CY Cergy-Paris Université

RAPPORT

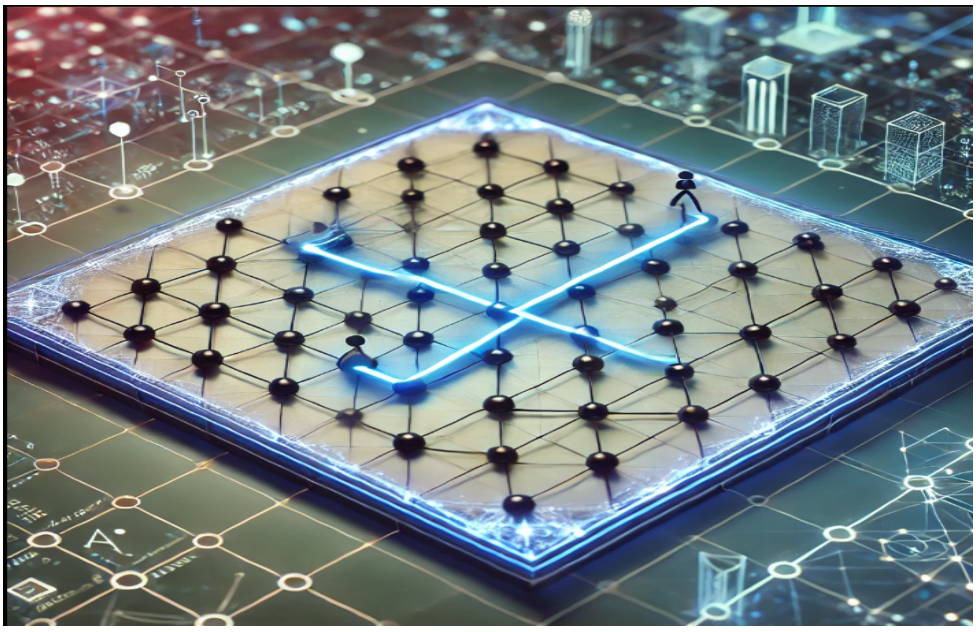
pour le projet Génie Logiciel
L2 informatique, 2024/2025

sur le sujet

Intélligence artificielle

rédigé par

ACHAB Ouardia
ISSAD Lisa
DE ANGELIS Enzo



Avril 2025

Table des matières

Table des figures

Liste des tableaux

Remerciements

Nous tenons à exprimer notre sincère gratitude à notre enseignant de Génie Logiciel, Monsieur Tianxiao LIU, pour son encadrement rigoureux et ses conseils avisés tout au long de ce projet. Sa disponibilité, son expertise technique et ses retours constructifs nous ont permis d'améliorer considérablement notre travail.

Nous souhaitons également remercier le département d'informatique de CY Cergy-Paris Université pour nous avoir fourni les ressources nécessaires à la réalisation de ce projet, notamment l'accès aux environnements de développement et aux références bibliographiques essentielles.

Notre reconnaissance s'adresse aussi à nos camarades de classe qui ont participé aux séances de test et nous ont fourni des commentaires pertinents pour améliorer l'interface utilisateur de notre application.

Enfin, nous remercions nos familles respectives pour leur soutien moral et leurs encouragements constants durant les périodes intensives de travail sur ce projet.

1 Introduction

1.1 Contexte du projet

Imaginez-vous perdu dans un labyrinthe complexe, cherchant désespérément la sortie. Dans votre main, deux outils magiques vous sont proposés : une boussole enchantée qui calcule toujours le chemin optimal (notre algorithme A*) et une conscience curieuse capable d'apprendre de ses erreurs et de s'améliorer à chaque tentative (notre perceptron). Lequel choisiriez-vous ? C'est précisément ce dilemme fascinant que notre projet explore à travers deux applications interactives.

Notre voyage a commencé par une simple question : comment visualiser et comparer ces deux approches fondamentalement différentes pour résoudre le même problème ? Les labyrinthes ont toujours captivé l'imagination humaine, des mythes anciens aux jeux vidéo modernes, et les techniques pour les résoudre représentent un défi intellectuel passionnant.

1.2 Objectif du projet

Ce projet a pour objectif principal d'implémenter et de comparer deux approches distinctes pour la recherche de chemin : l'algorithme A* [Pat25] et une approche basée sur le perceptron. Nos objectifs spécifiques sont les suivants :

- Développer deux applications Java distinctes visualisant les processus de recherche de chemin
- Implémenter l'algorithme A* avec une visualisation en temps réel de son exécution
- Concevoir des interfaces graphiques intuitives permettant d'observer le comportement des algorithmes
- Comparer les performances et caractéristiques des deux approches

Notre exploration s'est matérialisée sous forme de deux applications Java interactives, où l'utilisateur peut observer en temps réel comment chaque algorithme navigue à travers une grille parsemée d'obstacles. L'algorithme A* calcule méthodiquement son chemin, tandis que le perceptron apprend par l'expérience les directions à suivre.

1.3 Organisation du rapport

Ce rapport s'articule autour de plusieurs chapitres structurés selon une progression logique et méthodique. En premier lieu, dans la section 2 nous exposerons le cadre théorique et les fondements conceptuels des deux approches algorithmiques étudiées. Nous aborderons ensuite dans la section 3 la phase de conception de nos applications, suivie d'une présentation détaillée de l'implémentation technique et des choix architecturaux adoptés. La section 4 sera consacré à la documentation technique et au manuel utilisateur, permettant une prise en main efficace des outils développés. L'analyse comparative des performances constituera une partie substantielle du rapport présentée dans la section 5, avec une évaluation quantitative et qualitative des deux approches selon divers critères d'efficacité . La conclusion dans la section 6 synthétisera les résultats obtenus et les enseignements principaux de cette étude comparative, tout en proposant des perspectives d'amélioration et des pistes de recherche pour des développements ultérieurs. Cette organisation méthodique permet une progression cohérente de l'analyse, allant des aspects théoriques aux considérations pratiques, en passant par l'implémentation technique et l'évaluation empirique des solutions proposées dans le domaine des algorithmes de recherche de chemin.

2 Spécification du projet

Nous avons présenté l'objectif du projet dans la section 1. Dans cette section, nous présentons la spécification de notre logiciel réalisé. Ce logiciel vise à simuler les principes fondamentaux de l'intelligence artificielle afin de faciliter l'apprentissage pour les personnes désireuses de découvrir ce domaine. Notre environnement de simulation illustre les solutions d'IA typiques telles que l'algorithme A* et le perceptron, offrant ainsi une plateforme interactive et pédagogique qui rend ces concepts complexes accessibles à un public plus large.

2.1 Notions de base et contraintes du projet

2.1.1 Fonctionnement général du logiciel

Le logiciel est une plateforme éducative interactive de simulation d'algorithmes d'intelligence artificielle. Il vise à permettre aux utilisateurs de comprendre et d'explorer les principes fondamentaux de l'IA de manière intuitive et visuelle.

2.1.2 Notions et terminologies de base

- Intelligence Artificielle (IA) : Système capable de simuler une intelligence humaine.
- Algorithme de recherche : Méthode systématique de résolution de problèmes.
- Apprentissage supervisé : Technique d'apprentissage automatique utilisant des données étiquetées.
- Classification binaire : Processus de séparation des données en deux catégories distinctes.

2.1.3 Contraintes et limitations connues

- Algorithmes limités : Le projet se concentrera initialement sur deux algorithmes (A* et Perceptron)
- Interface utilisateur : Doit être intuitive et accessible aux débutants
- Performance : Simulations en temps réel avec des interactions dynamiques
- Compatibilité : Application multiplateforme développée en Java

Notre projet IA se positionne comme une solution pédagogique innovante dans le domaine de l'intelligence artificielle. En développant deux interfaces interactives distinctes en Java — l'une dédiée à l'algorithme A* et l'autre au perceptron — nous permettons aux utilisateurs de visualiser concrètement le fonctionnement de ces approches fondamentales. L'A* met en lumière les techniques de recherche de chemin optimisé, tandis que le perceptron illustre les principes de base des réseaux neuronaux et de l'apprentissage supervisé [RN10]. Chaque interface propose une zone de visualisation claire et dynamique, facilitant la compréhension par l'observation directe du comportement des algorithmes.L'expérience proposée reste volontairement épurée et centrée sur l'essentiel : observer directement comment ces intelligences artificielles analysent leur environnement et adaptent leur comportement pour atteindre un objectif.

Outils de développement :

1. Langage de programmation : Java
2. IDE : Eclipse
3. Outil de rédaction : Latex

2.2 Fonctionnalités attendues du projet

Fonctionnalités Fonctionnalités du programme :

- Visualisation du fonctionnement de l'algorithme A*
- Simulation du modèle Perceptron
- Zones de visualisation graphique

— Intégration d'un système d'apprentissage interactif permettant aux utilisateurs de comprendre les concepts d'apprentissage

Document	Coefficient	Commentaire
Cahier des charges	37.5%	Premier document
Rapport	62.5%	Rapport final du projet

TABLE 1 – Répartition des coefficients d'évaluation du projet

Comme illustré dans le tableau 1, la répartition des coefficients entre les livrables met en évidence l'importance du rapport final qui représente près des deux tiers de l'évaluation globale du projet.

3 Conception et mise en œuvre

3.1 Architecture globale du logiciel

Dans la figure 1, on peut observer l'architecture modulaire complète de notre application. Le diagramme illustre les trois composants principaux du système : le module de l'algorithme A*, le module du Perceptron, et le module de visualisation. Chaque module possède ses propres sous-composantes spécifiques qui communiquent entre elles et partagent un modèle de données commun. Cette organisation modulaire facilite la maintenance du code, permet une évolution indépendante des différentes parties du système, et assure une séparation claire des responsabilités entre les algorithmes de recherche de chemin et leur représentation visuelle.

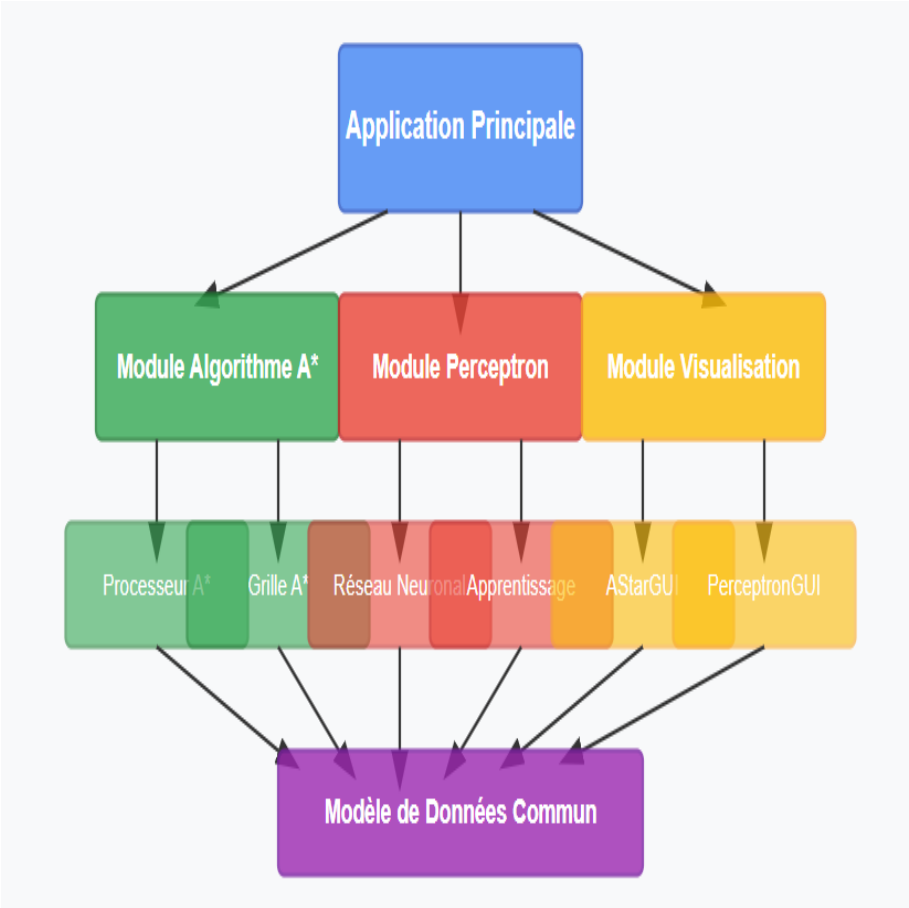


FIGURE 1 – Structure modulaire du projet

3.1.1 Structure Modulaire du Projet

Notre projet est organisé selon une architecture modulaire qui favorise la séparation des préoccupations et la réutilisation du code. Comme illustré dans le diagramme, les composants principaux sont :

- **Module Algorithme A*** : Ce module implémente l'algorithme A* pour la recherche de chemin optimal. Il comprend deux sous-composants principaux :
  - **Processeur A*** : Contient la logique algorithmique pour calculer le chemin le plus court entre deux points
  - **Grille A*** : Gère la représentation spatiale de l'environnement pour l'algorithme A*
- **Module Perceptron** : Responsable de l'implémentation neuronale pour la recherche de chemin par apprentissage. Il se décompose en :

  - **Réseau Neuronal** : Implémente la structure du perceptron et ses mécanismes de propagation
  - **Apprentissage** : Gère l'entraînement du réseau et l'ajustement des poids synaptiques
- **Module Visualisation** : S'occupe de l'affichage graphique et de l'interaction utilisateur :
  - **AStarGUI** : Interface graphique dédiée à l'algorithme A*
  - **PerceptronGUI** : Interface graphique pour la simulation du perceptron

Tous ces modules s'appuient sur un **Modèle de Données Commun** qui assure la cohérence des structures partagées et facilite la communication entre les composants.

3.1.2 Module de visualisation

Le module de visualisation est responsable de la représentation graphique des algorithmes en action. Il traduit les données abstraites en représentations visuelles compréhensibles pour l'utilisateur.

AStarGUI : Interface graphique dédiée à la simulation de l'algorithme A*
- Affiche une grille graphique représentant l'environnement, avec des cellules accessibles, des obstacles, un point de départ et un but
- Permet de visualiser l'exécution de l'algorithme A* étape par étape
- Met à jour dynamiquement les couleurs des cellules selon leur statut (fermée, accessible, parent, etc.)
- Utilise un thread pour gérer le déplacement progressif de l'agent sans bloquer l'interface graphique

PerceptronGUI : Interface graphique pour la simulation du perceptron
- Affiche une grille interactive représentant l'environnement de déplacement
- Permet l'entraînement du perceptron pour prédire les directions à suivre
- Offre une animation du parcours du perceptron jusqu'à la cellule d'arrivée

3.1.3 Design Patterns utilisés

Pour structurer notre code de manière modulaire, maintenable et évolutive, nous avons intégré plusieurs patrons de conception dans notre projet présentés dans le cours de génie logiciel [Uni25]. :
- **Template Method** : Ce patron est illustré par la hiérarchie `AbstractGrid`/`AGrid` et `AbstractCell`/`ACell`. Il permet de définir l'ossature générale des algorithmes de simulation (par exemple, l'entraînement ou le parcours) tout en laissant aux sous-classes le soin de redéfinir certaines étapes spécifiques. Cela facilite l'extension des comportements sans modifier la structure globale.
- **Factory Pattern** : Le pattern Factory est mis en œuvre dans la classe `GridGenerator`, qui encapsule la logique complexe de création des grilles adaptées (A* ou Perceptron) à partir d'un fichier `map.txt`. Cela permet d'instancier dynamiquement des objets selon le contexte, tout en séparant la logique d'instanciation du reste du programme.
- **Singleton Pattern** : Utilisé pour garantir qu'un seul objet central de configuration (ou gestionnaire de carte) soit partagé dans tout le système, ainsi que pour la gestion des graphiques statistiques. Ce pattern est implémenté dans la classe `ChartManager` qui centralise la création et la maintenance des différents graphiques (camembert pour la distribution des types de cellules, histogramme pour le statut des cellules, et courbe d'évolution de la valeur F). Ce design pattern assure que la grille chargée depuis `map.txt` et les données statistiques ne soient instanciées qu'une seule fois, évitant ainsi les redondances et les incohérences dans l'affichage des résultats.

Ces patterns ont permis une meilleure séparation des responsabilités, la réutilisabilité des composants et une plus grande flexibilité dans l'évolution du projet.

Génération de la grille Pour générer notre grille fixe, nous utilisons un fichier `map.txt` qui contient des valeurs binaires 0 et 1 (1 représente un mur et 0 représente une cellule vide). Le programme lit ce fichier pour construire l'environnement de navigation utilisé par l'algorithme A* et le Perceptron.

3.1.4 Journalisation pour le débogage

Pour faciliter le débogage et l'analyse des résultats de tests, nous avons implémenté un système de journalisation basé sur Log4j. La classe `LoggerUtility` nous permet de générer des logs dans deux formats différents :

— Format texte : pour une consultation rapide des événements du système
— Format HTML : pour une analyse plus structurée et visuelle des résultats

Cette approche de journalisation nous a permis de tracer précisément l'exécution de l'algorithme A* , documentant ainsi chaque étape du processus de décision et facilitant l'identification des problèmes potentiels.

3.2 Classes de données

La conception des classes de données pour notre simulation d'intelligence artificielle est organisée autour d'une hiérarchie orientée objet qui modélise deux aspects principaux : la représentation spatiale pour l'algorithme A* et la structure des données pour le Perceptron.

3.3 IHM Graphique

Pour créer l'interface graphique de l'application, nous avons utilisé la bibliothèque *Swing* de Java, qui permet de construire des interfaces utilisateur riches et interactives [Cor25].

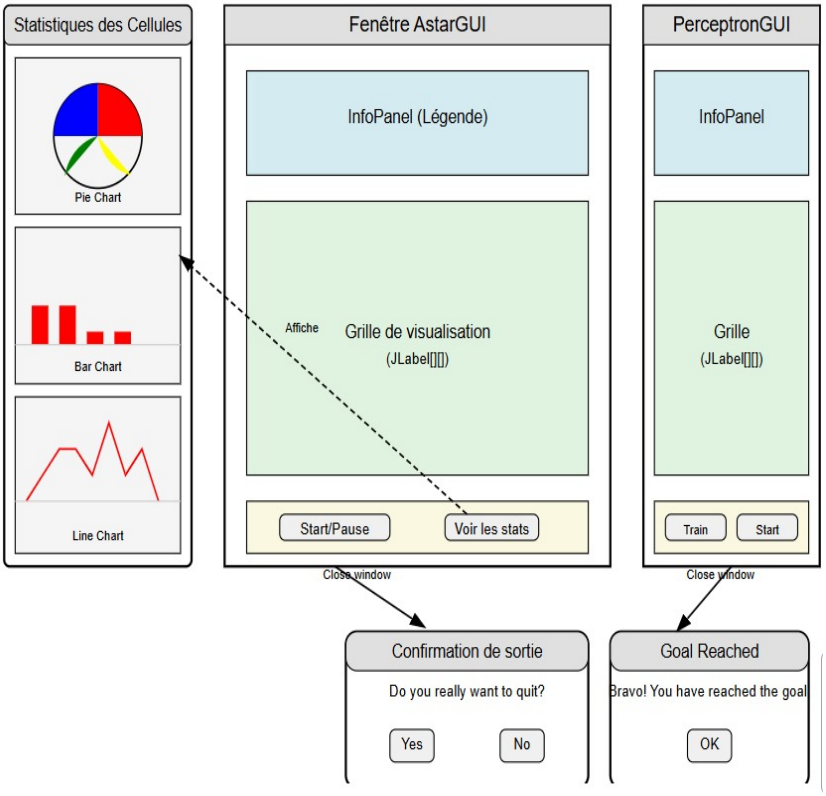


FIGURE 2 – IHM Graphique

La Figure 2 illustre l'architecture des interfaces graphiques de notre application. Le diagramme présente les deux fenêtres principales - AStarGUI et PerceptronGUI - ainsi que leurs interactions avec les boîtes de dialogue modales.

— Un panneau d'information supérieur (InfoPanel) qui affiche la légende des couleurs et symboles utilisés
— Une zone centrale de visualisation constituée d'une grille de labels représentant l'environnement
— Une section inférieure pour les contrôles utilisateur (boutons Start/Pause, Train Network et Voir statistiques)
— Un bouton "Voir statistiques" permettant d'accéder aux métriques de performance des algorithmes

Le diagramme montre également le flux d'interaction lors de trois événements principaux :

1. La fermeture de l'application, qui déclenche une boîte de dialogue de confirmation
2. L'atteinte de l'objectif, qui affiche un message de félicitations
3. L'accès aux statistiques détaillées via le bouton "Voir statistiques"

Cette architecture d'interface utilisateur respecte le principe de cohérence, en maintenant une structure similaire entre les deux applications tout en adaptant les contrôles spécifiques à chaque algorithme.

3.4 Conception des traitements (processus)

Dans la formule mathématique 1, on peut voir que l'algorithme A* utilise une fonction d'évaluation notée $f(n)$ pour chaque nœud n [RN10]. Cette fonction combine le coût réel du chemin depuis le départ ($g(n)$) et une estimation heuristique du coût restant jusqu'à l'objectif ($h(n)$). Cela permet à l'algorithme de trouver efficacement le chemin le plus court.

$$f(n) = g(n) + h(n) \tag{1}$$

— $f(n)$: coût total estimé du chemin passant par le nœud n.
— $g(n)$: coût réel depuis le départ jusqu'au nœud n.
— $h(n)$: estimation heuristique du coût restant jusqu'à l'objectif.

Dans la formule mathématique 2, on peut voir que la sortie d'un perceptron est déterminée par une somme pondérée des entrées, à laquelle on ajoute un biais [Gé19]. Cette somme est ensuite passée dans une fonction d'activation (souvent une fonction seuil), qui décide si le neurone s'active.

$$z = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b \tag{2}$$

Ensuite, une fonction d'activation $f(z)$ est appliquée à z. Pour une fonction seuil :

$$f(z) = \begin{cases} 1 & \text{si } z \geq \text{seuil} \\ 0 & \text{sinon} \end{cases} \tag{3}$$

— w_i : poids associé à l'entrée x_i.
— x_i : valeur d'entrée.
— b : biais.
— z : somme pondérée des entrées.

4 Manuel utilisateur

Ce manuel explique comment utiliser les deux composants principaux de notre application : l'explorateur de chemins A* et l'explorateur de chemins par Perceptron.

4.1 Introduction

Notre logiciel propose deux algorithmes de recherche de chemin différents :
— **A* Pathfinding Explorer** : utilise l'algorithme A* pour trouver le chemin optimal
— **Perceptron Pathfinding Explorer** : utilise un réseau neuronal (perceptron) pour déterminer le chemin

Chaque explorateur visualise en temps réel la progression de l'algorithme de recherche de chemin à travers une grille. Les deux applications partagent des fonctionnalités similaires tout en présentant des spécificités propres à leur algorithme respectif.

4.2 Explorateur de chemins A*

4.2.1 Interface utilisateur

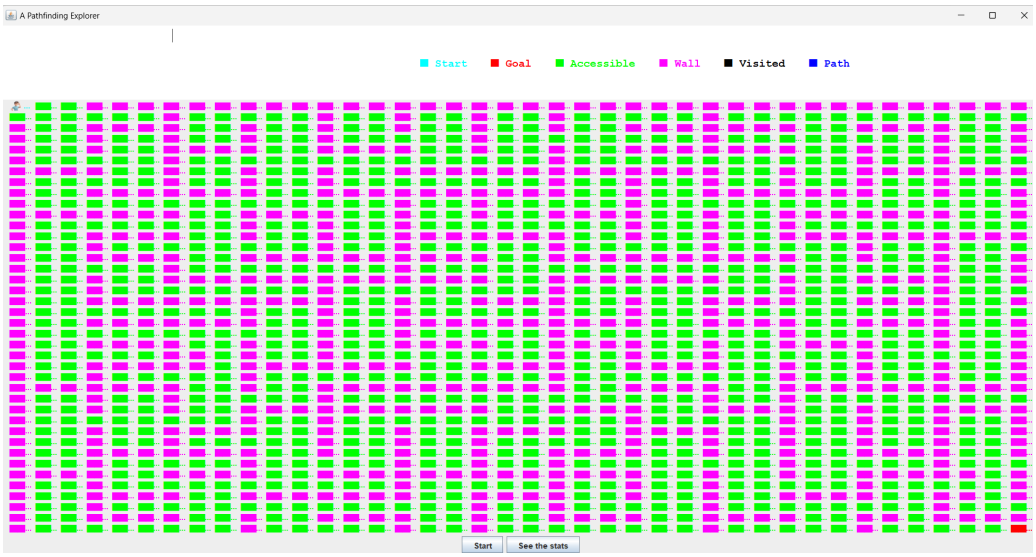


FIGURE 3 – Interface de l'explorateur A*

Comme illustré dans la Figure 3 l'interface de l'explorateur A* comprend :

1. Un panneau d'information en haut (indiquant la signification des couleurs)
2. Une grille centrale affichant l'environnement
3. Un bouton `Start/Pause` en bas pour contrôler la simulation

4.2.2 Utilisation

Pour utiliser l'explorateur A* :

1. **Lancer l'application** : Exécutez la classe `AstarGUI` pour démarrer l'application.
2. **Comprendre la grille** :
   — Les cellules en cyan représentent le point de départ
   — Les cellules en rouge représentent l'objectif à atteindre
   — Les cellules en vert sont les chemins accessibles
   — Les cellules en magenta représentent les obstacles (murs)
3. **Démarrer la simulation** : Cliquez sur le bouton `Start` pour lancer l'algorithme A*.
4. **Mettre en pause** : Cliquez sur le même bouton (devenu `Pause`) pour suspendre la simulation.
5. **Observer le processus** : L'agent (représenté par une icône) se déplace à travers la grille en explorant les différentes possibilités.

4.2.3 Comprendre l'affichage

Durant l'exécution de l'algorithme A* illustrée dans la Figure 4 :
— Les nombres affichés dans chaque cellule représentent la valeur F ($F = G + H$) de la cellule dans l'algorithme A*
— Les cellules en noir sont des cellules déjà explorées (liste fermée)
— Les cellules en bleu représentent le chemin optimal actuel

— L'icône de l'agent se déplace pour montrer la progression de l'exploration

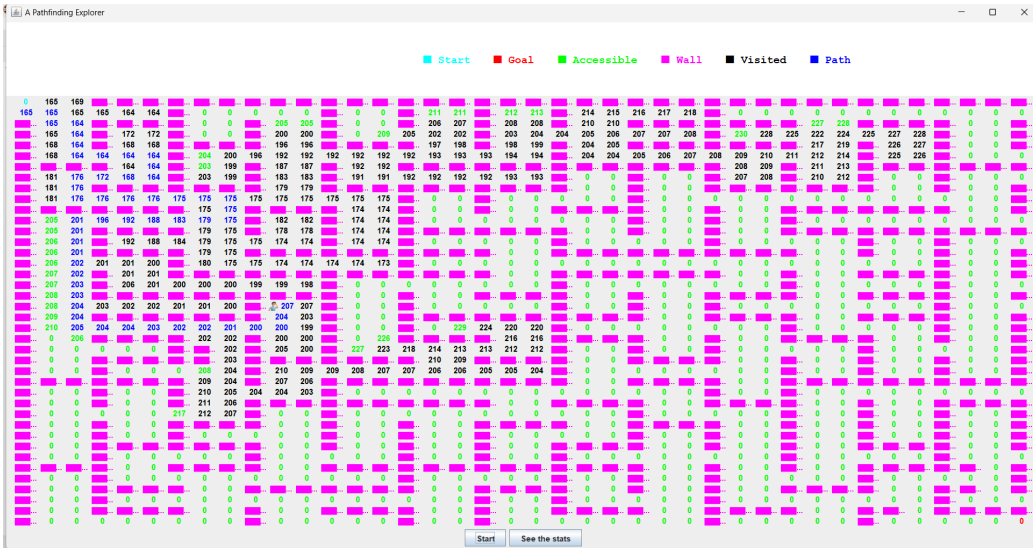


FIGURE 4 – Exécution de l'algorithme A* avec valeurs F affichées

4.3 Explorateur de chemins par Perceptron

4.3.1 Interface utilisateur

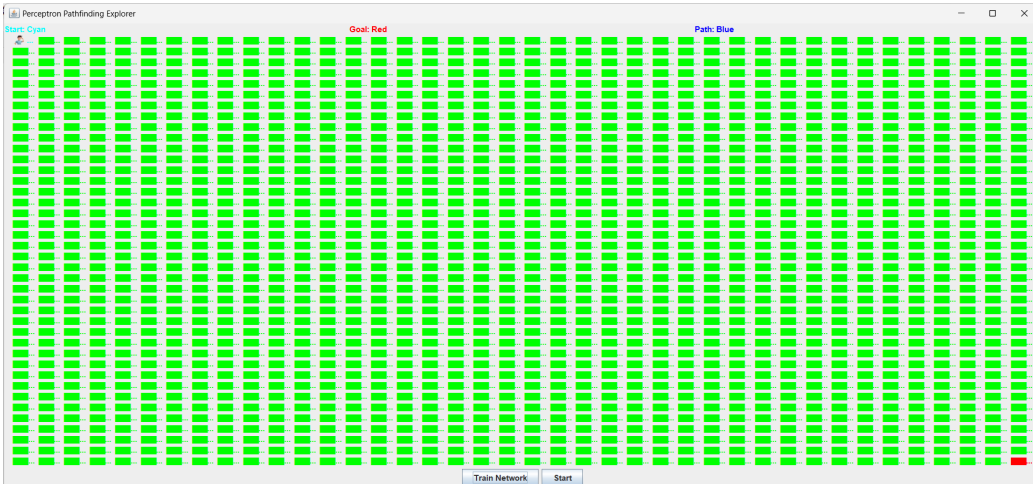


FIGURE 5 – Interface de l'explorateur Perceptron

L'interface de l'explorateur Perceptron présentée dans la Figure 5 comprend :

1. Un panneau d'information en haut
2. Une grille centrale affichant l'environnement
3. Deux boutons en bas : `Train Network` et `Start/Pause`

4.3.2 Utilisation

Pour utiliser l'explorateur Perceptron :

1. **Lancer l'application** : Exécutez la classe `PerceptronGUI` pour démarrer l'application.
2. **Comprendre la grille** :
   — Les cellules en cyan représentent le point de départ
   — Les cellules en rouge représentent l'objectif à atteindre

— Les cellules en vert sont les chemins accessibles
— Les cellules en magenta représentent les obstacles (murs)

3. **Entraîner le réseau** : Cliquez sur le bouton `Train Network` pour entraîner le perceptron.
4. **Démarrer la simulation** : Après l'entraînement, cliquez sur `Start` pour voir le chemin trouvé.
5. **Observer le processus** : L'agent se déplace le long du chemin prédit par le réseau neuronal.

4.3.3 Entraînement du réseau

La figure 6 montre qu'avant de pouvoir simuler un chemin avec le perceptron, vous devez entraîner le réseau neuronal :

1. Cliquez sur le bouton `Train Network`
2. Un message confirme que le réseau a été entraîné avec 1000 itérations
3. Si l'entraînement échoue ou si aucun chemin n'est trouvé, un message d'avertissement s'affiche

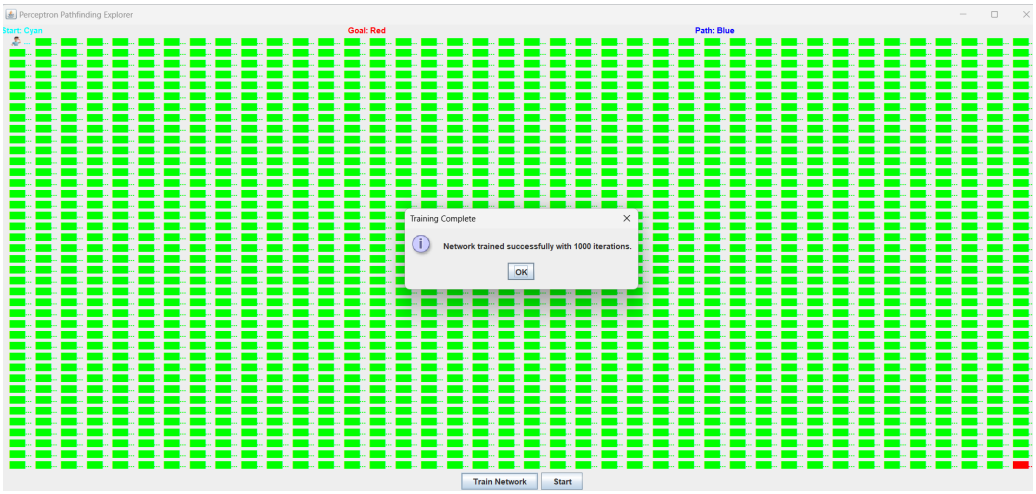


FIGURE 6 – Message de confirmation d'entraînement du réseau

L'entraînement est essentiel car le perceptron doit apprendre les directions optimales pour chaque cellule de la grille. Dans cette implémentation, le système utilise 1000 itérations d'entraînement avec un taux d'apprentissage de 0.01.

4.4 Fonctionnalités communes

4.4.1 Confirmation de sortie

Lorsque vous tentez de fermer l'une des applications, une boîte de dialogue de confirmation apparaît comme dans la figure 7 :

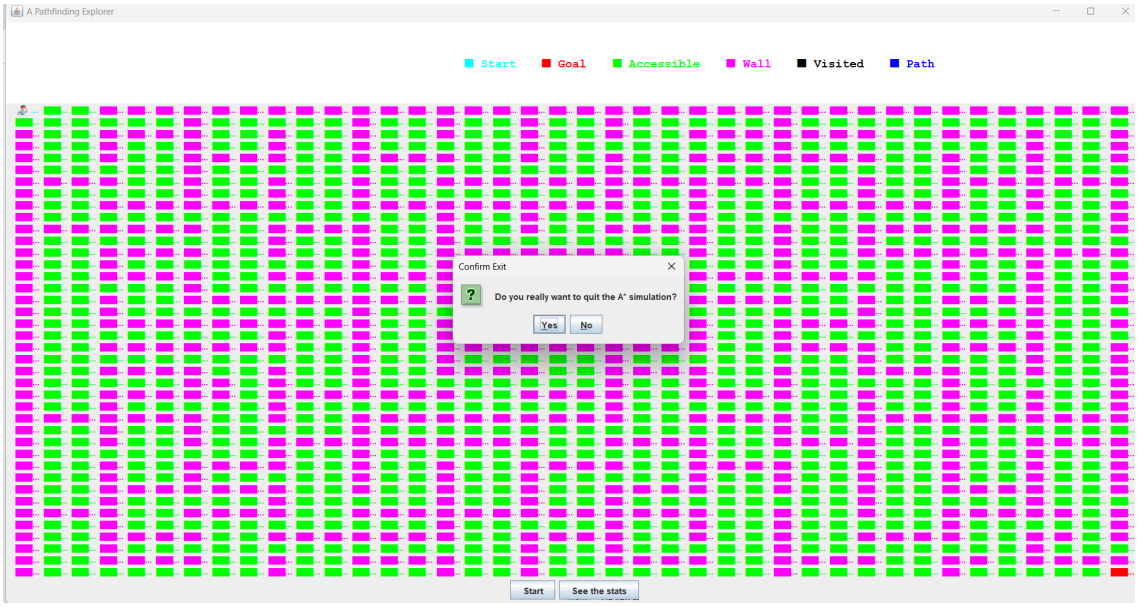


FIGURE 7 – Boîte de dialogue de confirmation de sortie

Cliquez sur `Yes` pour quitter l'application ou `No` pour continuer à l'utiliser.

4.4.2 Message de félicitations

Lorsque l'agent atteint l'objectif dans l'une ou l'autre des applications, un message de félicitations s'affiche comme dans la figure 8 :

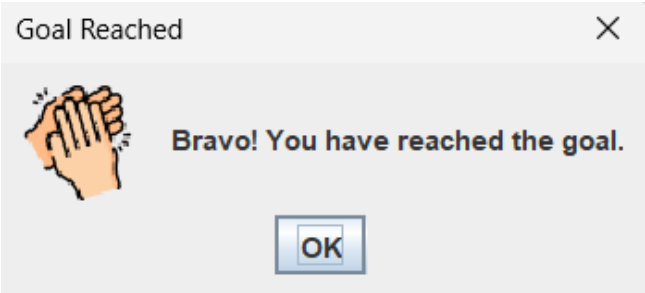


FIGURE 8 – Message de félicitations lorsque l'objectif est atteint

Ce message confirme que l'algorithme a réussi à trouver un chemin jusqu'à l'objectif et s'affiche avec une icône d'applaudissement.

4.4.3 Affichage des flèches directionnelles (Perceptron)

Dans la figure 9, les cellules accessibles affichent des flèches directionnelles ($\uparrow$, $\downarrow$, $\leftarrow$, $\rightarrow$) qui indiquent la direction prédite par le réseau neuronal pour atteindre l'objectif depuis cette cellule. Ces indications visuelles permettent de comprendre le comportement du réseau neuronal après son entraînement.

L'orientation des flèches montre comment le perceptron a "appris" à naviguer dans l'environnement, permettant ainsi de visualiser la politique de déplacement que le système a développée lors de son entraînement.

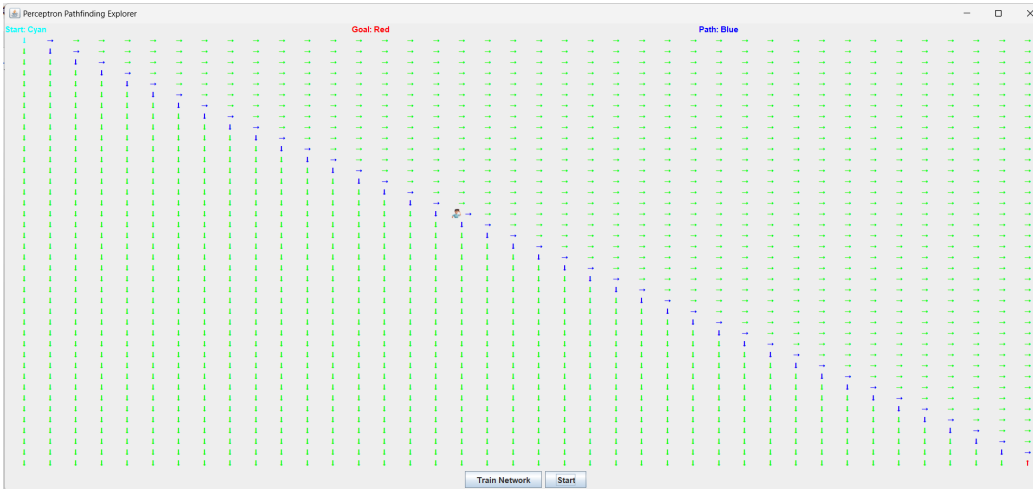


FIGURE 9 – Affichage des flèches directionnelles dans l'explorateur Perceptron

4.4.4 Affichage des statistiques

L'explorateur A* offre la possibilité de consulter des statistiques sur l'exécution de l'algorithme :

1. **Accéder aux statistiques** : Cliquez sur le bouton `Voir les statistiques` situé en bas de l'interface.
2. **Fenêtre des statistiques** : Une nouvelle fenêtre s'ouvre et affiche différentes métriques de performance :

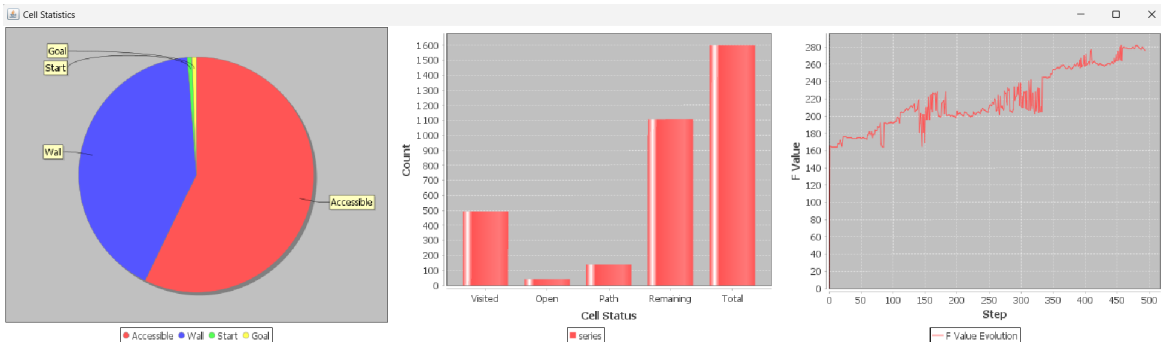


FIGURE 10 – Fenêtre des statistiques d'exécution

Dans la figure 10, les statistiques incluent :

— Un diagramme circulaire (camembert) qui représente la répartition des types de cellules : accessibles, murs, point de départ et point d'arrivée.

— Un diagramme en barres qui montre le nombre de cellules visitées, ouvertes, sur le chemin, restantes, et le total de cellules.

— Un graphique en courbe qui trace l'évolution de la valeur F au fil des étapes pendant l'exploration.

La fenêtre des statistiques peut être fermée à tout moment sans affecter la simulation en cours.

5 Déroulement du projet

Dans cette section, nous décrivons comment le projet a été réalisé en équipe : la répartition des tâches, la synchronisation du travail entre les membres de l'équipe, ainsi que les méthodes de communication et d'intégration utilisées.

5.1 Réalisation du projet par étapes

Le projet a été réalisé en plusieurs étapes successives afin de garantir une progression structurée et cohérente :

1. **Analyse du besoin et cadrage du projet** : nous avons commencé par définir les objectifs du projet, les fonctionnalités attendues et les contraintes techniques (grille, algorithmes à intégrer, interface graphique, etc.).
2. **Conception des classes et architecture générale** : une phase de modélisation a permis de concevoir les classes principales (comme `ACell`, `AStarProcessor`, `PGrid`, etc.) et d'organiser l'architecture logicielle.
3. **Développement des algorithmes** : l'algorithme A* a été développé et testé en premier, suivi de l'intégration du réseau de neurones avec perceptron.
4. **Création de l'IHM graphique** : une interface utilisateur a été mise en place, avec une fenêtre principale et différents composants pour afficher la grille, interagir avec les algorithmes et consulter la console.
5. **Tests et ajustements** : chaque fonctionnalité a été testée individuellement avant l'intégration globale. Des cas d'utilisation ont été simulés pour valider la robustesse du système.
6. **Finalisation et documentation** : le projet a été nettoyé, documenté, et préparé pour la soutenance.

5.2 Répartition des tâches, communication et intégration

Table 2 – Répartition des tâches entre les membres de l'équipe

Tâche	**ACHAB Ouardia**	**ISSAD Lisa**	**DE ANGELIS Enzo**
Analyse des besoins	✓	✓	✓
Conception des classes	✓	✓	
Développement de l'algorithme A*	✓	✓	
Développement du Perceptron	✓	✓	
Interface graphique AStarGUI	✓	✓	
Interface graphique PerceptronGUI	✓	✓	
Tests unitaires	✓	✓	
Spécification du projet	✓	✓	
Rédaction du rapport	✓	✓	
Organisation des réunions	✓	✓	
Communication d'équipe	✓	✓	
Gestion de version (GitHub)			✓

Pour assurer la synchronisation, nous avons utilisé :
- Une plateforme de gestion de version (GitHub) pour le partage du code.
- Des réunions hebdomadaires (en présentiel ou à distance) pour faire le point sur l'avancement.
- Un canal de communication instantanée (Discord/WhatsApp) pour résoudre les problèmes rapidement.

Comme présenté dans le tableau 2, l'intégration a été continue : chaque partie développée était testée de manière autonome, puis intégrée au fur et à mesure dans le projet principal. Cela a permis d'éviter les conflits et de garantir une cohérence globale du système.

6 Conclusion et perspectives

Dans cette section, nous résumons la réalisation du projet et nous présentons également les extensions et améliorations possibles du projet.

6.1 Résumé du travail réalisé

Ce projet nous a permis de mettre en œuvre plusieurs concepts fondamentaux de l'intelligence artificielle et du développement logiciel atteignant ainsi un achèvement global de 78%. Nous avons conçu une application interactive permettant de visualiser et comparer différentes approches de résolution de cheminement dans une grille (A*, Perceptron).

Sur le plan technique, nous avons :

- Implémenté une structure de grille générique compatible avec différents algorithmes.
- Programmé l'algorithme A* pour une recherche de chemin optimale.
- Développé un réseau de neurones simple (perceptron) capable d'apprendre les directions à suivre en fonction de son environnement.
- Intégré une interface graphique conviviale permettant d'interagir avec les différentes fonctionnalités du système.
- Utilisé des outils de gestion de projet (Git, réunions régulières) pour assurer une bonne coordination entre les membres de l'équipe.

6.2 Améliorations possibles du projet

Bien que le projet soit fonctionnel, plusieurs améliorations pourraient être envisagées pour le perfectionner et l'enrichir :

- **Ajout de nouveaux algorithmes** : comme Dijkstra, QLearning, ou encore ACO (Ant Colony Optimization) pour enrichir les possibilités de comparaison.
- **Optimisation des performances** : en améliorant la gestion mémoire, en réduisant le temps de traitement pour les grandes grilles ou en parallélisant certaines parties.
- **Entraînement du perceptron sur un dataset externe** : pour tester la robustesse de l'apprentissage supervisé dans un contexte plus complexe.
- **Personnalisation de la grille** : permettre à l'utilisateur de dessiner ses propres labyrinthes avec des outils visuels simples.
- **Support multiplateforme ou web** : adapter l'application à un environnement mobile ou à une interface web pour faciliter l'accès et la démonstration.

Ces perspectives ouvrent la voie à un projet évolutif et potentiellement exploitable dans un contexte pédagogique, expérimental ou même professionnel.

Références

[Cor25] Oracle Corporation. The java™ tutorials – creating a gui with swing. `https://docs.oracle.com/javase/tutorial/uiswing/`, 2025. Consulté en avril 2025.

[Gé19] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, Sebastopol, CA, USA, 2nd edition, 2019.

[Pat25] Amit Patel. Amit's a* pathfinding tutorial. `https://www.redblobgames.com/pathfinding/a-star/`, 2025. Consulté en avril 2025.

[RN10] Stuart Russell and Peter Norvig. *Artificial Intelligence : A Modern Approach*. Pearson, Upper Saddle River, NJ, USA, 3rd edition, 2010.

[Uni25] CY Cergy Paris Université. Cours de génie logiciel – licence 2 informatique, 2025. Supports de cours internes.