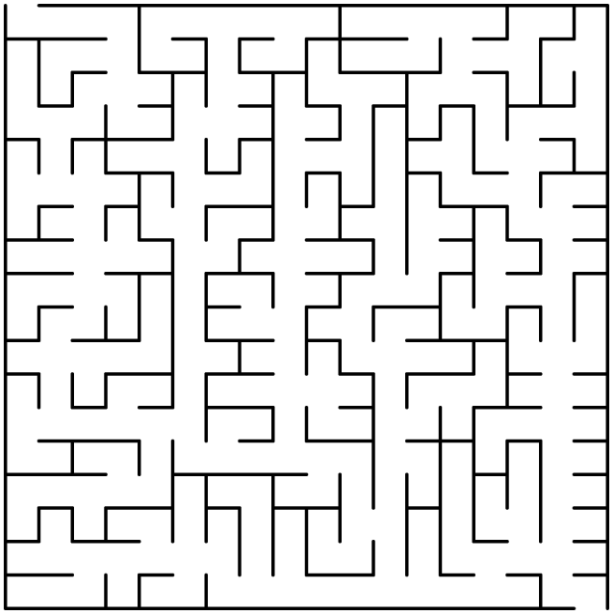
Homework 1



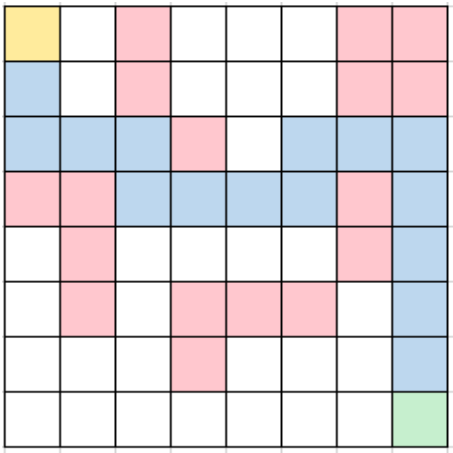
1. Descrizione e rappresentazione del gioco.

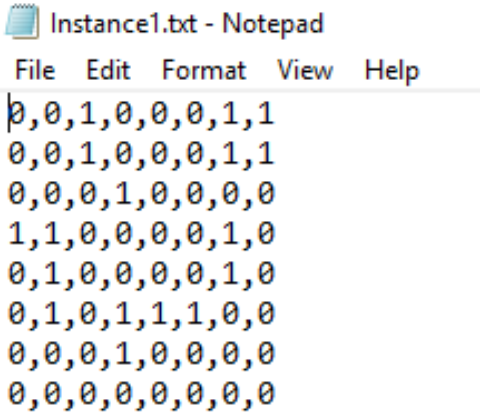
Il gioco che si è scelto di rappresentare è quello del **labirinto**. Il gioco consiste nel trovare la strada più breve che, dal punto di partenza, porti a quello di arrivo.

Si è deciso di rappresentare il labirinto come una matrice (non per forza quadrata) in cui ogni cella poteva avere al suo interno un valore pari a 0 o ad 1, dove lo 0 rappresenta una cella "visitabile" e il valore 1 rappresenta il muro, cioè quelle celle non "visitabili" che deviano il cammino.

Si è scelto di dare l'opportunità all'utente di scegliere un labirinto (e di conseguenza la sua grandezza) a piacere, in cui l'ingresso sia posizionato alle coordinate (0,0) e l'uscita alle coordinate (n,m), dove n e m rappresentano la dimensione della matrice. Per fare ciò, è sufficiente che l'utente generi un file ".txt" in cui rappresenti l'istanza da prendere in esame e, poi dia tale file come input al codice, quando viene richiesto.

Di seguito vi sono un esempio di labirinto rappresentato secondo la convenzione descritta (il riquadro in giallo e in verde sono rispettivamente quello di partenza e quello di arrivo, i riquadri in rosso rappresentano i muri del labirinto e quelli in azzurro è il percorso scelto) e la corrispondente rappresentazione in 0-1 nel file ".txt":





Quando verrà eseguito il codice, la prima cosa che verrà visualizzata è:

```
Insert the name of the file containing the input maze in this format
 0,0,0,1
 1,0,0,1
 1,0,0,0
 0,0,0,0
where 0 is an empty cell and 1 is a wall;
starting point will be (0, 0), ending will be (n, m) of the matrix nxm: Instance1.txt
```

2. Descrizione del codice

2.1 GameModel

Nella cartella **GameModel** vi è il file **__init__.py**, dove viene eseguita la rappresentazione del gioco.

Prima di tutto viene creato il modello con le caratteristiche del gioco tramite la classe **MazeGameRepresentation**, la quale è caratterizzata dai seguenti attributi:

1. **matrix:** è la matrice che rappresenta il labirinto.
2. **pos:** è la posizione iniziale da dove parte l'esplorazione del labirinto.
3. **ending:** è la posizione finale che permette di uscire dal labirinto.

A seguire vi è la classe **MazeState** la quale si occupa di rappresentare lo stato. Ha come attributi lo stato **parent**, l'**euristica** applicata e la **rappresentazione** del gioco.

Segue la classe **Game** ereditata dalla classe **MazeGame**. Quest'ultima è quella che descrive le "regole" del gioco. Infatti è caratterizzata dai metodi **neighbors** e **solution**, i quali si occupano rispettivamente di individuare le mosse che è possibile fare e di verificare se quella data è oppure no la soluzione finale del gioco.

2.2 Heuristic

All'interno della cartella **Heuristic** vi si trova il file **__init__.py**, nel quale sono elencate tutte le euristiche scelte che verranno applicate nella scelta del miglior percorso da intraprendere. Il file è strutturato nel seguente modo: vi è la definizione della classe **Heuristic**, ereditata dalla classe **MazeHeuristic**. In quest'ultima vi sono le applicazioni delle varie euristiche allo stato passato in input.

L'implementazione delle euristiche viene fatta da metodi esterni alla classe.

Le euristiche scelte sono tre:

1. **Manhattan Heuristic:** E' un'euristica la quale data la posizione attuale, calcola la distanza di Manhattan tra essa e la posizione finale. Questo valore sarà il peso assegnato allo stato. Si basa sulla seguente formula:

   Dati due coordinate C1=(x1,y1) e C2=(x2.y2), la distanza di Manhattan tra queste due coordinate è

   $$H(s)= |(x2-x1)|+|(y2-y1)|$$

2. **Chebyshev Heuristic**: Come l'euristica di Manhattan, utilizzando però la distanza di Chebyshev.

   Dati due coordinate C1=(x1,y1) e C2=(x2.y2), la distanza di Chebyshev tra queste due coordinate è

   $$H(s)= \max\{ |(x2-x1)|,|(y2-y1)|\}$$

3. **Euclide Heuristic**: Come le due precedenti euristiche, utilizzando però la distanza euclidea.

   Dati due coordinate C1=(x1,y1) e C2=(x2.y2), la distanza di Euclide tra queste due coordinate è

   $$H(s)= sqrt{[(x2-x1)+(y2-y1)]^2}$$

2.3 Main

All'interno della cartella di progetto vi è uno script python chiamato **main.py**, che è il "cuore" di tutto il progetto, in quanto composto da metodi che si occupano di applicare le euristiche sulle istanze in input del labirinto.

È infatti possibile trovarvi i seguenti metodi:

- **main:** È il segmento che permette all'utente di inserire l'istanza da lui scelta e che, una volta inserita tale istanza richiama i metodi per calcolare il miglior percorso da fare per arrivare all'uscita del labirinto. Restituisce poi una stampa dei risultati: in particolar modo stampa a video la soluzione trovata, il tempo impiegato per trovarla e il numero totale di stati visitati.
- **search:** È il segmento richiamato dal main che, presi in input l'istanza del gioco e lo stato iniziale, inizia la ricerca del miglio cammino da percorrere nel labirinto. Prima di tutto inserisce lo stato iniziale nell'orizzonte (che sono tutti i possibili stati a cui è possibile arrivare da quello attualmente sotto esame) e viene eseguito un ciclo sul numero di elementi contenuti nell'orizzonte. Dopodiché richiama la funzione **pick**, la quale torna uno stato a cui è associata la mossa con il minor valore euristico associato (cioè la miglior mossa da fare) e viene assegnato a view. Se lo stato iniziale corrisponde alla soluzione finale, allora viene chiamata la funzione **backpath**, la quale torna la lista degli stati che hanno portato alla soluzione finale; in caso contrario, quello stato viene inserito tra quelli esplorati, si recupera il "vicinato" di quello stato (cioè le possibili mosse) e si aggiorna l'orizzonte e si prosegue con il ciclo sugli elementi dell'orizzonte. Se invece a view non viene assegnato alcuno stato allora la funzione ritorna None.
- **pick:** È la prima funzione chiamata dal metodo search: ritorna lo stato a cui è associato il minor valore euristico, calcolato tramite la funzione argMin, richiamata proprio in questo metodo.
- **argMin:** È il metodo che applica le euristiche di cui discusso sopra al set di stati passati in input. Ogni volta che valuta uno stato, in ogni applicazione di euristica, salva tale stato e il valore euristico ottenuto ad esso associato, nel set locale localState. Una volta esaminati tutti gli stati nel set passato in input, viene scelto quello con il minimo valore euristico associato e viene inserito nel set globale dictOfState.
- **Backpath:** È l'ultimo metodo nello script main, il quale si occupa, preso in input uno stato, di ritornare una lista con tutti gli stati (e quindi le mosse da fare) che portano alla soluzione trovata.

Per concludere, nella cartella dell'Homework vi sono diverse instanze descritte in file ".txt", su cui verranno effettuati i test di cui al seguito saranno mostrati i risultati.

3. Test effettuati

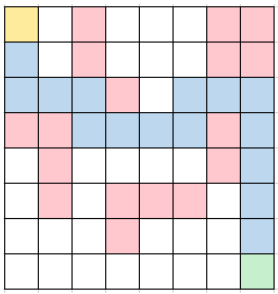
Nella rappresentazione dei labirinti viene usata la seguente convenzione:

La cella gialla rappresenta l'ingresso al labirinto, mentre quella verde rappresenta l'uscita. Le celle rosse sono i muri, mentre quelle in azzurro indicano il cammino migliore selezionato dalle euristiche

3.1 Istanza 1

Matrice 8x8

Rappresentazione del labirinto:



Rappresentazione matriciale del labirinto:

```
0,0,1,0,0,0,1,1
0,0,1,0,0,0,1,1
0,0,0,1,0,0,0,0
1,1,0,0,0,0,1,0
0,1,0,0,0,0,1,0
0,1,0,1,1,1,0,0
0,0,0,1,0,0,0,0
0,0,0,0,0,0,0,0
```

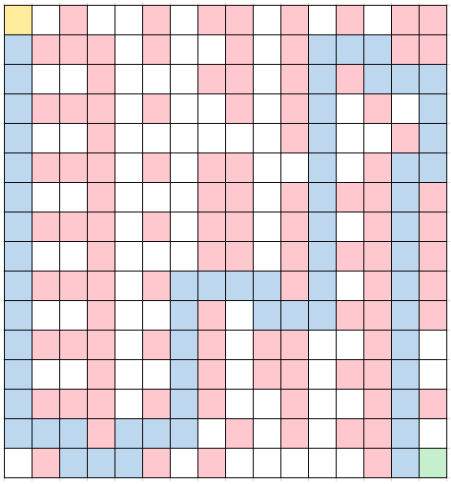
I risultati ottenuti sono stati i seguenti:

	Manhattan Heuristic	Chebyshev Heuristic	Euclide Heuristic
Tempo impiegato per trovare la soluzione	0.01567554473876953 secondi	0.04683995246887207 secondi	0.046911001205444336 secondi
Mosse da fare per arrivare alla soluzione	16	16	16
Numero di stati visitati	20	22	20

3.2 Istanza 2

Matrice 16x16

Rappresentazione del labirinto:



Rappresentazione matriciale del labirinto:

```
0,0,1,0,0,1,0,1,1,0,1,0,1,0,1,1
0,1,1,1,0,1,0,0,1,0,1,0,0,0,1,1
0,0,0,1,0,0,0,1,1,0,1,0,1,0,0,0
0,1,1,1,0,1,0,0,1,0,1,0,0,1,0,0
0,0,0,1,0,0,0,0,0,0,1,0,0,0,1,0
0,1,1,1,0,1,0,1,1,0,0,0,0,1,0,0
0,0,0,1,0,0,0,1,1,0,1,0,1,1,0,1
0,1,1,1,0,1,0,1,1,0,1,0,0,1,0,1
0,0,0,1,0,0,0,1,1,0,1,0,1,1,0,1
0,1,1,1,0,1,0,0,0,0,1,0,0,1,0,1
0,0,0,1,0,0,0,1,0,0,0,0,1,1,0,1
0,1,1,1,0,1,0,1,0,1,1,0,0,1,0,0
0,0,0,1,0,0,0,1,0,1,1,0,1,1,0,0
0,1,1,1,0,1,0,1,0,0,1,0,0,1,0,1
0,0,0,1,0,0,0,0,1,0,1,0,1,1,0,0
0,1,0,0,0,1,0,1,0,0,0,0,0,1,0,0
```

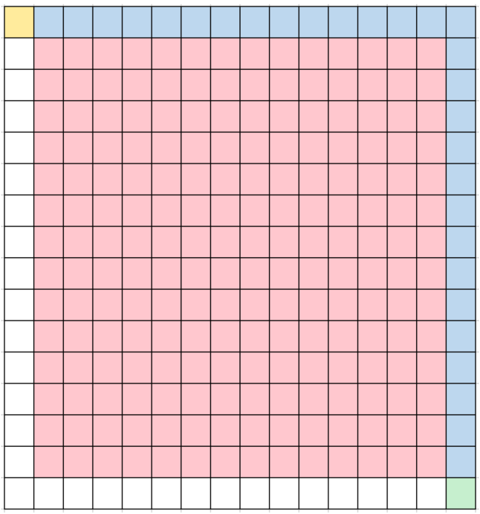
I risultati ottenuti sono stati i seguenti:

	Manhattan Heuristic	Chebyshev Heuristic	Euclide Heuristic
Tempo impiegato per trovare la soluzione	0.2968292236328125 secondi	0.48439741134643555 secondi	0.3749871253967285 secondi
Mosse da fare per arrivare alla soluzione	62	62	62
Numero di stati visitati	120	141	129

3.3 Istanza 3

Matrice 16x16

Rappresentazione del labirinto:



Rappresentazione matriciale del labirinto:

```
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

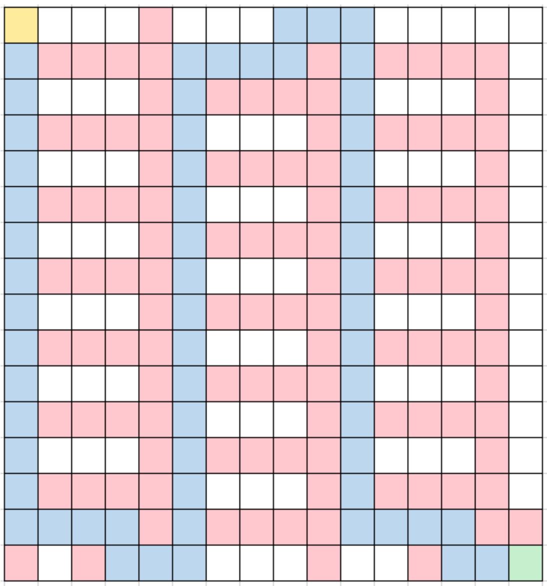
I risultati ottenuti sono stati i seguenti:

	Manhattan Heuristic	Chebyshev Heuristic	Euclide Heuristic
Tempo impiegato per trovare la soluzione	0.046936988830566406 secondi	0.0625 secondi	0.046877145767211914 secondi
Mosse da fare per arrivare alla soluzione	30	30	30
Numero di stati visitati	31	39	31

3.4 Istanza 4

Matrice 16x16

Rappresentazione del labirinto:



Rappresentazione matriciale del labirinto:

```
0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0
0,1,1,1,1,0,0,0,0,1,0,1,1,1,1,0
0,0,0,0,1,0,1,1,1,1,0,0,0,0,1,0
0,1,1,1,1,0,0,0,0,1,0,1,1,1,1,0
0,0,0,0,1,0,1,1,1,1,0,0,0,0,1,0
0,1,1,1,1,0,0,0,0,1,0,1,1,1,1,0
0,0,0,0,1,0,1,1,1,1,0,0,0,0,1,0
0,1,1,1,1,0,0,0,0,1,0,1,1,1,1,0
0,0,0,0,1,0,1,1,1,1,0,0,0,0,1,0
0,1,1,1,1,0,0,0,0,1,0,1,1,1,1,0
0,0,0,0,1,0,1,1,1,1,0,0,0,0,1,0
0,1,1,1,1,0,0,0,0,1,0,1,1,1,1,0
0,0,0,0,1,0,1,1,1,1,0,0,0,0,1,0
0,1,1,1,1,0,0,0,0,1,0,1,1,1,1,0
0,0,0,0,1,0,1,1,1,1,0,0,0,0,1,1
1,0,1,0,0,0,0,0,0,1,1,0,1,0,0,0
```

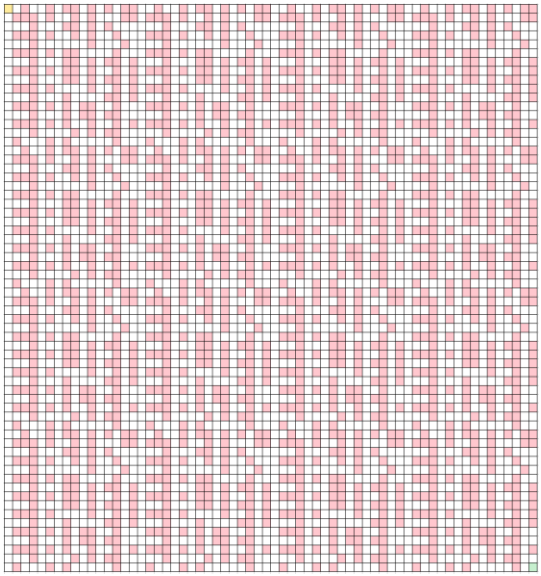
I risultati ottenuti sono stati i seguenti:

	Manhattan Heuristic	Chebyshev Heuristic	Euclide Heuristic
Tempo impiegato per trovare la soluzione	0.3437201976776123 secondi	0.21875357627868652 secondi	0.234358549118042 secondi
Mosse da fare per arrivare alla soluzione	60	60	60
Numero di stati visitati	141	111	110

3.5 Istanza 5

Matrice 64x64

Rappresentazione del labirinto:



Rappresentazione matriciale del labirinto:

Vedi file “Istance5.txt”

I risultati ottenuti sono stati i seguenti:

	Manhattan Heuristic	Chebyshev Heuristic	Euclide Heuristic
Tempo impiegato per trovare la soluzione	7.109443664550781 secondi	3.6563241481781006 secondi	4.140713214874268 secondi
Mosse da fare per arrivare alla soluzione	220	198	200
Numero di stati visitati	483	336	361

4 Conclusioni

Dai test si evince che per rappresentazioni di labirinto con matrici di dimensione ridotte l'euristica migliore da utilizzare è quella di Manhattan, mentre per matrici di dimensioni elevate l'euristica migliore è quella di Chebyshev. Per quanto riguarda l'euristica di Euclide, possiamo notare che, in ogni caso, tende ad avere prestazioni che si trovano tra la Manhattan e la Chebyshev.