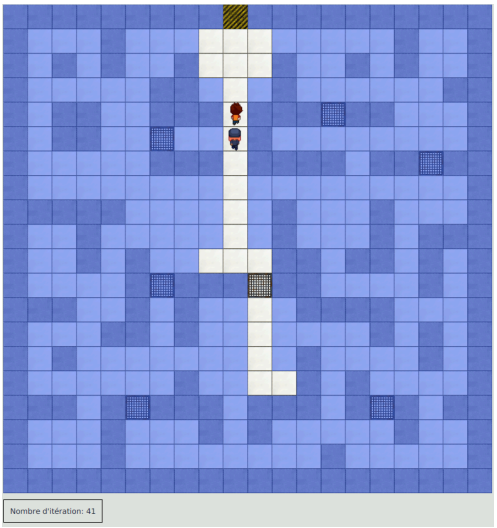
Rapport de fin de semestre
Poursuite-Évasion IA



Tuteur: Guénaël Cabanes

Matias Amaglio
Maëlle Bitsindou
Luc Dechezleprêtre
Célie Ponroy

Année universitaire 2024- 2025

Table des matières

I. Introduction

Dans le cadre de la 3ème année de BUT Informatique, ce document a pour objectif de présenter le déroulement du projet tutoré intitulé “Poursuite-Évasion IA”. Ce projet a été réalisé entre octobre 2024 et avril 2025, et encadré par M. Guénaël Cabanes.

Notre projet est une simulation de poursuite/évasion compétitive entre agents informatiques intelligents. La simulation prend place dans un labyrinthe ayant une seule sortie, dans lequel deux agents sont placés. Un agent incarne le rôle du "prisonnier", dont le but est de s'échapper du labyrinthe sans se faire attraper par le "gardien", qui lui, a pour but d'attraper le prisonnier. Les deux agents ont la même vitesse de déplacement, mais il existe dans le labyrinthe des bonus/malus octroyant un avantage/désavantage temporaire. La simulation s'arrête lorsque l'un des agents atteint son but, si le nombre d'itérations imparti est dépassé, alors la partie est déclarée nulle.

L'idée derrière ce projet est de nous permettre d'explorer le domaine de l'intelligence artificielle, tout en produisant un logiciel permettant aux utilisateurs de découvrir, à leur tour, les différents algorithmes et méthodes utilisés en IA. C’est pour cela que l’utilisateur peut choisir entre un mode interactif, où il incarne l’un des deux agents, et un mode non-interactif, dans lequel il observe la partie et accède aux données utilisées par l’IA. Pour familiariser l’utilisateur avec notre application, nous avons ajouté un tutoriel expliquant les mécaniques du jeu. Il est également possible de choisir un mode “Analyse”, offrant une vue globale sur un nombre de parties. Enfin, l’utilisateur peut revoir les parties qu’il a déjà jouées grâce au mode “Parties sauvegardées”..

Page d'accueil de l’application

Dans notre application, la base du comportement intelligent de nos agents est l'inférence bayésienne qui est une méthode pour calculer les probabilités de diverses causes à partir d'observations. En effet, les agents sont capables de se voir s'ils sont assez proches. De la même manière, ils sont capables de remarquer l'absence de leur adversaire dans leur champ de vision et donc augmenter la probabilité de présence dans les autres endroits de la carte. Nous avons deux grands types d'agents intelligents : les arbres de décision qui n'ont qu'une suite de conditions pour se déplacer et ne requièrent pas d'apprentissage ; et les réseaux de neurones qui apprennent de manière autonome selon un but prédéfini.

I. Analyse

Depuis les premières itérations, notre projet a beaucoup avancé. Lors de l'étude préalable, nous avons établi différentes conditions de validations que voici:

- ➢ Version minimale du produit:
  - Prendre le contrôle d'un des agents
  - Accéder à la vue de l'historique
  - Accéder au mode non interactif de la simulation

- ➢ Version médium du produit:
  - Obstacles
  - Avoir des niveaux de difficultés
  - Accéder au mode intéractif de la simulation

- ➢ Version avancée du produit:
  - Temps réel
  - Des cartes aléatoires
  - Point de départ aléatoire
  - Avoir plusieurs IA (à voir selon le temps d'apprentissage)

Plusieurs fonctionnalités ont été abandonnées depuis l'étude préalable:

- Temps réel:
  - Nous avons décidé de ne pas inclure cette fonctionnalité, car sa mise en place aurait nécessité des modifications importantes dans plusieurs parties de l'application. Nous avons estimé qu'il serait plus pertinent de consacrer notre temps à d'autres ajouts qui apportent davantage de valeur.
- Cartes aléatoires:
  - Cette fonctionnalité n'était finalement pas compatible avec les réseaux de neurones, ils sont entraînés pour une carte et ne sont donc pas performant sur une autre.
- Plusieurs IA
  - Pour nous différencier de sujet de projet similaire, nous avons fait le choix d'enlever cette partie.

Malgré quelques fonctionnalités non implémentées, notre projet est resté fidèle à notre vision établie lors de la phase d'étude préalable. Bien que nous ayons eu du retard sur l'implémentation du réseau de neurones dans l'application, nous avons quand même pu

ajouter des fonctionnalités supplémentaires à nos versions de base dans notre application comme l'ajout de sauvegarde, des comportements aléatoires, d'un tutoriel, de crédits et d'un ajout de musique.

II. Réalisation

Lors de la réalisation de notre projet, nous avons cherché à structurer notre travail de manière claire et efficace. Nous avons défini une architecture adaptée, mis en place des tests empiriques pour valider nos choix techniques et notamment en ce qui concerne l'intégration d'un réseau de neurones.

➢ Architecture logicielle :

Notre application possède une architecture en MVC (Modèle - Vues - Contrôleurs). Ce choix repose sur le fait que notre application n'a pas besoin de communiquer avec des services extérieurs. Mais nécessite tout de même une interface graphique car elle a un but éducatif et ludique. De plus, du point de vue de la programmation, cela nous permet de séparer plus clairement le code et de travailler tous ensemble sans se déranger.

➢ Tests de validations :

Pour valider le développement de l'inférence bayésienne, qui est une partie importante de notre application car plusieurs algorithmes utilisés pour les agents se basent dessus. Nous avons choisi d'avoir une confirmation visuelle plutôt que de tester avec une bibliothèque dédiée aux tests unitaires, car les calculs de probabilités sont faisables à la main, mais la comparaison et les tests d'égalité entre des réels ne sont pas toujours évidents en Java. Donc, nous avons développé en priorité une vue permettant une confirmation visuelle de nos calculs, ce qui a permis à notre tuteur de valider nos méthodes d'inférence bayésiennes.

➢ Difficultés rencontrés : Réseau de neurones

Nous avons rencontré plusieurs difficultés lors de l'apprentissage des réseaux de neurones pour nos agents.

Dans un premier temps, nous avons utilisé une bibliothèque fournie par l’un de nos enseignants (M.Boniface). L’objectif initial était d’entraîner le réseau à reproduire un arbre de décision destiné au rôle du gardien. Cependant, avec cette bibliothèque, le réseau de neurones

ne présentait aucun comportement intelligent, malgré la variété des données entrées, il produisait systématiquement la même sortie.

Nous avons alors décidé de changer de bibliothèque et de nous tourner vers Neuroph. Malheureusement, les résultats sont restés similaires, le gardien se déplaçait toujours dans la même direction. Par ailleurs, les possibilités de configuration du réseau étaient assez limitées.

Nous avons donc poursuivi nos recherches et adopté une bibliothèque plus complète : DJL. Les problèmes rencontrés étaient comparables à ceux des précédentes bibliothèques, bien que les résultats aient été légèrement plus encourageants. Nous avons choisi de conserver DJL, mais en modifiant cette fois le format des données utilisées pour l'entraînement.

Initialement, nous utilisons un vecteur unidimensionnel représentant la carte bayésienne, où chaque case indiquait la probabilité de présence de l'autre agent (et -1 pour les murs), complété par les coordonnées X et Y de l'agent.
Nous avons alors adopté un nouveau format de vecteurs unidimensionnels composé de :

- La carte "classique", telle que décrite dans nos fichiers texte,
- La carte bayésienne, sans les -1 pour les murs,
- Une carte remplie de zéros, avec un 1 à la position correspondant aux coordonnées X et Y de l'agent.

Ce changement de représentation des données a permis au réseau de neurones d'extraire des informations bien plus pertinentes, ce qui a considérablement amélioré les performances, faisant passer le taux de réussite de 30 % à 98 %.

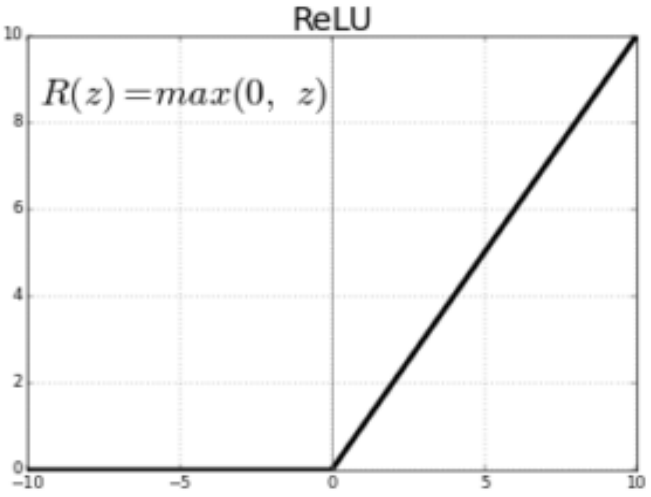
➢ Différents réseaux de neurones et apprentissages

Au cours de notre projet, nous avons développé différents types de réseaux de neurones et expérimenté plusieurs méthodes d'apprentissage.

Nous avons notamment réalisé un MLP (Multi-Layer Perceptron), qui est un réseau de neurones dit "classique". Il permet d'approximer des fonctions complexes, difficilement définissables par des méthodes traditionnelles. Dans le cadre de l'apprentissage de l'arbre de

décision, la fonction cible du réseau consistait à choisir un déplacement en fonction des différentes cartes fournies en entrée.

Le MLP est composé de neurones, qui sont des blocs recevant des entrées et produisant une sortie après application d'une fonction mathématique. La fonction d'activation que nous avons utilisée est la ReLU (Rectified Linear Unit), qui transforme toutes les valeurs négatives en zéro et laisse les autres inchangées :



Courbe de la fonction ReLU

La dernière couche du réseau subit ensuite un traitement supplémentaire avec la fonction Softmax. Cette fonction transforme les résultats bruts en un ensemble de probabilités dont la somme est égale à 1. Cela facilite l'utilisation de la fonction de perte Cross-Entropy, qui mesure la différence entre la distribution prédite par le modèle et la distribution réelle des classes. Grâce à cela, nous pouvons interpréter les sorties du réseau comme des pourcentages de chance que chaque déplacement soit le bon. La sortie finale du réseau est un tableau dont la taille correspond au nombre de déplacements possibles ; l'index associé à la probabilité la plus élevée indique le déplacement choisi par l'agent.

Par ailleurs, nous avons également expérimenté avec un CNN (Convolutional Neural Network) pour l'apprentissage de l'arbre de décision.

Les CNN sont des architectures de réseaux de neurones initialement conçues pour le traitement d'images, mais qui s'avèrent tout aussi efficaces sur des données structurées comme nos cartes de jeu. Leur principal avantage réside dans leur capacité à détecter automatiquement des motifs locaux grâce à l'utilisation de filtres (ou kernels). Ces filtres

“glissent” sur les données d'entrée (dans notre cas, les cartes du jeu) et effectuent des multiplications locales pour extraire des caractéristiques importantes, comme la présence d’un mur à proximité ou la position relative de l’adversaire. Après cette phase de convolution, nous appliquons une opération appelée max-pooling. Cette étape consiste à réduire la taille des données tout en conservant les informations les plus importantes, en prenant le maximum d’une petite région de la carte traitée. Cela permet d’alléger les calculs tout en conservant les caractéristiques essentielles détectées par les filtres. Une fois ces opérations réalisées, les données ainsi transformées sont "aplaties" pour pouvoir être traitées par la suite par un MLP de configuration similaire à celui que nous utilisions précédemment.

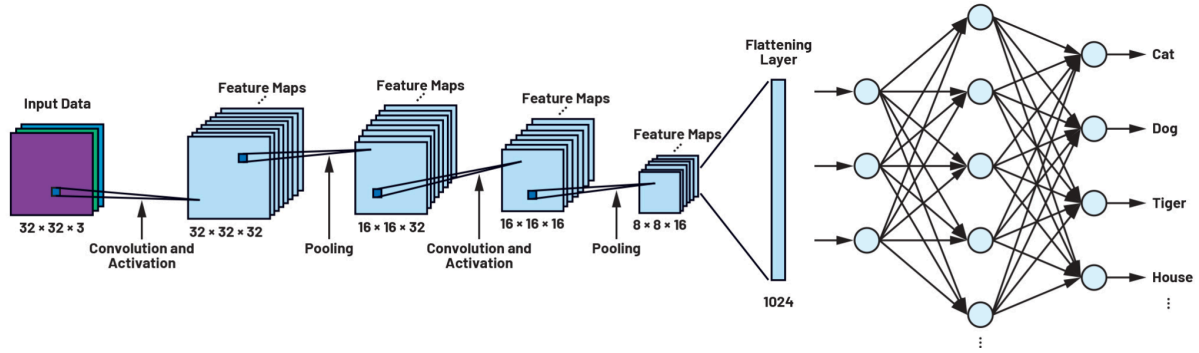


Schéma d’un réseau de neurones à convolution

Nous avons également utilisé un autre type d'apprentissage, l'apprentissage par renforcement, et plus particulièrement le Q-Learning.

Le Q-Learning consiste à permettre à un agent d’apprendre à prendre des décisions en interagissant directement avec son environnement, sans supervision externe directe. Plutôt que d’apprendre à partir de données étiquetées, l’agent expérimente différentes actions et reçoit des récompenses ou des pénalités selon les conséquences de ses choix. L’objectif est de construire progressivement une fonction de valeur Q, qui associe à chaque couple (état, action) la valeur estimée de la récompense future que l’agent peut espérer en suivant cette action. À chaque nouvelle interaction avec l’environnement, la valeur Q est mise à jour en fonction de la récompense obtenue et de l’estimation de la meilleure action possible depuis le nouvel état.

La mise à jour de la fonction Q suit la formule suivante :

$$Q(s, a) \leftarrow Q(s, a) + r + \gamma * \max Q(s', a')$$

Avec :
s : état actuel
a : action choisie
r : récompense immédiate reçue après l'action
s' : état suivant après avoir effectué l'action
γ : facteur d'actualisation, qui pondère l'importance des récompenses futures

Un état S est un vecteur unidimensionnel comprenant la position X et Y de l'agent sur la carte, ainsi que la carte "classique". Par manque de temps, la carte bayésienne n'a pas été prise en compte lors de l'apprentissage.

Ce réseau est entraîné de manière à réduire l'écart entre ses prédictions et la cible calculée à partir de l'expérience acquise, c'est-à-dire la somme de la récompense immédiate et de la valeur future estimée.

III. Planning de déroulement du projet durant toutes les itérations

Dès le début de ce projet tutoré, nous avons établi un planning de base, en prenant en compte les aptitudes et disponibilités de chaque membre. Le voici mis à jour de l'itération 1 à 7.

Itération 1:

- Base du projet (Célie)
- Base du moteur de jeu (Célie & Luc)
- Création de la vue du menu (Maëlle)
  L'utilisateur va pour voir choisir un des modes. (dans cette itération, seulement le mode interactif)
- Création de la vue de la simulation interactive (Maëlle)
  L'utilisateur peut voir son avancement et celui de l'adversaire sur cette vue.
- Mise en place de l'inférence bayésienne (Matias & Luc)

L'utilisateur va pouvoir connaître les probabilités de position de l'agent adverse.

- Ajout de la vision des personnages (Célie)
- Création de l'arbre de décision du gardien (Célie)

  Cette fonctionnalité n'a pas été terminée.
- Ajout du contrôle du prisonnier (Luc)

  L'utilisateur qui prendra contrôle du prisonnier et pourra se déplacer sur toute la carte.

Itération n°2:

- Création de l'arbre de décisions du gardien et du prisonnier (Célie)

  Le gardien et le prisonnier ont maintenant un comportement automatique.
- Ajout de calculs de chemins (Célie)

  Pour pouvoir implémenter les deux arbres il fallait qu'ils puissent trouver un chemin pour aller d'une case A a une case B.
- Ajout de l'historique (mode interactif) (avec vues) (Célie & Maëlle)

  Dans le mode interactif, l'utilisateur va pouvoir consulter après sa partie tous les déplacements de chaque personnage avec la réflexion de son adversaire sur sa position.
- Implémentation vue non interactive (Maëlle)

  Dans le mode non interactif, l'utilisateur va pouvoir consulter en même tant que la vue de la simulation principale, 2 vues pour voir chacun des personnages avec leur vue bayésienne. Il est possible de se déplacer entre les différents tours.
- Refactorisation des Vues (Célie)

  Ajout de la classe *VueSimulation* pour refactoriser le code
- Ajout des messages d'alerte de fin de partie (Maëlle)

  A chaque fin de partie dans le mode interactif , l'utilisateur va voir un message d'alerte lui indiquant que la partie est terminée et si il a gagné ou non.
- Mise en place réseau de neurones (Matias & Luc)
- Mise en place de l'apprentissage de l'arbre (Matias & Luc)

  Apprentissage du réseau de neurones des décisions de l'arbre de décision
- Lancement de l'apprentissage dans le terminal (Matias & Luc)

  Choix du nombre d'itération , le nombre de couche et leur taille
- Enregistrement du réseau de neurones dans un fichier (Matias & Luc)

Itération n°3 (Attention installer Neuroph ⚠️):

- Arbre de décision du Prisonnier amélioré (Arbre de décision 2.0) (Célie)

  Le premier arbre de décision était trop simple, nous avons donc amélioré sa façon de fuire.
- Amélioration de la vision (Célie)

  Des cases étaient visibles mais pas "naturelles", il a dû alors filtrer les cases voulues.
- Ajout de raccourcis pour le gardien (Célie)

  Comme le prisonnier gagnait trop facilement , nous avons ajouté un avantage pour le gardien. Il peut maintenant traverser des cases spéciales.(représentés par des grillages)
- Ajout d'un comportement aléatoire (Célie)

  Nous avons ajouté un comportement (pour les deux personnages) qui choisit au hasard où il va se diriger.
- Amélioration de l'interface (Célie et Maëlle)

  -Ajout d'un style.css qui va rendre l'interface plus agréable en général

  -Modification des ratio de l'application due à la modification de la carte

  -Ajout de boutons pour revenir au menu principal à partir de la partie
- Ajout du menu de choix de difficultés (Maëlle)

  Le joueur peut maintenant choisir la difficulté de l'IA de la partie parmi nos différentes versions, que ce soit en mode interactif ou en non interactif.
- Ajout du spawn aléatoire pour le gardien et le prisonnier.

  Les deux personnages apparaissent aléatoirement sur la carte (Luc)
- Ajout d'une classe permettant de charger la carte à partir d'un fichier texte (Luc)
- Correction de l'inférence bayésienne qui prend maintenant en compte la capacité de déplacement de l'autre agent (Matias)
- Implémentation de la librairie Neuroph dans le projet (Luc & Matias)
- Paramétrage des entrées du réseau de neurone en fonction de la taille de la carte (Matias & Luc)

Itération n°4 (Attention ajouter la librairie DJL ⚠️) :

- Implémentation de la librairie DJL dans le projet (Luc & Matias)

- Apprentissage de l’arbre de décision du gardien par le réseau de neurone (Matias & Luc)
- Apparition aléatoire de la sortie sur les cases désignées (Luc)
- Création d’un outil pour tenter d’optimiser les hyper-paramètres (Matias)
- Mise en place d’un système de sauvegarde des parties (Célie)
- Mise en place d’un mode analyse (Maëlle)
- Amélioration de l’affichage (Célie)

Itération n°5:

- Ajout d’un tutoriel (Célie)
- Amélioration graphique (Célie & Maëlle)
- Ajouts de sprite directionnels (Célie)
- Amélioration de la Vue analyse (Maëlle)
- Ajout d'une couche CNN au réseau de neurones (Matias)
- Modification du format de données d'apprentissage et d'entré des réseaux de neurones (Matias & Luc)
- Apprentissage de l’arbre de décision du gardien par les réseau de neurone (Matias & Luc)
- Ajout d'une Vue "d'informations" sur les différents comportements (Célie)

Itération n°6 (Attention ajouter la librairie Media ⚠️):

- Ajout de caméras pour le gardien (Célie)
- Ajout Page d'Accueil(Célie)
- Ajout de page de Crédits (Célie)
- Débug de Sauvegarde (Célie)
- Ajout de musique (Maëlle)
- Modification de l’analyse (Maëlle)
- Amélioration Interface (Célie & Maëlle)
- Optimisation des paramètres MLP(Luc)
- Optimisation des paramètres CNN(Matias)
- Refactoring choix comportement Simulation (Matias)
- Début refactoring MLP et CNN (Matias)

Itération n°7 :

- Modifications de l'analyse (Maëlle)
- Ajout du bouton muet de l'application (Maëlle)
- **ug** et Optimisation et Refactorisation (Célie, Maëlle & Matias)
- Amélioration de l'interface (Gestion du plein écran, des crédits) (Célie)
- Préparation de la présentation aux première années (Célie)
- Apprentissage par renforcement sans carte bayésienne (Matias & Luc)

IV. Répartition du travail entre étudiants

Pour une meilleure efficacité, nous avons décidé dès le début du projet de nous répartir les différentes tâches à faire en fonction de nos envies et aptitudes.

Célie a commencé par structurer le projet et a contribué tout au long de son développement à l'application des principes SOLID, optimisant ainsi l'architecture du projet. Par exemple, elle a refactorisé les vues en ajoutant la classe **VueSimulation**, ce qui a permis d'éviter la duplication de code. La majorité des tâches de Célie étaient orientées vers le backend de l'application. Elle a notamment travaillé sur le calcul des chemins avec l'implémentation de l'algorithme A*, les calculs de la vision, l'ajout des arbres de décision, ainsi que des bonus pour le gardien, tels que l'ajout de caméras et de grilles auxquelles seul le gardien a accès. Elle a également mis en place la sérialisation de la simulation via la classe **SimulationSerializable**, permettant ainsi de sauvegarder les parties jouées, ainsi que l'interface associée. Ces contributions sont dues à la préférence de Célie pour la conception et le développement backend, ainsi qu'à ses compétences dans ces domaines. En outre, Célie a consacré du temps à améliorer l'affichage, notamment en ajoutant un fichier CSS au projet et en rendant l'application plus responsive. Elle a également contribué à l'amélioration de l'expérience utilisateur en ajoutant des panneaux d'information sur les différents comportements des IA, en implémentant un tutoriel du jeu, et en ajoutant des crédits de fin.

Ayant un attrait pour le développement frontend et la gestion de projet, Maëlle s'est principalement chargée de l'aspect graphique de l'application, dans le but d'offrir une meilleure expérience utilisateur. Elle a ainsi développé les différentes vues nécessaires au déroulement de la simulation, en concertation avec l'ensemble de l'équipe. Elle a également

pris en charge l'implémentation complète (frontend et backend) du mode d'analyse, permettant à l'utilisateur d'examiner un grand nombre de parties en détail. Ce mode repose principalement sur deux classes : **VueAnalyse** et **LancerAnalyse**.
Maëlle s'est aussi occupée de l'ajout et de l'intégration des musiques dans l'application, via la classe **SoundManager**. Lors de l'optimisation du code, Maëlle a également pu réaliser une classe, **FabriqueComportement** permettant de gérer mes différentes créations de comportements en fonction des situations.
Par ailleurs, elle s'est chargé de la rédaction des rapports de fin d'itération, de la préparation des présentations de soutenance, ainsi que de la réalisation de la vidéo de démonstration.

Matias et Luc se sont principalement concentrés sur le développement de l'inférence bayésienne et l'implémentation des réseaux de neurones dans le projet. Leur priorité a été de développer des méthodes de calcul de probabilités à l'aide de l'inférence bayésienne, car les différents comportements que nous souhaitions mettre en œuvre utilisent ces probabilités pour guider leurs choix. Leur travail inclut également la création du dataset, le chargement des cartes à partir de fichiers texte, ainsi que la recherche et la compréhension des bibliothèques nécessaires pour les réseaux de neurones. Ils ont également dû comprendre les algorithmes d'apprentissage, paramétrer le réseau, et effectuer l'apprentissage des réseaux.

Luc a pris en charge le chargement des cartes par fichier et a créé les premiers datasets. Matias, quant à lui, s'est occupé de la refactorisation lors de l'ajout des différents réseaux, ainsi que des réglages des bogues et de l'amélioration liés à l'inférence bayésienne.

Deux types d'apprentissage ont été utilisés pour les réseaux de neurones, ce qui a nécessité un formatage spécifique des données pour chaque type. Ce processus s'est avéré chronophage. Bien que la théorie derrière les méthodes d'apprentissage reste inchangée, leur implémentation varie considérablement d'une bibliothèque à l'autre. Cela a non seulement compliqué le paramétrage et la mise en place des réseaux, mais a également rendu le débogage plus complexe.

V. Présentation d'un élément original dont vous êtes fiers

➢ Luc :

Ce dont je suis le plus fier dans ce projet c'est la mise en place des réseaux de neurones de type perceptron multicouche (MLP). La mise en place en place s'est étalée sur plusieurs itérations et Matias et moi avons dû faire preuve de beaucoup d'abnégation. Le perceptron multicouche est la première configuration que nous avons voulu mettre en place pour apprendre le comportement d'un arbre de décision.

Dans un premier temps nous nous sommes concentrés sur des outils de création des différents datasets. Pour cela nous avons joué en boucle des parties (8000 pour un seul dataset) et nous avons enregistré chaque coup joué dans la partie dans un fichier CSV. Chaque ligne du dataset possède deux champs, un représentant l'état de la partie à un instant t, et un autre avec la décision qui a été prise par l'arbre de décision par rapport à l'état de la partie.

Le déploiement des réseaux de neurones dans notre application a été long et complexe car nous avons mis du temps à avoir un résultat convenable, et nous pensions à tort que le problème venait des bibliothèques utilisées. Avec le recul nous savons que le problème n'était pas nécessairement les bibliothèques utilisées mais plutôt les données mise en entrée des réseaux. En effet les premières entrées étaient uniquement la carte des probabilités bayésienne et la position du personnage (x et y), et lorsque nous avons décidé de changer, des résultats convaincants sont apparus. Désormais, au lieu d'avoir uniquement la carte bayésienne et la position du personnage nous avons une carte bayésienne, une carte représentant la position du personnage (même dimension que la carte normale mais avec des zéros partout sauf un 1 là où se trouve le personnage) et une carte avec uniquement les murs et la sortie.

Lorsque le réseau de neurones s'est mis à fonctionner correctement, nous avons eu des résultats très intéressants car le réseau de neurones faisait mieux que l'arbre de décision sur lequel il s'est entraîné. Car l'arbre à pour directive principale d'explorer lorsqu'il ne voit pas le prisonnier, alors que le réseau de neurones à tendance à se diriger vers le milieu de la carte, là où la vision est la meilleure et là où le prisonnier a le plus de chance de passer. Donc finalement le réseau de neurones attrape plus fréquemment le prisonnier.

➢ Maëlle :

Ma contribution dont je suis la plus fière dans ce projet a été de réaliser entièrement un mode d'analyse. Ayant un projet à visée éducative et ludique, nous avons décidé d'ajouter mode d'analyse. Ce mode permet à l'utilisateur de pouvoir analyser différentes données sur un nombre choisi de parties. Le challenge lors de l'implémentation de cette fonctionnalité a été de mettre en place l'affichage de chaque graphique et de la carte en temps réel, ainsi que le bouton de pause du lancement.

L'utilisateur peut sélectionner le niveau de difficulté de chaque personnage, ainsi que le nombre de parties souhaitées puis lancer l'analyse.

Vue du mode analyse

Comme on peut le voir sur l'image ci-dessus, les données que l'utilisateur va pouvoir examiner sont :

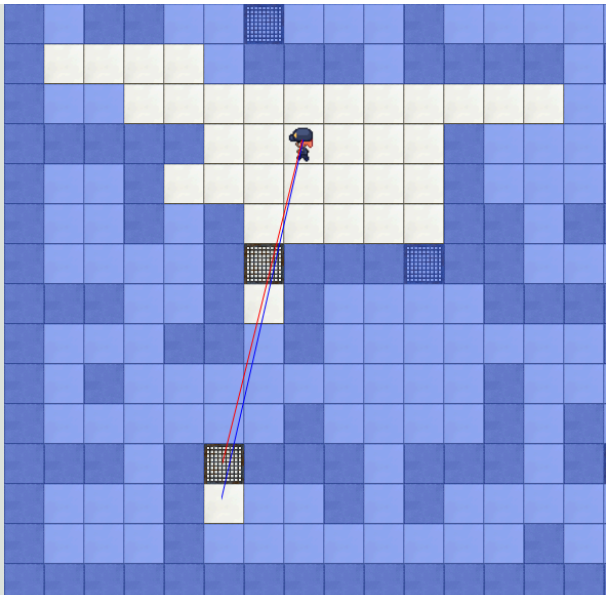
- le taux de victoire (gardien, prisonnier ou match nul),
- la carte des cases les plus utilisées ainsi que les apparitions de départ durant les différentes parties (prisonnier, gardien ou les 2)
- le nombre total de déplacements des personnages en fonction de chaque itération.

Ce mode a nécessité de mettre en place une nouvelle vue ***VueAnalyse***, qui contrôle tout l'aspect graphique de la vue ainsi que les mises à jour des différents éléments en temps réel. De plus, la classe ***LancerAnalyse*** a été développée pour calculer les données de sortie en fonction des entrées et générer chaque simulation

➢ Célie :

Ma contribution dont je suis la plus fière est le calcul de la vision des personnages. Étant donné que notre projet repose sur l'intelligence artificielle, il était important que ces calculs n'affectent pas les performances des apprentissages en consommant trop de ressources. Notre solution est d'effectuer les calculs au préalable et de stocker les résultats dans deux fichiers, car avec l'ajout des caméras, la vision du gardien se diffère de celle du prisonnier.

Pour calculer la vision, nous avons donc déterminé toutes les cases visibles à partir de chaque case de la carte. Pour ce faire, nous avons tracé une droite reliant le centre de la case en cours de calcul à celui de chaque case située dans un rayon défini par la portée de vision. Si la droite passe par une case mur, alors la case n'est pas visible, sinon elle l'est. Juste avec cette façon de calculer la vision , certaines cases qu'on ne voulait pas dans la vision était considérés :



Exemple de vision avant nettoyage

On peut voir sur l'image ci-dessus que les deux cases en dessous sont prises en compte car aucun mur n'est entre le personnage et eux, mais cette vision n'est pas naturelle. Alors j'ai rajouté un "nettoyage" des données a la fin des calculs. Ce nettoyage de données garde une case si il y a un chemin continue entre le personnage et la case qui passe uniquement par des cases vision. Si ce n'est pas le cas, cela veut dire qu' elle est excentrée et on les retire.

➢ Matias :

Ma contribution dont je suis le plus fier est le calcul de l'inférence bayésienne. L'inférence bayésienne permet d'estimer la probabilité de présence de l'adversaire sur chaque case. Cette fonctionnalité est fondamentale, car elle permet aux agents de connaître la position approximative de leur adversaire et de le voir lorsque la probabilité de présence atteint 1.

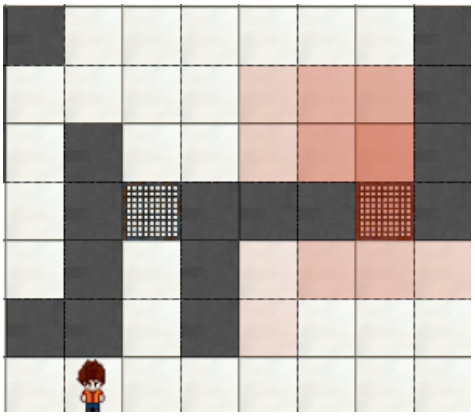
Le développement de cette inférence a été particulièrement difficile en raison de la complexité des tests. Comme les probabilités de présence sont souvent arrondies, il est difficile de prévoir avec exactitude la valeur retournée par le programme. Ainsi, tous les tests ont été effectués de manière empirique, ce qui a conduit à quelques corrections de bogues au fil des itérations.

La prise en compte des déplacements possibles de l'agent adverse a également été intégrée afin de rendre la probabilité de présence plus réaliste. Cependant, malgré ces ajouts, l'inférence repose sur une équiprobabilité des mouvements, ce qui signifie que la probabilité que l'agent explore l'espace est faible. En réalité, les agents ont tendance à explorer leur environnement davantage.

La probabilité de présence Pn+1 pour une case i,j et pour un tour n est calculés à partir de l'équation suivante:

$$P_{n+1}(i,j) = \sum_{(k,l)\in Voisins(i,j)} P_n(k,l)\cdot T((i,j)|(k,l)),$$

où $T((i,j)|(k,l))$ *est la probabilité de transition,* égale à $\frac{1}{NbVoisins(k,l)}$.



Exemple de probabilité de présence du point de vue prisonier

VI. Conclusion

Pour conclure, nous sommes très fiers du résultat final de notre projet, malgré les difficultés rencontrées tout au long de son développement. L'intégration des réseaux de neurones et les ajustements de l'inférence bayésienne ont notamment représenté des problèmes importants. Ces obstacles nous ont amenés à revoir nos approches, à expérimenter plusieurs solutions, et à renforcer nos compétences en intelligence artificielle et en optimisation d'algorithmes.

Grâce à une bonne organisation et une communication au sein de l'équipe, nous avons su respecter l'essentiel de nos objectifs initiaux, tout en enrichissant notre application avec des fonctionnalités supplémentaires qui la rendent plus complète et agréable à utiliser. Nous avons su adapter nos objectifs aux imprévus tout en gardant une cohérence dans la réalisation du projet.

Pour terminer, nous pensons que notre projet est très complet et d'un point de vue simplement développement applicatif il ne peut pas beaucoup évolué, bien que certains aspects peuvent être ajouté tels que un changement de carte ou encore d'autres cases spéciales. Il est encore possible de se pencher sur l'aspect intelligence artificiel notamment les réseaux de neurones avec l'aspect de compétition entre les deux IA que nous n'avons pas pu aborder et il serait également intéressant d'améliorer l'inférence bayésienne. Les arbres de décisions ne sont pas optimaux non plus, les améliorer serait également un bon ajout. Nous serions donc ravis qu'un groupe des années prochaines reprennent notre projet.

Annexes

Annexe 1 - Mode d'emploi pour installer notre application

Pour lancer notre application, deux options sont possibles.

Option n°1:

- Lancer l'application à l'aide du fichier .jar "projet_tut_poursuite-evasion.jar" qui se trouve dans le dossier application_finale_JAR sur GitHub.

Option n°2:

- Lancer l'application à l'aide d'un IDE (exemple: IntelliJ) et y ajouter les dépendances suivantes manuellement à l'aide de maven dans les librairies :
    - DJL :
        - ai.djl:api:0.28.0
        - ai.djl:model-zoo:0.28.0
        - ai.djl.mxnet:mxnet-engine:0.28.0
        - ai.djl.mxnet:mxnet-model-zoo:0.28.0
        - org.slf4j:slf4j-simple:1.7.36
        - ai.djl.basicdataset:0.28.0
    - Multimédia
        - javafx-media

Annexe 2 - Mode d'emploi de l'application

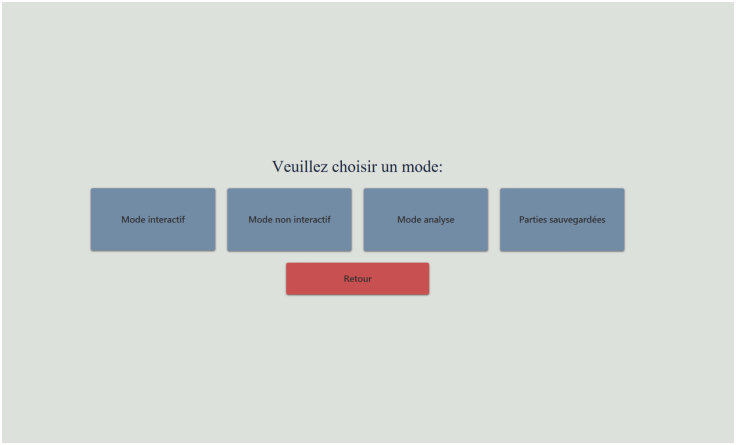
Au lancement de l'application cette page devrait apparaître :

Page d'accueil de l'application

Pour se familiariser avec l'application, nous vous conseillons de commencer par le tutoriel qui vous expliquera toutes les mécaniques à connaître. Sinon appuyez sur "Commencer" directement après le lancement de l'application.

Vous serez ensuite redirigé vers le menu principal, contenant 4 boutons différents :

- Mode interactif
- Mode non interactif
- Mode analyse
- Parties sauvegardées



Menu principal de l'application

Seul le mode “Parties sauvegardées” ne peut être lancé sans avoir au préalable joué et sauvegardé une partie, tous les autres modes sont indépendants.

NB: En cas d'interrogation sur les différents comportements, une icône d’information a été placée à côté de chaque choix de difficulté d’IA.

Affichage informatif des comportements