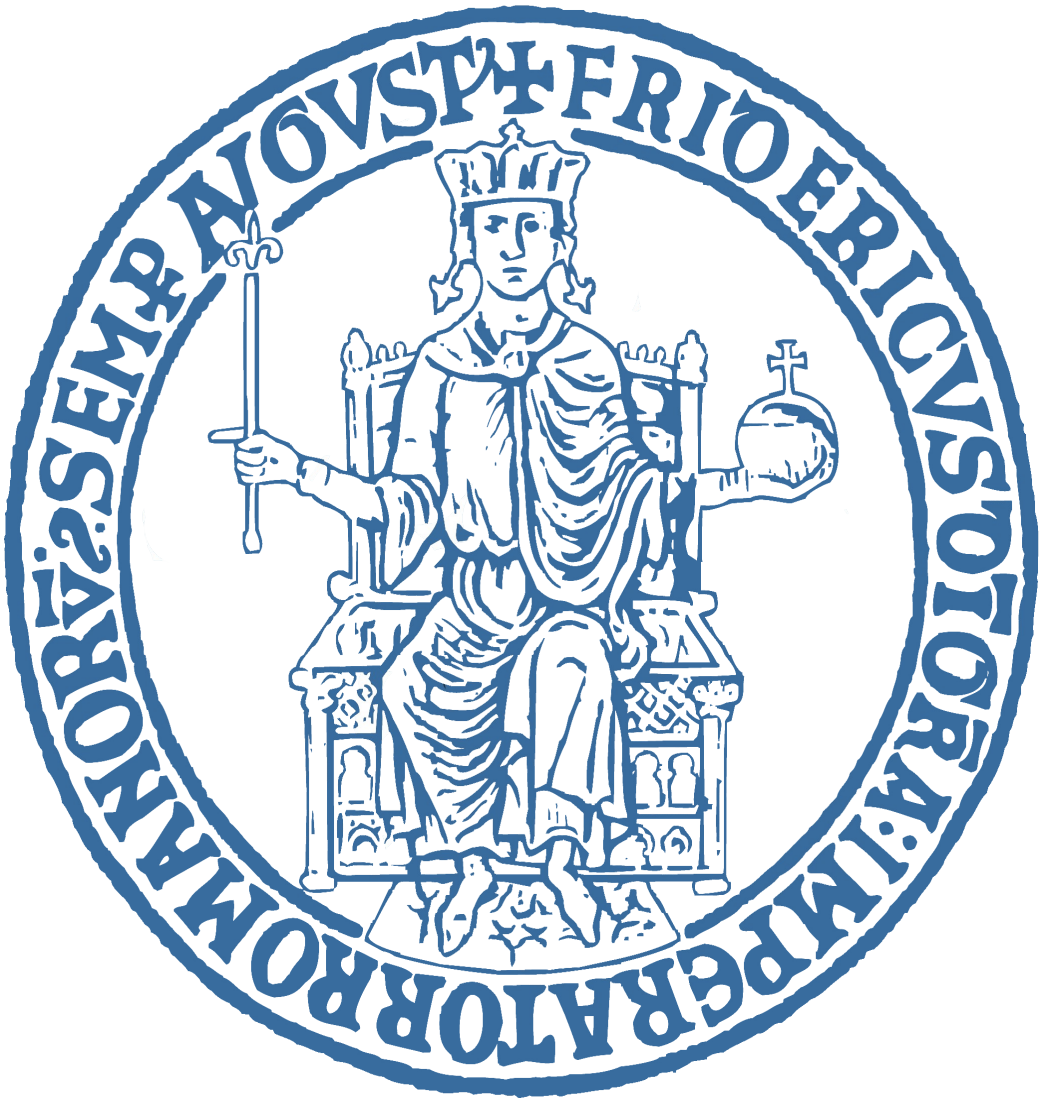
UNIVERSITÀ DEGLI STUDI FEDERICO II DI NAPOLI

Facoltà di Scienze Informatiche



Laboratorio Di algoritmi e strutture dati

Anno Accademico:2012/2013

"Fuga Dal Labirinto"

Studente:

Fattoruso Michele N86/299
Email: mi.fattoruso@studenti.unina.it

Sommario

1. Descrizione del problema

Si deve realizzare in linguaggio c un gioco interattivo che permetta all'utente, di muovere un personaggio lungo i corridoi di un labirinto. Il labirinto dovrà essere di tipo toroidale,ovvero in cui i bordi del labirinto risultano essere coincidenti. Ad aumentare la difficoltà del gioco,avremo un avversario che avrà lo scopo di raggiungere e quindi catturare il nostro personaggio.

L'utente potrà controllare il personaggio mediante l'utilizzo dei tasti freccia. Inoltre l'utente non sarà mai in grado di visualizzare l'intera composizione del labirinto,ma soltanto una piccola porzione (relazionata alla difficoltà scelta dall'utente) centrata sul nostro personaggio.

L'avversario avrà il compito di inseguire il nostro personaggio con lo scopo di catturarlo. I movimenti del mostro saranno implementati in due modi differenti:
quando l'avversario si trova al di fuori del riquadro visibile dall'utente,dovrà alternare mosse casuali a mosse lungo il percorso minimo tra la sua posizione e la posizione dell'avversario.
Quando l'avversario si trova all'interno del riquadro visibile,dovrà muoversi sempre lungo il percorso minimo tra la posizione dell'avversario e la posizione del nostro personaggio.

Durante la partita,dovranno apparire dei bonus in posizioni e tempi casuali. I bonus potranno avere i seguenti significati:

1. aumento della velocità del personaggio
2. visualizzazione del percorso minimo tra il nostro personaggio e l'uscita del labirinto
3. aumento della dimensione della finestra di visualizzazione dell'utente.

La raccolta del bonus avverrà attraverso il passaggio del nostro personaggio al di sopra della casella dove è situato il bonus. Questo così verrà raccolto e sommato a quelli già in possesso da parte dell'utente.
I bonus potranno essere successivamente utilizzati tramite la pressione dei seguenti tasti:

q) Aumento della dimensione della finestra visualizzata dall'utente
w) Aumento della velocità del nostro personaggio
e) Comparsa di un percorso che indica il percorso minimo tra la posizione attuale e l'uscita dal labirinto

inoltre è stata introdotta la possibilità di mettere in pausa il gioco. Questa funzionalità potrà essere attivata attraverso la pressione del tasto “p”.

è inoltre possibile aumentare o diminuire la difficoltà del gioco (difficoltà di default : facile) tramite la pressione dei tasti “1,2 e 3”,che rappresentano rispettivamente il grado di difficoltà facile,medio e difficile. L'aumento o la diminuzione della difficoltà

causerà variazioni nella velocità del mostro,la grandezza del riquadro visualizzato e l'intelligenza artificiale del mostro.

2.Scelte Implementative

La fase di progettazione del gioco potrà essere suddivisa nei seguenti sotto-problemi , che saranno analizzati separatamente:

1. implementazione del labirinto
2. Implementazione del personaggio guidato dall'utente
3. implementazione dell'avversario guidato dall'intelligenza artificiale
4. Implementazione del sistema di gestione dei bonus

2.1 Gestione del labirinto

per l'implementazione del labirinto si è scelto di rappresentare il tutto mediante una struttura a grafo in cui ogni nodo può potenzialmente essere adiacente ad altri quattro nodi,uno per ogni direzione di spostamento.
Per l'implementazione del grafo si è scelto di far uso di una matrice di interi in cui ad ogni posizione della matrice è associato un valore che rappresenta in modo univoco una tipologia di elemento che si trova in tale posizione. Tale valore potrà successivamente essere modificato in relazione alla tipologia di elemento presente nella casella.
I valori utilizzati per la rappresentazione degli elementi sono i seguenti:

CORRIDOIO :	0
MURO :	1
PERSONAGGIO DELL UTENTE :	2
USCITA DAL LABIRINTO :	-2
AVVERSARIO :	3 / 7 / 9 / 13
PERCORSO MINIMO	4
BONUS :	6 / 10

come si può notare l'avversario ed i bonus possono essere rappresentati da vari interi. Per la rappresentazione dell'avversario useremo “3” per rappresentare un mostro che si sposta su una cella vuota,7 per rappresentare l'avversario che si sposta sopra un percorso minimo,9 per la rappresentazione dell'avversario che si sposta al di sopra di un bonus,13 per la rappresentazione dell'avversario che si sposta sopra una posizione occupata sia da un bonus che dal percorso minimo.
Allo stesso modo per la rappresentazione del bonus useremo 6 per rappresentare il bonus che viene piazzato su una casella vuota e useremo 10 per rappresentare un

bonus che viene piazzato su una casella in cui era presente il percorso minimo.

Dopo aver definito la struttura del labirinto da creare,andremo a definire la modalità di creazione del labirinto. Tale operazione avverrà attraverso la lettura da un file esterno. Questo file dovrà essere conforme alle specifiche per essere correttamente interpretato dal programma.
I simboli ammessi saranno i seguenti:

- I simboli - , = , “ “ e | identificano i muri del labirinto
- Il simbolo E identifica l'unico punto di uscita del labirinto
- il simbolo . Identifica un corridoio percorribile dal personaggio

in più ogni labirinto non porta superare le massime dimensioni rappresentabili tramite terminale di Windows(80x50).

2.2 Gestione del personaggio

Dopo aver deciso il metodo implementativo del labirinto ed aver visto come gestire la creazione del labirinto,il passo successivo sarà nella gestione dell'implementazione del personaggio. Tale personaggio sarà implementato con il simbolo grafico “#” che permetterà di distinguere il personaggio dagli altri elementi del gioco.
Come richiesto questo personaggio sarà comandato tramite la pressione dei tasti freccia. Ovviamente per il movimento del personaggio sarà implementato un metodo di input asincrono per la lettura del tasto scelto dall'utente. Inizialmente il personaggio partirà da fermo,in attesa dell'immissione da parte dell'utente del primo comando. Dopo il primo input il personaggio seguirà tale direzione fin quando possibile,ovvero fino ad incontrare un muro oppure fino all'immissione di una direzione differente.

Lo spostamento del personaggio all'interno del labirinto sarà così gestito:
una volta calcolate le coordinate della posizione di destinazione sarà modificato il valore di tale posizione all'interno della matrice,sommando 2 (il valore che rappresenta il mostro) al valore presente. Alla posizione di partenza del personaggio invece sarà sottratto 2.ovviamente,nel caso in cui la posizione di destinazione del personaggio sia occupata da un elemento diverso dal corridoio vuoto,come un mostro,un bonus o il percorso minimo,il programma si preoccuperà di gestire in modo opportuno la modifica e visualizzazione dei dati.

2.3 Gestione dell'avversario

Dopo aver scelto il metodo implementativo del personaggio,bisogna fare la stessa operazione per l'avversario guidato dall'intelligenza artificiale. Tale personaggio,a differenza del nostro personaggio si muoverà in modi diversi a seconda della suo posizione nel labirinto. In particolare è richiesto che nel caso in cui l'avversario si trovi al di fuori del riquadro di visualizzazione dell'utente dovrà alternare movimenti casuali e movimenti lungo il percorso minimo. Nel caso in cui l'avversario si trovi all'interno del riquadro di visualizzazione dell'utente dovrà muoversi verso il percorso minimo tra l'avversario e la posizione del mostro.

Il movimento del mostro all'interno della mappa del labirinto sarà gestito come già descritto per il personaggio dell'utente.

Una volta che l'avversario avrà raggiunto e quindi catturato il nostro personaggio,questo sarà spostato ad una posizione casuale del labirinto.

2.4 Gestione dei bonus

per finire bisogna implementare il sistema di gestione dei bonus. Questi compariranno in posizioni ed in tempi casuali sulla mappa (ovviamente soltanto lungo i corridoi) e saranno identificati dalla lettera “B”.quando il personaggio guidato dall'utente si troverà nella stessa posizione del bonus,il programma accumulerà il bonus con quelli già in possesso,aumentando la durata dell'effetto del bonus una volta attivato.

come detto in precedenza abbiamo tre tipologie di bonus:

- aumento della velocità del personaggio guidato dall'utente
- aumento della dimensione della finestra di visualizzazione
- visualizzazione del percorso minimo tra il personaggio e l'uscita

la prima tipologia del bonus è implementata diminuendo il valore (Pacwait) di attesa associato all'intervallo di tempo che deve trascorrere tra due spostamenti.
l'aumento della dimensione della finestra di visualizzazione è implementando incrementando il valore (raggio) della finestra che rappresenta il numero di elementi da rappresentare attorno al nostro personaggio.
la visualizzazione del percorso minimo è implementata calcolando il percorso minimo tra la posizione attuale del personaggio e quella dell'uscita del labirinto attraverso l'algoritmo di ricerca A*.dopo il calcolo del percorso,sarà stampata sulla porzione di mappa visualizzata dall'utente ,una scia che indica il percorso da seguire.

3.Strutture dati utilizzate

3.1. Coda a Priorità

La coda a priorità è una struttura astratta che permette di rappresentare le strutture dati di una coda e di uno stack,dove ad ogni elemento è associata una priorità.

Le operazioni definite sulla seguente struttura sono:

- Inserimento(H, key , prio , prev) : operazione che permette di inserire l'elemento “key” lella coda a priorità;
- minimo(H) : operazione che restituisce l'elemento minimo della coda a priorità;
- EstraiMinimo (H) : operazione che estrae l'elemento minimo dalla coda a priorità;
- DeleteKey (H , i) : operazione che elimina dalla coda a priorità l'elemento in posizione i-esima;
- IncreakeKey (H , i , val) : operazione che incrementa il valore associato all'elemento in posizione i-esima;
- DecreaseKey (H , i , val) : operazione che decrementa il valore associato all'elemento in posizione i-esima.

3.2 Grafo

La struttura dati grafo è un tipo di dato astratto definito come una coppia ordinata G=(V,E) dove V è l'insieme dei vertici ed E è l'insieme degli archi.
Si distinguono due tipi basilari di grafi:i grafi orientati e i grafi non orientati.

Un grafo si dirà non orientato se la connessione tra due vertici i – j ha lo stesso significato della connessione j – i.
Un grafo si dirà orientato se la connessione tra due vertici x – y non ha lo stesso valore della connessione y – x.

Esistono fondamentalmente due modi per rappresentare un grafo con una struttura dati utilizzabile da un programma: la lista di adiacenza e la matrice di adiacenza.
In una lista di adiacenza la lista mantiene un elenco di nodi,e ad ogni nodo è collegata una lista di puntatori ai nodi collegati da un arco.
In una matrice di adiacenza,una matrice N per N,dove N è il numero di nodi,mantiene un valore vero un una cella (a,b) se il nodo a è collegato al nodo b.

Ognuna delle due rappresentazioni ha alcuni vantaggi:una lista di adiacenza risulta più adatta a rappresentare grafi sparsi o con pochi archi,perciò è facile trovare tutti i nodi adiacenti ad un dato nodo e verificare l'esistenza di connessione tra nodi;

una matrice di adiacenza è invece più indicata per descrivere grafi densi e con molti archi,inoltre rende più facile invertire i grafi e individuarne sotto-grafi.

Per la realizzazione del labirinto invece abbiamo preferito rappresentare il grafo mediante una matrice di interi. Questa rappresentazione si presta alla perfezione per il nostro problema visto che il numero di elementi adiacenti massimo è fissato a quattro ed inoltre abbiamo il vantaggio di non dover memorizzare ogni adiacenza dato che ci sarà più vantaggioso calcolarle ad ogni richiesta.

4. Principali algoritmi utilizzati

Algoritmo di ricerca euristica a*

A* è un algoritmo di ricerca su grafi che individua un percorso da un dato nodo iniziale verso un dato nodo finale. Questo algoritmo utilizza una stima euristica che classifica ogni nodo mediante una stima della strada migliore che passa attraverso tale nodo. Per il calcolo del valore associato ad ogni nodo viene usata una funzione euristica la cui relazione è la seguente:

$$f(v)=g(v)+h(v)$$

dove g(v) indica la valutazione del miglior percorso finora trovato tra il nodo sorgente e il nodo corrente v,mentre h(v) indica una stima del costo per raggiungere il nodo finale a partire dal nodo corrente. Tale funzione può essere data da una delle due seguenti relazioni:

- Distanza di Manhattan $h(v)=\left(\left|x_t-x_v\right|\right)+\left(\left|y_t-y_v\right|\right)$
- Distanza Euclidea $h(v)=\left(\left|x_t-x_v\right|\right)+\left(\left|y_t-y_v\right|\right)$

L'algoritmo partiziona l'insieme dei vertici in tre gruppi:

- Insieme dei vertici non raggiunti
- Openset,cioè l'insieme dei vertici raggiunti ma non ancora espansi
- ClosedSet,cioè l'insieme dei vertici già espansi

L'algoritmo riceve in input un riferimento al grafo su cui effettuare la ricerca e le coordinate dei vertici di partenza e di arrivo su cui effettuare la ricerca. L'algoritmo è il seguente:

```
int** A_star(MAPPA* map, int startX, int startY, int endX,int endY){
    int found = 0, i = 0, inOpen, inClosed;
    float cost = 0;

    HEAP *closed, *open;
```

```
    HEAP_EL *curr;

    closed = (HEAP*)malloc(sizeof(HEAP));
    open = (HEAP*)malloc(sizeof(HEAP));

    open->dim_heap = 0;
    closed->dim_heap = 0;

    open->array = NULL;
    closed->array = NULL;

    int **adj,  **path;
    int* elemento = (int*)malloc(sizeof(int) * 2);
    int* valore;

    elemento[0] = startX;
    elemento[1] = startY;

    Inserimento(open, elemento,dist_euclidea(startX, startY, endX, endY), NULL);

    while(open->dim_heap > 0 || found != 1){

        curr = EstraiMinimo(open);
        valore = (int*) curr->elem;

        if( valore[0] == endX && valore[1] == endY){

            found = 1;

        }
        else{

            adj = adiacenti(valore[0], valore[1], map->Nrighe, map->Ncolonne);

            for(i = 0; i < 4; ++i){

                if(map->Area[ adj[i][0]] [ adj[i][1] ] != 1){

                    cost=nodiVisitati(curr)+1+dist_euclidea(adj[i][0],adj[i][1],
endX, endY);
                    inOpen = XinCoda(open, adj[i][0], adj[i][1]);
                    inClosed = XinCoda(closed, adj[i][0], adj[i][1]);

                    if( inOpen > -1 && cost < open->array[inOpen].priorita){

                        elemento = (int*)open->array[inOpen].elem;
                        open->array[inOpen].prec = curr;
                        IncreaseKey(open, inOpen, cost);

                    }
                    else if(inClosed > -1 && cost < closed-
>array[inClosed].priorita){

                        DeleteKey(closed, inClosed);
                        elemento = (int*)malloc(sizeof(int) * 2);
                        elemento[0] = adj[i][0];
                        elemento[1] = adj[i][1];
                        Inserimento(open, elemento, cost, curr);

                    }
                    else if( inClosed < 0 && inOpen < 0){

                        elemento = (int*)malloc(sizeof(int) * 2);
                        elemento[0] = adj[i][0];
```

```
                        elemento[1] = adj[i][1];
                        Inserimento(open, elemento, cost, curr);

                    }
                }
            }
        }
        Inserimento(closed, valore, curr->priorita,curr->prec);
    }
    if (found){
        path = GeneratePath(closed, endX, endY);
    }
    else{
        path = NULL;
    }
    return path;
}
```

Algoritmo di visualizzazione del labirinto

Algoritmo fondamentale per il funzionamento del nostro labirinto il quale ci permette di ricreare in maniera grafica una rappresentazione del nostro grafo. Questo algoritmo ci permette di rappresentare come richiesto una porzione del labirinto centrata sul nostro personaggio. La versione base dell'algoritmo è stata modificata in modo da permettere una visualizzazione più fluida del labirinto,principalmente è stato tolto il Clearscreen all'inizio della funzione in modo da evitare sfarfallii del terminale. Si è poi gestita in modo opportuno la cancellazione delle eventuali parti di labirinto che superavano il raggio massimo di visualizzazione dell'utente e la stampa delle nuove parti da visualizzare.

L'algoritmo utilizzato è il seguente:

```
void VisualizzaMappa (MAPPA *M,int X,int Y,int size,int* Muro,int Direzione){
    int   i, j, k,OldX,OldY;

    if (M)
    {   Window (0,0, M->Ncolonne-1, M->Nrighe+2);
        TextColor (0x0F);

        X=X-size;
        Y=Y-size;
        OldY=Y;
        OldX=X;

        if (X<0){
            X=0;
        }
        if(Y<0){
            Y=0;
        }
        size=2*size;
        //Vengono stampati soltanto i muri presenti nel rettangolo di
//visualizzazione
        for (i=X; i<OldX+size && i<M->Nrighe; i++)
            for (j=Y; j<OldY+size && j<M->Ncolonne; j++)
                if (M->Area[i][j])
                {
                    if(M->Area[i][j]==2){ //mio omino
```

```
                    TextColor(0x0E);
                    Gotoxy (j, i);
                    printf ("#");
                    TextColor(0x0F);
                }
                else if(M->Area[i][j]==-2){ //uscita dal labirinto
                    k=16;
                    Gotoxy(j,i);
                    printf ("%c", Muro[k]);
                } //Visualizzazione fantasma
                else if(M->Area[i][j]==3 || //Fantasma
                        M->Area[i][j]==7 || //Fantasma sul percorso minimo
                        M->Area[i][j]==9 || //Fantasma sul bonus
                        M->Area[i][j]==13){ //Fantasma sul Bonus e percorso
                                            //minimo
                    Gotoxy(j,i);
                    TextColor(0x04);
                    printf("G");
                    TextColor(0x0F);  //la stampa del fantasma avviene sia
                                      //quando si refresha la mappa sia
                                      //quando il personaggio si muove
                }
                else if(M->Area[i][j]==4 ){  //percorso minimo
                    Gotoxy(j,i);
                    TextColor(0x0E);
                    printf(".");
                    TextColor(0x0F);
                }
                else if(M->Area[i][j]==6 ||  //bonus
                        M->Area[i][j]==10){  //bonus sul percorso minimo
                    Gotoxy(j,i);
                    TextColor(0x01);
                    printf("B");
                    TextColor(0x0F);
                }
                else{
                    k = 0;
                    if (j<M->Ncolonne-1 && M->Area[i][j+1]==1) k += 1;
                    if (i>0             && M->Area[i-1][j]==1) k += 2;
                    if (j>0             && M->Area[i][j-1]==1) k += 4;
                    if (i<M->Nrighe-1   && M->Area[i+1][j]==1) k += 8;
                    Gotoxy(j,i);
                    printf ("%c", Muro[k]);
                }
            }
            else{
                Gotoxy (j, i);
                printf (" ");
            }

        if(Direzione==CURSORUP){
            i=OldX+size;
            if(i<M->Nrighe){  //non cancella la riga inferiore se questa esce
                              //dalle dimensioni del labirinto
                for (j=Y; j<Y+size && j<M->Ncolonne; j++){ //cancella la riga
inferiore del riquadro
                    Gotoxy(j,i);
                    printf(" ");
                }
            }
        }
        else if(Direzione==CURSORDOWN){
            i=OldX-1;
            for (j=Y; j<Y+size && j<M->Ncolonne; j++){  //Cancella la riga
```

```
                                                        //superiore al riquadro
                Gotoxy(j,i);
                printf(" ");
            }

        }
        else if(Direzione==CURSORRIGHT){
            j=OldY-1;
            for (i=X; i<OldX+size && i<M->Nrighe; i++){  //cancella la riga alla
                                                        //sinistra del riquadro
                Gotoxy(j,i);
                printf(" ");
            }
        }
        else if(Direzione==CURSORLEFT){
            j=OldY+size;
            if(j<M->Ncolonne){            //impedisce la stampa al di fuori
                                         //dell'area delimitata dal labirinto
                for (i=X; i<OldX+size && i<M->Nrighe; i++){  //cancella la riga
                                              //alla sinistra del riquadro
                    Gotoxy(j,i);
                    printf(" ");
                }
            }
        }
    }
}
```

5. Funzionamento del gioco

Il gioco implementato prevede il seguente funzionamento:all'avvio sarà richiesto all'utente di inserire il nome del file (compreso di estensione) da cui prelevare il labirinto

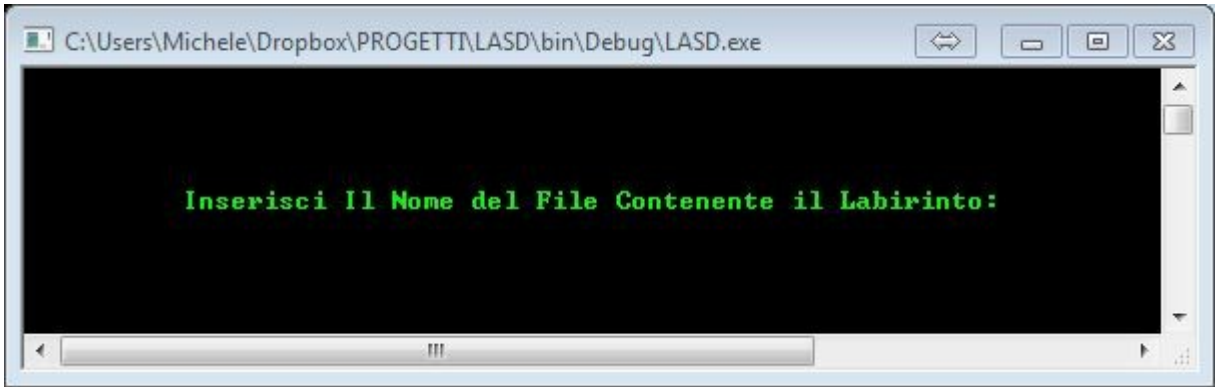


Illustrazione 1: Inserimento Nome file Labirinto

il file dovrà essere situato nella stessa cartella del file eseguibile del programma.
Inoltre l'applicazione è in grado,oltre a leggere i tre labirinti già in possesso dall'utente,di leggere qualsiasi labirinto creato dall'utente,a patto che il file rispetti le specifiche dichiarate in precedenza. Nel caso in cui il file non rispetti le specifiche o nel caso in cui il file sia inesistente verrà visualizzata la seguente schermata di errore:

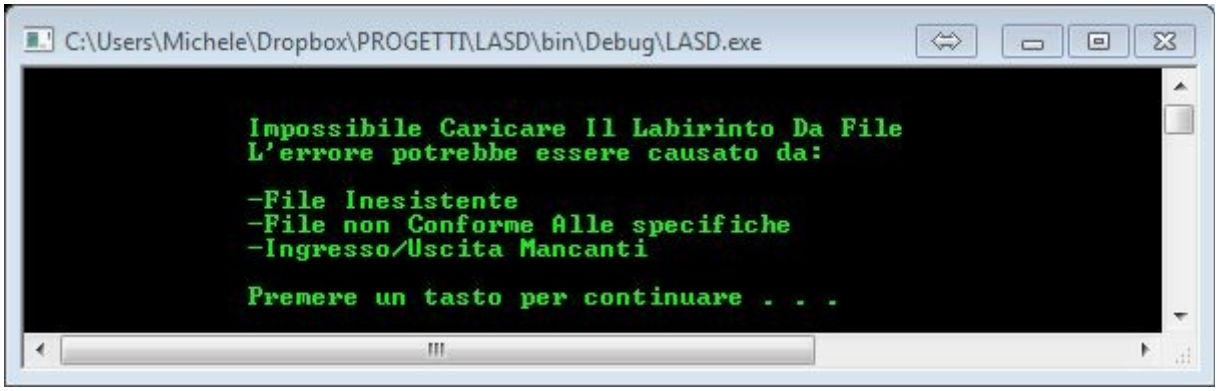


Illustrazione 2: Errore Caricamento

nel caso contrario,se il file soddisfa le specifiche verrà visualizzato il terminale con il labirinto precaricato.

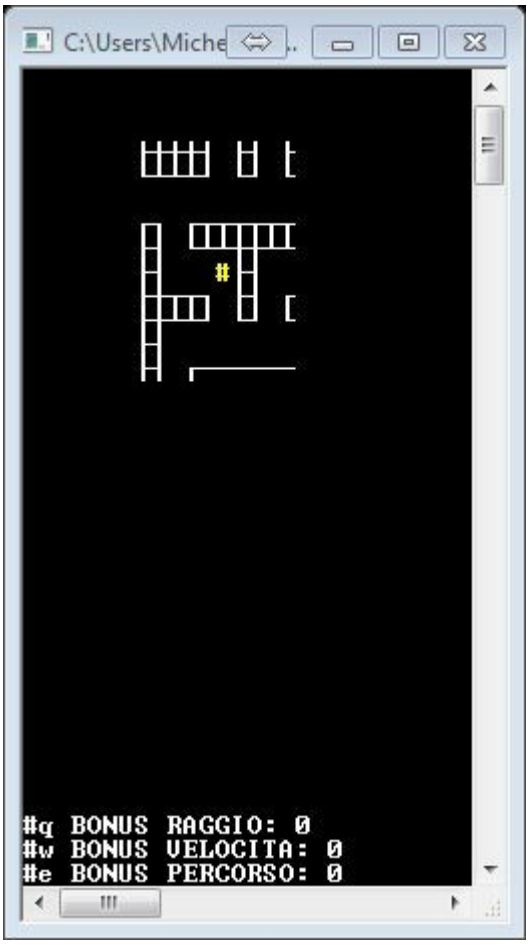


Illustrazione 3: Caricamento Completato

Dopo la lettura ed il caricamento del labirinto da file inizierà la partita vera e propria. Durante la partita l'utente potrà comandare il proprio personaggio mediante i tasti freccia.

Durante lo svolgimento del gioco l'utente potrà imbattersi in eventi che potranno favorirlo o sfavorirlo nella ricerca dell'uscita dal labirinto.

A favorire il completamento del gioco,incontreremo,lungo il percorso del labirinto alcuni bonus ,rappresentati tramite una “B” di colore blu che si andranno a cumulare a quelli raccolti precedentemente

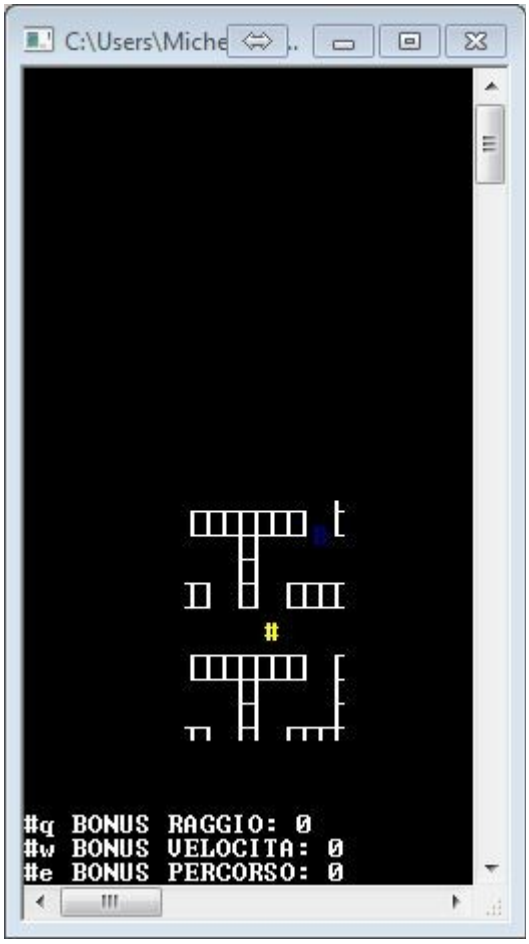


Illustrazione 4: Bonus sul percorso

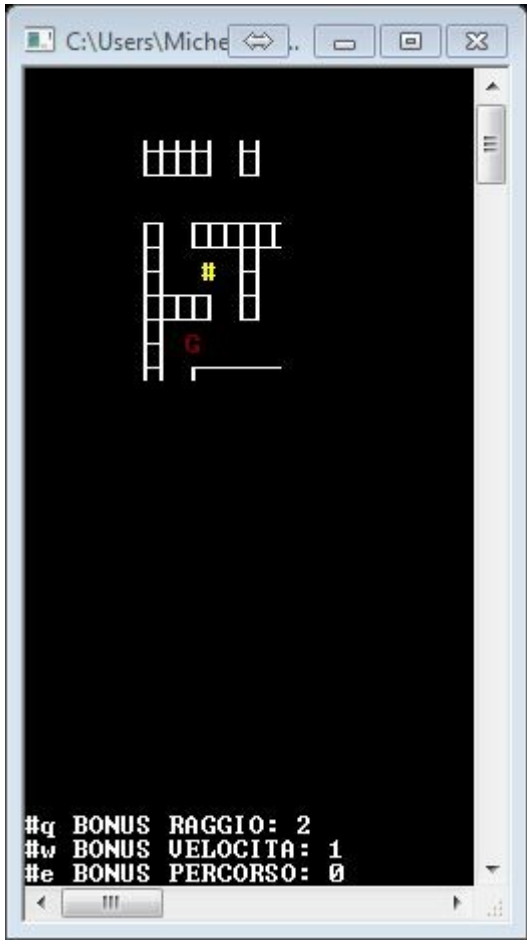


Illustrazione 5: Bonus Cumulato

questi bonus,quindi potranno essere utilizzati tramite la pressione degli appositi tasti. Tramite la pressione del tasto “q” avremo un aumento del raggio di visualizzazione del riquadro dell'utente mentre tramite la pressione del tasto “e” avremo la visualizzazione di un tratto del percorso minimo tra la posizione corrente del nostro personaggio e l'uscita dal labirinto.

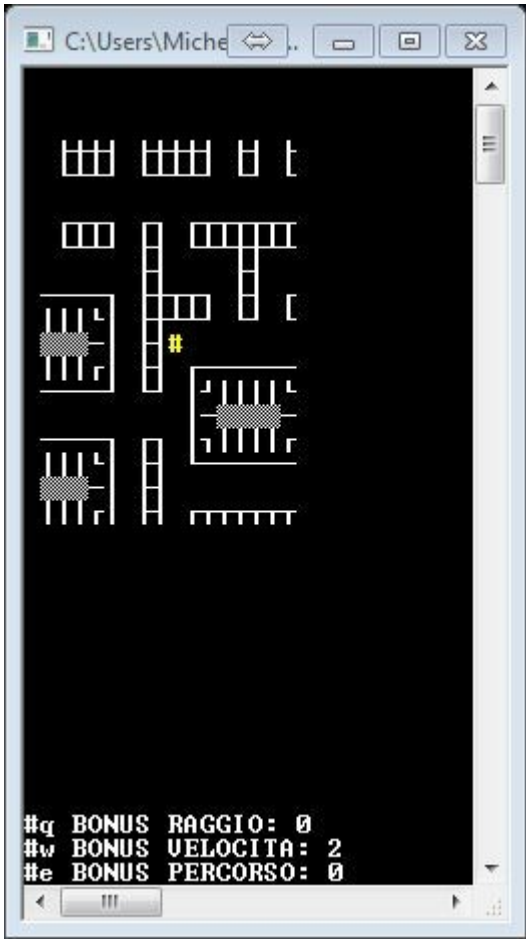


Illustrazione 6: Bonus Raggio

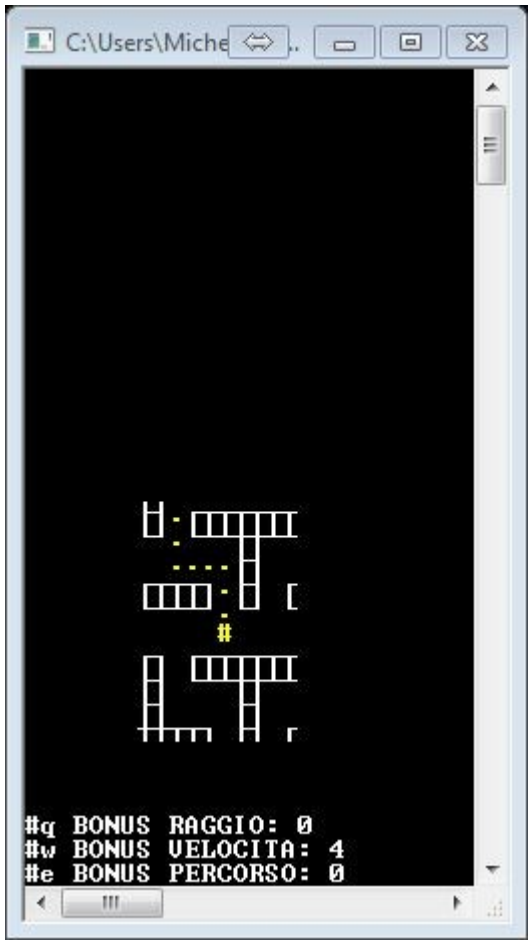


Illustrazione 7: Bonus percorso minimo

In caso di necessità è possibile anche mettere in pausa il gioco tramite la pressione del tasto "p".per riprendere il gioco basterà premere nuovamente p affinché il gioco riparta.

Illustrazione 8: Gioco in pausa

Nel caso in cui invece il nostro personaggio venga catturato dall'avversario si verrà reindirizzati ad una schermata a tempo che chiederà all'utente se voler continuare a giocare oppure terminare la partita. Nel caso in cui il giocatore sceglie di terminare la partita,il programma si chiuderà automaticamente. Nel caso in cui il giocatore sceglie di continuare la partita,gli sarà riposizionato il personaggio su una posizione casuale del labirinto,così da poter riprendere la partita.

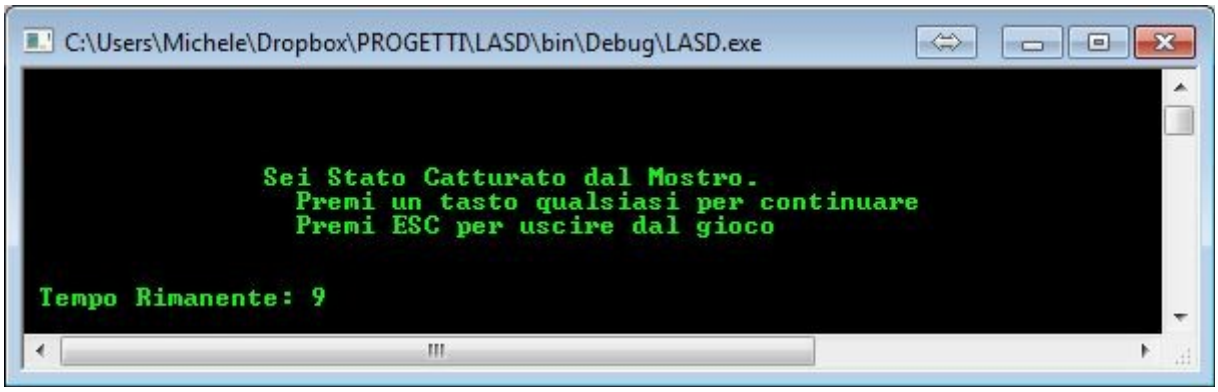


Illustrazione 9: Catturato dal mostro

Il gioco termina quando l'utente preme il tasto “ESC” oppure quando raggiunge l'uscita. Una volta raggiunta l'uscita sarà mostrato all'utente un messaggio di congratulazioni e successivamente il gioco terminerà.

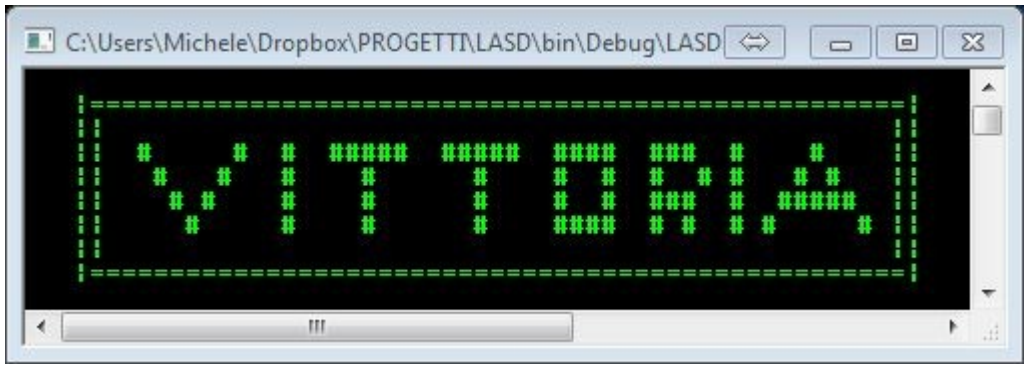


Illustrazione 10: Labirinto completato