

**COSTRUTTORE E RISOLTORE
DI LABIRINTI PERFETTI**



POLITECNICO
MILANO 1863

Relazione di progetto del corso di
Computer Animation

Facoltà di Computer Science and Engineering
e Design della Comunicazione

Redatta da
Marilena Coluccia

Professori referenti
Prof. Mattia Penati
Prof. Edie Miglio



Tutti noi viviamo dentro un labirinto e non c'è filo di Arianna che riesca a farcene uscire, non c'è Teseo che vi riesca, non c'è Dedalo che lo costruisca e non c'è Minotauro che lo abiti. Il labirinto non è altro che il groviglio di contraddizioni che vivono dentro di noi, alimentano la nostra vita, la rendono felice e infelice, danno a ciascuno di noi un destino che non sta scritto da nessuna parte ma che si forma giorno per giorno emergendo dal contrasto interiore e dal come, giorno per giorno, si risolve misurandosi con il sé stesso e con quello degli altri.

Eugenio Scalfari



<u>1. INTRODUZIONE</u>	7
1.1. IL LABIRINTO	7
1.2. OBIETTIVO	8
1.3. THE CODE	9
1.4. L'ALGORITMO A*	11
1.5. EDITOR E LINGUAGGI	12
<u>2. INTERAZIONE CON IL BROWSER</u>	13
2.1. JAVASCRIPT	14
<u>3. ANALISI</u>	15
3.1. IL GENERATORE	15
3.2. IL RISOLUTORE	23
<u>4. CONCLUSIONI E MIGLIORAMENTI</u>	27
<u>5. SITOGRAFIA</u>	30

Indice di figure e sezioni

Figura 3.1 - Cambio della lunghezza del passo di avanzamento	20
Figura 3.2 - Esempio di esecuzione completa	22
Figura 3.3 - Risultato del risolutore	26
Figura 4.1 - Labirinto realizzato con texture ed immagine di riferimento	28
Figura 4.2 - Altri modelli di labirinto	29
Sezione 2.1 - Html e Css code	13
Sezione 3.1 - Inizializzazione di Phaser	16
Sezione 3.2 - Funzione Create	18
Sezione 3.3 - Generazione del percorso tramite EasyStar.js	23
Sezione 3.4 - Disegno del percorso	24
Sezione 3.5 - Funzione di ridimensionamento	25

1. INTRODUZIONE

Quasi cinquemila anni fa, partendo dall'area mediterranea, un semplice disegno geometrico al quale venne dato il nome “labirinto” iniziò a diffondersi in tutto il mondo, permettendo a ciascun contesto culturale in cui si trovava di mutarne forma, dimensione, significato e funzione.

Grazie a questa sua duttilità il labirinto è diventato un simbolo universale, o meglio, un complesso simbolico fin dai lontani tempi della sua comparsa.

1.1. IL LABIRINTO

Un labirinto è un disegno geometrico, più o meno complesso, costituito da varie linee e corsie, disposte a spirale oppure costituire un quadrato che tracciano un percorso verso il centro o altri punti del terreno a disposizione.

L'impressione creata è quella di un groviglio inestricabile di meandri, nei quali è facile smarrirsi, motivo per cui usiamo spesso la metafora del labirinto per indicare situazioni e problemi complicati, anche se, come vedremo, il disordine è quasi sempre solo apparente.

1.2. OBIETTIVI

Questa relazione è stata redatta a completamento del progetto realizzato per il corso di Computer Animation della facoltà di Design della Comunicazione del Politecnico di Milano tenuto dal prof. E. Miglio.

Uno degli obiettivi prepostisi per l'individuazione dell'oggetto del progetto è stato la volontà di trovare un argomento che potesse mettere in risalto le conoscenze apprese in entrambe le facoltà, oltre a quelle inerenti al corso stesso ovviamente.

Su suggerimento dei docenti, ci si è focalizzati sulle tecniche di animazione realizzate tramite la programmazione informatica. In particolare, viste le conoscenze personali si è pensato di concentrare il lavoro sulle tecniche utilizzate per riprodurre le animazioni sul web.

Dopo attente ricerche e analisi di lavori riguardanti animazioni/simulazioni tramite l'utilizzo di diversi linguaggi di programmazione e/o librerie si è deciso di focalizzare l'attenzione sull'analisi del lavoro fatto da altri programmatore per poterne apprendere a pieno le tecniche e il pensiero di analisi in modo da poterli fare propri ed eventualmente riutilizzarli alla necessità.

In particolare, si è scelto di analizzare il codice redatto da Emanuele Feronato per la costruzione di un labirinto perfetto a pianta randomica e conseguente calcolo della soluzione dello stesso.

Il progetto risulta aderente agli obiettivi in quanto permette di visionare nel tempo la costruzione, come sarà possibile vedere nei prossimi capitoli, inizializzando dei loop temporali per la visualizzazione dei passi effettuati come se fossero frame appartenenti ad un'animazione.

1.3. THE CODE

Lo scopo che il programmatore ha cercato raggiungere con il codice, che verrà analizzato in seguito, è quello di realizzare, in maniera del tutto randomica, labirinti perfetti e per ognuno di essi, dato un punto di partenza e di arrivo, trovarne la soluzione.

I labirinti in generale (e quindi gli algoritmi per creare labirinti) possono essere organizzati secondo sette diverse classificazioni:

- Dimensione, il numero di dimensioni nello spazio coperto da Labirinto;
- Hyperdimension, la dimensione dell'oggetto che si sposta attraverso il labirinto, in contrapposizione alla dimensione dell'ambiente labirinto stesso;
- Topologia, la geometria dello spazio in cui il labirinto nel suo complesso esiste;
- Tassellazione, la geometria delle singole celle che compongono il labirinto;
- Routing, i tipi di passaggi che si possono effettuare all'interno di qualsiasi geometria definita nelle categorie sopra;
- Texture, lo stile dei passaggi in qualunque routing in qualsiasi geometria;
- Focus: suddivide i labirinti in base alla modalità utilizzata per la loro costruzione: addizionatori di muri o intagliatori di passaggi.



Un labirinto "perfetto" indica dedalo senza loop o circuiti chiusi e senza aree inaccessibili. È chiamato anche labirinto semplicemente connesso. Un ulteriore condizione necessaria per l'identificazione di un labirinto perfetto è l'esistenza di uno ed un solo percorso da ogni punto ad un altro, il che implica l'esistenza di una ed una sola soluzione per ogni coppia di punti entrata-uscita.

In termini di informatica, un labirinto di questo tipo può essere descritto come uno spanning tree sull'insieme di celle o vertici.

Per chiarezza sono riportate di seguito alcune definizioni di ricerca operativa che sono utili alla comprensione.

Definizione 1.1 Un grafo G è costituito da una coppia di insiemi (V, E) dove V è detto insieme dei nodi ed E è detto insieme di archi ed è un sottoinsieme di tutte le possibili coppie di nodi in V . Se le coppie di nodi sono ordinate, il grafo è detto orientato, se non sono ordinate è detto non orientato.

Definizione 1.2 Un albero (tree) è un grafo non orientato che

- (a) È connesso e non possiede circuiti; o ,equivalentemente ,
- (b) è connesso ed ha un numero di spigoli pari al numero di vertici meno uno; o, equivalentemente,
- (c) comunque scelti due nodi al suo interno, possiede uno ed un solo cammino da uno all'altro.

Definizione 1.3 Dato un grafo $G = (V, E)$, un suo sottografo $G' = (V, E')(E' \subseteq E)$ che forma un albero è detto *spanning tree* (ST, albero ricoprente) di G

Esistono degli algoritmi per la ricerca di un albero ricoprente minimo (cioè il cui costo associato agli archi è il minore tra quelli

disponibili) come quello di Kruscal e quello di Prim. In questo caso non sarà necessario l'utilizzo di nessuno di questi in quanto, per costruzione il grafo derivante dal labirinto risulta essere un albero di copertura.

Per la ricerca del cammino si usano degli algoritmi di pathfinding ed in particolar modo verrà preso in esame l'algoritmo A*.

1.4. L'ALGORITMO A*

L'algoritmo di ricerca A* è una delle tecniche più utilizzate negli attraversamenti di percorsi e grafi.

Considerando un grafo composta da un insieme di nodi V e denotati con S il nodo di partenza e T il nodo di arrivo l'obiettivo è quello di raggiungere il T partendo da S il più velocemente possibile.

L'algoritmo A* sceglie i nodi intermedi da utilizzare in base al valore di una funzione f che risulta essere la somma di altre due funzioni g e h . Ad ogni passo viene scelto il nodo il cui valore di f minore.

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

La funzione $g(n)$ calcola il costo del percorso dal nodo S al nodo n, al contrario la funzione $h(n)$ utilizza un metodo euristico per ottenere un calcolo previsionale del restante costo fino a T.

Questo porta alla accorta scelta del percorso ad ogni passo, senza permettere però di poter tornare indietro sui propri passi.

1.5. EDITOR E LINGUAGGI

Per poter analizzare il codice è necessario avere a disposizione almeno un Plain Text Editor ed in questo caso si è utilizzato Visual Studio Code.

I linguaggi che verranno analizzati sono:

HTML, per struttura della pagina web;

Css, per la formattazione grafica degli elementi;

JS, in particolare la libreria Phaser.io e EasyStar.js per la costruzione geometrica e l'interattività;

Le porzioni di codice che è stato necessario inserire, per semplicità, sono state definite “Sezioni” e numerate.



Visual Studio Code

HTML



EasyStar.js

asynchronous pathfinding in javascript

2. INTERAZIONE CON IL BROWSER

Per poter visualizzare la pagina web si ha bisogno di un file HTML

In questo caso può bastare una semplice pagina vuota a cui aggiungere tutti gli elementi tramite gli altri linguaggi che vi si possono associare.

Senza creare un nuovo file CSS è possibile inserire la formattazione grafica direttamente all'interno del codice Html: questo risulta utile solo se le impostazioni sono costituite da poche righe, altrimenti si rischierebbe di fare troppa confusione all'interno di uno stesso documento.

Essendo i testi Js più lunghi è preferibile tenerli distinti dal documento iniziale e inserirli tramite riferimento.

The diagram illustrates the structure of an HTML document. It features a large green bracket on the right side labeled 'HTML' that covers the entire document. Inside this bracket, there are two nested curly braces: one labeled 'CSS' that encloses the CSS code within the head section, and another labeled 'JS' that encloses the three script tags. The CSS code defines a body with zero padding and margin. The JS code includes three script tags pointing to external files: 'phaser.min.js', 'easystar-0.4.3.min.js', and 'game.js'.

```
<!DOCTYPE html>
<html>
  <head>
    <style type="text/css">
      body{
        padding:0px;
        margin:0px;
      }
    </style>
    <script src="phaser.min.js"></script>
    <script src="easystar-0.4.3.min.js"></script>
    <script src = "game.js"></script>
  </head>
  <body>
  </body>
</html>
```

Sezione 2.1 - Html e Css code

E.1. JAVASCRIPT

JavaScript è un linguaggio leggero, interpretato, funzionale e orientato agli oggetti, conosciuto per lo più come linguaggio di script per pagine web. Permette di implementare oggetti complessi sulle pagine web - ogni volta che una pagina web fa di più che visualizzare informazioni statiche come aggiornamenti di contenuti tempestivi, mappe interattive, grafica 2D / 3D animata, video jukebox scorrevoli ed altro, è altamente probabile che sia coinvolto del codice JavaScript nella sua implementazione.

Phaser è un software di gioco 2D gratuito per la creazione di giochi HTML5 per desktop e dispositivi mobili. È stato creato da Photon Storm.

Phaser utilizza internamente un renderer Canvas e WebGL e può scambiare automaticamente tra loro in base al supporto del browser. Ciò consente un rendering veloce su desktop e dispositivi mobili. Usa la libreria Pixi.js per il rendering.

I giochi possono essere compilati su iOS, Android e applicazioni desktop native tramite strumenti di terze parti come Apache Cordova

EasyStar.js è un asincrono A * pathfinding API scritto in JavaScript da utilizzare nei giochi HTML5 e progetti interattivi, che ha l'obiettivo di rendere più semplice e veloce l'implementazione della ricerca di un cammino.

Nei prossimi paragrafi verrà analizzato il codice relativo ai file JS del progetto in cui saranno presenti metodi e funzionalità relative ad entrambe le librerie

3. ANALISI

Durante l'analisi dei due progetti si è notato che il programmatore ha preferito utilizzare un solo file js (esclusa l'importazione delle librerie) piuttosto che inserire un file per la costruzione ed uno per la risoluzione, cosa insolita dal momento che quello della costruzione era stato implementato in precedenza.

Per chiarezza e completezza si procederà lo stesso prima all'analisi del codice riguardante solo la costruzione e poi di quello riguardante costruzione e risoluzione.

3.1. IL GENERATORE

Questa porzione di codice è stata realizzata utilizzando Phaser 2.6 e quindi presenta ancora le funzioni riguardanti la vecchia versione (attualmente è possibile utilizzare Phaser 3 in maniera stabile, che verrà poi usata nella seconda porzione di codice)

Inizialmente, come in molti programmi vengono definite delle variabili globali o delle costanti, come ad esempio le dimensioni delle caselle del labirinto.

```
var game;
var maze = [];
var mazeWidth = 81;
var mazeHeight = 61;
var tileSize = 10;
var mazeGraphics;
```

```
window.onload = function() {  
    game = new Phaser.Game(810, 610, Phaser.CANVAS, "");  
    game.state.add("PlayGame",playGame);  
    game.state.start("PlayGame");  
}
```

Sezione 3.1 - Inizializzazione di Phaser

Per iniziare un gioco in Phaser bisogna inizializzare il gioco definendone almeno dimensioni e tipo di render. Phaser permette di avere più stati del gioco che possono corrispondere ai vari livelli di gioco e permette di cambiare completamente il gioco con una costruzione personalizzata per ogni tipo di stato.

Per comodità spesso gli stati vengono identificati attraverso una nuova classe di JS, ma se si tratta di un solo stato si può omettere e passare alla definizione delle funzioni necessarie al corretto funzionamento di Phaser.

Le 4 funzioni previste sono:

- **preload**, serve a definire/caricare tutto il necessario prima che venga creato il gioco;
- **create**, definisce tutte le caratteristiche del gioco, come elementi, sia statici che dinamici, comportamento della telecamera, ect;
- **update**, in caso di presenza di dinamicità definisce le regole da seguire per ottenere il risultato desiderato;
- **render**, per le impostazioni di rendering.

Non dovendo utilizzare alcuna texture esterna (verranno create tramite le primitive preesistenti in JS) e utilizzando un trucco tecnico per simulare la dinamicità, l'unica funzione necessaria per la generazione del labirinto è *create*.

```
var playGame = function(game){};

playGame.prototype = {
    create: function(){
        mazeGraphics = game.add.graphics(0, 0);
        var moves = [];
        for(var i = 0; i < mazeHeight; i++){
            maze[i] = [];
            for(var j = 0; j < mazeWidth; j++){
                maze[i][j] = 1;
            }
        }
        var posX = 1;
        var posY = 1;
        maze[posX][posY] = 0;
        moves.push(posY + posY * mazeWidth);
        game.time.events.loop(Phaser.Timer.SECOND/25, function(){
            if(moves.length){
                var possibleDirections = "";
                if(posX+2>0 && posY+2<mazeHeight-1
                    && maze[posX+2][posY]==1){
                    possibleDirections += "S";
                }
                if(posX-2>0 && posY-2<mazeHeight-1
                    && maze[posX-2][posY]==1){
                    possibleDirections += "N";
                }
                if(posY-2>0 && posY-2<mazeWidth-1
                    && maze[posX][posY-2]==1){
                    possibleDirections += "W";
                }
                if(posY+2>0 && posY+2<mazeWidth-1
                    && maze[posX][posY+2]==1){
                    possibleDirections += "E";
                }
            }
        })
    }
}
```

```

if(possibleDirections){
    var move = game.rnd.between(0,
        possibleDirections.length-1);
    switch (possibleDirections[move]){
        case "N":
            maze[posX - 2][posY] = 0;
            maze[posX - 1][posY] = 0;
            posX -= 2;
            break;
        case "S":
            maze[posX + 2][posY] = 0;
            maze[posX + 1][posY] = 0;
            posX += 2;
            break;
        case "W":
            maze[posX][posY - 2] = 0;
            maze[posX][posY - 1] = 0;
            posY -= 2;
            break;
        case "E":
            maze[posX][posY + 2]=0;
            maze[posX][posY + 1]=0;
            posY += 2;
            break;
    }
    moves.push(posY + posX * mazeWidth);
}else{
    var back = moves.pop();
    posX = Math.floor(back/mazeWidth);
    posY = back % mazeWidth;
}
drawMaze(posX, posY);
}, this); }
}

```

Sezione 3.2 - Funzione Create

Come primo passo si è creato una variabile grafica che conterrà tutte le primitive grafiche che verranno inserite sullo schermo: questa variabile oltre a rendere visibile il labirinto permetterà, come già accennato, di simulare l'animazione durante la costruzione.

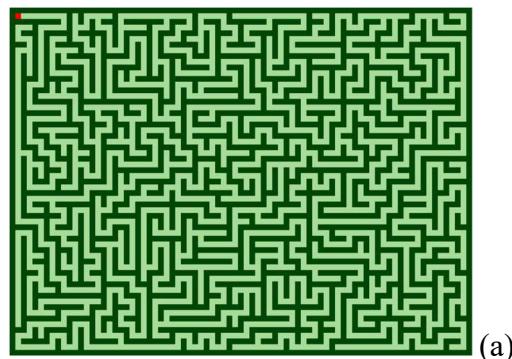
Il labirinto è stato tradotto dal programmatore con una matrice bidimensionale binaria ove 1 indica la presenza di un muro mentre 0 la strada libera. In fase di inizializzazione ogni valore della matrice è stato settato ad 1: idealmente si potrebbe pensare come ad un enorme siepe e l'algoritmo, come un giardiniere, provvederà a creare i passaggi al suo interno.

L'entrata è stata pensata nella casella [1][1] come si può vedere in figura. Per poter ripercorrere tutti i passi vi è a disposizione una collezione (array) di tutte le mosse effettuate. Queste vengono salvate utilizzando la seguente formula

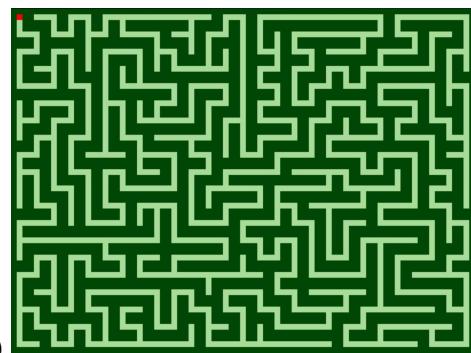
$$\begin{aligned} \text{Valore salvato} = & \text{ PosizioneVerticale} * \text{LarghezzaLabirinto} \\ & + \text{PosizioneOrizzontale} \end{aligned}$$

Al posto della larghezza del labirinto si sarebbe potuto utilizzare qualsiasi valore visto che per recuperare i valori delle posizioni basta utilizzare la divisione intera e la funzione resto.

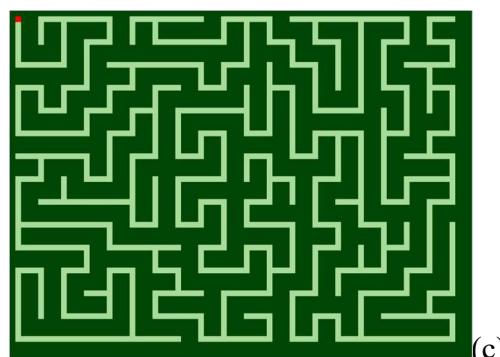
L'avanzamento è stato gestito a passi di due caselle: è possibile aumentare il passo a discapito però della complessità del labirinto, poiché ogni passo aggiunto aumenta lo spessore delle mura del labirinto.



(a)



(b)



(c)

*Figura 3.1 -Cambio della lunghezza del passo di avanzamento
(a) Passo di 2 unità, (b) passo di 3 unità, (c) passo di 4 unità.*

L’algoritmo ad ogni movimento controlla le possibili direzioni da poter intraprendere e sceglie in modo casuale quella in cui proseguire. Se dovesse raggiungere un punto in cui non è più possibile proseguire in una direzione allora rimuove l’ultima mossa inserita, come da legge FILO, e controlla se è possibile intraprendere un’altra direzione da quella posizione.

Per la lettura della mossa si ricorre alla formula inversa come già detto. Alla fine dell’esecuzione ci si ritroverà al punto di partenza e con l’insieme delle mosse vuoto.

Per poter rendere il tutto animato è stato inserito un ciclo a tempo all’interno del quale viene eseguita una sola mossa alla volta e mostrato a schermo il risultato ottenuto.

È possibile modificare l’intervallo di tempo facendo attenzione ad inserire il tempo in millisecondi, unità di misura usata in molti metodi e in molti linguaggi di programmazione.

Per attirare l’attenzione dell’utente sui movimenti che man mano avvengono sullo schermo è stato inserito un riquadro rosso indicante la posizione attuale che l’algoritmo sta valutando.

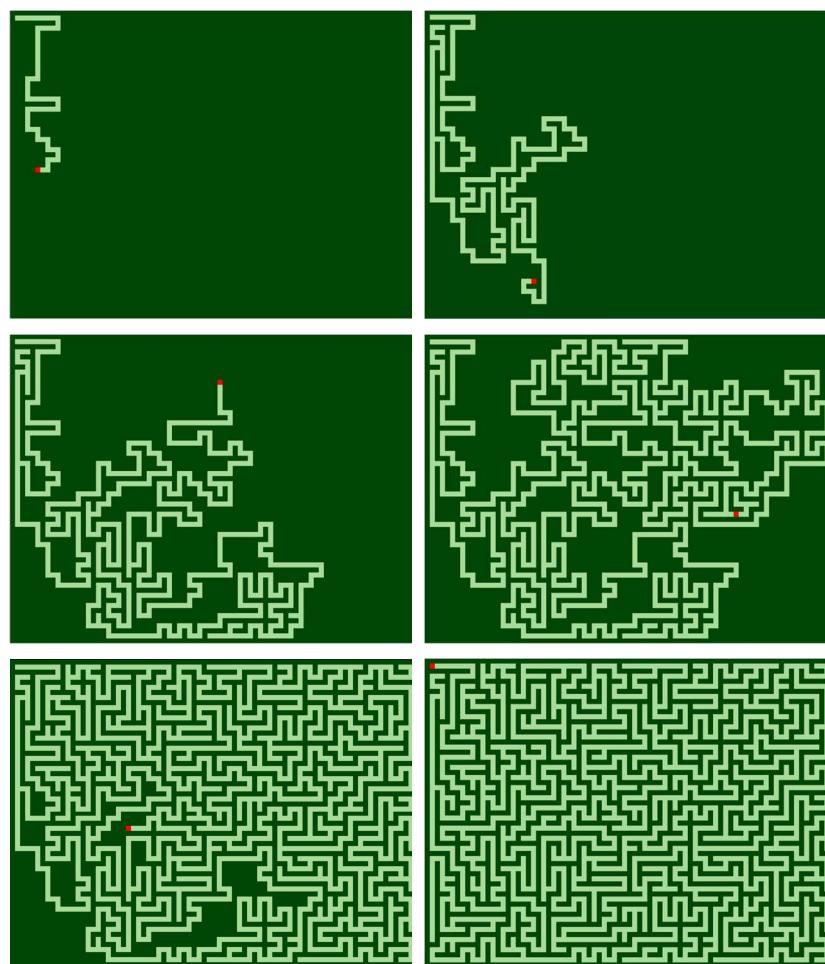


Figura 3.2- Esempio di esecuzione completa

3.2. IL RISOLUTORE

Come già accennato nei paragrafi precedenti il codice riguardante la risoluzione del labirinto contiene al suo interno la costruzione del labirinto stesso evitando, però, l'animazione durante la sua costruzione.

Grazie alla libreria EasyStar.js è possibile ricevere il risultato dell'algoritmo A* applicato al nostro labirinto semplicemente fornendo la matrice del labirinto stesso, il punto da cui si vuole partire(entrata) e quello dove si vuole arrivare(uscita).

Il programmatore ha ben pensato di renderli statici (entrata [1][1], uscita [80] [60]).

```
var easystar = new EasyStar.js();
    easystar.setGrid(this.maze);
    easystar.setAcceptableTiles([0]);
    easystar.findPath(1, 1, gameOptions.mazeWidth-2,
        gameOptions.mazeHeight-2, function(path){
            this.drawPath(path);
        }.bind(this));
    easystar.calculate();
```

Sezione 3.3 - Generazione del percorso tramite EasyStar.js

La libreria restituisce come elemento un array di coordinate ordinato rappresentati tutti i punti del cammino. Come già fatto nel codice precedente con ciclo a tempo, si vivacizza la presentazione mostrando man mano i passi trovati.

```
drawPath(path){
    var i = 0;
    this.time.addEvent({
        delay: 5,
        callback: function(){
            if(i < path.length){
                this.mazeGraphics.fillStyle(0x660000);
                this.mazeGraphics.fillRect
                    (path[i].x*gameOptions.tileSize+1,
                     path[i].y*gameOptions.tileSize+1,
                     gameOptions.tileSize-2,
                     gameOptions.tileSize-2);
                i++;
            }else{
                this.scene.start("PlayGame");
            }
        },
        callbackScope: this,
        loop: true
    });
}
```

Sezione 3.4 - Disegno del percorso

Il programmatore per rendere visibile il suo programma su tutti i diti di device ha ben deciso di implementare una funzione che permette di ridimensionare il labirinto in base alla grandezza della pagina del browser. Questa proprietà di una pagina web si chiama scalabilità e molto spesso è implementata tramite una libreria di Js chiamata Bootstrap.

```
function resize() {  
    var canvas = document.querySelector("canvas");  
    var windowWidth = window.innerWidth;  
    var windowHeight = window.innerHeight;  
    var windowRatio = windowWidth / windowHeight;  
    var gameRatio = game.config.width /game.config.height;  
    if(windowRatio < gameRatio){  
        canvas.style.width = windowWidth + "px";  
        canvas.style.height=(windowWidth/gameRatio)+"px";  
    }  
    else{  
        canvas.style.width=(windowHeight*gameRatio)+"px";  
        canvas.style.height = windowHeight + "px";  
    }  
}
```

Sezione 3.5 - Funzione di ridimensionamento

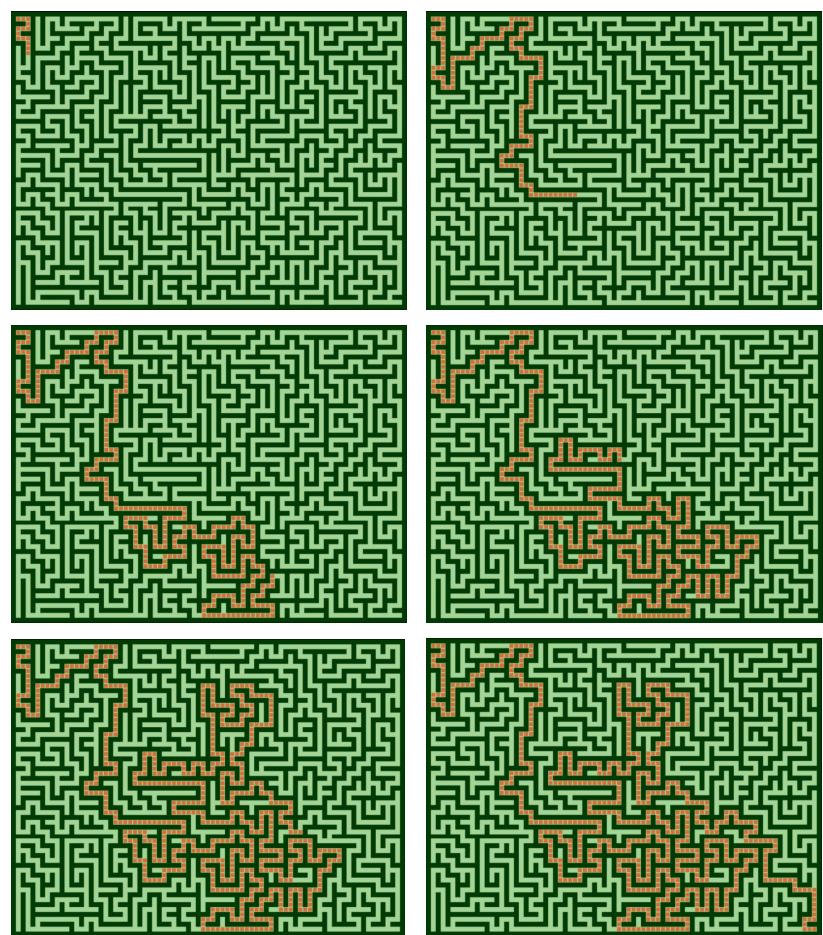


Figura 3.3 - Risultato del risolutore

4. CONCLUSIONI E MIGLIORAMENTI

Il codice dal punto di vista algoritmico è perfettamente funzionante e risponde all'obiettivo che il programmatore si era preposto.

Purtroppo, spesso all'interno dei codici c'è la possibilità di trovare degli errori che possono riguardare sia l'algoritmo che le convenzioni.

All'interno del codice esaminato non vi sono errori riguardanti l'algoritmo ma solo riguardanti le convenzioni. In particolare, la completa assenza di commenti risulta essere un grosso problema, soprattutto perché il codice è presentato come esempio per altri programmatori per poter apprendere come procedere in situazioni analoghe.

Un errore che ha aumentato la complessità della comprensione è stato l'inversione delle variabili *posX* e *posY* che normalmente si riferirebbero alle posizioni rispettivamente lungo l'asse X e l'asse Y.

Buona abitudine sarebbe dividere il codice in funzioni separate: all'interno del codice del risolutore ciò non è stato fatto in maniera completa poiché la costruzione del labirinto poteva essere inserita in una funzione a sé stante.

Sono stati ipotizzati alcuni miglioramenti sia dal punto di vista visivo che da quello implementativo:

- l'inserimento di texture, di cui si può vedere un esempio in Figura 4.1 - Labirinto realizzato con texture ed immagine di riferimento;
- la possibilità di costruire e risolvere labirinti costituiti da altre geometrie (es. cerchi, esagoni).

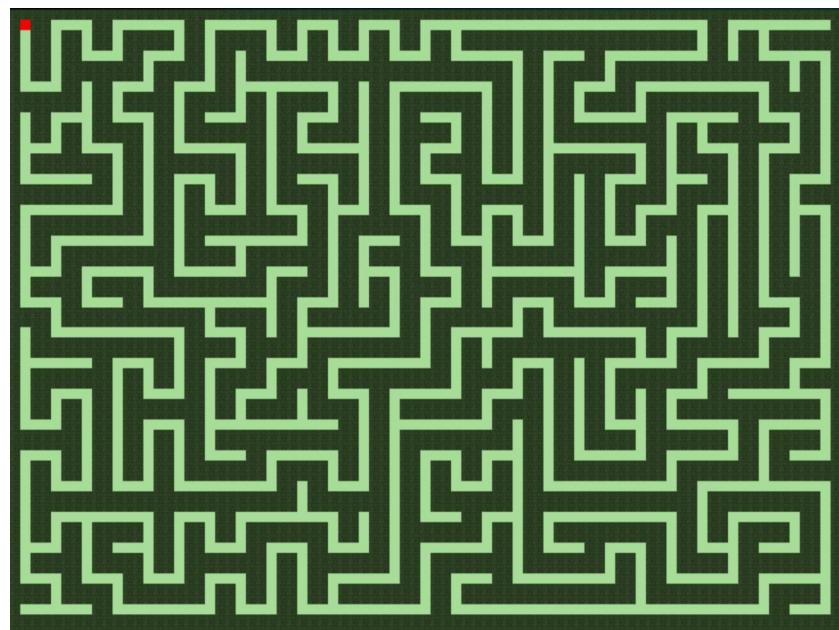


Figura 4.1 - Labirinto realizzato con texture ed immagine di riferimento

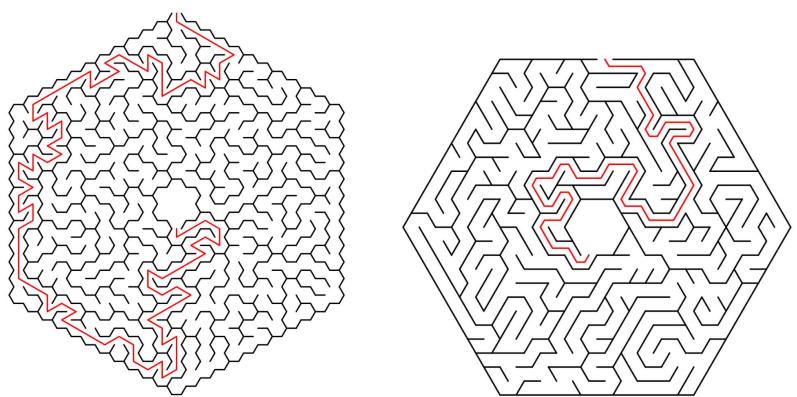
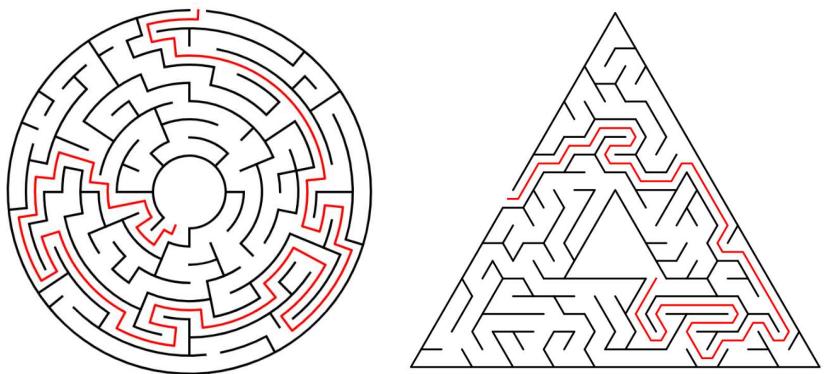


Figura 4.2 - Altri modelli di labirinto

5. SITOGRAFIA

Pure JavaScript perfect tile maze generation – Emanuele Feronato

Pure JavaScript A* maze solving – Emanuele Feronato

<https://www.emanueleferonato.com/>

Progetto Labirinto

http://www.mat.unimi.it/users/alzati/Geometria_Computazionale_98-99/apps/labirinto/index.html

Think Labyrinth: Maze Algorithms

Think Labyrinth: Maze Glossary

<http://www.astrolog.org/>

GitHub - Asynchronous A* pathfinding API written in Js.

<https://github.com/prettymuchbryce/easystarjs>

Getting Started with Phaser 3

<https://phaser.io/tutorials/getting-started-phaser3>

Maze Generator

<http://www.mazegenerator.net/>

GitHub - Plugin for Phaser which includes Easystar.js

https://github.com/appsbu-de/phaser_plugin_pathfinding

easystar.js

<https://easystarjs.com/>

A* Search Algorithm - GeeksforGeeks

<https://www.geeksforgeeks.org/a-search-algorithm/>

Quel labirinto che chiamiamo vita

<http://espresso.repubblica.it/visioni/cultura/2016/04/03/news/eugenio-scalfari-quel-labirinto-che-chiamiamo-vita-1.256713>

Focus – Storia del labirinto

<https://www.focus.it/cultura/storia/al-centro-del-labirinto?gimg=6027#img6027>

