Labyrinthe génétique

Participant : Clement Wang

Description :

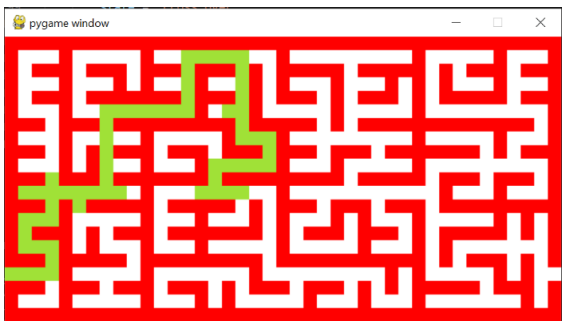
Ce projet consiste à résoudre un labyrinthe par un algorithme génétique. On s'impose des contraintes particulières : on ne voit pas le labyrinthe, les seules informations auxquelles on a accès sont le nombre de fois où on rentre dans un mur et la distance à la sortie.

Mais que sont les algorithmes génétiques ? Et bien, ce sont des algorithmes évolutifs inspirés de la sélection naturelle 😊. En effet, on part d'une population aléatoire d'individus caractérisés par un génome. A chaque génération, une partie de la population meurt et est remplacée par une nouvelle des progénitures obtenues en réalisant des « cross overs » de manière aléatoire. Les nouveaux individus sont formés à partir d'individus déjà existants. Se déroulent en parallèle des mutations aléatoires modifiant le génome d'individus existants. Mais comment choisir la population mourante ? Par sélection naturelle bien évidemment ! Pour cela, on suppose que l'on dispose d'une fonction de score permettant d'attribuer un score à un individu en particulier. On peut alors faire mourir les pires individus.

J'ai dit des choses assez générales donc ça doit encore être un peu flou. Le cross over, les mutations et la fonction de score dépendent du problème. Je ne peux qu'inciter à faire un tour vers notre formation sur les algorithmes génétiques : ici.

L'idée du projet vient d'Axel D. :D

Je me suis fortement inspiré de ce site pour coder ce projet.



Un petit aperçu

Le but :

- Avoir un projet visuel simple à faire, entre deux partiel (projet réalisé juste avant le partiel de physique quantique 😊)
- Refaire de la POO en Python
- Réaliser un squelette d'algorithme génétique facilement réutilisable pour d'autres projets

Des difficultés rencontrées :

- Adaptation à de grands labyrinthes
- Pygame pas adapté pour afficher des choses

Implémentation :

Génération aléatoire de labyrinthe :

Afin de pouvoir adapter facilement la taille du labyrinthe, j'ai décidé de générer des labyrinthes aléatoires. Plusieurs façons de faire s'offraient à moi dont celles que j'avais déjà implémentées en Coding Weeks. Je me suis laissé tenté par l'algorithme récursif que je n'avais encore jamais vu (Lien). (Va savoir pourquoi il y a plus d'algorithmes sur le wiki anglais que français).

C'est assez explicite : on prend une partie rectangulaire, on la coupe en deux aléatoirement par un mur vertical ou horizontal et on crée une ouverture entre les deux parties. Et on continue récursivement sur les deux parties créées. Pour contourner le problème de la pile limitée, j'utilise simplement une file, implémentée par une liste Python. La partie compliquée ici a été de bien jouer sur la parité pour ne pas avoir de labyrinthe « moche ».

Algorithme génétique :

La partie générale est codée dans l'objet GA(). On y trouve toutes les méthodes utiles pour les algorithmes génétiques. Le seul point à changer pour un autre projet serait l'attribut max_length, qui est nécessaire pour l'implémentation d'individus dans ce projet.

La classe Sample() permet de créer les individus de notre algorithme génétique. Elle contient, les méthodes de créations aléatoires d'individus, de cross over et de mutation. Dans ce projet, le génome correspond à une suite de mouvements (haut, bas, gauche, droite, ou rien). Tous les génomes sont des listes de même taille, le mouvement « rien » servant à réguler cette taille si besoin.

Un point assez important est le Cross Over et la mutation employée. L'implémentation du site sur lequel je me suis basé consistait, pour le cross over, à prendre les k premiers mouvements d'un individu et les n-k derniers d'un autre en les choisissant de façon aléatoire. La mutation consistait à remplacer seulement un mouvement. Cette implémentation fonctionne pour les petits labyrinthes avec peu de mouvements mais est bien évidemment insuffisante pour les grands labyrinthes avec beaucoup plus de mouvements.

Le cross over n'est plus fait à l'aveugle : on prend les k premiers mouvements du meilleur parent et les n-k mouvements du deuxième parent, k étant tiré au hasard. L'idée est de ne pas prendre un début inutile car cela ruinerait la progéniture. La question s'est également posée de ne pas mettre de cross over du tout car en un sens, les mutations pourraient tout faire. Mais j'ai décidé de les garder pour avoir l'esprit de l'algorithme génétique. De plus, ça fonctionne bien à la fin.

La mutation que j'ai employée consiste à tirer aléatoirement le point de départ de changement dans le génome ainsi que le nombre de mouvements à changer. Les mouvements à changer sont tirés au hasard parmi les 5 actions possibles ET le fait de refaire la même action que la précédente. On favorise ainsi des mouvements plus longs.

Jeu :

Le jeu est perçu comme un environnement, renvoyant le score d'un individu. La fonction de score a été faite par itérations en voyant ce qui marche le mieux. On pénalise la distance finale à la sortie, le nombre de fois où on rentre dans un mur et on favorise l'exploration.

Interface graphique :

L'interface a été faite avec Pygame. La première version était très mal réalisée car je calculais chaque génération puis j'affichais le résultat, sans jamais écouter les événements de la fenêtre Pygame. Plusieurs solutions s'offraient à moi :

- faire du multithreading
- faire du pseudo multithreading en passant par deux programmes, un pour calculer et mettre les variables dans un fichier txt et un autre pour afficher en parallèle

Ces deux options étaient beaucoup trop laides à implémenter avec Pygame. J'ai alors plutôt décomposé toutes les étapes de l'algorithme génétique et remplacé toutes les boucles for par la boucle while principale et mis des « états » de l'avancée de la génération pour que tout fonctionne en online.

Mot de la fin - Ouverture :

Ce projet n'amène pas beaucoup de possibilités d'amélioration intéressantes. On peut bien sûr s'amuser à trouver de meilleures fonctions de cross over, de mutation et de score ou même trouver d'autres idées d'implémentation pour l'individu mais l'algorithme est déjà capable de résoudre les labyrinthes 10x10 assez rapidement. Une idée pourrait être de comparer cet algorithme génétique à un agent en RL.

Enfin, ça a été assez fun à coder, même si techniquement ce n'était pas très difficile, cela m'a permis de découvrir Pygame. De plus, on tire toujours de la satisfaction d'un projet qui ne réussit pas du premier coup, mais où on a passé quelques heures à réfléchir pour l'améliorer.

Liste de références :

- Idée de départ : https://tonytruong.net/solving-a-2d-maze-game-using-a-genetic-algorithm-and-a-search-part-2/
- Génération de labyrinthe : https://en.wikipedia.org/wiki/Maze_generation_algorithm#Recursive_implementation
- TP algo gen : https://automatants.cs-campus.fr/TPs#2020_algogen