GeneratoreLabirinti2D

Documentazione

[1 Introduzione 2](#_Toc59061644)

[1.1 Informazioni sul progetto 2](#_Toc59061645)

[1.2 Abstract 2](#_Toc59061646)

[1.3 Scopo 2](#_Toc59061647)

[2 Analisi 3](#_Toc59061648)

[2.1 Analisi del dominio 3](#_Toc59061649)

[2.2 Analisi e specifica dei requisiti 3](#_Toc59061650)

[2.3 Use case 4](#_Toc59061651)

[2.4 Pianificazione 5](#_Toc59061652)

[2.5 Analisi dei mezzi 5](#_Toc59061653)

[2.5.1 Software 5](#_Toc59061654)

[2.5.2 Hardware 5](#_Toc59061655)

[3 Progettazione 6](#_Toc59061656)

[3.1 Design dell’architettura del sistema 6](#_Toc59061657)

[3.2 Design dell’interfaccia 6](#_Toc59061658)

[3.3 Design dei flussi principali 7](#_Toc59061659)

[4 Implementazione 9](#_Toc59061660)

[5 Test 18](#_Toc59061661)

[5.1 Protocollo di test 18](#_Toc59061662)

[5.2 Risultati test 20](#_Toc59061663)

[5.3 Mancanze/limitazioni conosciute 21](#_Toc59061664)

[6 Consuntivo 22](#_Toc59061665)

[7 Conclusioni 23](#_Toc59061666)

[7.1 Sviluppi futuri 23](#_Toc59061667)

[7.2 Considerazioni personali 23](#_Toc59061668)

[8 Bibliografia 24](#_Toc59061669)

[8.1 Sitografia 24](#_Toc59061670)

[9 Allegati 24](#_Toc59061671)

# Introduzione

## Informazioni sul progetto

Allievi coinvolti nel progetto: Michea Colautti

Classe: Informatica 3AC presso la sede Scuola Arti e Mestieri Trevano

Docenti responsabili: Geo Petrini, Luca Muggiasca

Data inizio: 01.09.2020  
Data consegna: 17.12.2020

## Abstract

In momenti di noia, o semplicemente quando si ha voglia di svagarsi un po’, provare a risolvere degli enigmi di tipologia e difficoltà variabili può essere un ottimo passatempo. Il Generatore di Labirinti 2D offre questa possibilità; grazie alla sua espandibilità e alla grande personalizzazione che si può dare al labirinto (forma e dimensione, numero di soluzioni possibili, inserimento di loop, ecc.) questo programma può diventare un buon passatempo con cui mettersi alla prova. Non solo, data la possibilità di esportare i parametri e di condividerli in maniera agevole per permettere la generazione di enigmi identici, è possibile confrontarsi e sfidarsi con gli amici. Anche la sua facilità d’uso e di accesso contribuisce a renderlo un papabile passatempo per tutti.

## Scopo

L’obiettivo del progetto è lo sviluppo di un’applicazione per la generazione di labirinti 2D.

In questa generazione dovrà essere possibile configurare dei parametri come la forma esterna e del percorso, l’inserimento di percorsi a loop, le posizioni di partenza e fine, le dimensioni del labirinto, l possibilità di esportare la soluzione tramite un’immagine e la generazione di soluzioni multiple. l’utente

L’utente non deve solo avere la possibilità di salvare i parametri oppure un seed di generazione, ma deve essere anche in grado di completare il labirinto online, lasciando dietro di sé la “scia” con il percorso seguito.

La generazione non dovrà essere del tutto casuale, infatti caricando poi i parametri salvati in precedenza, il labirinto dovrà sempre essere uguale. Il codice di generazione dovrà quindi dipendere fortemente dai parametri inseriti.

# Analisi

## Analisi del dominio

Per questo progetto è stato chiesto di sviluppare un software, web nel mio caso, che permetta di generare labirinti 2D. Gli utenti a cui esso sarà indirizzato non dovranno avere particolari conoscenze informatiche, l’interfaccia grafica semplice permetterà infatti a tutti gli utenti il libero.

I labirinti saranno generati in maniera randomica e anche se esistono diversi siti web capaci di generare enigmi simili pochi di questi combinano la grande personalizzazione del labirinto con la possibilità di completarlo comodamente online, oppure integrano la possibilità di scaricare o caricare dei parametri da file.

Ciò toglie quindi un aspetto che mi è piaciuto del progetto, ovvero la possibilità di condividere il labirinto.

Non sarà necessario creare strati di sicurezza, in quanto il progetto non richiede un database ma sarà un'unica pagina internet. Il codice java script è dunque in chiaro.

## Analisi e specifica dei requisiti

|  |  |
| --- | --- |
| **ID: REQ-001** | |
| **Nome** | Home Page |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Un pagina html semplice in cui disporre i controlli per la personalizzazione del labirinto |

|  |  |
| --- | --- |
| **ID: REQ-002** | |
| **Nome** | Campi di controllo per la personalizzazione |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Bottoni e campi di testo per la personalizzazione del labirinto |

|  |  |
| --- | --- |
| **ID: REQ-003** | |
| **Nome** | Logica per la generazione del labirinto (partendo da un seed) |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Un algoritmo che generi il labirinto partendo da un numero casuale/inserito dall’utente |

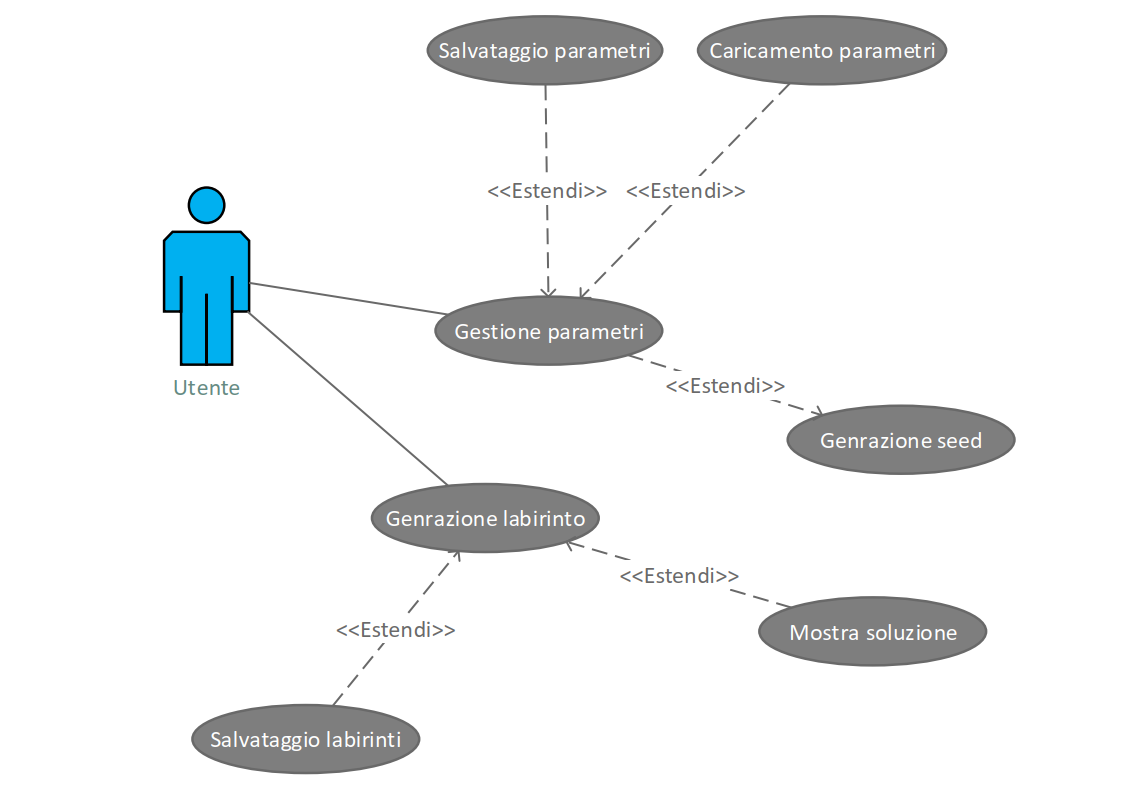
|  |  |
| --- | --- |
| **ID: REQ-004** | |
| **Nome** | Metodo che permetta il salvataggio del labirinto |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** |  |

|  |  |
| --- | --- |
| **ID: REQ-005** | |
| **Nome** | Libreria (JQuery o simili) |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Per permettere all’utente di disegnare il labirinto |

|  |  |
| --- | --- |
| **ID: REQ-006** | |
| **Nome** | Sistema di salvataggio dei parametri di personalizzazione |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** | Si può utilizzare il metodo per salvare il labirinto per fare una foto ai parametri selezionati |

|  |  |
| --- | --- |
| **ID: REQ-007** | |
| **Nome** | Metodo per la conversione da caratteri ascii a muri |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** | Si può una canvas e un metodo dedicato trovato su internet |

## Use case



In questo digramma dell’use case possiamo osservare come l’utente accede al sito e imposta i parametri, dopodiché genera il suo labirinto. A questo punto può scegliere tra molte opzioni, tra cui mostrare la soluzione, salvare il labirinto oppure i suoi parametri. L’utente può anche caricare i parametri che creeranno il labirinto. Si noti che la generazione del seed avviene solamente se un seed non è stato immesso.

## Pianificazione

|  |
| --- |
|  |

## Analisi dei mezzi

Per questo progetto saranno necessari i seguenti mezzi:

* Microsoft Project: Per la creazione del Gantt.
* Visual Studio Code: Per la scrittura del codice
* HTML/CSS/Javascript
* Windows 10 o un altro OS compatibile con tutti i programmi utilizzati
* Chrome: per visualizzare il risultato
* Microsoft Visio: Per lo schema di use case.
* Microsoft Project: Per il Gantt
* ProjectLibre: Per il Gantt

### Software

Windows 10: Come OS base

Visual Studio Code: per scrivere il codice

Google Chrome: Per testare il sito

mockflow.com: Per la progettazione dei flussi e della GUI

### Hardware

L’applicazione sarà eseguibile su un qualsiasi browser da PC o Mac, il browser con la migliore ottimizzazione sarà Google Chrome.

Durante lo sviluppo utilizzerò il PC messo a disposizione dalla scuola e Visual Studio Code per lo sviluppo.

# Progettazione

## Design dell’architettura del sistema

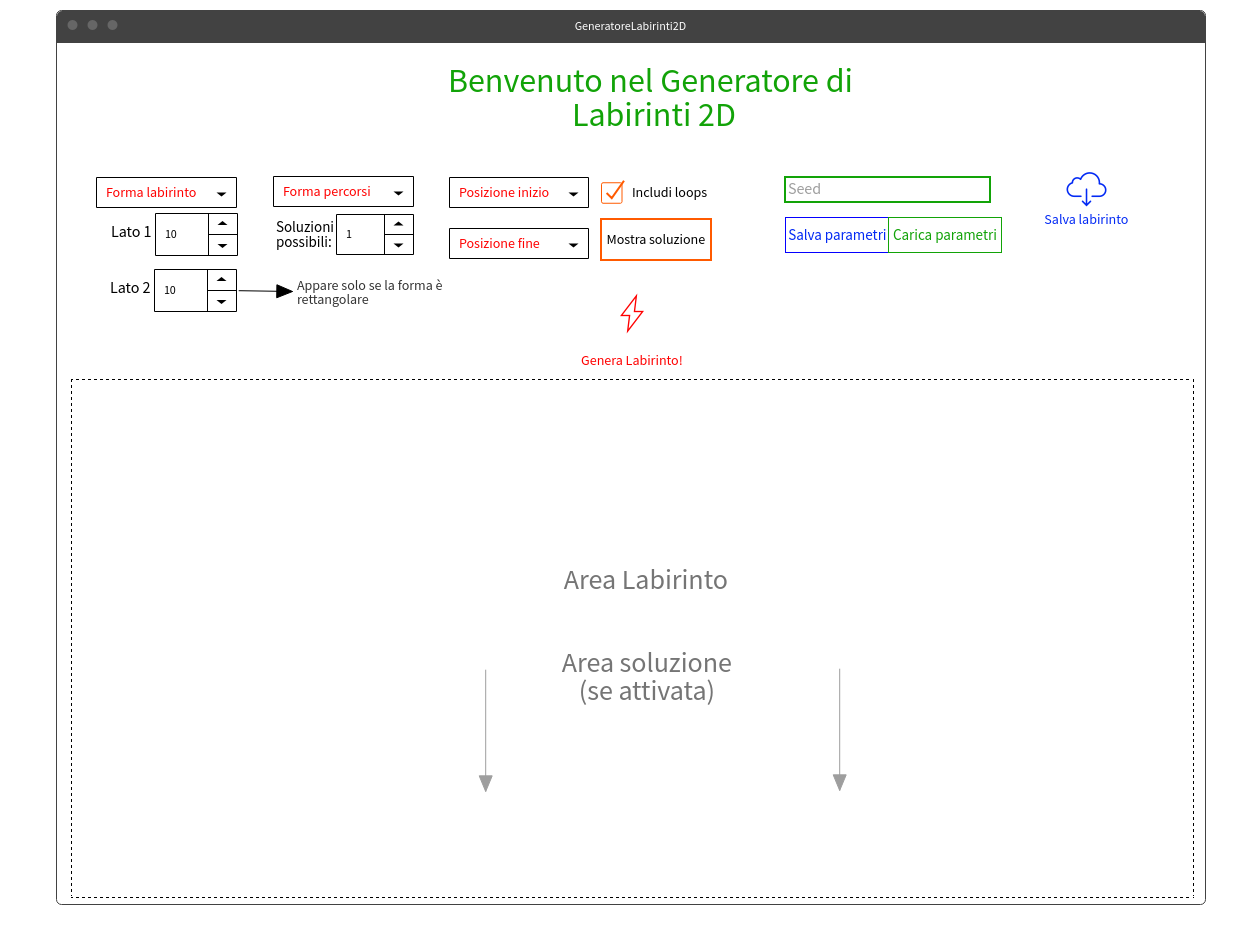
Quella che sto realizzando sarà un applicazione web stand-alone.

Pur cercando di abbellirla diversificando i colori la mia applicazione avrà una grafica estremamente semplice, e sarà tesata principalmente su Google Chrome, per quanto anche Safari sarà in grado di supportarla.

Come detto anche prima l’app sarà sviluppata solo con Java script. Non prevedo di rendere l’applicazione responsive, sarà quindi difficile da utilizzare su un telefono ma se si presenterà l’opportunità, in fase di implementazione farò in modo che il tutto sia responsive.

La logica è esplicata nel capitolo **design dell’interfaccia**

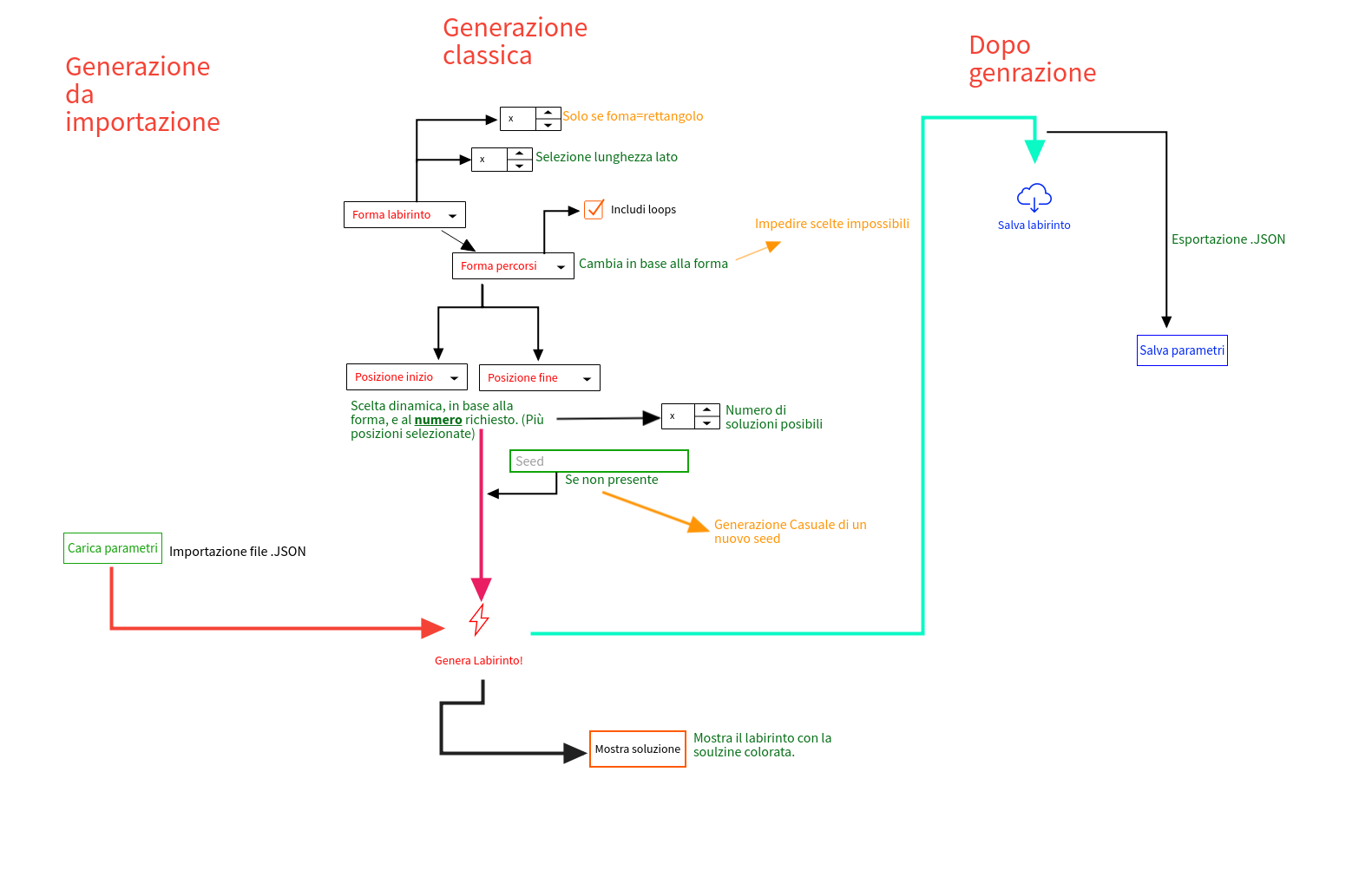
## Design dell’interfaccia



Qui sopra c’è una rappresentazione, creata grazie a MockFlow, di come apparirà la mia applicazione una volta aperta. Ho deciso di porre tutti i controlli nella parte superiore della pagina, quando un labirinto verrà generato l’area tratteggiata verrà riempita da esso.

Le frecce stanno a significare che, quando verrà richiesta la soluzione, essa apparirà sotto il labirinto già presente. Non l’ho rappresentato graficamente per non appesantire e rendere troppo grande inutilmente l’immagine. Il campo “Lato 2” è presente principalmente per rappresentare l’applicazione nella sua interezza ma, come spiegato meglio nel prossimo capitolo, per la maggior parte del tempo sarà nascosto.

## Design dei flussi principali



Qui sopra ho schematizzato quello che sarà il funzionamento generale della mia app.

Ho diviso questo schema in 3 sezioni, immaginando quello che potrebbe essere l’uso tipico.

Nella prima sezione, ho rappresentato la prima situazione possibile, ovvero quella in cui l’utente carica i parametri (scelte multiple e seed) tramite un file .json, dopodiché crea il labirinto.

Ora le scelte possibili sono 3, nessuna delle quali esclude l’altra.

La prima è quella in cui l’utente può decidere se salvare il labirinto, la seconda è se mostrare immediatamente la soluzione. La terza, non mostrata nel flusso, è quella in cui l’utente risolva il labirinto direttamente sulla pagina, trascinando il mouse.

Nella seconda sezione, è invece rappresentato quello che è il funzionamento dell’app nel caso in cui l’utente decida di creare un labirinto da 0. La struttura che ho scelto per rappresentare quelli che sono i vari controlli non è casuale, ho cercato di partire con le scelte più generiche, come la forma, fino a quelle più specifiche come le posizioni di inizio e fine. Cercherò di mantenere questa struttura anche nello sviluppo del codice.

Durante la scelta delle opzioni sono costretto a eseguire un controllo per evitare che l’utente faccia scelte irrealizzabili. Ad esempio, la forma del labirinto e la forma dei percorsi devono sempre coincidere, solo quando la forma è esagonale i percorsi possono essere esagonali e triangolari, e viceversa. Inoltre se la forma prescelta è quella rettangolare devo far apparire un campo numerico per scegliere la dimensione del secondo lato, altrimenti uscirebbero sempre labirinti quadrati. Questo aspetto è evidenziato anche nella GUI

Infine la terza sezione è comune ad entrambe le sezioni precedenti, l’unica cosa che permette di fare è infatti salvare i parametri e il labirinto, definendo la dimensione del svg.

Si noti che, unendo con una freccia il pulsante “Salva parametri” con quello “carica parametri” si può vedere l’app come un ciclo. È anche questo che mi ha spinto al posizionare i due pulsanti vicini nella progettazione della GUI.

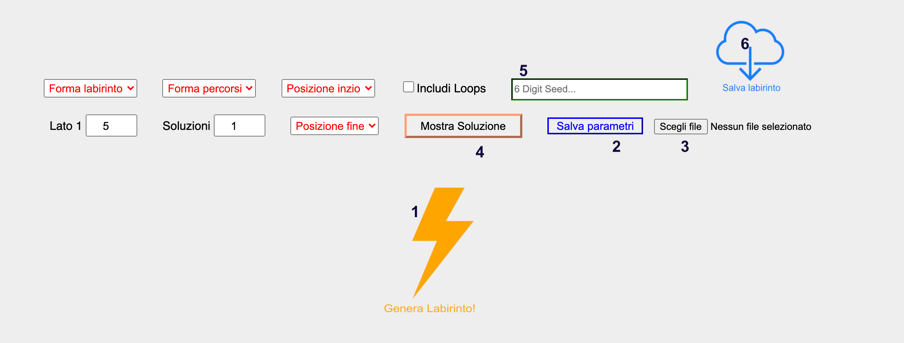
# Implementazione

* 1. **GUI**

Dopo aver definito bene come sarebbe stata l’architettura della mia applicazione, ho incominciato a scrivere il codice vero è proprio. In realtà. A questo punto non avevo un idea precisa dell’algoritmo di generazione, ma prevedevo di pensarci in continuazione mentre sviluppavo il resto…

Ho quindi decido di iniziare dalla GUI, in quanto volevo prima avere una base solida in cui inserire i parametri di generazione.

Usando quindi HTML e CSS ho definito tutti i controlli.



In questa foto si può vedere la mia GUI, i controlli contrassegnati da un numero sono quelli che dovrebbero effettua un operazione, ma ad oggi solo 3 effettuano la loro operazione, gli altri non fanno pressoché nulla.

Per i controlli ho usato i classici controlli di HTML, quindi:

* Select
* Button
* Text/number
* Bottoni mascherati da immagini

Il controllo più “complesso” è la select, eccone un esempio:

<select id="mShape" class="dropD" onchange="showControl()">//maze Shape

<option value='def' selected>Forma labirinto</option>

<option value='Rect'>Rettangolare</option>

<option value='Circ'>Rotondo</option>

<option value='Tria'>Triangolare</option>

<option value='Esag'>Esagonale</option>

</select>

🡪In questa select il valore selezionato tra le forme di labirinto papabili è quello di “default”. Sarà proprio questo il valore di cui si verificherà la non-presenza nei controlli Javascript.

Ho scelto questa select perché presenta un caso particolare, ovvero la funzione showControl().

Questa funzione permette, tramite un istruzione specifica, di controllare la forma prescelta, e se questa corrisponde al quadrato allora verrà mostrato un campo per il secondo lato dello stesso. Ecco l’istruzione:

var comp=document.getElementById("lato2");

if(document.getElementById("mShape").selectedIndex==1){

comp.style.display = "block";

}else{

comp.style.display = "none";

}

* 1. **Controllo parametri**

Questa parte sarà più complicata da documentare, tutta la fase di controllo dei parametri è infatti una lunga sequenza di if-else, che controllano la correttezza e compatibilità dei parametri inseriti. Ho cercato di seguire l’ordine definito nella creazione dello schema di flusso. Questa fase si basa su due funzioni principali:

* getControls();
* areValidParam();

**FUNZIONE 1:**

Come si può forse evincere dal nome, la prima serve ad ottenere i valori dai loro campi, occupandosi quindi di assegnare alle variabili globali il valore contenuto nei campi. Si occupa però anche di validare un minimo questi parametri sfrutta infatti una serie di metodi aiuto per controllare i campi, come ad esempio:

isDropFull(control)

🡪Si occupa di verificare se i 4 drop down sono completati con dei valori differenti da quello di default, menzionato prima.

Il metodi viene quindi eseguito 4 volte e restituisce un valore booleano.

Se viene ritornato true il programma prosegue, altrimenti se arresta e viene restituito un messaggio di errore.

Il parametro passato corrisponde a una delle variabili globali che rappresentano i dropdown, ad esempio mShape.

isValidSeed(num)

🡪Si occupa di verificare se il seed è presente e valido o meno, se è valido ritorna true, se non è valido false.

Se viene ritornato false perché non è compatibile con la sintassi viene restituito un messaggio di errore, dopodiché ne viene generato uno nuovo. Se invece non è presente del tutto viene creato un nuovo seed.

isNumFull()

🡪Viene richiamato ogni volta che bisogna controllare un campo di tipo number, verifica non solo la presenza, ma anche che vengano messi numeri validi. (maggiori di 0, senza lettere, ecc.)

Ritorna true solo se i tutti i test sono superati.

La funzione completa e i suoi metodi si possono trovare nel file Control.js

**FUNZIONE 2:**

La seconda funzione, areValidParam(), viene eseguita solamente se la prima è andata a buon fine:

nel file html principale è presente, infatti, la funzione master(num), che esegue queste due funzioni in sequenza. Eccone un estratto

var dum=getControls();

if(dum){

var dum2=areValidParam();

..

Il parametro num è un semplice identificativo, mi serve per capire chi esegue la funzione master, in quanto viene usata anche nella procedura di salvataggio dei parametri.

Dopo aver passato il controllo di master, la funzione può essere eseguita.

Molto più semplice della precedente, areValidParam(), si occupa solamente di verificare che forma di labirinto e percorsi siano compatibili, secondo specifiche.

Per completezza riporto il messaggio di errore che verrà restituito nel caso in cui le scelte fatte siano incongruenti.

alert(

"INSERISCI FORME COMPATIBILI:"+'\n'+

"Rettangolare-Paralleli"+'\n'+

"Rotondo-Curvilieni"+'\n'+

"Triangolare-Triangolari"+'\n'+

"Esagonale-Esagonali"+'\n'+

"Esagonale-Triangolari"

)

La funzione completa e i suoi metodi si possono trovare nel file CheckParam.js

* 1. **Salvataggio parametri**

La procedura di salvataggio dei parametri è relativamente semplice, e si può eseguire cliccando sul bottone “Salva parametri”.

Questo bottone eseguirà nuovamente la funzione master(num), questa volta con un parametro differente rispetto a prima.

Master eseguirà in maniera analoga a prima entrambe le funzioni getControls() e areValidParam(), se entrambe ritornano true potrà eseguire la funzione getControlsValue().

Questa funzione, situata in Control.js permette di inserire in un **array associativo,** ovvero un array con degli elementi identificati da una stringa e non da un numero, tutti i valori salvati nella variabili globali.Questo array viene poi ritornati. Eccone un estratto.

var values = {

mazeShape: mazeShape,

pathShape: pathShape,

loops: loops,

...

};

Questo array verrà poi passato, sempre tramite master, alla funzione saveParam(values).

Quest’ultima si occuperà di inserire l’array in un blob, ma solo dopo averla “convertita” in una stringa compatibile con JSON. Dopo una serie di istruzioni il file viene scaricato, simulando un click su un url di download. (Javascript generalmente non permette il salvataggio di file). Riporto la funzione per intero, in quanto non è semplicissima ed è corta.

function saveParam(param){

var saveData = (function () {

var a = document.createElement("a");

document.body.appendChild(a);

a.style = "display: none";

return function (data, fileName) {

var json = JSON.stringify(data),

blob = new Blob([json], {type: "octet/stream"}),

url = window.URL.createObjectURL(blob);

a.href = url;

a.download = fileName;

a.click();

window.URL.revokeObjectURL(url);

};

}());

var data = param,

fileName = "Parameters.json";

saveData(data, fileName);

}

La funzione è situata Saving.js, mentre getContorlsValue(), è situata in Control.js.

* 1. **Upload parametri**

In linea teorica, la funzione Upload, che non sono riuscito a completare, dovrebbe caricare i parametri nei campi. Come detto la funzione è ancora in fase embrionale, quello che per ora è “catturare” il file JSON e basta. Non la includo in quanto non funzionante, ma può essere trovata nel file Uploading.js.

* 1. **Disegno del labirinto**

Per la generazione del labirinto torno a sfruttare la funzione master, ho aggiunto una porzione di codice sotto areValidParam(), che in base al valore di ritorno eseguirà nuovamente getControlsValue(), per poi passare il tutto alla funzione setParam(), che darà poi inizio alla sequenza di funzioni in maze.js.

Maze.js contiene tutti i metodi per il disegno dei labirinti, nello specifico:

* setParam(values)
* generateMaster()
* genrateRect(ctx)
* generateCiric(ctx,canvas)
* generateTria(ctx)
* helperRect(ctx,i)
* draw(ctx,posW,posH,interruptW,interruptH)
* helperCirc(ctx,arco,canvas)
* findNum(sValue,eValue)

Queste non sono tutte le funzioni necessarie per generare tutto ciò che è richiesto, ma solo quelle che io ho creato e sviluppato.

**setParam(values)**

È la funzione che mette in comunicazione il codice HTML con l’algoritmo di generazione, tramite questa funzione il programma riceve l’array con tutti i valori (values), e assegna i parametri necessari per la generazione che ho sviluppato fino a questo punto. I valori vengono salvati direttamente nelle nuove variabili globali, simi a quelle presenti in Control.js, ma modifica i valori dei lati. Infatti ho previsto che dovessi esserci un rapporto di 1 a 10 tra la dimensione scelta dall’utente e le coordinate della canvas. Per questo ho fatto una variabile globale w che regola appunto la dimensione.

Finiti i processi primari, viene eseguito generateMaster.

**generateMaster()**

È una delle funzioni principali del programma, si occupa di definire la canvas e il riferimento che ci servirà per disegnare:

canvas = document.getElementById("maze");

width=canvas.width;

var c = document.getElementById("maze");

var ctx = c.getContext("2d");

Dopodiché si occupa di modificare il seed, da un numero viene trasformato in un array.

var dum = new Array();

for(var i=0; i<seed.length; i++){

dum[i]= parseInt(seed.charAt(i));

}

seed=dum;

Completato anche questo processo la funzione richiama uno dei metodi di disegno dei contorni (genrateRect, generateCirc, generateTria), che eseguiranno il resto del lavoro.

**genrateRect(ctx)**

Questa funzione si occupa di disegnare solamente il contorno dell’labirinto, occupandosi però di svuotare la canvas prima.

Dopodiché viene chiamato helperRect(ctx,i);

🡪ctx è il riferimento per disegnare, mentre i è il numero di volte che il programma verrà eseguito.

(rappresenta un tentativo di espansione del codice, che ho preferito tenere)

**generateCiric(ctx,canvas)**

Questa funzione si occupa di disegnare solamente il contorno dell’labirinto, occupandosi però di svuotare la canvas prima.

Dopodiché viene chiamato helperCirc(ctx,arco,canvas);

🡪Ho bisogno, come prima, di ctx per il disegno, ma qui passo anche l’arco e il riferimento alla canvas, mi serviranno entrambi nel disegno.

**generateTria(ctx)**

Questa funzione permette solamente di generare il boro del triangolo, per poi fermarsi senza proseguire.

Per farlo ho dovuto trovare alcuni dati come il centro della canvas, per poi sfruttare il teorema del seno per determinare l’altezza del triangolo e lo spostamento a destra/sinistra. Una volta trovati questi due dati (down e lefRrig), posso tracciare le line che compongo il triangolo, in quanto ottengo le coordinate x/y all’interno della canvas. Allego il codice per una maggiore comprensione.

var width = width;

var cent=width/2;

ctx.beginPath();

ctx.moveTo(cent, 10);

var down=(Math.sin(60)\*(rows/Math.sin(90))\*-1);

var lefRig=(Math.sin(60)\*(rows/Math.sin(90))\*-1)/2;

ctx.lineTo(cent-lefRig,10+down);

ctx.moveTo(cent-lefRig,10+down);

ctx.lineTo(cent+lefRig,10+down);

ctx.lineTo(cent, 10);

ctx.stroke();

**helperRect(ctx,i)**

Questo metodo permette di calcolare svariati dati, li esplicherò uno per uno.

**posW e posH**

Sono la posizione su x e su y da dove partiranno i miei percorsi, tracciando delle linee immaginarie partendo da questi punti e andando fino in fondo alla forma dividere il labirinto in 4.

Entrambi questi valori vengo trovati grazie al metodo findNum(sValue,eValue), che restituirà un numero basandosi sui due numeri passati, che saranno due indici del seed, ora trasformato in array.

Entrambi i valori restituiti verranno moltiplicati per w per una coerenza nel disegno

**interruptW e interruptH**

Sono la posizione su x e y corrispondendoti a dove le linee si interromperanno, per permettere un passaggio nel percorso dell’ipotetico labirinto. I numeri sono trovati in maniera analoga a sopra.

Facendo un po’ di prove mi sono accorto che era necessario sistemare e approssimare questi valori, infatti non solo dovevano essere multipli di 10 (avrei ottenuto spazi minuscoli e disuguali all’interno della forma), ma dovevano anche essere maggiori di un certo valore e minori di un altro, per permettere al disegno di genarsi al meglio. Ho dovuto quindi implementare dei controlli aggiuntivi, che riporto qui.

//Delimitazione valori

if(posW<15){

posW=w;

}else if(posW>cols){

posW-=cols;

}

if(posH<15){

posH=w;

}else if(posW>rows){

posH-=rows;

}

if(interruptW<15){

interruptW+=5;

}else if(interruptW>45){

interruptW-=5;

}

if(interruptH<15){

interruptH+=-5;

}else if(interruptH>45){

interruptH-=5;

}

//Arrotondamento a multiplo di 10

posW=Math.ceil(posW/10)\*10;

posH=Math.ceil(posH/10)\*10;

interruptH=Math.ceil(interruptW/10)\*10;

interruptW=Math.ceil(interruptH/10)\*10;

Una volta trovati tutti i valori posso passarli a draw che, come il nome suggerisce, disegnerà il labirinto.

**draw(ctx,posW,posH,interruptW,interruptH)**

Questo metodo è colui che disegna tutto, mi trovo in difficoltà a commentarlo in quanto è molto procedurale, riporterò e spiegherò quindi solo un istruzione alla volta, in quanto le altre sono delle copie leggermente modificate.

**STEP 1**

ctx.beginPath();

ctx.moveTo(posW, 0);

ctx.lineTo(posW,cols-interruptW);

ctx.moveTo(posW, cols-interruptW+w);

ctx.lineTo(posW,cols);

Questa sequenza di codice mi permette di disegnarne una linea che parte dal lato sinistro del labirinto ad altezza posW fino alla coordinata definita da posW e la differenza tra la lunghezza totale del labirinto – il punto di interruzione.

Mi muovo poi di uno spessore (w), in modo da lasciare un buco, e continuo la linea fino in fondo.

w

ctx.moveTo(posW+w, 0);

ctx.lineTo(posW+w,cols-interruptW);

ctx.moveTo(posW+w, cols-interruptW+w);

ctx.lineTo(posW+w,cols);

ctx.stroke();

La seconda parte di istruzioni mi permette di disegnare una linea uguale a quella sopra, ma shiftata di w, in modo da creare un cerchio.

**Ripeto poi lo stesso processo per quanto riguarda le linee verticali**

ctx.moveTo(0, posH);

ctx.lineTo(rows-interruptH,posH);

ctx.moveTo(rows-interruptH+w, posH);

ctx.lineTo(rows,posH);

ctx.moveTo(0, posH+w);

ctx.lineTo(rows-interruptH,posH+w);

ctx.moveTo(rows-interruptH+w, posH+w);

ctx.lineTo(rows,posH+w);

**Una volta fatto questo avrò un qualcosa di questo tipo:**

§

Mi muoverò quindi alle 8 estremità dei segmenti blu, e disegnerò delle line verso le pareti adiacenti, nel seguente modo:

ctx.moveTo(rows-interruptH, cols-interruptW);

ctx.lineTo(0,cols-interruptW);

ctx.moveTo(rows-interruptH, cols-interruptW+w);

ctx.lineTo(0,cols-interruptW+w);

ctx.moveTo(cols-interruptH+w, cols-interruptW);

ctx.lineTo(cols,cols-interruptW);

ctx.moveTo(cols-interruptH+w, cols-interruptW+w);

ctx.lineTo(cols,cols-interruptW+w);

ctx.moveTo(cols-interruptW, 0);

ctx.lineTo(cols-interruptW,rows);

ctx.moveTo(cols-interruptW+w, 0);

ctx.lineTo(cols-interruptW+w,rows);

Una volta fatto questo dovrò togliere le pareti dei percorsi all’ interno, che si possono vedere anche nle disegno. Per farlo svuoterò la canvas in corrispondenza dei percorsi, nel seguente modo.

ctx.clearRect(1, posH+1, rows-2, w-2.5);

ctx.clearRect(posW+1, 1, w-2, cols-2);

**helperCirc(ctx,arco,canvas)**

Questo metodo permette di disegnare un principio di labirinto circolare, quello che fa è, banalmente, trovare il numero di cerchi, circNum, ottenuto sempre col solito metodo findNum(sValue,eValue).

Dopodiché in base al numero dei cerchi trova la dimensione dei cerchi, circSizes, dividendo il numero dei cerchi per la dimensione del labirinto.

Infine viene trovato il numero di raggi, radNum, che determinerà in quanti spicchi sarà diviso il labirinto. (non completato).

Infine ottenuti questi dati disegna dei cerchi concentrici, tracciando un primo raggio.

var circNum=findNum(0,5);

var circSizes=cols/circNum;

var radNum=findNum(1,4)\*2;

var amp=360/radNum;

for(var i=0;i<=circNum;i++){

ctx.arc(canvas.width/2, canvas.height/2, arco-i\*circSizes, 0, 2\*Math.PI);

}

ctx.stroke();

**findNum(sValue,eValue)**

Il metodo findNum è un metodo che ho ideato per ottenere valori sempre uguali con un determinato seed, per fare questo prende i due parametri d’entrata sValue e eValue, che stanno per startValue e endValue.

Dopodiché crea un nuovo seed, ordinando in ordine crescente quello originale e sottraendo i 2 valori.

function findNum(sValue,eValue){

var newSeed=seed.sort();

return (newSeed[eValue] - newSeed[sValue])/2;

}

**Ho rimosso la funzione che permetteva il tracciamento del mouse, i quanto completamente non funzionante.**

# Test

## Protocollo di test

In questo capitolo, basandomi sui requisiti da me definiti, ho cercato di elencare tutti i test che eseguirò, essi mi aiuteranno molto nel verificare il buon funzionamento di ogni parte del mio lavoro.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-001  REQ-002 | **Nome:** | Test sui campi |
| **Descrizione:** | Verificare che tutti I campi per il controllo utente non permettano scelte impossibili | | |
| **Prerequisiti:** | Campi per il controllo completati | | |
| **Procedura:** | 1. Selezionare l’opzione “rettangolare” come forma del labirinto 2. Verificare che l’unica opzione per la forma dei percorsi sia “rettangolari” 3. Verificare la presenza dei due campi numerici per i lati 4. Ripetere per tutte le forme a parte triangolare e esagonale 5. Se viene selezionato “triangolare” verificare la presenza dell’opzione “esagonali” per i percorsi e viceversa. | | |
| **Risultati attesi** | Scelte funzionanti | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-002  REQ-002 | **Nome:** | Test sull’integrità dei campi |
| **Descrizione:** | Verificare che non sia possibile lasciare campi non completati | | |
| **Prerequisiti:** | Campi per il controllo | | |
| **Procedura:** | 1. Completare tutti i campi (eccetto seed) e lasciarne uno vuoto 2. Provare a genare il labirinto 3. Ripetere operazione lasciando ogni volta un campo vuoto | | |
| **Risultati attesi** | Avviso che inviti l’utente a completare tutti i campi | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-003  REQ-003 | **Nome:** | Generazione seed |
| **Descrizione:** | Se il seed non è presente deve essere generato | | |
| **Prerequisiti:** | Campi per il controllo | | |
| **Procedura:** | 1. Completare tutti i campi (eccetto seed) 2. Provare a genare il labirinto | | |
| **Risultati attesi** | Campo seed deve essere riempito | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-004  REQ-003 | **Nome:** | Algoritmo di generazione |
| **Descrizione:** | L’algoritmo, basandosi sul seed e sui parametri, genera un labirinto consono alle scelte dell’utente | | |
| **Prerequisiti:** | Algoritmo di generazione completo, campi di controllo e seed | | |
| **Procedura:** | 1. Scegliere dei parametri 2. Provare a genare il labirinto 3. Ripetere con tutte le combinazioni possibili | | |
| **Risultati attesi** | Il labirinto deve attenersi alle istruzioni dell’utente | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-005  REQ-003 | **Nome:** | Soluzione |
| **Descrizione:** | Ad un click dell’utente, la soluzione viene mostrata | | |
| **Prerequisiti:** | Un labirinto generato | | |
| **Procedura:** | 1. Scegliere dei parametri 2. Provare a genare il labirinto 3. Generazione della soluzione 4. Ripetere con tutte le combinazioni possibili | | |
| **Risultati attesi** | Il programma deve mostrare una copia del labirinto con la soluzione in chiaro | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-006  REQ-006 | **Nome:** | Procedura di salvataggio/caricamento parametri |
| **Descrizione:** | Una volta generato un labirinto, al click dell’utente, il programma salva un file json con i parametri dell’utente | | |
| **Prerequisiti:** | Campi di controllo e seed completati | | |
| **Procedura:** | 1. Scegliere dei parametri 2. Provare a genare il labirinto 3. Provare a salvare i parametri 4. Eseguire un refresh della pagina 5. Caricare i parametri 6. Provare a genare il labirinto | | |
| **Risultati attesi** | 1. I parametri devono essere effettivamente salvati 2. I parametri devono essere effettivamente caricati 3. Il due labirinti devono essere identici | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-007  REQ-004 | **Nome:** | Procedura di salvataggio labirinto |
| **Descrizione:** | Una volta generato un labirinto, al click dell’utente, il programma salva un svg con il labirinto | | |
| **Prerequisiti:** | Un labirinto generato | | |
| **Procedura:** | 1. Provare a genare il labirinto 2. Scegliere la dimensione della foto 3. Provare a salvare il labirinto | | |
| **Risultati attesi** | La foto salvata è delle dimensioni specificate dall’utente. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-008  REQ-005 | **Nome:** | Disegno sul labirinto |
| **Descrizione:** | Una volta generato un labirinto, l’utente ha la possibilità di tracciare un percorso sopra la labirinto appena generato | | |
| **Prerequisiti:** | Un labirinto generato, mouseListener | | |
| **Procedura:** | 1. Provare a genare il labirinto 2. Provare a tracciare un percorso tenendo il mouse premuto | | |
| **Risultati attesi** | Il percorso viene tracciato. | | |

## Risultati test

| **Test Case** | **Stato** |
| --- | --- |
| TC-001 | Riuscito |
| TC-002 | Riuscito |
| TC-003 | Quasi riuscito |
| TC-004 | Fallito |
| TC-005 | Fallito |
| TC-006 | Quasi riuscito |
| TC-007 | Fallito |
| TC-008 | Fallito |

## Mancanze/limitazioni conosciute

Per quanto riguarda i test case 003 e 006, quelli parzialmente riusciti, manca poco alla possibile soluzione.

Il test case **003** riguarda il seed, a causa di una modifica che ho nel codice il seed non è una combinazione di 6 cifre, viene infatti trattato come numero a 6 cifre. Di conseguenza non è possibile inserire numeri come “023589”, ma solamente numeri tra 100000 e 999999.

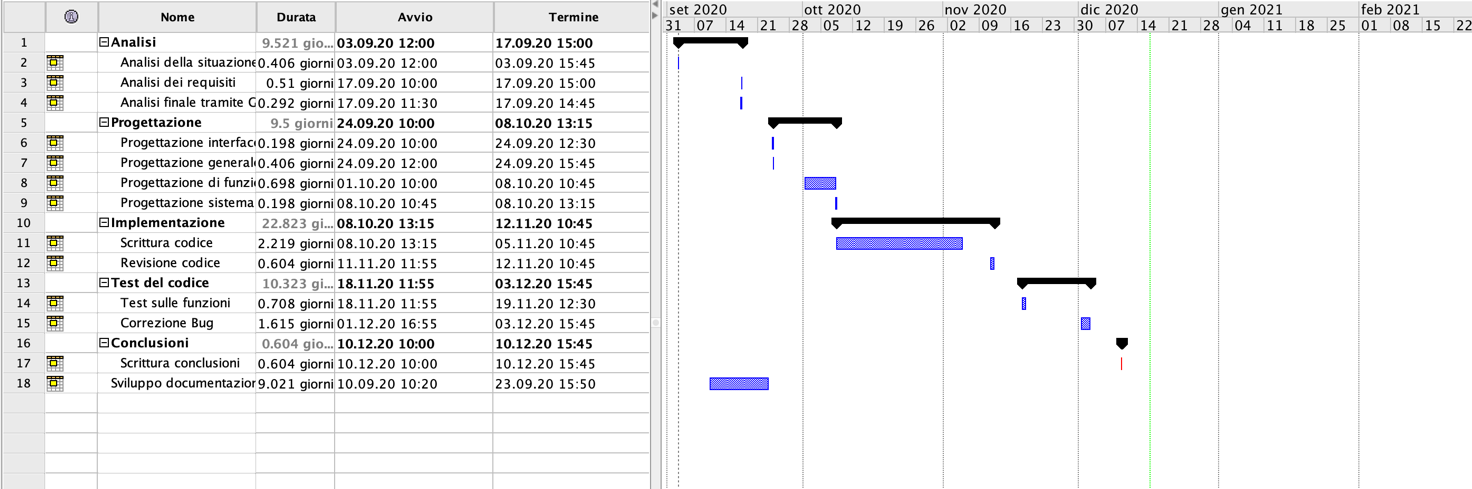
Il test case **006** invece è fallito a metà, se il salvataggio dei parametri avviene ed è corretto, verificando il Json i dati sono infatti corretti, l’upload non è invece funzionante. Per risolvere dovrei effettuare altre ricerche sull’upload dei file e sulla gestione dei Json.

Per i test case **004**, **005**, **007**, **008,** il discorso cambia: i test sono infatti falliti per assenza di codice.

L’unica differenza è che il test case **004** non fallisce del tutto.

Soprattutto la forma quadrata permette di generare una forma vagamente rassomigliante ad un labirinto, sempre ugnale se i parametri vengono inseriti in maniera analoga alla precedente.

# Consuntivo



Come detto non sono riuscito a completare il progetto, già quando dovevo aver già iniziato a scrivere il codice, quello della generazione del labirinto, ero indietro…

Pensavo di riuscire a recuperare il ritardo accumulato durante le mie assenze, dovute a operazione chirurgica e malattia, ma non è stato così.

Per il prossimo progetto, quello a gruppi, cercherò sicuramente di gestire meglio il mio tempo.

# Conclusioni

Ponendo che io abbia finto il progetto, la mia soluzione è particolare. Anche girando online non ho trovato molte soluzioni simili alla mia. Ho infatti adottato una variante del metodo a divisione ricorsiva, ovvero mi sono basato sull’idea di dividere in parti sempre minori lo spazio del labirinto. Oltre a essere relativamente rara la mia soluzione non è particolarmente dinamica; mi spiego molto. Sono riuscito solamente a disegnare delle forme in maniera procedurale, scrivendo quindi il codice per ogni singolo segmento. Volevo provare a “metodizzare” il tutto, ma per un motivo o l’altro non ci sono riuscito.

Posso quindi dire che i risultati che ho ottenuto sono piuttosto scarni, non solo perché l’algoritmo è poco efficiente e eccessivamente complesso per quello che fa, ma anche perché non sono riuscito a generare un labirinto vero e proprio.

## Sviluppi futuri

In futuro penso che riprenderò questo progetto; il primo passo sarà riuscire a metodizzare il mio codice, e soprattutto renderlo riutilizzabile. Infatti essendomi dedicato alla realizzazione di due forme differenti, cerchio e quadrato, ho dovuto scrivere codici piuttosto diversi, anche se comunque simili in certe dinamiche.

Perciò voglio rendere il codice il più riutilizzabile possibile, ma soprattutto rendere possibile la generazione tramite l’esecuzione di una porzione di codice sempre identica, inserita in un ciclo.

## Considerazioni personali

Devo dire che non sono particolarmente soddisfatto del mio progetto.

Ho imparato sì molti nuovi aspetti della programmazione in Java Script, primo fra tutti la gestione di una canvas oppure la grafica. Anche la gestione e la creazione dei file JSON sono stati una parte importante del mio lavoro…

Tuttavia avrei voluto essere riuscito un labirinto per almeno una delle forme richieste.

Questo non è stato però possibile per fattori come la mia gestione del tempo e alcune assenze che ho fatto.

Sono comunque soddisfatto di essermi confrontato con questo progetto, ho imparato molto anche sulla gestione del tempo e sulla necessità di focalizzarsi sulle cose giuste, cosa che io non ho fatto in quanto ho prediletto uno sviluppo a cascata piuttosto che uno più “modulare”, come sarebbe stato meglio.

In futuro spero quindi di riuscire a sviluppare un progetto in maniera più efficiente rispetto a questa volta.

# Bibliografia

## Sitografia

https://www.mockflow.com, *Mockflow.com*, 24-09-2020 | 22.10.2020.

http://www.mazegenerator.net, *Mazegenerator.com*, 24-09-2020 | 22.10.2020

https://en.wikipedia.org/wiki/Maze\_generation\_algorithm, *Wikipedia.com*, 24-09-2020

https://it.wikipedia.org/wiki/Algoritmi\_per\_la\_generazione\_di\_un\_labirinto, *Wikipedia.com*, 24-09-2020

https://www.W3schools.com/js/js\_events.asp*, W3schools.com,* 01-10-2020.

https://support.microsoft.com/en-us/office/use-case-shape-3aa9183f-a32d-4b71-8d54-ed18e67f6f53,

*Microsoft.com*, 7-10-2020

https://www.W3schools.com/tags/tag\_input.asp*, W3schools.com,* 15-10-2020.

https://www.W3schools.com/html/html\_form\_input\_types.asp*, W3schools.com*, 15-10-2020

https://www.W3schools.com/js/js\_input\_examples.asp*, W3schools.com,* 15-10-2020

https://www.W3schools.com/jsref/dom\_obj\_text.asp*, W3schools.com,* 15-10-2020

https://www.W3schools.com/jsref/event\_onclick.asp*, W3schools.com,* 15-10-2020

https://www.bitdegree.org/learn/javascript-input, *bitdegree.org*, 15-10-2020

https://www.valentinog.com/blog/link-download, *valentinog.com*, 22-10-2020

https://jsfiddle.net/cowboy/hHZa9, *jsfiddle.com*, 22-10-2020

https://www.W3schools.com/js/js\_json.asp, *W3schools.com*, 12-11-2020 | 19.11.2020

https://stackoverflow.com/questions/36127648/uploading-a-json-file-and-using-it,

*Stackoverflow.com*, 12-11-2020 | 19.11.2020

https://www.hostingtalk.it/array-associativi-in-javascript\_-c000000DH/*, Hostingtalk.it,* 19-11-2020

https://www.w3schools.com/tags/ref\_canvas.asp, *W3schools.com*, 26.11.2020 | 10.12.202

<https://www.w3schools.com/graphics/canvas_drawing.asp>, *W3schools.com*, 26.11.2020 | 10.12.202

https://www.w3schools.com/html/html5\_canvas.asp, *W3schools.com*, 26.11.2020 | 10.12.202

<https://www.w3schools.com/tags/canvas_lineto.asp>, *W3schools.com*, 26.11.2020 | 10.12.202

https://www.w3schools.com/tags/canvas\_rect.asp, *W3schools.com*, 26.11.2020 | 10.12.202

https://www.w3schools.com/tags/canvas\_arc.asp, *W3schools.com*, 26.11.2020 | 10.12.202

# Allegati

Allegato a questo lavoro si può trovare:

* Diari di lavoro
* Codici sorgente
* Qdc