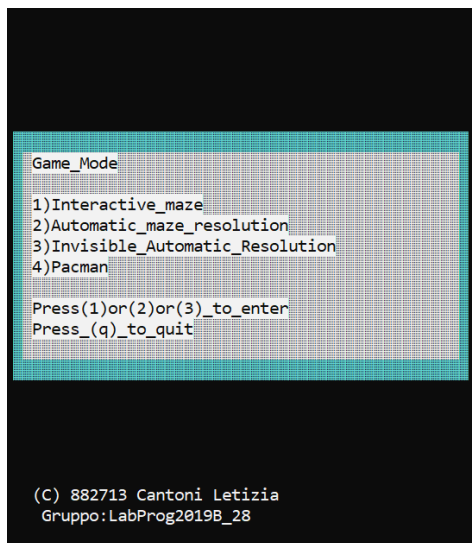


Figura 3

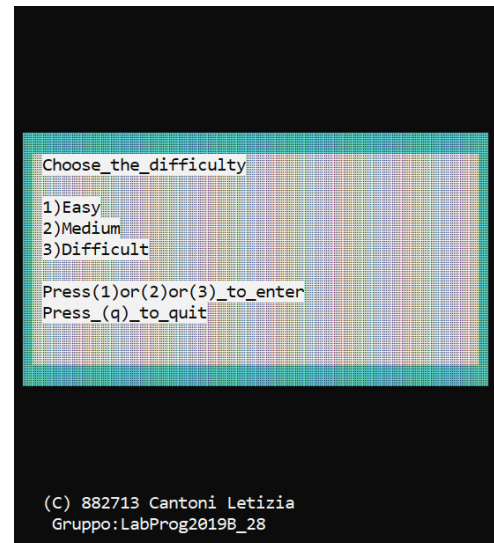


Figura 4

Per fare il menù ho utilizzato principalmente 2 sprites, uno per la scelta e l'altro per inserire i miei dati (matricola - cognome nome - gruppo).

Main.fs

Per quanto riguarda questo file, ho inserito soltanto un ciclo while, all'interno del quale ho richiamato varie funzioni utili all'avvio del gioco, per primo ho chiamato il main del file Beforemenu per avviare l'animazione F#, di seguito ho chiamato il main del Menù, per consentire all'utente di scegliere fra le diverse opzioni, infine ho sfruttato il pattern matching per gestire le varie scelte che l'utente poteva effettuare così da chiamare nello specifico il main della modalità di gioco scelta, passandogli come parametro il livello di difficoltà scelto.

Maze.fs

In questo sorgente è presente l'algoritmo di generazione del labirinto, che richiamerò anche nei file AutoResolve e Pacman.

L'algoritmo di generazione è Algoritmo di Prim randomizzato, per chi non lo conoscesse la documentazione che lo spiega la si può trovare al seguente link: [Algoritmo di Prim randomizzato](#)

Riporto di seguito un breve riassunto:

1. Inizia con una griglia piena di muri.
2. Scegli una cella, contrassegnala come parte del labirinto. Aggiungi i muri della cella all'elenco dei muri.
3. Mentre ci sono muri nell'elenco:
 1. Scegli un muro casuale dall'elenco. Se viene visitata solo una delle due celle che divide il muro, allora:
 1. Trasforma il muro in un passaggio e segna la cella non visitata come parte del labirinto.
 2. Aggiungi i muri vicini della cella all'elenco dei muri.
 2. Rimuovere il muro dall'elenco.

Cercando qualche informazione in internet ho trovato, su una piattaforma, un utente che spiegava perché secondo lui la definizione data da wikipedia andava rivista in alcuni particolari, e mostrava esempi del suo codice per la generazione. Grazie a questo spunto ho implementato l'algoritmo utilizzando il motore grafico fornito dal professore.

Nel caso in cui si volesse visionare il codice da cui ho preso spunto lascio qui il link:

<https://stackoverflow.com/questions/29739751/implementing-a-randomly-generated-maze-using-prim-algorithm>

In questo sorgente viene gestito inoltre lo spostamento da parte del player, così da permettere la modalità di gioco interattiva.

Per spostarsi l'utente deve premere i tasti W (muoversi verso l'alto) A (muoversi verso sinistra) S (muoversi verso il basso) D (muoversi verso destra).

L'utente deve raggiungere la fine del labirinto identificata da un rettangolino rosso (**Figura 5**).

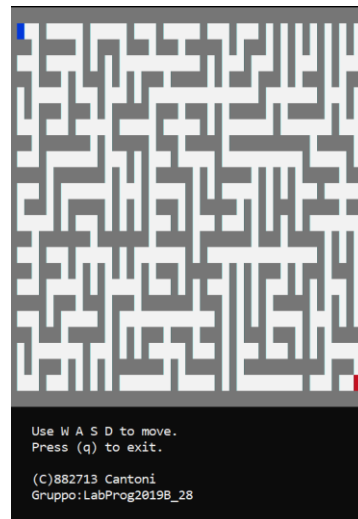


Figura 5

MazeAutoRes.fs

In questo file implementato l'algoritmo di risoluzione del labirinto, l'algoritmo scelto è una rivisitazione del DFS (Depth-first search).

Per riassumere: L'algoritmo inizia dal nodo radice (selezionando un nodo arbitrario come nodo radice nel caso di un grafico) ed esplora il più possibile lungo ciascun ramo prima di tornare indietro, se il percorso è già stato visitato lo evita.

Naturalmente l'algoritmo non mostra il percorso più efficiente, causa le scelte random dello spostamento, ma è una buona soluzione per spostarsi dall'inizio alla fine del labirinto.

Nel main viene richiamata la funzione per la generazione del labirinto che si trova nel file maze, in seguito viene richiamata la funzione per l'avvio automatico della risoluzione.

L'utente per avviare la risoluzione deve premere il tasto S (Figura 6).

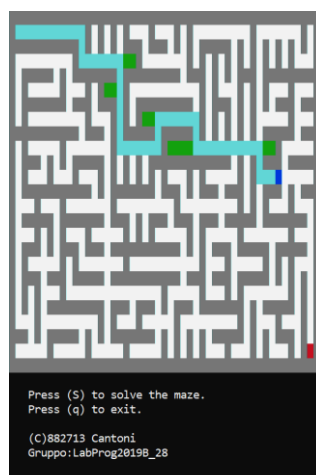


Figura 6

Pacman.fs

Per quanto riguarda questa modalità di gioco, ho cambiato un poco la grafica, per far sì che somigliasse a quella del gioco vero.

In questo file creo una lista di sprites per i nemici ed una per le monete. Le monete rimangono posizionate in modalità statica.

I nemici si muovono casualmente all'interno delle celle libere del labirinto.

Ci sono inoltre 3 sprites sotto il labirinto, uno utilizzato per visualizzare il punteggio, uno per la modalità trapano (drill): viene visualizzato il countdown da 5 a 1, alla fine uno sprite per il cooldown (cooldown): appena finita la modalità trapano comincerà il countdown del cooldown, da 15 a 0 (**Figura 7**).

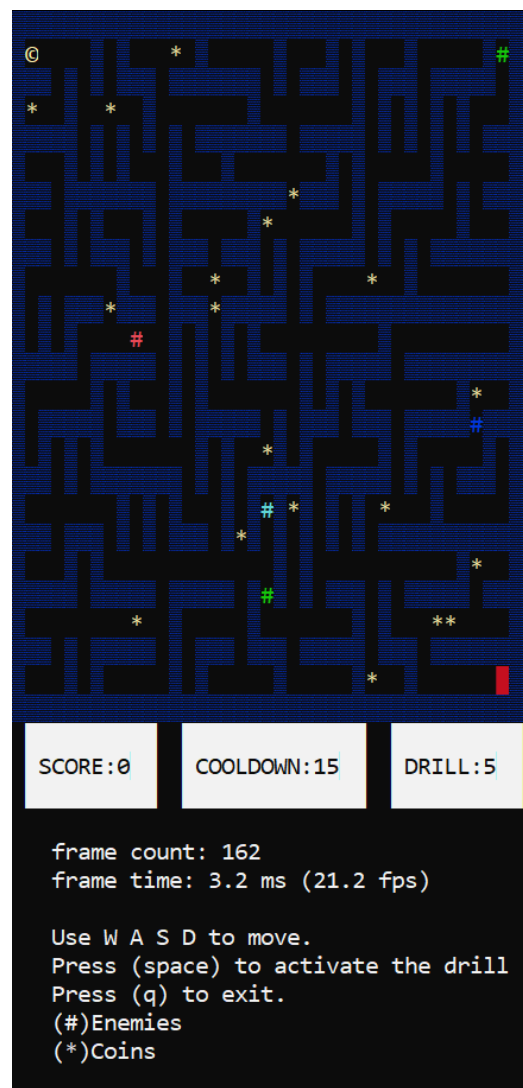


Figura 7

FinishScreen.fs

In questo sorgente viene creato uno sprite utilizzato come schermata finale nel caso di vittoria(**Figura 8**) o sconfitta(**Figura 9**).

Per la modalità di gioco : Pacman, la schermata è differente perché viene stampato anche il punteggio che si ha ottenuto giocando(**Figura 10**).



Figura 8



Figura 9

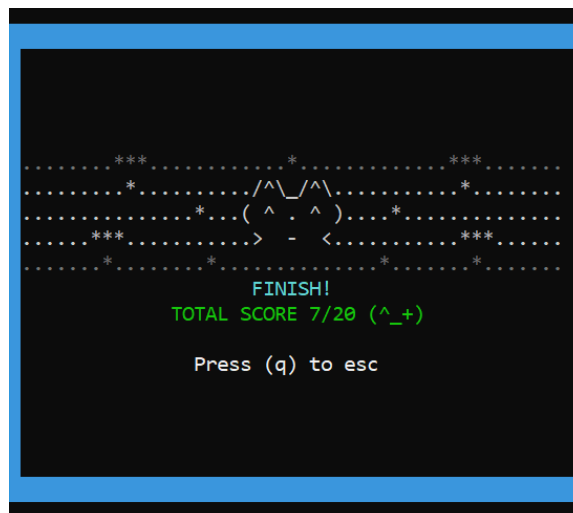


Figura 10

CONCLUSIONI

Il progetto finale è funzionante per ogni modalità di gioco.

Il codice è quasi interamente commentato, in modo da consentire a chi lo visualizza di comprendere al meglio.

Sono stati implementati correttamente tutti i task e tutte le richieste definite nei requisiti di progetto, in aggiunta è stata implementata una nuova modalità di gioco chiamata “Invisible_Automatic_Resolution”.

Questa modalità mostra la risoluzione automatica di un labirinto, senza mostrare il labirinto (**Figura 11**).



Figura 11