

|  |
| --- |
| HE-Arc INF3-dlmb |
| Rapport TP IA |
| Algorithmes génétiques |

|  |
| --- |
| Monnet Joris  29/11/2020 |

Table des matières

[Introduction 2](#_Toc57587708)

[Choix lors de l’implémentation 3](#_Toc57587709)

[Génération du labyrinthe : 3](#_Toc57587710)

[Affichage des labyrinthes : 3](#_Toc57587711)

[Implémentation de l’algorithme : 3](#_Toc57587712)

[Encodage : 5](#_Toc57587713)

[Fitness : 5](#_Toc57587714)

[Sélection : 5](#_Toc57587715)

[Crossover : 6](#_Toc57587716)

[Mutation : 6](#_Toc57587717)

[Arrêt de l’algorithme : 6](#_Toc57587718)

[Population : 6](#_Toc57587719)

[Résultats : 7](#_Toc57587720)

[Grille 10x10 : 7](#_Toc57587721)

[Grille 15x15 : 8](#_Toc57587722)

[Grille 20x20 : 9](#_Toc57587723)

[Grille 30x30 : 10](#_Toc57587724)

[Grille 40x40 : 11](#_Toc57587725)

[Conclusion : 12](#_Toc57587726)

[Sources : 12](#_Toc57587727)

# Introduction

Ce TP propose de mettre en place un algorithme génétiques pour trouver un chemin. Le plateau est en réalité une grille d’une taille MxN ou chaque case à deux valeurs possibles, 0 ou 1, libre ou mur. Le but de notre algorithme est de trouvé un chemin en ne passant que par des cases libres du point (0,0) au point (M-1, N-1). Les tailles de grille à tester sont déjà définies avec un temps maximum d’exécution pour chaque taille de grille :

|  |  |
| --- | --- |
| Taille | Temps max |
| 10x10 | 10s |
| 15x15 | 15s |
| 20x20 | 30s |
| 30x30 | 60s |
| 40x40 | 90s |

Dans le cadre de ce TP je ne testerai donc que des grilles où M = N. De plus, le Framework DEAP sera utilisé.

Tout d’abord je vais vous présenter les choix que j’ai réalisé lors de l’implémentation de l’algorithme, notamment au niveau de ses paramètres. Je montrerai ensuite les résultats obtenus pour différentes grilles.

# Choix lors de l’implémentation

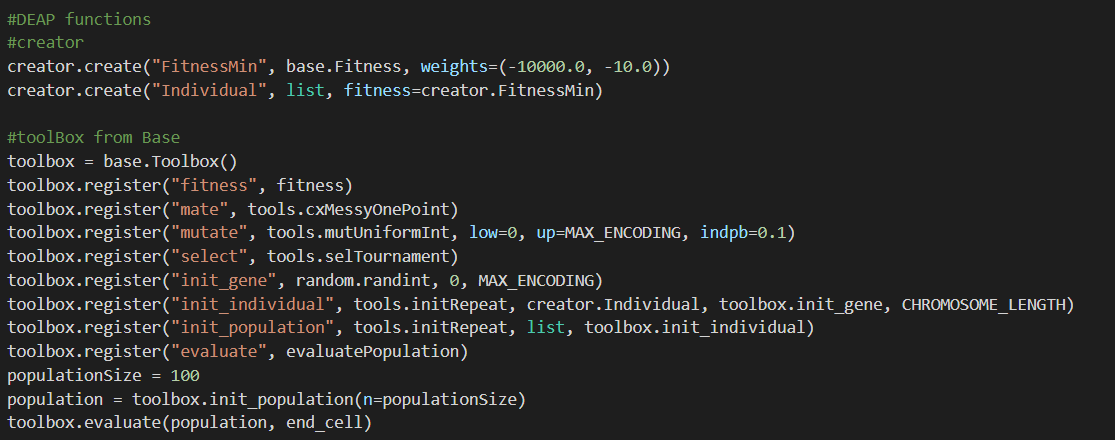
## Génération du labyrinthe :

Afin de générer un labyrinthe, j’ai utilisé une fonction pour créer le labyrinthe aléatoirement suivant une taille MxN avec 10% de 1 et 90% de 0. Je rappelle que les cases 0 sont les cases libres et les cases avec mur sont les cases 1. Ce labyrinthe est interprété par le programme comme étant une matrice.

## Affichage des labyrinthes :

J’ai gardé l’implémentation fourni avec des cases blanches lorsqu’elles sont libres, des cases grisées lorsqu’il s’agit d’un mur et des cases bleues pour montrer le chemin.

## Implémentation de l’algorithme :

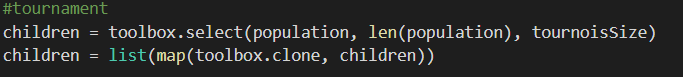
D’abord, on génère la grille aléatoire, Ensuite on créer une population, on l’évalue, on sélectionne via les tournois (10 ici) des membres de la population, on accouple ces individus pour donner naissance à des enfants, on en mute certains et cela en boucle jusqu’ atteindre notre objectif. D’un point de vu code, d’abord on lance les fonctions utiles de DEAP : 

On voit qu’ici la population est créée, évaluée un première fois.

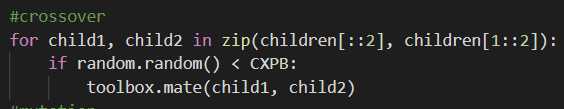
Ensuite on met en place la boucle d’exécution des étapes listées ci-dessus :



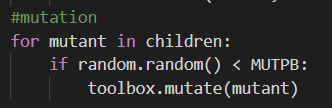
On sélectionne via le tournois :



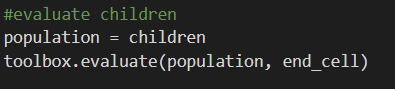
On met en place l’accouplement(crossover) avec un paramètre choisi :



Ensuite on mute certain Individus avec un paramètre choisi :



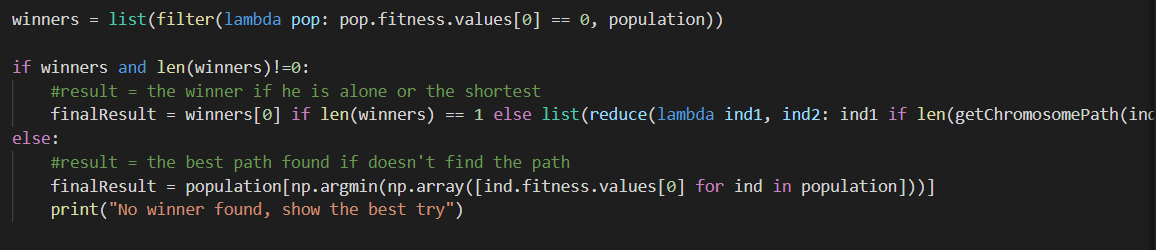
Enfin on évalue :



Une fois un chemin trouvé, on le fait muter en essayant de trouver un meilleur chemin :



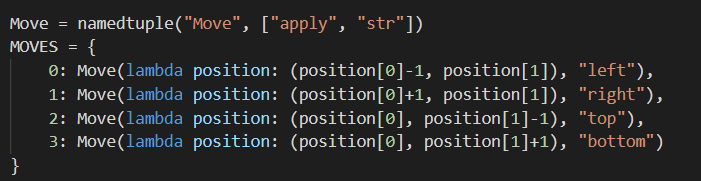
Enfin on récupère le winner, c’est-à-dire le seul chemin trouvé ou le meilleur des chemins trouvés ou l’essai qui s’est le plus rapproché du point d’arrivée :



Enfin, on finit l’algorithme par un affichage simple du temps de process, du nombre d’itérations, des paramètres et du résultat sur la grille.

## Encodage :

Les gènes sont encodés sur deux bits puisqu’il n’y a que quatre déplacements : gauche, droite, haut, bas. Dans mon cas j’utilise un namedTuple pour pouvoir directement associer un mouvement à une action sur une position. Je remercie M. Maxime Welcklen pour cette idée, très utile et efficace.

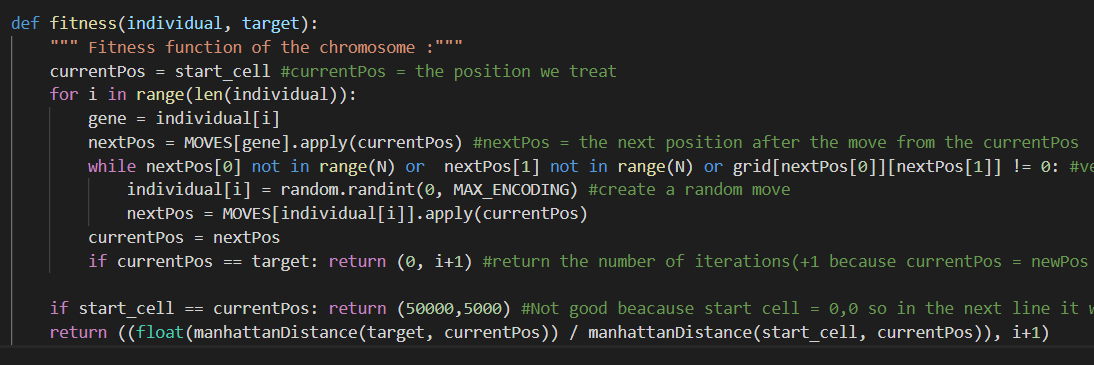


## Fitness :

J’utilise dans ma fonction de fitness une distance de Manhattan et je prends en compte la distance qu’il reste à parcourir en créant un coefficient. Le but du programme est donc de tendre vers 0, c’est-à-dire quand la position actuelle sera au bout de son voyage, au contraire elle vaut 1 lorsque la position actuelle est la position de start :



La fonction fitness consiste a appliquer le mouvement à une case et directement corriger si la valeur pour bouger aléatoire fait sortir de la grille ou lorsque le chemin passe dans un mur :



Deux return avant la distance de Manhattan : le premier est simplement si on vient d’arriver à la dernière étape du chemin, c’est-à-dire au point d’arrivée. Le deuxième est la mise en place d’une valeur énorme (5000,5000) dans le return dans le cas où le point actuel serait le point de départ, en effet cela induirait une division par 0 dans le coefficient retourné avec la distance de Manhattan un ligne en dessous (si il n’y avait pas ce return).

## Sélection :

J’utilise un tournois à 10. Ce paramètre est modifiable via la variable tournoisSize :



## Crossover :

Ce paramètre a été défini via des essais successifs, il est de 0.75 dans mon cas. Il est modifiable dans la variable CXPB :



## Mutation :

Ce paramètre a été trouvé via des essais successifs conjointement avec CXPB. Il est de 0.65 et est modifiable dans la variable MUTPB :

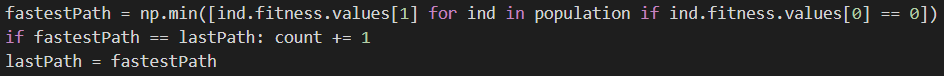


## Arrêt de l’algorithme :

Le choix a été fait de s’arrêter lors de la fin du temps maximum alloué ou lorsqu’un chemin a été confirmé comme étant le plus cours 5 fois.



La variable count permet de voir combien de fois le plus court chemin a été confirmé :



## Population :

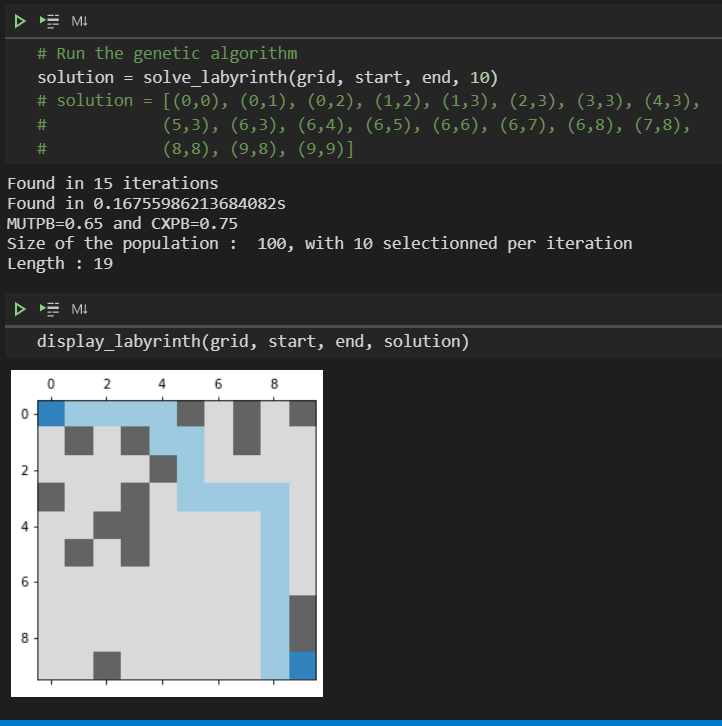
La population est créée suivant une taille définie (100 dans mon cas) elle peut être modifiée via la variable populationSize :



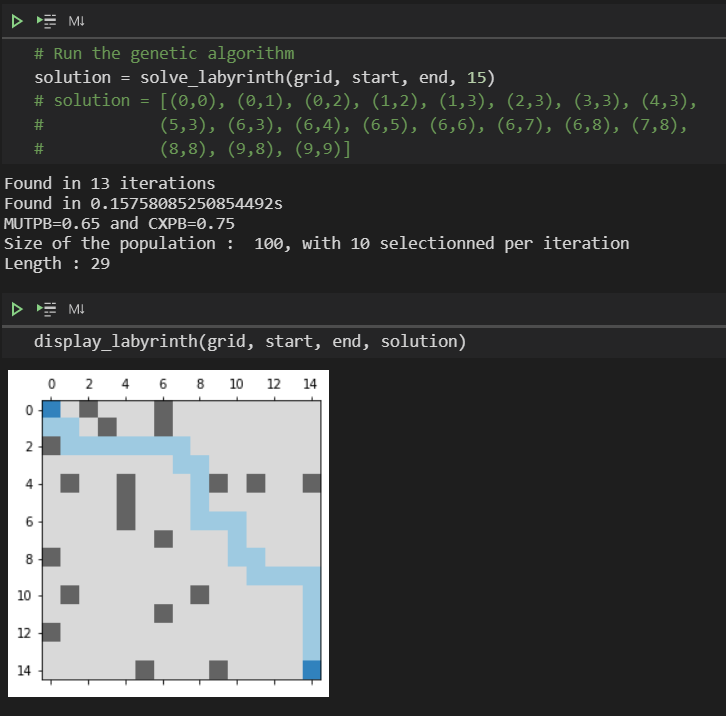
Avec tous ces paramètres, je vais donc à présent vous présenter les résultats observés :

# Résultats :

## Grille 10x10 :



## Grille 15x15 :

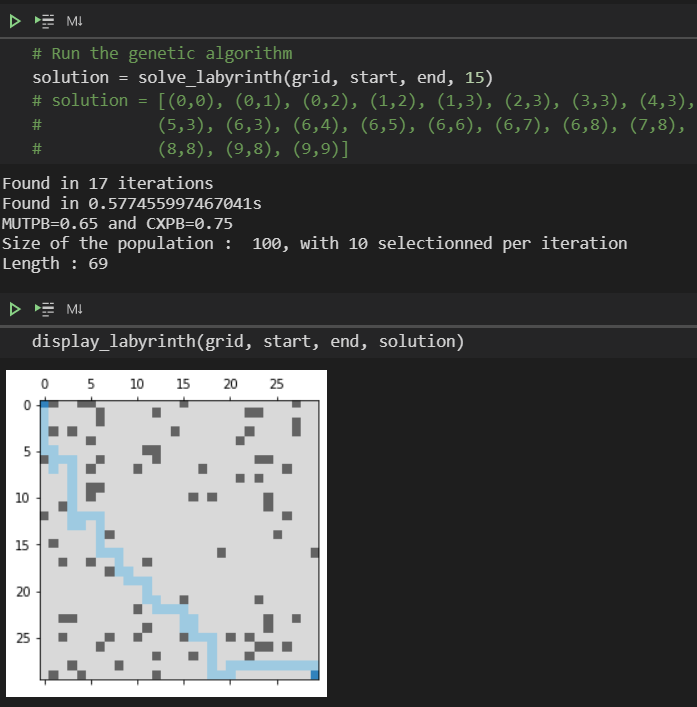


## Grille 20x20 :

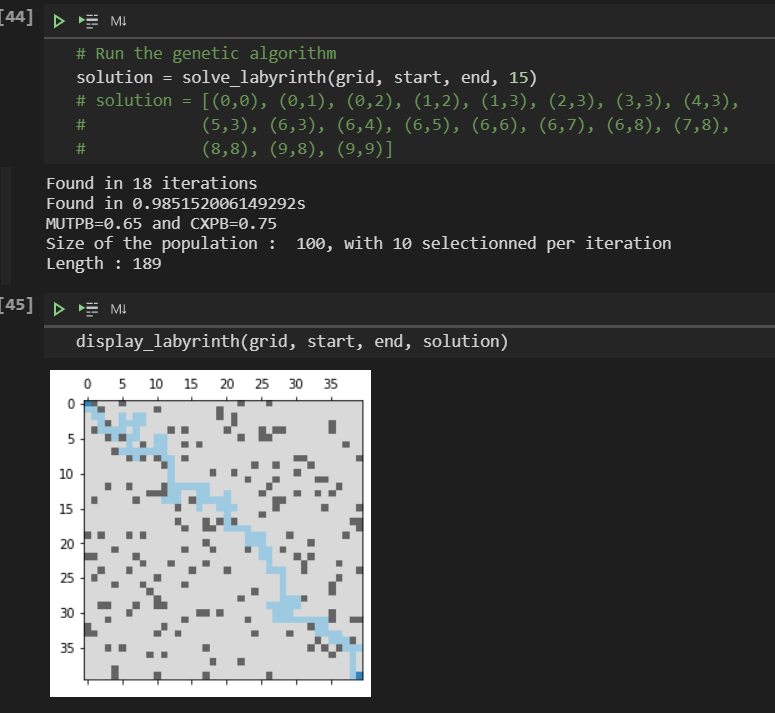


## Grille 30x30 :

## 



## Grille 40x40 :



# Conclusion :

On peut voir au vu de ces résultats que les tailles 30x30 et 40x40 ne sont pas sur les chemins les plus courts à chaque fois notamment les deux exemples ci-dessus. Il est donc utile de trouver de meilleurs paramètres pour du pathfinding. Cela étant dit le but de ce TP était surtout de trouver des solutions en dessous d’un temps donné. On voit ici qu’on reste en dessous des 1 s pour toutes les tailles de grilles testées soit largement en dessous des limites accordées. Les résultats sont donc satisfaisants avec une marge de progression.

Ce TP aura permis de prendre en main les outils du Framework DEAP et de mettre en place un algorithme génétique. Dans un cadre plus grand, cela m’a donné des idées pour mon P3 qui étant une application de randonnée demande la mise en place d’algorithme de path finding.

# Sources :

Documentation officielle DEAP, aide de M. Welcklen cité plus haut, <https://en.wikipedia.org/wiki/Genetic_algorithm>

<https://deap.readthedocs.io/en/master/examples/>

https://www.researchgate.net/publication/235707001\_DEAP\_Evolutionary\_algorithms\_made\_easy