REPUBLIQUE ALGERIENE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITIE MOHAMED BOUDIAF - M'SILA

FACULTE DE MATHEMATIQUES ET D'INFORMATI(

DEPARTEMENT D'INFORMATIQUE

MEMOIRE de fin d'études

Présenté pour l'obtention du diplôme de **MASTER**

Domaine : Mathématiques et Informatique

Filière : Informatique

Spécialité : Informatique Décisionnelle et Optimisation

Par

- LADJEDEL ASMA
- GHACHA YASSMINE

Thème

GENETIC PROGRAMMING FOR INTELLIGENT BEHAVIOR OF A VIRTUAL ROBOT IN A RANDOM ENVIRONMENT

Soutenu publiquement le : / /2022 devant le jury composé de :

Dr. Hemmak Allaoua	**UMB M'sila**	**Encadreur**
Pr. Akhrouf Samir	**UMB M'sila**	**Président**
Dr. Mehenni Tahar	**UMB M'sila**	**Examinateur**

Promotion : 2021 /2022

إهداء

الحمد له الذي أعاننا بالعلم وزيننا بالحلم وأكرمنا بالتقوى وأجملنا بالعافية

أريد أن أشكر نفسي على المجهودات التي بذلتها, أريد أن أشكر نفسي لعدم استسلامها و أ نها لم تأخذ أياما للراحة و التوقف, أريد أن أشكر نفسي لأنها معطاءة و تحاول دائما أن تعطي أكثر مما تأخذ, أريد أن أشكر نفسي كونها أنا فقط

أتقدم بإهداء عملي المتواضع إلى الدرع الواقي والكنز الباقي إلى من جعل العلم منبع اشتياقي لك أقدم وسام الاستحقاق ... أبي الغالي "لجدل إبراهيم" بارك الله في عمرك

إلى من ساندتني في صلاتها و دعائها ... رمز العطاء وذروة العطف والحنان حبيبتي أمي "بوطيبة سعاد" بارك الله في عمرك

إلى من هم سندي في الدنيا من يوم قدمت الى الحياة و مدعميني ماديا و معنويا إخوتي "محمد" "أمينة" "إيمان" "عبود" "سارة" "قيس".

الى جداتي ' خيرة ' و ' أم الخير' لطالما كنتما رمزا للطيبة و الحنان

الى جدي ' محمد ' و ' حسين ' الله يرحمهما ما زلتما في قلبي اتذكركم بدعائي

أتقدم بشكري الى خالاتي الغاليات

الى بنات خالتي حبيباتي ' مسعودة ' ' سارة ' ' ياسمن '

دائما كنتم جنبي في فرحي و في مواقفي الصعبة

أسُّوم

إهداء

إلى الجسر الذي يوصلني بالجنة و المرآة التي أرتني النور عندما كنت في الظلام إلى أعظم مؤثرة في شخصيتي و رفيقتي في كل خطواتي أمي الحبيبة "غشة نعناعة".

إلى الظل الذي آوي إليه في كل حين أبي الغالي " غشة مسعود".

إلى منبع آمالي وبهجتي إخوتي الأعزاء "اكرام"،"أنس"، "علي" .

إلى كل أقاربي الذين وقفو إلى جانبي و دعموني قولا و فعلا واختص بذكري خالي"رحماني" وزوجته الغالية"نعيمة".

إلى من مدت اياديهم في لحظات الضعف صديقاتي جميعا و الى من شجعني وكان سببا في استمراري .

ياسمين

شكر و عرفان

أولا وقبل كل شي لله الحمد والشكر و الفضل على ما نحن عليه ، و نبدأ شكرنا للأستاذ السيد ' هماك علاوة ' الذي أشرف

على بحثنا و كان نعم المرشد والموجه طوال فترة عملنا نشكره على جهده و كل ماقدمه لنا من علم و ابحاث و مناقشات

في كل سنوات تدريسه لنا.

و نتقدم بشكرنا الى اللجنة

نود ايضا ان نشكر الاستاذ ' لوناس بلال' و ' بن عزي مخلوف ' على قيمة النصائح والتعليقات و المساعدة ، اذ لهما

الفضل في اتضاح الاتجاه الذي كان علينا اتباعه.

الى رمز الصداقة أتقدم بشكري الى زميلي

الذي كان بجانبي في أعسر أوقاتي 'بلفار يوسف '

.

Table des matières

ملخص

في هذا المشـروع، يتحرك الروبوت عبر متاهة، ويتوقف بمجرد أن يصـطدم بالحائط ، أو يعود إلى الموقع الذي زاره بالفعل ، أو يصـل إلى المخرج. الهدف هو إيجاد طريق يؤدي إلى المخرج. نسـتخدم البرمجة الجينية لإنتاج سلوك ذكي للروبوت في منطقة عشوائية. نحن بحاجة إلى البحث عن أفضل تكوين للمعامل لتحسين كفاءة النهج. يجب أن يمتد العمل إلى حالة التعلم غير الخاضعة للإشراف.

الكلمات المفتاحية: البرمجة الجينية؛ التحسينات التوافقية؛ محاكاة الروبوت المتحرك.

Abstract

In this project, a robot moves through a maze, and stops as soon as it hits a wall, returns to a location it has already visited, or reached the exit. The goal is to find a route leading to the exit. We use genetic programming to produce an intelligent behavior of the robot in a random area. We need to look for the best parameter configuration to improve the approach efficiency. The work should be extended to unsupervised learning Case.

Keywords: Genetic Programming; Combinatorial Optimization; Mobile Robot Simulation.

.

Résumé

Dans ce projet, un robot se déplace dans un labyrinthe et s'arrête dès qu'il heurte un mur, retourne à un endroit qu'il a déjà visité ou atteint la sortie. Le but est de trouver un itinéraire menant à la sortie. Nous utilisons la programmation génétique pour produire un comportement intelligent du robot dans une zone aléatoire. Nous devons rechercher la meilleure configuration des paramètres pour améliorer l'efficacité de l'approche. Le travail doit être étendu au cas d'apprentissage non supervisé.

Mots-clés : Programmation génétique ; Optimisation Combinatoire ; Simulation de robots mobiles.

Table des matières

Table des matières

Liste des figures

Liste des algorithmes

Liste des tableaux

Introduction générale

Les algorithmes génétiques permettent une connexion entre le monde informatique et le monde naturel, ce qui a permis à un ordinateur d'imiter un humain. Cela relève du domaine de l'intelligence artificielle, qui a facilité la résolution de nombreuses complexités auxquelles une personne est confrontée dans sa vie quotidienne, car il existe de nombreux problèmes difficiles à résoudre par les méthodes traditionnelles, et parfois même à mettre en œuvre - et cela nécessite un beaucoup de temps. Doublez le temps et les efforts.

Cela a conduit les scientifiques à réfléchir au développement et à la numérisation d'appareils plus intelligents et à la création d'une machine intellectuellement puissante qui peut faire à peu près n'importe quel travail qu'un humain peut faire. De plus, il l'a fait apprendre tout seul sans avoir besoin d'intervention humaine pour le guider.

En général, une grande partie du développement mathématique du siècle s'est concentrée sur ce sujet (rappelez-vous ces problèmes constamment récurrents où il fallait obtenir la dérivée d'une fonction pour trouver ses maxima).

Des méthodes combinant analyse mathématique et recherche aléatoire sont apparues. Imaginez que vous dispersez de petits robots dans une région montagneuse. Ces robots peuvent suivre le chemin le plus raide qu'ils ont trouvé. Lorsque le robot atteint le sommet, il prétend avoir trouvé le meilleur. Cette méthode est très efficace, mais rien ne prouve que le niveau optimum soit trouvé, chaque bot peut se coincer au niveau optimum local. Ce type d'approche ne fonctionne qu'avec des espaces de recherche réduits.

Quel pourrait être le lien entre les méthodes d'optimisation et la vie artificielle ? [1]

Dans cette thèse, nous allons essayer d'employer des algorithmes génétiques et d'en tirer parti et de commencer à les appliquer à un robot virtuel dans un labyrinthe afin de résoudre le problème de déplacer le robot seul sans interférence afin qu'il puisse trouver les meilleures solutions dans n'importe quel environnement aléatoire.

Notre travail dans le but est de proposer une méthodologie pour l'optimisation et la simulation de robot dans le labyrinthe, ainsi que le développement d'un logiciel supportant cette méthodologie.

Le premier chapitre de ce mémoire propose, Un aperçu de l'algorithme génétique, ses mécanismes et voilà nous leurs Principes de base (Individu/chromosome/

Population), après nous sommes passés par les opérateurs des algorithmes génétiques (sélection, croisement, mutation).

Dans le deuxième chapitre nous avons continué en la présentation générale mais maintenait la programmation génétique, Les étapes préparatoires de la programmation génétique (Les terminaux fonctions, La fonction de performance, Les paramètres du contrôle ...), Les phases de la programmation génétique.

Dans le troisième chapitre est l'état de l'art nous présentons quoi les autres faire dans notre thème et présenter leur travail par un petite résume.

Dans le quatrième chapitre, simulation d'un robot mobile dans un labyrinthe par programmation génétique.

Enfin dans le dernier chapitre nous présentons notre application et expliquons comment nous le faisons.

Chapitre I

Les algorithmes génétiques

1. Introduction

Dans ce chapitre nous commence par une définition d'un algorithme génétique, puis ses mécanismes, les principes de base des algorithmes génétiques, puis les opérateurs génétiques, Enfin, nous avons mentionné certains des avantages et des inconvénients des algorithmes génétiques.

2. Définition

(GA), proposé pour la première fois par John Holland en 1975, est un type d'algorithmes de recherche et d'optimisation méta-heuristiques inspirés du principe de Darwin sélection naturelle. L'idée centrale de la sélection naturelle est que le plus apte survive. À travers le processus de sélection naturelle, les organismes s'adaptent pour optimiser leurs chances de survie dans un environnement donné. Des mutations aléatoires se produisent dans la description génétique d'un organisme, qui est ensuite transmise à ses enfants. Si une mutation s'avérait utile, ces enfants sont plus susceptibles de survivre pour se reproduire. Si cela devait être nocif, ces enfants sont moins susceptibles de se reproduire, donc le mauvais trait mourra avec eux. [2] Cela signifie qu'un GA peut trouver des solutions à des problèmes même lorsque le programmeur ne sait pas comment parvenir à une bonne solution mais a des connaissances sur les propriétés d'une telle solution. [6]

3. Mécanismes des algorithmes génétiques

L'AG de base se déroule comme suit : [3]

- Créez une population d'individus aléatoires, dans laquelle chaque individu représente une solution possible au problème posé.
- Évaluez l'aptitude de chaque individu - sa capacité à résoudre le problème spécifié.
- Sélectionnez des membres individuels de la population comme parents.
- Produisez des enfants en recombinant le matériel parental par croisement et mutation, et ajoutez-les à la population.
- Évaluez la condition physique des enfants.
- Répétez les étapes 3 à 5 jusqu'à ce qu'une solution avec l'objectif de remise en forme souhaité soit obtenue.

Dans lequel utilise une série d'étapes pour passer d'un ensemble de «chromosomes» à un autre. Ces chromosomes sont constitués d'un certain nombre de gènes, et chaque gène est une série de bits.

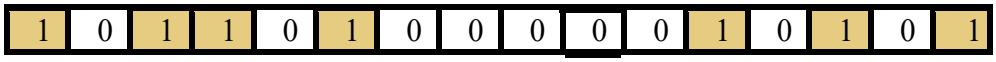


Figure 1.1 : codage d'un individu.

4. Principes de base des algorithmes génétiques

Les AGs font partie des algorithmes méta-heuristiques, ils travaillent sur un espace de recherche sous forme de population en y opérant d'une manière stochastique. Les caractéristiques qui font l'unicité de cette méthode sont : - Les AGs ne travaillent pas directement sur les paramètres, ces derniers sont codés avant l'opération. - Ils opèrent sur une population de solutions au lieu d'une seule solution. - Les AGs se basent sur des données aléatoires, donc ils sont probabilistes. Les AGs utilisent les éléments suivant pour se définir : [5]

- **Individu/chromosome/séquence** : une potentielle solution du problème abordé.
- **Population** : un ensemble de chromosomes de l'espace de l'exploration.
- **Environnement :** l'espace de l'exploration et de la recherche.
- **Fonction d'évaluation (fitness) :** la fonction objective qui sert à évaluer la précision de la solution du problème.

5. Fonctionnement des AG [6]

- **Initialisation** : Un ensemble de N chromosomes est aléatoirement créé formant une population de solutions potentielles.
- **Évaluation** : Décodage puis évaluation de chaque individus/chromosomes.
- **Sélection :** Après l'évaluation, une nouvelle population de N individus est créée à l'aide d'une méthode de sélection adéquate.
- **Opérations génétiques :** Croisement et mutation de certains individus au sein de la nouvelle population.
- **Retour :** Tant que la condition d'arrêt n'est pas satisfaite, l'algorithme recommence à partir de la phase d'évaluation.

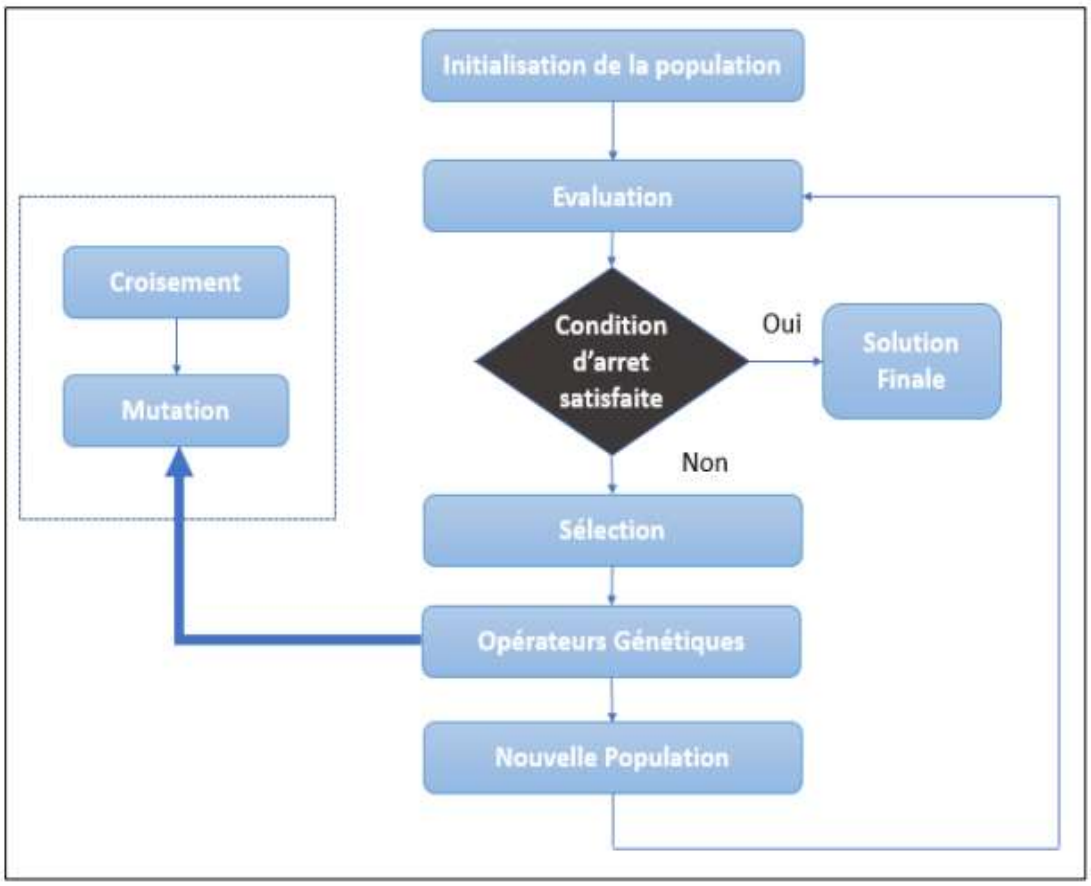


Figure 1.2 : Fonctionnement de l'algorithme génétique. {5]

6. Principes des algorithmes génétiques

L'application d'un algorithme génétique à un problème particulier nécessite cinq éléments suivants: [4]

- **Le codage des solutions (individus)**

Associe à chacun des points de l'espace de solution une structure de données. Elle vient généralement après une phase de modélisation mathématique du problème traité. La qualité du codage conditionne le succès de l'algorithme.

- **Une méthode de génération de la population initiale**

La population initiale qui servira de base pour les générations futures doit être non homogène.

- **Une fonction à optimiser**

Cette fonction sert à évaluer chaque individu, elle retourne une valeur réelle nommée : « fitness », qui sera utilisée dans le calcul de probabilité de sélection d'un individu.

- **Les opérateurs génétiques**

La puissance des algorithmes génétique réside dans leurs opérateurs génétiques (croisement et mutation). En effet, ces opérateurs permettent d'établir un balayage partiel de l'espace de recherche, ce qui réduit son temps de réponse.

Le croisement est considéré comme l'opérateur génétique le plus important. Son rôle consiste à recomposer les gènes d'individus de la population afin de balayer l'espace de solutions.

Le deuxième opérateur génétique (Mutation) est moins important par rapport au premier opérateur. Il sert à introduire des petites perturbations au niveau de la population dans le but d'éviter une convergence prématurée de l'algorithme génétique vers un ou plusieurs optimums locaux.

- **Algorithme général**

Les étapes des algorithmes génétiques peuvent varier d'un problème à un autre. Mais, ils se basent sur le même principe. La figure suivante, illustre le principe de base des algorithmes génétiques :

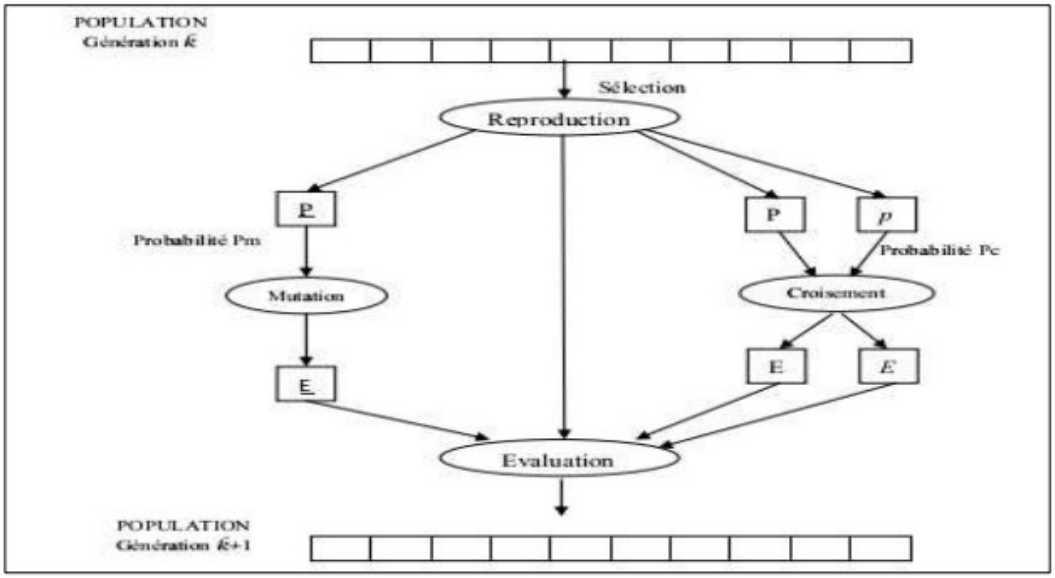


Figure 1. 3 : Principe général des algorithmes génétiques [4].

L'algorithme suivant résume le principe de base de l'algorithme génétique :

Début

Génération d'une population initiale de *k* individus ;

Répéter

1. Evaluation de la fonction objectif *f* de chaque individu ;
2. Sélection des meilleurs individus ;
3. Croisement entre deux individus sélectionnés : génération de deux enfants issus de deux parents sélectionnés ;
4. Mutation : transformation aléatoire des gènes de certains individus de la population ;

Jusqu'à la satisfaction de la condition d'arrêt.

Retourner la meilleure solution.

Fin

Figure 1.4 : Algorithme générale de l'algorithme Génétiques. [4]

7. Les opérateurs des algorithmes génétiques : [8]

7.1 L'opérateur de sélection :

L'idée de la phase de sélection est de sélectionner les individus les plus aptes et de les laisser transmettre leurs gènes à la génération suivante. Deux paires d'individus (parents) sont sélectionnés en fonction de leurs scores de condition physique. Les individus ayant une bonne condition physique ont plus de chances d'être sélectionnés pour la reproduction. [4]

Il y a des techniques de sélection nous citons :

- **Sélection par rang** : on va ranger les individus à partir de leur scores et on va choisir les individus qui possèdent les meilleurs scores d'adaptation.
- **Probabilité de sélection proportionnelle à l'adaptation** : c'est la technique de la roulette ou roue de la fortune, en effet la probabilité de choisir chaque individu est liée son adaptation au problème.

individu	Fonction objectif	pourcentage
1	30	39%
2	5	7%
3	16	21%
4	10	13%
5	15	15%

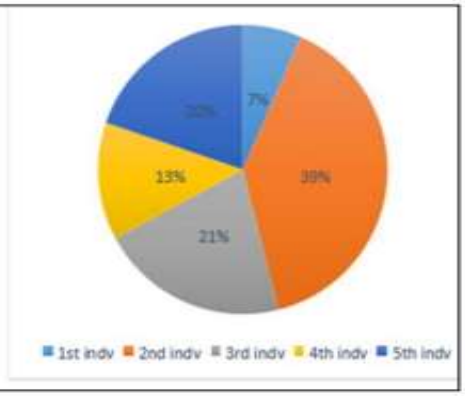


Figure 1.5 : Probabilité de sélection proportionnelle à l'adaptation. [8]

- **Sélection par tournoi :** on va sélectionner proportionnellement sur des paires d'individus, après on choisit parmi ces paires l'individu qui a la meilleur score d'adaptation.

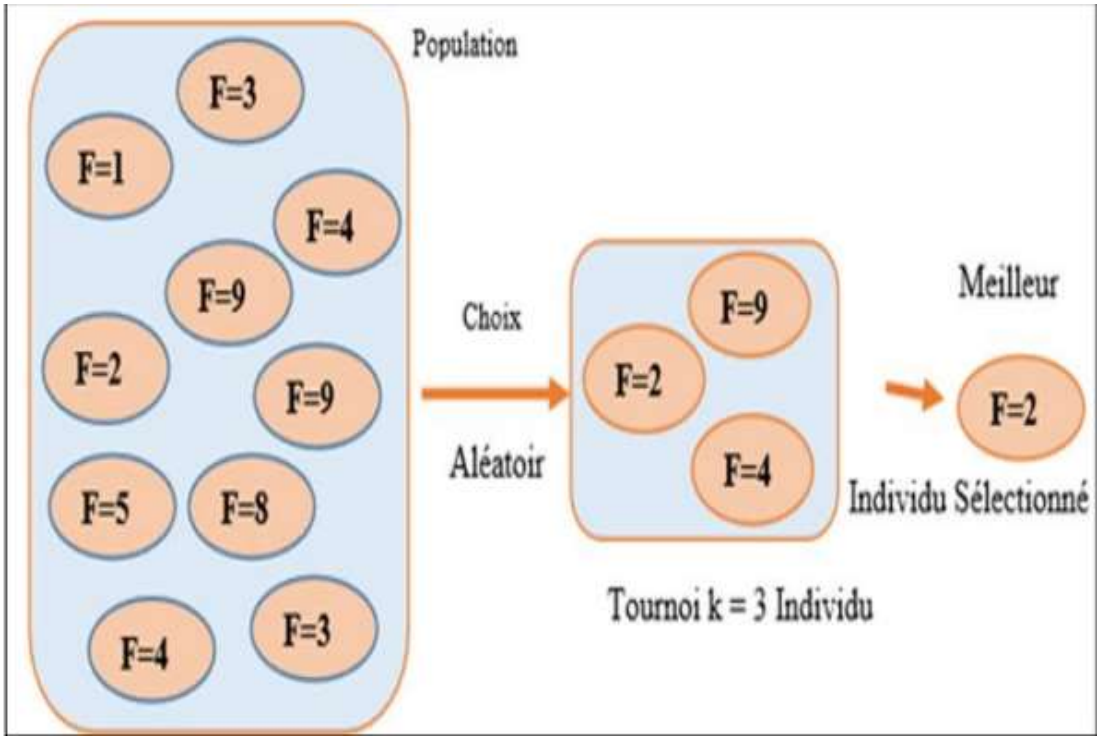


Figure 1.6 : Sélection par tournoi.

- **Sélection uniforme** : on va sélectionner aléatoirement, uniformément sans tenir compte de la valeur d'adaptation.

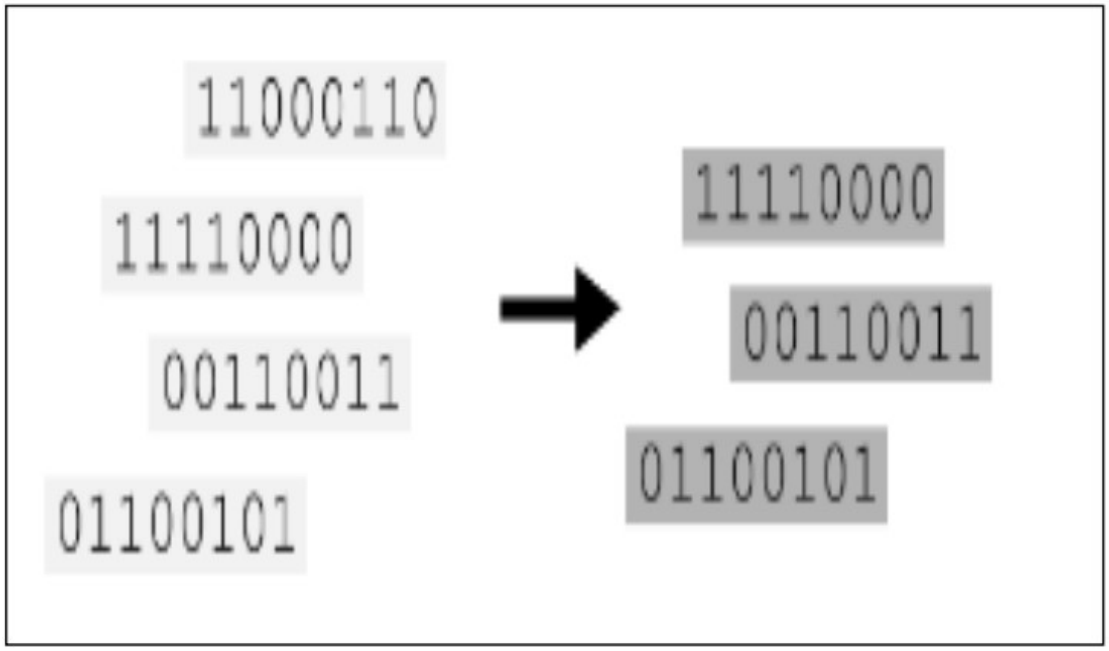


Figure 1.7 : individus en représentation binaire une fois la sélection effectuée. [8]

7 .2. L'opérateur de croisement : Le croisement est la phase la plus importante d'un algorithme génétique. Pour chaque paire de parents à accoupler, un point de croisement est choisi au hasard à l'intérieur des gènes. [4]

- **Simple croisement (un seul point):**

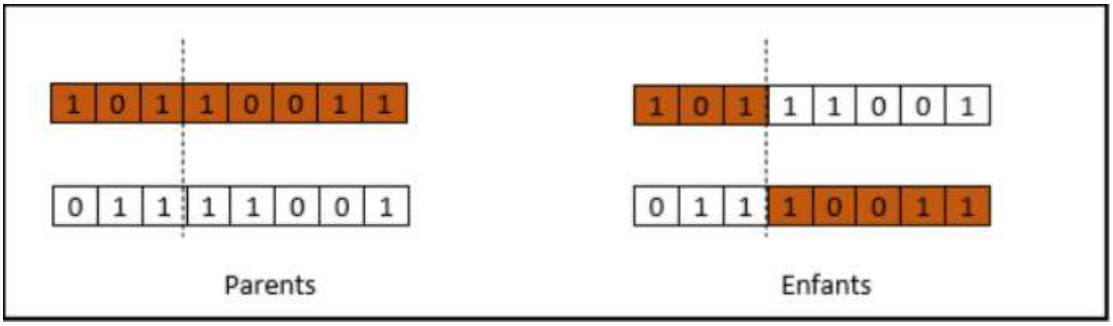


Figure 1.8 : croisement sur une seul point.[6]

- **Double croisement (deux points) :**

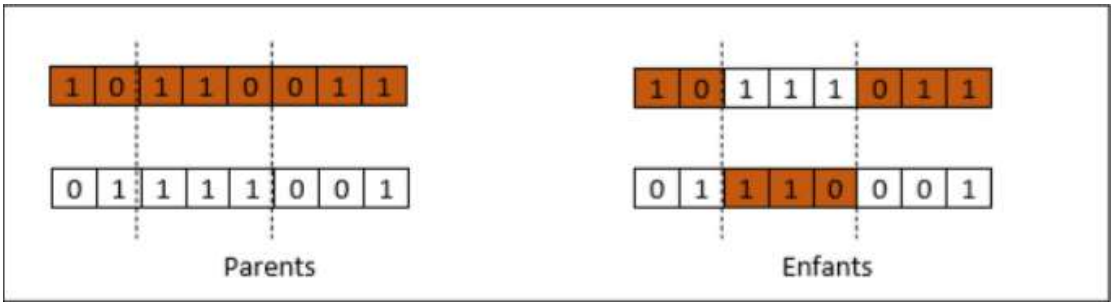


Figure 1.9 : croisement sur deux points. [6]

7.3. L'opérateur de mutation :

Cet opérateur tire au hasard un ou plusieurs gênes d'un individu, puis les modifie Aléatoirement à condition que cet individu reste une solution possible au problème. Chaque bit d'un individu a une probabilité Pm de subir une mutation, un réel $r \in \{0,1\}$ est Généré, si r < Pm le bit sera donc remplacé par complémentaire. [5]

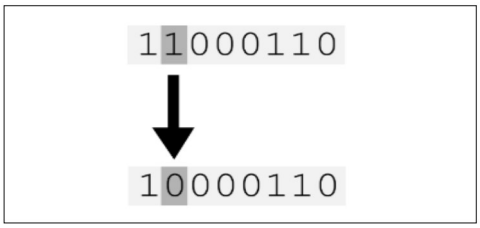
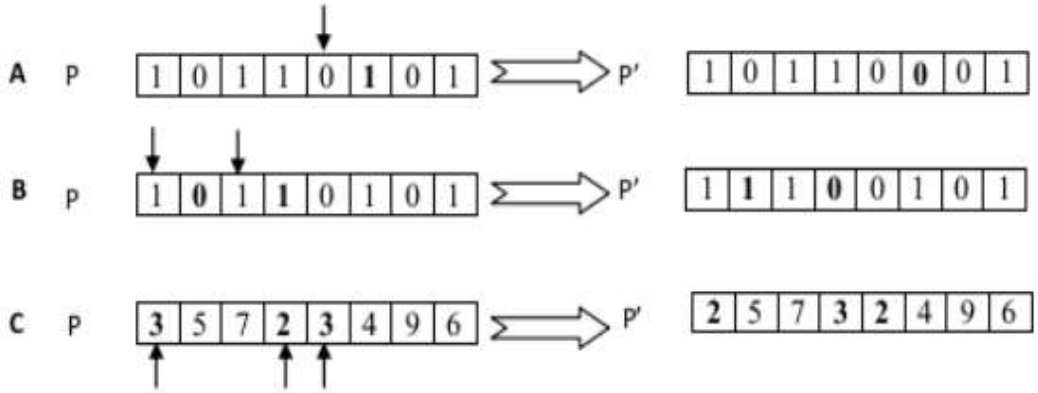


Figure 1.10 : Représentation d'opérateur de la mutation**. [8]**

On trouve plusieurs stratégies de mutation [2] :

- **La mutation uni-point** Ce type de mutation se fait par altération d'une seule valeur sur le chromosome.
- **La mutation bipoints et multipoints** Cette mutation se fait par altération de plusieurs valeurs sur le chromosome.
- **La mutation par valeurs** La mutation par valeur se fait par transformation d'une valeur donnée en une autre valeur déterminée, sur tous les gènes du chromosome.



A : Mutation uni-point. B : Mutation bipoints. C : Mutation par valeurs 3 et 2 Exemple d'opérateurs de mutation.

Figure 1.11 : Exemple d'opérateurs de mutation. [8]

.

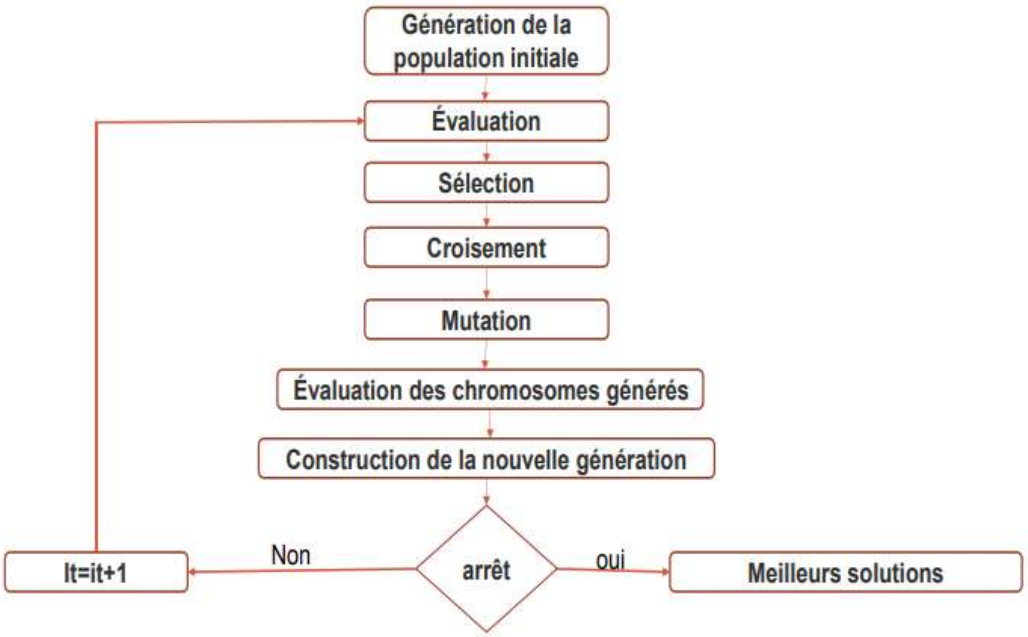


Figure 1.12 : organigramme générale de l'algorithme génétique implémenté. [8]

8. Les avantages des algorithmes génétiques [4]

Les algorithmes génétiques ont plusieurs avantages, citons :

- Ils sont adaptables à plusieurs types des problèmes.
- Faciles à paralléliser.
- Faciles à implémenter.
- Robustes.
- Faciles à hybrider.

9. Les inconvénients des algorithmes génétiques [4]

- ❖ Pas de garantie de convergence.
- ❖ Temps de calcul important (en cas où la taille de population est grande).

10. Conclusion

Dans ce chapitre, nous introduisons les concepts des algorithmes génétiques et leurs principes (Le codage des solutions «individus », la méthode de génération de la population initiale, la fonction à optimiser « fitness »), après nous sommes passés par les opérateurs dès l'algorithme génétique (sélection, croisement, mutation).

Chapitre II

Programmation Génétique

1. Introduction

Dans ce chapitre, nous présenterons un aperçu de la programmation génétique dans l'intelligence artificielle, qui est une technique dans laquelle un programme informatique est interprété comme un ensemble de gènes, puis modifié à l'aide d'un algorithme développé, généralement un algorithme génétique. Les chromosomes artificiels sont utilisés dans le codage des programmes informatiques. La programmation génétique est une application des algorithmes génétiques.

2. Définition

La programmation génétique est un algorithme évolutif caractérisé par la capacité à développer des expressions symboliques. Une structure peut être représentée comme une série d'une ou plusieurs flèches ou options avec un nombre limité d'actions. Par conséquent, les algorithmes et les programmes informatiques peuvent représenter la plupart des expressions mathématiques ou logiques. Sous forme personnelle. Ces individus sont regroupés en populations dites et évoluent de manière itérative à chaque itération, liés à la biologie appelée générations. La solution peut être symbolisée différemment dans des graphiques séparés, et même le code machine est utilisé comme représentation pour mieux s'adapter à différentes applications. Il s'agit d'un encodage arborescent. De manière générale, le développement de la programmation génétique nécessite la définition de quatre composantes principales : un ensemble primitif, une méthode d'évaluation, un ou plusieurs facteurs de variation et une sélection indicative. [9]

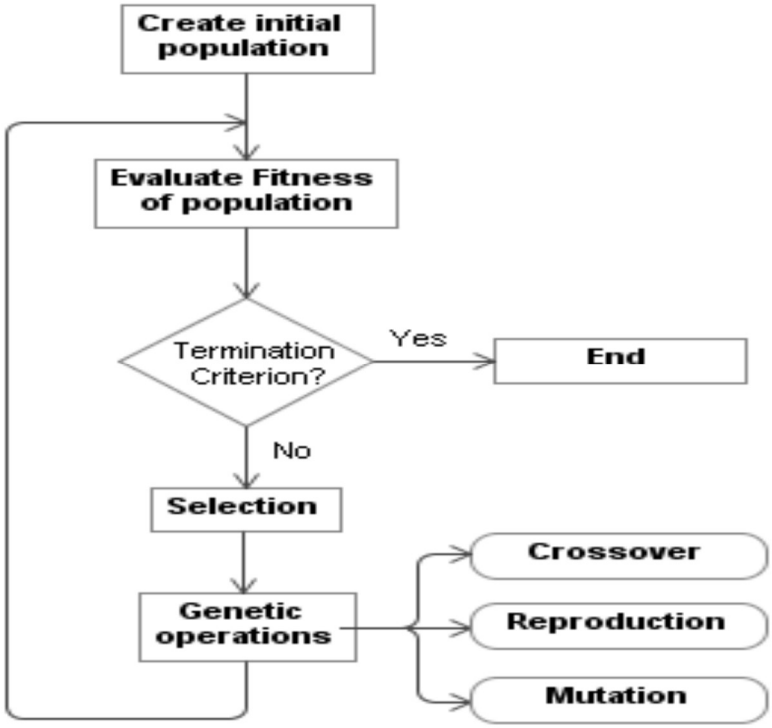


Figure 2.1 : Schéma de la programmation génétique.

3. Historique

Afin de bien comprendre d'où vient la programmation génétique, nous allons tout d'abord identifier quelques dates importantes pour cette recherche :

- 1958 – Friedberg : Mutation aléatoire d'instruction dans un programme génétique, attribution de « crédit » aux instructions des programmes les plus efficaces.
- 1963 – Samuel : Utilisation du terme « machine Learning » dans le sens de programmation automatique.
- 1966 – Fogel, Owen & Walsh : Automates à états finis pour des tâches de prédiction, obtenus par sélection de parents efficaces auxquels on applique des mutations : «evolutionary programming ».
- • 1985 – Cramer : Utilisation d'expression sous forme d'arbre. Cross-over entre sous-arbres.
- 1986 – Hicklin : Evolution de programmes de jeux en LISP. Sélection des parents efficaces, combinaisons des sous-arbres communs ou présents dans un des parents et de sous-arbres aléatoires.
- • 1989/1992 – Koza : Systématisation et démonstration de l'intérêt de cette approche pour de nombreux problems. Définitions d'un paradigme standard dans le livre « Génétique programming. On the programming of computers by means of natural selection » [Koza, 1992]. Ce paradigme inclus plusieurs concepts : programmation structurée en expression arborescentes, définition d'une grammaire de langage, type de retour unique pour chaque fonction, définition des proportions de mutation et de cross-over pour chaque génération, etc. [10]

4. Principe

- La programmation génétique travaille directement sur un codage de la solution du problème
- Les chromosomes deviennent alors une composition hiérarchique d'opérateurs et d'arguments codant le comportement des individus
- Il y a le même type d'opérateurs sur cette hiérarchie que sur les chaines de bits mais de nouveaux opérateurs peuvent être ajoutés aux opérateurs de sélection, croisement, mutation. [11]

5. Les étapes préparatoires de la programmation génétique

Il existe six étapes préliminaires pour résoudre un problème en utilisant la programmation génétique.

Ces étapes sont :

1. Le choix des terminaux.
2. Le choix des fonctions.
3. La fonction de performance.
4. Les paramètres du contrôle.
5. Le critère de terminaison et en fin déterminer.
6. En fin déterminer la structure du programme cette étape est optionnel.

- Les terminaux fonctions : sont les composants d'un programme informatique, ou les terminaux sont les nœuds externe (les feuille) dans l'arbre, et les fonctions sont les nœuds interne.
- La fonction de performance : dans cette étape on va déterminer la fonction d'adaptation (la fonction objective pour calculer le fitness de chaque programme), la détermination de cette fonction dépend à la nature du problème.
- Les paramètres du contrôle : il existe plusieurs paramètres du contrôle mais le plus important c'est déterminé le nombre de la population initiale.
- Le critère de terminaison : simplement c'est la condition d'arrêt du programme génétique. [11]

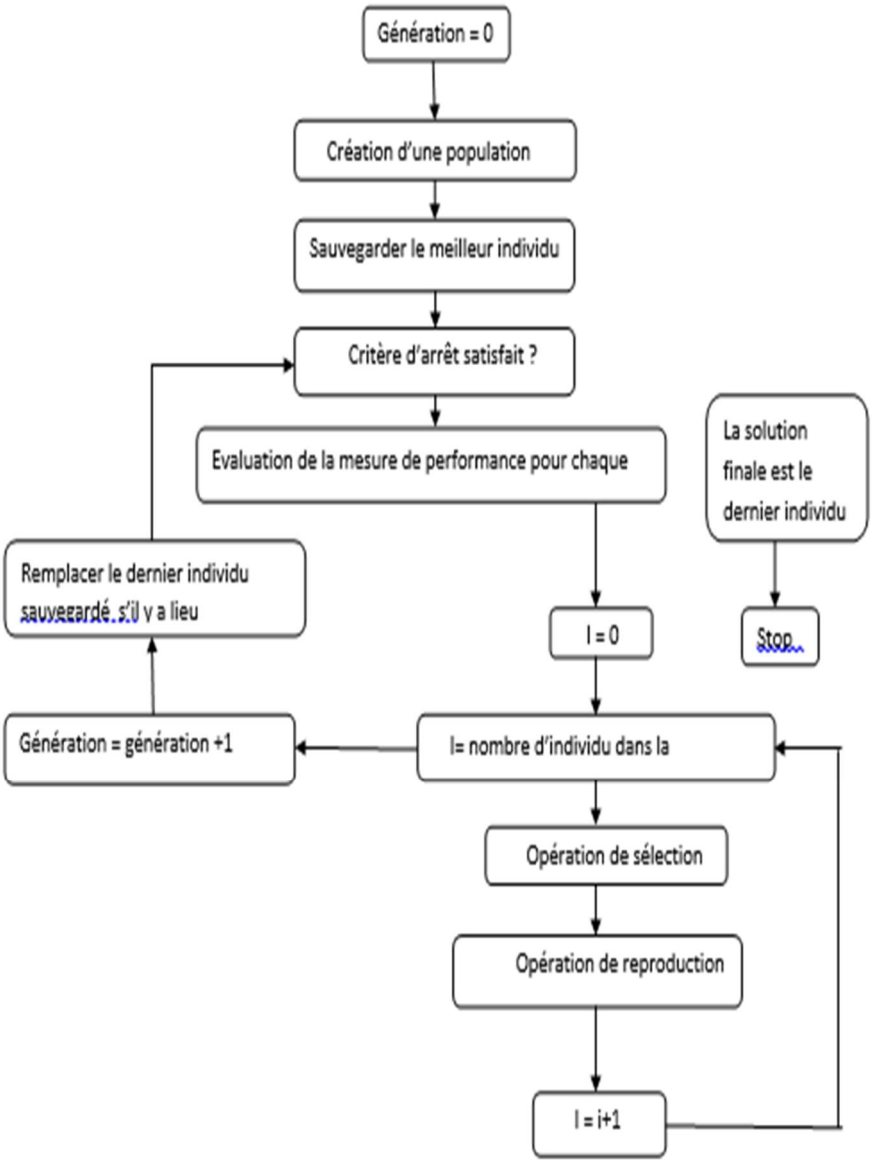


Figure 2.2 : Organigramme de la programmation génétique.[11]

6. Les phases de la programmation génétique

1. La première étape de la programmation génétique consiste à générer l'ensemble initial au hasard (un individu = un programme).
2. Utilisez la fonction fitness pour évaluer l'adéquation de chaque individu du groupe (évaluer l'adéquation du programme au problème à résoudre).
3. La fonction adaptative sélectionne les individus les mieux adaptés à leur environnement
4. Appliquez l'opérateur d'intersection pour modifier la population afin de créer une nouvelle population.

5. Répétez les étapes 2, 3 et 4 plusieurs fois, lorsque le niveau de performance souhaité est atteint, le processus évolutif s'arrête, ou le nombre de générations fixées passe sans améliorer l'individu. Plus efficace
6. Il existe trois principaux types de génération de programmes aléatoires lors de l'initialisation : la méthode d'incrémentation (croissance), la méthode complète (complète) et la méthode <demi-pente>. Les trois méthodes ont une profondeur maximale qui ne peut pas être dépassée et doit être spécifiée. à l'avance.
7. Après cela, un certain nombre d'individus doivent être créés.[11]

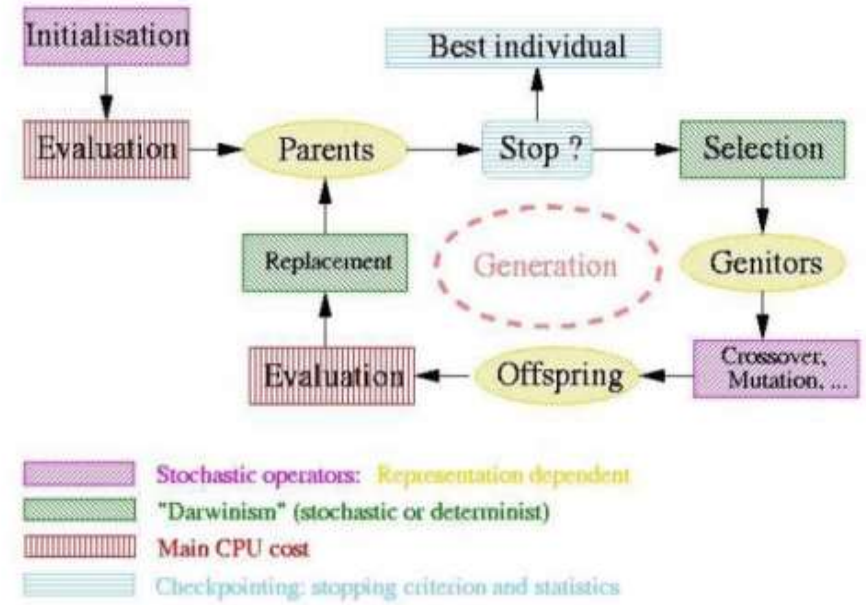


Figure 2.3 : Les phases de la programmation génétique.

7. Les opérateurs

7.1. L'opérateur de sélection [12]

On distingue deux catégories de sélection : la sélection déterministe et la sélection stochastique

7.1.1. Sélection déterministe : choisir le meilleur individu (au sens d'une fonction de performance), mais cela crée des problèmes de temps de calcul dans les grandes populations. Les individus les moins

efficaces sont entièrement éliminés de la population et le meilleur individu est toujours sélectionné.

7.1.2. Sélection stochastique : privilégiant toujours les meilleurs, donnant une chance aux moins performants. Il peut y avoir des situations où le meilleur individu n'est pas sélectionné et aucun enfant n'obtient d'aussi bons résultats que le meilleur parent.

7.1.3. Les différentes méthodes de sélection : on distingue plusieurs méthodes de sélection :

7.1.4. Sélection par tournoi : Cette méthode n'utilise que des comparaisons entre individus et n'a même pas besoin de classer la population en fonction du résultat de la fonction d'évaluation de chaque individu.

7.1.5. Sélection de Roulette : C'est la méthode la plus couramment utilisée. Les choix individuels des parents sont proportionnels à leurs performances. Les plus performants ont une forte probabilité d'être sélectionnés. Le nombre de fois qu'un individu est sélectionné est égal à sa note divisée par la note moyenne de la population totale. Cette méthode de sélection peut être assimilée à une roue de jeu sur laquelle tous les individus sont placés, et la largeur attribuée à chaque individu est liée à leur valeur d'évaluation. Ensuite, la balle est lancée et repose sur une personne, voir photo...

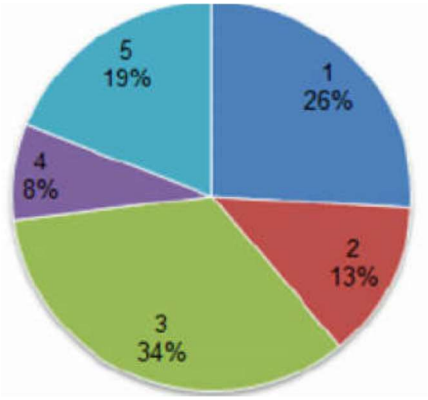


Figure 2.4 : L'espace réservé par chaque programme dans la roulette. {11]

7.2. Les opérateurs génétiques

A l'aide de ces opérateurs génétiques on peut créent de nouveaux individus au cours de l'évolution. On trouve généralement des opérateurs de croisement et de mutation. L'opérateur de croisement combine deux individus parents pour créer un ou plusieurs nouveaux individus enfants avec des caractéristiques proches de l'individu parent, tandis que l'opérateur de mutation modifie l'individu parent pour créer un individu enfant.

7.2.1 Croisement

L'opérateur de croisement est généralement des opérateurs binaires. Il associe deux individus appelés parents pour donner deux nouveaux individus appelés enfants. Cet opérateur est adapté à la programmation génétique, cat il utilise la recombinaison de sous-barbes pour combiner le patrimoine génétique des programmes. Une intersection (un nœud) est sélectionnée au hasard dans les deux programmes indépendamment. Le sous-programme sera créé à partir d'une copie de l'un des parents, mais la sous-barbe est remplacée à l'intersection par une copie de la sous-barbe du second parent.

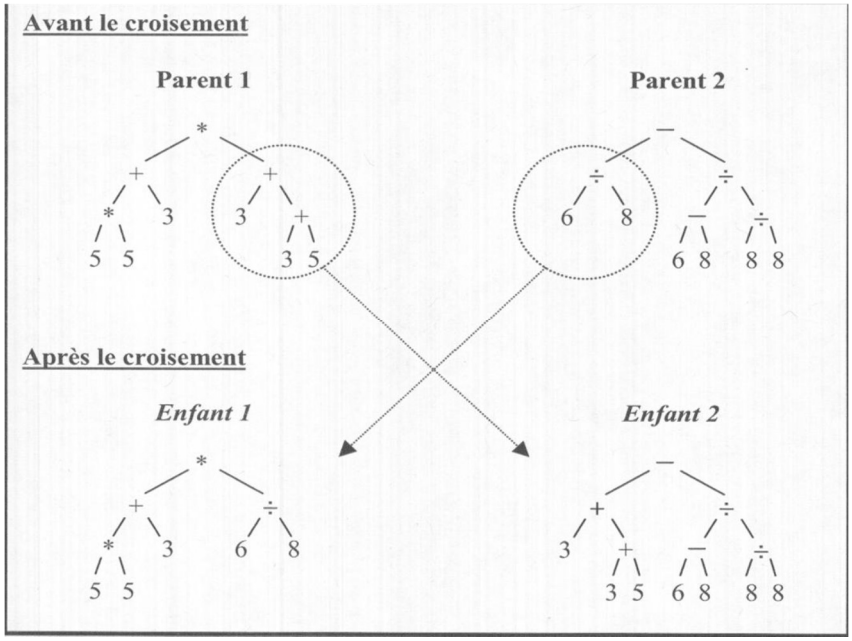


Figure 2.5 : Exemple de croisement entre deux arbres.

7.2.2 Mutation

La mutation est un opérateur qui, comme le croisement, s'inspire des algorithmes évolutionnaires, notamment dans le domaine des algorithmes génétiques. Contrairement au croisement, qui permet à deux individus de se recombiner, les mutations agissent sur un seul individu et « essaient » de lui infliger un minimum de dommages. [13]

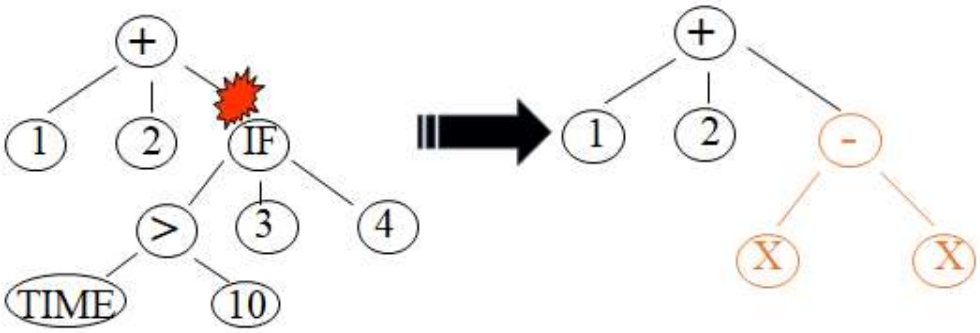


Figure 2.6 : Exemple d'un opérateur de mutation.[14]

8. Avantages de la PG sur les AG

- Approche évolutive similaire, mais PG ne se limite pas a priori aux chromosomes de longueur fixe.
- Les éléments dans les expressions peuvent avoir une complexité différente de celle des éléments dans les chaînes utilisées dans GA.
- Comme avec AG, la représentation chromosomique du problème n'est pas évidente et, si elle est mal faite, peut conduire à de mauvaises solutions. [14]

9. Conclusion

Ce chapitre fournit un résumé que, La programmation génétique s'applique également à l'évolution du matériel. Appliquer des algorithmes de programmation génétique pour résoudre des problèmes de régression symbolique. L'objectif est de trouver une fonction qui corresponde le plus possible aux données d'entrée et de sortie données.

Chapitre III

Etat de l’art

1. Introduction

Dans ce chapitre, on représente de

s articles des chercheures fissent des recherches en notre thème du mémoire, alors en résume chaque article comme suite :

(***2012)*** **Nitin S. Choubey** décrit l'approche de résolution du problème de labyrinthe avec l'algorithme génétique. Il comprend également un procédé de développement d'une structure de labyrinthe rectangulaire, La méthode est mise en œuvre et s'avère efficace pour les structures de labyrinthe avec différents niveaux de complexité de la taille 20 □ 20. Un labyrinthe est une zone bidimensionnelle en forme de grille de n'importe quelle taille, généralement rectangulaire. Un labyrinthe est composé de cellules. Une cellule est un élément de labyrinthe élémentaire, un espace formellement délimité, interprété comme un site unique. Le labyrinthe peut contenir différents obstacles dans n'importe quelle quantité. La complexité du labyrinthe est déterminée par le nombre de cellules, le nombre de murs/obstacles, le nombre de couloirs, les impasses et la distance entre la cellule départ-arrivée/départ-nourriture dans la structure du labyrinthe, Généralement, les labyrinthes nécessitent une séquence de décisions à prendre afin d'atteindre l'état d'objectif à partir de l'état initial. La structure de labyrinthe considérée dans cet article est un labyrinthe rectangulaire aléatoire construit en utilisant la séquence de décisions stochastiques prises au fil des itérations pour créer le même. L'article se .concentre également sur la solution du labyrinthe rectangulaire en utilisant l'algorithme génétique, une méthode heuristique évolutive pour trouver une solution optimale.

Les algorithmes génétiques sont de bonnes heuristiques qui ramènent près de la solution optimale. L'objectif de cet article est de fournir une méthode pour trouver un chemin à travers un labyrinthe en utilisant un algorithme génétique. La méthode pour générer la structure de labyrinthe rectangulaire aléatoire avec un labyrinthe.

(2016) Anton Jonasson, Simon Westerlind Dans ce rapport, les algorithmes génétiques ont été comparés à BFS et DFS pour résoudre labyrinthes en ce qui concerne le temps de calcul et la longueur du chemin de la solution. C'était fait en créant des labyrinthes avec l'algorithme de Wilson et en exécutant plusieurs tests dans un environnement contrôlé. La longueur du chemin de la solution trouvée par l'algorithme

génétique était en général meilleur que le DFS et souvent presque aussi bon que celui trouvé par le BFS. Le temps de calcul nécessaire à l'algorithme génétique cependant était nettement pire que le temps nécessaire à la fois au DFS et au BFS. Avec l'utilisation accrue des métadonnées, nous trouvons des problèmes plus complexes contenant de plus grandes quantités de données. Données que jamais auparavant. De nombreuses recherches ont été menées sur l'utilisation des GA pour résoudre problèmes d'optimisation, peut être utilisé pour trouver des chemins à partir de n'importe quel point dans des labyrinthes n'importe quel point dans des labyrinthes de différentes tailles et la complexité à un objectif fixé, mais le chemin le plus court n'a pas toujours été trouvé. Les labyrinthes sont étroitement liés à la théorie moderne des graphes, puisqu'un labyrinthe peut être facilement représenté par un graphe où chaque sommet représente une cellule du labyrinthe, et chaque bord représente une connexion entre les cellules.

C'est là que ce rapport entend contribuer, en rendant comparaisons entre GAs et BFS et DFS en ce qui concerne la résolution de labyrinthes. Ce est notre aspiration que l'AG montrera pour compléter le BFS et le DFS en étant plus rapide que le BFS tout en construisant une meilleure solution que le DFS. Les tests porteront sur le temps de calcul et la durée de la solution, et puisque l'AG n'est pas garanti de trouver une solution, le nombre de fois où il échoue sera également pris en compte. C'est là que ce rapport entend contribuer, en rendant comparaisons entre GAs et BFS et DFS en ce qui concerne la résolution de labyrinthes. Ce est notre aspiration que l'AG montrera pour compléter le BFS et le DFS en étant plus rapide que le BFS tout en construisant une meilleure solution que le DFS. Les tests porteront sur le temps de calcul et la durée de la solution, et puisque l'AG n'est pas garanti de trouver une solution, le nombre de fois où il échoue sera également pris en compte.

Thomas Pasquier, Julien Erdogan Une étude de la sélection par croisement et de l'optimisation locale d'un algorithme génétique. En particulier nous étudierons comment améliorer les performances d'un algorithme génétique pour un problème de résolution de labyrinthe et étudierons l'influence de différents facteurs, croisement et mutation, sur la diversité et les performances. Nous essaierons également de déterminer les paramètres qui donnent les meilleurs résultats.

(2006) Yao Zhou A Thesis L'étude de cette thèse est à la fois théorique et appliquée. Sur la théorie côté, une amélioration GA Évolution Direction Guidé GA (EDG-GA) est proposée sur la base sur l'analyse de la théorie des schémas et de l'hypothèse des blocs de construction. De plus, une méthode est développé pour étudier la structure de l'espace des solutions GA en caractérisant les interactions entre les gènes. Cette méthode est en outre utilisée pour déterminer les points de croisement pour les croisement. Côté application, GA est appliqué pour générer une tolérance optimale plans d'affectation pour une série de processus de fabrication. On montre que l'optimum le plan d'affectation des tolérances réalisé par GA est meilleur que celui réalisé par d'autres des méthodes d'optimisation telles que l'analyse de sensibilité, à temps de calcul comparable.

HANNACHE Aboubakr , EMMOUIS Abdelhamid nous avons utilisé les Algorithmes Génétiques dont nous avons pu développer un algorithme qui nous a permet à résoudre le problème dans des temps très faibles.

(2020)DAHMANE Lamia et HADROUG Selma la présentation générale mais maintenait les algorithmes génétiques, et voila nous leurs principes (Le principe de variation, Le principe d'adaptation, Le principe d'hérédité), après nous sommes passés par les opérateurs des algorithmes génétiques (sélection, croisement, mutation), Ensuite, nous avons examiné les algorithmes génétiques parallèles et certains de leurs modèles, et encore leurs mécanismes de fonctionnement.

(2020) Matteo Iovino_z, Jonathan Styrudyz, Pietro Falco_ and Christian Smith Les applications industrielles modernes nécessitent des robots pour pouvoir fonctionner dans des environnements imprévisibles et des programmes être créé avec un minimum d'effort, car il peut y avoir de fréquents modifications apportées à la tâche. Dans cet article, nous montrons que la génétique la programmation peut être utilisée efficacement pour apprendre la structure de un arbre de comportement (BT) pour résoudre une tâche robotique de manière imprévisible environnement. De plus, nous proposons d'utiliser un simulateur simple pour l'apprentissage et démontrer que les BT apprises peuvent résoudre la même tâche dans un simulateur réaliste, atteignant la convergence sans avoir besoin d'heuristiques spécifiques à la tâche. Le savant la solution est tolérante aux défauts, ce qui rend notre méthode attrayante pour de vraies

applications robotiques. Termes de l'index Arbres de comportement, programmation génétique, mobile Manipulation.

(2014) Marc-André Gardner La programmation génétique est une approche d'optimisation hyper heuristique qui a été appliquée à unum large éventail de problèmes impliquant des représentations symboliques ou des structures de données complexes. Cependant, le procédé peut être gravement entravé par les ressources de calcul accrues requise set convergence prématurée causée par une croissance incontrôlée du code. Nous introduisons HARM-GP, une nouvelle approche d'égalisation d'opérateur qui façonne de manière adaptative la distribution de taille de génotype des individus afin de contrôler efficacement la croissance du code. Son caractère probabiliste minimisa Le surcoût sur le processus évolutif alors que sa formulation générique permet à cette approche détestent indépendants du problème et des opérateurs génétiques utilisés. Des résultats comparatifs sont fournis plus de douze problèmes avec des dynamiques différentes, et plus de neuf autres algorithmes tirés de la littérature. Ils montrent que HARM-GP est excellent pour contrôler la croissance du code tout en maintenant bonnes performances d'ensemble. Les résultats démontrent également l'efficacité de HARM-GP à limiter le surentraînement et le sur ajustement dans les problèmes d'apprentissage supervisé du monde réel.

(2001) Jens- Uwe Dolinsky L'étalonnage cinématique du robot est la condition essentielle pour une application réussie de la programmation hors ligne à la robotique industrielle. Pour compenser un robot imprécis positionnement de l'outil, les poses générées hors ligne doivent être corrigées à l'aide d'un outil calibré modèle cinématique, conduisant le robot aux poses désirées. Robot conventionnel les techniques d'étalonnage dépendent fortement des méthodes d'optimisation numérique pour estimation des paramètres du modèle. Cependant, les non-linéarités des équations cinématiques, para métrisations inappropriées du modèle avec d'éventuelles discontinuités de paramètres ou redondances, entraînent généralement une identification des paramètres mal conditionnée. Rechercher dans l'étalonnage de robots cinématiques s'est donc principalement concentré sur la recherche de modèles de robots et des méthodes numériques adaptées appropriées pour augmenter la précision de ces des modèles. Cette thèse présente une approche alternative à un robot cinématique classique calibration et développe une nouvelle méthode de calibration cinématique

statique inverse basée sur le récent paradigme de la programmation génétique. Dans cette méthode, le processus de robot l'étalonnage est entièrement automatisé en appliquant une régression de modèle symbolique au modèle synthèse (structure et paramètres) sans faire appel à des méthodes numériques itératives pour l'identification des paramètres, évitant ainsi leurs inconvénients tels que convergence, instabilité numérique et discontinuités de paramètres. L'approche développé dans ce travail est axé sur la conception évolutive et la mise en œuvre de programmes informatiques qui modélisent tous les effets d'erreur, en particulier les effets non géométriques telles que les erreurs de transmission de vitesse, qui affectent de manière significative la position globale précision d'un robot. La programmation génétique est utilisée pour tenir compte de ces effets et d'induire des modèles de correction conjointe utilisés pour compenser les erreurs de position. Lee potentiel de cette méthode portable est démontré dans des expériences d'étalonnage effectuées sur un robot industriel.

Chapitre 4 :

Simulation d'un robot mobile par programmation génétique.

1. Introduction

Dans ce chapitre, nous introduisons l'idée de résoudre le problème du labyrinthe à l'aide d'algorithmes génétiques.

2. Problématique

Dans ce projet, un robot se déplace dans un labyrinthe et s'arrête dès qu'il touche un mur, retourne à un endroit qu'il a déjà visité ou atteint la sortie. Le but est de trouver un itinéraire menant à la sortie. Nous utilisons la programmation génétique pour produire un comportement intelligent du robot dans une zone aléatoire. Nous devons le faire rechercher la meilleure configuration de paramètres pour améliorer l'approche Efficacité. Le travail doit être étendu au cas d'apprentissage non supervisé.

3. Formulation mathématique

3.1. Labyrinthes

est une technique utilisée pour créer une structure selon certaines règles ,est un système complexe de passages ou de chemins entre les murs conçu pour confondre les personnes essayant de s'y retrouver

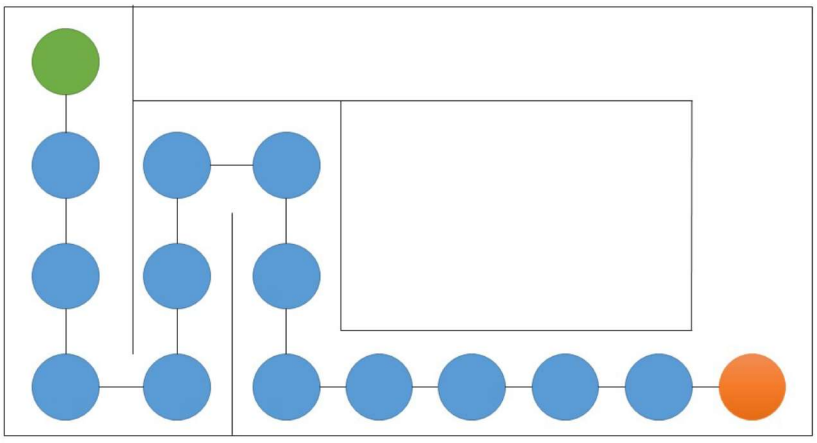


Figure 4.1 : Labyrinthe simple. [7]

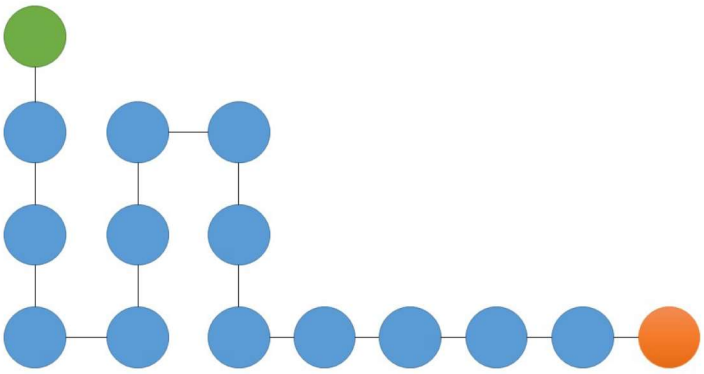


Figure 4.2 : graphe conceptualisé à partir du labyrinthe. [7]

3.2 Les règles

Le labyrinthe se compose de 4 types d'etats la route :

- la route.
- Le mur.
- Entrée du labyrinthe.
- Sortie du labyrinthe.

L'entrée est située dans le coin supérieur gauche et la sortie est située dans le coin inférieur droit, comme le montre l'image.

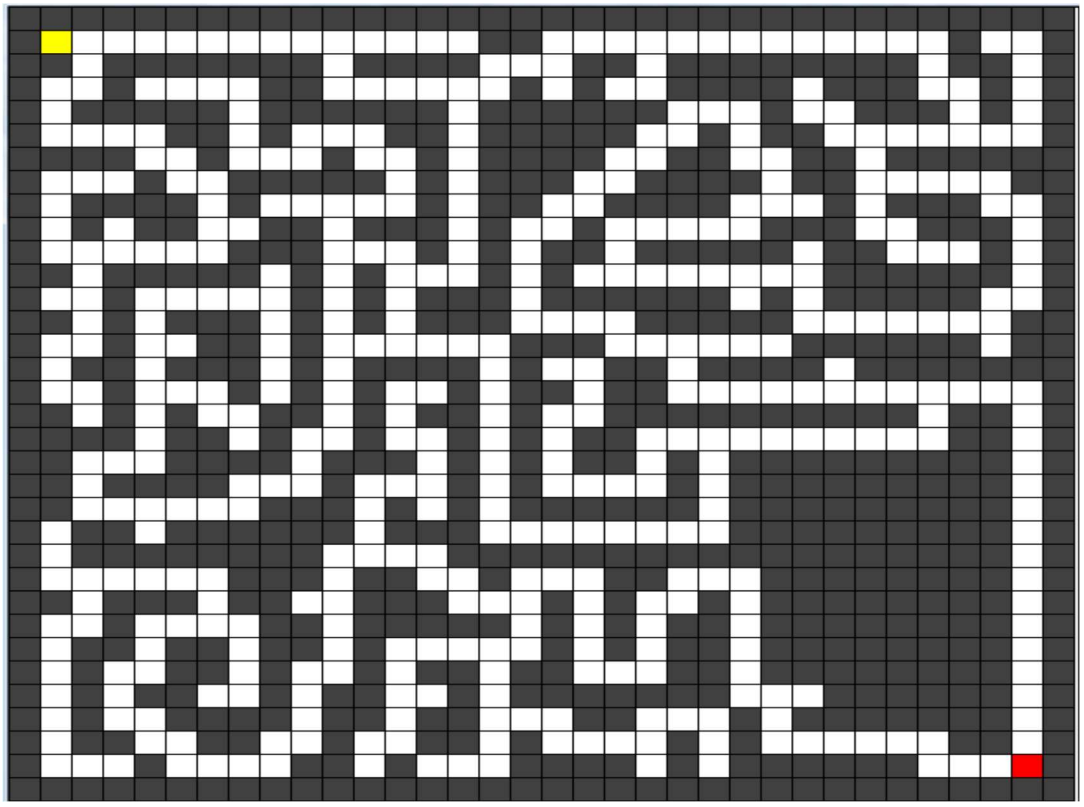


Figure. 4.3 : représentation d'un labyrinthe. [18]

4. Présentation physique

4.1. Les Données

Nous étudions un labyrinthe carré

- la taille : la longueur * la largeur.
- Obstacles : 0,1 (0 = franchissement, 1 = obstacle).
- les coordonnées du point d'entrée et du point de sortie**.**

Exemple :

- Nous prenons la longueur : 3, la largeur : 3, la taille = 3*3=9.
- 0 = franchissement, 1 = obstacle.
- Entrée (0.0), Sortie (2.2).

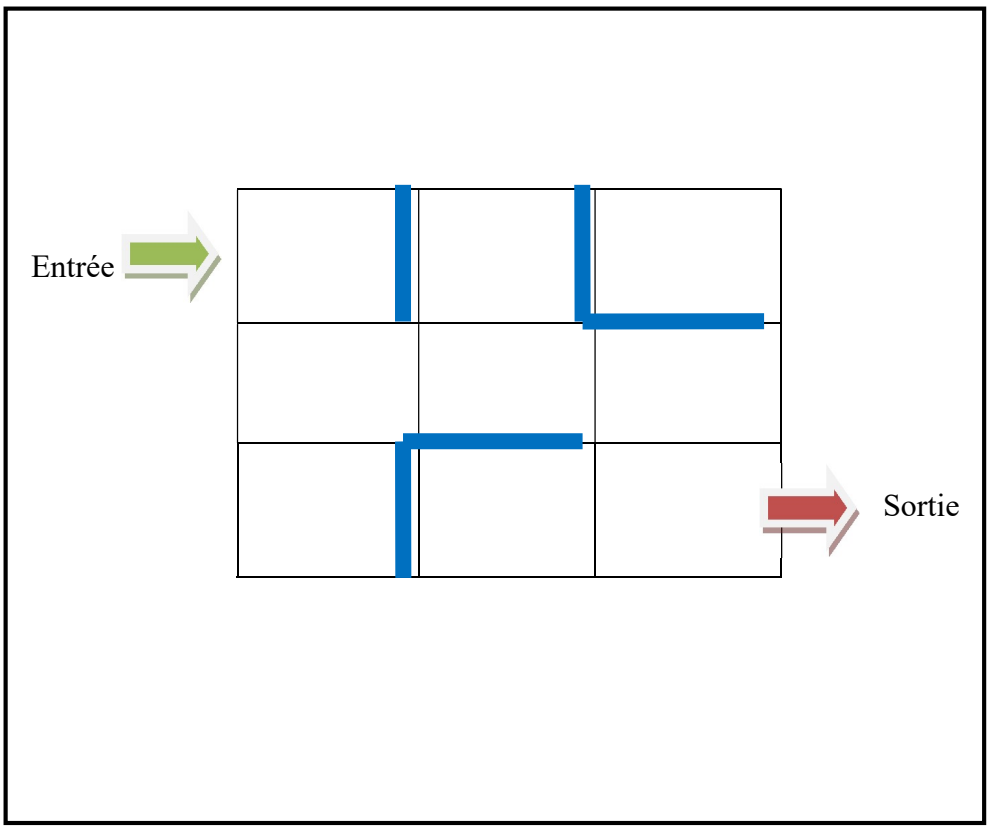


Figure 4.4 : forme de labyrinthe.

- Nous avons maintenant une perception de la forme du labyrinthe. Ce qui compte pour nous, c'est où le robot peut traverser et où il ne peut pas. Alors, où était l'emplacement du point d'entrée et l'emplacement du point de sortie. Il peut passer moins de 9 cases ou plus au cas où il y aurait des obstacles qui pourraient le pousser à revenir au même case qu'il a passé avant. Il s'agit d'essayer de trouver un meilleur point de passage pour atteindre la sortie finale

4.2. Les méthodes

Nous avons deux méthodes :

4.2.1. Méthode 1 : Tableau

1. On numérote les cellules du labyrinthe à partir de l'entrée

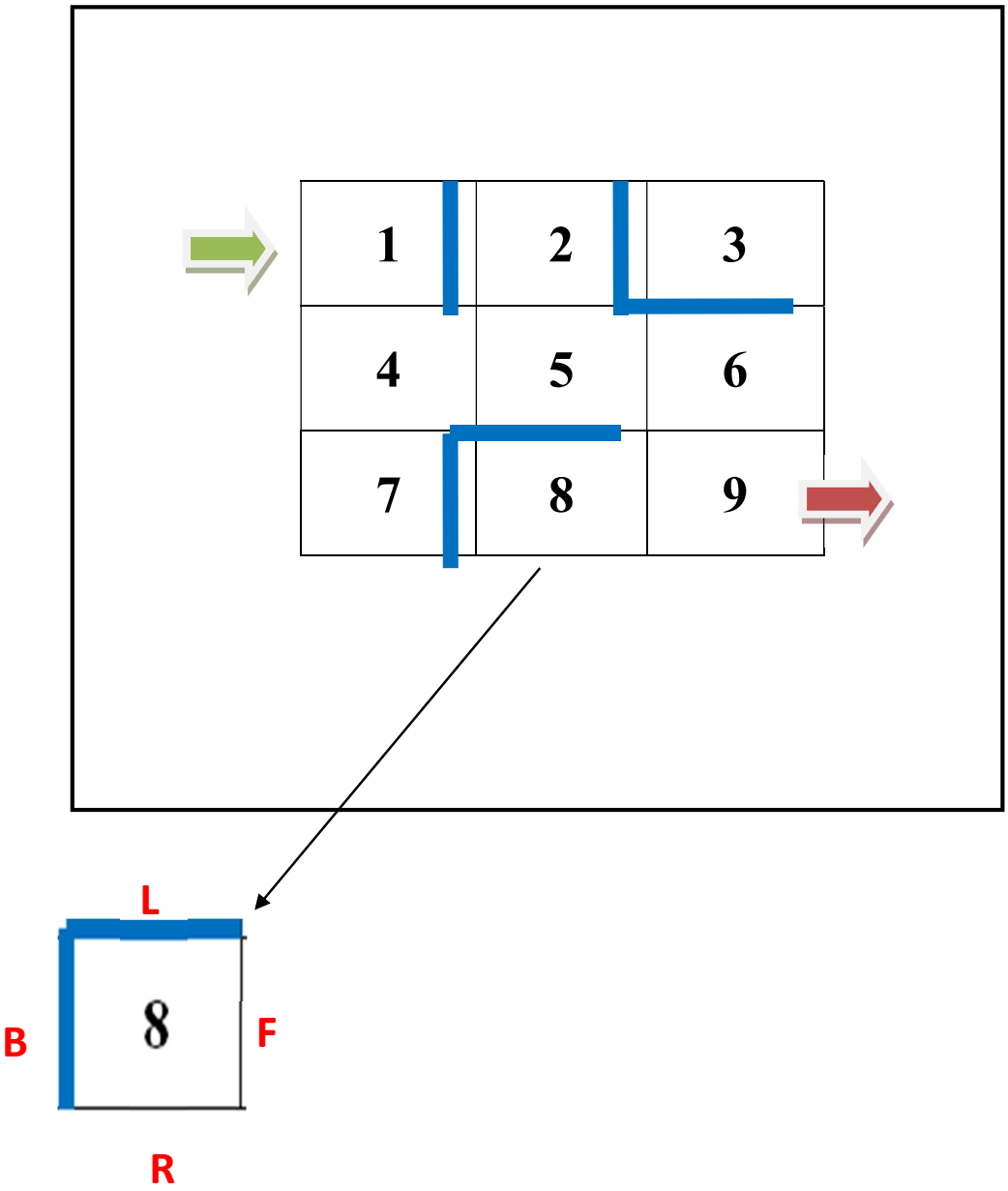


Figure 4.5 : Numérotation des bacs labyrinthe.

2. Nous créons un tableau composé de :
    - Nombre de lignes : Les directions.
    - Le nombre de colonnes. : Le nombre des cases de labyrinthe.

	1	2	3	4	5	6	7	8	9
R	0	0	1	0	1	0	1	1	1
L	1	1	1	0	0	1	0	1	0
F	1	1	1	0	0	1	1	0	0
B	0	1	1	1	0	0	1	1	0

Table 4.1 : Tableau des directions d'encodage des limites de chaque cellule du labyrinthe.

Cellule = 1 (obstacle).

Cellule = 0 (traversée).

Méthode 2: Matrice

Dans cette méthode, on imagine le labyrinthe comme un cube tridimensionnel dont les dimensions sont (i, j, k).

- i : nombre de ligne.
- j : le nombre de colonne.
- k : directions (bordures de case).
- m : Prendre soit un obstacle (1) soit une traversée (0).

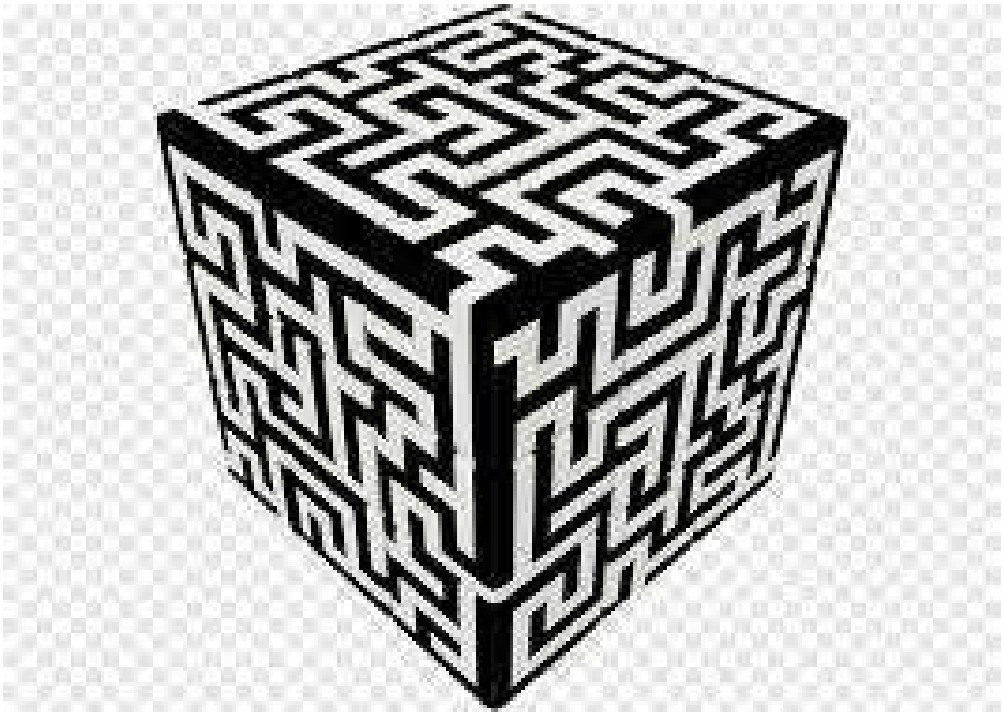


Figure 4.6 : labyrinthe 3D.

- Chaque case a 4 directions et afin de les implémenter dans la matrice, nous exprimons chaque direction avec un nombre comme indiqué :
  R = 1.
  L = 2.
  F = 3.
  B = 4.

- Chaque case est représentée comme suite :

$$\begin{cases} [i] & [j] & [R] & = [i] & [j] & [1] & = & m. \\ [i] & [j] & [L] & = [i] & [j] & [2] & = & m. \\ [i] & [j] & [F] & = [i] & [j] & [3] & = & m. \\ [i] & [j] & [B] & = [i] & [j] & [4] & = & m. \end{cases}$$

- Appliqué à Notre exemple, on trouve :

$$\text{Case 1} \quad \rightarrow \begin{cases} [0] & [0] & [1] & = & 0. \\ [0] & [0] & [2] & = & 1. \\ [0] & [0] & [3] & = & 1. \\ [0] & [0] & [4] & = & 0. \end{cases}$$

$$\text{Case 2} \rightarrow \begin{cases} [0] & [1] & [1] & = & 0. \\ [0] & [1] & [2] & = & 0. \\ [0] & [1] & [3] & = & 1. \\ [0] & [1] & [4] & = & 1. \end{cases}$$

$$\text{Case 3} \rightarrow \begin{cases} [0] & [2] & [1] & = & 1. \\ [0] & [2] & [2] & = & 0. \\ [0] & [2] & [3] & = & 0. \\ [0] & [2] & [4] & = & 1. \end{cases}$$

$$\text{Case 4} \rightarrow \begin{cases} [1] & [0] & [1] & = & 0. \\ [1] & [0] & [2] & = & 0. \\ [1] & [0] & [3] & = & 0. \\ [1] & [0] & [4] & = & 1. \end{cases}$$

$$\text{Case 5} \rightarrow \begin{cases} [1] & [1] & [1] & = & 1. \\ [1] & [1] & [2] & = & 0. \\ [1] & [1] & [3] & = & 0. \\ [1] & [1] & [4] & = & 0. \end{cases}$$

$$\text{Case 6} \rightarrow \begin{cases} [1] & [2] & [1] & = & 0. \\ [1] & [2] & [2] & = & 1. \\ [1] & [2] & [3] & = & 1. \\ [1] & [2] & [4] & = & 0. \end{cases}$$

$$\text{Case 7} \rightarrow \begin{cases} [2] & [0] & [1] & = & 1. \\ [2] & [0] & [2] & = & 0. \\ [2] & [0] & [3] & = & 1. \\ [2] & [0] & [4] & = & 1. \end{cases}$$

$$\text{Case 8} \rightarrow \begin{cases} [2] & [1] & [1] & = & 1. \\ [2] & [1] & [2] & = & 1. \\ [2] & [1] & [3] & = & 0. \\ [2] & [1] & [4] & = & 1. \end{cases}$$

$$\text{Case 9} \rightarrow \begin{cases} [2] & [2] & [1] & = & 1. \\ [2] & [2] & [2] & = & 0. \\ [2] & [2] & [3] & = & 0. \\ [2] & [2] & [4] & = & 0. \end{cases}$$

5. Présentation de solution

- **Labyrinthe** : est un graphe alors la solution est un chemin.

5.1 La solution de labyrinthe : est un suite de cas (1,….,9), ou suite de direction (R, L, F, B).

Les algorithmes

L'algorithme de génération de population initiale comme suite :

```
Sub inialisation()
    Dim i, j, c As Integer
    ReDim pool(taille, n)
    For i = 1 To taille
      c = 0
      j = Int(n * Rnd()) + 1
      While c + pi(j) <= capacity
        pool(i, j) = 1
        c = c + pi(j)
        Do
          j = Int(n * Rnd()) + 1
        Loop Until pool(i, j) = 0
      End While
    Next
  End Sub
```

Algorithme 4.1 L'algorithme de génération de population initiale.

La fonction objective « fitness »

Fitness = distance (première cellule, dernière cellule)

F(s) = distance (fin cellule, début cellule)Tapez une équation ici.

$=\sqrt{(i_1 - i_2)^2 + (j_1 - j_2)^2}$

$= |i_1 - i_2| + |j_1 - j_2|$

Algorithme initialisation

L'algorithme de génération initialisation comme suite :

```
Sub inisialisation()
        Dim i, j, c As Integer
        ReDim pool(taille, n)
        For i = 1 To taille
            c = 0
            Do While True
                j = Int(Rnd() * n) + 1
                If pool(i, j) = 0 Then
                    If c + wi(j) > capacity Then Exit Do
                    pool(i, j) = 1
                    c += wi(j)
                End If
            Loop
        Next
    End Sub
```

Algorithme 4.2 Algorithme initialisation**.**

Algorithme de roulette

L'algorithme de génération de roulette comme suite :

```
Function roulette() As Integer
    Dim i As Integer = Int(Rnd() * n)
    Dim tirage As Double = Rnd() * sum_fitness
    Dim s As Double = 0
    While s <= tirage
        i += 1
        If i = n + 1 Then i = 1
        s += fitness(i)
    End While
    Return i
End Function
```

Algorithme 4.3 Algorithme de roulette**.**

Algorithme croisement

L'algorithme de génération de croisement comme suite :

```
Sub croisement()
    Dim j, m, x, z As Integer
    For m = 1 To taille - 1 Step 2
        If Rnd() <= pc Then
            z = 1 + Int(Rnd() * n)
            For j = z To n
                x = pool(m, j)
                pool(m, j) = pool(m + 1, j)
                pool(m + 1, j) = x
            Next
            If Not feasible(m) Then makefeasible(m)
            If Not feasible(m + 1) Then makefeasible(m + 1)
        End If
    Next
End Sub
```

Algorithme 4.4 Algorithme croisement**.**

Algorithme mutation

L'algorithme de génération de mutation comme suite :

```
Sub mutation()
    Dim i, randombit1, randombit2 As Integer
    For i = 1 To taille
        If Rnd() <= pm Then
            randombit1 = Int(n * Rnd()) + 1
            pool(i, randombit1) = 1 - pool(i, randombit1)
            Do
                randombit2 = Int(n * Rnd()) + 1
            Loop Until pool(i, randombit2) = pool(i, randombit1)
            pool(i, randombit2) = 1 - pool(i, randombit2)
            If Not feasible(i) Then makefeasible(i)
        End If
    Next
End Sub
```

Algorithme 4.5 Algorithme mutation**.**

Algorithme évaluation

L'algorithme de génération évaluation de comme suite :

```
Sub evaluation()
    Dim i, j As Integer, x As Single
    ReDim fitness(taille)
    sum_fitness = 0
    For i = 1 To taille
        If paretofitness.Checked Then
            fitness(i) = nbrdominate(i)
        Else
            fitness(i) = fobj(i)
        End If
        sum_fitness += fitness(i)
        x = fobj(i)
        If x > fopt Then
            fopt = x
            iteropt = iter
            For j = 1 To n : solution(j) = pool(i, j) : Next
        End If
    Next
End Sub
```

Algorithme 4.6 Algorithme évaluation.

6. **La méthodologie utilisée** : l 'AG produit des générations successives d'individus, calculer leur « fitness » à chaque étape et sélectionner le meilleur des eux .lorsque la condition de réalisation survient.

Labyrinthe : est un technique utilisée pour la création de structure de labyrinthe.la structure de labyrinthe est créée en utilisant le processus trois étapes suivante :

1. La création d'une grille rectangulaire de taille n*n avec porte au hasard .la structure est un cube tridimensionnelle comme dans la figure ci-dessous :

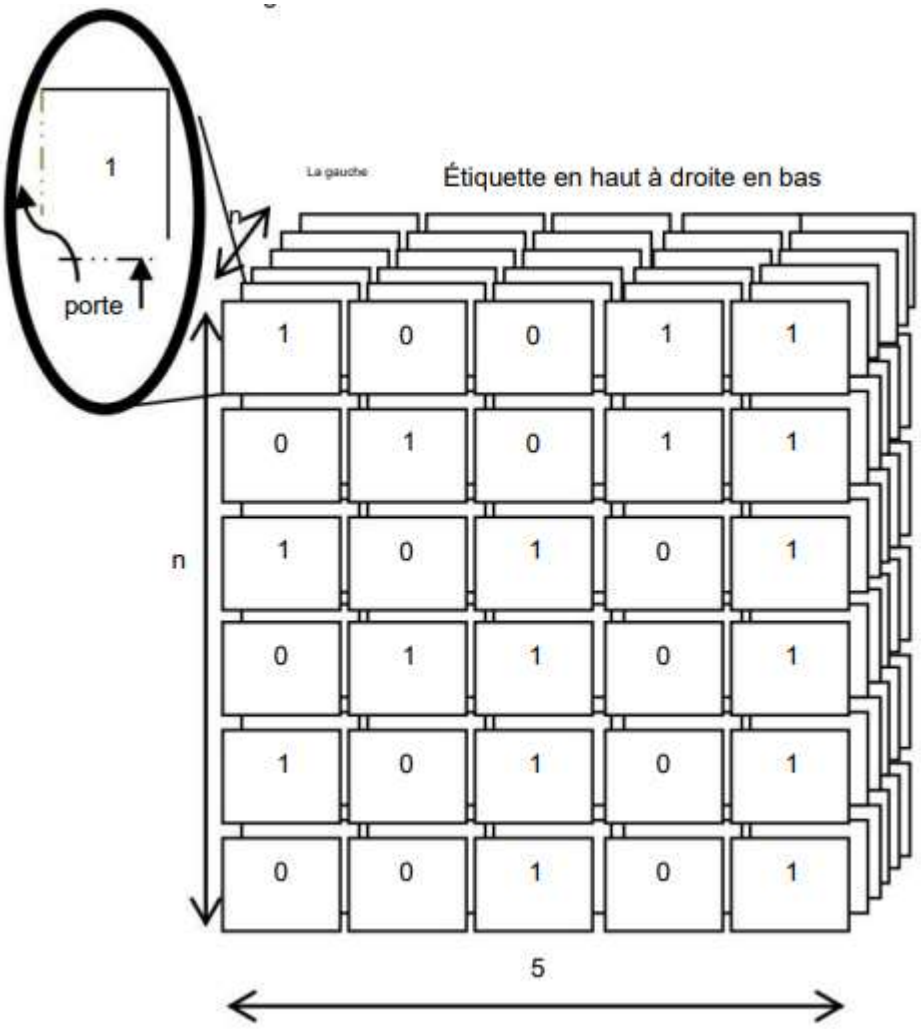


Figure 4.7 : Structure de données utilisée pour stocker le labyrinthe. [15]

2. Marquage des chemin :les cellules débuts(S)et fin(F)sont sélectionnes au hasard,0<=s , F<=n. l'étiquetage de la cellule est démarré avec l'étiquetage «1 » en utilisant l'algorithme Flood-Fill (S,0) pour chaque cellule connectée par porte(Valeur=1).si l'étiquetage est fait pour terminer la cellule (0,F)aussi, return (1),sinon utilisée Flood-Fill (F,n) avec étiquetage « 4.7 ».la figure montre le cas de return. [14]

1. Créez une structure nÿn, Maze M, contenant cinq cellules pour indiquer les portes Gauche, Haut, Droite et Bas avec la valeur de porte „0ÿ pour les murs.
2. Répétez l'étape 3 pour toutes les cellules M(i,j), à l'exception des cellules limites.
3. Répétez les étapes pour chaque valeur de Gauche, Haut, Droite et bas.
   un. La gauche :
      je. Générer un nombre aléatoire n(0,1)
      ii. Si(n==1), attribuez Gauche =1 (Créer porte) & Droite = 1 dans la cellule (i,j-1).
   b. À droite :
      je. Générer un nombre aléatoire n(0,1)
      ii. Si(n==1), attribuez Droiteÿ=1 (Créer une porte) et Gaucheÿ=ÿ1ÿpo la cellule (i,j+1).
   c. Haut :
      je. Générer un nombre aléatoire n(0,1)
      ii. Si(n==1), affectez Haut =1 (Créer porte) & Bas = 1 dans la cellule (i-1,j).
   ré. Bas :
      je. Générer un nombre aléatoire n(0,1)
      ii. Si(n==1), attribuez Bottom =1 (Create Door) & Top = 1 dans la cellule (i,j+1).
4. Testez la disponibilité du chemin à l'aide de la méthode d'étiquetage.

Figure 4.8 : Création d'une structure de labyrinthe aléatoire.

3. Rejoignez les étiquette : le suivi de l'étiquette se fait en utilisant 'algorithme Flood-Fil .si les étiquettes attribues ne correspondent pas l'assemblage des étiquettes est fait par le processus de rupture de mur. La figure suivante montre le cas de Return(1).

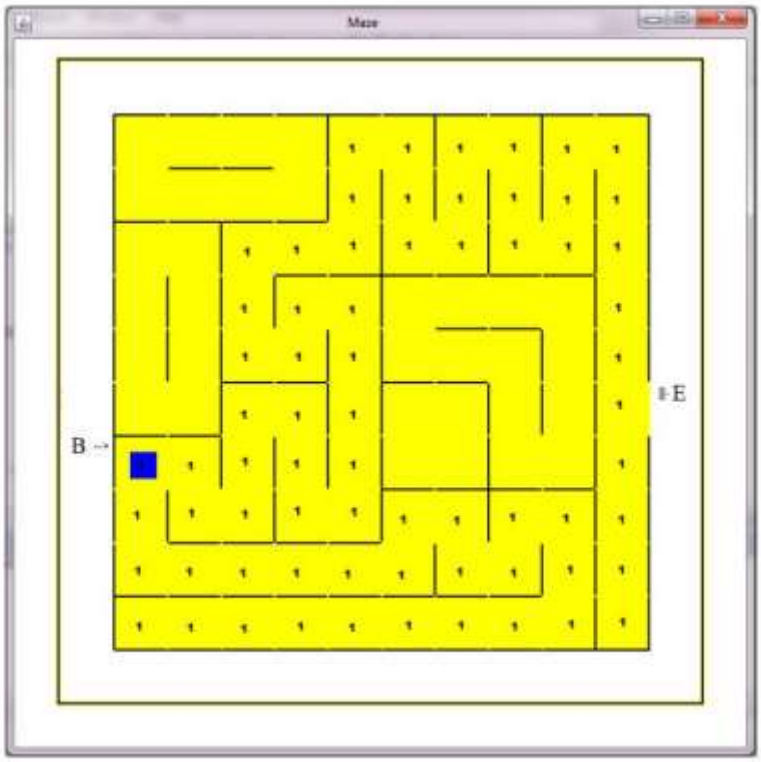


Figure 4.9 : Cas d'un labyrinthe aléatoire valide. [15]

7. Conclusion :

Dans ce chapitre, nous nous sommes efforcés d'étudier les méthodes les mieux adaptées au problème posé, et d'expliquer les deux méthodes proposées et de donner des exemples afin d'essayer de les appliquer au moyen d'algorithmes génétiques.

Chapitre 05 :

RESOLUTION DE PROBLEME

1. Introduction

Dans ce chapitre, nous avons bien expliqué notre travail et appliqué ce qui a été présenté dans les chapitres précédents.

2. Environnement matériel

Nous avons utilisé comme environnement matériel un ordinateurs DELL possède les caractéristique suivant :

- Processeur : Intel (R) Core (TM) i3 CPU M 350 @ 2.27GHz 2.27 GHz.
- RAM : 4.00 Go.
- Type De System : Système d'exploitation 64 bits.

3. Outils et environnement de développement

3.1 Langage JAVA :

Nous avons choisi Java car il est simple et plus formaté, il y a moins de lignes dans le code du programme, et chaque ligne est facile à expliquer à un programmeur novice sans difficulté.

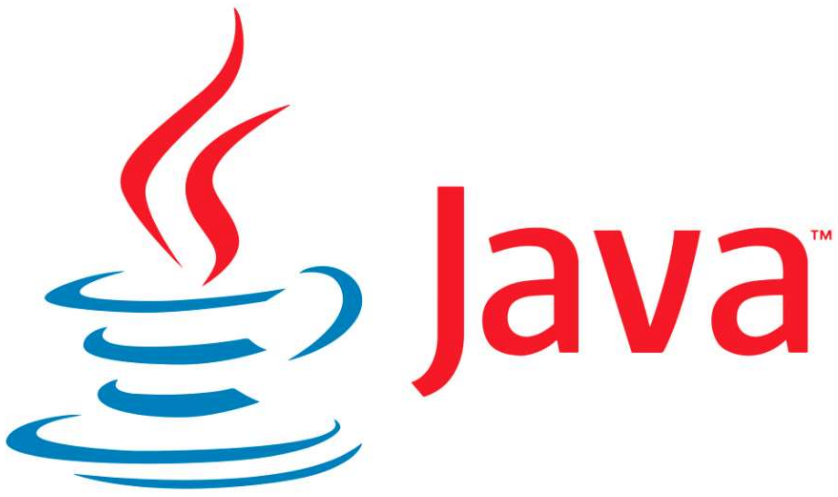


Figure 5.1: java.

3.2 Eclipse

Nous avons choisi Eclipse en raison de son interface principalement conviviale et de son système de composants internes étendu, car la diversité des composants internes facilite la création de votre propre espace de travail pour le développement d'applications Java. [17]
https://e3arabi.com/%D8%A7%D9%84%D8%AA%D9%82%D9%86%D9%8A%D8%A9/eclipse/

Figure 5.2: eclipse.

4. Le problème

Nous allons résoudre le problème en utilisant la méthode matricielle et en utilisant les capteurs du robot pour le guider afin qu'il puisse sortir avec succès du labyrinthe.

Le labyrinthe que nous allons explorer se compose de murs que le robot ne peut pas traverser et aura un chemin spécifique, comme illustré à **la figure 1.5** que nous voulons que le robot suive.

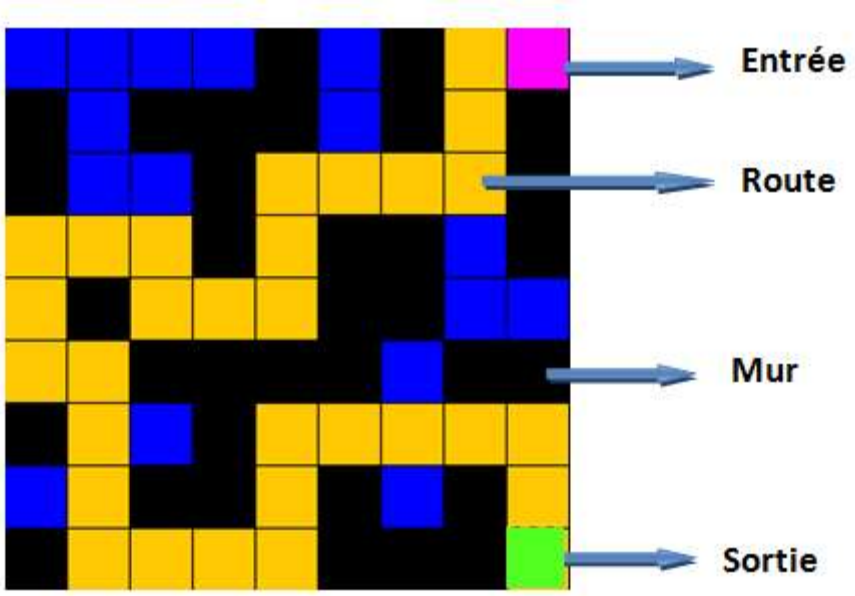


Figure 5.3 : L'itinéraire que nous voulons que le robot suive.

5. L'interface générale du notre programme

Dans notre programme, nous pouvons créer des labyrinthes et déjà progresser ou trouver un chemin, alors regardons les listes déroulantes, nous avons la possibilité de quitter l'application, nous avons la possibilité d'enregistrer le fichier créé avec un nom ou d'ouvrir un fichier précédemment enregistré. Le labyrinthe ou que nous partageons et notre panneau de défilement, nous avons la possibilité de créer de nouveaux panneaux qui laissent tout dans le port ou où cette application fournit également une visualisation des résultats précédemment appliqués en suivant les algorithmes et nous avons des options de recherche que nous avons la possibilité de définir le temps pris pour visiter chaque note. Ceci est vraiment utile lorsque vous voulez vraiment collecter les plus anciens.

Alors voyons comment nous pouvons créer un exemple de labyrinthe :

- ✓ un clic gauche, vous pouvez simplement créer l'état du mur
- ✓ clic droit vous pouvez sélectionner la cible ;
- ✓ Le clic du milieu de la souris permet de sélectionner le chemin de départ.

Lorsqu'ils sont activés, les éléments bleus sont représentés comme la recherche la plus ancienne et l'orange clignotant représente les notes recherchées ou le chemin emprunté par le robot.

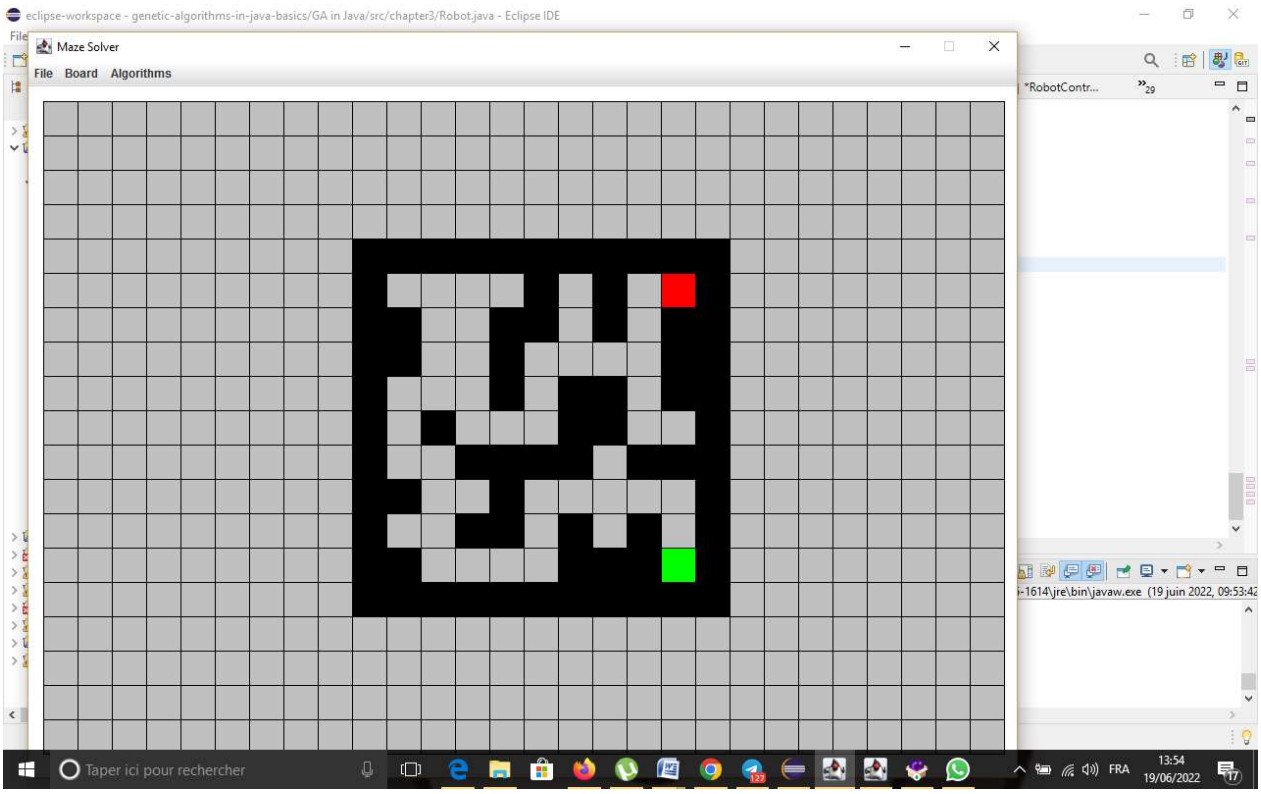


Figure 5.4 : interface générale.

6. Implémentation

Le chapitre est représentée sous forme de package dans le projet Eclipse qui l'accompagne. Le package contiens les classes suivantes:

1. class GeneticAlgorithm , qui résume l'algorithme génétique lui-même et fournit des implémentations spécifiques aux problèmes des méthodes d'interface, telles que le croisement, la mutation, l'évaluation de la condition physique et la vérification des conditions de terminaison.
2. class individuel, qui représente une seule solution candidate et son chromosome.
3. Une classe Population, qui représente une population ou une génération d'Individus, et leur applique des opérations au niveau du groupe.
4. class RobotController contenant la méthode 'main' .
5. class Maze, Cette classe résume un labyrinthe dans lequel un robot devra naviguer. Une abstraction robotique. nous lui donnons un labyrinthe et un jeu d'instructions
6. class Robot tenté de naviguer jusqu'à l'arrivée.

6.1 Codage

Le codage précis et correct des données est la partie la plus importante et la plus difficile de l'algorithme génétique, nous avons besoin d'une représentation binaire du jeu d'instructions complet du contrôleur de robot pour toutes les combinaisons possibles de contributions.

On a les Instructions Toutes les entrées possibles pour contrôler le robot :

- « 00 » : ne rien faire
- "01" : avancer
- "10" : tourner à gauche
- "11": tournez à droite

Nous avons également six capteurs marche/arrêt différents :

- Capteur #1 (avant) : activé
- Capteur #2 (avant-gauche) : désactivé
- Capteur #3 (avant-droit) : activé
- Capteur #4 (gauche) : désactivé
- Capteur #5 (droit) : désactivé
- Capteur #6 (arrière) : désactivé

  Nous donnant 2^6 (64) combinaisons possibles d'entrées de capteur. Si chaque action nécessite 2 bits pour être encodée, on peut représenter le réponse du contrôleur à toute entrée possible en 128 bits. Autrement dit, nous avons 64 scénarios différents dans les quels notre robot peut se trouver, et notre contrôleur doit avoir une action définie pour chaque scénario. Puisqu'une action nécessite deux bits, notre contrôleur nécessite

  64*2 = 128 bits de stockage.

Par souci de clarté, l'explication et la méthode de codage sont plus simples. Prenons un exemple d'instruction : tourner à gauche (action "10" telle que définie ci-dessus) avec 63 entrées supplémentaires pour représenter toutes les combinaisons possibles, ce format est peu pratique. Évidemment, ce genre d'énumération ne fonctionnera pas pour nous. Nous faisons donc un autre petit pas en raccourcissant tout et en traduisant "on" et "off" en 1 et 0 :

- #1 (avant) : 1
- #2 (avant-gauche) : 0
- # 3(avant-droit) : 1
- #4 (gauche) : 0
- #5 (droit) : 0
- #6 (arrière) : 0

  Instruction: 10

Alignons-les de droite à gauche et supprimons le mot "Instruction" de la sortie :
#6:0, #5:0, #4:0, #3:1, #2:0, #1:1 => 10
000101 => 10
Si nous convertissons maintenant la chaîne de bits des valeurs du capteur en décimal, nous obtenons ce qui suit :
5 => 10
Maintenant, nous sommes sur quelque chose. Le "5" sur le côté gauche représente les entrées du capteur, et le "10" sur la droite représente ce que le robot doit faire face à ces entrées (la sortie). Parce que nous sommes arrivés ici à partir d'une représentation binaire des entrées de capteur, il n'y a qu'une seule combinaison de capteurs qui peut nous donner le nombre 5.
Nous pouvons utiliser le nombre 5 comme position dans le chromosome qui représente une combinaison d'entrées de capteur. Si nous construisions ce chromosome à la main et que nous savions que "10" (tourner à gauche) est la bonne réponse à "5" (capteurs avant et avant droit détectant un mur), nous placerions "1" et "0" dans les 11e et 12e points du chromosome (chaque action nécessite 2 bits, et on commence à compter les positions à partir de 0), comme ceci :
xx xx xx xx xx 10 xx xx xx xx (…..54 autres paires……)

Sensors						Action	Chromosome
B	R	L	FR	FL	F		
0	0	0	0	0	0	11	**11**001001101001...11
0	0	0	0	0	1	00	11**00**1001101001...11
0	0	0	0	1	0	10	1100**10**01101001...11
0	0	0	0	1	1	01	110010**01**101001...11
0	0	0	1	0	0	10	11001001**10**1001...11
0	0	0	1	0	1	10	1100100110**10**01...11
0	0	0	1	1	0	01	110010011010**01**...11
1	1	1	1	1	1	11	11001001101001...**11**

Figure 5.5 : Mappage des valeurs des capteurs aux actions.

6.2 Initialisation

Dans cette implémentation, nous devons d'abord créer et configurer un labyrinthe pour exécuter un fichier, robot dedans. Pour ce faire, nous avons créé la classe Maze suivante pour la gestion des labyrinthes.

- ✓ **class Maze :**

```
package chapter3;

import java.util.ArrayList;

public class Maze {
	private final int maze[][];
	private int startPosition[] = { -1, -1 };

	public Maze(int maze[][]) {
		this.maze = maze;
	}

	public int[] getStartPosition() {
		// Check we already found start position
		if (this.startPosition[0] != -1 && this.startPosition[1] != -
1) {
			return this.startPosition;
		}

		// Default return value
		int startPosition[] = { 0, 0 };

		// Loop over rows
		for (int rowIndex = 0; rowIndex < this.maze.length;
rowIndex++) {
			// Loop over columns
			for (int colIndex = 0; colIndex <
this.maze[rowIndex].length; colIndex++) {
				// 2 is the type for start position
				if (this.maze[rowIndex][colIndex] == 2) {
					this.startPosition = new int[] { colIndex,
rowIndex };
					return new int[] { colIndex, rowIndex };
				}
			}
		}

		return startPosition;
	}


	public int getPositionValue(int x, int y) {
		if (x < 0 || y < 0 || x >= this.maze.length || y >=
this.maze[0].length) {
			return 1;
		}
		return this.maze[y][x];
```

```
	}


	public boolean isWall(int x, int y) {
		return (this.getPositionValue(x, y) == 1);
	}

	public int getMaxX() {
		return this.maze[0].length - 1;
	}

	public int getMaxY() {
		return this.maze.length - 1;
	}


	public int scoreRoute(ArrayList<int[]> route) {
		int score = 0;
		boolean visited[][] = new boolean[this.getMaxY() +
1][this.getMaxX() + 1];

		// Loop over route and score each move
		for (Object routeStep : route) {
			int step[] = (int[]) routeStep;
			if (this.maze[step[1]][step[0]] == 3 &&
visited[step[1]][step[0]] == false) {
				// Increase score for correct move
				score++;
				// Remove reward
				visited[step[1]][step[0]] = true;
			}
		}

		return score;
	}
}
```

Le labyrinthe représenté comme un tableau 2D d'entiers, avec un environnement différent

Types représentés par des nombres entiers comme suit :

0 = vide.
1 = Mur.
2 = position de départ.
3 = Itinéraire.
4 = position de but.

La méthode la plus significative de cette classe est `score Route`, qui renverra un score d'aptitude pour un chemin ; c'est ce score que l'algorithme génétique va optimiser.

Cette méthode inspecte une route donnée sous forme de tableau, et ajoute un point pour chaque bonne étape faite. Nous devons également faire attention

à ne pas récompenser la visite corriger les chemins, sinon vous pourriez obtenir un score infini juste en se tortillant aller-retour sur le parcours. [17]

✓ **Class individuel**

Initialise un individu aléatoire. Ce constructeur suppose que le chromosome est entièrement constitué de 0 et 1.

```
package chapter3;

public class Individual {
	private int[] chromosome;
	private double fitness = -1;
	public Individual(int[] chromosome) {
		// Create individual chromosome
		this.chromosome = chromosome;
	}


	public Individual(int chromosomeLength) {

		this.chromosome = new int[chromosomeLength];
		for (int gene = 0; gene < chromosomeLength; gene++) {
			if (0.5 < Math.random()) {
				this.setGene(gene, 1);
			} else {
				this.setGene(gene, 0);
			}
		}

	}


	public int[] getChromosome() {
		return this.chromosome;
	}

	public int getChromosomeLength() {
		return this.chromosome.length;
	}

	public void setGene(int offset, int gene) {
		this.chromosome[offset] = gene;
	}

	public int getGene(int offset) {
		return this.chromosome[offset];
	}

	public void setFitness(double fitness) {
		this.fitness = fitness;
	}

	public double getFitness() {
		return this.fitness;
	}
```

```
	public String toString() {
		String output = "";
		for (int gene = 0; gene < this.chromosome.length; gene++) {
			output += this.chromosome[gene];
		}
		return output;
	}
}
```

✓ **Class Population**

La classe est généralement utilisée pour effectuer des opérations au niveau du groupe sur ses individus, comme trouver les individus les plus forts, collecter des statistiques sur la population dans son ensemble, et en sélectionnant les individus à muter ou à croiser.

```
package chapter3;
import java.util.Arrays;
import java.util.Comparator;
import java.util.Random;

public class Population {
	private Individual population[];
	private double populationFitness = -1;

	public Population(int populationSize) {
		// Initial population
		this.population = new Individual[populationSize];
	}

	public Population(int populationSize, int chromosomeLength) {
		// Initialize the population as an array of individuals
		this.population = new Individual[populationSize];

		// Create each individual in turn
		for (int individualCount = 0; individualCount <
populationSize; individualCount++) {
			// Create an individual, initializing its chromosome to
the given
			// length
			Individual individual = new
Individual(chromosomeLength);
			// Add individual to population
			this.population[individualCount] = individual;
		}
	}


	public Individual[] getIndividuals() {
		return this.population;
	}

	public Individual getFittest(int offset) {
		// Order population by fitness
		Arrays.sort(this.population, new Comparator<Individual>() {
```

```
                @Override
                public int compare(Individual o1, Individual o2) {
                        if (o1.getFitness() > o2.getFitness()) {
                                return -1;
                        } else if (o1.getFitness() < o2.getFitness()) {
                                return 1;
                        }
                        return 0;
                }
        });

        // Return the fittest individual
        return this.population[offset];
}

public void setPopulationFitness(double fitness) {
        this.populationFitness = fitness;
}


public double getPopulationFitness() {
        return this.populationFitness;
}


public int size() {
        return this.population.length;
}

public Individual setIndividual(int offset, Individual individual) {
        return population[offset] = individual;
}


public Individual getIndividual(int offset) {
        return population[offset];
}

public void shuffle() {
        Random rnd = new Random();
        for (int i = population.length - 1; i > 0; i--) {
                int index = rnd.nextInt(i + 1);
                Individual a = population[index];
                population[index] = population[i];
                population[i] = a;
        }
}
}
```

✓ **class RobotController :**

main exécutive class pour le problème du contrôleur de robot.

Nous allons créer un labyrinthe à la main et l'alimenter au(GeneticAlgorithm's) méthode (évalPopulation) , qui est ensuite responsable de la notation d'un résumé robot avec capteurs contre le labyrinthe.

Limite supérieure du nombre de générations à courir. 200 générations c'est suffisant pour trouver le chemin environ 50% du temps, mais pour la démonstration fine, nous avons défini ce niveau élevé par défaut.

- Les 3 représentent le bon chemin à travers le labyrinthe. Vous pouvez suivre visuellement les 3 pour trouver le bon chemin à travers le labyrinthe.

- Les (1) sont des murs qui ne peut pas être parcouru,
  - Les(0) sont des positions valides, mais pas le bon itinéraire.

```
package chapter3;

public class RobotController {

	public static int maxGenerations = 1000;

	public static void main(String[] args) {


		Maze maze = new Maze(new int[][] {
			{ 0, 0, 0, 0, 1, 0, 1, 3, 3 },
			{ 1, 0, 1, 1, 1, 0, 1, 3, 1 },
			{ 1, 0, 0, 1, 3, 3, 3, 3, 1 },
			{ 3, 3, 3, 1, 3, 1, 1, 0, 1 },
			{ 3, 3, 3, 3, 3, 1, 1, 0, 0 },
			{ 3, 3, 1, 1, 1, 1, 0, 1, 1 },
			{ 1, 3, 4, 1, 3, 3, 3, 3, 3 },
			{ 0, 3, 1, 1, 3, 1, 0, 1, 3 },
			{ 1, 3, 3, 3, 3, 1, 1, 1, 4 }
			});
```

Nous devons initialiser une population d'individus aléatoires. Chacun de ces individus devrait avoir une longueur de chromosome de 128. Comme expliqué précédemment, 128 bits nous permettent de mapper les 64 entrées à une action. Puisqu'il n'est pas possible de créer un chromosome invalide pour ce problème, nous prendrons simplement des mesures aléatoires

lorsqu'ils seront confrontés à différentes situations, et à travers des générations d'évolution, nous espérons affiner comportement.

- ✓ **class GeneticAlgorithm :**

Cette classe GeneticAlgorithm est conçue pour résoudre les "Contrôleur de robot dans un labyrinthe",

nous avons apporté trois modifications simples : premièrement, nous avons ajouté "protected int tournoiSize" aux propriétés de la classe. Deuxièmement, nous avons ajouté « int tournoiSize » comme cinquième argument du constructeur. Enfin, nous avons ajouté l'affectation « this.tournament Size = tournoiSize » au constructeur. Avec la propriété tournoiSize gérée, ce code ira dans la méthode "main" de la classe exécutive, que nous avons nommée RobotController. Le code ci-dessous ne fera rien, bien sûr - nous n'avons implémenté aucune des méthodes de algorithme génétique.

```
package chapter3;


public class GeneticAlgorithm {

	private int populationSize;
	private double mutationRate;
	private double crossoverRate;
	private int elitismCount;


	protected int tournamentSize;

	public GeneticAlgorithm(int populationSize, double mutationRate,
double crossoverRate, int elitismCount,
			int tournamentSize) {

		this.populationSize = populationSize;
		this.mutationRate = mutationRate;
		this.crossoverRate = crossoverRate;
		this.elitismCount = elitismCount;
		this.tournamentSize = tournamentSize;
		}
```

Nous n'avons encore mis en œuvre aucune des techniques dont nous avons besoin, mais nous avons fait quelques ajouts et commentaires simples pour nous aider également à rester sur la bonne voie en ce qui concerne les méthodes que nous devons encore mettre en exécuter.

```
package chapter3;

public class RobotController {

	public static int maxGenerations = 1000;

	public static void main(String[] args) {


		Maze maze = new Maze(new int[][] {
			{ 0, 0, 0, 0, 1, 0, 1, 3, 3 },
			{ 1, 0, 1, 1, 1, 0, 1, 3, 1 },
			{ 1, 0, 0, 1, 3, 3, 3, 3, 1 },
			{ 3, 3, 3, 1, 3, 1, 1, 0, 1 },
			{ 3, 3, 3, 3, 3, 1, 1, 0, 0 },
			{ 3, 3, 1, 1, 1, 1, 0, 1, 1 },
			{ 1, 3, 4, 1, 3, 3, 3, 3, 3 },
			{ 0, 3, 1, 1, 3, 1, 0, 1, 3 },
			{ 1, 3, 3, 3, 3, 1, 1, 1, 4 }
```

```
        GeneticAlgorithm ga = new GeneticAlgorithm(200, 0.05, 0.9, 2, 10);
              Population population = ga.initPopulation(128);
              ga.evalPopulation(population, maze);
              int generation = 1;

              while (ga.isTerminationConditionMet(generation,
maxGenerations) == false) {

                    Individual fittest = population.getFittest(0);
                    System.out.println(
                                "G" + generation + " Best solution (" +
fittest.getFitness() + "): " + fittest.toString());


                    population = ga.crossoverPopulation(population);


                    population = ga.mutatePopulation(population);
                    ga.evalPopulation(population, maze);
                    generation++;
              }

              System.out.println("Stopped after " + maxGenerations + "
generations.");
              Individual fittest = population.getFittest(0);
              System.out.println("Best solution (" + fittest.getFitness() +
"): " + fittest.toString());

        }
}
```

6.3 Évaluation

Dans la phase d'évaluation, nous devons définir une fonction de fitness qui peut évaluer chaque contrôleur robotique. Nous pouvons le faire en augmentant la condition physique de l'individu pour chaque mouvement unique correct effectué sur l'itinéraire. Rappelez-vous que la classe Maze que nous avons créée précédemment a une méthode scoreRoute qui effectue cette évaluation. Cependant, l'itinéraire lui-même provient d'un robot sous contrôle autonome. Donc, avant de pouvoir donner à la classe Maze un itinéraire à évaluer, nous devons créer un Robot qui peut suivre des instructions et générer un itinéraire en exécutant ces instructions. Créez une classe Robot pour gérer les fonctionnalités du robot.

- ✓ **Class Robot**

```
package chapter3;
import java.util.ArrayList;
```

```
public class Robot {
    private enum Direction {NORTH, EAST, SOUTH, WEST};

    private int xPosition;
    private int yPosition;
    private Direction heading;
    int maxMoves;
    int moves;
    private int sensorVal;
    private final int sensorActions[];
    private Maze maze;
    private ArrayList<int[]> route;


    public Robot(int[] sensorActions, Maze maze, int maxMoves){
        this.sensorActions = this.calcSensorActions(sensorActions);
        this.maze = maze;
        int startPos[] = this.maze.getStartPosition();
        this.xPosition = startPos[0];
        this.yPosition = startPos[1];
        this.sensorVal = -1;
        this.heading = Direction.EAST;
        this.maxMoves = maxMoves;
        this.moves = 0;
        this.route = new ArrayList<int[]>();
        this.route.add(startPos);
    }

    public void run(){
        while(true){
            this.moves++;

            // Break if the robot stops moving
            if (this.getNextAction() == 0) {
                return;
            }

            // Break if we reach the goal
            if (this.maze.getPositionValue(this.xPosition, this.yPosition)
== 4) {
                return;
            }

            // Break if we reach a maximum number of moves
            if (this.moves > this.maxMoves) {
                return;
            }

            // Run action
            this.makeNextAction();
        }
    }

    private int[] calcSensorActions(int[] sensorActionsStr){
        // How many actions are there?
        int numActions = (int) sensorActionsStr.length / 2;
        int sensorActions[] = new int[numActions];

        // Loop through actions
```

```
        for (int sensorValue = 0; sensorValue < numActions; sensorValue++){
            // Get sensor action
            int sensorAction = 0;
            if (sensorActionsStr[sensorValue*2] == 1){
                sensorAction += 2;
            }
            if (sensorActionsStr[(sensorValue*2)+1] == 1){
                sensorAction += 1;
            }

            // Add to sensor-action map
            sensorActions[sensorValue] = sensorAction;
        }

        return sensorActions;
    }


    public void makeNextAction(){
        // If move forward
        if (this.getNextAction() == 1) {
            int currentX = this.xPosition;
            int currentY = this.yPosition;

            // Move depending on current direction
            if (Direction.NORTH == this.heading) {
                this.yPosition += -1;
                if (this.yPosition < 0) {
                    this.yPosition = 0;
                }
            }
            else if (Direction.EAST == this.heading) {
                this.xPosition += 1;
                if (this.xPosition > this.maze.getMaxX()) {
                    this.xPosition = this.maze.getMaxX();
                }
            }
            else if (Direction.SOUTH == this.heading) {
                this.yPosition += 1;
                if (this.yPosition > this.maze.getMaxY()) {
                    this.yPosition = this.maze.getMaxY();
                }
            }
            else if (Direction.WEST == this.heading) {
                this.xPosition += -1;
                if (this.xPosition < 0) {
                    this.xPosition = 0;
                }
            }

            // We can't move here
            if (this.maze.isWall(this.xPosition, this.yPosition) == true) {
                this.xPosition = currentX;
                this.yPosition = currentY;
            }
            else {
                if(currentX != this.xPosition || currentY !=
this.yPosition) {
                    this.route.add(this.getPosition());
```

```
                }
            }
        }
        // Move clockwise
        else if(this.getNextAction() == 2) {
            if (Direction.NORTH == this.heading) {
                this.heading = Direction.EAST;
            }
            else if (Direction.EAST == this.heading) {
                this.heading = Direction.SOUTH;
            }
            else if (Direction.SOUTH == this.heading) {
                this.heading = Direction.WEST;
            }
            else if (Direction.WEST == this.heading) {
                this.heading = Direction.NORTH;
            }
        }
        // Move anti-clockwise
        else if(this.getNextAction() == 3) {
            if (Direction.NORTH == this.heading) {
                this.heading = Direction.WEST;
            }
            else if (Direction.EAST == this.heading) {
                this.heading = Direction.NORTH;
            }
            else if (Direction.SOUTH == this.heading) {
                this.heading = Direction.EAST;
            }
            else if (Direction.WEST == this.heading) {
                this.heading = Direction.SOUTH;
            }
        }

        // Reset sensor value
        this.sensorVal = -1;
    }


    public int getNextAction() {
        return this.sensorActions[this.getSensorValue()];
    }


    public int getSensorValue(){
        // If sensor value has already been calculated
        if (this.sensorVal > -1) {
            return this.sensorVal;
        }

             boolean frontSensor, frontLeftSensor, frontRightSensor,
leftSensor, rightSensor, backSensor;
             frontSensor = frontLeftSensor = frontRightSensor = leftSensor
= rightSensor = backSensor = false;

        // Find which sensors have been activated
        if (this.getHeading() == Direction.NORTH) {
            frontSensor = this.maze.isWall(this.xPosition, this.yPosition-
1);
```

```
            frontLeftSensor = this.maze.isWall(this.xPosition-1,
this.yPosition-1);
            frontRightSensor = this.maze.isWall(this.xPosition+1,
this.yPosition-1);
            leftSensor = this.maze.isWall(this.xPosition-1,
this.yPosition);
            rightSensor = this.maze.isWall(this.xPosition+1,
this.yPosition);
            backSensor = this.maze.isWall(this.xPosition,
this.yPosition+1);
        }
        else if (this.getHeading() == Direction.EAST) {
            frontSensor = this.maze.isWall(this.xPosition+1,
this.yPosition);
            frontLeftSensor = this.maze.isWall(this.xPosition+1,
this.yPosition-1);
            frontRightSensor = this.maze.isWall(this.xPosition+1,
this.yPosition+1);
            leftSensor = this.maze.isWall(this.xPosition, this.yPosition-
1);
            rightSensor = this.maze.isWall(this.xPosition,
this.yPosition+1);
            backSensor = this.maze.isWall(this.xPosition-1,
this.yPosition);
        }
        else if (this.getHeading() == Direction.SOUTH) {
            frontSensor = this.maze.isWall(this.xPosition,
this.yPosition+1);
            frontLeftSensor = this.maze.isWall(this.xPosition+1,
this.yPosition+1);
            frontRightSensor = this.maze.isWall(this.xPosition-1,
this.yPosition+1);
            leftSensor = this.maze.isWall(this.xPosition+1,
this.yPosition);
            rightSensor = this.maze.isWall(this.xPosition-1,
this.yPosition);
            backSensor = this.maze.isWall(this.xPosition, this.yPosition-
1);
        }
        else {
            frontSensor = this.maze.isWall(this.xPosition-1,
this.yPosition);
            frontLeftSensor = this.maze.isWall(this.xPosition-1,
this.yPosition+1);
            frontRightSensor = this.maze.isWall(this.xPosition-1,
this.yPosition-1);
            leftSensor = this.maze.isWall(this.xPosition,
this.yPosition+1);
            rightSensor = this.maze.isWall(this.xPosition, this.yPosition-
1);
            backSensor = this.maze.isWall(this.xPosition+1,
this.yPosition);
        }

        // Calculate sensor value
        int sensorVal = 0;

        if (frontSensor == true) {
            sensorVal += 1;
```

```
        }
        if (frontLeftSensor == true) {
            sensorVal += 2;
        }
        if (frontRightSensor == true) {
            sensorVal += 4;
        }
        if (leftSensor == true) {
            sensorVal += 8;
        }
        if (rightSensor == true) {
            sensorVal += 16;
        }
        if (backSensor == true) {
            sensorVal += 32;
        }

        this.sensorVal = sensorVal;

        return sensorVal;
    }


    public int[] getPosition(){
        return new int[]{this.xPosition, this.yPosition};
    }

    private Direction getHeading(){
        return this.heading;
    }


    public ArrayList<int[]> getRoute(){
        return this.route;
    }


    public String printRoute(){
        String route = "";

        for (Object routeStep : this.route) {
            int step[] = (int[]) routeStep;
            route += "{" + step[0] + "," + step[1] + "}";
        }
        return route;
    }
}
```

Cette classe contient le constructeur pour créer un nouveau Robot. Il contient également des fonctions pour lire les capteurs du robot, pour obtenir le cap du robot et pour déplacer le robot dans le labyrinthe. Cette classe Robot est notre façon de simuler un robot simple afin que nous n'ayons pas à exécuter 1 000 générations d'évolution sur une population de 100 robots réels. Vous trouverez souvent des classes comme Maze

et Robot dans des problèmes d'optimisation comme ceux-ci, où il est efficace de simuler via un logiciel avant d'affiner vos résultats dans le matériel de production.

Calcul Fitness :

class Maze qui évalue l'aptitude d'un itinéraire. Cependant, nous devons encore implémenter la méthode calcFitness dans notre classe GeneticAlgorithm. Plutôt que de calculer directement le score de fitness, la méthode calcFitness est chargée de lier les classes Individu, Robot et Labyrinthe en créant un nouveau Robot avec le chromosome de l'Individu (c'est-à-dire, le jeu d'instructions du contrôleur de capteur) et en l'évaluant par rapport à notre Labyrinthe.

La fonction de Fitness est :

```
public double calcFitness(Individual individual, Maze maze) {

		// Get individual's chromosome

		int[] chromosome = individual.getChromosome();



		// Get fitness

		Robot robot = new Robot(chromosome, maze, 100);

		robot.run();

		int fitness = maze.scoreRoute(robot.getRoute());



		// Store fitness

		individual.setFitness(fitness);



		return fitness;

	}
```

Ajoutez la méthode suivante à la classe GeneticAlgorithm, où vous le souhaitez :

```
public void evalPopulation(Population population, Maze maze) {
		double populationFitness = 0;


		for (Individual individual : population.getIndividuals()) {
			populationFitness += this.calcFitness(individual, maze);
		}

		population.setPopulationFitness(populationFitness);
```

```
}
```

Et avec cela, Nous pouvons résoudre les deux lignes "Évaluer la population" dans la méthode "main" du RobotController.

Et remplacez-les par :

Evaluate population ga.evalPopulation(population, maze);

7. Méthode de sélection et croisement et mutation

Nous utilisons la sélection de tournoi et le croisement à point unique.

7.1 Sélection du tournoi

Dans GeneticAlgorithm class :

```
public Individual selectParent(Population population) {
		// Create tournament
		Population tournament = new Population(this.tournamentSize);

		// Add random individuals to the tournament
		population.shuffle();
		for (int i = 0; i < this.tournamentSize; i++) {
			Individual tournamentIndividual =
population.getIndividual(i);
			tournament.setIndividual(i, tournamentIndividual);
		}

		// Return the best
		return tournament.getFittest(0);
	}
```

7.2 Mutation :

```
// Apply mutation population = ga.mutatePopulation(population);
```

7.3 Croisement à point unique :

Le chromosome est un ensemble codé d'instructions basé sur six entrées de capteur, et chaque instruction a plus d'un bit de long. Imaginez une situation de croisement idéale comme suit : parent1 est excellent lors des 32 premières opérations de capteur, et parent2 est excellent lors, disons, des 16 dernières opérations. Si nous devions utiliser la technique de croisement uniforme du chapitre 2, nous aurions des morceaux mélangés partout ! Les instructions individuelles seraient modifiées et corrompues dans le croisement en raison du croisement uniforme choisissant des bits au hasard à

échanger. Les instructions à deux bits peuvent ne pas être conservées du tout, car l'un des deux bits de chaque instruction peut être modifié. Cependant, le croisement en un seul point nous permet de capitaliser sur cette situation idéale. Si le point de croisement est directement au milieu du chromosome, la progéniture se retrouverait avec 64 bits ininterrompus représentant 32 instructions de parent1, ainsi que les 16 grandes instructions de parent2. Ainsi, la progéniture excelle désormais dans 48 des 64 états possibles. Ce concept est à la base des algorithmes génétiques : que la progéniture peut être plus forte que l'un ou l'autre parent parce qu'elle tire les meilleures qualités des deux.

Pour implémenter un croisement à point unique, nous ajoutons le code suivant à la classe GeneticAlgorithm :

```
public Population crossoverPopulation(Population population) {

            Population newPopulation = new Population(population.size());


            for (int populationIndex = 0; populationIndex <
population.size(); populationIndex++) {
                  Individual parent1 =
population.getFittest(populationIndex);


                  if (this.crossoverRate > Math.random() &&
populationIndex >= this.elitismCount) {

                        Individual offspring = new
Individual(parent1.getChromosomeLength());


                        Individual parent2 =
this.selectParent(population);

                        int swapPoint = (int) (Math.random() *
(parent1.getChromosomeLength() + 1));

                        for (int geneIndex = 0; geneIndex <
parent1.getChromosomeLength(); geneIndex++) {


                              if (geneIndex < swapPoint) {
                                    offspring.setGene(geneIndex,
parent1.getGene(geneIndex));
                              } else {
                                    offspring.setGene(geneIndex,
parent2.getGene(geneIndex));
                              }
                        }
                        newPopulation.setIndividual(populationIndex,
offspring);
                  } else {
```

```
                    newPopulation.setIndividual(populationIndex,
parent1);
                }
            }

            return newPopulation;
        }
```

8. Exécution

Notre IDE montre 1 000 générations d'évolution, et notre algorithme s'est terminé avec un score de fitness de 29, ce qui est le maximum pour ce labyrinthe particulier. (Vous pouvez compter le nombre de tuiles "Route" - représentées par "3" - dans la définition du labyrinthe pour obtenir ce nombre et la chaîne de chromosome.

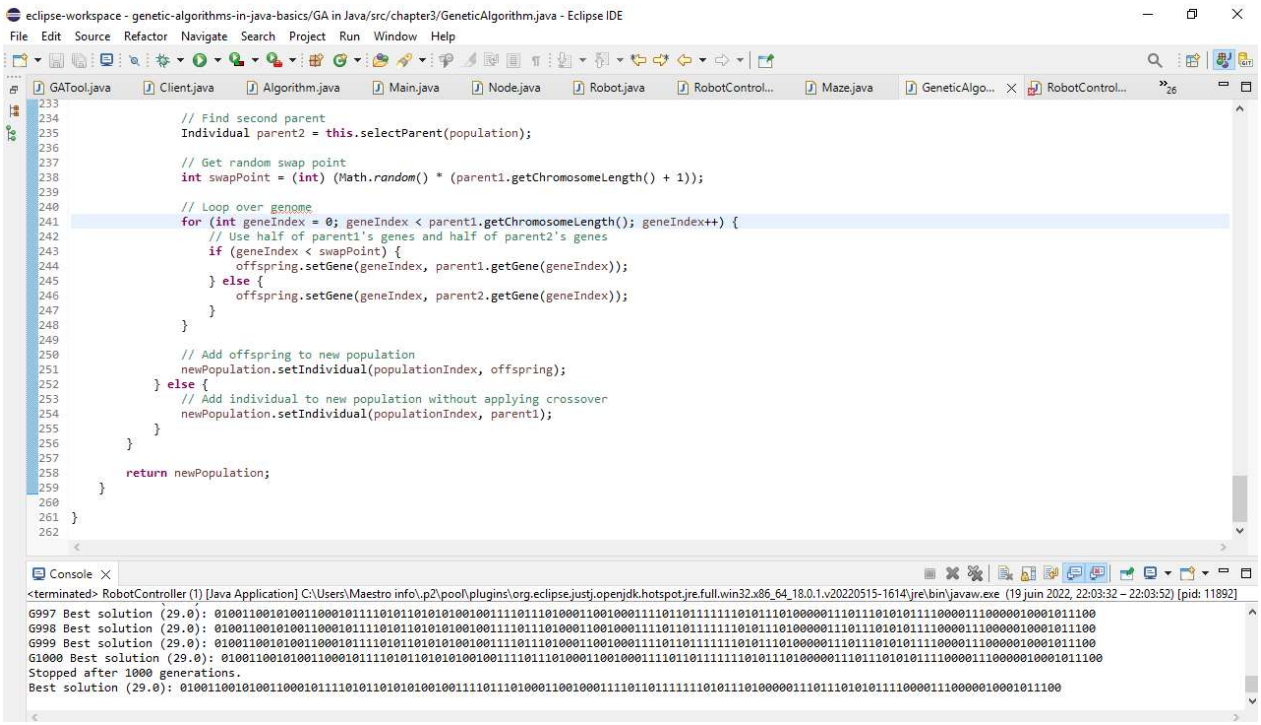


Figure 5.6 : Exécution.

9. Exemple :

Exemple 1 :

```
Maze maze = new Maze(new int[][] {
        { 0, 0, 0, 0, 1, 0, 1, 3, 2 },
        { 1, 0, 1, 1, 1, 0, 1, 3, 1 },
        { 1, 0, 0, 1, 3, 3, 3, 3, 1 },
        { 3, 3, 3, 1, 3, 1, 1, 0, 1 },
        { 3, 4, 3, 3, 3, 1, 1, 0, 0 },
        { 3, 3, 1, 1, 1, 1, 0, 1, 1 },
        { 1, 3, 0, 1, 3, 3, 3, 3, 3 },
        { 0, 3, 1, 1, 3, 1, 0, 1, 3 },
        { 1, 3, 3, 3, 3, 1, 1, 1, 4 }
```

Figure 5.7 : Exemple 1.

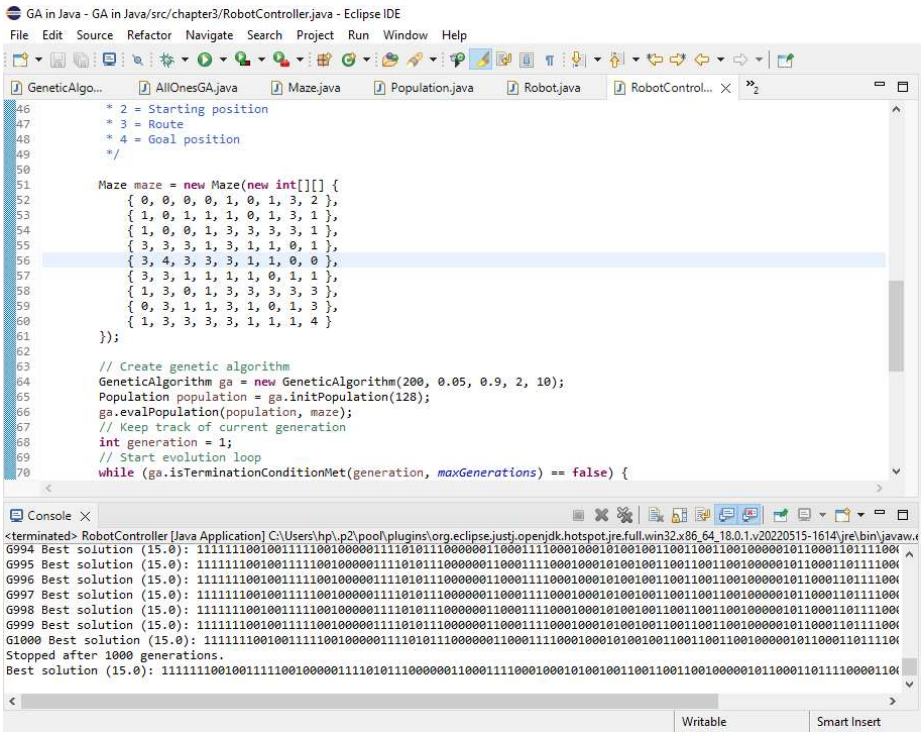


Figure 5.8 : Exécution exemple 1.

Exemple 2 :

```
Maze maze = new Maze(new int[][] {
        { 0, 0, 0, 0, 1, 0, 1, 3, 3 },
        { 1, 0, 1, 1, 1, 0, 1, 3, 1 },
        { 1, 0, 0, 1, 3, 3, 3, 3, 1 },
        { 3, 3, 3, 1, 3, 1, 1, 0, 1 },
        { 3, 3, 3, 3, 3, 1, 1, 0, 0 },
        { 3, 3, 1, 1, 1, 1, 0, 1, 1 },
        { 1, 3, 4, 1, 3, 3, 3, 3, 3 },
        { 0, 3, 1, 1, 3, 1, 0, 1, 3 },
        { 1, 3, 3, 3, 3, 1, 1, 1, 4 }
```

Figure 5.9 : Exemple 2.

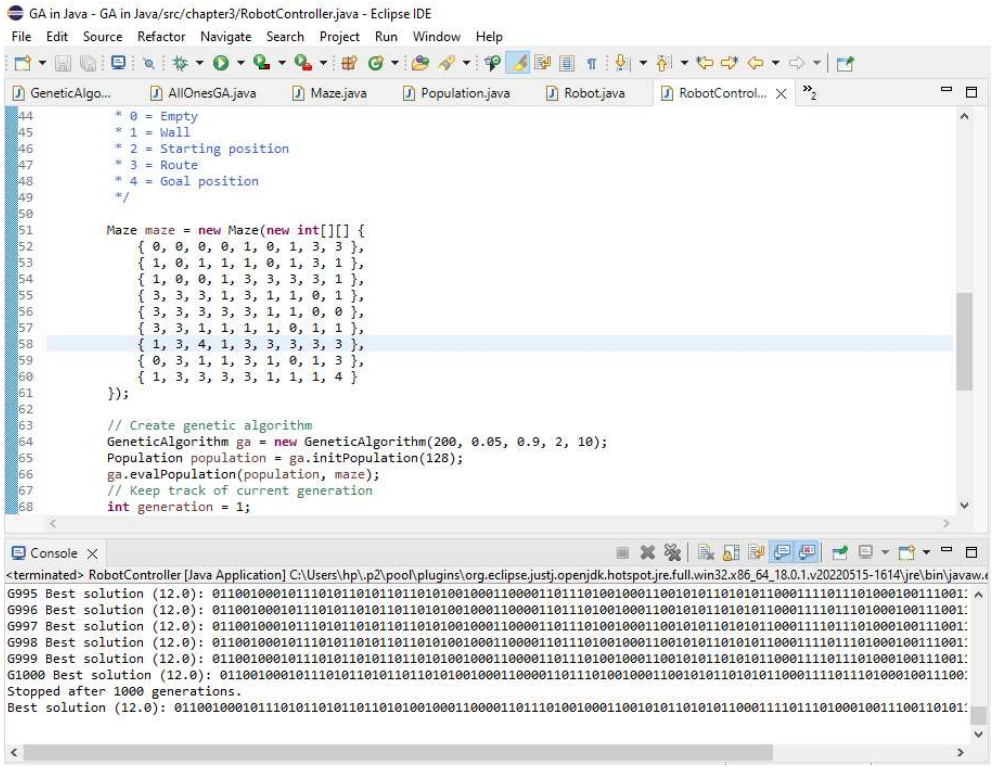


Figure 5.10 : Exécution exemple 2.

10. Conclusion :

Connaître toutes les opérations des algorithmes génétiques afin de résoudre un problème spécifique nécessite du temps et de la concentration, car le but que nous voulons atteindre est de rendre le robot indépendant de l'intervention humaine, quel que soit l'environnement dans lequel il a été placé, et en raison de la manque de temps, l'effort s'est limité à ce qui a été atteint dans ce chapitre uniquement Nous avons utilisé le livre Genetic Algorithms in Java Basics(Vous pouvez le trouver dans les références [9]). et beaucoup de recherches puis l'avons formulé sous la forme qui est apparue afin d'atteindre l'image et l'interface finales, et l'objectif est de poursuivre ce programme et d'essayer de le développer davantage dans un avenir proche.

Conclusion générale

L'objectif principal de notre projet était d'appliquer des algorithmes génétiques pour résoudre le problème du labyrinthe, qui est un problème complexe et difficile à résoudre.

Au cours de cette étude, nous sommes passés par différentes étapes de recherche et d'investigation. Notre projet C'est un projet presque terminé et est joint avec tous les documents et concepts nécessaires pour le développer correctement.

Pour développer ce travail, nous avons d'abord présenté une étude sur les algorithmes génétiques en général, c'est-à-dire que nous en avons fait une recherche exhaustive, et dans un deuxième temps, nous avons étudié la programmation génétique en général et une recherche détaillée. Et troisièmement, nous avons examiné les recherches précédentes, et cela nous a beaucoup aidés à connaître le chemin. Et quatrièmement, nous avons découvert le problème du labyrinthe lié aux robots.

Enfin, nous avons abordé toutes les étapes nécessaires à l'élaboration de ce travail. A ce stade, nous avons appris à bien manier le langage Java, et j'ai approfondi ma connaissance d'Eclipse.

De plus, l'objectif principal de ce projet était de découvrir le monde des robots à l'intérieur du labyrinthe. Ce projet a pleinement répondu à mes attentes, et des améliorations à cette recherche peuvent également être apportées. Au terme de cette thèse, notre travail est un pas en avant

Références bibliographiques :

[2] Yao Zhou, STUDY ON[1] http://www.rennard.org/alife/english/gavintrgb.html visite le 21/03/2022

GENETIC ALGORITHM IMPROVEMENT AND APPLICATION, WORCESTER POLYTECHNIC INSTITUTE in partial fulfillment of the requirements for the Degree of Master of Science in Manufacturing Engineering, May 2006.

[3] L. Howard ,D. D'Angelo, "The GA-P: A Genetic Algorithm and Genetic Programming Hybrid", IEEE Expert, Vol. 10, No. 3, 1995, pp.11-15.

[4] A.HANNACHE, A. EMMOUIS, Résolution d'un problème d'ordonnancement de type job shop avec contrainte de transport, Projet de Fin d'Études master, université Abou Bekr Belkaid Tlemcen, 2016.

[5] Univ bejaia,http://www.univbejaia.dz/dspace/bitstream/handle/123456789/8269/Etude%20comparative%20des%20algorithmes%2 0génétiques%20multi-objectifs.pdf?sequence=1&isAllowed=y ,consulté le 6 Mars 2020.

[6] KACIMI EL HASSANI Aicha ABASSI Hayat, L'optimisation à base de simulation pour les chaînes logistiques, MEMOIRE de fin d'étude Présenté pour l'obtention du diplôme de MASTER,msila,2020.

[7] ANTON JONASSON, SIMON WESTERLIND, Genetic algorithms in mazes, 2016.

[8] L.DAHMANE, S.HADROUG , Algorithme génétique parallèle pour un problème d'ordonnancement d'atelier Job Shop , MEMOIRE de fin d'études de MASTER , 2020.

[9] Marc-André Gardner, Contrôle de la croissance de la taille des individus en Programmation génétique, 2014.

[10]https://perso.liris.cnrs.fr/alain.mille/enseignements/master_ia/rapports_2006/Programmation%20Genetique_4p.pdf.

[9] Lee Jacobson Burak Kanber, .Genetic Algorithms in Java Basics, 2015.

[10]https://e3arabi.com/%D8%A7%D9%84%D8%AA%D9%82%D9%86%D9%8A%D8%A9/eclipse

[11] M.ACHACH, Résoudre le problème de régression symbolique par la programmation génétique, MEMOIRE de fin d'études de MASTER, 2019.
[12]https://perso.liris.cnrs.fr/alain.mille/enseignements/master_ia/rapports_2006/Programmation%20Genetique_4p.pdf

{13] https://www-lisic.univ-littoral.fr/~fonlupt/Recherche/PG/prog-mutation.html

[14] A.Hemmak, programmation génétique, cours 3, 2021.

[15] Nitin Choubey, A-Mazer with Genetic Algorithm, November 2012.

[16] Lee Jacobson Burak Kanber, .Genetic Algorithms in Java Basics, 2015.

[17]https://e3arabi.com/%D8%A7%D9%84%D8%AA%D9%82%D9%86%D9%8A%D8%A9/eclipse.

[18] Thomas Pasquier, Julien Erdogan, Genetic Algorithm Optimization in Maze Solving Problem.