

Diplôme de deuxième cycle en développement du jeu vidéo

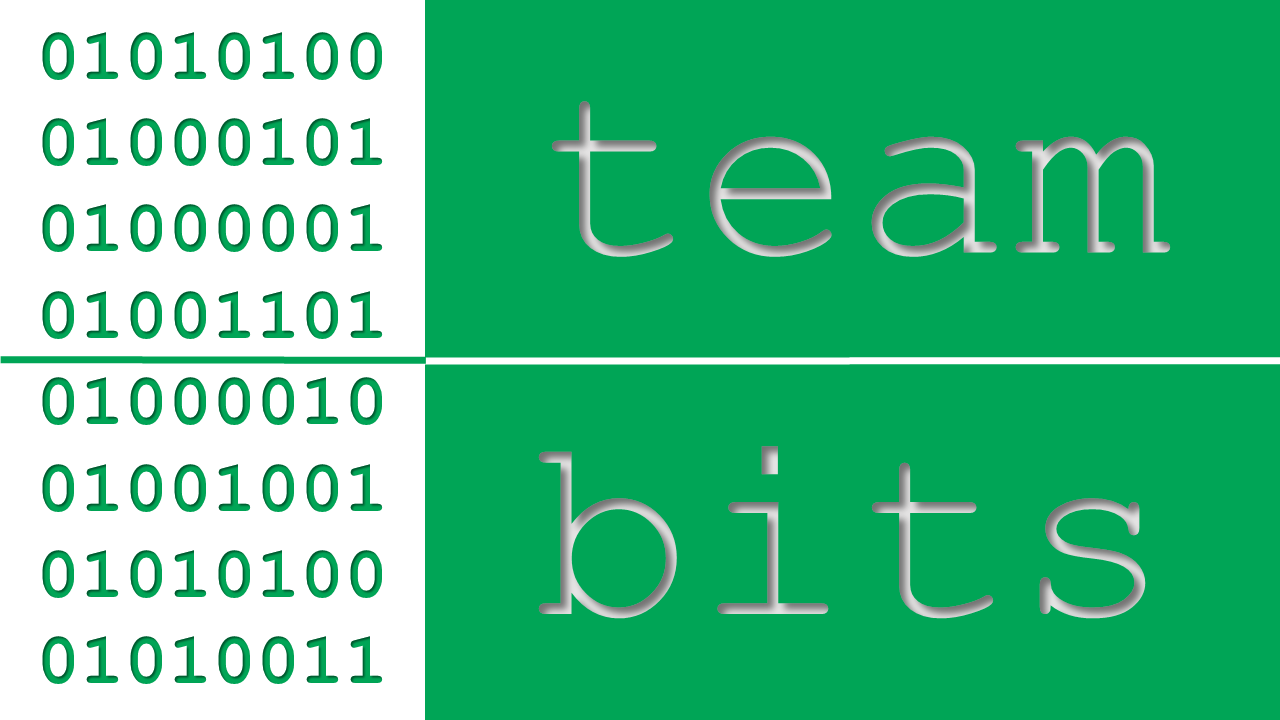
AIBot

Document de conception

Remis à

Carle Coté

Conçu par



-

Éric Maines

Karl-Étienne Perron

Louis Pougis

Vincent Rigaud

Cédric Sentenac

11 novembre 2016

Table des matières

[Modèle 3](#_Toc466542451)

[Graphe & Noeuds 3](#_Toc466542452)

[Obstacles 4](#_Toc466542453)

[Obstacles statiques 4](#_Toc466542454)

[Obstacles dynamiques 5](#_Toc466542455)

[PNJ 5](#_Toc466542456)

[Agents principaux 5](#_Toc466542457)

[Ennemis 5](#_Toc466542458)

[Missions 5](#_Toc466542459)

[Fonctionnalité 6](#_Toc466542460)

[Gestionnaire de missions 6](#_Toc466542461)

[PNJs 7](#_Toc466542462)

[Propriétés non fonctionnelles 7](#_Toc466542463)

[Cartes d’influence 8](#_Toc466542464)

[Outils de débogage 9](#_Toc466542465)

Table de figures

[Figure 1 - Schéma de conception 3](#_Toc466542476)

[Figure 2 - Orientations de tuiles hexagonales supportées 4](#_Toc466542477)

[Figure 3 - Composition d'une mission 6](#_Toc466542478)

Introduction

Ce document a pour but de présenter l’approche prévue pour aborder le projet du cours d’Intelligence Artificielle (INF 781). Il reflète le consensus auquel est parvenu le groupe au sujet de la structure et de l’architecture à utiliser.

Dans un premier temps, nous exposerons le modèle que nous allons employer. Ensuite, une partie concernant les différentes stratégies à mettre en place pour répondre aux fonctionnalités requises fera l’objet d’une description détaillée. Enfin, nous montrerons les outils de débogage qui nous permettront de satisfaire les objectifs fixés par le projet.

À des fins de compréhension, le schéma qui suit illustre la conception globale de notre architecture.

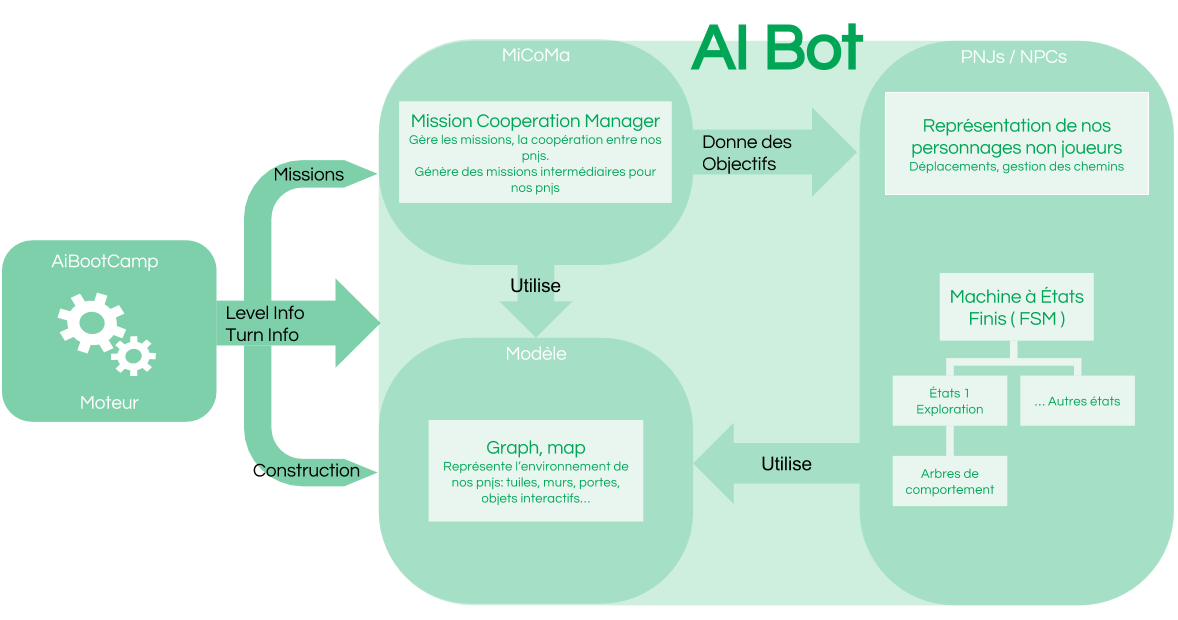


Figure - Schéma de conception

# Modèle

Le monde dans lequel évoluent nos personnages nous est décrit à l’aide de deux structures essentielles : les informations du niveau (très macroscopiques) et les informations du tour (plus précises que celles du niveau). Nous devons donc modéliser les données que le moteur nous fournit, nous avons fait les choix suivants pour modéliser notre environnement.

## Graphe & Noeuds

Dans un premier temps, nous récupérons les informations sur la taille de la carte. À partir de ces dernières, nous construisons un graphe qui représentera les tuiles de la carte du niveau correspondant. Notre modèle peut prendre en charge des tuiles orientées de la manière suivante :

Figure - Orientations de tuiles hexagonales supportées

Chaque tuile de la carte sera modélisée par une structure que nous appellerons “Noeud”. Dans cette dernière, les “noeuds” voisins associés correspondent aux voisins de la tuile en question. Les informations fournies par le moteur seront représentées dans chaque noeud telle que décrit ci-dessous :

* Les obstacles
  + Ceux entourant une tuile (murs, portes, etc.) sont gardés en mémoire dans une structure “enum“ pour différencier leur type. De cette façon, il sera donc facile de rajouter d’autres éléments tels que des interrupteurs ou autres objets interactifs.
* Un attribut
  + Une structure similaire à celle utilisée pour le type des obstacles nous permettra de définir si la tuile est accessible, occupée, interdite ou autre.

## Obstacles

Un obstacle est modélisé à partir d'un “enum“, pour symboliser son type, et une valeur booléenne, indiquant si l'obstacle peut bloquer le PNJ ou non. Jusqu’à maintenant nous avons recensé deux (2) catégories d’obstacles, la première étant statique et l’autre dynamique, À l’inverse de ceux dynamiques, les obstacles statiques ne changeront pas en cours de partie. Ces obstacles peuvent être opaques ou transparents. Pour ce dernier cas, cela signifie qu’il est alors possible de récupérer les informations qui se trouvent derrière cet objet transparent. Il est aussi possible qu’un obstacle soit dissimulé, une action spécifique est alors requise pour pouvoir le révéler.

### Obstacles statiques

Les objets statiques peuvent être des murs ou des vitres. Dans la modélisation de notre carte, un attribut est défini pour chaque tuile et nous permet de savoir si les côtés de cette dernière possède un certain type d’obstacle.

### Obstacles dynamiques

Les obstacles dynamiques peuvent voir leur état modifié au cours de la partie. À titre d’exemple, une porte peut être fermée à un tour - et donc bloquer les PNJs de la même façon qu’un mur - et être ouverte au tour suivant - libérant ainsi le passage pour les PNJs.

Tout comme les obstacles dynamiques, une porte peut être totalement opaque ou transparente, ce qui impliquerait, dans ce dernier cas, que seule la mobilité des PNJs serait bloquée et non leur visibilité.

## PNJ

### Agents principaux

Un PNJ (Personnage Non Joueur) sera composé - en plus de ses informations principales telles que sa position ou ses points de vie - d’un état, connaîtra son objectif ainsi que le chemin pour y parvenir, dont il sera responsable de trouver. En revanche, un gestionnaire de mission lui assignera cet objectif (cf. la section sur le [MiCoMa](#_Gestionnaire_de_missions)).

### Ennemis

Les PNJs ennemis seront représentés de manière différente. Les seuls ennemis connus à ce jour sont les “sentinelles”. En effet, contrairement aux agents principaux, nous n’avons aucun contrôle sur leurs comportements. Ces “sentinelles” restent immobiles sur une grille monde spécifique. Toutefois, ces ennemis sont en mesure d’utiliser un “système de vision” afin de détecter la possible présence d’un agent sur la ou les tuiles visées. Dans ce dernier cas, la partie résulte en un échec. Il est donc nécessaire pour les agents principaux de les contourner et d’éviter les tuiles ciblées par les sentinelles.

Une sentinelle possèdera donc principalement une position ainsi que le modèle de surveillance suivi par la sentinelle. Le modèle de surveillance représente la séquence de direction dans laquelle la sentinelle va regarder. Ce modèle pourra facilement être déduit parmis les différents modèles possibles : … , il nous permettrat de pouvoir prévoir la zone observée par la sentinelle sans avoir de visuel sur la sentinelle.

## Missions

Les missions sont des séquences d’actions que devront effectuer les différents PNJs afin de remporter la partie. Il sera donc nécessaire de tracer leur état d’accomplissement ainsi que de pouvoir distinguer les différents types de missions. Pour ce faire, un gestionnaire de mission, le “MiCoMa” (Mission & Cooperation Manager) sera chargé de récupérer les informations relatives aux missions, de les analyser et enfin d’assigner un objectif spécifique à chaque PNJ afin que chacun coopère pour atteindre le but commun.

# Fonctionnalité

## Gestionnaire de missions

L’évolution des missions sera régie dans le “MiCoMa” par une machine à état finie, où chaque état illustrera le statut de l’objectif des PNJs, et/ou par un arbre de comportement (behaviour tree). Ce dernier cas pourrait nous permettre de gérer plus facilement les séquences d’exécution des différents objectifs à atteindre afin de remplir la mission principale. À noter qu’une mission est un ensemble d’actions, d’objectifs à atteindre nécessaire à la complétion de celle-ci. La mission représente les conditions à remplir pour remporter la partie alors qu’un objectif vise à remplir une condition spécifique, tel que montré dans le schéma ci-dessous :

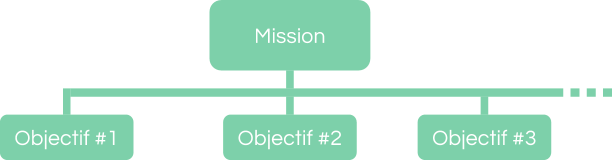


Figure - Composition d'une mission

À l’initialisation d’une partie, le MiCoMa récupèrera et traduira toute l’information concernant les missions qui sera transmise par le moteur. Il s’agira alors à cette étape de déterminer le type de la mission, la séquence d’actions nécessaire pour l’accomplir et initialiser son état. Notre modèle du MiCoMa introduira une notion de priorités afin de simuler un système de coopération entre les différents PNJs. Un PNJ pourra, par exemple, se rendre sur une tuile afin d’ouvrir une porte pour laisser passer les autres. Plus précisément, nous pensons que l’exécution adéquate d’une partie devra être priorisée ainsi :

1. Explorer jusqu’à la découverte des diverses composantes qui permettront la réussite de la mission.
2. Résoudre la séquence concrète et adéquate d’actions de chaque PNJ par rapport aux composantes en question.
3. Exécuter les actions de façon coordonnée.

En cours de partie, les PNJs se verront assigner un objectif précis dont la réussite résultera en l’accomplissement de la mission générale. Nous nous attendons à ce que le MiCoMa puisse offrir aux PNJs des objectifs tels que :

● Trouver/atteindre but final

● Trouver/atteindre une porte

● Trouver/actionner un objet interactif

● etc.

À chaque tour, l’état de l’objectif pour chaque PNJ sera analysé afin d’évaluer la validité de la mission en cours. Dans le cas où l’objectif n’est plus valide, par exemple lorsque l’on constate que l’objectif ne pourra pas être atteint, le MiCoMa pourra offrir un nouvel objectif au PNJ concerné. Au contraire, si l’objectif assigné à un PNJ est complété, on passe à l’objectif suivant - s’il y en a - jusqu’à ce qu’il n’y en ait plus, signifiant alors que la mission principale a été remplie et donc la partie remportée.

## PNJs

Dans un premier temps, chaque PNJ se voit assigner un objectif par le MiCoMa tel que mentionné dans la section précédente. En fonction de cet objectif, chaque PNJ évaluera son état, et ajustera alors son comportement selon l’état dans lequel il se trouvera à l’aide d’un arbre de comportement. De plus, nous avons décidé de rendre le PNJ entièrement responsable de ses actions, impliquant la recherche et gestion de conflits de chemin avec les autres PNJs.

Actuellement, les deux états principaux sont l’exploration et le suivi de chemin. L’exploration correspondra à la recherche de composantes utiles à son objectif, à défaut d’avoir l’information nécessaire à l’accomplissement de ce dernier. Dans le cas contraire, s’il possède cette information, il lui suffira de trouver un chemin vers cet objectif et de le suivre. Il est important cependant de noter qu’en tout temps, tous les PNJs devront prendre en compte les sentinelles à éviter - s’il y en a.

L’étape d’exploration sera réalisée à l’aide de cartes d’influence - nous reviendrons sur ce point plus en détails dans la section [Cartes d’influence](#_Cartes_d’influence). L’exploration sera divisée en plusieurs catégories, qui nous permettront d’être plus efficace dans celle-ci. En effet, la méthodologie d’exploration pour trouver un objet caché ou trouver une plaque de pression ouvrant une porte ne sera pas la même.

Concernant le suivi de chemin, nous utiliserons un algorithme de recherche de chemin classique tel que celui d’A\*. Cette partie regroupera l’essentiel de gestion des conflits de chemin entre PNJs.

# Propriétés non fonctionnelles

● Nombre de PNJs croissants / temps calcul réduit

Au cours de la session et de l’évolution de la nature des livrables, le nombre de PNJs va augmenter. En parallèle, le temps alloué à l’exécution des calculs nécessaires va diminuer. Pour répondre à cela, nous avons décidé de prendre les mesures suivantes :

* Dans la mesure où le nombre de PNJs augmente, les calculs de chemins de plusieurs PNJs se feront en parallèle. Nous prévoyons d’implémenter un principe de multi-thread. Un système de gestion de synchronisation et d’échanges de données (si besoin) sera utilisé pour mener à bien cette implémentation.
* Une conservation de données sera mise en place à travers un mécanisme de mémoïsation. Nous projetons de conserver les valeurs retournées par certaines fonctions (par leurs appels précédents), de les stocker dans une structure de données adaptée et, lorsqu’elles seront de nouveau appelées (avec les mêmes paramètres), les valeurs stockées correspondantes seront retournées plutôt que d’être calculées une nouvelle fois.
* Nous allons aussi, le plus souvent possible, gérer directement les allocations et libérations mémoire, de contrôles d’adresses libérées,… comme vu lors du laboratoire sur l’Arena dans le cours de Patrice Roy.
* Dans un but d’optimisation de cache, le choix et la gestion des structures de données employées seront contrôlés manuellement.

## Cartes d’influence

Lors de l’exploration par nos PNJs, nous allons essentiellement utiliser des cartes d’influence. Ces cartes pourront dépendre du type de mission actuellement confié aux PNJs. Nous pourrions ainsi contrôler avec plus de finesse leur exploration. Par exemple, si un PNJ est en exploration sans but spécifique les cases avec des vitres seront des points d’influence, tandis qu’un PNJ à la recherche d’une porte cachée favorisera, si possible, les murs proches d’une tuile de type “target”.

L’utilisation des cartes d’influence permettra d'alléger les calculs lors de l’exploration de nos PNJs car il nous suffira de prendre la tuile proche avec la plus grande influence. Pour des soucis de performance, chaque type de carte d’influence ne sera calculé qu’une seule fois durant un tour et sera accessible par tous nos PNJs. Les cartes d'influence étant utilisées uniquement (ou presque, nous reviendrons sur ce point juste après) pour l’exploration, elles ne seront pas calculées si nous n’en avons pas besoin. Pour l’instant nous utiliserons un algorithme de diffusion par remplissage pour le calcul des influences. Cependant, si besoin il y a, nous explorerons un algorithme plus performant pour réaliser ce calcul. Il est également possible que dans certains cas, une combinaison entre un comportement de foule (flocking behavior) et une carte d’influence soit utilisée (pour les mêmes fins). Nous pourrions par exemple définir le dirigeant comme étant le seul centre d’influence pour les suivants.

## Outils de débogage

Une partie non négligeable de notre temps sera investie dans la réalisation d’outils pour déboguer notre IA. Nous développerons tout d’abord un outil rudimentaire composé d’une simple journalisation dans des fichiers texte. À ce moment-ci, cet outil est déjà suffisamment avancé pour que nous puissions nous en servir. Il nous permet d’afficher toutes les informations telles que l’identifiant de la tuile sur laquelle se trouve un PNJ, le numéro du tour, l’état de connaissance de la carte à un tour précis, etc. Nous possédons ce type de débogage pour les entités de PNJs, le graphe représentant la carte ainsi que la carte d’influence. Il nous faut encore réaliser ces outils pour les nouvelles entités, issues de la fusion de nos différents codes.

À ce jour, nous possédons un script permettant d’exécuter la totalité des niveaux de façon automatisée. Le script .bat appelle une version modifiée de PlayStandaloneMatch.bat sans pause d’exécution. Les résultats sont stockés dans des répertoires individuels, basés sur le nom du tableau. Les fichiers individuels permettent d’ouvrir une page de navigateur, par carte, de façon automatisée. Il est sujet à amélioration.

À plus long terme, il serait intéressant d’avoir un outil plus graphique pour déboguer, afin d’afficher un maximum d’information, mais filtrée dynamiquement de façon graphique (exemple par un clic sur un NPC individuel, pour voir son état de comportement). Cet outil sera plus dans l’esprit de la visualisation proposé dans le moteur mais en proposant de bien plus amples informations.

# Estimation de l’échéancier

Dans cette partie nous estimerons le temps que nous pensons attribuer au développement du projet. Nous séparerons cet échéancier en deux (2) parties : une partie concernant le temps alloué au fonctionnalités et une au temps octroyé par semaine au projet.

## Fonctionnalités

Mise en place d’une nouvelle structure fonctionnelle : 15h

Portes (pleine vs vitrée) : 8h

Objets Interactifs : ?h

Missions / Coopération : 15h

Sentinelles : 10h

Portes cachés : 5h

Gestion nombre NPC vs CPU : 15h

Outils de débogage

Texte : 3h

Graphique : - (Nice to have)

Behaviour Tree : 15h