Ball Maze

Titolo del progetto: Ball Maze
Alunno: Lorenzo Spadea
Classe: I3AC
Anno scolastico: 2022/23
Docente responsabile: Guido Mondalbetti

Contents

1 Introduzione

1.1 Informazioni sul progetto

Questo progetto è richiesto dalla Scuola Arti e Mestieri di Trevano , Lugano. Il progetto serve ad insegnare interattivamente come si gestisce, documenta e implementa un progetto da inizio a fine. Più dettagli sul progetto:

- Sezione: Informatica
- Anno: Terzo
- Modulo: 306
- Responsabile: Guido Montalbetti
- Alunno: Lorenzo Spadea
- Data inizio: 09.09.2022
- Data consegna: 23.12.2022

Il progetto deve includere le seguenti specifiche:

- Documentazione svolta durante l'arco di tutto il progetto.
- Rapporto giornaliero di tutti i cambiamenti apportati al progetto.
- Codice sorgente dell'applicazione reperibile sempre.

La versione completa del progetto può essere trovata al seguente indirizzo:
https://github.com/SpadeaLorenzo/Ball-Maze

1.2 Abstract

Nowadays having fun while programming or studying is quite hard, especially if i require mathematics too. In this document you will find a step to step build of a yet simple yet complex game: The Ball Maze game. While the actual game is pretty basic and simple, the logic behind it is a complex math algorithm used to generate and solve mazes. Walking through this document you will understand more about mathematics function and the implementation of such algorithm. This document provides the knowledge of maze creating and solving algorithm, logic of basic movements and gameplay. This can be a fun way to learn coding while having fun.

1.3 Scopo

Lo scopo di questo Progetto è quello di richreare il gioco/passatempo "Ball Maze" in una versione digitale. Il gioco sarà disponibile come eseguibile da poter utilizzare su qualunque computer munito di Java.

2 Analisi

2.1 Analisi del dominio

- Processing 4.0 : Per la creazione e il rendering grafico (IDE).
- Github : Per la repository del progetto.
- Overleaf latex : Documenti del progetto.
- Microsoft Project : Realizzazione progettazione.

2.2 Analisi e specifica dei requisiti

2.2.1 Req-00

Req - 00	
Nome	Generazione labirinto grafico
Priorità	1
Versione	1.0
Note	-
Descrizione	Il programma genra labirinti randomici

2.2.2 Req-01

Req - 01	
Nome	Risoluzione Labirinto
Priorità	1
Versione	1.0
Note	-
Descrizione	Il programma risolve i labirinti trovando il percorso migliore
Subrequirements	
Req 01.1	Calcola il tempo migliore per la risoluzione del labirinto
Req 01.2	Calcola un punteggio da attribuire in base a difficoltà e tempo di risoluzione
Req 01.3	Calcola il tempo massimo entro il quale risolvere il labirinto

2.2.3 Req-02

Req - 02	
Nome	Generazione punto di inizio e fine
Priorità	1
Versione	1.0
Note	-
Descrizione	Il programma genera due punti casuali nel labirinto: inizio e traguardo
Subrequirements	
Req 02.1	Ci deve essere una logica che definisca la vittoria del labirinto
Req 02.2	In caso di vittoria assegnare il punteggio adeguato

2.2.4 Req-03

Req - 03	
Nome	Home page
Priorità	1
Versione	1.0
Note	-
Descrizione	Schermata iniziale del programma
Subrequirements	
Req 03.1	Si devono vedere i dati utente(punteggio)
Req 03.2	Si deve poter scegliere il livello di difficoltà

2.2.5 Req-04

Req - 04	
Nome	Salvataggio dati
Priorità	1
Versione	1.0
Note	-
Descrizione	I dati dei punteggi vengono salvati ed aggiornati

2.2.6 Req-05

Req - 05	
Nome	Ostacoli
Priorità	2
Versione	1.0
Note	-
Descrizione	Il labirinto viene riempito casualmente di trappole

2.2.7 Req-06

Req - 06	
Nome	Vite
Priorità	2
Versione	1.0
Note	-
Descrizione	L'utente avrà a disposizione delle vite per completare i singoli livelli

2.2.8 Req-07

Req - 07	
Nome	3D
Priorità	3
Versione	1.0
Note	-
Descrizione	Il labirinto verràimplementato in un piano 3D
Subrequirements	
Req 07.1	Verrano implementate funzioni di luce e movimento del piano per giocare

2.3 Use case

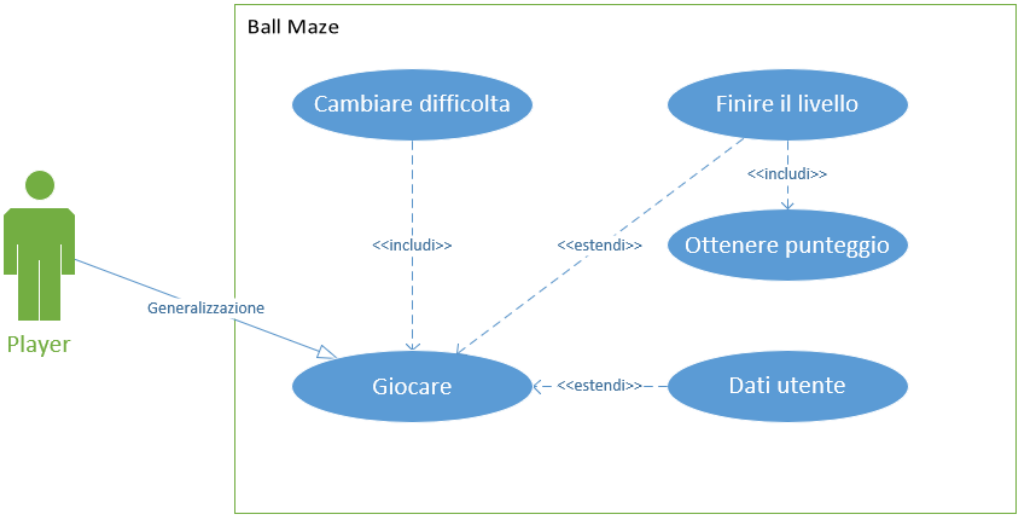


Figure 1: Use Case

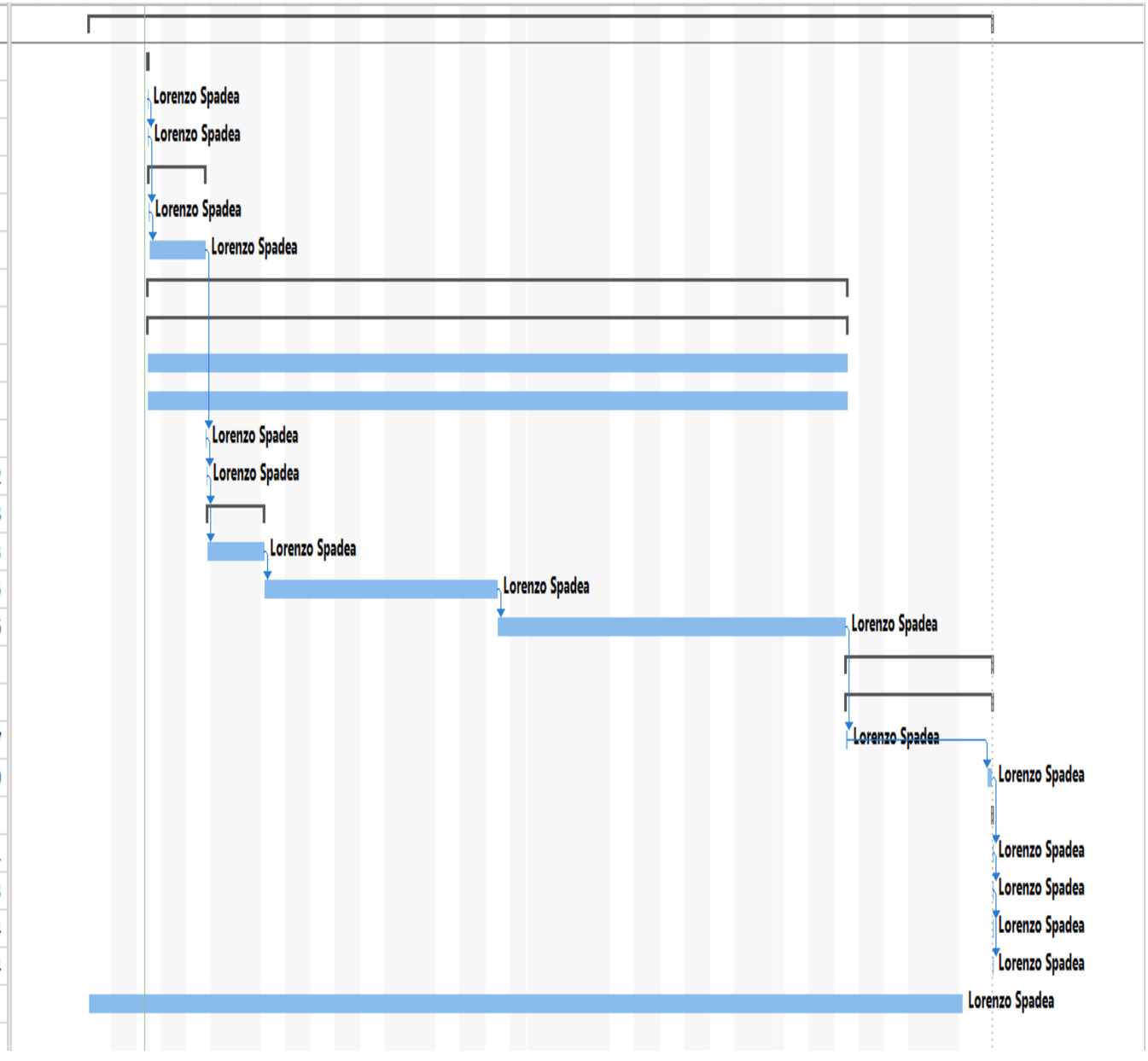
Il diagramma serve a definire i casi d'uso del programma. Da un'idea del come funiona l'applicativo in base ai ruoli che l'utente può avere al suo interno e le azione che ne derivano.

2.3.1 Player

Il player può lanciare il gioco ed alla schermata d'avvio decidere il livello di difficoltà dei labirinti generati. Una volta che il gioco viene lanciato, vengono caricati i dati dell'utente(anche se fossero a zro vengono comunque caricati).L'utente una volta fatto partire il gioco può ottenere punti che verrano salvati nel profilo utente ed aggiornati sulla schermata di partenza ogni qualvolta ci ritorna.

2.4 Pianificazione

1	Ball Maze	75 h	ven 09.09.22	lun 26.12.22	
1.1	Fase iniziale	2 h	ven 16.09.22	ven 16.09.22	
1.1.1	Stesa requisiti	1 h	ven 16.09.22	ven 16.09.22	
1.1.2	Creazione github Repository	1 h	ven 16.09.22	ven 16.09.22	3
1.2	Progettazione	2 h	ven 16.09.22	ven 23.09.22	
1.2.1	Gantt	1 h	ven 16.09.22	ven 16.09.22	4
1.2.2	Use case	1 h	ven 16.09.22	ven 23.09.22	6
1.3	Implementazione	52 h	ven 16.09.22	ven 09.12.22	
1.3.1	Training	52 h	ven 16.09.22	ven 09.12.22	
1.3.1.1	Processing	52 h	ven 16.09.22	ven 09.12.22	
1.3.1.2	Java	52 h	ven 16.09.22	ven 09.12.22	
1.3.2	Installazione Processing	2 h	ven 23.09.22	ven 23.09.22	7
1.3.3	Configurazione Wokspace	2 h	ven 23.09.22	ven 23.09.22	12
1.3.4	Crazione ambientazione 3D	1 h	ven 23.09.22	ven 30.09.22	13
1.3.4.1	Installazione pacchetti grafici	1 h	ven 23.09.22	ven 30.09.22	13
1.3.5	Sviluppo algorimo labirinti	20 h	ven 30.09.22	ven 28.10.22	15
1.3.6	Implementazione labirinti in grafica	20 h	ven 28.10.22	ven 09.12.22	16
1.4	applicazione	17.83 h	ven 09.12.22	lun 26.12.22	
1.4.1	creazione GUI	17.83 h	ven 09.12.22	lun 26.12.22	
1.4.1.1	Schermata iniziale	4 h	ven 09.12.22	ven 09.12.22	17
1.4.1.2	Grafica di gioco	3 h	lun 26.12.22	lun 26.12.22	20
1.5	Testing	3 h	lun 26.12.22	lun 26.12.22	
1.5.1	Apertura applicazione	1 h	lun 26.12.22	lun 26.12.22	21
1.5.2	Generazione labirinti	1 h	lun 26.12.22	lun 26.12.22	23
1.5.3	Superamento livelli	1 h	lun 26.12.22	lun 26.12.22	24
1.5.4	incremento difficoltà livelli	1 h	lun 26.12.22	lun 26.12.22	24
1.6	Diari + documentazione	20 h	ven 09.09.22	ven 23.12.22	



2.4.1 Gantt

Per questo progetto ho deciso di adattare un modello di pianificazione Waterfall come si vede nell'immagine precedente. Per la realizzazine del diagramma delle attività ho usato un prodotto Microsoft su licenza chiamato Microsoft Project che permette di realizzare diagrammi di Gantt. Questo diagramma mi permetterà di avere un filo logico di attivtà che dovrò svoglere per arrivare al risultato del progetto completo.
Durante la fase di implementazione sarò in grado di vedere se sono stato in grado di fare delle buone predizioni e se le tempistiche vengono rispettate.
Una volta terminato il progetto tutte le differenza di tempistiche verrano conforntate in un Gantt cosnuntivo che verrà successivamente allegato.

2.4.2 Fase iniziale

Questa sezione è relativamente breve e serve a dare un inizio al progetto. In questa fase ho definito i requisiti del progetto ed ho creato la cartella dove verranno salvati i file. Il tempo stimato è di due ore.

2.4.3 Progettazione

In questa fase viene implementato il Gantt nel quale verranno definite le attività e lo use case che andrà a definire il comportamento degli attori dell'applicazione. Tempo stimato tre ore.

2.4.4 Training

Questa Attività durerà per tutta la durata del progetto e serivrà per acquisire le informazioni necessarie allo svolgimento del progetto.

2.4.5 Creazione Ambientazione 3D

In questa fase creerò l'ambientazione 3D dove verrà generato il labirinto e dunque conseguentemente i due algoritmi principale del gioco. Per la durata di questa attività stimo una durata di circa 20 ore.

2.4.6 creazione GUI

In questa fase di progetto andrò invece a creare le pagine di controllo del gioco come ad esempio la home page.

2.4.7 Testing

L'ultima fase è quella di testing dove andrò a testare il gioco nel suo complesso.

2.4.8 Documentazione e diari

In questa fase che dura lungo tutto il progetto scriverò la documentazione e i diari giornaglieri.

2.5 Analisi dei mezzi

Il programma è interamete basato su una libreria grafica open-source basata sul linguaggio di programmazione Java. La libreria e il suo workspace si chiamano Processing v 4.0

2.5.1 Software

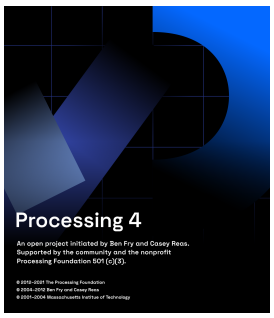


Figure 2: Processing

Il software permette la gestione semplificata della grafica con GUI basata sul linguaggio Java.

- Prodotto: Processing
- Versione: 4.0.1
- Licenza Open source
- Download: https://processing.org/download

Il software è stato installato su una macchina con sistema operativo Windows.

2.5.2 Hardware

Il software ed il progetto sono stati realizzati su un Computer fornito dalla scuola con le seguenti caratteristiche:

- Sistema Operativo : Windows 10 Enterprise
- Versione: 21H2
- Processore: Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz 3.00 GHz
- RAM: 32.0 GB
- GPU : NVIDIA GeForce RTX 2060
- Base: 64 bit , processore su x64

3 Progettazione

Questo progetto non è stato pensato per avere un sistema di archiviazione classico dunque non è stato implementato nessun sistema che richieda software di terza parti per la realizzazione.
Il gioco contiene dei dati che possono essere aggiornati ad ogni fine livello. Questi dati sono il punteggio della risoluzione del labirinto ed il nome del giocatore che lo ha completato.
Data la semplicità dei dati da manipolare ho optato per un semplice file di testo nel quale viene inserita la stringa contenente i due dati citati sopra.
I dati vengono successivamente visualizzati nel gioco come "classifica" dei migliori punteggi associati al nome inserito.

3.1 Design dell'architettura del sistema

3.2 Design dei dati e database

3.3 Design delle interfacce

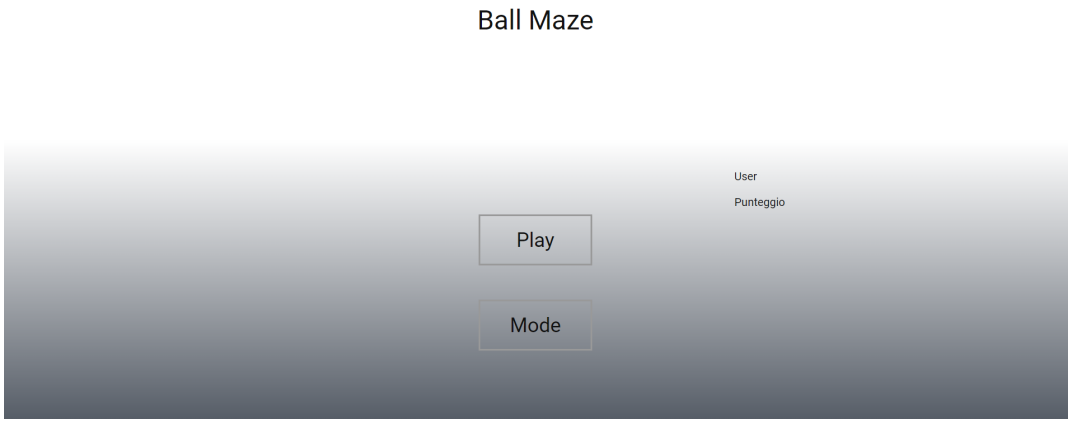


Figure 3: Home Page

La home page è molto minimalista e semplice. Il giocatore vedrà il proprio user name ed il punteggio accumulato fino a quel momento. Il pulsante "Play" farà partire il gioco mentre il pulsante "Mode" serve a definire la difficoltà del livello.

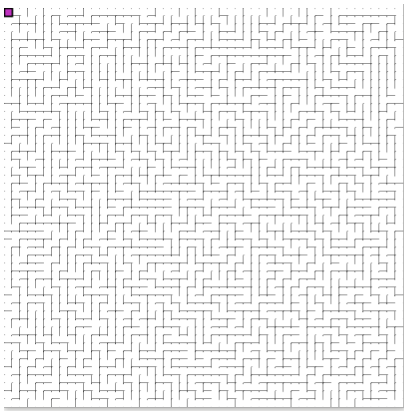


Figure 4: Labirinto

Una volta che il giocatore avrà premuto il puslante play il gioco partirà e verrà generato un labirinto casuale(vedi immagine) in base alla difficoltà impostata.
Terminato il livello (sia in caso di vittoria che di sconfitta) il giocatore tornerà alla schermata principale dove potrà vedere il proprio punteggio aumentato (o non in caso di sconfitta) e riprendere a giocare su una nuova mappa

Figure 5: Livello finito

Una volta che l'utente avrà terminato il livello(sia in caso di sconfitta che di successo) si aprirà una seconda schermata nella quale il giocatore vedrà il punteggio ottenuto(rappresentato graficamente e qualitativamente dalle classiche stelline).
A questo punto il giocatore potrà decidere se ripetere il livello ottenendo un punteggio migliore o se tornare alla schermata principale(vedi immagine: Figura 3: Home Page) e far partire un nuovo livello.
Tornando sulla schermata di Home il punteggio verrà aggiornato e si vedrà la differenza rispetto a quello precedente.

3.4 Design procedurale

Figure 6: Diagramma classi

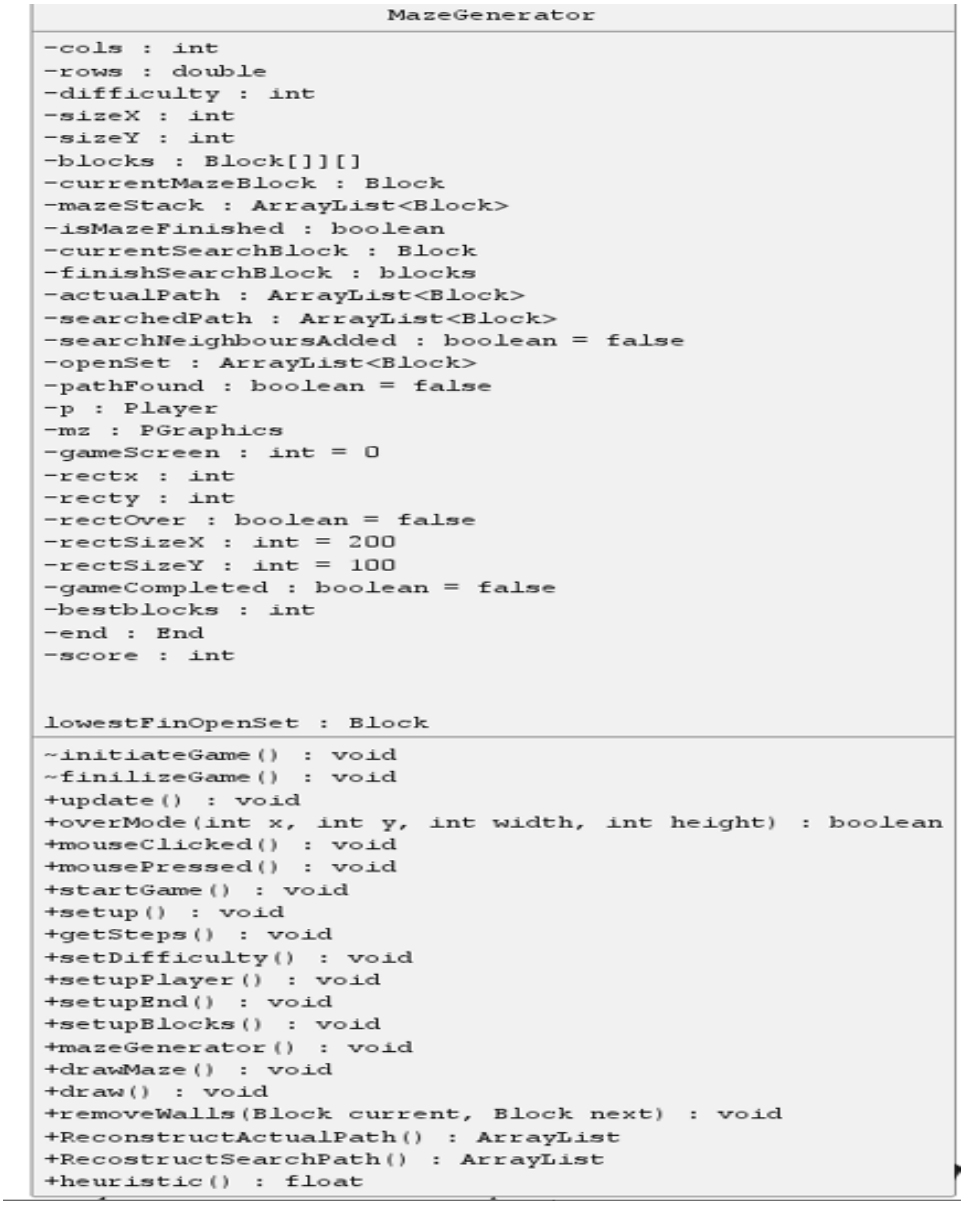


Figure 7: MazeGenerator class

Questa è la classe principale verso la quale fanno riferimento anche le altre. Si occupa di tutto quello che è il disegno grafico e logico del labirinto. Gestisce le collisioni, i movimenti del Player, la ricerca del percorso, l'assegnazione dei punti e delle schermate del gioco.

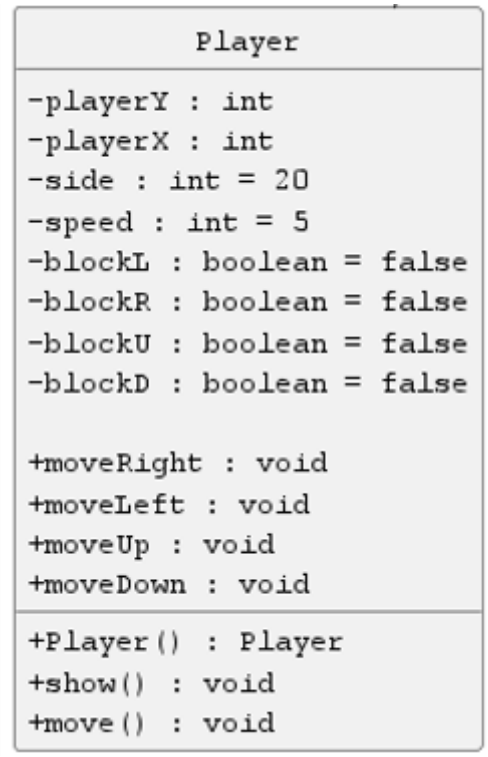


Figure 8: Player class

La classe player è molto minimalista: si occupa di generare un Player alla posizione (0,0) delle coordinate del labirinto e di poi mostrarlo successivamente quando la mappa è generata.

Figure 9: Block class

La classe Block è l'unità sulla quale si basa tutta la struttura del labirinto. La classe è definita da una posizione nell' area grafica ed da un riferimento rispetto alla riga e alla colonna della griglia. Ogni blocco è composto da delle pareti che genereranno il labirinto. I blocchi inoltre si "conoscono" tra di loro grazie a delle funzioni che associano ad ogni blocco dei blocchi vicini.

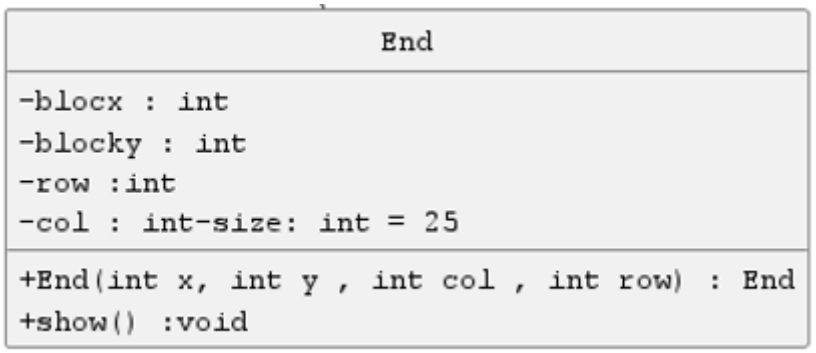


Figure 10: End class

La classe End definisce semplicemente il traguardo al quale il player deve arrivare. Anche a fine è definita da delle coordinate e da un riferimento alle righe e alle colonne del labirinto.

4 Implementazione

4.1 Blocchi

I blocchi sono il componente base che sta dietro alla creazione dei labirinti. Per la realizzazione di questo progetto ho optato a creare una classe di tipo Block. Questo mi ha permesso di automatizzare le opzioni di creazione del labirinto in fase di implementazione. Ogni blocco nella griglia è univocamente identificato dalla propria posizione(x,y) all'interno dell'area grafica e dal numero della colonna e della riga entro il quale si trova. Figurativamente possiamo vedere come il blocco evidenziato abbia delle coordinate x ed y che corrispondono all angolo in altro a sinistra e un riferimento alla colonna e alla riga nella quale si trova(colonna 2 riga 1).

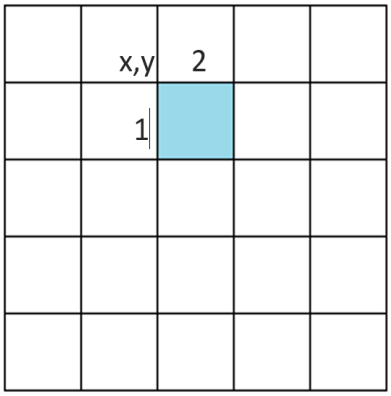


Figure 11: Blocco

Questo andrà a semplificare molto la gestione grafica della generazione del labirinto siccome si potrà modificare ed interagire con l'area grafica di gioco passando per ogni blocco alla volta. Ci sono altre soluzioni al problema della generazione di labirinti. Nel caso che ho deciso di implementare io, quella di sfruttare dei blocchi come base di partenza, è l'opzione migliore e più funzionale.
I blocchi inoltre sono identificati da quattro pareti che vengono salvate come valori booleani. Queste pareti verrano utili più avanti nella realizzazione delle collisioni tra le pareti dei blocchi e il player nella mappa.

4.2 Maze generator

L'implementazione di questo progetto si differenzia in due categorie principali: La creazione e la risoluzione tramite algoritmi dei labirinti e il gioco. Per la creazione dei labirinti e la loro conseguente risoluzione sono stati usati degli algoritmi di costruzione e ricerca. Il gioco invece si appoggia sulla base dell'implementazione algoritmica dei labirinti alla quale viene poi aggiunta la logica di gioco. Il primo step della creazione del Ball maze game è la creazione dei labirinti. Il labirinto possiamo immaginarlo inizialmente come una griglia di blocchi.

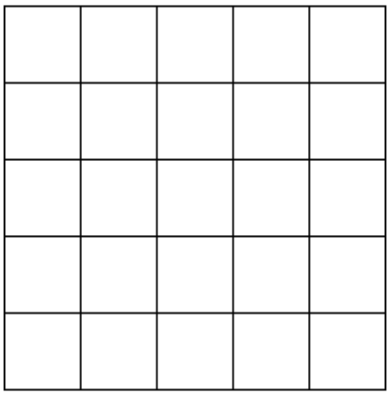


Figure 12: griglia labirinto

La generazione del labirinto parte dal primo blocco in alto a sinistra ed ogni blocco della griglia è formato da 4 pareti. Il concetto di base per la costruzione del labirinto è basato sull'algoritmo depth-first-search e funziona cosi:

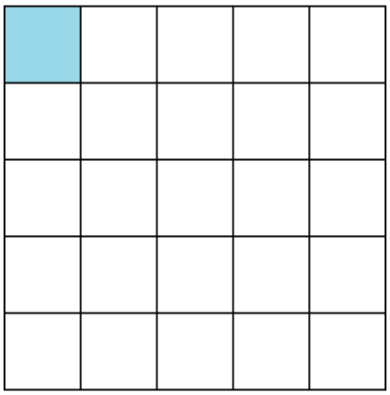


Figure 13: Primo passaggio

Il primo passaggio consiste nel dare la prima cella come paramentro all algoritmo che marcherà la stessa cella come "visitata".

```
  //Returns the position to the top left corner.
  currentMazeBlock = blocks[0][0];
  //The top left corner block is set to "visited".
  currentMazeBlock.visitedByMaze = true;
}
```

Figure 14: Inizializzazione labirinto

Come possiamo vedere abbiamo impostato la prima cella della griglia (riga e colonna 0) come la cella corrente sul quale l'algoritmo lavorerà. La stessa cella viene anche settata come visitata per far si che l'algoritmo di ricerca delle celle non torni su di essa.
Il passo successivo consiste nella ricerca delle celle vicine(sempre partendo dal presupposto che ci siano celle non ancora visitate).

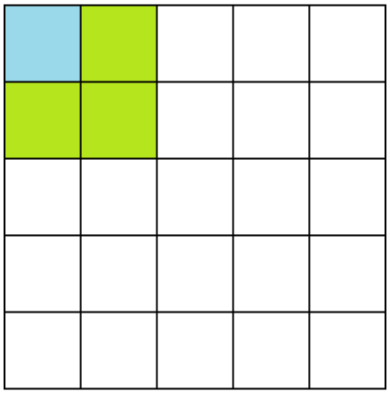


Figure 15: cercando i vicini

In questo momento l'algoritmo sta visualizza i propri blocchi vicini che sono stati identificati e randomicamente decide in quale muoversi. Una volta decisa

la cella nella quale spostarsi il blocco di partenza andrà a rimuovere la parete nella stessa direzione della quale si trova il vicino verso il quale sta andando.

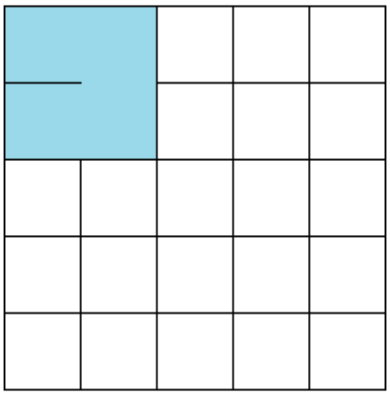


Figure 16: Primo ipotetico passaggio

La generazione del labirinto prosegue poi recursivamente fin quando non ci sono più celle visitate all'interno della griglia di blocchi.

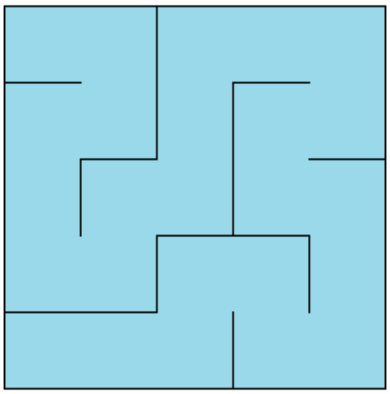


Figure 17: Labirinto finito

In questa figura vediamo lo stato finale del labirinto una volta completato. Non ci sono più blocchi da visitare e il percorso randomico è completo.

I passaggi in meta codice sono descritti in questo modo:

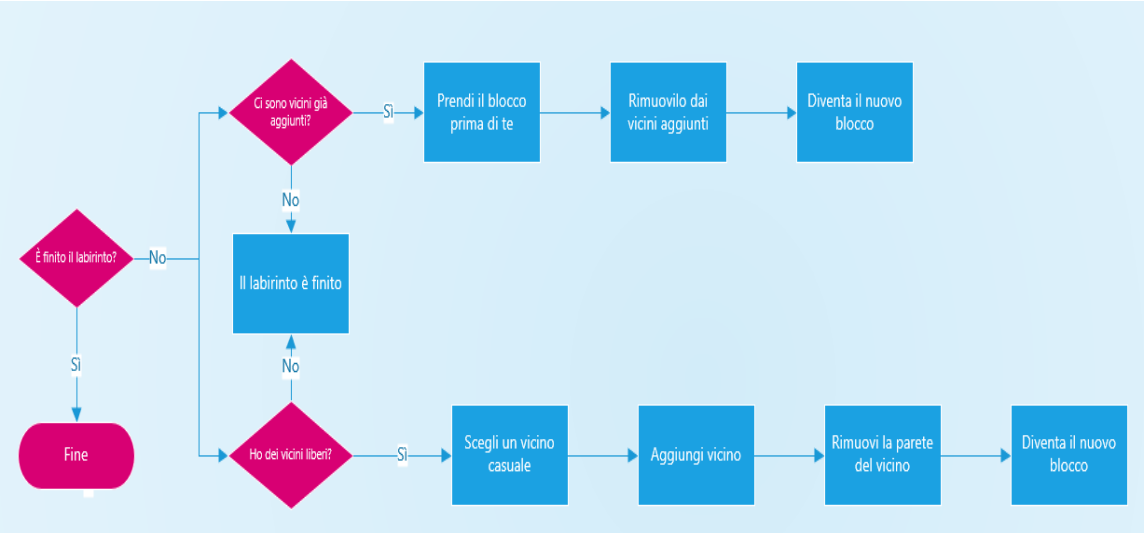


Figure 18: Flow diagram

La generazione del Labirinto viene poi salvata come immagine grafica dal programma che poi la andrà a reinderizzare quando iniizia il gioco.

4.3 Risoluzione

Per la risoluzione del labirinto ho implementato un algoritmo di ricerca che si chiama A* search (A-star search). È un algoritmo di ricerca di percorsi che sfrutta la sua logica per trovare una soluzione migliore dopo averne sperimentate diverse contemporaneamente. È molto noto nel campo dell'informatica siccome offre una soluzione precisa con un costo di computazione relativamente basso. Non solo offre la possibilità di trovare un percorso per arrivare alla fine ma restituisce la migliore soluzione(la più breve) per raggiungerla.

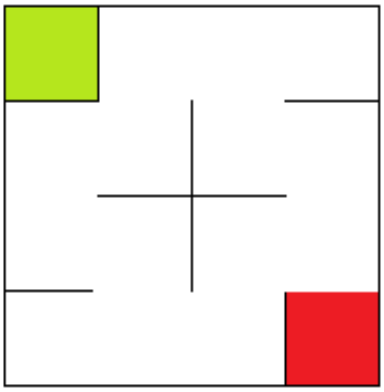


Figure 19: Start and end point

Supponiamo dunque di trovarci in una situazione dove il labirinto è stato generato. Nel labiritno verranno definiti un punto di inizio ed un punto di fine(randomico). L'algoritmo di ricerca si basa su un principio molto semplice:
attibuire un "costo" (corrispondente al "quanta strada devo fare") per ogni blocco della griglia.

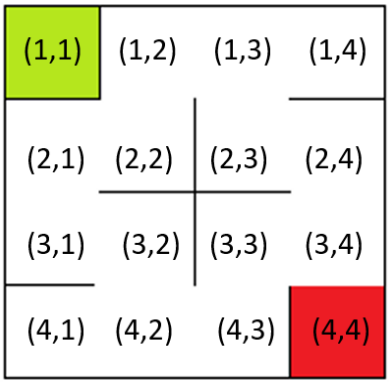


Figure 20: Costi delle celle

Qui possiamo vedere gli ipotetici costi assegnati ad ogni cella dall'algoritmo.

Ogni cella ha un costo specifico che chiameremo "n" ed è definito in due valori:

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

f(n) è definito come il costo totale per il raggiungimento della cella.
g(n) è il costo effettivo per raggiungere la cella n dalla cella iniziale.
h(n) è il costo euristico per raggiungere l'obbiettivo partendo dalla cella n.

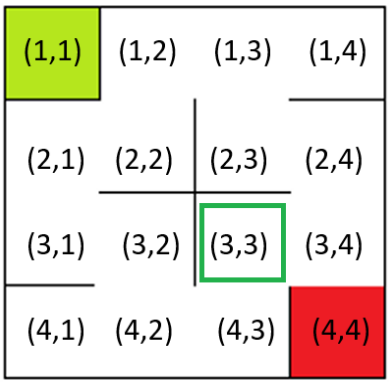


Figure 21: Esempio A*

Il costo g(n) della cella presa in considerazione nell'immagine è di 2 siccome dalla cella di partenza la si può ragiungere con due passaggi. h(n) è dunque il costo stimato per raggiungere l'obbiettivo (1,1) dalla cella (3,3). Non conoscendone il costo è bisogna ottenere una stima ed per farlo useremo la distanza euclidea.

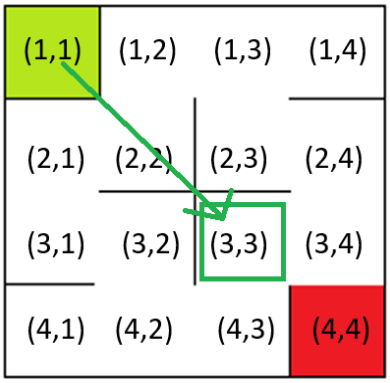


Figure 22: Distanza euclidea

È la distanza lineare tra la cella (3,3) e l'obbiettivo in questo caso.

La funzione euristica è il parametro pricipale dell'algoritmo A* e per la realizzazione ho optato per usare la distanza di Manhattan come funzione euristica.

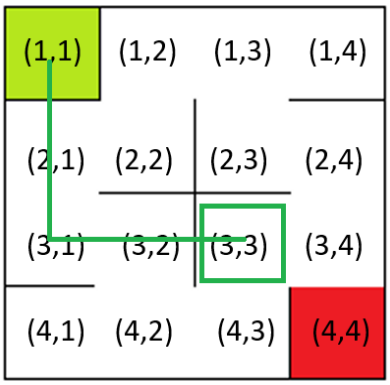


Figure 23: Distanza Manhattan

Questa distanza a differenza della prima è composta dalla distanza verticale e dalla distanza orizzontale tra la cella di partenza e l'obbiettivo.
Possiamo dunque dire che la distanza tra la cella(3,3) e l'obbiettivo è di 4. Da questo ne ricaviamo il costo della cella (3,3):

f(n) = g(n) + h(n) = 2 + 2 = 4.

Partendo da questo esempio possiamo dunque spiegare come l'algoritmo A* arriverà a trovare la soluzione migliore per raggiungere l'obbiettivo.

Partendo dalla cella (4,4) possiamo dire che il costo g(n) è pari a 0 siccome per raggiungere la prima cella partendo dalla prima cella non dobbiamo muoverci. h(n) invece è pari a 3 + 3 = 6. Per le altre celle non abbiamo ancora a dipossizione sufficienti elementi e dunque assegneremo un valore arbitrario g(n) = x — h(n) = y ad ogni cella.

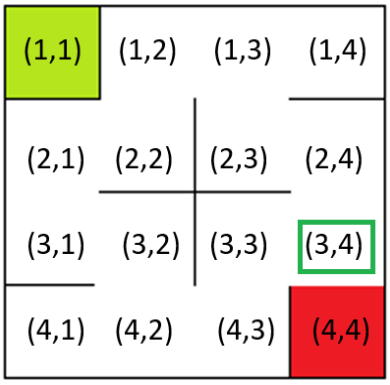


Figure 24: Secondo passaggio

Il costo della cella più vicina a noi e di 5 e per raggiungerla bisogna muoversi di un solo blocco. Ora ci troviamo dunque nella cella di posizione (3,4).

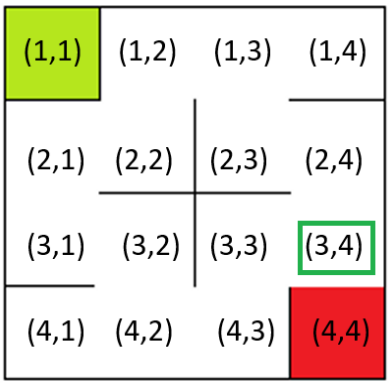


Figure 25: Secondo passaggio

Come possiamo vedere nella figura ora siamo difronte a tre possibilita:
Avanzare alla cella (2,4)
Avanzare alla cella (3,3)
Tornare indietro alla (4,4) il cui costo ora è 8 ed inoltre è già stata visitata.

Siamo di fronte ad un problema però in questo momento: il costo delle due celle rimanenti è lo stesso dunque quale strada deciderà di prendere l'algoritmo?
Prenderà entrame le strade suddividendosi in due percorsi diversi. Da quel momento in poi l'algoritmo andrà avanti fino a raggiungere l'obbiettivo sfruttando il concetto dei costi delle celle.

Entrambe possono raggiungere l'obbiettivo ed entrambe lo raggiungeranno sfruttando il costo minore possibile di ogni cella. In ogni caso la prima a raggiungere l'obbiettivo sarà quella con il minor numero di passaggi e con un costo totale delle celle minore. Una volta raggiunta la meta L'algoritmo smette di cercare siccome il percorso migliore è già stato trovato.

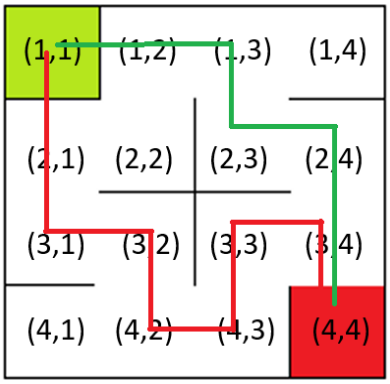


Figure 26: Possihbili soluzioni

4.4 Player

Dopo aver creato il labirinto ed aver trovato il miglior percorso per il suo svolgimento entra in gico l'unico componente dinamico del gioco: il Player.
In questo caso il player non è niente altro che un un blocco rivisionato:
è caratterizzato da due coordinate(x,y) che ne definiscono la posizione all'interno del labirinto ed una dimensione che ne definisce la grandezza.
Il player viene sempre generato nell'angolo in alto a sinistra della mappa.

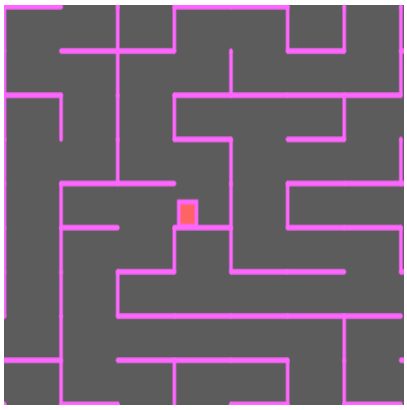


Figure 27: Player

La parte principale di questo oggetto è il motore collisionale che permette di muoversi all'intenro del labirinto senza però oltrepassare pareti. Il software che ho usato non ha implementato un collisore integrato dunque ho dovuto crearne uno.

Il primo passo è quello di capire la posizione del player all'interno della griglia di gioco(ovvero in che blocco si trova.). per farlo ho usato il seguente calcolo:
x(blocco) = floor(x(player)/width(blocco))
y(blocco) = floor(x(player)/height(blocco))

Grazie a questo posso determinare la posizione (x,y) del blocco dove il player si trova.
Il passo successivo determina la posizione del player all'interno del blocco in cui si trova. Il player è in grado di muoversi tra i blocchi finché il lato nella direzione nella quale si sta muovendo non è a contatto con una parete del labirinto. Per semplificare questo processo di riconoscimento delle pareti ho attribuito ai blocchi delle priprietà:
Ogni blocco è infatti composto da quattro pareti che sono interpretate come valori booleani. Dunque per far si che il player si blocchi quando entra a contatto con una parete mi basta una semplice condizione(controllare se c'è una parete) e se le coordinate del suo lato combaciano con essa.

5 Test

5.1 Protocollo di test

Test Case: Riferimento:	TC-001 REQ-00	Nome:	Generazione labirinti
Descrizione:	L'algoritmo deve generare dei labirintici grafici randomici		
Prerequisiti:	-		
Procedura:	1. Lanciare il programma e vedere il labirinto generato		
Risultati:	Genereazione di labirinti randomici		

Test Case: Riferimento:	TC-002 REQ-01	Nome:	Risoluzione labirinti
Descrizione:	L'algoritmo deve risolvere dei labirintici grafici randomici		
Prerequisiti:	1. Un labirinto già presente 2. Un punto di inizio ed un traguardo		
Procedura:	1. Lanciare il programma e vedere il labirinto risolto		
Risultati:	percorso di risoluzione labirinto		

Test Case: Riferimento:	TC-003 REQ-02	Nome:	Punto di inizio e fine
Descrizione:	L'algoritmo genera due punti randomici: inizio e traguardo.		
Prerequisiti:	Un labirinto già presente		
Procedura:	1. Lanciare l'applicazione 2. Far partire il gioco 3. Osservare i punti di inizio e fine		
Risultati:	Visualizzare inizio e fine		

Test Case:	TC-004	Nome:	Generazione punteggi
Riferimento:	REQ-02		
Descrizione:	L'algoritmo genera dei punteggi in base a diversi fattori: 1. La difficoltà del labirinto 2. Il tempo massimo di risoluzione del labirinto 3. La presenza di ostacoli 4. Il numero di vite a disposizione		
Prerequisiti:	1. Un labirinto già presente 2. Un inizio ed una fine 3. La risoluzione perfetta dell'algoritmo		
Procedura:	1. Richiamare la funzione draw() 2. Osservare il punteggio massimo		
Risultati:	visualizzare punteggio ottenuto a fine partita		

Test Case:	TC-005	Nome:	Home page
Riferimento:	REQ-03		
Descrizione:	L'applicazione una volta lanciata aprirà una schermata di Home		
Prerequisiti:	-		
Procedura:	1. Lanciare l'applicazione 2. Visualizzare la chermata home		
Risultati:	Schermata Home dalla quale giocare e settare impostazioni		

Test Case:	TC-006	Nome:	Impostazioni schermata Home
Riferimento:	REQ-03		
Descrizione:	L'utente sulla pagine Home può impostare la difficoltà del livello		
Prerequisiti:	La schermata Home		
Procedura:	1. Lanciare l'applicazione 2. Impostare la difficoltà		
Risultati:	Genreazione di labirinti di diverse difficoltà		

Test Case: TC-007	Nome:	Impostazioni utente e punteggi
Riferimento: REQ-03		
Descrizione:	L'utente sulla pagine Home può impostare il proprio nome e vedere il punteggio	
Prerequisiti:	La schermata Home	
Procedura:	1. Lanciare l'applicazione 2. Impostare il proprio nome utente 3. Visualizzare il proprio punteggio	
Risultati:	Poter impostare il nome utente e visualizzare il punteggio	

Test Case: TC-008	Nome:	Salvataggio punteggio
Riferimento: REQ-04		
Descrizione:	L'utente sulla pagine Home può visualizzare i punteggi dopo ogni partita	
Prerequisiti:	1. La schermata Home 2. Il completamento di almeno un labirinto	
Procedura:	1. Lanciare l'applicazione 2. Completare un livello 3. Visualizzare il proprio punteggio aumentato	
Risultati:	Vedere l'incremento dei punti accumulati	

Test Case: TC-009	Nome:	Ostacoli
Riferimento: REQ-05		
Descrizione:	All'interno del labirinto vengono generati ostacoli e trappole	
Prerequisiti:	1. La schermata Home 2. Il completamento di almeno un labirinto	
Procedura:	1. Lanciare l'applicazione 2. Completare un livello 3. Visualizzare il proprio punteggio aumentato	
Risultati:	Vedere l'incremento dei punti accumulati	

Test Case:	TC-010	Nome:	Ostacoli
Riferimento:	REQ-06		
Descrizione:	Il giocatore avrà delle vite per completare i labirinti		
Prerequisiti:	1. Un labirinto da completare		
Procedura:	1. Lanciare l'applicazione 2. Far partire un livello 3. Visualizzare le proprie vite nel gioco		
Risultati:	Vedere le vite all'interno del gioco		

Test Case:	TC-011	Nome:	Movimento
Riferimento:	REQ-05		
Descrizione:	Il giocatore Potrà muoversi nel labirinto		
Prerequisiti:	1. Un labirinto da completare		
Procedura:	1. Lanciare l'applicazione 2. Far partire un livello 3. Muoversi all'interno del gioco.		
Risultati:	Muoversi all'interno del gioco		

Test Case:	TC-012	Nome:	3D
Riferimento:	REQ-07		
Descrizione:	Generazione di labirinti in 3D		
Prerequisiti:	-		
Procedura:	1. Lanciare l'applicazione 2. Far partire un livello 3. Vedere il labirinto in 3D.		
Risultati:	vedere il labirinto in 3D		

5.2 Risultati test

5.2.1 TC-001

L'algoritmo deve generare dei labirintici grafici randomici — Passato

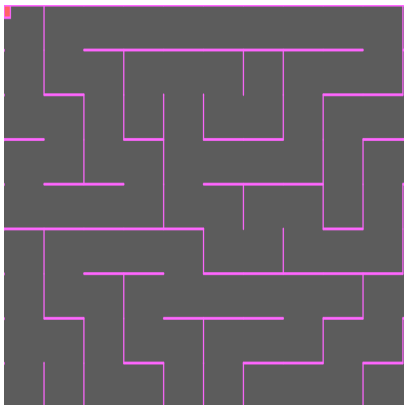


Figure 28: TC-001

5.2.2 TC-002

L'algoritmo deve risolvere dei labirintici grafici randomici — Passato
L'algoritmo di ricerca A* funziona

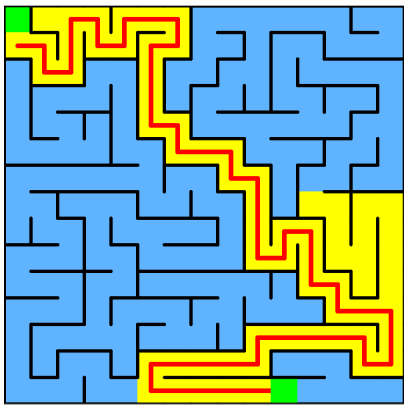


Figure 29: TC002

5.2.3 TC-003

L'algoritmo genera due punti randomici: inizio e traguardo —Passato

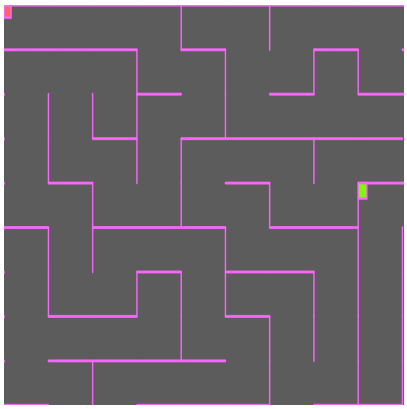


Figure 30: TC003

5.2.4 TC-004

L'algoritmo genera Dei punteggi in base a diversi fattori —Non Passato
L'algoritmo genera un punteggio ma non è basato sui vari punti descritti nel requisito di riferimento.

5.2.5 TC-005 — TC006

L'applicazione una volta lanciata aprirà una schermata di Home
L'utente sulla pagine Home pu'o impostare la difficolt'a del livello

Figure 31: TC005 . TC006

5.2.6 TC-007

L'utente sulla pagine Home può impostare il proprio nome e vedere il punteggio — Non passato

5.2.7 TC-008

L'utente sulla pagine Home può visualizzare i punteggi dopo ogni partita — Non passato

5.2.8 TC-009

All'interno del labirinto vengono generati ostacoli e trappole — Non passato

5.2.9 TC-010

l giocatore avrà delle vite per completare i labirinti — Non passato

5.2.10 TC-011

Il giocatore Potrà muoversi nel labirinto — passato

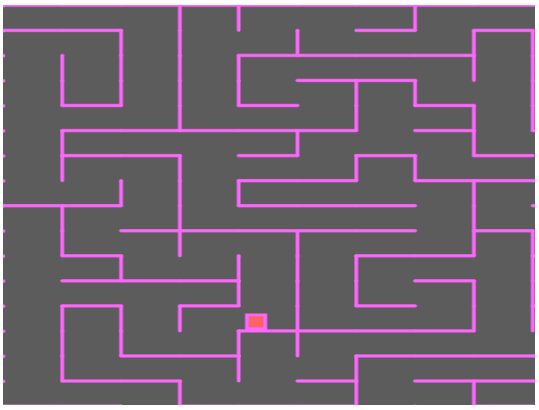


Figure 32: TC011

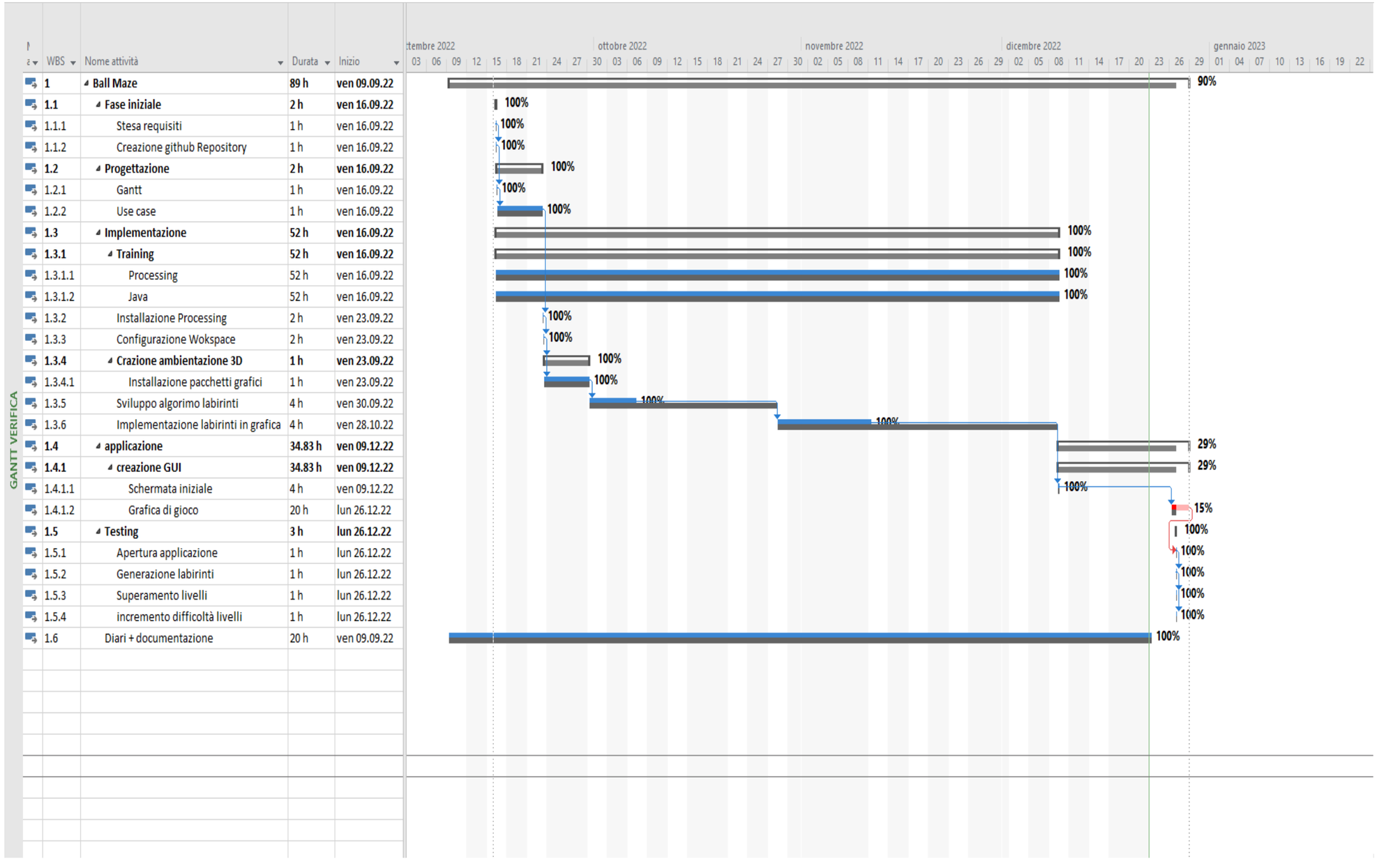
5.2.11 TC-010

Generazione di labirinti in 3D — non passato

5.3 Mancanze/limitazioni conosciute

I test inerenti al punteggio e al salvataggio dei dati possono essere eseguiti solo in parte siccome sono stati integrati solo nella versione di demo e non nella versione del gioco.
Per quanto riguarda la grafica 3D l'implementazione è risultata troppo complessa e pesante per far si che potesse avere un senso di sviluppo all'interno del gioco.

6 Consuntivo



Il progetto per la parte iniziale è andato seguento molto precisamente le tempistiche che mi ero preimpostato.
La più grande differenza invece è stata quella dell'implementazione degli algoritmi per la risoluzione e la creazione dei labirinti. Inizialmente avevo pensato che sarebbero state le due attività che avrebbero richiesto più attenzione e tempo ma si è rivelato sbagliato. Inizialmente mi ero prefissato di spendere un totale di venti ore nella loro realizzazione ma alla fine ne ho usate solo quattro.
Mi sono trovato in grande anticipo rispetto ai piani. Quel tempo in più è servito per la realizzazione della grafica di gioco che comprendeva attività come la creazione del motore collisionale. il rendering dei labirinti e le logiche di gioco.
Rispetto a quello che mi ero prefissato questo lasso di tempo aggiuntivo è durato all'incirca il tempo che avevo salvato in precedenza.
In conclusione il progetto è arrivato ad un risultato che considero buono entro i tempi a disposizione. Le uniche differenze tra la pianificazione e il Gantt verifica sono l'inversione della tempistica degli algoritmi e della logica/grafica di gioco.

7 Conclusioni

Questo progetto si è rivelato molto interessante nella sua realizzazione. Mi ha permesso di approfondire e di studiare degli algoritmi matematici che ho dovuto poi rappresentare graficamente come gioco.

7.1 Sviluppi futuri

Il gioco al momento è single player, ovvero che l'unica sfida che esiste è con se stessi. Penso che sarebbe interessante, come sviluppo, la messa in piedi di un server web nel quale caricare l'applicazione. Questo permetterebbe di poter avere delle sfide in simultanea tra più giocatori. La sfida sarebbe dunque non più con se stessi ma contro amici o persone sconosciute.
Si possono anche considerare cambiamenti a livello di gioco come l'implementazione di trappole, la riduzione di visibilità e molti altri dettagli.

7.2 Considerazioni personali

Per lo svolgimento di questo progetto ho usato un linguaggio chiamato processing che è basato su Java. Non essendo il linguaggio in cui mi destreggio meglio è stato interessante riuscire ad applicare concetti complessi e riuscire ad arrivare ad un risultato che ritengo essere buono. Ci sono state diverse problematiche, specialmente in gestione del tempo che mi hanno costretto a spendere più tempo di quel che pensassi per lo svolgimento di alcuni segmenti di codice.
In fine posso comunque essere felice del risultato ottenuto con il tempo a disposizione.

8 Bibliografia

8.1 Bibliografia per articoli di riviste

Non consultato

8.2 Bibliografia per libri

Non consultato

8.3 Sitografia

8.3.1 Processing

Documentazione: https://processing.org/reference — consultato per tutta la durata del progetto.

Graphics: https://processing.org/reference/createGraphics.html — 21.10.2022.

8.3.2 Overleaf

Overleaf: https://www.overleaf.com/learn — consultato per tutta la durata del progetto.

8.3.3 Algoritmo recursivo crazione labirinti

Fonte: https://en.wikipedia.org/wiki/Depth-first$_s earch$|23.09.2022 – 30.09.2022.

Metacodice: http://www.migapro.com/depth-first-search/ —23.09.2022 - 30.09.2022.

8.3.4 Algoritmo risoluzione labirinti

Fonte: https://en.wikipedia.org/wiki/A*$_s earch_a lgorithm$|30.09.2022 – 14.09.2022.

metacodice: https://levelup.gitconnected.com/a-star-a-search-for-solving-a-maze-using-python-with-visualization-b0cae1c3ba92 —30.09.2022 - 14.09.2022.

8.3.5 Collisioni

Collisioni: https://happycoding.io/tutorials/processing/collision-detection —28.10.2022 - 18.11.2022.

8.3.6 altri link

Youtube: https://www.youtube.com/ — Alcune volte per ispirazione e chiarimenti.

9 Allegati

Diari di lavoro

Abstract - it

Abstract - eng