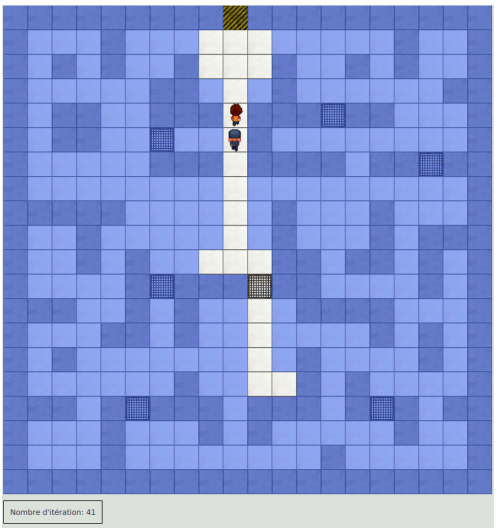
Rapport de fin de semestre

Simulation de poursuite-évasion compétitive entre agents informatiques intelligents



Tuteur: Guénaël Cabanes

Matias Amaglio
Maëlle Bitsindou
Luc Dechezleprêtre
Célie Ponroy

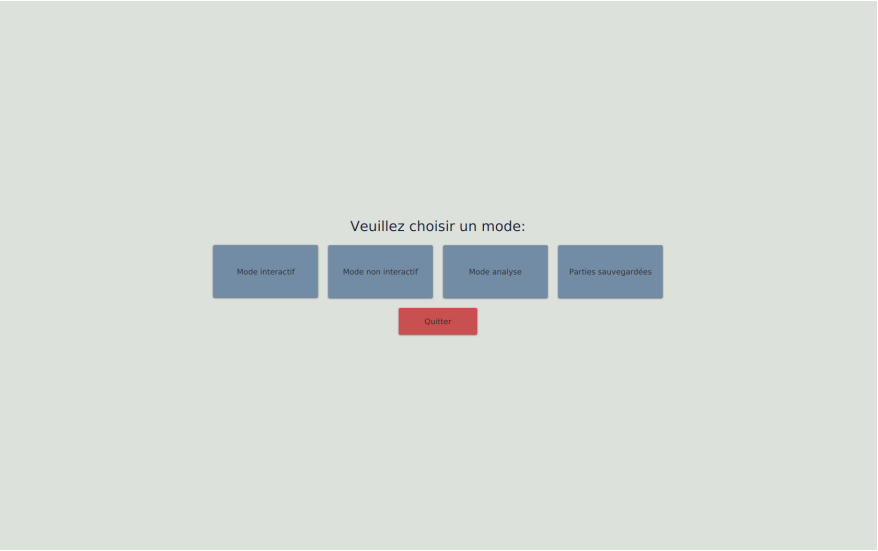
Année universitaire 2024- 2025

Table des matières

I. Introduction

Notre projet est une simulation de poursuite/évasion compétitive entre agents informatique intelligents. La simulation prend place dans un labyrinthe avec une seule sortie actuellement, dans lequel deux agents sont placés. Un agent a le rôle du “prisonnier”, dont le but est de s’échapper du labyrinthe sans se faire attraper par le “gardien” qui lui, à 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 se finit en match nul.

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 où il regarde le déroulement de la partie, en ayant accès aux données qui ont permis à l’agent intelligent de faire ses choix. En plus de ces deux modes il est également possible de choisir le mode “Analyse” qui permet à l’utilisateur d’avoir une vue globale sur un grand nombre de parties. Il est possible aussi de revoir ses parties déjà jouées via l’application.



Menu principale 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 si 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 ont juste une suite de conditions pour se déplacer et ne requiert pas d'apprentissage; et les réseaux de neurones qui apprennent de manière autonome selon un but.

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)

Après quatre itérations, nous pouvons constater que nous venons de commencer la version avancée, seule la fonctionnalité "avoir plusieurs IA" ne va pas être implémentée dans notre application finale. Mais jusqu'à présent seule la fonctionnalité avancée "Temps réel" n'a pas été entamée.

Notre projet est resté fidèle à notre vision établie lors de la phase d'analyse, malgré un retard pour l'implémentation du réseau de neurones dans l'application. Nous avons même pu ajouter des fonctionnalités dans notre application qui n'étaient pas prévues dans notre étude préalable comme l'ajout de sauvegarde et des comportements aléatoires. Nous avons aussi écarté le menu de pause de notre application, nous ne l'avons pas jugé nécessaire à l'application et préférer avoir d'autres ajouts de fonctionnalités à la place.

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 ensembles 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 évident 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 beaucoup de difficultés avec l'utilisation d'un réseau de neurones pour l'un de nos agents dans notre application. Tout d'abord nous avons utilisé une bibliothèque fournie par l'un de nos enseignants (M.Boniface). Notre premier objectif avec le réseau de neurones est de lui faire apprendre un arbre de décision utilisé pour le rôle du gardien. Mais avec cette première bibliothèque, nous n'avons obtenu aucun comportement intelligent de la part du réseau de neurones, malgré des différentes données entrées il renvoyait toujours la même sortie. Nous avons donc changé de bibliothèque pour une dont la

stabilité est garantie: Neuroph. Mais de la même manière nous n'avons eu aucun comportement intelligent, le gardien se déplaçait toujours dans la même direction. Et surtout nous n'avions que peu de choix sur la configuration de notre réseau. Donc nous avons encore une fois changé de bibliothèque pour avoir plus de liberté : DJL. Encore une fois nous n'avons pas réussi à avoir un comportement intéressant même si cette fois le personnage pouvait arriver à se déplacer dans une à deux directions par partie. Il est important de souligner que nous avons aussi réduit et simplifié la carte pour que dans un premier temps, le gardien apprenne à suivre le prisonnier lorsqu'il l'aperçoit. Mais les résultats n'étaient toujours pas convaincants.

III. Planning de déroulement du projet durant les 4 itérations de ce semestre

Dès le début de ce projet tutoré, nous avons établi un planning de base, le voici mis à jour en fonction de notre réel avancement:

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)

<u>Itération n°4:</u>

- 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)

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 compétences.

Célie a commencé par créer une structure au projet et a tout du long aider à appliquer les principes *SOLID* au projet en optimisant la conception de l'architecture. Comme par exemple la refactorisation des vues avec l'ajout de ***VueSimulation*** qui permet d'éviter la copie de code. Ensuite, la plupart des tâches de Célie étaient orientées vers le backend de l'application comme le calcul des chemins avec l'implémentation de *A** et les calculs de la vision, ainsi que l'ajout des arbres de décisions. Cela est dû aux préférences de Célie pour la conception et le backend et ainsi qu'à ces compétences. Une autre partie de son temps est passée à améliorer l'affichage comme ajouter un fichier CSS au projet et rendre l'application plus responsive.

Matias et Luc ont principalement travaillé sur le développement de l'inférence bayésienne et l'implémentation du réseau de neurones dans le projet. Ils ont développé en priorité les méthodes de calcul de probabilités grâce à l'inférence bayésienne car beaucoup d'algorithmes que nous voulions développer utilisent les probabilités pour guider leur choix. Le réseau de neurones et toutes les fonctionnalités périphériques nécessaires ont également fait partie de leurs tâches. Le paramétrage, la mise en place et la découverte des bibliothèques pour les réseaux de neurones a été très chronophage De plus, bien que la méthode d'apprentissage soit toujours la même, son implémentation varie en fonction de la bibliothèque utilisée, ce qui impacte également le débogage.

Ayant un attrait pour le côté frontend et gestion de projet, Maëlle s'est principalement occupée de l'aspect graphique de l'application, en développant les différentes vues nécessaires au déroulement de la simulation. Chaque vue a nécessité tout de même une concertation de toute l'équipe. Un mode d'analyse a également été entièrement implémenté afin que l'utilisateur puisse examiner les parties en détail. En parallèle, Maëlle s'est aussi occupée de la rédaction des différents rapports de fin d'itération et des présentations de soutenance.

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

➢ Luc :

L'élément dont je suis le plus fier dans ce projet est la mise en place de l'architecture qui accueille toutes nos fonctionnalités. Dans un premier temps nous nous sommes tous concerté pendant la phase d'analyse pour accorder notre vision du projet, car cette dernière influence énormément les contraintes techniques de notre application. Donc une fois satisfait de notre projet et en accord avec notre sujet, nous avons décidé de créer une application avec laquelle il est possible d'interagir facilement avec les périphériques d'un ordinateur comme le clavier ou la souris, et dans laquelle l'utilisateur est seul. Il est confronté uniquement à des algorithmes, aucun autre joueur. L'utilisateur n'a donc pas besoin de rentrer ses informations et nous n'avons pas besoin de les stocker. C'est pour cela que l'application n'a pas besoin de base de données ou API.

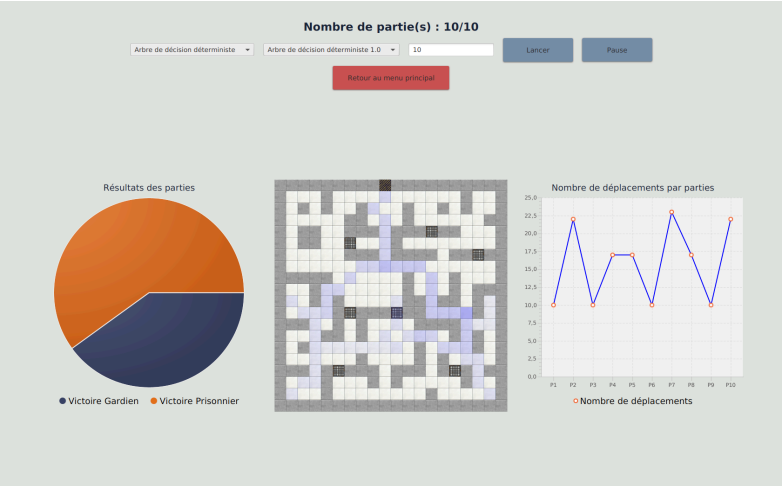
Nous avons donc opté pour une architecture en MVC, car étant donné les raisons énumérées ci-dessus notre application reste assez cloisonnée donc le choix du MVC plutôt qu'une architecture hexagonale nous semblait plus judicieux. Notre modèle contient plusieurs classes mais il y a tout de même une classe que l'on pourrait qualifier de "classe mère" c'est-à-dire une classe qui contient toutes les autres classes du modèle et qui se charge de diriger leurs interactions. Cette classe est la classe ***Simulation***, c'est d'ailleurs cette classe qui possède toutes les vues de l'application et qui se charge de réclamer leurs mise à jour lors d'une modification du modèle.

Je suis aussi très fier d'avoir travaillé avec Matias sur l'implémentation des méthodes d'inférences bayésiennes et leurs utilisations dans notre projet. Ce qui a été très intéressant c'est d'appliquer des notions théoriques de probabilité et de statistique vus pendant mon parcours scolaire/universitaire, pour un projet et surtout pour un problème concret.

➢ Maëlle :

Ma contribution dont je suis la plus fière dans ce projet a été de réaliser 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 :

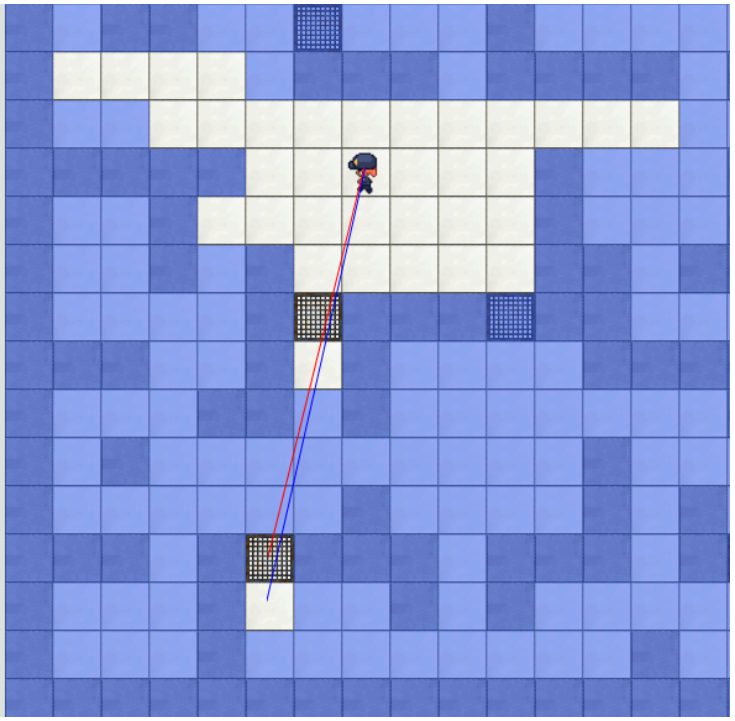
- le taux de victoire (gardien, prisonnier ou match nul),
- la carte des cases les plus utilisées durant les différentes parties,
- le nombre de déplacements des personnages en fonction de chaque itération.

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 un fichier.

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 nous avons 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.

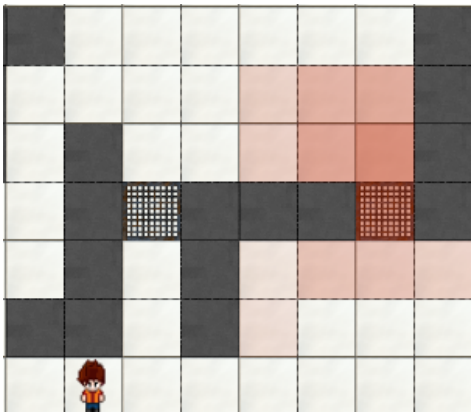
Le développement de cette inférence a été particulièrement difficile en raison de la complexité des tests. Comme les probabilités de présence sont souvent arrondies, il est difficile de prévoir avec exactitude la valeur retournée par le programme. Ainsi, tous les tests ont été effectués de manière empirique,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

Afin d'être plus les plus efficaces possibles lors des 3 dernières itération, nous avons établis un nouveau planning clair, ciblé et mis à jour en fonction de notre avancement jusqu'à présent:

Itérations	Tâches
Itération n°5	- Changement des données mise en entrées du réseau de neurones pour des images représentant la situation - Ajout d'un système de caméra de surveillance pour le gardien - Modification de l'affichage de la vision en une version plus réaliste - Ajout du temps réel dans le mode non interactif et dans l'historique du mode interactif - Ajout d'une page d'information sur les différents comportements - Optimisation du code
Itération n°6	- Perfectionnement de l'interface graphique - Ajout d'un menu de choix de carte - Optimisation/Nettoyage du code
Itération n°7	- Rédaction du rapport fin de projet tutoré - Préparation de la présentation finale - Préparation de la démonstration finale

VII. Conclusion

Pour conclure, nous sommes très fiers des avancées réalisées tout au long des quatres premières itérations malgré les difficultés rencontrées. Certaines difficultés, notamment liées à l'intégration du réseau de neurones et aux ajustements de l'inférence bayésienne, ont nécessité de revoir nos approches et de tester différentes solutions avant d'obtenir des résultats satisfaisants. Cependant, ces obstacles nous ont permis d'approfondir nos compétences en intelligence artificielle, en optimisation d'algorithmes et en architecture logicielle.

Grâce à une bonne organisation et à une communication efficace au sein de l'équipe, nous avons non seulement respecté une grande partie de notre cahier des charges initial, mais avons également enrichi notre application avec des fonctionnalités supplémentaires qui la rendent plus complète et plus intéressante à utiliser. Nous avons su adapter nos objectifs en fonction des imprévus tout en veillant à maintenir une cohérence dans le développement de notre projet.