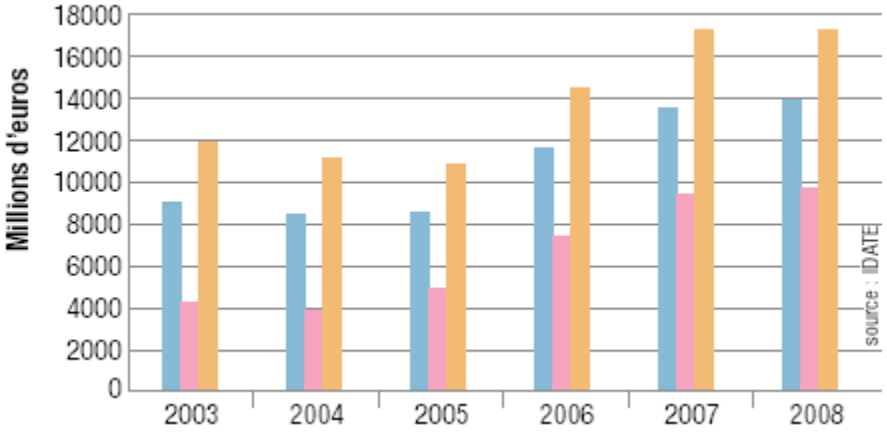
Eléments sémantiques tactiques dans un environnement 3D

Rapport d'analyse

Table des matières

-I- Introduction

Le marché du jeu vidéo est important et en forte croissance : le chiffre d'affaires en 2004 s'élève à 24 milliards de dollars, ce qui est supérieur au chiffre d'affaires du cinéma ou de la musique. Ce chiffre est en augmentation depuis le début de l'année 2000. Certaines études estiment d'ailleurs que ce chiffre doublera d'ici 2008.



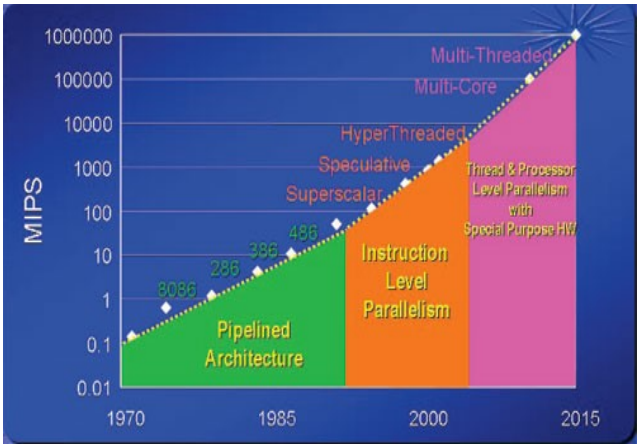
Taille des marchés (en Millions d'euros):

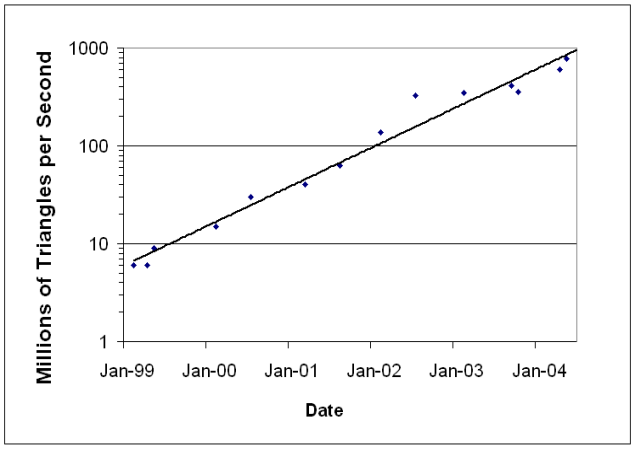
Bleu: Europe

Rose: Japon

Orange: USA

De plus, on observe une forte évolution technologique durant les 15 dernières années :





Toujours plus de puissance CPU (Central Process Unit) et GPU (Graphical Process Unit) en échelle logarithmique.

Ne serait-ce qu'en l'espace de 5 ans, les performances de calculs ont été multipliées par plus de 130. Cette évolution est logarithmique et doit se poursuivre durant les 10 prochaines années.

Afin de tirer partie de cette évolution de marché et des possibilités offertes par la puissance de calcul actuelle, les jeux doivent alors se complexifier. Les jeux actuels et futurs intègrent donc plus de contenu, qui se résume de la manière suivante :

Pour une surface de terrain donnée et une durée de jeu identique, le jeu nouveau doit fournir plus d'éléments. Soit :

- Plus d'objets passifs, étant donnés les GPU (Graphical Process Unit) toujours plus puissants.
- Plus d'objets actifs, dont plus de PNJ (Personnage Non Joueur) avec une intelligence accrue.

Exemple d'augmentation de contenu, ici graphique.

L'un des enjeux principaux d'un jeu est d'offrir la possibilité au joueur de refaire une même partie sans que celle ci ne soit similaire à la précédente.

C'est dans cette optique que la conception des PNJ est une tâche ardue et peu évidente: elle nécessite un long travail de réalisation afin de les rendre réalistes. Leurs comportements doivent être satisfaisants et permettre à l'utilisateur d'avoir une bonne expérience de jeu. En effet, le joueur ne doit pas pouvoir anticiper les réactions de l'IA, la partie doit être jouable plusieurs fois en suscitant à chaque fois l'intérêt du joueur.

Trois modèles typiques peuvent alors être analysés:

- Un jeu d'action, tel que Half-Life, fait appel à des scripts de scénarisation de la partie. Celle ci peut être donc très agréable et surprenante la première fois, mais perdra énormément de son intérêt les fois suivantes.
- Un jeu de stratégie, comme Age of Empires, utilise une IA plus souple ne suivant pas aveuglement un scénario, mais guidée par différentes stratégies prédéfinies. Ce système offre une rejouabilité correcte, mais le joueur repère rapidement les comportements émergeant de l'IA, ceux-ci étant peu variés.
- Un jeu d'échec, basé sur un monde simple, permet d'avoir une IA plus poussée. Le nombre de possibilités étant fortement limité, beaucoup plus de stratégies peuvent être évaluées par l'IA. En résulte une variété de parties quasi infinie.

C'est pour ces raisons que, dans le domaine du jeu, l'IA doit être conçue afin de rester cohérente avec le contexte choisi. Par exemple, la stratégie d'un joueur de tennis requise lors d'un match n'intervient pas dans les stratégies d'embuscade d'un jeu d'action ou vice-versa. L'IA d'un jeu ne dispose donc pas d'une autonomie totale : elle doit obéir à certaines règles définies par le programmeur.

Classification du jeu	Occupation du marché	Complexité de l'IA
Course	17%	Presque inexistante
FPS/ACTION	22%	Script et IA simple
Plate-forme	15%	Inexistante
Sport (hors course)	15%	Script
RPG	15%	Script
RTS	17%	IA : **la seul vraie IA**

Rapport entre type de jeu, marché et IA

- Le jeu de course est en général un jeu de sport mécanique dans lequel le joueur essais de battre l'ordinateur sur circuit ... L'IA ne sert en général qu'à régler la vitesse des opposants au joueur et à choisir une trajectoire.
- Le FPS (First Person Shooter) est un jeu d'action dans lequel le joueur a pour objectif de se rendre d'un point A à un point B. Le joueur voit au travers des yeux du personnage qu'il incarne. L'IA gère les PNJ comme les différents ennemis qui s'opposent à lui durant son trajet.
- Le jeu de plate-forme est un jeu qui met à l'épreuve l'habileté du joueur à se déplacer dans un environnement tout en évitant ses ennemis... Contrairement au FPS, la vue du personnage est externe au personnage incarné. Le jeu est beaucoup moins orienté sur l'action contre les adversaires, mais plus sur l'interaction avec les éléments du décor. L' IA est donc inexistante.
- Le jeu de sport ont une IA qui s'adapte aux capacités du joueur. Peu d'informations sur les IA de ce type de jeu sont divulguées par les éditeurs.
- Le RPG (Role Playing Game) est l'adaptation du Jeu de rôle “papier” en jeu vidéo. Le joueur a pour but de faire évoluer les caractéristiques de son personnage. Les résultats des actions du joueur dépendent seulement des choix qu'il réalise au long de l'aventure et des capacités de son personnage. L'IA se contente de scénariser le jeu.
- Le RTS (Real Time Strategy) met en scène deux armées (ou plus) qui s'affrontent. Le joueur dirige une armée et doit vaincre la stratégie de son adversaire: l'ordinateur. Plusieurs IA existent offrant divers niveau de rejouabilité malgré leurs limites.

Dans tous les types de jeu utilisant une IA, le but de cette dernière est ambiguë : elle doit perdre tout en opposant une résistance. Le joueur doit toujours avoir une impression de challenge tout en ayant le moyen de gagner. Aujourd'hui peu de jeu arrive à pondérer efficacement cette notion.

Notre but est donc de créer l'IA d'un RTS pour la rendre plus réfléchie. Cela permettra au jeu d'avoir une diversité de tactique et de comportement adverse accrue. L'IA sera alors perçue comme véritablement “intelligente” car elle sera peu prévisible et non reproductive: pour une même situation, sa réaction pourra être différente. Néanmoins, nous ne devons pas construire une IA chaotique, dont le comportement serait irrationnel.

Afin d'observer une réaction tactique à petite échelle, nous nous basons sur un RTS avec peu d'unité se déplaçant sur un terrain restreint. Dans ce cas, l'IA sera capable de faire une analyse précise de la situation afin de prendre en compte le joueur adverse dans ses décisions. L'IA ne sera donc pas purement réactive.

-II- Models de Comportement

Depuis toujours, les organismes vivants perçoivent, décident et agissent ce qui les rend autonomes et capables d'évoluer dans un environnement dynamique. L'intelligence artificielle s'appuie sur le modèle général de boucle perception-décision-action, pour animer le comportement d'entités virtuelles.

Définissons, tout d'abord, les composantes de cette boucle:

La perception apporte les informations propres à l'environnement dans lequel l'organisme vit. Ainsi, la perception fournit à l'organisme un ensemble d'informations lui permettant d'adapter son comportement en accord avec ce qu'il perçoit de l'environnement.

L'action représente les moyens dont l'organisme dispose pour modifier l'environnement qui l'entoure.

La décision réalise la corrélation entre la perception et l'action en choisissant le comportement adapté au contexte environnemental.

Selon Newell, on peut distinguer quatre grandes bandes temporelles, caractérisant différents niveaux de systèmes participant au comportement :

- Bande biologique : Circuit neuronal.
- Bande cognitive : Opérations élémentaires (réflexes).
- Bande rationnelle : Hiérarchie d'opérations élémentaires répondant à un choix.
- Bande sociale : Interaction et relation entre individus.

Plusieurs types de comportements sont alors identifiables si on fait interagir les états du modèle différemment. On retrouve alors l'homéostasie qui permet à l'organisme de se maintenir vivant, le comportement de perception qui le pousse à s'informer davantage sur son entourage et les comportements d'interaction qui affectent l'environnement.

On regroupe ces termes en deux systèmes : les comportements réactifs (reflèxe) et les comportements cognitifs (réflexion).

Il existe plusieurs systèmes de gestion des comportements réactifs comme les systèmes stimuli-réponses, systèmes à base de règles et les automates.

-A- Systèmes Stimuli-réponses

Dans les systèmes stimuli-réponses, les comportements sélectionnés sont le résultat direct d'interactions avec l'environnement sous forme de réseaux de noeuds inter connectés à l'image du cerveau humain. Les connections entre neurones sont pondérées. Elles déterminent l'activation de ces neurones et par conséquent le comportement de l'entité. Plusieurs algorithmes jouent sur cette pondération pour faire adopter à l'entité un comportement cohérent. Elles définissent le niveau d'apprentissage de l'entité.

Il existe principalement deux types de réseaux utilisant ce système :

- les réseaux SAN (Sensor Actuator Networks) permettent de contrôler le déplacement d'entités. Ils sont efficaces dans certaines situations non évolutives mais exhibent des comportements incohérents lorsqu'ils sont confrontés à des conditions nouvelles.
- les boucles SCA (Sense Control Action) contrôlent également le déplacement d'entités mais intègrent l'environnement dans leurs réseaux de neurones.

Ces systèmes sont particulièrement efficaces et utiles pour des comportements très réactifs mais peu complexes ou de faible niveau.

-B- Systèmes à Base de règles

Les systèmes à base de règles définissent l'ensemble des situations possibles et les comportements associés. Le comportement émergeant d'un système de règle est limité en autonomie en fonction de la richesse de l'ensemble des règles.

C'est alors qu'un modèle plus général se dégage: des actions complexes sont décomposées en action primitives qui seront elle même régit par un système de règles. Un arbre de décision met en place ce système. Il s'organise en niveau de décisions plus ou moins précis que l'entité sélectionne en fonction de son état.

Ce concept est à réaliser avec précaution car il peut entraîner un enchaînement contradictoire d'action primitive notamment si deux comportements se décrivent par une même action primitive.

-C- Les Automates

Les automates permettent de réaliser un enchaînement d'un certain nombre de tâches considérées comme élémentaires à un certain niveau d'abstraction. On distingue trois types de systèmes basés sur l'utilisation d'automates : les piles d'automates, les automates parallèles et les automates parallèles hiérarchiques.

Les piles d'automates permettent d'empiler une suite d'actions à réaliser. Un automate se terminant rend actif l'automate suivant dans la pile.

Le principe des automates parallèles est de coupler plusieurs automates pour gérer des actions simultanées d'une entité. Un automate peut se voir attribuer des fonctions divers telles que : l'exécution d'une tâche atomique, le lancement d'un nouvel automate, ou la planification de tâches à réaliser.

On introduit la notion d'automate père et fils dans le système d'automates parallèles hiérarchiques. Les automates fils fonctionnent en parallèle de leur père. L'automate père réalise une synthèse des données collectées.

1) Schéma simple (alpha)

Les automates sont très adaptées dans le milieu commercial pour la gestion d'acteurs autonomes ou encore dans l'automobile dans le cadre de la conduite de véhicules ou encore la simulation de piétons virtuels.

Nous pouvons constater plusieurs représentations des connaissances des comportements cognitifs comme le situation Calculus, STRIPS, la planification HTN, les mécanismes de sélection d'actions et les agents BDI.

2) Situation Calculus

Le situation calculus (calcul situationnel) est un langage logique qui sert à représenter des changements ou évolutions. Les changements proviennent d'une action. Une situation est une histoire possible du monde, c'est à dire une suite d'actions. Formellement, le langage comporte quatre types d'objets :

- Les situations correspondent à l'état complet du monde à un instant donné.
- Les actions permettent de changer une situation.
- Les flux décrivent une propriété sur le monde qui peut changer au cours du temps.
- Enfin, la connaissance augmente le niveau d'abstraction du formalisme en rendant possible l'acquittement d'informations utiles durant la construction d'un plan.

L'utilisation de ces concepts permet de calculer des ensembles de mondes futurs, résultats de l'exécution d'une ou plusieurs actions sur la situation courante. Ce formalisme connaît un problème lié à la description

des actions qui ne changent pas dans le monde après l'exécution d'une action. Plusieurs environnements permettent d'exploiter la puissance du situation calculus en contournant ces problèmes comme le GOLOG et plus récemment le CML.

3) STRIPS

STRIPS est un langage de descriptions d'actions qui permettent de d'exploiter un sous ensemble du situation calculus. Un schéma d'action est une règle qui contient un nom, un ensemble de paramètres, et un ensemble de formules comportant chacune des pré conditions, des effets et un corps.

Les pré conditions sont les conditions nécessaires pour que l'action puisse être exécutée, les effets sont des faits qui sont ajoutés ou supprimés après l'exécution de l'action, et le corps est une description détaillée de l'action sous la forme d'une séquence de sous buts ou de sous actions.

Une occurrence d'action est un schéma d'action instancié (les paramètres sont fixés) et indexé par le temps. Un plan est une séquence linéaire d'occurrences d'actions, mettant en relation un état initial du monde et un état final but. Cette séquence est organisée sous la forme d'un graphe représentant un ordre partiel sur les moments d'exécution de ces actions. Une action est intentionnelle lorsque son auteur a voulu qu'elle se réalise. Les pré conditions de l'action contiennent alors la formule (Auteur VEUT Action). Une action intentionnelle est un acte de langage (informer, demander, demander-si, demander-ref) lorsque le locuteur et l'auditeur sont des paramètres du schéma d'action, et lorsque l'exécution de l'action conduit à la production d'un discours :

INFORMER (Locuteur, Auditeur, Proposition)
Pré conditions :
Locuteur VEUT INFORMER (Locuteur, Auditeur, Proposition)
Locuteur SAIT Proposition
Effets : Auditeur SAIT Proposition
Corps : Locuteur DIRE Proposition à Auditeur

4) Planification HTN

La planification HTN est différente des modes de planification puisqu'elle fonctionne de manière hiérarchique en s'appuyant sur trois concepts : les tâches, les méthodes et les actions.

Les tâches constituent les buts dans les méthodes de planification. Elles correspondent généralement à la satisfaction d'un ensemble de propriétés sur le monde, ou encore à la réalisation d'un ensemble d'actions.

Les méthodes correspondent à une suite de tâches et/ou d'actions à réaliser pour remplir la tâche à laquelle elles correspondent.

Les actions sont représentées par un formalisme de type STRIPS.

Une demande de planification se résume donc en une demande de réalisation d'une tache. La planification consiste ensuite en la décomposition de cette tache par l'intermédiaire des méthodes contenu dans le modèle de planification.

Ce formalisme est limité puisque d'une part il faut que les tâches d'un modèle soient décomposables en sous taches et que d'autre part les effets des actions n'invalident pas les pré conditions des actions suivantes. La planification HTN est cependant efficace puisqu'elle permet de contrôler de manière fine la réalisation des tâches. On l'utilise notamment dans les jeux vidéo sous le genre fiction interactive.

5) Mécanismes de Sélection

Les mécanismes de sélections d'actions ont été introduits pour gérer la réactivité (exploitation d'opportunité) et la recherche d'une suite d'actions permettant de satisfaire un but donné. Ces mécanismes sont basés sur un graphe d'actions.

Le graphe de Maes se définit ainsi: chaque actions sont définies par ses conditions d'exécution (c), les faits qu'elles vont ajouter au monde après exécution (a) et ceux qu'elles vont déduire (d) et leur niveau d'activation (l).

Il existe trois types de relations entre deux actions:

- relation de succession de A vers B si un fait f appartient à A(a) ou à B(c).
- relation de précédence de A vers B si un fait f appartient à A(c) ou à B(a).
- relation de conflit de A vers B si un fait f appartient à A(c) ou à B (d).

Le graphe est alors exploité de façon a pondéré selon le type de relation entre chaque action, les actions elles-mêmes, en offrant un but (suite de fait) en entrée du graphe. Quand une actions atteint un certain coefficient et que ses conditions sont satisfaites, elle est exécutée.

La deuxième phase consiste à mettre à jour le graphe (à le pondérer à nouveau) quand une action effectuée à changer l'environnement (les faits existant).

Ces deux propriétés permettent d'obtenir une planification pouvant exploiter les opportunités offertes par l'environnement.

Cependant, le mécanisme montre ses limites dans le cadre de plusieurs but: il peut osciller entre plusieurs actions a effectuer sans faire de choix définitif. De plus, la pondération défavorise les plans a choix multiples. On a donc des plans prioritaires qui peuvent passer au second plan.

Un modèle adapté, dénommé PHISH-Nets, améliore ces points. Il permet d'ajouter un paramètre aux actions (comme opérateurs STRIPS) pour affiner le lien de précédence. La pondération a été optimisée ainsi que la phase de dispersion. Ce système gagne en facilité de description et évite de défavoriser des actions.

Une autre forme de mécanismes de sélection d'action existe: en réseau.
Les noeuds du réseau sont de deux types:

- perception, qui sont les faits de l'environnement caractérisé par leur présence ou non.
- comportement correspondant a une action exécutable.

De même pour les liens:

- liens perception-action qui correspondent au lien pré condition-action.
- liens action-perception qui correspondent au lien action-conséquences.

Comme dans le graphe de Maes, ces liens sont pondérés mais ici, de deux manières différentes:

- direct : qui permet d'activer les actions dont les pré conditions sont vraies.
- indirect : qui permet de prendre en compte l'influence des buts dans l'activation des actions.

Ce système possède les mêmes propriétés que celui de Maes donc les mêmes défauts. Mais un système de hiérarchie de sélection d'actions est introduit: une action atomique qui a un certain niveau d'abstraction correspond a un autre mécanisme de sélection d'actions spécialisé.

Cette approche est efficace pour éviter les hésitations mais le niveau de hiérarchisation n'est pas formel et a besoin d'être défini par l'utilisateur.

En confrontant les deux modèles, on peut voir que le nombre de lien du deuxième modèle est bien moins important que dans le premier, ce qui améliore les performances du mécanisme de sélection d'action.

Malgré des possibilités d'oscillation due à des buts conflictuels, ces mécanismes sont très adaptés à une modélisation comportementale. L'environnement pouvant changer sans l'agent, la suppression de plan et la sélection d'opportunités peuvent être envisagé.

6) BDI

Le mode de raisonnement associé aux agents BDI s'inspire du raisonnement pratique humain. Il est associé aux concepts suivants:

- croyances (conditions): désigne la vérité de l'agent induites par les capacités de perception réduites de l'agent, sur la dynamique du monde et l'impossibilité de prévoir son état.
- désirs (Desires): représentent la motivation de l'agent (peut être assimilé a un but mais sans formalisation telle que dans le situation calculus)
- intentions: ensemble des plans que l'agent exécute en vue de réaliser ses désirs.

Le mode de raisonnement fait des croyances, des conditions à la réalisation d'un plan choisi pour répondre à un désir.

Lorsqu'un plan s'exécute il peut être interrompu par l'ajout d'un désir, la modification des croyances de l'agent ou de l'exécution d'une action atomique.

Ce système permet de rapidement prendre en compte les changements d'environnement donc d'intentions et de décrire des plans généraux grâce a des désirs décrivant des sous buts (exemple: « réalisation des désirs ») qui se raffineront en fonction des croyances de l'agent.

Cependant, l'agent n'est pas autonome devant un désir pour lequel il n'a pas de plan, il « oubliera » son désir. Et il n'est pas possible non plus de contrôler la compatibilité des désirs.

-D- Conclusion

Les différents modèles qui viennent d'être présentés permettent de décrire différents aspects du comportement allant de la modélisation de comportements réactifs, à la modélisation de processus de raisonnement en vue de trouver des enchaînements d'actions cohérents pour atteindre un but fixé.

Les approches à base d'automates sont les plus adaptées à la description de systèmes réactifs.

Pour les notions de but, le situation calculus permet de prendre en compte beaucoup plus d'information et permet de résonner sur la connaissance. Cependant, c'est un système très coûteux en terme de calcul, ce qui ne permet pas une certaine réactivité. La sélection d'action permet d'offrir un compromis entre réactivité et planification malgré ses défauts. La planification HTN permet une planification rapide pour garder une certaine réactivité et une utilisation d'une connaissance experte même si elle est nécessite une conception particulière. Elle est donc plus complexe que le formalisme STRIPS qui se base sur une description indépendante des actions. Les systèmes BDI ont le même problème même si leur grande qualité est leur réactivité.

En conclusion, un système adapté à la simulation du comportement doit posséder une architecture réactive et cognitive. L'architecture réactive doit pouvoir réagir rapidement à l'environnement mais doit être contrôlée par un processus plus lourd en calculs. De plus, l'exécution simultanée de plusieurs tâches doit être incluse par l'architecture réactive, pour qu'elles soient en réponse à l'environnement (non cognitif) et en action sur l'environnement (passage par l'architecture cognitif). Ceci implique donc un système suffisamment abstrait pour mélanger les comportements sans gérer la concurrence des comportements réactifs dans les comportements cognitifs.

Source: F. Lamarche. - Humanoïdes virtuels, réaction et cognition : une architecture pour leur autonomie. Thèse de doctorat. - Université de Rennes I, décembre 2003.

-III- Cahier des Charges

Nous avons pour objectif de concevoir et réaliser une intelligence artificielle pour un jeu de stratégie en temps réel. Cette intelligence artificielle doit suivre certains critères que nous allons fixer ici.

Notre IA ne sera pas réactive, nous voulons que celle-ci formalise un but. Celui-ci n'est pas simple, il s'agit d'un jeu: le joueur doit donc pouvoir gagner, mais il doit également avoir un défi à relever. Nous aimerions également que - contrairement aux jeux récents - les mouvements et autres actions des unités manipulées par l'IA ne soient pas scénarisés. Un bon jeu doit être rejouable tout en suscitant un intérêt dans la diversité des comportements. Notre choix se porte donc naturellement vers un système cognitif.

Cependant, comme nous avons pu le voir dans le chapitre précédent, nous ne pouvons pas utiliser intégralement un système cognitif, pour des raisons de lourdeur de calcul d'une part, pour le manque de réactivité d'autre part: Une partie du système sera donc réactive.

Pour alléger le système cognitif nous devons également nous placer dans un environnement relativement simple. Le nombre d'unités de chaque armée sera donc limité, il en sera de même pour leur variété. De plus, les actions des unités seront limités.

Néanmoins, dans un soucis d'évolutivité, nous optimiserons l'architecture afin de pouvoir ajouter facilement de nouvelles fonctionnalités. Nous pourrons ainsi faire évoluer le jeu tout en gardant l'architecture intacte en intégrant d'autres unités, d'autres types de terrains ou comportement.

Nous devons de plus créer une recherche de chemin de façon à pouvoir déplacer les unités vers un point précis. Ce chemin doit être autant que possible fluide et rapide, c'est à dire naturel et intuitif tout en étant optimal. L'algorithme correspondant ne doit pas monopoliser le processeur afin de permettre au système cognitif ses calculs.

Le système cognitif sera développé à base de règles floues. Cela permettra de ne pas avoir de comportements binaires en utilisant la logique flou et de développé le coté humain de l'IA.

Le document que vous avez entre les mains vous présente le détail des solutions analytiques à ce cahier des charges.

-IV- Spécifications

-A- Unités

1) Définitions des Termes

Unité : Objet 3D mobile qui possède des caractéristiques.

Exemple : Fantassin, Archer, Piquier, Cheval.

Points de vie : Nombre représentant l'endurance d'une unité. Quand ils arrivent à zéro, l'unité meurt. Chaque type d'unité possède un nombre de points de vie différent.

Dégâts de l'arme : Valeur de dommage que l'unité inflige grâce à son arme sans prendre en compte aucun modificateur.

Dégâts effectifs : Valeur de dommages calculée selon l'armure, le taux de critique, etc... Il est soustrait aux points de vie de l'unité ennemie visée.

Exemple : Un fantassin « monté » a plus de chance de faire un coup critique qu'un fantassin à pied.

Coup critique : Chaque unité a une probabilité d'infliger un coup critique. Un coup critique multiplie les dégâts effectifs.

Coup fatal : Chaque unité a une probabilité d'infliger un coup fatal. Un coup fatal tue l'ennemi quelque soit son nombre de points de vie restants.

Type d'attaque : Chaque unité possède un type d'attaque qui amplifie ou diminue les dommages qu'elle peut infliger en fonction de l'armure de l'adversaire.

Exemple : Perçant (flèche), Perforant (lance), Tranchant (épée)

Type de défense : Chaque unité possède un type d'armure qui reflète son niveau de défense.

Exemple : Léger (cuir), Moyen (mailles), Lourd (plaques)

Distance d'attaque : Chaque unité peut attaquer à une certaine distance. La distance maximale est fixe.

Vitesse d'attaque : Nombre de secondes entre chaque coup. Chaque unité a une valeur différente.

Chance de désarçonner : Le coup critique du piquier désarçonne les ennemis montés.

Probabilité de toucher un ennemi monté : Lorsqu'une unité combat un adversaire monté, il a une probabilité de toucher le cheval ou le cavalier.

Monture : Les unités peuvent toutes monter à cheval (sauf celles qui montent déjà un cheval). Cela modifie certaines conditions de combat (coup critique, coup fatal...). On s'en rend compte avec le tableau en annexe.

Calcul des dégâts : passage des dégâts de l'arme aux dégâts effectifs.

2) Description des unités

Archer : Cette unité a un arc. Il tire des flèches. Il porte du cuir. Ses dégâts sont de type perçant.

Fantassin : Il a une épée. Il porte une armure en plaques. Il inflige des dégâts de type tranchant.

Piquier : Il a une lance et porte de la maille. Il inflige des dégâts de type perforant. Son coup critique

désarçonne les unités montés.

Cheval : Peut être monté. Il n'a pas d'armure.

3) Règles du Combat

On va prendre comme exemple les unités suivantes :

Piquier

Points de Vie : 55

Type d'attaque : Perforant

Type de défense : Moyen

Dégâts de l'arme : 4

Pourcentage de critique : 5%

Pourcentage de coup fatal : 3%

Vitesse d'attaque : lent

Portée : 2 mètres

Pourcentage de chance de toucher une unité montée : 90%

Archer (monté)

Points de Vie : 30

Type d'attaque : Perçant

Type de défense : Moyen

Dégâts de l'arme : 4

Pourcentage de critique : 5% / 20% quand monté

Pourcentage de coup fatal : 1% / 4% quand monté

Vitesse d'attaque : rapide

Portée : 60 mètres (parfait) / entre 60 et 100 mètres (chance de rater accrue)

Pourcentage de chance de toucher une unité montée : 60% / 95% quand monté

Déroulement d'une attaque (celle du piquier) :

Si l'adversaire est sur un cheval (c'est le cas), on tire un nombre aléatoire pour savoir si on touche le cheval ou le cavalier. Si on touche le cheval, le cavalier ne subira pas les dommages.

On tire un nombre aléatoire entre 1 et 100.

Si le résultat est inférieure ou égale à la probabilité de faire un coup fatal (3 dans le cas du piquier)

Sinon, s'il obtient un nombre inférieur à la probabilité de faire un coup critique (5), les dégâts de l'arme sont multipliés par 2.

Selon l'armure de l'archer, l'attaque perforante du piquier fera des dégâts plus ou moins importants. On se réfère au tableau en annexe.

Dans le cas présent, les dégâts de l'arme seront multipliés par 75% (après application éventuelle du coup critique).

Les dégâts effectifs obtenus par ce calcul sont retranchés aux points de vie de l'archer.

Exemple :

On tire un 4, le coup n'est pas fatal.

Mais il est tout de même critique.

Dégâts de l'arme x 2 = 4 x 2 = 8

On applique ensuite les modificateurs d'armure de l'archer.

8 x 75% = 6.

L'archer a 55 points de vie, il en perd 6, il lui en reste 49 points de vie, il n'est pas mort.

4) Tableaux des Unités

Ces tableaux ne concernent que le combat, d'autres données caractérisent l'unité (perception, déplacement...)

Attaque / Défense	Léger (cuir)	Moyen (mailles)	Lourd (plaque)
Tranchant	150%	125%	75%
Perçant	100%	100%	75%
Perforant	75%	100%	150%

Tableau d'absorption des dégâts

Nom	PV	Type d'attaque	Type de défense	Dégâts de l'arme	% crit	% fatal	Portée max.	* % montée	Vitesse attaque
Archer	30	Perçant	Léger	4 (9 à distance)	5	1	60**	60	rapide
Fantassin	60	Tranchant	Lourd	9	10	2	1	15	lent
Piquier	55	Perforant	Moyen	4 (15 contre un cheval)	5	3	2	90	moyen

Tableau des unités à pied

(*) : pourcentage de toucher une unité sur un cheval (sinon on attaque le cheval).

(**) : l'archer peut tirer jusqu'à 100 mètres, mais au delà de 60 mètres, il vise moins bien.

Nom	PV	Type d'attaque	Type de défense	Dégâts de l'arme	% crit	% fatal	Portée max.	* % montée	Vitesse attaque
Archer	30	Perçant	Léger	4 (9 à distance)	5	1	60**	60	rapide

Fantassin	60	Tranchant	Lourd	9	15	10	1	75	lent
Piquier	55	Perforant	Moyen	4 (15 contre un cheval)	20	4	2	95	moyen

Tableau des modifications quand l'unité monte un cheval.

Le cheval a 60 points de vie et pas d'armure (100% des dégâts quelle que soit l'arme utilisée).

5) Variantes dynamiques

Selon les conditions de combat, ces caractéristiques fixes vont subir des modifications qui permettront de rendre le combat plus riche et d'élaborer ainsi des stratégies.

Les Effets de l'altitude

En principe, une unité se situant plus haut qu'une autre à plus de chance de mieux "frapper". Dans le cadre du jeu, les archers seront les seuls influencés par une différences d'altitude.

L'archer surélevé par rapport à son ennemi subit les modifications suivantes :

A définir : coup critique/fatal augmenté.

Attaque en Mouvement

Une unité en mouvement peut avoir plus ou moins de difficultés à se battre contre son adversaire. Le mouvement pénalise énormément un archer alors qu'un fantassin ne sera presque pas affecté par le déplacement.

A définir : modification de la précision et de la cadence de tir pour l'archer

fantassin et piquier, non affectés

Ennemi en Mouvement

Un ennemi en mouvement est plus difficile à attaquer qu'un ennemi statique.

Si la cible est mouvante, l'archer à moins de chances de toucher, le fantassin augmente ses chances de critique et de coup fatal sur cible fuyante, le piquier augmente sur cible chargeante.

Charger un Ennemi

La charge permet notamment au fantassin d'augmenter ses chances d'infliger un coup critique / fatal.

Chances de critiques et de coups fatals augmentés pour le fantassin, le fantassin monté, le piquier monté et l'archer monté.

Pose de Combat

Un archer qui a pris le temps de bien se positionner fera beaucoup plus de degats et visera beaucoup mieux que lorsqu'il n'a pas le temps.

Coup critique et coup fatals augmentés pour l'archer et le piquier.

6) Description de Déplacements

Type de Déplacement : Toutes les unités possèdent deux types de déplacements, la marche ou la course.

Mode de Déplacement : Toutes les unités possèdent deux mode de déplacements, la charge est une course offensive, la fuite est une course défensive, la marche simple est le mode de déplacement de base, l'approche est une marche discrète qui prend en compte les zones à couvert (André ?) ou non.

Vitesse de Déplacement : Selon l'unité, le type et le mode de déplacement, une vitesse est calculée est modifié par le terrain.

Archer : Rapide, Fantassin : Lent, Piquier : Normal, Unité montée : Très rapide.

Effets du Terrain : Le terrain conditionne la vitesse de déplacement. En montée, la vitesse est réduite. En descente, elle est accrue.

Dans les bois, la vitesse est diminuée. Une unité montée ne peut pas traverser les bois.

7) Perception

Type de Perception

Toutes les unités possèdent deux types de perception, qui sont l'ouïe et la vue.

Ces deux notions sont regroupées en une seule pour simplifier, on voit mieux devant que derrière, mais on possède une perception limitée de ce qui se passe dans notre dos.

Insertion d'un dessin très chouette montrant le cône de perception.

Effets du Terrain

Dans les bois, la perception visuelle est réduite.

L'altitude augmente la perception visuelle.

(être près de l'eau réduit la perception auditive)

-B- Analyse du Terrain

Pour analyser le terrain, nous avons décidé de distinguer plusieurs zones stratégiques. Celles ci sont les places fortes, les zones risquées, les zones cachées et les “axes naturels”. Les zones n'ayant pas de particularité, tel que le centre d'une large plaine, sont considérées comme neutres.

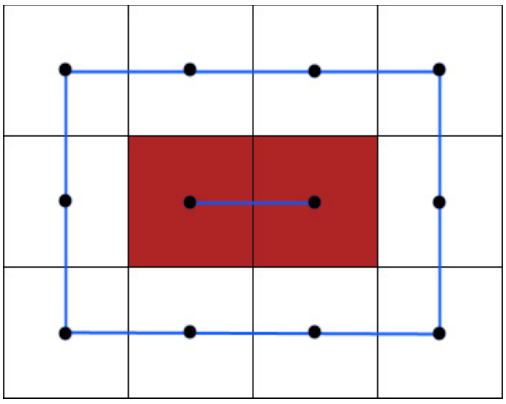
Nous procédons a l'analyse dans l'ordre suivant :

- Un graphe de déplacement est calculé en premier. Chaque noeud est une partie du terrain.
- Les axes naturels traduisent les axes de déplacement les plus utilisés. Un passage entre deux forêts par exemple.
- Les places fortes sont les zones donnant un avantage stratégique potentiel. Par exemple une colline est une place forte pour un groupe d'archers.
- Les zones cachés permettent de dissimuler des unités. Une forêt par exemple.
- Les zones risquées représentent les zones ou les unités sont désavantagées lors d'un combat. Les alentours d'une place forte seront risqués.
- Les points de controle de la carte, qui définissent les zones de passage favorisées entre plusieures places fortes.

Ces différentes zones sont déterminées une premiere fois statiquement, selon la géographie de la carte. Nous avons aussi prévu une modification des données produites lorsque l'IA recherche les unités à utiliser pour la partie, on l'appelle l'analyse semi-statique. Cette analyse permet d'avoir des unités cohérentes avec la carte. Les informations des zones sont ensuite modifiées dynamiquement au fil de la partie, selon les informations récoltées par l'IA. L'analyse du terrain servira au moteur décisionnel afin de reproduire un comportement intelligent.

1) Le graphe de déplacement

Le terrain est découpé en zones de petites tailles connexes. Chacune de ces zones est un noeud du graphe. Chaque zone est composé d'un seul type de terrain. Une fois le terrain découpé, on relie chaque noeud avec ses zones connexes par des arcs. Les arcs sont créés entre deux noeuds si une unité peux passer de l'un à l'autre. Chaque arc représente alors une possibilité de déplacement entre deux zones connexes.



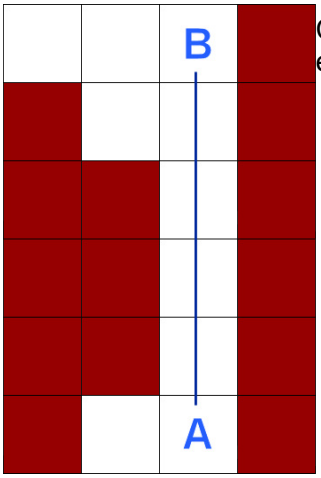
Ci dessus, un exemple de graphe de déplacement, les unités ne peuvent pas passer des zones blanches aux zones rouges.

2) Les axes naturels

Ces axes sont définis à partir du graphe de déplacement. Un ensemble de noeuds reliés par un faible nombre d'arc constitue un axe naturel. Il décrit les endroits qui ont des déplacements réduits. Ces endroits sont notamment les vallées, les ponts, les culs de sac. Le moteur décisionnel pourra ainsi préparer des stratégies intelligement.

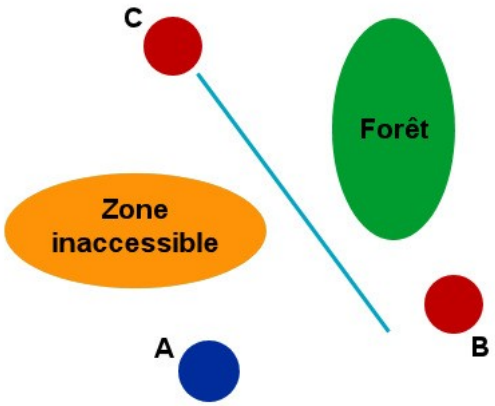
Analyse statique	**Analyse dynamique**
- Ponts. - Vallées - Bordure des rivières. - Passage obligatoire entre deux zones. - Zones bordées de falaises	- En fonction des unités ennemies aperçues, les valeurs des axes naturels sont pondérés.

L'analyse statique utilise le graphe de passage, donnant ainsi les chemins pris par les unités ennemis lors d'un déplacement rapide entre deux points. Les axes naturels sont particulièrement pertinent concernant les déplacements de la cavalerie, celle ci ne pouvant passer montée a travers les forets.



Ci-contre, les zones rouges sont inaccessibles depuis les zones blanches, il existe donc un axe naturel en les points A et B.

L'analyse dynamique permet a l'IA d'utiliser au mieux les axes naturels en fonction des unités ennemies. Une armée ennemie composée majoritairement d'archer aura de grande chance d'utiliser certains axes naturels. De plus, lors d'une attaque sur un groupe ennemi, les axes naturels proches de la zone de combat permettent d'anticiper le chemin pris par d'éventuels renforts.



Ci- contre, le groupe d'unités A veut attaquer le groupe B. L'IA ayant localisé et analysé le groupe C, composé de cavaliers, elle pourra anticiper la voie empruntée par le groupe C pour secourir le groupe A.

3) Place forte

Zone accessible qui donne un interet stratégique pour atteindre un objectif.

Nous avons comme place forte les zones surélevées, les zones difficilement accessibles ou les zones cachées.

Analyse statique	**Analyse semi-statique**	**Analyse dynamique**
- Colline. - Pont (a lentrée et a la sortie). - Bordure d'axe naturel fait de foret ou de zones surélevées. - Haut d'une falaise. - Bord des rivières.	- Selon les unités choisies, les valeurs des différentes places fortes sont pondérées.	- Selon les unités perdues et les unités enemies repéré, les valeurs des différentes places fortes sont pondérées.

L'analyse statique relève les zone géographique ayant un interêt stratégique important. Poster ses archers sur une colline ou controller un pont donne un avantage crucial lors d'une bataille. L'interet des différentes places fortes est pondéré par le choix des unités, une colline étant moins avantageuse pour une cavalerie que pour un groupe d'archer.

Lors de la partie, l'IA pourra modifié en temps réel la valeur des places fortes. Ayant perdu tous ses archers, une colline perdra énormément d'interet. De plus, face a une armée composée essentiellement d'archer, controller la colline voisine sera un avantage indéniable.

4) Zone cachée

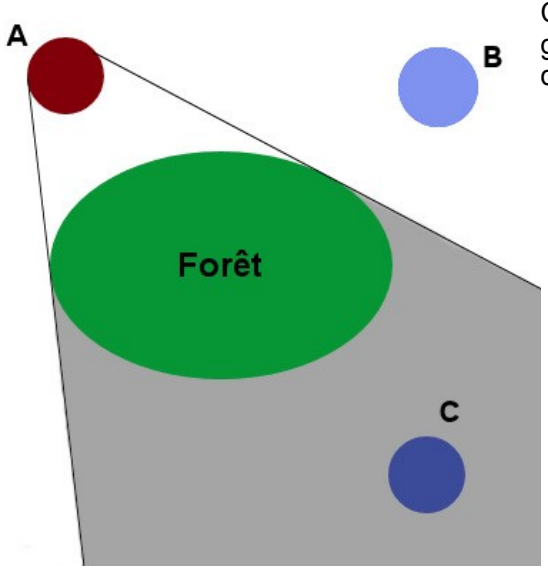
Zone ou les unités ne sont pas visibles depuis une zone extérieure.

Analyse Statique	Analyse dynamique
- Forêts. - Zone partiellement entourées de falaise.	- Zones masquées par un obstacle, tel une forêt ou une colline.

L'analyse statique détermine les zones ou les unités seront forcément cachés aux unités hors de cette zone.

Seules les forets et les zones entourées de falaises ou de surélevements sont donc éligibles.

L'analyse dynamique donne des zones cachées en temps réels, selon les zones vues par les unités a un moment donné. Si un groupe se trouve en contrebas d'une colline, tout le terrain qui n'est pas en vue formera un cone de zone caché.



Ci-contre, le groupe B n'est pas en zone caché pour le groupe A, au contraire du groupe C, qui se trouve caché derrière la forêt.

5) Zone risquée

Zone ou des unités attaquées sont en désavantage. On y retrouve les zones propices au guet-apens, ou les zones voisines aux positions surélevées.

Analyse statique	Analyse dynamique
- A portée d'une ou plusieurs places fortes. - Zone de déplacement difficile. - Cul-de-sac. - Axe naturel bordé de zone cachées ou surélevées. - Extrémités des axes naturels. - Points de croisement d'axes naturels.	- Les zones riquées sont pondérées par les types d'unités de l'armée ennemie, et par leurs positions possible.

Les zones risquées sont determinées en dernier puisqu'elles dépendent des autres types de zone.

L'analyse statique donne a l'IA les zones ou elle ne devra pas s'aventurer sans avoir une connaissance minimale des positions ennemies.

L'analyse dynamique modifiera la valeur du zone a risque, si les alentours de la zone risquée contiennent ou pas des unités ennemies. Ainsi l'IA ne prendra pas de chemin illogique, tel un long détour, lorsque l'ennemi est localisé.

6) Les points de controle

Ce sont des zones à fort taux de passage. Nous considérons que tous points utiles de la carte sont les places fortes. Les points de controle ont une valeur plus ou moins importante en fonction du taux de passage.

En utilisant le pathfinding entre les places fortes, les zones ou les chemins se coupent déterminent les points de contrôle. Les points de controle obtiennent une importance en fonction du nombre de recoupement de chemin lors de l'analyse statique. Suivant les unités choisies et les unités ennemies, l'importance d'un point de contrôle peut être modifée.

Ci-dessus, les chemins entre les 3 places fortes A,B et C sont calculés. Le pont devient alors un point de controle, 4 chemins se croisant a cet endroit.

-C- Recherche de Chemin

L'objectif de la recherche de chemin est de trouver un chemin entre deux points donnés. Ce chemin doit répondre à un certain nombre de contraintes définies avant le début de la recherche. Celles-ci ne varient donc pas au cours de la recherche. Par exemple, une contrainte basique est que certaines zones ne sont pas franchissables (rivière).

Certaines contraintes sont facilement représentables du point de vus mathématique (coût du chemin) alors que d'autres sont plus difficilement évaluable objectivement (chemin qui semble humain). Toutes ces contraintes forment le critère d'optimalité qui permet de trouver le chemin qui satisfait le mieux à la situation (maximiser la satisfaction).

Selon les critères choisi, on peut utiliser différentes modélisations du problème. Par exemple, une représentation de la carte par une matrice de cases ou bien par un graphe de zones. Nous étudierons plus en détails ce problème dans la section 5.3.

1) Les Algorithmes

Il existe plusieurs méthodes pour effectuer une recherche de chemin. La plupart proviennent de la théorie des graphes. Les plus connus sont les algorithmes de recherche en largeur ou en profondeur. Ce qui diffère entre ces différentes méthodes est la structure de donnée qu'ils utilisent.

L'algorithme d'exploration en largeur d'abord, utilise une file pour stocker les sommets en attente de traitement. L'algorithme de profondeur d'abord utilise une pile. Le problème est qu'ils ne prennent pas en compte la spécificité des sommets. Une des solutions trouvés pour résoudre ce problème est d'utiliser une structure de tas. Le principe du tas est de prendre le meilleur sommet de l'ensemble selon un critère d'optimalité. Un des algorithmes utilisant cette méthode est appelé l'algorithme de Dijkstra. C'est un algorithme non glouton puisqu'il peut remettre en cause un sommet déjà traité. Il prend en compte le poids des arcs entre les sommets pour évaluer le meilleur arc à analyser.

Cependant il est possible d'améliorer Dijkstra en prenant en compte plus de critère que le seul poids de l'arc. Cet algorithme est appelé A* (A-Star). Il utilise une heuristique qui permet de définir les critères d'optimalités. Chaque arc n, se voit ainsi attribuer un score $f(n)$ qui est calculé suivant la formule :

$$f(n) = g(n) + \mu.h(n)$$

avec μ : une constante de pondération de l'heuristique,

$g(n)$: le poids de l'arc

$h(n)$: le coût calculé de l'heuristique

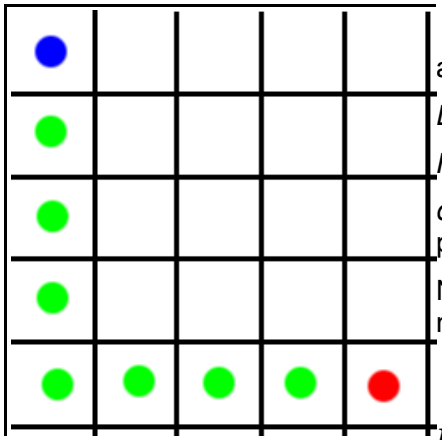
Parmi les différentes heuristique possible, nous avons choisi d'étudier:

- Manhattan
- Distance euclidienne
- Produit scalaire

Manhattan : cette euristique consiste à calculer le nombre de cases maximum restantes à parcourir. Par exemple, les coordonnées du point analysé sont (x1,y1) et celles du points d'arrivé sont (x2,y2) : la distance de Manhattan entre ces deux points est: $|x1- x2| + |y1-y2| -1$

a) Distance de Manhattan sans diagonal

Ce calcul n'est prévu que pour un déplacement horizontal ou vertical. Si on veut prendre en compte le déplacement en diagonal, il faut changer le calcul en :



*racineCarrée (2)*diag + (max(H,L) – diag)*

avec :

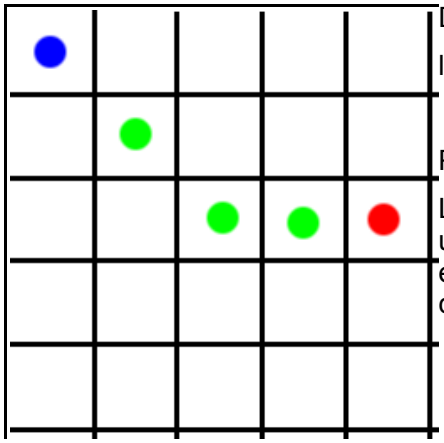
L = | *x1 – x2* | : Le nombre de case en longueur

H = | *y1 – y2* | : Le nombre de case en hauteur

diag = *min(H, L)* : Le nombre de case diagonal maximum pour un chemin direct optimum.

Nous sommes devant une grille de case carré, c'est pour cela que le nombre de case parcourue en diagonal et pondéré par racine de 2.

b) Distance de Manhattan avec diagonale



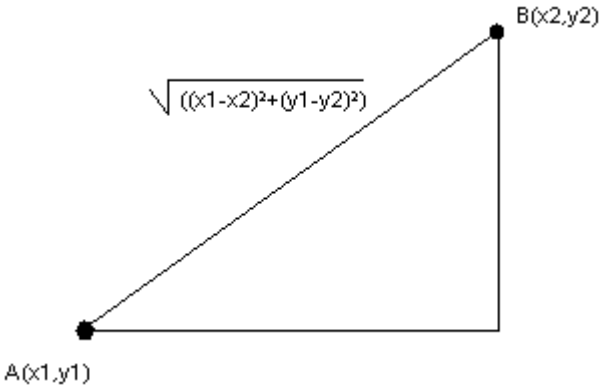
Distance euclidienne:

le calcul de distance ce fait cette fois de la façon suivante

racineCarrée ((x1-x2)² + (y1-y2)²)

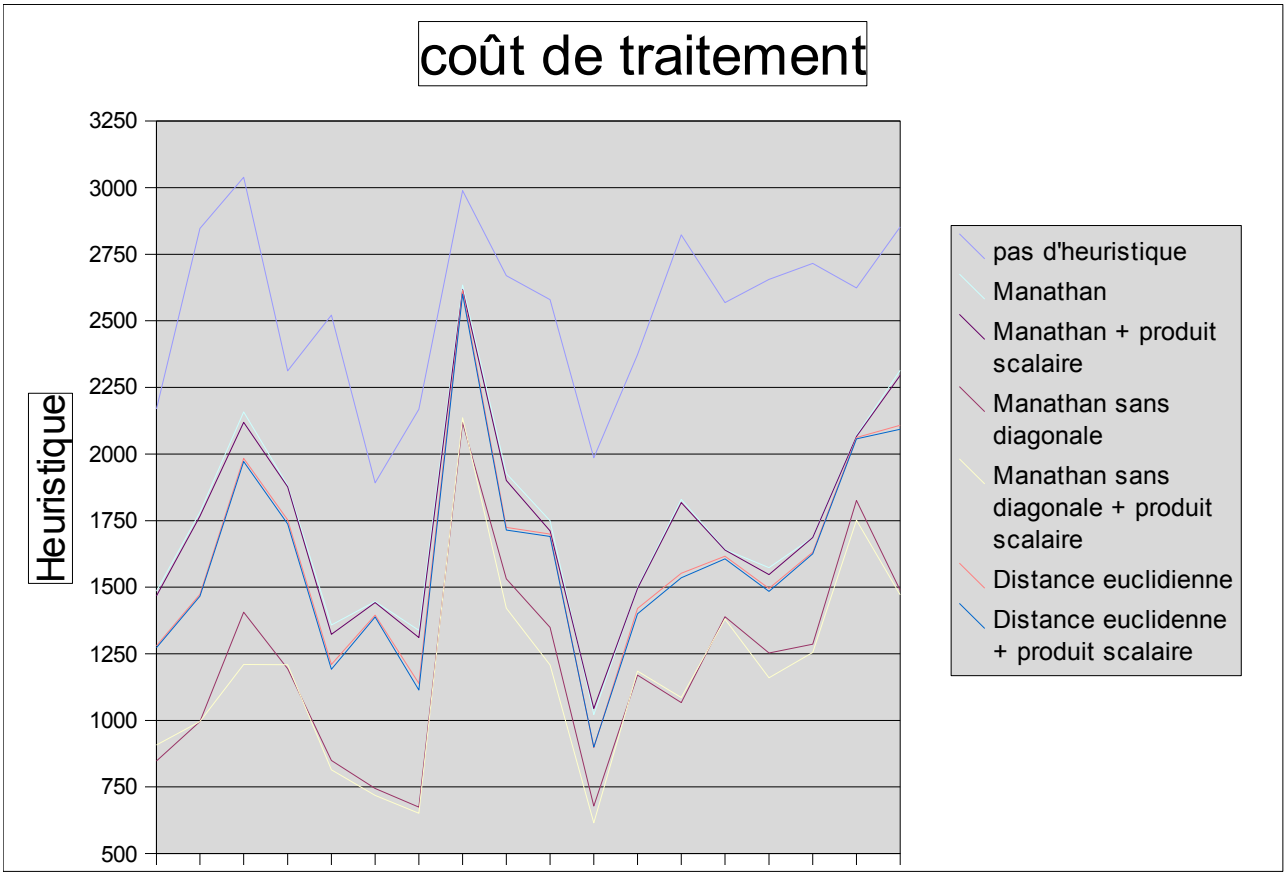
Produit scalaire:

Le produit scalaire est un calcul supplémentaire prévu pour s'ajouter à une heuristique plus générale. C'est une valeur très petite, appelé epsilon qui permet de départager des case ayant le même coût. Ce calcul consiste a favorisé le chemin le plus droit vers le point final.

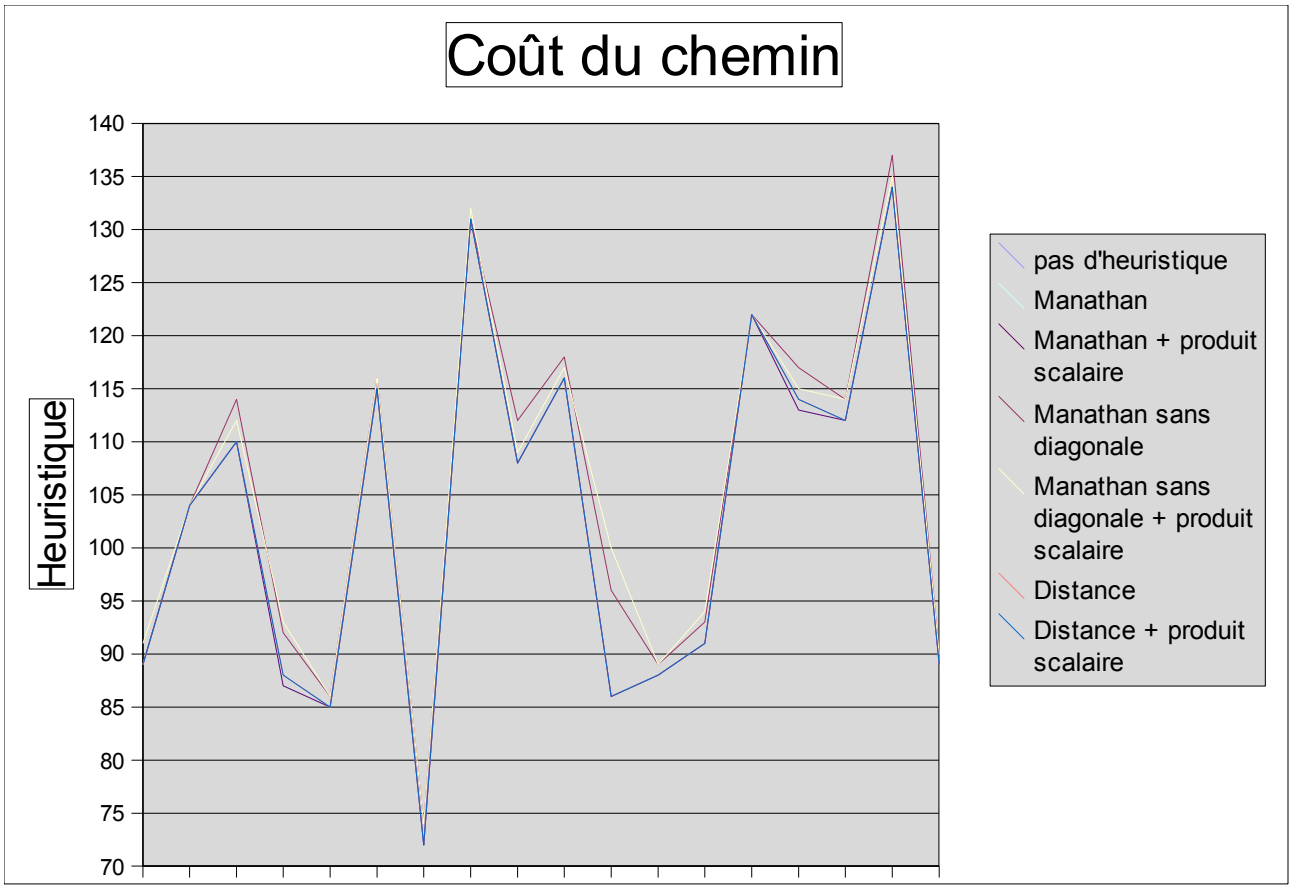


2) Performances des Heuristiques

Nous avons testé l'algorithme A* sur un terrain de 64x64 cases. A chacune des cases est attribué un coût de 1 à 3. L'algorithme recherche le chemin de coût minimum. Voici les résultats pour une série de 18 tests avec différents terrains, points de départ et points d'arrivé.



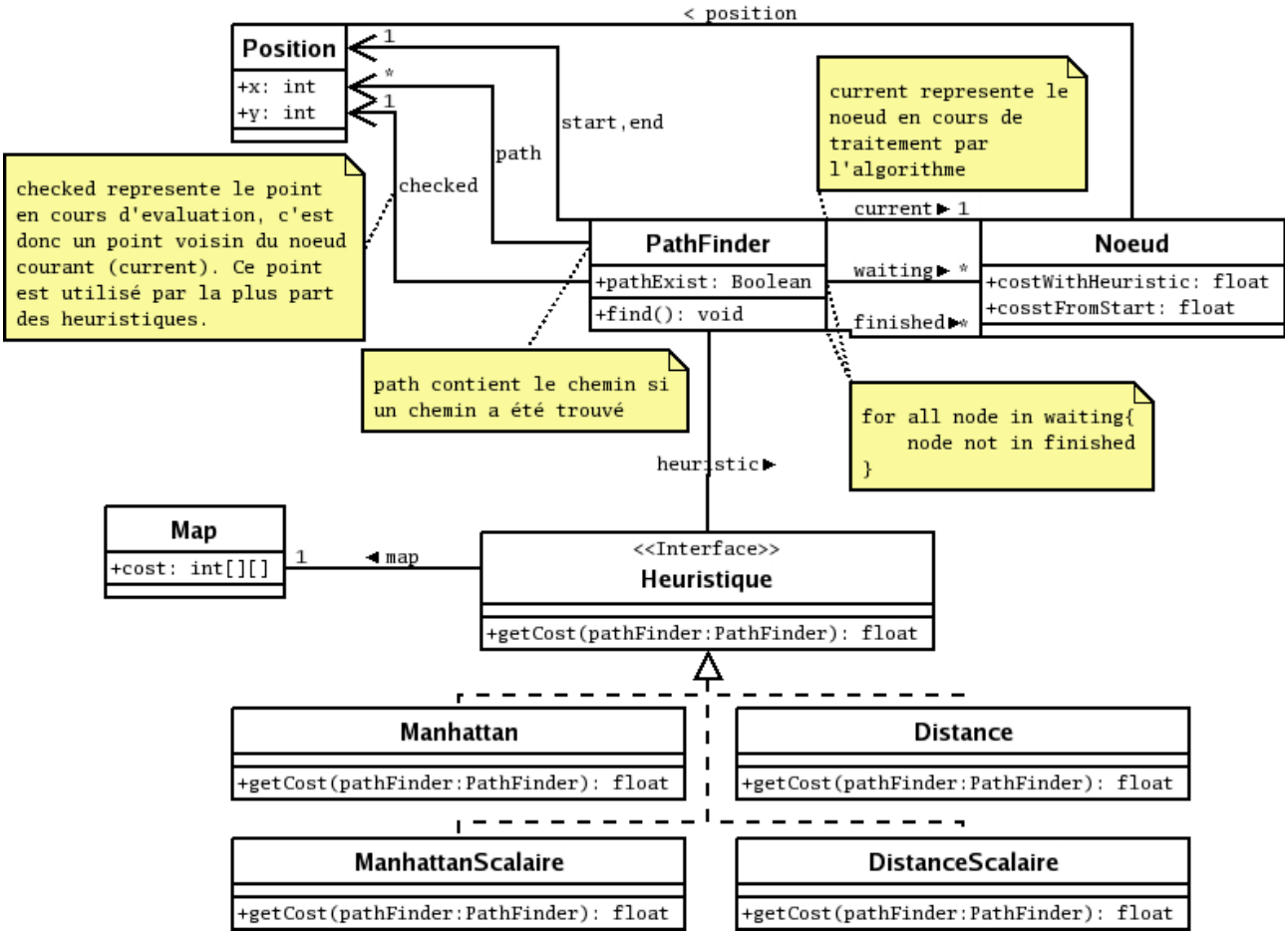
Le coût du traitement représente le nombre de cases testés par l'algorithme. On remarque qu'il y a une grande différence entre les algorithmes utilisant des heuristique et celui n'en utilisant pas. De plus, on remarque que les différentes heuristiques sans produit scalaire et avec produit scalaire sont proches, mais l'ajout du produit scalaire tend à améliorer légèrement le coût du chemin. On peut cependant se demander si le coût du calcul du produit scalaire est acceptable face aux améliorations qu'il apporte.



Le coût du chemin représente la somme des poids des cases du chemin trouvé. Notons que la différence entre le coût du chemin trouvé par l'algorithme sans heuristique est proche des coûts trouvé par les autres heuristiques. Notamment, l'heuristique de Manhattan conserve l'optimalité.

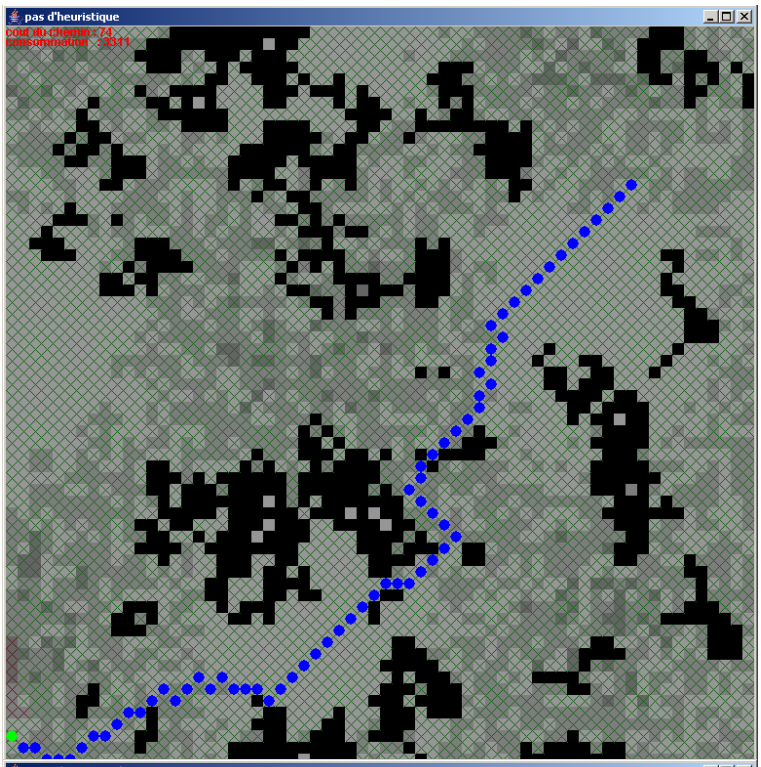
3) Diagramme de Classes UML

Ce diagramme de classe concerne une mise en oeuvre pour une carte 2D composée de cases carré (matrice).

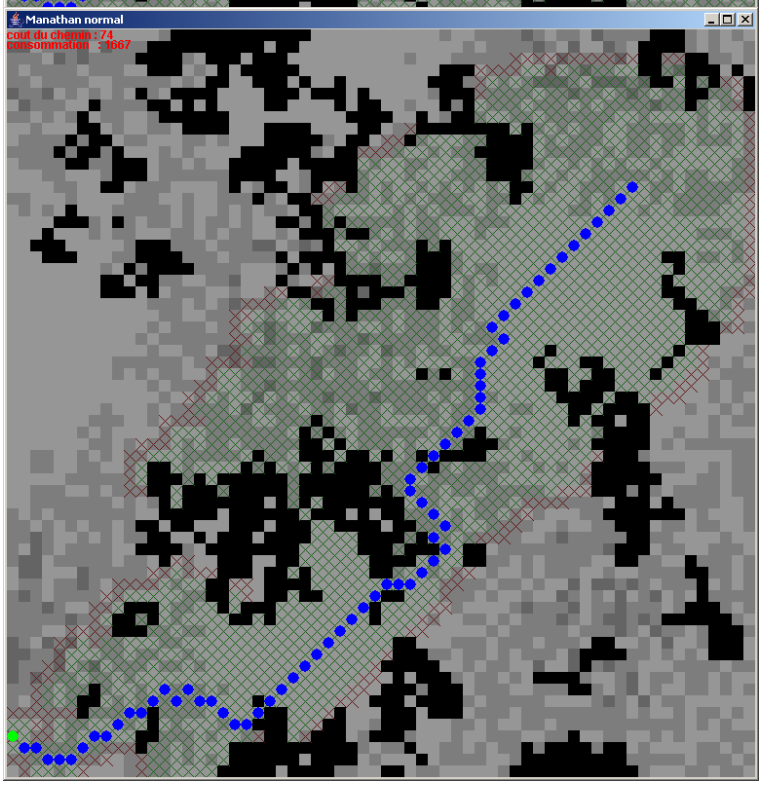


4) Résultats des Tests des heuristiques

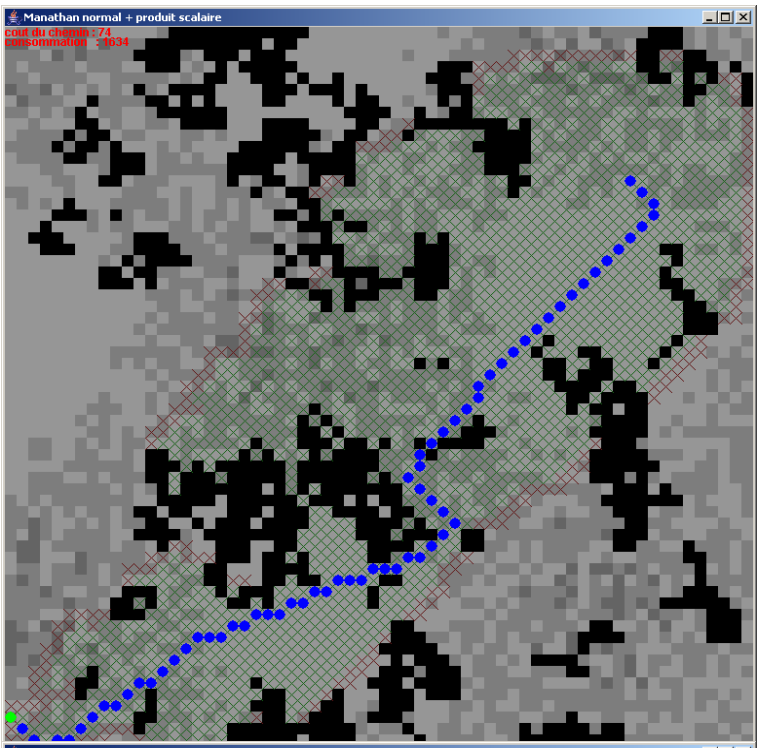
a) Sans Heuristique



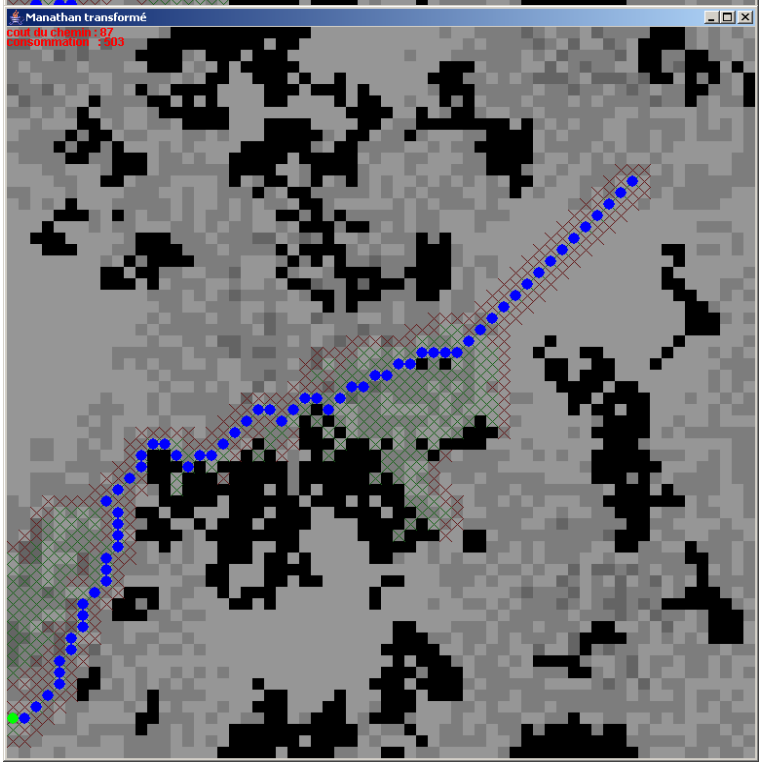
b) Manhattan



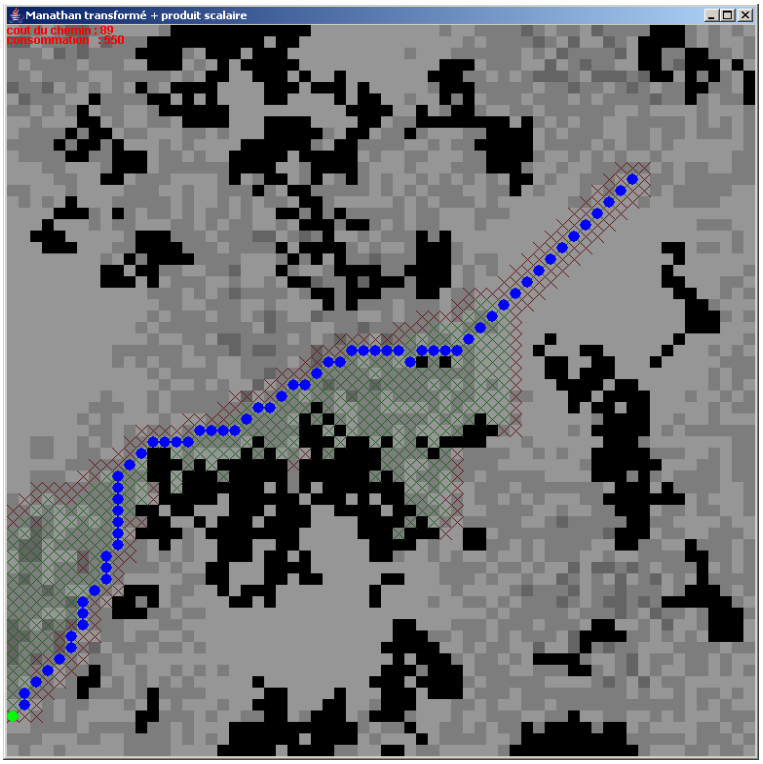
c) Manhattan + Produit scalaire



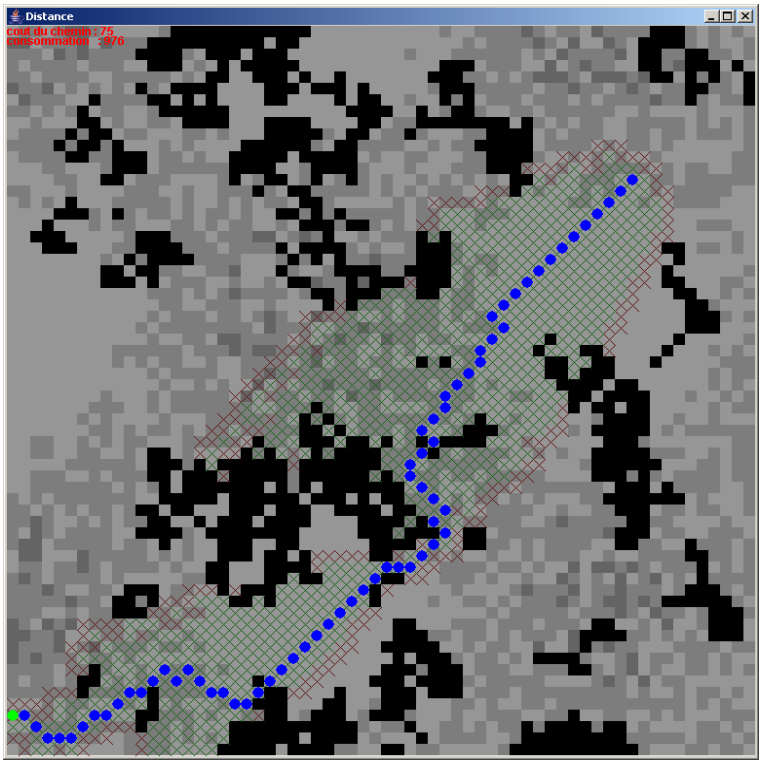
d) Manhattan sans Prise en compte de la diagonale



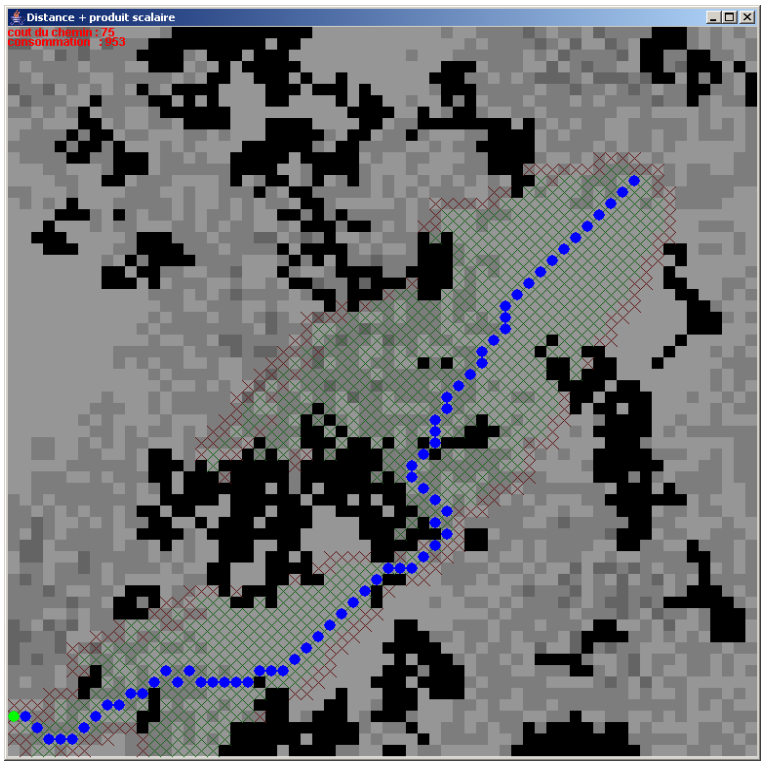
e) Manhattan sans Prise en compte de la diagonale + produit scalaire



f) Distance euclidienne



g) Distance euclidienne + Produit scalaire



Légende

- Le point vert est le point de départ.
- Les points bleus représentent le chemin parcouru.
- Les croix vertes sont les possibilités explorées.
- Les croix rouges sont les points d'arrêt d'exploration de possibilités.
- Le niveau de gris des cases indique un coût de déplacement.
- Le niveau noir est infranchissable.

-D- Logique Floue

1) Introduction

Le concept de la logique floue (Fuzzy logic en anglais) est introduit en 1965 par Lotfi ZADEH dans son son livre intitulé "fuzzy set theory" où il définit les ensembles flous et les opérateurs qui y sont associés. La logique floue est utilisée pendant 25 ans à des fins industrielles et dans le domaine de la recherche. Depuis 1990, l'utilisation de cette technique s'est généralisée dans les appareils de grande consommation (appareils photo, vidéo,...).

Au delà de l'argument publicitaire évident, il est intéressant de comprendre ce nouveau concept et de l'appliquer à certains types de problèmes d'intelligence articielle dans les jeux vidéos.

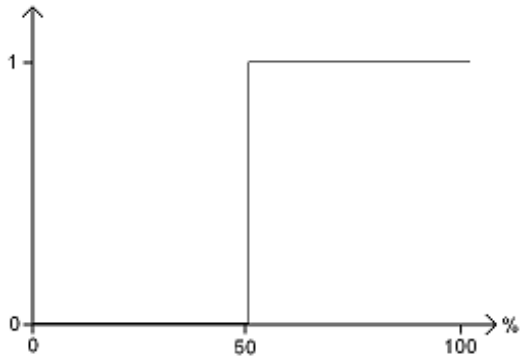
Le principe de choix par la logique floue s'approche de la démarche humaine dans le sens où les variables traitées ne sont pas des variables logiques (au sens de la logique booléenne par exemple), mais des variables linguistiques proches du langage humain de tous les jours. La logique floue permet de traiter d'une manière plus pragmatique des ensembles de données ayant une sémantique propre. Sa programmation est relativement souple car elle est basée sur des prédicats exprimés dans le langage courant.

La logique floue constitue une alternative à des problèmes que les mathématiques ne peuvent pas résoudre ou dont la représentation serait trop complexe. Nous allons énoncer dans la suite les différentes composantes de cet outil et son fonctionnement général.

2) Ensembles flous, valeurs floues

Contrairement aux variables binaires qui sont définies par les deux états "vrai" ou "faux", les variables floues sont graduées de la valeur "vrai" à la valeur "faux" selon plusieurs états intermédiaires. Par exemple si on prend dans notre cadre d'étude le prédicat "être en supériorité numérique". On peut le présenter en logique booléenne de la manière suivante :

Fig 1: supériorité numérique booléenne



On possède en abscisse le pourcentage d'unités que l'on dispose par rapport à la totalité des unités considérées (ennemies + alliées). La fonction PourcentageEff (Pourcentage effectif) qui décrit ce pourcentage est définie comme :

*PourcentageEff (nbAlliés, nbEnn)=[nbAlliés / (nbAlliés + nbEnn)]*100*

, où *nbAlliés* correspond au nombre d'unités alliés et *nbEnn* au nombre d'unités Ennemies. *nbAlliés* et *nbEnn* sont des entiers compris entre 0 et *n* (*n* entier).

On représente ici l'état de la variable à l'aide de son degré de vérité en associant la valeur 1 (degré de vérité de 100%) à la valeur "vrai" et le degré de vérité nul à la valeur "faux".

On constate que cette façon de faire est très éloignée de ce que fait l'être humain lorsqu'il résout ce type de problème. En effet, l'homme ne fait pas naturellement une distinction significative sur la supériorité numérique de son armée. Il considèrera être en supériorité numérique seulement si son groupe d'unités est sensiblement plus important au point de vue de la taille.

En définitive, la logique booléenne reste simple mais tout de même assez éloignée de la logique utilisée naturellement par l'être humain. Si on représente le même problème à l'aide de la logique floue, les variables ne sont plus binaires mais présentent une infinité de valeurs possibles entre le « vrai » et le « faux » (cf. figure 2).

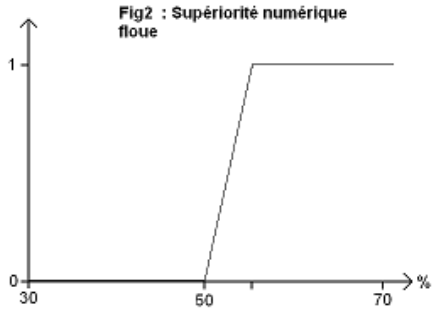


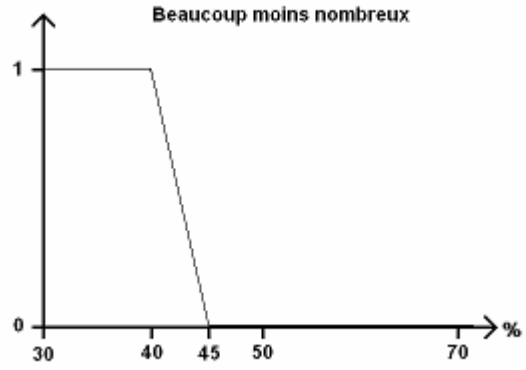
Fig2 : Supériorité numérique floue

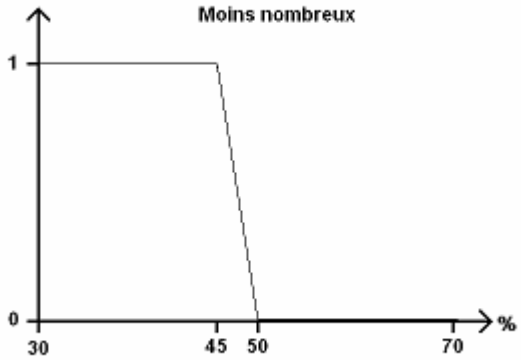
La représentation, propre à la logique floue, permet de faire intervenir des notions telles que « plus nombreux », « équivalent »... se qui la rend plus riche.

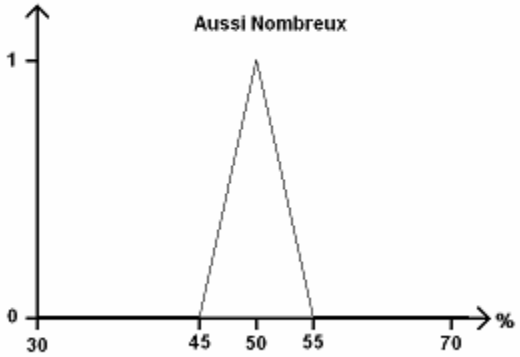
Il est bon de définir plusieurs variables floues pour une même donnée. C'est à dire de décrire de plusieurs manières une même valeur numérique. De ce fait, on va représenter un même pourcentage d'unités totales par les valeurs floues "beaucoup plus nombreux", "plus nombreux", "aussi nombreux", "moins nombreux", "beaucoup moins nombreux". Cette étape est la fuzzyfication.

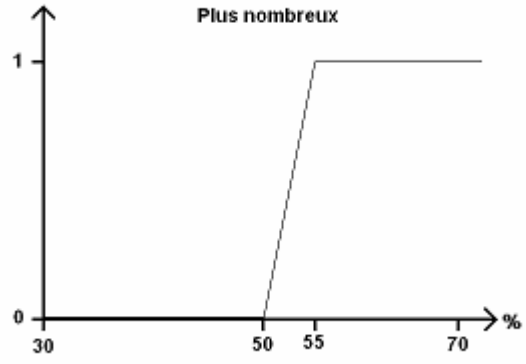
3) Fuzzyfication

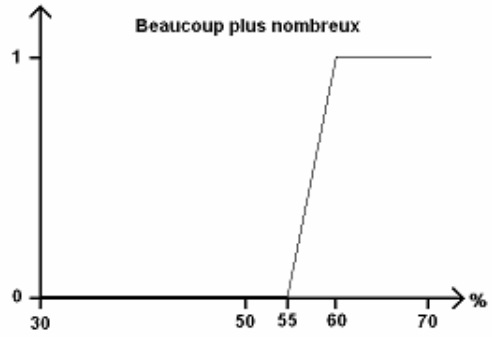
La première étape du traitement d'un problème par la logique floue consiste à modéliser chacune des entrées du système par des courbes. Ces courbes représentent les degrés d'appartenance à différents états pour ces entrées. Pour les valeurs floues "beaucoup plus nombreux", "plus nombreux", "aussi nombreux", "moins nombreux", "beaucoup moins nombreux", on obtient les graphes suivants :



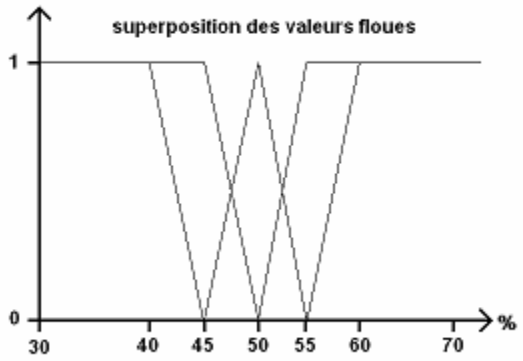








Si l'on superpose les graphiques précédents, on voit que les courbes se chevauchent :



Ce chevauchement est tout à fait logique : Si on rajoute une unité à notre groupe, notre statut ne passe pas strictement de "aussi nombreux" à "plus nombreux". Plus on rajoute d'unités, plus le statut "aussi nombreux" sera faux et plus le statut "plus nombreux" sera vrai. Ce procédé sera en outre une garantie de stabilité des asservissements basés sur la logique floue.

La fuzzyfication des variables d'entrée est une phase délicate du processus mis en oeuvre par la logique floue. Elle est souvent réalisée de manière itérative et requiert de l'expérience.

4) Opérateurs flous

Comme en logique booléenne, on dispose des opérateurs "et", "ou", "non" pour relier les différentes valeurs floues. Dans le cas de la logique booléenne, les opérateurs sont définis de manière unique. Pour la logique floue il existe différentes manières de représenter ces opérateurs :

Opérateur	Opération sur le degré de vérité des variables
et	minimum
	produit
ou	maximum
	valeur moyenne
non	complément à 1

On va préfèrer les minimum et maximum pour que le calcul soit efficace et que l'implémentation soit simple. Il sera toutefois intéressant de tester les différences avec le système précédent qui est un peu plus riche.

5) Règles d'inférence

Après avoir "fuzzyfié" les variables d'entrée et de sortie, il faut établir les règles liant les entrées aux sorties. En effet, il ne faut pas oublier d'analyser l'état ou la valeur des entrées du système pour déterminer l'état ou la valeur de toutes les sorties pendant le traitement.
La logique floue fonctionne suivant le principe suivant : plus la condition sur les entrées est vraie, plus l'action préconisée pour les sorties doit être respectée. Par exemple on peut écrire :

Si *(Beaucoup plus nombreux)* ***alors*** *(Aller attaquer)*

Dans la pratique, les règles font appel à des conditions d'entrée plus complexes mettant en oeuvre des conditions logiques du type, "ou", "et" ou "non". On va prendre pour l'exemple nos valeurs floues introduites au départ et on va en rajouter une autre. Rajoutons, sans le faire de explicitement, les catégories de valeurs floues "Etats de santé allié" et "Etat de santé ennemi". Chacune de ces catégories est divisée comme notre état numérique par le biais de plusieurs adjectifs. On obtient alors une règle d'inférence de la forme :

Si *(Beaucoup plus nombreux)* ***et*** *(Etat de santé allié bon)* ***et*** *(Etat de santé ennemi mauvais)* ***alors*** *(Aller attaquer)*

On obtient donc une valeur floue "Aller attaquer". Cette valeur floue est égale à l'évaluation de la partie gauche de la règle d'inférence. Après avoir défini un ensemble de règles, on doit exploiter leurs résultats.

6) Composition de règles ou défuzzyfication

Pour un état donné des entrées, plusieurs règles peuvent être validées simultanément et fournir des consignes différentes pour les sorties. Il faut donc disposer d'une méthode de composition des règles pour obtenir la valeur finale des sorties. La composition des règles doit tenir compte de toutes les règles qui sont validées au prorata de leur degré de validité.

Il existe au moins 3 types de composition des règles :

- La *technique du maximum* est la plus simple : elle consiste à ne considérer, pour chaque sortie, que la règle présentant le maximum de validité. Cette règle, simple voire simpliste, ignore les règles secondaires qui peuvent néanmoins être importantes pour le fonctionnement et la stabilité du système. Elle est peu employée.

Dans le cadre de ce projet, les résultats des règles d'inférence ne servent pas à quantifier une influence sur une sortie mais à choisir la bonne action à effectuer. Cette méthode est donc la plus appropriée car les valeurs floues validées en sortie représentent l'action à effectuer et la valeur qui lui est associée, son degré de pertinence. L'action la plus pertinente sera alors réalisée. Le problème des degrés de priorité sera résolu par un poids associé à la valeur floue avant traitement.

- *La technique de la moyenne pondérée* est plus évoluée. Elle considère, comme valeur de sortie, la moyenne des valeurs préconisées par chaque règle, pondérées par leurs degrés respectifs de validité.

Exemple : soient deux règles donnant les valeurs de consignes suivantes :

- règle 1 : valeur intermédiaire (50) avec une validité de 0,8,

- règle 2 : valeur forte (78) avec une validité de 0,2,

On prend comme valeur de sortie : [(50 x 0,8) + (78 x 0,2)] / (0,8 + 0,2) = 56.

Cette méthode, également simple à mettre en oeuvre, présente néanmoins certaines ambiguïté sur la valeur de sortie.

- *La technique du centre de gravité* : cette technique étant assez difficile à expliquer de manière résumée et sachant que la défuzzification est une étape qui n'entre pas dans le cadre de ce projet nous ne détaillerons pas plus ce point.

7) Conclusion

L'utilisation de la logique floue, de part son caractère non "strict", s'effectue donc dans des domaines très variés lorsque un "flou" persiste, notamment dans le domaine de l'économie, des sciences naturelles et des sciences humaines. On peut aussi l'utiliser dans des systèmes complexes dans lesquels la modélisation est difficile, voire impossible, dans des systèmes contrôlés par des experts humains, ou bien encore dans des systèmes ayant de nombreuses entrées/sorties continues ou discontinues et des réponses non linéaires. De manière plus générale, la logique floue est utilisée quand l'observation humaine est à l'origine d'entrées ou de règles de contrôle du système.

La logique floue a déjà fait ses preuves dans les domaines d'applications tels que la gestion de projet, la modélisation de prix et d'analyse démographique des marchés, l'analyse de rentabilité de compagnies, la détection de fraude à la protection sociale ou l'identification de criminels, la gestion des aspirateurs, des systèmes de ventilation et de régulation thermique, le pilotage des systèmes d'autofocus des appareils photos, le système d'approche d'une station orbitale pour la navette spatiale américaine, la lecture automatique et la reconnaissance de caractères ou bien encore le traitement d'images.

-E- Test d'un comportement d'armée simple

1) Le but

Le but de ce test est de créer un petit logiciel permettant de tester des algorithmes de contrôle d'armées. Nous pourrons ainsi voir des comportements émergents et des stratégies avantageuses. Le monde mis en place est simple et facile à comprendre. Par la même occasion, cela nous permet de tester une implémentation de la recherche de chemin grâce à l'algorithme A*.

Ce test doit être simple et rapide à développer car il ne sert qu'à nous faire une idée basic de l'IA final. Pour l'instant, nous n'avons aucune contrainte temporelle donc le langage JAVA sera utilisé car il permet de développer rapidement et simplement de petit logiciel.

2) La règle mise en place

Les unités évolue sur un terrain composé de cases carrées. Chaque case est soit franchissable soit bloquante. Sur le plateau de jeu, deux armées sont modélisées grâce à différentes unités placées pseudo aléatoirement sur le terrain : L'armée bleue à gauche et l'armée rouge à droite. Chaque armée possède le même nombre d'unités mais le type des unités est tiré au hasard. La part de hasard nous permet ainsi de voir différentes situations suivant les types d'unités présentes, leurs positions et leurs nombres.

Nous avons aussi prévu le fait de calculer les zones de danger qui représentent le danger de se faire attaquer et de se prendre plus ou moins de dégât en un certain point suivant les unités adverses présentent. L'intensité du danger représente ainsi la probabilité de se faire toucher mais aussi le niveau de dégât que l'unité peut recevoir.

D'autre part, nous autorisons le déplacement et l'attaque en diagonal.

3) Les unités

Nous avons mis en place 2 types d'unités :

- l'unité rapide (symbolisé par un rond) qui peut se déplacer de deux cases à chaque frame et qui possède une force de 1. Elle génère une faible mais large zone de danger.
- l'unité forte (symbolisé par un carré) qui se déplace d'une case à chaque frame et qui possède une force de 2. Elle génère une forte mais petite zone de danger.

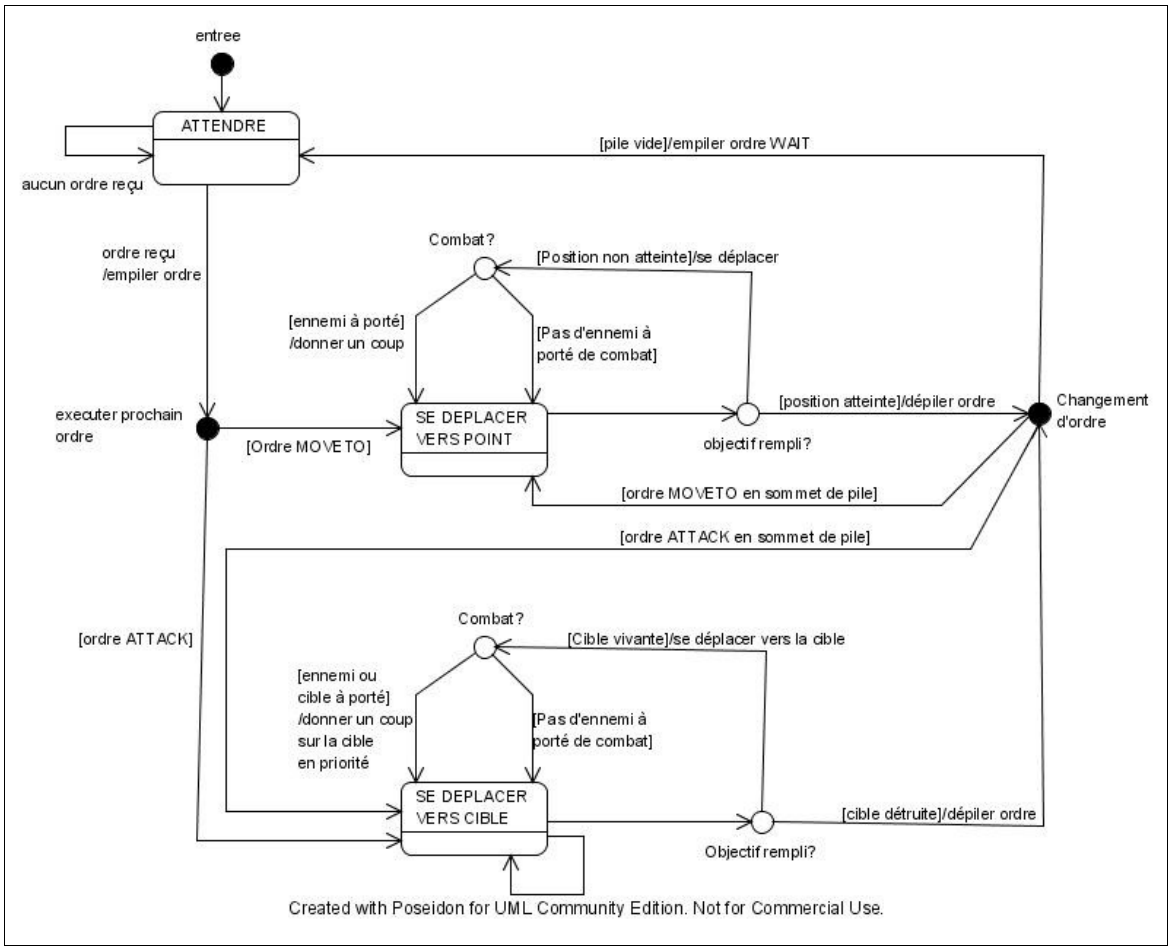
Chaque unité possède le même nombre de point de vie : 7 points de vie. Lorsque qu'une unité est sur une case, les autres unités ne peuvent pas se déplacer : une seul unité par case.

L'unité voit sont comportement déterminé par un automate à état fini. L'unité peut ainsi recevoir différents ordres (se déplacer, attaquer…) et ensuite, elle se gère elle-même jusqu'au prochain ordre. Ainsi, il n'est pas nécessaire de donner un ordre à l'unité à chaque frame. Grâce à une pile interne, une unité peut changer d'objectif pour un sous objectif en poussant ce nouvel objectif sur la pile d'ordres. Par exemple, si l'unité se fait attaquer pendant qu'elle se déplace en un point, elle peut riposté pour se défendre. Ensuite, une fois l'ennemi mort, elle reprendra sont objectif précèdent. Par contre, dès que l'IA général envoie un ordre à l'unité, celle-ci dépile tout ces objectifs et s'occupe de remplir ce dernier ordre reçu.

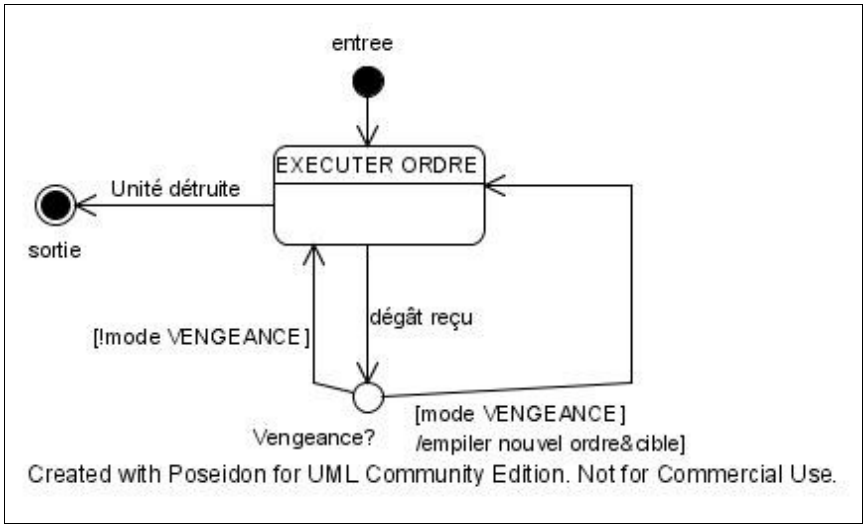
L'unité peut recevoir 3 types d'ordre :

- Attendre : l'unité reste sur place en attendant un ordre.
- Se déplacer à un point : L'unité se déplace en un point donné le plus rapidement en évitant au mieux les ennemis.
- Attaquer : L'unité se déplace vers l'unité cible le plus rapidement possible en évitant au mieux les autres ennemis. Si l'unité cible est allié, on ne fait que la suivre sans l'attaquer, sinon, on attaque.

A tout moment, si l'unité est attaquée, elle peut mettre son objectif de coté pour se défendre et détruire l'unité qui l'a offensée. Dès que la pile d'ordre est vide, l'unité se met en attente.



L'automate à états fini représentant le comportement de l'unité



L'automate à états fini régissant les changements d'objectifs de l'unitée

4) Le path-finding

Le path-finding utilisé permet de chercher un chemin au travers de la carte en évitant les obstacles avec en plus le contournement des zones de danger (faible pondération pour le choix du chemin) afin d'éviter les unités ennemis. Le chemin prend aussi en compte les unités ennemies pour les contourner.

Nous avons utilisé l'heuristique de Manhattan prenant en compte le déplacement en diagonal avec une petite pondération en plus qui essaye de faire déplacer l'unité le plus directement possible vers le point final.

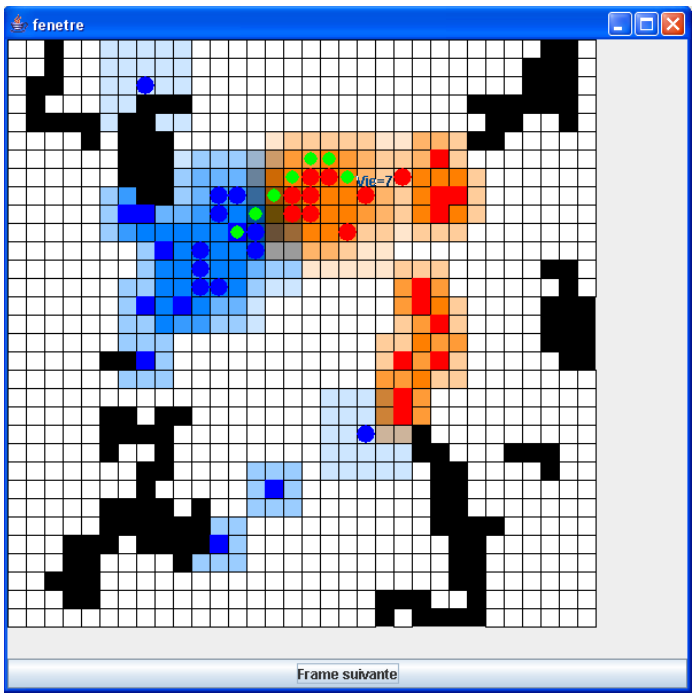
5) Les algorithmes de contrôle

Nous avons mise en place différent type d'algorithme de contrôle de l'armée. Nous ne pouvons parler que d'IA réactive. Voici les différents algorithmes testés :

- On sélectionne l'unité la plus isolée (celle qui est dans une zone de danger la moins forte) et toutes les unités de l'armée l'attaquent jusqu'à ce qu'elle meure. Ensuite, on sélectionne une nouvelle unité à tuer.
- Toute l'armée suit un chef qui lui attaque l'unité ennemie la plus isolée.
- Chaque unité de l'armée doit tuer une unité de l'armé adverse.
- Chaque unité attaque l'unité ennemie la plus proche.
- On essaye de faire attaquer les unités faibles de l'ennemi par nos unités rapides qui font moins de dégât. Les unités ennemis fortes seront attaqués par nos unités fortes.
- On regroupe notre armée en petits groupes. Chaque groupes tentent d'attaquer un autre groupe. Si un groupe est un petit peu en sous nombre, les unités se regroupent en bloc en attendant l'ennemi. Si la bataille semble perdu d'avance, le groupe fuit.

Une des difficultés rencontrées est de ne pas envoyer d'ordres à chaque unités à chaque frames : Savoir quand est-ce qu'il faut dire à une unité d'attaquer ou de se retirer d'une zone sans la marteler d'ordres. La difficulté est de savoir à quel moment on doit modifier sa stratégie : quand fuir? Quand réattaquer? D'où l'importance d'un comportement d'unité bien réalisé : elle devra réagir au mieux à certaines situations durant un laps de temps non négligeable. Le fait de donner le moins d'ordre permet d'optimiser la rapidité de jeu (n'oublions pas que nous sommes dans un contexte temps réel), mais aussi d'éviter des effets d'hésitation rapide entre deux états : fuir, attaquer, fuir, attaquer et ainsi de suite.... Cet effet est très désagréable à voir s'il est très répété sur un laps de temps très cours.

L'autre problème rencontré est celui du groupage d'unité de manière cohérente selon les zones puis ensuite de donner une unité ou un groupe d'ennemies à attaquer. Savoir reconnaître les différents groupes ennemis n'est pas chose facile.



Le logiciel représentant chaque armées avec leur zone de danger. Une unité est sélectionner : on voit sa vie restante, son objectif et le chemin qu'elle désire emprunter pour le remplir.

6) Les comportements émergent

Le fait d'avoir a disposition des unités plus rapides que d'autres permet l'interception d'unités en fuite plus lente. Le combat ralenti l'ennemi en fuite en attendant que les unités plus puissantes arrivent.

La fuite est un très bon moyen de survivre. La fuite d'une unité vers un autre groupe permet de voir des retournement de situation ou d'avantage. La fuite avec séparation permet d'affaiblir l'ennemi en le divisant (diviser pour régner) et nous observons de nouveau des retournements de situation dans le cas ou l'avantage n'est pas certain.

Après avoir effectué différent testes avec différents algorithmes, il s'est avéré que le fait de grouper les unités donnait un avantage considérable. En lançant une partie 'unités groupées' (UG) contre 'unités isolées ayant chacune un ennemi à tuer' (UI); il est évident que la stratégie UG apparaît plus puissante. Nous avons observer un score de 17 à 3 pour UG contre UI. Alors que UG contre UG donne 10-10 et UI contre UI donne un score de 11-9 (on remarque bien la l'équivalence des stratégies).

-F- Modélisation du Moteur décisionnel

1) Ensemble des Actions envisageables

La modélisation d'un moteur de décision passe par la phase de définition des besoins et des actions nécessaires pour modifier le monde environnant. Dans cette première étape nous allons lister l'ensemble des actions dont une entité doit disposer ainsi que les celles dont le superviseur se sert pour influer sur ces entités. L'entité est un groupe d'unités ou une unité seule. Les actions de base sont pour le groupe comme pour l'unité majoritairement identiques. Le superviseur est un moteur flou qui supervise une stratégie globale et coordonnée sur un ensemble d'entités.

- *Fuir :* éloigne l'entité du danger, l'entité s'en éloigne en bloc.
- *Se disperser :* éloigne chaque unité du danger et des autres unités alliées.
- Lancer charge : le superviseur demande à l'entité d'attaquer.
- *Charger :* lance une attaque, conserve la formation définie. Suit l'entité visée si elle s'enfuit.
- *Tenir Position :* défend même en situation d'échec. Laisse partir l'entité adverse si elle choisit de s'enfuir.
- Patrouiller : effectue des aller retour entre différents points définis.
- Reconnaissance : déplacement vers les zones inexplorées.
- Regrouper : fusionne deux entités.
- Diviser : divise une entité en deux.
- Couper : se déplace sans chercher à éviter les endroits risqués ou exposés.
- Se faufiler : se déplace vers un point en cherchant le chemin le moins risqué.
- Se cacher : cherche une zone où l'entité n'est plus visible et non exposée.
- Choisir formation : réorganise les unités selon un schéma défini.
- Réorganisation : réorganise les unités dans une phase d'attaque. Les schémas de formation ne sont pas définis statiquement, on peut demander à un sous groupe d'attaquer des unités spécifiques.
- Monter à cheval : monte une unité seule sur un cheval.
- Descendre de cheval : descend une unité seule de son cheval.
- Suivre : va vers une entité et la poursuit même si cette entité se déplace.
- *Groupe rejoint :* prévient le superviseur que l'entité est prête a fusionner avec une autre entité. Et demande la fusion.
- Être caché : prévient le superviseur que l'entité est cachée.
- *Lancer fuite :* lance l'action de fuir mais ne décide pas comment faire la fuite.
- Fin combat : prévient le superviseur que le combat est terminé.
- Ennemi hors porté : prévient le superviseur que l'ennemi est devenu hors porté pour remettre la stratégie entamée en cause.
- Déplacer : Demande un déplacement à l'entité. Une coordonnée peut lui être donné.
- Voir : Prévient le superviseur que l'entité a vu une unité ennemi.

- Être touché : L'entité signale qu'elle a subit des dégâts.
- Réveiller : le superviseur demande à une unité de se réveiller.
- Endormir : l'entité s'endort et attend d'être réveillée.
- Réveil d'une unité : L'entité prévient le superviseur qu'elle est réveillée.
- Point atteint : L'unité s'endort après avoir atteint son but.

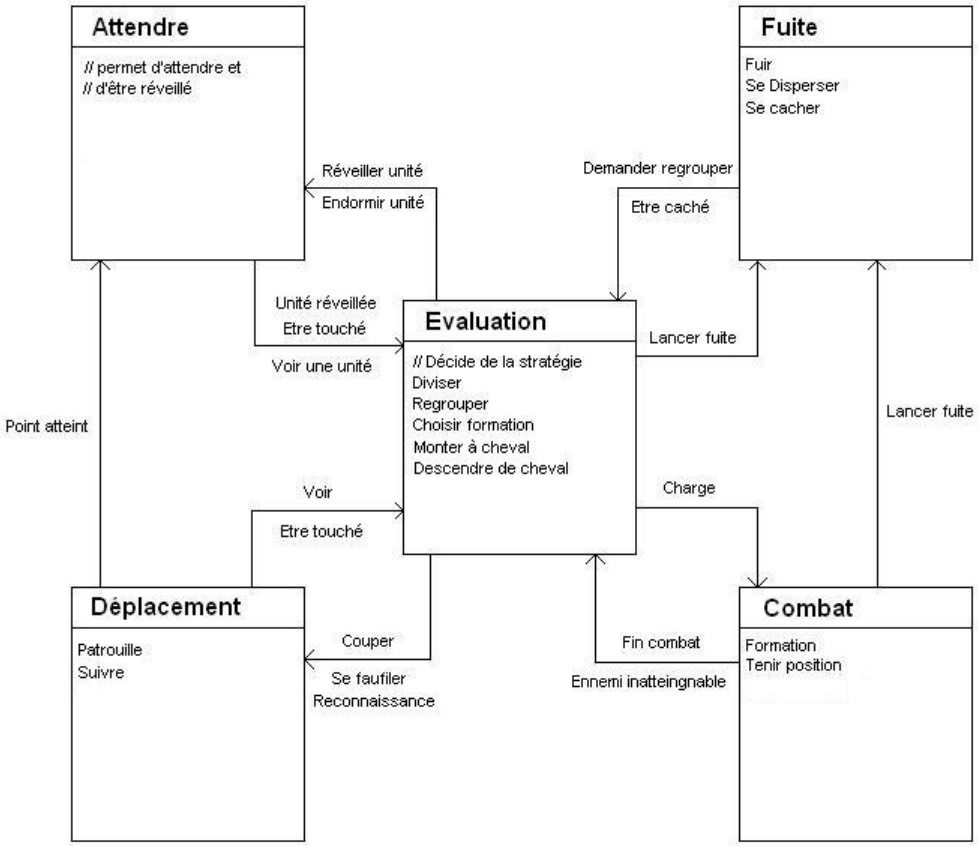
2) Etats

Pour organiser les actions, et faire en sorte de limiter les séquences d'actions illogiques, on va définir des États. L'idée est de restreindre l'étendu des actions quand l'entité a entamé une manoeuvre. Ensuite on va classer les actions de base par état dans lequel l'action pourrait être effectuée et son état après avoir été effectué dans un tableau de transition. Le résultat est un automate.

États	Évaluation	Combat	Déplacement	Fuite	Attente
Évaluation	-Diviser -Regrouper -Choisir formation -Monter à cheval -Descendre de cheval	-Lancer charge	-Déplacer	-Lancer fuite	-Réveiller -Endormir
Combat	-Fin combat -Ennemi hors portée	-Charge -Tenir Position -Réorganiser -Descendre de cheval monter -Monter à cheval		-Lancer fuite	
Déplacement	-Voir -Être touché		-Patrouiller -Suivre -Reconnaissance -Couper -Se faufiler -Descendre de cheval -Monter à cheval		-Point atteint
Fuite	-Groupe rejoint -Être caché			-Fuir -Se disperser -Se cacher -Descendre de cheval -Monter à cheval	
Attente	-Être touché -Voir -Être réveillé				

3) Automate

Du tableau précédent, on obtient l'automate suivant



L'état é*valuation* est le coeur de la stratégie globale du moteur décisionnel. C'est lui qui coordonne les actions des différents groupes en leur donnant des directives. Il est prioritaire sur les actions décidées par l'entité elle-même. C'est un processus qui découpe le groupe de départ (toutes les unités) en sous groupe pour donner des directives à ces sous unités. Il crée un nouveau processus à chaque création de sous groupe, et pour fusionner deux groupes, il tue les deux processus et en recréé un nouveau à partir des données des deux anciens groupes. Il est initialisé dans l'état évaluation. Tous les processus tournent en parallèle. Afin de garder les petits moteurs décisionnels réactifs, on pourra les faire marcher plus souvent que le superviseur car une stratégie globale n'a pas besoin d'être remise en cause très souvent.

4) Règles de Décisions

Pour chaque état de l'automate, l'entité devra choisir entre les différentes actions qui lui sont proposées. Les actions étant différentes selon l'état, les règles qui choisissent l'action la plus appropriée sont elles aussi dépendantes de l'état dans lequel se retrouve l'unité. On associe donc un moteur de décision à chaque état (état attendre non inclus car cet état endort l'entité). On dispose pour ces règles de données statiques et dynamiques données par l'analyse de terrain, l'historique des rencontres entre les entités, la position des entités alliées et ennemies visibles, etc... Chaque moteur doit choisir des actions en disposant de toutes les informations mais sans avoir de notion de stratégie globale, sans coordination entre entités. Ces moteurs sont réactifs, il choisissent de répondre à une situation sans affecter les actions des autres

entités. Ces réactions sont à l'image d'un groupe d'unité qui agit sans les ordre d'un général pour coordonner les différentes manoeuvres.

a) Déplacement

Patrouille

- Si place forte à proximité
- Si plusieurs zones à fort passage à proximité

Reconnaissance

- Si peu d'unités visibles
- Si zones inexplorées proches
- Si début de partie
- Si objectif = attaquer
- Si objectif != défendre
- Si nombre d'unités récemment vues petit

Suivre

- Si objectif = escorte
- Si nombre d'unités en train d'escorter est faible
- Si proche de l'unité à escorter

Couper

- Si beaucoup d'unités visibles ^ unités ennemies éloignées
- Si beaucoup d'unités récemment vues ^ unités ennemies éloignées
- Si passage emprunté beaucoup de fois
- Si zone peu risquée
- Si passage récemment emprunté

Se faufiler

- Si peu d'unités visibles
- Si peu unités récemment vues
- Si passage peu emprunté
- Si passage risqué
- Si passage pas emprunté récemment
- Si unités visibles proches

b) Combat

Tenir position

- Si beaucoup d'unités lentes dans l'entité
- Si objectif = défendre ^ on est à l'endroit que l'on doit défendre

- Si on est sur une place forte
- Si beaucoup d'unités alliées proches

Lancer fuite

- Si entité moins grosse
- Si type global adverse est fort contre entité
- Si points de vie globaux des alliés est faible
- Si beaucoup d'unités alliées à proximité
- Si vitesse globale du groupe est grande
- Si pas beaucoup d'archers ennemies ^ accès aux archers bloqué
- Si l'ennemi est sur une place forte

Attaquer

- Si pas beaucoup d'unité ennemies
- Si ennemi est faible en points de vie
- Si le type global de l'entité alliée est fort contre type global de l'entité ennemie
- Si ennemi n'est pas sur une place forte
- Si unité alliée proche et en état combat

Descendre de cheval

- Si ennemi proche ^ beaucoup d'archers ^ archers en forêt
- Si zone à très faible mobilité ^ forêt à proximité

c) Fuir

Disperser

- Si sur une zone à fort degré de liberté
- Si entités alliées loin

Se cacher

- Si forêt proche ^ dans la direction opposée à l'ennemi
- Si beaucoup de cavaliers chez l'ennemi
- Si pas beaucoup de chevaux dans l'entité alliée

Rejoindre

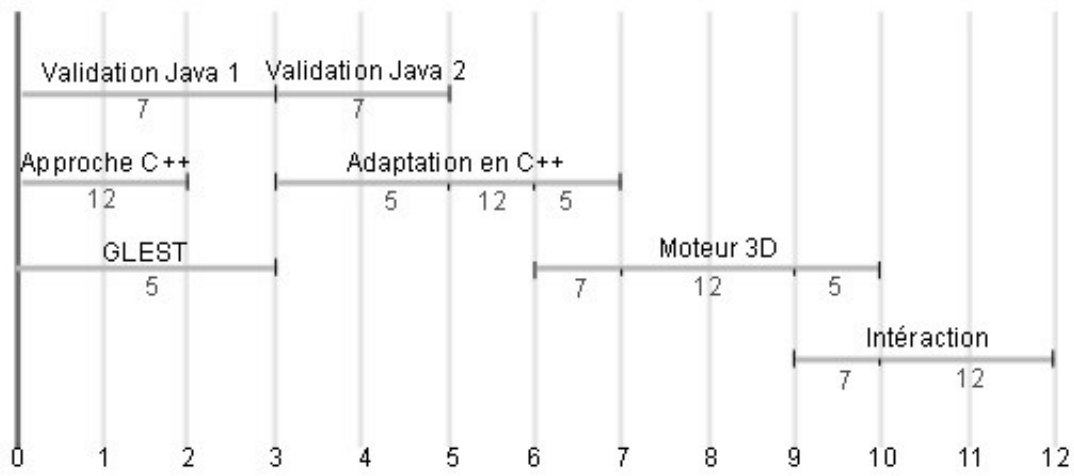
- Si entité proche ^ en grand nombre

-V- Planning et Organisation

Pour pouvoir mené le projet au mieux, nous avons définis plusieurs versions intermédiaires. Chacune d'entre elle sert à vérifier un point du projet et le valider pour pouvoir passer à la suite.

- La première version permet de vérifier l'architecture globale. Le but est que cette version soit validée le plus rapidement possible. Pour cela, nous la réaliserons en Java, langage que nous connaissons le mieux. Le but étant uniquement la validation de l'architecture, cette version ne mettra en oeuvre qu'une interface 2D et une sortie texte. La réalisation de cette version se décompose en deux parties, que nous appellerons « validation Java 1 » et « validation Java 2 ». Cette division a pour but de valider une première partie de l'architecture (la partie 1) au plus vite pour pouvoir commencer les autres tâches en parallèle. La « validation Java 1 » s'attachera à confirmer la bonne gestion des unités, la structure des actions et le *pathfinder*. La « validation Java 2 » se concentreras sur deux points : le moteur d'IA, en validant le moteur d'inférence et la gestion des valeurs floues, ainsi que la boucle de perception en validant l'analyse de terrain et l'analyse de l'armée ennemie.
- La deuxième version sera l'adaptation de l'architecture dans le langage C++. L'élaboration de cette version pourra commencer dès que la première partie de l'architecture sera validée. Elle ne mettras pas en oeuvre d'interface graphique mais uniquement des sorties texte qui pourront facilement être comparées aux sorties texte de la version 1.
- La troisième version proposera l'ensemble du moteur d'IA de la version précédente intégré avec le moteur 3D de GLEST. Les graphismes resteront cependant très limités. La validation de cette version constitue le but principal de notre projet.
- La quatrième version est une version facultative qui met en oeuvre des graphismes plus élaborés et l'interaction avec le joueur. Le but de cette version est de finaliser le projet afin de mieux apprécier les capacités de l'IA.

Voici les différentes relations entre les tâches :



Chaque trait vertical correspond à la fin d'une semaine. Les chiffres en dessous des tâches correspondent au nombre de personne dédiés à cette tâche, certaines personnes pouvant être affectées à plusieurs tâches. La tâche d'adaptation de l'architecture en C++ commence dès que la première partie d'évaluation de l'architecture en Java est terminée. Les cinq personnes affectées à cette tâche commenceront le projet par étudier l'interfaçage avec le moteur 3D de GLEST afin d'anticiper les problèmes possibles. De plus, une tâche « Approche C++ », affectées à toutes les personnes, permettra à chacun de s'initier aux particularités de C++. Les tâches « Moteur 3D » et « interaction » commence par une partie d'analyse ou seuls quelques membres de l'équipe y sont consacrés. L'objectif est de préparer au mieux le travail à douze sur la tâche.

-VI- Conclusion

Un moteur décisionnel pour un jeux vidéo est en soit la meilleure manière de faire un projet sur le domaine de l'intelligence artificielle. De part la taille de l'univers et sa complexité, le nombre de possibilité est restreint. En contrepartie le jeu se doit d'être en temps réel et donc d'être mis en oeuvre de manière efficace. Ceci répartit la complexité entre la phase de développement et de mise en oeuvre.

La boucle perception, décision, action caractérise l'intelligence artificielle. Ce schéma général permet de classer les différents outils utilisés pour créer ce moteur. Par exemple la logique floue se classe dans la décision, le pathfinding dans l'action ou encore l'analyse de terrain dans la perception. C'est ainsi que la partie analyse de ce projet nous a permis de diviser et de répartir un travail relativement complexe, d'apprendre à recueillir des informations et de les exploiter, de se spécialiser et d'avoir chacun son importance tout en s'intéressant au travail des autres.

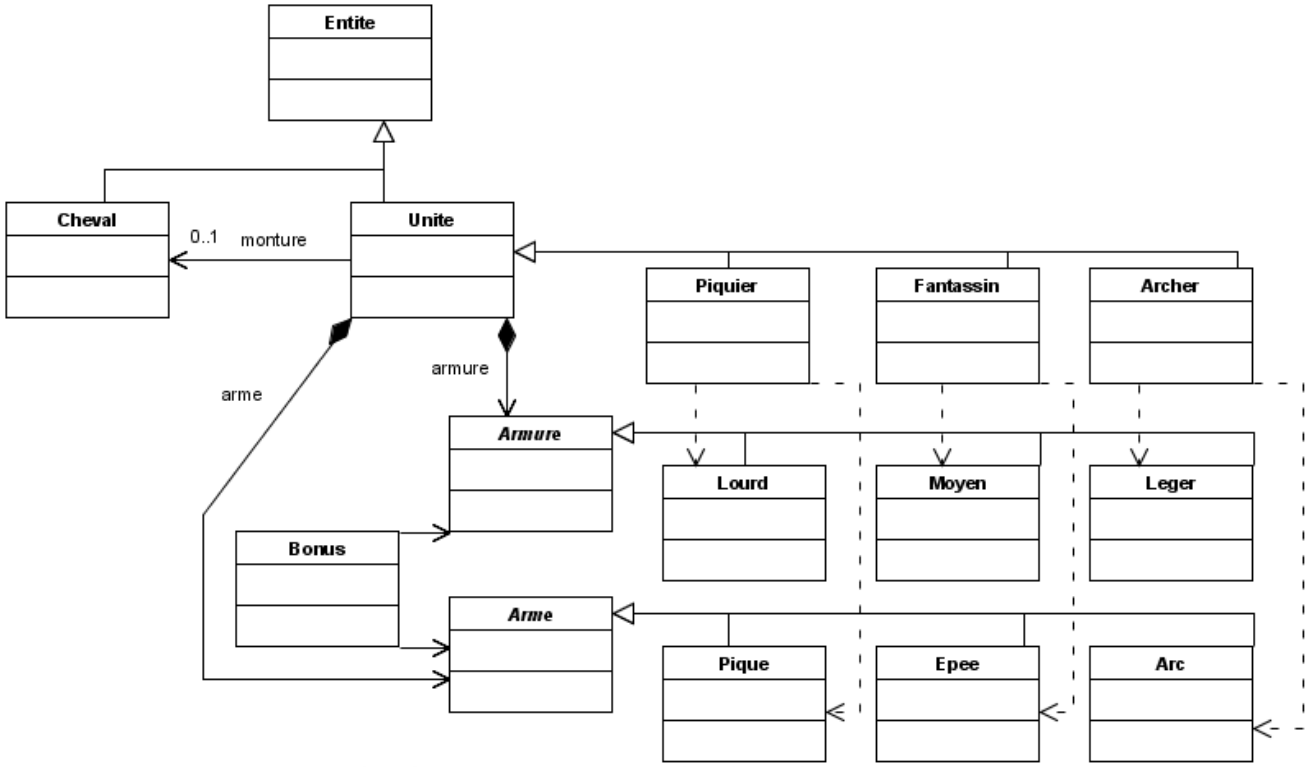
En séparant les différentes parties qui composent ce projet, on assure modularité et évolution. Par exemple il pourrait être intéressant de remplacer la partie moteur décisionnel sans utiliser de logique floue ou encore de de rajouter des règles floues pour enrichir le système.

Le test de l'architecture en java puis son portage en C++ et enfin l'intégration dans le moteur 3D sont des étapes qui assurent un passage souple vers un langage inconnu tout en validant l'architecture au plus tôt.

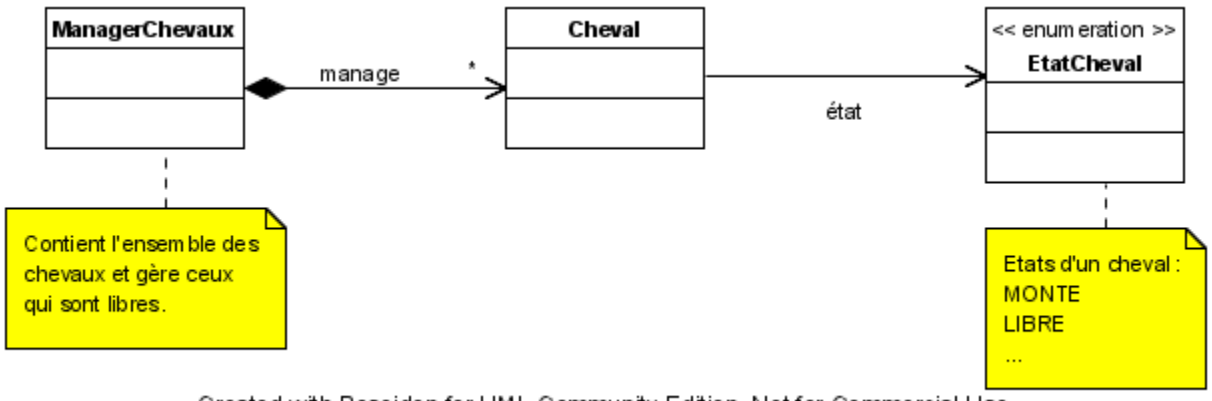
L'ensemble de ces points sont les raisons qui font de ce projet une expérience très enrichissante.

-VII- Annexes

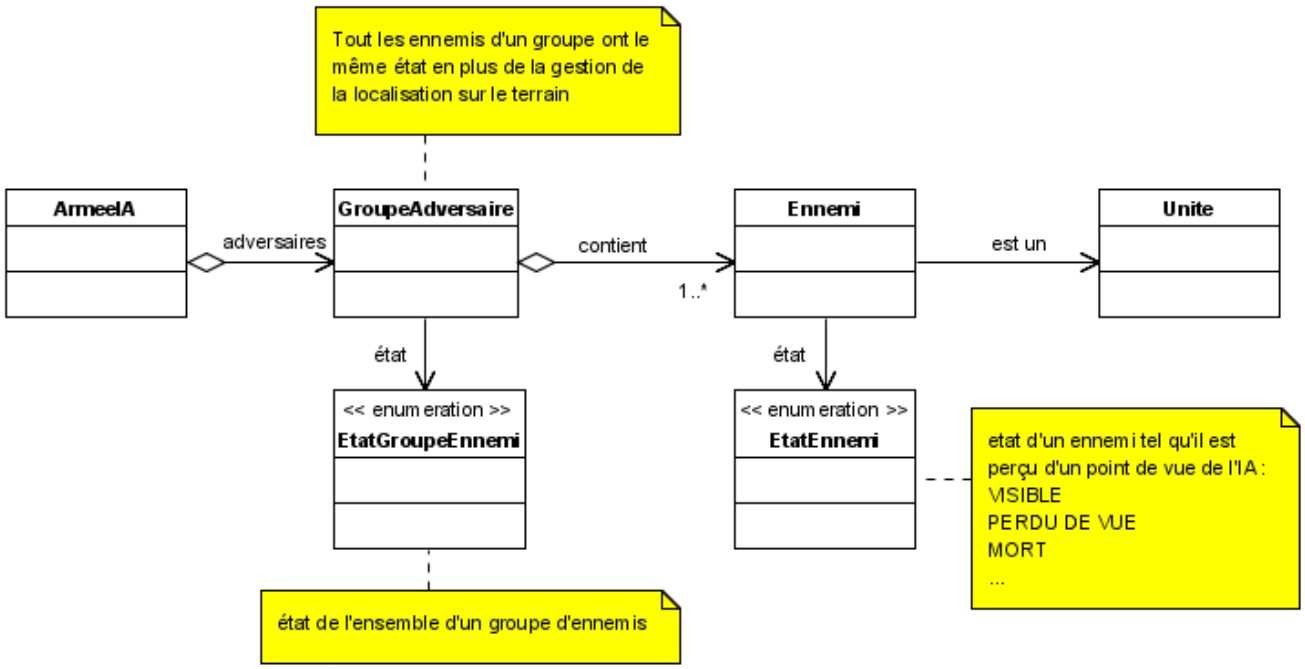
-A- Unités



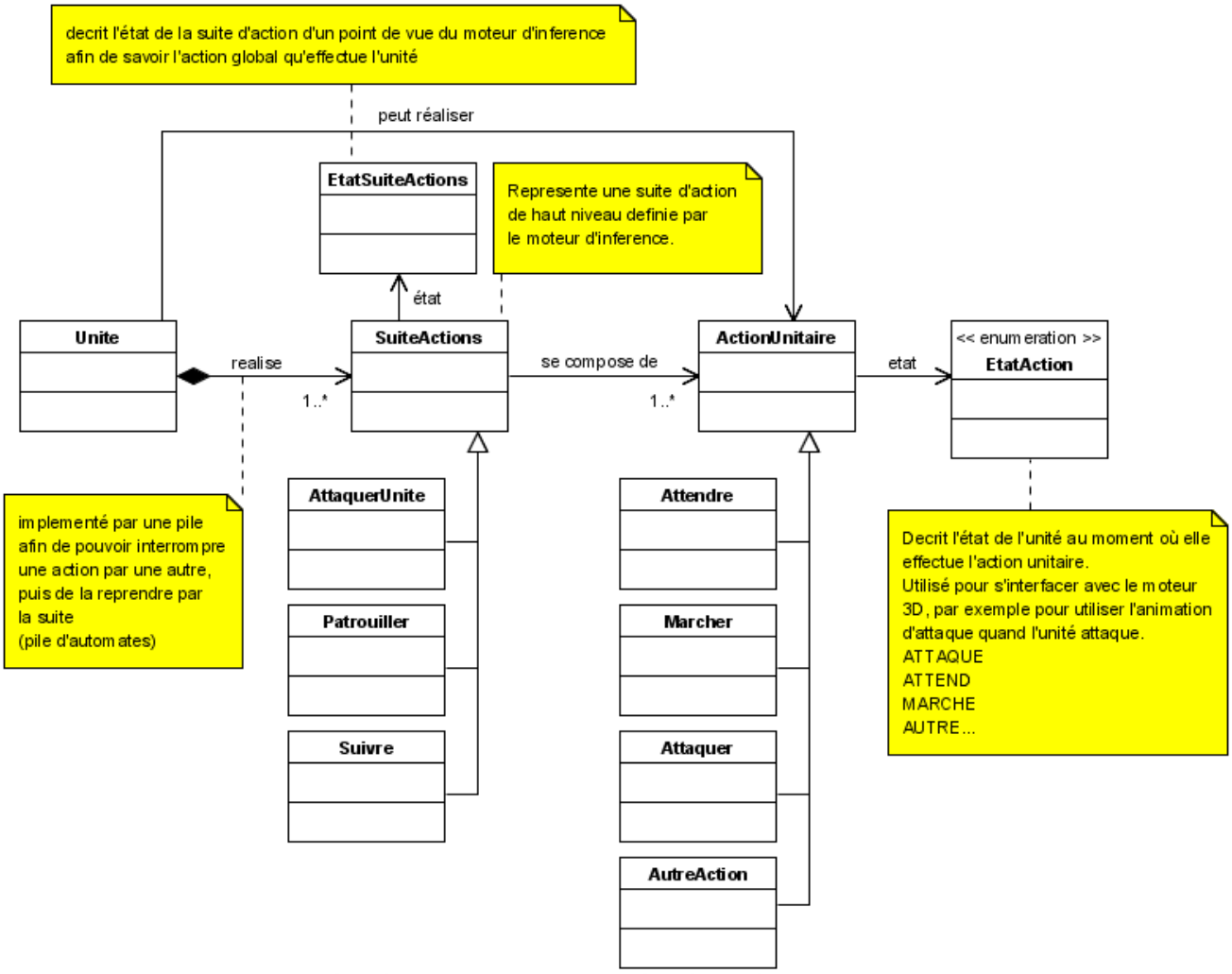
-B- Cheval



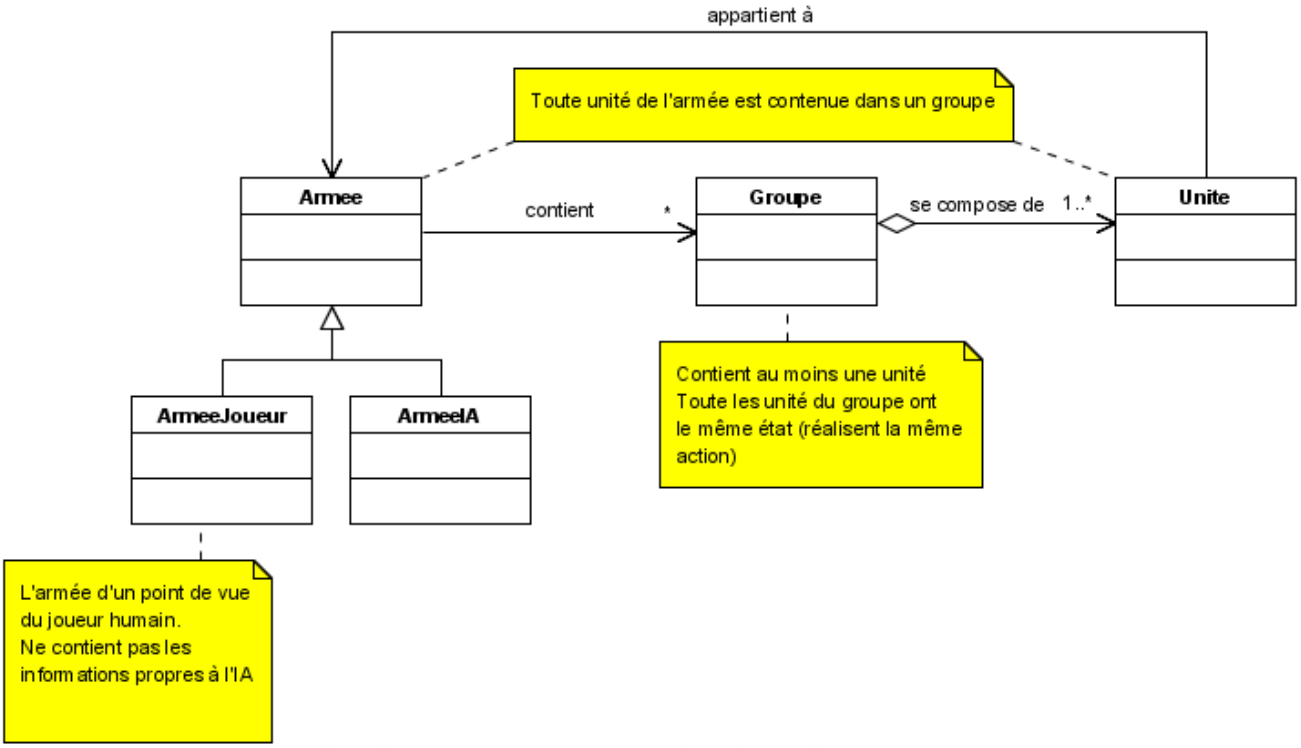
-C- Ennemis


Tout les ennemis d'un groupe ont le même état en plus de la gestion de la localisation sur le terrain
ArmeeIA
adversaires
GroupeAdversaire
contient
1..*
Ennemi
est un
Unite
état
état
<< enumeration >>
EtatGroupeEnnemi
<< enumeration >>
EtatEnnemi
etat d'un ennemi tel qu'il est perçu d'un point de vue de l'IA :
VISIBLE
PERDU DE VUE
MORT
...
état de l'ensemble d'un groupe d'ennemis
Created with Poseidon for UML Community Edition. Not for Commercial Use.

-D- Actions


decrit l'état de la suite d'action d'un point de vue du moteur d'inference afin de savoir l'action global qu'effectue l'unité
peut réaliser
EtatSuiteActions
Represente une suite d'action de haut niveau definie par le moteur d'inference.
état
Unite
realise
1..*
SuiteActions
se compose de
1..*
ActionUnitaire
etat
<< enumeration >>
EtatAction
implementé par une pile afin de pouvoir interrompre une action par une autre, puis de la reprendre par la suite (pile d'automates)
AttaquerUnite
Patrouiller
Suivre
Attendre
Marcher
Attaquer
AutreAction
Decrit l'état de l'unité au moment où elle effectue l'action unitaire.
Utilisé pour s'interfacer avec le moteur 3D, par exemple pour utiliser l'animation d'attaque quand l'unité attaque.
ATTAQUE
ATTEND
MARCHE
AUTRE...
Created with Poseidon for UML Community Edition. Not for Commercial Use.

-E- Gestion de l'Armée



-F- Gestion de l'IA

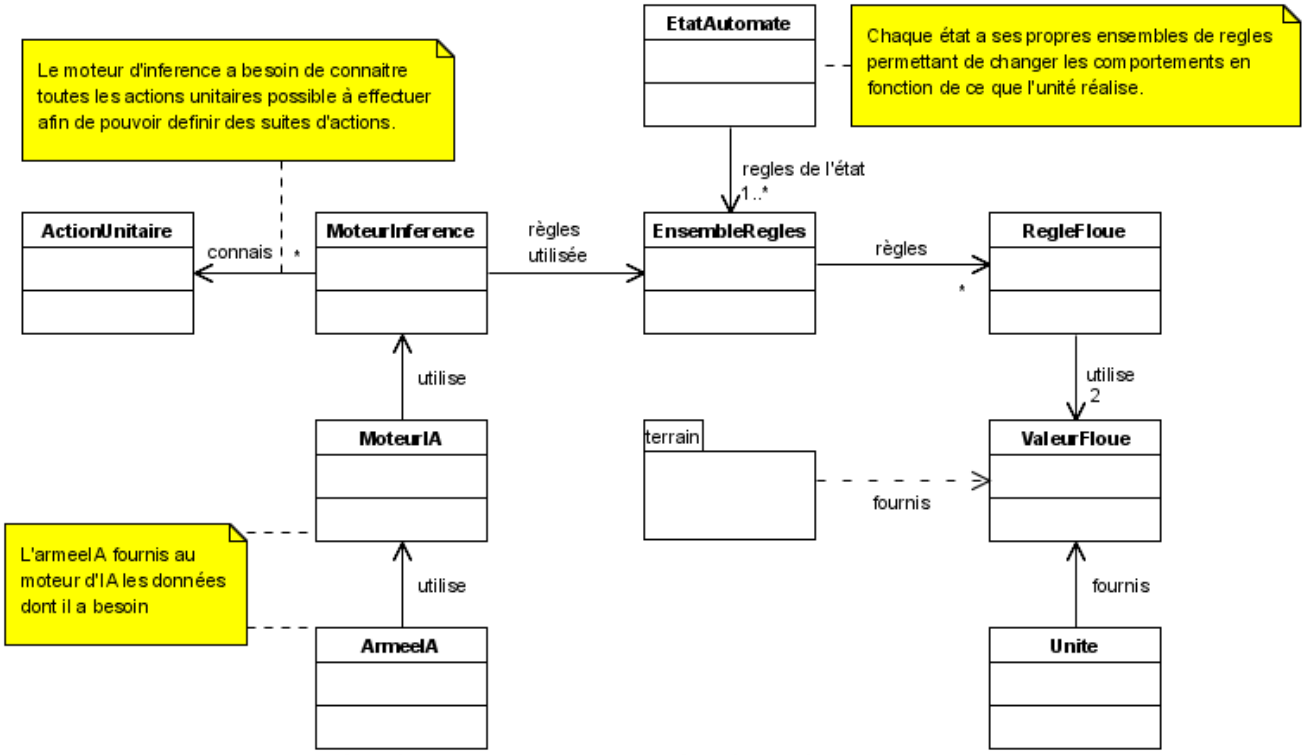


Diagramme d'analyse