UNIVERSITÉ DE SHERBROOKE

Diplôme de deuxième cycle en développement du jeu vidéo

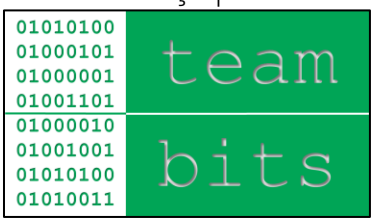
AIBot

Rapport de séminaire #2

Remis à

Carle Coté

Conçu par



-

Éric Maines

Karl-Étienne Perron

Louis Pougis

Vincent Rigaud

Cédric Sentenac

12 décembre 2016

Table des matières

Table des figures

Introduction

Ce document a pour but de présenter l'approche avec laquelle nous avons abordé le projet du cours d'intelligence artificielle (INF781). Il reflète la démarche appliquée par le groupe.

Ce projet est basé sur la conception et le développement d'un système de comportements inspiré du modèle adaptatif. Le défi lancé fut de développer des algorithmes permettant à des agents de compléter les niveaux d'un jeu tour par tour sans l'intervention d'un être humain pendant le déroulement de la partie. Ces agents ont la possibilité de se déplacer et d'interagir avec certains éléments de leur environnement. Ils peuvent cependant avoir un champ de vision limité, ce qui reflète très bien le modèle adaptatif. Le réel but dans ce projet fut alors d'analyser et d'appliquer diverses stratégies permettant à un agent intelligent d'évoluer de manière adéquate lorsque placé dans certains types d'environnement. Contrairement au milieu du jeu vidéo où l'on cause volontairement des imperfections chez l'agent afin d'offrir aux joueurs une expérience de jeu intéressante, nous visons ici la perfection d'exécution. Ce projet nous permettait cependant d'analyser et de comprendre les différents enjeux présents dans le monde de l'intelligence artificielle que ce soit au niveau du développement concret du comportement des agents que le respect des contraintes telles que le temps de calcul alloué.

Notre équipe s'est donc inspirée du paradigme « *Sense-Think-Act* » afin de réaliser une intelligence artificielle faible et adaptative. Cette dernière a été mise en place tout au long de la session de façon incrémentale, nous poussant ainsi à revoir continuellement notre conception afin de l'adapter pour qu'elle soit apte à intégrer les nouvelles fonctionnalités exigées. Le système développé en fin de compte permet ainsi de récupérer et d'analyser les nouvelles informations de l'environnement reçues à chaque tour. Les résultats de cette analyse serviront alors à déterminer les actions qui seront entreprises par les agents disponibles en leur assignant des objectifs précis tout en respectant le temps de calcul maximal imposé entre deux tours. Pour cela, plusieurs techniques utilisées en intelligence artificielle dans le domaine du jeu vidéo furent développées. Parmi celles-ci, on compte notamment un arbre de comportement (« *behaviour tree* »), la création d'une carte d'influence, une machine à états finis hiérarchique, bien entendu une technique de recherche de chemin et plus encore. Toutes ces techniques nous ont ainsi permis de définir des comportements intéressants chez nos agents pour faire face à diverses situations telles que, par exemple, explorer différentes zones perçues comme étant distinctes dans l'environnement afin d'obtenir de l'information, activer des dispositifs pour ouvrir des portes ou encore gérer des conflits de chemin avec d'autres agents.

Il s'agira, dans la première partie de ce document, de présenter en détail les différents aspects du projet ainsi que l'architecture mise en place pour répondre au problème posé. Tout ce qui concerne les règles et possibilités du jeu ainsi que la description des niveaux, de même qu'une explication détaillée et la justification de l'architecture mise en place seront présentées dans cette section. Suivront ensuite les résultats qui ont été obtenus et, enfin, une analyse de ceux-ci. Cette partie sera consacrée à critiquer la conception initiale mise en place et la réalisation concrète finalement produite.

Des senseurs vers le modèle (*Sense*)

La première étape à réaliser pour mettre en place l'architecture « *Sense-Think-Act* » est sans surprise l'étape du « *Sense* », francisée ici par l'étape senseur. Cette partie du document vise à décrire le modèle de traduction de l'environnement dans lequel évoluent nos Personnages Non Joueurs (PNJs).

La représentation du monde dans lequel évoluent nos personnages est primordiale dans notre modèle puisque toute la partie d'analyse et de prise de décision se basera sur l'état courant du monde et donc sur cette représentation.

De plus, il est à noter que toute l'information concernant l'environnement est récupérée par le moteur « *AI Boot Camp* ». Nous n'avons donc aucun contrôle sur l'accessibilité des données et leur format. Le monde nous est alors décrit à l'aide de deux structures essentielles : les informations du niveau - très macroscopiques telles que le mode de vision des agents (omniscient ou à portée limitée), la taille de la carte du niveau, etc. - et les informations du tour - plus précises, notamment les tuiles visibles, les objets présents ou encore le changement d'état d'une porte. Nous avons donc dû modéliser les données que le moteur nous fournissait. Les structures utilisées pour représenter l'environnement dans lequel s'aventurent les agents sont décrites dans cette section.

La carte

Dans un premier temps, nous récupérons toutes les informations liées au niveau correspondant. Parmi les plus importantes se trouvent celles concernant la carte, elle nous donne la majorité des caractéristiques du niveau chargé. Nous nous servons ainsi de ces caractéristiques pour construire un graphe qui représentera les tuiles de la carte utilisée. Notre modèle pourrait prendre en charge des tuiles orientées des deux (2) façons suivantes :

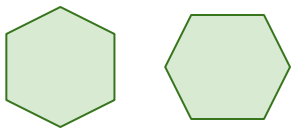


Figure 1 - Orientations de tuiles hexagonales supportées

Cependant, au cours de la session, l'équipe de conception de jeux ("*game design*") nous ayant fait part de leur décision de ne pas conserver la possibilité d'un changement d'orientation des tuiles, cette spécificité a été retirée du modèle. La seule orientation de tuile considérée est maintenant le modèle se situant à la gauche dans la figure ci-dessus.

Chaque tuile de la carte est modélisée par une structure que nous appellerons un « *nœud* ». Il est habituel de stocker pour un graphe les arêtes qui représentent les jonctions entre les différents nœuds du graphe, dans notre cas toutes les arêtes possèdent le même coût (un tour de déplacement pour passer d'une tuile à une autre), par conséquent nous avons choisi de directement conserver les voisins d'un nœud dans la structure le représentant. Grâce aux informations qui nous sont fournies par le moteur nous pouvons extraire certaines données pertinentes pour notre représentation et les stockées dans la structure nœud, voici les principales informations qui composent cette structure :

- Les obstacles

Ceux entourant une tuile (murs, portes, etc.) sont gardés en mémoire dans une structure spécifique aux obstacles, un nœud possède alors huit de ces structures, une pour chaque direction cardinale. De cette façon, il est facile de rajouter d'autres éléments tels que des interrupteurs ou autres objets interactifs. Nous reviendrons sur les obstacles dans une partie qui leur sera dédiée.

- Un attribut descriptif

Une énumération (`enum`) nous permet de définir si la tuile est accessible, si un dispositif est présent sur cette tuile, si elle est occupée, interdite ou autre. Cet attribut nous est particulièrement utile dans l'analyse de prise de décision de nos PNJs lorsque l'on génèrera une carte d'influence ou lors de la recherche de chemin. Ces deux (2) aspects seront décrits plus loin dans le document.

- Une zone d'appartenance

Cet attribut nous fournit une information supplémentaire dans le cas où deux (2) espaces sont apparemment séparés. Le partitionnement du monde en différentes zones a été estimé très pertinent depuis l'apparition des portes. Une section, décrite plus bas, dans ce document est réservée à la description plus détaillée de cet attribut et de la manière dont il est tenu à jour.

Compte tenu du fait que les agents ont seulement un accès partiel aux informations de l'environnement dans lequel ils évoluent, il est primordial d'actualiser notre représentation de l'environnement à chaque tour afin que l'analyse qui s'en suivra permette la prise de décision la plus adéquate possible. En ce sens, nous mettons à jour le modèle de notre carte à chaque tour de la manière suivante.

En premier lieu, nous actualisons les obstacles portés par les arêtes. Cette mise à jour vérifie si nos agents ont découvert de nouveaux murs, fenêtres ou portes (vitrées ou non). Ce processus est effectué avant le reste afin d'éviter l'ajout d'une tuile cible inaccessible à un agent. Ensuite, nous ajustons l'état de nos tuiles. Au cours de cette étape, nous ajoutons alors l'information des tuiles découvertes depuis le tour précédent en spécifiant notamment leur attribut. En dernier lieu, nous tenons à jour notre carte d'influence ainsi que nos zones. Durant cette dernière partie de mise à jour de la carte, nous appliquons notre algorithme de diffusion de zone.

Les cartes d'influence

Nos personnages évoluant dans un environnement qu'ils ne connaissent pas entièrement, l'exploration de celui-ci est quelque chose de primordial pour que nos personnages puissent découvrir de nouvelles informations sur le monde. Cette nécessité d'explorer nous à motiver à trouver un moyen simple et efficace pour réaliser cela ; nous avons alors choisi d'implémenter une carte d'influence. Nos cartes d'influence sont des représentations du monde uniquement destinées à l'exploration et permettent d'associer à chacune de nos tuiles une valeur d'influence. Plus cette valeur d'influence sera élevée plus nos personnages seront "motivés" à se diriger sur cette tuile. Grâce à ce mécanisme, nous pouvons facilement, rapidement et efficacement diriger l'exploration de nos personnages vers des points d'intérêt.

Pour implémenter ce principe, nous avons choisi d'utiliser la méthode suivante : nous déterminons tout d'abord quelles sont les tuiles présentant de l'intérêt dans la quête de nos personnages afin d'obtenir des informations sur leur environnement. Ensuite, l'influence de ces centres d'intérêt aux tuiles voisines est diffusée pour au final obtenir, pour chaque tuile pertinente de la carte, une valeur d'influence.

Nous pensions réaliser deux types de cartes d'influence différentes, l'une pour l'exploration dans le but d'obtenir de la simple information et l'autre, plus spécialisée, pour chercher exclusivement des portes cachées. Malheureusement, le temps nous ayant manqué, nous avons dû choisir de couper certaines fonctionnalités et la gestion des portes cachées en fait partie. Nous allons donc maintenant voir plus en détail comment a été mis en place le calcul de la carte d'influence pour nos agents.

Tout d'abord, nous avons dû définir des règles pour déterminer quels sont les points d'intérêt. Nous avons choisi des règles plutôt simples, mais qui nous ont paru être suffisantes et efficaces sur les cartes d'entraînement.

La toute première est qu'une tuile déjà visitée par l'un de nos agents n'est pas digne d'être un centre d'intérêt, car nous connaissons déjà tout ce que l'on peut savoir à partir de cette tuile. Le fait de ne rien apprendre de nouveau a pour conséquence qu'il est alors inintéressant de se rendre sur cette tuile.

La seconde règle est la suivante : les fenêtres sont des centres d'intérêt non négligeables. En effet, elles offrent aux PNJs une source d'informations qui peut être cruciale pour la suite du déroulement du niveau. En connaissance de l'importance de la présence de ces fenêtres, la valeur d'influence des tuiles qui les hébergent est incrémentée.

Pour parvenir à une construction de cartes d'influence satisfaisant notre modèle et représentant fidèlement l'environnement, nous procédons en deux (2) étapes. Dans un premier temps, nous cherchons à identifier les tuiles susceptibles d'être intéressantes. Une fois ce discernement effectué, nous les stockons pour pouvoir nous en servir par la suite. Dans un second temps, nous travaillons sur les tuiles récupérées à l'étape précédente puis, pour chacune d'elle, nous propageons l'influence qui lui est associée dans un certain rayon. Cette propagation se fait alors pour chaque voisin remplissant certaines conditions, de la tuile considérée. Pour pouvoir accueillir et propager de l'influence, une tuile doit remplir les conditions suivantes : elle doit être connue, et doit être accessible à partir de la tuile considérée. La valeur de l'influence propagée diminue proportionnellement à la distance de la tuile qui est diffusée. L'influence diminue en fonction de la distance à la tuile d'intérêt originelle, mais aussi par rapport à une distance maximale de diffusion. Dans une optique de performance nous avons décidé de ne pas diffuser l'influence d'une tuile sur une trop grande distance, voire sur une distance "infinie", nous avons alors choisi de nous limiter à une distance égale à la distance de vision plus une case. Cette distance de diffusion nous permet de presque toujours avoir une valeur d'influence sur les cases proches d'un agent (du moins pour les cas où il serait seul), et d'avoir finalement une marge de sécurité, sans pour autant avoir une distance de diffusion trop importante par rapport à nos besoins. Pour une meilleure compréhension, prenons le schéma suivant :

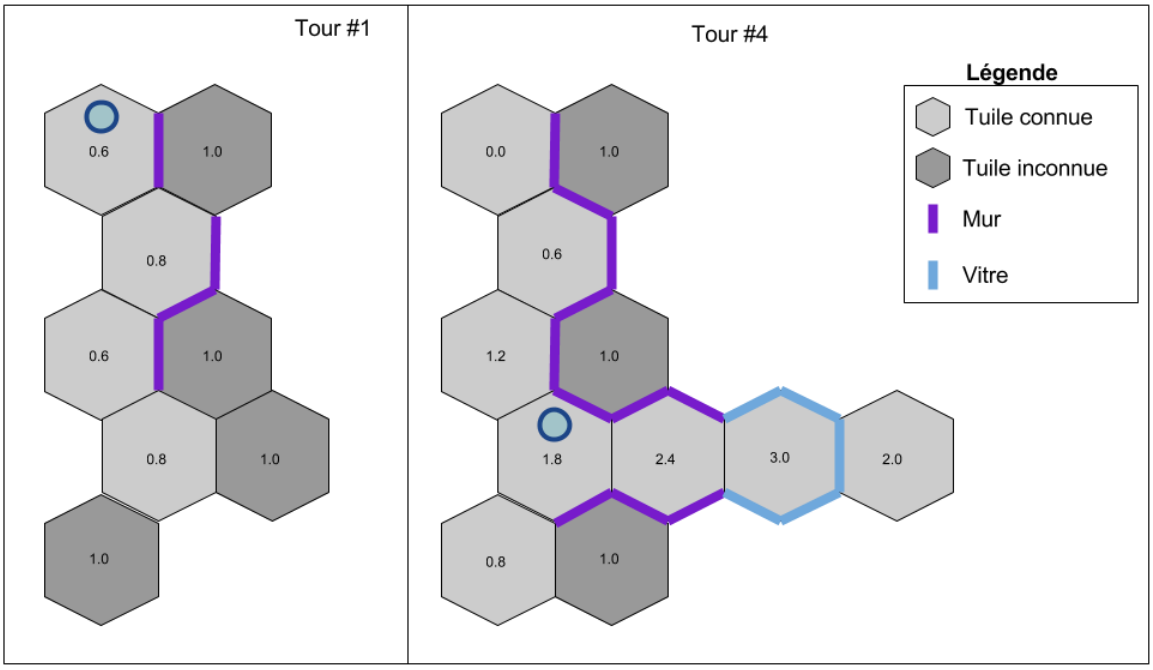


Figure 2 - Fonctionnement de la carte d'influence

Dans le schéma d'illustration ci-dessus, deux (2) cas sont visibles. En premier lieu, au premier tour, nous pouvons constater que les tuiles ayant une valeur d'influence de `1.0` sont celles qui se trouvent de l'autre côté d'un mur et donc non visible. Ces valeurs traduisent la première règle énumérée précédemment qui dicte qu'une tuile déjà connue ne représente aucun intérêt pour l'agent et donc, indirectement, que les tuiles non connues sont jugées plus intéressantes. Au quatrième tour, on constate tout d'abord que les tuiles qui ne sont toujours pas connues depuis le tour #1 ont encore une influence de `1.0`. Ensuite, la tuile jugée la plus intéressante pour notre agent devient celle entourée de fenêtres. En effet, l'information qui se trouve derrière ces trois (3) obstacles bloquants reste tout de même accessible et pourrait nous être très bénéfique pour la suite de la partie, d'où l'influence de `3.0`.

Les obstacles

Un obstacle est modélisé à partir d'une structure particulière pour symboliser son type, s'il est bloquant (dans un sens comme dans l'autre), ou encore s'il est temporairement ouvert (comme pour une porte par exemple). 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 changent pas en cours de partie. Tout obstacle peut être opaque ou transparent. 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'une porte soit dissimulée, une action spécifique est alors requise pour pouvoir la 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 possèdent 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 agents de la même façon qu'un mur - et être ouverte au tour suivant - libérant ainsi le passage. Dans le modèle actuel, seules les portes font partie de ce type d'obstacle.

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

Un obstacle de cette catégorie possède une particularité supplémentaire. En effet, il peut être activé grâce un dispositif, que ce dernier soit un interrupteur ou une tuile à pression. Il possède alors un attribut qui permet de déterminer à quel(s) dispositif(s) il est lié. Dans le cas où un obstacle dynamique serait présent dans le niveau chargé, un seul et unique dispositif doit être activé pour débloquer l'obstacle.

À l'origine, un obstacle dynamique devait posséder une politique d'activation qui spécifiait de quelle manière il devait être débloqué. À titre d'exemple, une porte liée à des plaques de pression pouvait être ouverte de l'une des façons suivantes :

- L'activation d'un seul dispositif parmi un ensemble spécifié ;
- L'activation simultanée d'un ensemble donné de dispositifs ; ou bien
- L'activation d'un seul et unique dispositif.

Il n'existe cependant dorénavant que deux politiques d'ouverture d'un obstacle dynamique : s'il est lié à un dispositif tel qu'une plaque de pression, l'activation d'un seul et unique dispositif est nécessaire pour ouvrir le passage ; en revanche si aucun dispositif n'est relié à cet obstacle, une simple interaction directe avec celui-ci suffit pour le déverrouiller. Cette différence a bien entendu changé notre conception puisque nous n'avions plus à tenir compte du dernier cas de la liste des politiques énoncée précédemment.

Les dispositifs

Les dispositifs sont des éléments permettant de débloquer des obstacles dynamiques tels que les portes. Un dispositif ne peut être lié qu'à un seul obstacle, même si un obstacle peut lui être activé par plusieurs dispositifs comme il a été expliqué dans la section

précédente. Un dispositif contient alors un identifiant unique, l'identifiant de la tuile sur laquelle il se situe et un autre indiquant à quel obstacle il est lié.

Les zones

L'ajout de cette structure fut motivé par deux principaux faits. Tout d'abord, cela nous permet de donner des objectifs à nos PNJs. Par exemple, dans la figure ci-dessous, en présumant que seules les zones #1 et #3 sont connues, nous pouvons donner un objectif tel que : « explorer la zone #2 ».

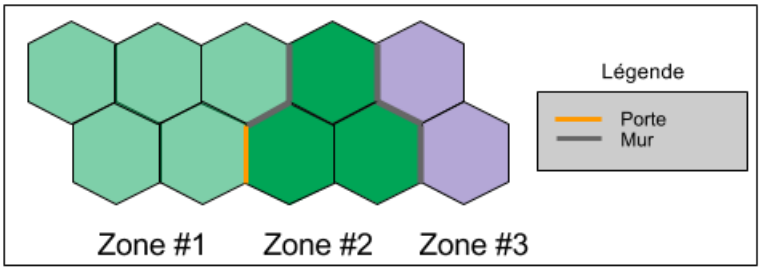


Figure 3 - Partionnement de la carte

La seconde motivation est plus de l'ordre de la modélisation même. En effet, avec l'ajout des portes, cette information nous permet d'identifier plus facilement les passages communiquant entre deux zones - par une porte, entre autres.

Par définition, une zone est un espace dans lequel il est possible d'accéder à n'importe quelle tuile incluse dans celui-ci à partir de n'importe quelle autre tuile de ce même espace. Ceci implique alors que tout objet, obstacle ou tout autre élément de l'environnement n'est pas directement accessible si ce dernier n'est pas compris dans la zone dans laquelle se situe un agent.

Pour pouvoir définir et représenter les diverses zones qui composent l'environnement, le principe d'algorithme de diffusion a été développé. L'idée ici est donc de partir d'une tuile de départ sur la carte puis de propager l'identifiant de la zone à laquelle elle appartient. Pour appliquer notre algorithme de diffusion de zones, nous prenons comme tuile de départ celle sur laquelle se situe chaque agent, puis l'identifiant de sa zone est propagée par récursion jusqu'à ce que l'espace dans lequel l'agent se situe ou bien que la limite de vision soit atteinte. Partant alors de la tuile de l'agent, la liste de ses voisins est parcourue et pour chacun d'entre eux les vérifications suivantes sont effectuées. Lorsqu'une tuile est évaluée pour la première fois, si un obstacle entre ces deux tuiles est présent, on suppose alors temporairement que la tuile voisine évaluée fait partie d'une autre zone et elle est ajoutée à une liste qui permet de représenter les tuiles ouvertes, donc déjà évaluée et faisant partie d'une autre zone, on lui assigne donc un identifiant supérieur au plus grand identifiant de zone connue. Si la tuile pour laquelle on souhaite définir un identifiant de zone est interdite, le mécanisme de choix

de l'identifiant de zone est semblable à celui décrit juste avant. À chaque tour cette diffusion des zones est mise à jour afin de nous assurer d'avoir en tout temps une connaissance des zones la plus complète possible.

Dans le cas contraire, cette tuile fait alors partie de la même zone et le processus de propagation est réitéré par récursivité sur la nouvelle tuile évaluée. Cette opération est toutefois aussi effectuée lorsque l'identifiant d'une tuile nécessite d'être de nouveau évaluée et donc de propager sa nouvelle valeur. Ceci arrive typiquement lorsqu'une tuile est supposée appartenir à une autre zone, car elle se situe derrière un mur, mais est en réalité dans le même espace. Une fois la zone courante complètement diffusée, on récupère dans la liste de tuiles ouvertes une tuile appartenant à une autre zone et on la diffuse de la même. Ce processus est répété jusqu'à ce que cette liste ouverte soit vide.

Voici comment nous définissons et mettons à jour nos différentes zones. Un autre élément très important sur les zones est de savoir si une zone est "fermée" ou si elle ne l'est pas donc si elle est "ouverte". Une zone fermée est une zone que l'on connaît en totalité, c'est-à-dire que nous avons découvert toutes les informations à son propos et qu'il ne reste aucune case pouvant appartenir à cette zone qui nous est inconnue. Cette information est importante, car c'est la première motivation pour chercher à faire changer de zone nos agents. L'algorithme de diffusion décrit au-dessus étant très gourmand en temps de calcul *UCT (*unité centrale de traitement – *CPU* en anglais*)* nous avons trouvé impensable de refaire un algorithme similaire pour déterminer si une zone est fermée, par conséquent nous avons décidé de faire le nécessaire pour savoir si une zone est fermée pendant la diffusion de celle-ci. Cela nous a finalement permis de rajouter facilement sans une grande perte de performance une information importante[1].

Outre cette représentation de notre environnement, nous utilisons en plus une modélisation d'un niveau sous la forme de cartes d'influence.

Les règles du jeu

Notre but est d'être en mesure de valider différentes cartes générées, normalement, de manière procédurale. Pour cela, nos personnages doivent suivre quelques règles. Dans notre modèle, les règles que doivent suivre nos agents sont relativement simples. Nous pouvons en distinguer trois. La toute première règle est d'amener nos agents sur les cases cibles pour terminer la partie. Une autre consiste à éviter de marcher sur les tuiles interdites. La dernière est une contrainte spécifique à chaque niveau, elle impose un nombre limité de

[1] L'implémentation actuelle n'est cependant pas parfaite car elle ne permet de déterminer si une zone est fermée que pour les zones dans lesquelles est présent au moins un de nos agents.

tours laissés aux agents pour compléter un niveau. Ces trois règles n'ont pas, à nos yeux, la même importance. De là découle l'ordre de priorité suivant : atteindre une tuile cible, éviter les tuiles interdites et enfin respecter le nombre de tours autorisé. Bien entendu, ces règles ont été un fil directeur pour notre prise de décisions.

Les actions

Nous avons limité l'interaction entre nos personnages et leur environnement à trois types d'action : se déplacer, interagir, ne rien faire. Les déplacements étaient restreints aux huit directions cardinales (nord, nord-ouest …), mais finalement ils ne sont plus restreints qu'à seulement six directions (nord-est, est, sud-est, sud-ouest, ouest et enfin nord-ouest). Ils ne sont pas enclins à évoluer au cours de la session. Dans le cas où un PNJ est amené à activer un dispositif, deux (2) types de mécanismes sont différenciés : les plaques de pression ("*pressure plate*") et les interrupteurs ("*button*"). Tandis que les plaques de pression ne requièrent que la présence d'un PNJ sur elles pour ouvrir une porte, les interrupteurs doivent être explicitement activés, c'est-à-dire que le PNJ doit effectuer une action "appuyer sur le bouton" pour provoquer l'ouverture de la porte associée.

Les agents

Un PNJ (Personnage Non-Joueur) est un agent dont nous avons le total contrôle. Ce type d'agent est composé – en plus de ses informations principales telles que sa position – d'un état dont l'ensemble des états possibles est représenté par une machine à états finie. L'agent a pour responsabilité de trouver comment remplir son objectif, en trouvant par exemple le chemin pour y parvenir. En revanche, un gestionnaire de mission lui assigne cet objectif (cf. la section sur le MiCoMa), dans certains cas l'agent peut lui-même s'assigner un nouvel objectif. Nous reviendrons plus en détail sur certains points dans la partie *Think*.

L'analyse (*Think*)

La seconde étape de notre architecture est le *Think* c'est le moteur de décision qui, en s'appuyant sur le *Sense* que nous venons de présenter, produit une ou plusieurs actions à réaliser. Nous avons choisi de découper cette étape en deux sous étapes. La première est une analyse et une prise de décisions que l'on peut qualifier de macroscopiques, donc une prise de décisions à grande échelle. Autrement dit, cette prise de décisions se fait au niveau de l'ensemble des agents, prenant directement en compte les buts énoncés dans notre modèle. La seconde analyse, plus microscopique, s'effectue au niveau de chaque agent.

Gestionnaire de missions

Le MiCoMa est responsable de prendre les décisions à l'échelle macroscopique. Il assigne les objectifs aux PNJs et permet ainsi de contrôler leur objectif temporaire si nécessaire.

Initialement, son principal rôle consistait à récupérer les informations transmises par le moteur concernant les missions. Il devait alors les décomposer en sous-objectifs et les assigner aux agents les plus adaptés. Il assurait ainsi le bon déroulement des missions, leur correcte exécution et donc, indirectement, la bonne réalisation du niveau concerné. Suite à une modification des modalités du cahier des charges, les missions au sens littéral ont totalement disparu. Le moteur ne fournit alors aucune information directe indiquant la nature d'une mission. La tâche nous incombe donc d'adapter notre modèle en conséquence.

Dorénavant, le MiCoMa contient un arbre de comportement ("*behaviour tree*") qui nous permet d'orienter les PNJs vers le bon type de comportement. La structure de l'arbre de décision est illustrée par la figure ci-dessous. Son évaluation s'effectue de gauche à droite, de haut en bas.

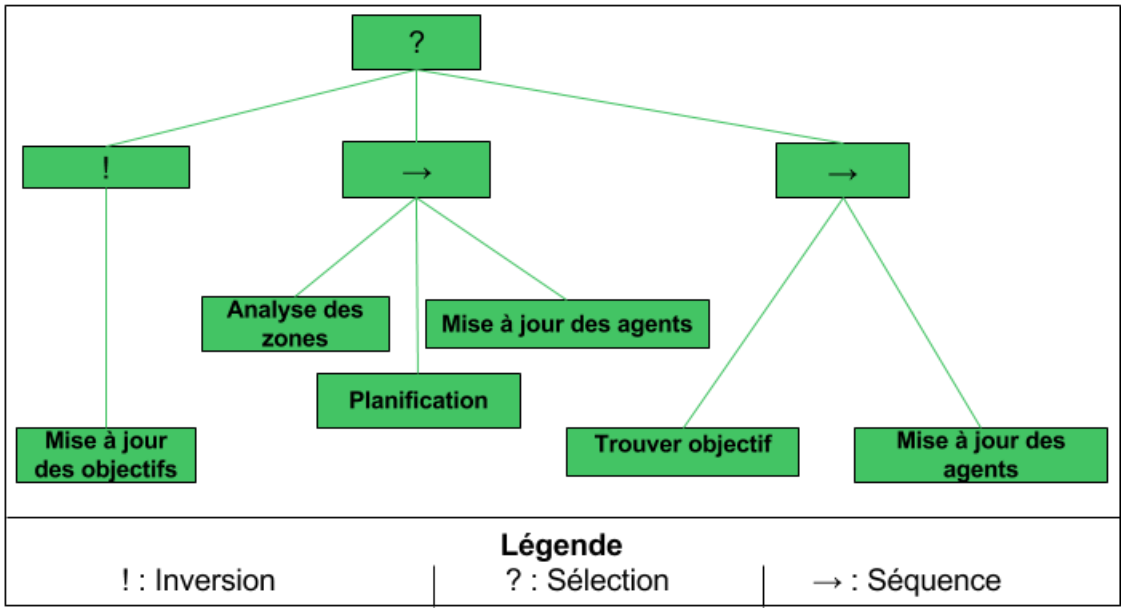


Figure 4 - Arbre de comportement du gestionnaire de mission

Avant de continuer dans le détail, voici une définition des types de blocs utilisés dans cet arbre de comportement :

- Inversion : le résultat retourné par le sous-arbre est inversé ;

- Sélection : exécution séquentielle des blocs enfants tant que l'un de ces enfants n'offre pas une valeur de retour vrai ; et
- Séquence : exécution séquentielle des blocs enfants tant que l'un de ces enfants n'offre pas une valeur de retour fausse.

On amorce alors l'exécution de notre arbre avec un bloc de mise à jour des tuiles en fonction de l'état de l'objectif de chaque agent. En effet, une tuile sur laquelle se situe un PNJ qui a atteint une tuile cible est alors définie, via son attribut descriptif, comme étant "occupée". Cet aspect sera utile afin d'établir la relation spatiale et le déplacement prévu chez les PNJs. Ce processus doit être effectué, quelle que soit la situation rencontrée. L'exécution des deux blocs *séquences* suivants dépendent cependant de cette situation, d'où la nécessité d'utiliser un bloc *sélection* à la racine de l'arbre de comportement. Ainsi, si la mise à jour réussit, nous retournons un échec à la racine afin de pouvoir continuer l'exécution.

Le second enfant de la racine fera la gestion adéquate des cartes possédant des portes. En effet, nous ne sommes pas parvenus, dans le temps investi, à produire un algorithme générique pour le traitement unifié des cartes avec et sans portes. Dans le cas où la carte ne possèderait pas, ou ne semblerait pas posséder a priori, de portes, les PNJs possèdent peu de raisons de devoir collaborer. Plutôt, ils évoluent sur un plan individuel au travers de l'environnement. Ils cherchent à retrouver et se déplacer sur une tuile cible. Cependant, dès qu'un dispositif ou une porte est découvert, l'approche doit forcément être différente. Il faut s'assurer de connaître l'ensemble de la zone contenant ces éléments, mettre en place la coordination des PNJs en fonction des différentes zones de tuiles séparées par les portes, s'assurer qu'un PNJ pourra maintenir l'appui d'une tuile à pression (par exemple), l'envoie d'un ou plusieurs PNJs de l'autre côté de la porte, etc. La gestion des portes est donc beaucoup plus complexe et demande un ajustement conséquent du comportement de nos agents, un comportement spécialisé que nous n'avons pas eu le temps d'agencer à nos algorithmes comportementaux de base.

Avant de se lancer dans la planification, il faut d'abord analyser les zones connues - principalement celles dans lesquelles se trouvent des agents - et ainsi valider la présence, ou non, de portes et/ou de dispositifs. Le bloc "analyse des zones" permet de faire cette vérification. Puisque nous parcourons un arbre de comportement de gauche à droite et que nous ne désirons pas poursuivre l'exécution du second enfant de la racine si nous ne trouvons pas de portes et/ou de dispositifs, la première vérification effectuée dans le bloc nous permet de rapidement poursuivre chez le troisième enfant. En revanche, dans le cas où des obstacles dynamiques ou des dispositifs sont présents dans l'environnement, nous gardons plutôt en mémoire la zone ainsi que les agents qui serviront à la collaboration en transition vers une nouvelle zone (le bloc "planification"). À la base, cette architecture devait permettre d'explorer

différentes zones de manière "pseudo-récursive". En effet, l'idée était que pour une zone de départ avec un certain nombre de PNJs, il fallait d'abord s'assurer de connaître l'ensemble de la zone de départ, obtenir une répartition des tuiles importantes (tuiles cibles, tuiles à pression, tuiles soutenant une porte, etc.), associer les dispositifs accessibles aux portes accessibles, dédier un PNJ à l'ouverture de la porte et au moins un autre PNJ à l'exploration de la zone se situant de l'autre côté de la porte. Une fois l'explorateur dans la zone suivante, la démarche initiale est reprise "de manière récursive" (mais sur plusieurs tours) à partir des tous les PNJs "disponibles" (par exemple, le PNJ devant maintenir la porte ouverte n'était pas "disponible"). Selon les tuiles importantes des différentes zones, nous comptions alors faire faire passer les PNJs nécessaires par le bloc de mise à jour des agents afin de pouvoir, d'abord, accéder à de nouvelles zones ou sinon remplir une tuile cible.

Le dernier bloc séquence est donc exécuté si aucune porte n'est a priori présente dans l'environnement. Ce sous-arbre est responsable d'assigner les tuiles cibles adéquatement aux agents présents dans le niveau. Le MiCoMa s'occupe alors dans cette partie de déterminer la tuile cible la plus proche de chaque agent et de la leur assigner en tant qu'objectif. S'il s'avère qu'aucune ou pas assez de tuiles cibles ne sont disponibles pour tous les agents présents, et donc qu'une partie d'entre eux ne se voit assigner aucune tuile cible, le MiCoMa leur spécifie de partir en exploration. Le second bloc de ce sous-arbre consiste alors à mettre à jour les agents en fonction de l'objectif qui leur a été fourni.

L'objectif principal de cette structure en général est de fournir différents objectifs aux PNJs. Cet arbre peut leur donner, par exemple, les objectifs suivants :

- Atteindre le but final (tuile cible) ;
- Atteindre une tuile en particulier - autre qu'une tuile cible ; ou
- Explorer l'environnement.

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.

Il est à noter que puisque de nouvelles fonctionnalités pouvaient être ajoutées à tout moment au cours de la session, nous avons estimé judicieux d'utiliser un arbre de comportement ici. En effet, le MiCoMa joue un rôle crucial dans l'assignation d'objectifs aux agents. Or, cette assignation dépend énormément de la situation dans laquelle les agents se retrouvent ainsi que l'environnement qui les entoure, donc beaucoup de conditions sont à vérifier avant de prendre une décision concrète. Un arbre de comportement est une structure très flexible : de nouvelles feuilles, de nouveaux sous-arbres sont aisément intégrables. De

plus, elle est très adaptée lorsque de nombreux cas sont à vérifier. Il va de soi alors que l'utilisation d'un arbre de comportement est totalement adéquate pour le rôle que possède notre gestionnaire de missions.

Agents

Les PNJs représentent nos agents contrôlables. Dans un premier temps, chacun d'entre eux se voit assigner un objectif par le MiCoMa tel que mentionné dans la section précédente (un et un seul à la fois). En fonction de cet objectif, chaque PNJ évalue son état et ajuste alors son comportement selon l'état dans lequel il se trouvera. De plus, nous avons décidé de rendre les PNJs entièrement responsables de leurs actions, impliquant la recherche et la gestion des conflits de chemin avec les autres PNJs. Pour une plus grande précision d'action et un meilleur contrôle de nos PNJs, nous avons mis en place une machine à états finis hiérarchique. Elle se compose des états suivants (les noms présentés sont ceux qui sont présents dans le code même) :

- Exploring
    - Explore_Map : exploration générale pour obtenir de l'information sur l'environnement ;
    - Explore_H_Door : exploration dans le but de dénicher l'emplacement de portes cachées ;
    - Explore_DNPC : gestion de conflits de chemin entre agents ;
    - Explore_Waiting : état d'attente, aucune action résultante à la fin du tour ;
    - Explore_Move : déplacement déterminé à l'aide de la carte d'influence ;
- Waiting
- Moving
    - Search_Path : recherche du chemin vers l'objectif assigné
    - Moving_DNPC : gestion de conflits de chemin entre agents ;
    - Moving_Waiting : état d'attente, aucune action résultante à la fin du tour ;
    - Follow_Path : suivi de chemin trouvé dans l'état "Search_Path" ;
    - Arrived : l'agent a atteint sa tuile cible ;
- Interacting : interaction avec un objet tel qu'une porte qui s'ouvre par activation d'un bouton.

Voici un schéma récapitulatif de la machine à états finis hiérarchique illustrant les différents liens et conditions principales pour entrer dans chacun des états :

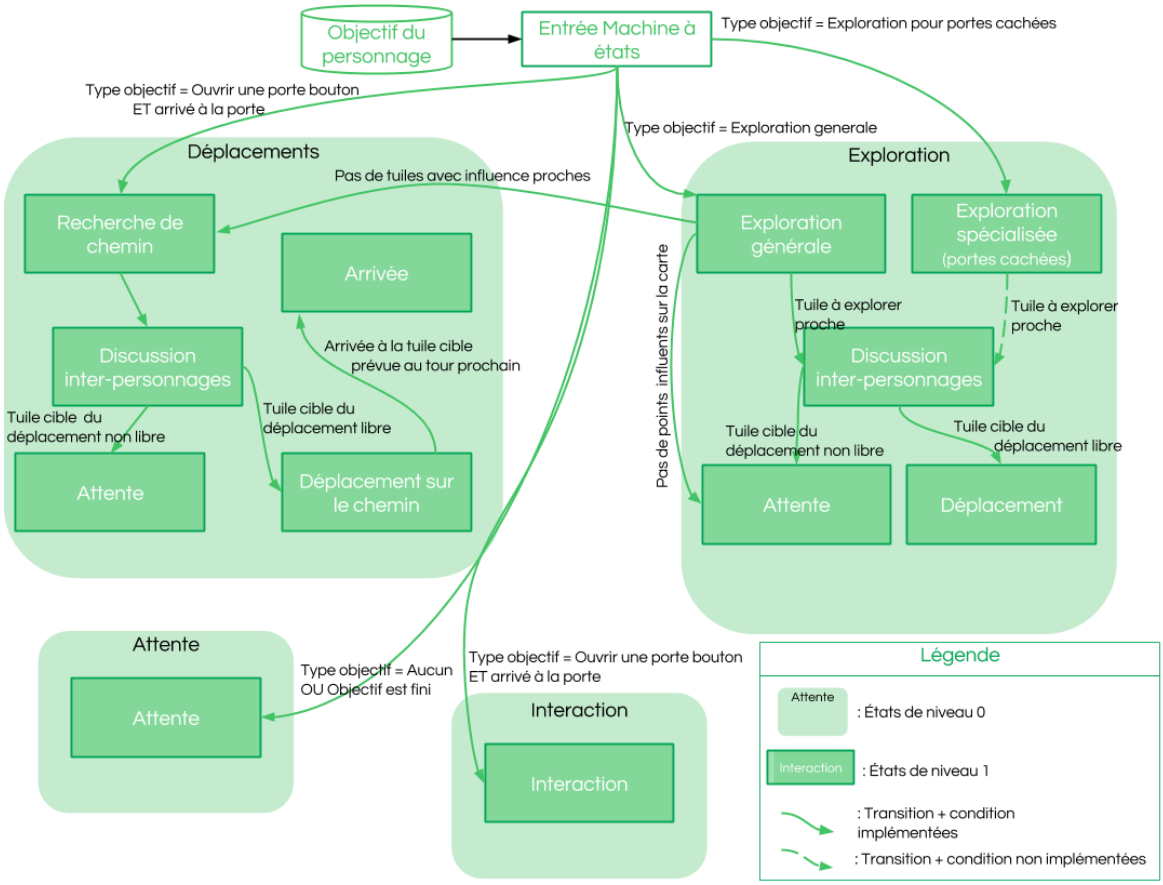


Figure 5 - Machine à états finis hiérarchique des agents

À chaque tour une fonction évaluera l'objectif assigné au personnage et mettra son état actuel à jour à partir de cette unique information. Suite à cela nous allons progresser dans la machine à état de façon classique. Nous avons choisi de réaliser une machine à états finis pouvant évaluer plusieurs états à chaque mise à jour, c'est-à-dire que la mise à jour de la machine à état prendra fin lorsqu'elle sera stabilisée et que l'état suivant sera le même que l'état courant.

Les objectifs

Toute la prise décision faite par un agent est basée sur son objectif, car c'est grâce à celui-ci et uniquement celui-ci que nous choisissons le point d'entrée dans notre machine à états finis. Comme nous l'avons vu, la plupart du temps ces objectifs sont donnés aux agents par le MiCoMa - il existe cependant une exception à cela qui sera expliquée dans la description des différents objectifs que nous avons utilisé pour remplir les fonctionnalités recherchées. Nous avions pensé utiliser seulement quatre objectifs différents au début du développement, mais avec l'avancée et la complexité croissante des cartes proposées, nous avons dû évoluer vers la liste d'objectifs suivante (six objectifs différents) :

- *NONE* : cet objectif est du plus simple qu'il soit, il symbolise le fait de ne rien faire. C'est l'objectif par défaut de nos agents, il fait entrer nos agents dans l'état d'attente.

- *SEARCH_MAP*, *SEARCH_DOOR* : ce sont ici des objectifs d'exploration. Comme nous l'avons mentionnée plus tôt dans le document, l'exploration est divisée en deux parties, l'une pour obtenir de l'information générale sur la carte et la seconde pour trouver des portes cachées, ces deux types correspondent respectivement aux objectifs : *SEARCH_MAP* et *SEARCH_DOOR*.

- *GO_TO* : cet objectif est moins simple que le précédent, il représente une mission de déplacement vers une tuile spécifique désignée par le MiCoMa.

- *GO_TO_BUTTON_DOOR* : très semblable au précédent, cet objectif est apparu lors de la gestion des portes avec bouton (interaction directe), il permet à l'agent de savoir à quelle tuile se rendre puis avec quel objet il devra interagir.

- *GO_TO_EXPLORE* : cet objectif est une exception aux autres objectifs du même type. En effet, il n'est jamais donné à un agent par le MiCoMa mais par l'agent lui-même. C'est un objectif temporaire inclus en quelque sorte dans l'objectif d'exploration de la carte. Il fonctionne de la même manière que l'objectif *GO_TO* à l'exception suivante prêt : lorsque l'agent est arrivé, il se donne de lui-même un objectif de type *SEARCH_MAP*. Un agent va se donner cet objectif lorsqu'il se trouvera dans un cas où chacune des tuiles adjacentes à sa position aura une influence de zéro - soit pas d'influence.

Limites connues du modèle

Malgré des résultats présents ainsi qu'un temps d'exécution satisfaisant – décrits dans la section concernant les résultats – il n'en demeure pas moins que notre approche possède des limites. Jusqu'à aujourd'hui, nous avons identifié dans notre modèle les limites qui sont exposées et expliquées ci-après, il est bien sûr possible que d'autres limites existent et cela a été démontré lors de la présentation des résultats aux cartes de test.

Les objectifs

L'exploration implémentée dans notre modèle, à l'heure actuelle, présente une faille lorsqu'un agent se retrouve sur une tuile dont tous les voisins directs ne possèdent aucune influence. Dans ce cas-là, il reçoit pour instruction de se diriger vers la tuile influente la plus proche. Aucune vérification n'est cependant faite lors de la sélection de cette tuile en tant que

cible. En effet, si cette tuile se trouve malheureusement inaccessible, le cas typique étant qu'elle se situe dans une autre zone que celle dans laquelle se trouve l'agent.

La surprise derrière la porte

Lorsqu'un agent a pour mission de se diriger dans une autre zone, deux objectifs lui sont successivement assignés. Le premier est de se rendre à la porte puis, une fois qu'elle est ouverte, il a ensuite pour instruction de se rendre sur la tuile adjacente à la porte et se trouvant dans la nouvelle zone. Or, elle n'est validée à aucun moment et le problème réside alors dans le fait que cette tuile est possiblement interdite. En effet si cette situation survient, sa prochaine action sera de se rendre sur la tuile "interdite". Avant d'effectuer son déplacement, une validation de tuile est cependant appliquée lors de la recherche de chemin - justement pour éviter les déplacements entraînant un échec du niveau. L'agent entre alors dans un état d'attente, et ce, jusqu'à un dépassement du nombre de tour limite et donc, un échec du niveau. Bien que ce cas soit spécial et plutôt rare, nous préférons indiquer explicitement le comportement que nos agents auraient adopté.

Multizones

La dernière limite connue concerne les zones. Actuellement, nous sommes en mesure de procéder à tout type de traitement et de vérification sur les zones. Cependant, dans la version présentée, les traitements se font sur une seule zone car celle jugée valide est constamment la même. En effet, une structure conservant la totalité des zones valides n'est pas encore disponible, les opérations sur les zones se font donc sur la première zone respectant les contraintes fixées.

Outils de débogage actuels

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. Le script ouvre la page de navigateur correspondant à chaque carte. Il nous permet de visualiser rapidement les résultats obtenus. Une série de fichiers de journalisation est également disponible. Nous sommes effectivement en mesure de journaliser pour chaque exécution, et donc pour chaque carte, les informations suivantes :

- État de la carte : à savoir les tuiles connues/vues, leur type (interdite, plaque de pression, but), celles non vues, si elles appartiennent au chemin d'un PNJ et si elles sont déjà occupées ou non.

- État des arêtes : contenant les informations sur le numéro de tuile sur laquelle se trouve un obstacle, la nature de cet obstacle (mur, porte – vitrée ou pleine – ou fenêtre) et la direction dans laquelle il se trouve.

- État de la carte d'influence : précision sur l'influence de chaque tuile et la portée de vision.

- État de la diffusion des zones : énonçant la valeur de l'identifiant de zone le plus élevé, le numéro des zones fermées, et la répartition des zones sur la carte.

- État de chaque PNJ : mettant en relief la tuile sur laquelle il se situe, l'état dans lequel il se trouve ainsi que son prochain état, l'action qu'il exécute et la liste des tuiles de son chemin qu'il lui reste à parcourir.

Chacune de ces informations est affichée pour chaque tour. Nous pouvons donc retracer le comportement de nos PNJ, leurs causes et leurs conséquences. Les informations concernant la carte, la carte d'influence, l'état d'avancement de la diffusion des zones et les obstacles nous offrent la possibilité d'examiner l'exactitude ou les faiblesses de notre modèle de conception.

Initialement, nous avions l'intention d'investir du temps dans le développement ou l'apprentissage de l'utilisation d'un outil de débogage graphique. En raison du manque de temps ainsi que de l'ampleur qu'aurait représentée un tel développement, cet outil n'a pas vu le jour. Développer un outil à la hauteur de nos attentes aurait requis un projet à part entière. De plus, le système de journalisation mis en place était déjà très utile et selon nous suffisant pour l'instant. Nous avons ainsi préféré mettre la priorité sur l'évolution du projet principal afin de pouvoir valider le plus de niveaux possible.

Présentation des résultats

Cette section présente sous forme tabulaire les résultats obtenus sur les cartes d'entraînement (voir Table 1) ainsi que sur les cartes créées et exécutées pour la compétition finale (voir Table 2).

Table 1 - Résultats sur les cartes d'entraînement

Module	Carte	# Agents	Particularité	Résultat	Cause
Introduction	TC_000	1	Multi-cibles / 1 agent	Réussite	Succès
	TC_001	1	Murs / cibles	Réussite	Succès
	TC_002	1	Murs / 1 cible	Réussite	Succès
	TC_003	2	Multi-agents	Réussite	Succès
	TC_004	2	2 Agents / 2 cibles	Réussite	Succès
	TC_005	2	2 Agents / Goulot d'étranglement	Réussite	Succès
A*	TC_010	1	Corridor	Réussite	Succès
	TC_011	1	Zone interdite	Réussite	Succès
	TC_012	2	2 Agents / Goulot d'étranglement	Réussite	Succès
	TC_013	2	2 Agents / Interblocage	Réussite	Succès
Vision - Partie 1	TC_020	1	Vision 1 tuile	Réussite	Succès
	TC_021	2	2 Agents / Info. partielle	Réussite	Succès
	TC_022	2	Info. partielle / Goulot d'étranglement	Réussite	Succès
	TC_023	2	Info partielle / Goulot d'étranglement	Réussite	Succès

Vision - Partie 2	TC_030	1	Vision 5 tuiles/ Fenêtres	Réussite	Succès
	TC_031	1	Pièges / Multi-cible	Réussite	Succès
	TC_032	1	Vision 5 tuiles	Réussite	Succès
Labyrinthe	TC_040	1	Labyrinthe	Réussite	Succès
Portes	TC_050	1	Tuile Pression / Porte opaque	Réussite	Succès
	TC_051	1	Tuile Pression / Porte transparente	Réussite	Succès
	TC_052	1	Tuile Pression / Choix de porte	Réussite	Succès
	TC_053	2	Collaboration inter-agents / Systémiques	Échec	Nombre limité de tours atteint
	TC_054	1	Porte à bouton (vitrée)	Réussite	Succès
	TC_055	1	Porte à bouton (opaque)	Réussite	Succès
	TC_056	1	Porte cachée (opaque)	Échec	Dépassement du temps imparti
	TC_057	1	Porte cachée (vitrée)	Échec	Nombre limité de tours atteint
	TC_58	1	Tout type de portes	Échec	Nombre limité de tours atteint
Optimisation	TC_060	1	15 ms par tour max	Réussite	Succès
	TC_061	5	5 Agents / Goulot d'étranglement	Réussite	Succès
	TC_062	5	5 Agents / Labyrinthe	Réussite	Succès
	TC_063	15	15 agents	Réussite	Succès

Taux de réussite	87% (31-4)/31*100

Table 2 - Résultats sur les cartes de test

Module	Carte	Résultat	Cause
Final_Cohorte12	F_0	Échec	Dépassement du temps imparti
Final_Cohorte12	F_1	Échec	Dépassement du temps imparti
Final_Cohorte12	F_2	Échec	Nombre limité de tours atteint
Final_Cohorte12	F_3	Échec	Nombre limité de tours atteint
Final_Cohorte12	F_4	Échec	Nombre limité de tours atteint
Final_Cohorte12	F_5	Échec	Nombre limité de tours atteint
Final_Cohorte12	F_6	Échec	Dépassement du temps imparti
Final_Cohorte12	F_7	Réussite	Succès
Final_Cohorte12	F_8	Réussite	Succès
Final_Cohorte12	F_9	Échec	Déplacement vers tuile interdite
Taux de réussite			2/10 (20%)

Table 3 - Résultats des temps d'exécution avec les cartes d'optimisation

Carte	Temps de traitement moyen par tour (en µs)
TC_060	199.1
TC_061	87.2
TC_062	604.9

TC_063	667.8

Ce dernier tableau ci-dessus regroupe les temps moyens d'exécution par tour sur les derniers niveaux dont le temps maximal imparti était de 15 millisecondes. Ces chiffres sont le résultat de 10 exécutions par niveau dont le temps total d'exécutions de tous ces niveaux a été divisé par le nombre d'exécutions. Les tests de performance ont été effectués sur une machine possédant les caractéristiques suivantes : processeur i7-6700 à 4 cœurs, architecture x64, cadencé à 3.4Ghz et 8Mo de cache L3 "*SmartCache*", 32Go de mémoire vive, Windows 10 Éducation 64 bits.

Analyse

Tel que le démontrent les tableaux, ci-dessus, notre implémentation des agents évoluant dans « *AI Boot Camp* » a permis de réussir une majorité des cartes lors de l'entraînement et quelques cartes de l'ensemble de tests.

En effet, suite à un effort combiné de développement, nous remarquons que nous avons pu perfectionner nos algorithmes de manière à réussir 87% des cartes disponibles pour l'entraînement, telle que rapportés dans la Table 1. Plus particulièrement, nous sommes parvenus à offrir une solution pouvant résoudre toutes les cartes des catégories précédant celle concernant la gestion des portes. Pour cette dernière catégorie, nous avons fait face, au moment de l'implémentation, à un temps de développement restreint dû à la charge élevée de travail de la fin de session. Limités en temps, nous avons dû nous satisfaire de l'implémentation partielle de notre solution théorique planifiée, permettant de réussir les cartes 050 et 52 ainsi que 054 et 055. Ces cartes représentent, respectivement, les niveaux d'introduction aux portes utilisant des tuiles à pression et celles nécessitant une activation directe. C'est pourquoi notre implémentation partielle a suffi à leur réussite.

La carte 053, fournie en annexe, demandant une forme de coopération entre les agents afin d'interagir avec les portes liées à tuile à pression, nous n'avons pas été en mesure d'ajuster notre algorithme dans le temps disponible. Les cartes 056 à 058 possédaient des "portes cachées", des portes à activation directe n'étant pas directement disponibles au niveau des "senseurs". Il était alors nécessaire de déduire la (possible) présence de celles-ci et entreprendre une exploration étoffée des divers murs d'une zone candidate en fonction du manque d'information disponible lors de l'exploration de la carte. Cette tâche demandant un haut niveau de gestion de l'information, une exploration suffisante de la carte et le temps nous ayant été compté, nous n'avons tout simplement pas essayé d'aborder cette problématique.

En ce qui concerne les cartes faisant suite aux portes, les cartes 060 à 063, le principal défi relevait de la limitation du temps disponible pour l'évaluation, à chaque tour, de nos algorithmes. Avant de tester notre solution avec ces cartes, nous redoutions les résultats, nous croyant dans l'obligation de mettre en pratique diverses stratégies d'optimisation et de gestion du temps. Nous planifions alors de devoir implémenter des calculs parallèles et du "*time slicing*". Toutefois, il semble que notre implémentation de base ait su répondre aux limites imposées. Cela est d'autant plus évident dans la Table 3 dans laquelle nous pouvons voir que nous atteignons, au maximum, environ 700µs de temps de calcul utile. Cela reste donc bien loin des limites de 15 ms imposées sur les dernières cartes. Par rapport à nos appréhensions, nous ne sommes pas certains de quel facteur entre notre simple concentration sur l'utilisation de structures de données et de fonctions optimales dès le début du projet, la puissance du matériel utilisé ou une simple sous-estimation des restrictions nous a permis de réussir ces cartes.

Suite à notre développement sur les cartes d'entraînement, nous avons lancé nos algorithmes sur des cartes de test, un ensemble de cartes que nous n'avions jamais vues et donc pour lesquelles nous ne pouvions nous préparer, afin de mettre à l'épreuve notre conception. Ces cartes reprennent les divers modules auxquels nous avons dû faire face durant la session, mais dans de toutes nouvelles configurations. Le but étant de pouvoir pratiquer sur les cartes d'entraînement afin de pouvoir espérer bien généraliser les situations dans lesquelles un ou plusieurs agents peuvent évoluer. Nous avons donc soumis notre architecture à ces nouvelles cartes. Comme démontré dans la Table 2, notre modèle de conception est loin d'être sans faille. Nous ne validons malheureusement que deux niveaux sur les dix proposés. Ces deux niveaux réussis sont pour l'un une version alternée du labyrinthe des cartes d'entraînement (#040), et pour l'autre un niveau dans lequel les agents se trouvent au centre de la carte et les tuiles cibles aux extrémités. Il est compréhensible que celles-ci soient complétées avec succès, car elles sont simplement une variation de niveaux qui existent déjà dans les cartes d'entraînement.

Concernant les huit autres niveaux échoués, certains sont tout à fait compréhensibles, notamment ceux incluant des portes cachées ou de la collaboration. En effet, les cartes F_0, F_1 et F_6 sont trois niveaux intégrant une limite de notre architecture, expliquée dans la section faisant part des limites identifiées jusqu'à ce jour concernant le cas dans lequel une tuile interdite était présente derrière une porte.

Le comportement dans agents perçu dans les niveaux F_2 et F_3 dénotent cependant une mauvaise gestion et mauvaise assignation des tuiles cibles découvertes sur la carte. On peut effectivement s'apercevoir que certains agents ne se rendent jamais à une tuile cible découverte. L'hypothèse que nous avons émise serait que deux agents se retrouvent

malheureusement avec la même tuile cible et lorsque l'un d'entre eux l'atteint, l'autre reste en attente, car le premier lui bloque le passage.

La carte des niveaux F_4 et F_5 nous fait part d'une limite qui nous a échappé. Nous ne gérons en effet pas le cas dans lequel plusieurs agents et tuiles cibles se situent dans un corridor. Il faudrait ici assigner la tuile cible la plus éloignée au premier agent et la plus proche au dernier afin que les agents ne se bloquent pas le passage.

Dans le dernier niveau, F_9, un agent se retrouve sur une tuile interdite. On y dénote alors une autre faille de notre implémentation. Il arrive apparemment que, dans certains cas, une mauvaise gestion de chemin ait lieu. En effet, au tour #3, l'agent #5 se déplace "théoriquement" à la position (tuile #155) à laquelle se trouve l'agent #4. Or, puisque ce dernier y est, il bloque ainsi le passage et l'agent #5 reste immobile. Au tour suivant, le premier considère qu'il s'est déplacé sur la tuile sur laquelle il voulait se rendre (tuile #155) : à l'interne du programme, il a effectué l'action de se rendre sur cette tuile et l'a stocké en mémoire en tant que position courante, mais le déplacement étant invalidé par le moteur, il reste donc sur la tuile de départ (#140). Voulant alors se diriger de la tuile #155 à la tuile #170, l'agent #5 effectue l'action de se déplacer dans la direction sud-est, qui s'avère malheureusement être en fait la tuile #156 et qui est interdite.

Il est intéressant de constater que nous réussissons toutes les cartes d'entraînement pour lesquelles nous avons entrepris concrètement les démarchent d'implémentation.

Conclusion

Au terme de ce projet, dont le but était de produire une forme d'intelligence artificielle faible permettant à des agents d'évoluer avec succès, tour après tour, dans un ensemble d'environnements nécessitant une certaine forme d'adaptation face à différents obstacles, nous avons produit une architecture laissant encore place à amélioration, mais orientée dans la bonne direction.

En effet, suite à plusieurs semaines de développement basé sur le modèle « *Sense-Think-Act* », nous avons réussi à produire une structure de "modèle" pouvant contenir l'information utile à nos algorithmes à partir des données de "perception" nous étant fournies par le moteur de « *AI Boot Camp* ». Avec ces informations mises à jour à chaque tour, une classe de gestion haut niveau, le "MiCoMa", évalue à partir d'un arbre de comportement la meilleure stratégie afin de pouvoir diriger les divers agents dans le niveau et ainsi répondre, avec succès, aux contraintes de celui-ci. Plus concrètement, la résultante de cet arbre était une série d'objectifs spécifiques pour les agents tels que trouver et suivre un chemin vers une tuile spécifique ou explorer la carte. Ces agents ont été implémentés avec une machine à états finis hiérarchique à titre de système principal de prise de décision. Selon l'état, il était

nécessaire d'utiliser une carte d'influence ou l'algorithme A* pour finalement assurer une action de déplacement voulant répondre à l'objectif établi par le gestionnaire de missions qu'est le MiCoMa.

Dans le cadre de l'entraînement de notre algorithme, nous sommes parvenus à adapter et ajuster notre algorithme de manière satisfaisante permettant ainsi de répondre à 87% des cartes sur des thématiques telles que la recherche de chemin, la vision partielle, l'optimisation et plus encore. Les 13% de cartes non réussies correspondent à des thématiques pour lesquelles nous n'avions tout simplement pas eu le temps d'implémenter nos solutions théoriques. Plus précisément, nous n'avons pas été en mesure de développer les aspects de collaboration et de gestion des portes cachées. Malgré tout, nous pensions avoir produit une architecture généralisant bien l'adaptation dans les environnements proposés. Or, avec un taux de réussite de seulement 20% sur l'ensemble de tests, il est clair que notre solution nécessite encore certains ajustements, notamment au niveau de l'assignation des tuiles cibles aux agents ou lors de la gestion de conflits de chemin entre agents.

Une alternative à explorer pour la suite serait de mettre en place un système d'apprentissage par renforcement. Nous pourrions ainsi "entraîner" nos agents pour qu'ils puissent choisir eux-mêmes la meilleure décision qu'ils auraient à disposition, et ce, à chaque pas de temps. Cette optique nous permettrait d'éviter d'avoir une exécution déterministe dans la résolution des niveaux. L'application d'un système de réseaux neuronaux ou d'une architecture de processus décisionnels de Markov pourrait, selon nous, fournir des résultats intéressants. Dans ce dernier cas, l'agent possèderait alors une table des états auxquels il aurait accès ainsi qu'une table des actions qu'il pourrait effectuer. En observant l'état actuel de l'environnement, l'agent pourrait mettre à jour sa table de perception de l'environnement. En appliquant une politique, c'est-à-dire en choisissant quelle action faire, il pourrait alors apprendre de son expérience. De plus, il aurait la possibilité de ne faire qu'un nombre limité d'erreurs, principalement durant les premières exécutions du programme d'apprentissage. Pour encore améliorer cet apprentissage, nous pourrions également permettre à l'agent d'effectuer des prédictions en fonction de la représentation interne de l'environnement ou encore lui permettre d'appliquer une politique dite "gourmande" qui lui accorderait la possibilité d'appliquer des politiques pour lesquelles il ne connaîtrait pas le résultat, et ce, dans le but de découvrir quelles conséquences en découleraient. Il serait alors en mesure d'être totalement autonome dans le choix de résolution d'un niveau. Bien que l'application de réseaux neuronaux nous semble intéressante, nous pensons cependant que le nombre actuel de niveaux ne serait pas suffisant pour laisser entrevoir des résultats satisfaisants.

Annexe

Echec de la collaboration

Explication de l'échec de la collaboration avec illustration de la carte TC_053.

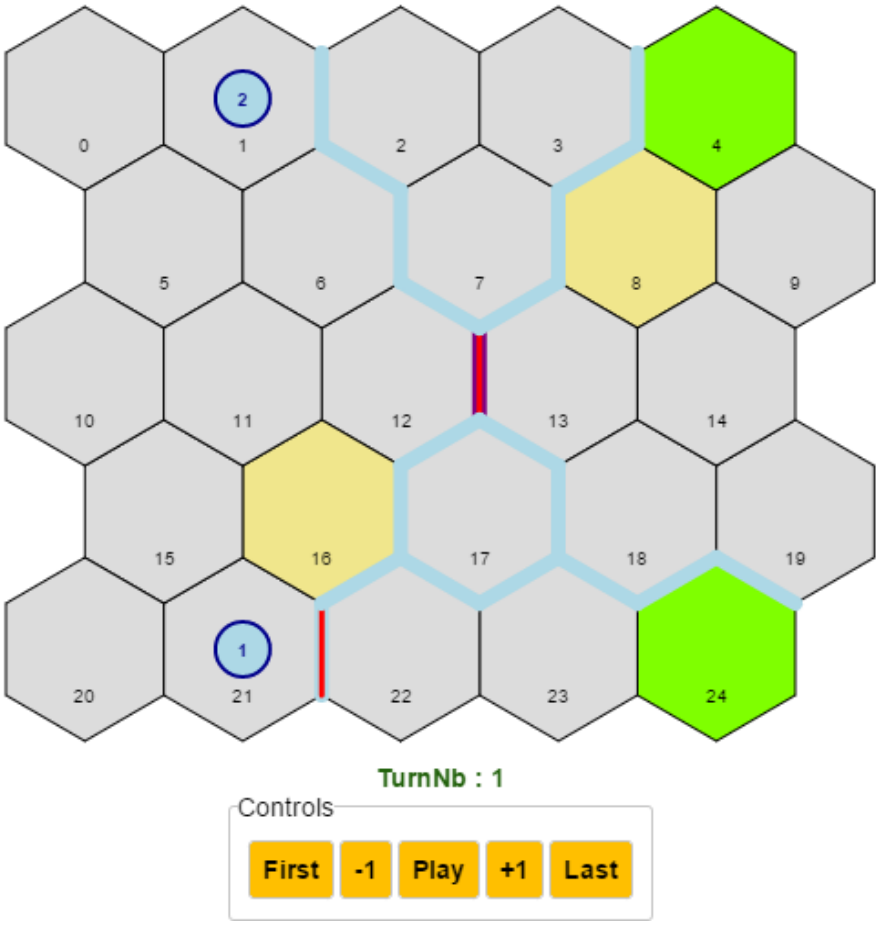


Figure 6 - Premier tour du niveau

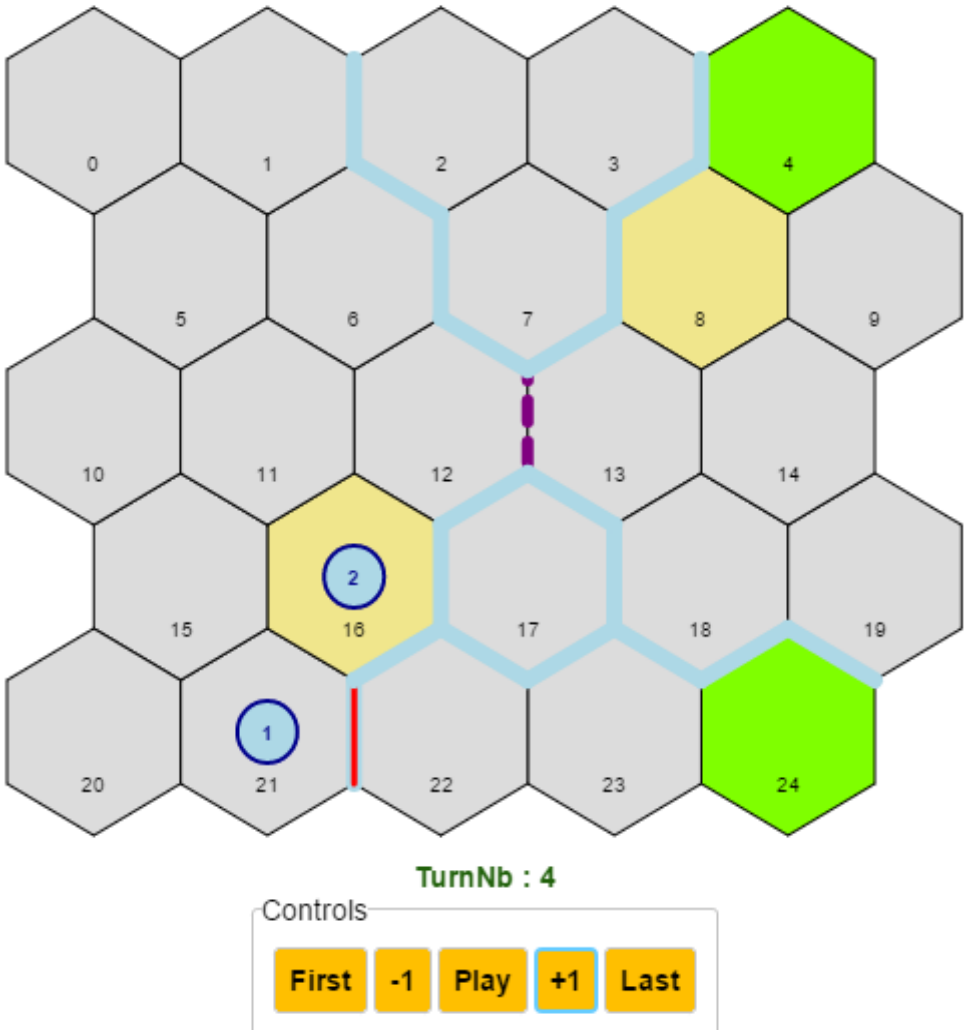


Figure 7 - Tour #4 : on constate que l'agent est bien arrivé sur une plaque de pression. Hélas, l'envoi d'un éclaireur dans la zone suivante n'étant pas implémenté, rien ne se passe et le nombre de tour limite finit par être atteint.