Bordeaux INP
ENSC

C09SFM00 : Modélisation et systèmes complexes

Gestion du personnel minier en zone de creusement intensif

Par Jean BASSET et Aline ROC

Introduction

Les nains[1] sont connus pour aimer creuser, la bière et les zeugmes. Les nôtres ne diffèrent pas : cette simulation appelée « Gestion du personnel minier en zone de creusement intensif » est donc composée de nains qui creusent du granite[2] dans des mines et perdent leur motivation lorsqu'ils manquent de bière. Cette étude ne se concentre pas ici sur leur faculté à creuser - qu'on sait exemplaire - mais bien sur un phénomène naturel multi-échelle impossible à superviser : les comportements et interactions du personnel minier d'une population nanique, ainsi que les problématiques d'approvisionnement en bière inhérentes au creusement intensif dans du granite.

On s'intéresse tout particulièrement à la question de l'influence du nombre de nains et de leur rétention mémorielle sur l'efficacité générale du système.

L'objectif de ce projet, qui s'inscrit dans le module SMA, est de réaliser un projet original, depuis la spécification des comportements jusqu'à l'analyse des résultats de la simulation, impliquant la réalisation d'un environnement 3D et sa population par un système multi-agents à l'aide du moteur moteur de jeu multi-plateforme Unity (dans sa version 5.3.8f2) développé par Unity Technologies.

[1] On assimilera ici les « nains » à des créatures humanoïdes imaginaires de petite taille, avec une barbe, deux bras, deux jambes et un casque.
[2] Il s'agit bien de la roche spécifique « granite » et non de « granit », terme commercial utilisé dans l'industrie extractive, indépendamment de sa lithologie.

Sommaire

Principe de la simulation

La simulation concerne une population nanique évoluant dans un environnement 3D de dimensions 500*500. Chaque représentant de cette population est un **agent** autonome, réactif, proactif, et doté de capacités sociales.

Les agents disposent de jauges mesurant :

- Le désir de travailler ;
- La satisfaction de leur soif - directement corrélée à leur consommation de bière ;
- L'état de leur pioche

Ainsi, quoique l'objectif global des nains demeure l'extraction de granite, leur comportement est tempéré par leurs désirs.

Chaque nain dispose à chaque instant d'**une et une seule activité** choisie parmi les possibilités suivantes :

Liste des activités possibles					
Explorateur	Déviant	Vigile	Approvisionneur	Mineur	En chemin vers la forge

Le comportement *observable* des agents se résume à leurs **déplacements** (vers une mine, vers la forge, les uns vers les autres). Ledit déplacement ayant systématiquement un but, car issu d'un choix - ou d'une réaction directe à l'environnement. Notons cependant qu'une partie du processus de décision implique une **dimension aléatoire** (via System.Random de .Net).

Chaque agent dispose d'**informations** valides relatives à son propre état (position sur la carte, soif, motivation, usure de la pioche, activité actuelle et temps passé depuis le début de cette activité). A ces informations s'ajoutent une **représentation du monde** basée sur des connaissances incomplètes : la mémoire. Ainsi, un nain conserve en mémoire les informations les plus récentes relatives aux mines et aux autres nains. Ces informations sont issues des perceptions de l'agent (s'il voit une mine ou rencontre un nain, il garde en mémoire les informations telles qu'elles sont au moment de la rencontre) et des communications entre agents.

Notons que la mémoire des agents n'est pas faillible - ils ne peuvent pas se tromper sur leur propre représentation du monde - mais elle est **limitée dans le temps** : les nains oublient une information (une mine, une personne) s'il ne l'a pas vue et n'en a pas entendu parler depuis un certain temps.

Notice d'installation et d'utilisation

A. Lancement de la simulation

Pour lancer la simulation, ouvrir le projet dans l'éditeur d'Unity, ouvrir l'onglet game (cocher maximise on play pour une meilleure lecture du jeu), et appuyer sur Play.

Entrer les paramètres de la simulation dans le panneau de lancement, et appuyer sur « Start the simulation ! ». Les paramètres de ce panneau sont décrits plus bas dans la partie décrivant l'UI.

Notons que le projet a été entièrement codé sous la version 5.3.8f2 de Unity. Des simulations ont été lancées avec succès sous la version 5.6.4f1 mais le positionnement de l'UI semble affecté par le changement de version. Si un problème survient lors du lancement ou de l'exécution du projet, nous recommandons l'utilisation de la version 5.3.8f2.

B. Contrôle de la caméra

La caméra se pilote à partir de la position du curseur de la souris. En sortant le curseur de l'écran, la caméra se déplace dans la direction du curseur. La molette de la souris permet de modifier l'élévation de la caméra, s'approchant ou s'éloignant du sol. Il est de plus possible d'effectuer une rotation de la caméra en maintenant enfoncé le bouton droit de la souris et en déplaçant le curseur le long de l'axe horizontal.

La caméra peut être centrée sur un agent ou sur une mine. Pour ce faire, placer le curseur de la souris au-dessus de la cible et réaliser un clic gauche. La cible sera sélectionnée, et les informations relatives à celle-ci apparaîtront sur le côté gauche de l'écran. La position de la caméra sera bloquée au-dessus de la cible, et suivra son mouvement si besoin. Tant que la cible reste sélectionnée, la caméra ne pourra pas être déplacée, mais il est toujours possible de modifier son élévation. Pour dé-sélectionner la cible et reprendre le contrôle de la caméra, cliquer sur le terrain.

Le fait de centrer la caméra sur un agent fait de plus apparaître une sphère au dessus de lui. Cette sphère permet dans un premier temps de mieux distinguer l'agent sélectionné de ses voisins, et change de couleur en fonction de l'activité actuelle du nain.

Le code couleur est le suivant : Explorateur : vert ; Mineur : jaune ; Approvisionneur : bleu ; Vigile : rouge ; Déviant : noir.

C. Lecture et utilisation de l'interface utilisateur (UI)

Afin de faciliter la compréhension de l'état de la simulation, une interface utilisateur est mise place. Celle ci comprend six panneaux.

1. Le panneau de lancement de simulation

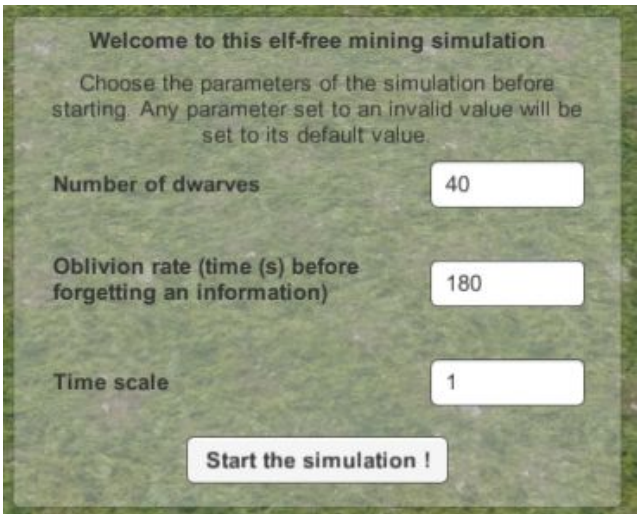

Figure I : panneau de lancement

Ce panneau permet de modifier les paramètres de la simulation et de la lancer.

Les paramètres sont :

- le nombre de nains qui apparaitront dans la simulation ;
- le temps de stockage des informations en mémoire ;
- l'échelle de temps.

Une fois le **temps de stockage des informations** écoulé, toute information stockée par un nain datant d'une période antérieure sera oublié. Dans le cadre de cette simulation, ces informations sont les autres nains et les mines (on parle alors de nains et de mines *connus*).

Augmenter ou baisser l'**échelle de temps** influe sur la vitesse d'écoulement du temps. Attention cependant, cela peut avoir des effets négatifs sur les comportements des nains.

2. Le panneau titre

Figure II : panneau titre, **Figure III :** statistiques générales

3. Les statistiques générales

Cet onglet présente les données actuelles de la simulation. Il contient :

- Time(s) since start ;
- Total gold[3] mined ;
- Total beer drank.

Il s'agit respectivement du temps écoulé en seconde depuis le lancement de la simulation, de la quantité totale de granite extraite depuis le lancement de la simulation et de la la quantité totale de bière bue par les nains depuis le début de la simulation.

4. Sélection de nain

Figures IV, V, VI : sélecteur de nain, panels d'informations de nain (IV) et de mine (V)

Cet onglet contient dans un panneau défilant des boutons permettants de sélectionner et de centrer la caméra sur un nain. Cette sélection a le même effet que celle décrite dans la partie B. Contrôle de la caméra.

Il est possible de faire apparaître et disparaître cet onglet grâce au bouton en haut à droite du panneau.

[3] Cette population nanique semble utiliser le granite comme unité monétaire, allant parfois jusqu'à utiliser l'appellation « gold ».

5. Informations liées à un nain

Ce panneau contient les données liées à un nain.

Il s'agit d'abord d'informations générales (le nom du nain ; son activité actuelle ; le nombre d'autres nains qu'il connait ; le nombre de mines qu'il connait ; la quantité de bière qu'il porte sur lui ; ses jauges de satisfaction de sa soif, d'envie de travailler et d'état de sa pioche).

Ce panneau fournit également quelques statistiques sur le comportement du nain au cours de la simulation :

- la quantité totale de granite qu'il a pu miner ;
- la quantité de bière qu'il a bue ;
- la quantité de bière qu'il a apportée à d'autres nains ;
- le nombre de déviants qu'il a renvoyé travailler ;
- Les temps passés sur chaque activité depuis le début de la simulation.

6. Informations liées à une mine

Ce panneau contient les informations liées à une mine, organisées en informations générales puis statistiques, de même que le panneau d'information des nains.

Les informations générales sont :
- Le nom de la mine ;
- La quantité de granite actuellement disponible dans la mine ;
- le nombre de nains travaillants actuellement dans la mine ;
- un panneau permettant de sélectionner ces nains et d'afficher leurs informations.

Par ailleurs, les statistiques contiennent :
- la quantité de granite extraite de cette mine depuis le début de la simulation ;
- le nombre d'interaction depuis le début de la simulation.

Ce dernier nombre correspond au nombre de fois où un nain a interagit avec la mine. Les interactions prises en compte sont : entrer dans la mine, découvrir la mine.

Gestion de projet, Répartition du travail

Figure VII : Outils utilisés : GitHub ; Google Drive; Unity 3D; Visual Studio (2015 et 2017)

L'ensemble du code et des commentaires est fait en anglais[4].

La gestion du versionning s'est fait via git, le projet étant stocké dans un dépôt GitHub. Dans le cadre d'un travail en binôme avec communication quasi-constante entre les membres, aucun choix de normalisation n'a été fait sur les commentaires des commit : ceux-ci sont réalisés en français ou en anglais et décrivent généralement la ou les modifications associée(s) à ce commit.

Le travail sur le projet a été principalement réparti et réalisé suivant le tableau de synthèse page suivante.

[4] Nous assumons pleinement ce choix mais souhaitons faire part de nos regrets concernant le nom de la variable knownDwarf dont le nom français - nainConnu - aurait adouci nos austères sessions de code

Partie		Synthèse de répartition du travail
Conception, réflexions préliminaires		Jean + Aline
Map, décors		Jean
UI		Jean
Implémentation de la structure des agents	Structure de la mémoire	Aline
	Pondération	Aline
	Decision-making	Aline
	Gestion des déplacements	Jean
Interactions	Vision	Jean
	Réaction des agents	Aline + Jean
	Communication	Aline + Jean
Tests	Adaptation des variables	Aline + Jean
	Traitement des résultats	Jean
Rédaction du rapport		Aline + Jean

Notons que cette répartition du travail n'est pas absolue : des points réguliers et de nombreux échanges (aide, questions) ont eu lieu tout au long du travail de réflexion, de code et d'analyse. Par ailleurs, certains points sensibles ont été réalisés en pair-programming - ainsi que quelques séances de débogage.

Les réflexions ont été structurées dans des documents tableurs stockés dans un drive commun. En terme d'ordonnancement des tâches, nous avons procédé de manière relativement incrémentale, considérant des fonctionnalités *nécessaires* et *optionnelles* comme décrit ci-dessous.

Map, environnement

Fonctionnalité	Type	Priorité (1-5)
Map praticable	Base	⋆⋆⋆⋆⋆
Présence de mines (entre 4 et 15)	Base	⋆⋆⋆⋆
Minerai dans les mines (spawn, etc)	Base	⋆⋆⋆
Décors divers	Bonus	⋆

UI

Fonctionnalité	Type	Priorité
Display les informations générales (granite miné, etc)	Base	⋆⋆⋆⋆
Display des informations d'un nain	Base	⋆⋆⋆
Lock sur un agent	Base	⋆⋆
Menu permettant de paramétrer des variables	Base	⋆⋆
Mouvement de la caméra (droite, gauche, rotation, zoom blocage en dehors des limites, etc)	Base	⋆
Display des informations d'une mine	Bonus	⋆⋆⋆
Liste cliquable des agents	Bonus	⋆⋆

Déplacements

Fonctionnalité	Type	Priorité
Déplacement des nains	Base	⋆⋆⋆⋆⋆
Pathfinding	Base	⋆⋆⋆⋆
Identification d'une destination acceptable	Base	⋆⋆⋆
Entrée / sortie d'une mine	Base	⋆⋆
Routes, accelerations	Bonus	⋆
Animation (marche, course)	Bonus	⋆

Activités, Mémoire

Fonctionnalité	Type	Priorité
Choix d'activité	Base	⋆⋆⋆⋆⋆
Initialisation de la mémoire	Base	⋆⋆⋆⋆
Apparition des nains	Base	⋆⋆⋆
Baisse des jauges à intervalles fixes	Base	⋆⋆⋆
Trigger la remontée des jauges	Base	⋆⋆⋆
Déplacement vers le stockage de bière	Base	⋆⋆
Déplacement vers la forge	Base	⋆⋆

Interactions

Fonctionnalité	Type	Priorité
Mémorisation des informations observées	Base	⋆⋆⋆⋆⋆
Communication	Base	⋆⋆⋆⋆⋆
Réaction à l'entrée dans le champ de vision	Base	⋆⋆⋆⋆
Vision des nains (mines)	Base	⋆⋆⋆
Vision des nains (autres nains)	Base	⋆⋆
Système déviant - approvisionneur	Bonus	⋆⋆
Système déviant - vigile	Bonus	⋆

Présentation des comportements

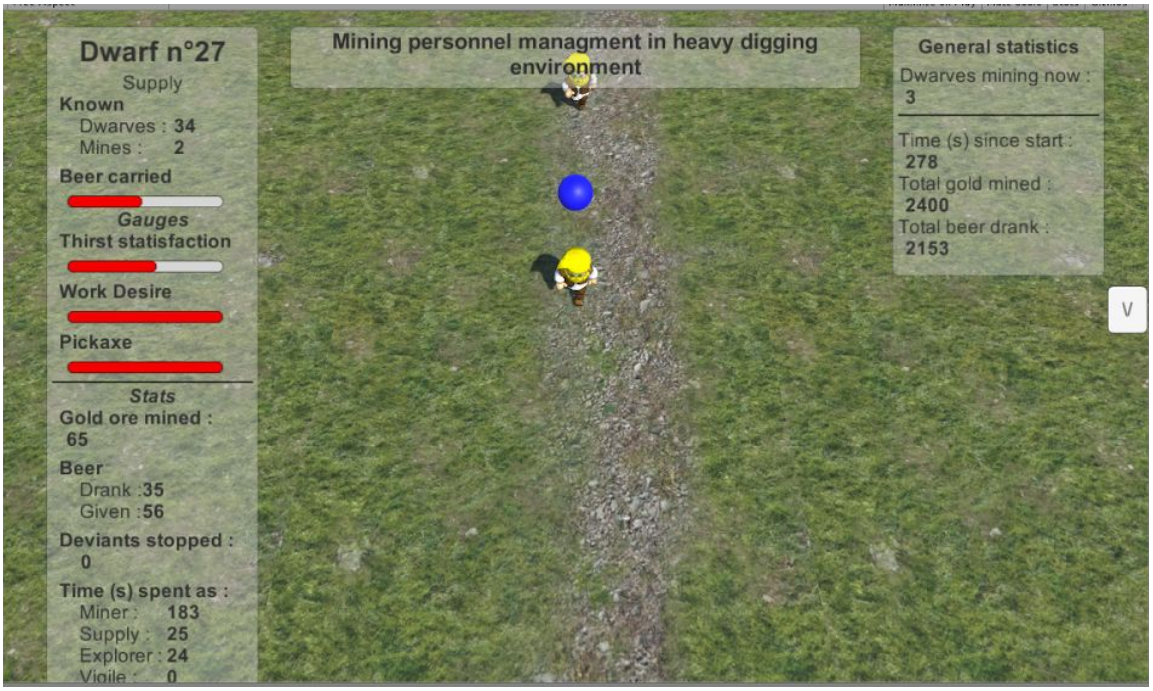

Figure VIII : Approvisionneur qui court sur une route, capture d'écran

Le comportement global de notre SMA relève de la coopération : les agents partagent un même but, qui est l'augmentation de la quantité *globale* de granite miné. En cela, les vigiles et approvisionneurs tendent tout autant vers cet objectif que les mineurs, puisqu'ils visent à diminuer la quantité de nains déviants.

La simulation a lieu dans un monde clos : le nombre de nain et de mines est limité, les limites physiques de l'environnement ne sont pas dépassables, les jauges ont une capacité limitée. Ainsi, il nous est possible d'élaborer une structure de prise de décision pour nos agents selon un cycle perception-décision-action. En l'occurence, nos agents suivent un modèle mi-délibératif mi-réactif.

A. Mobilité des agents

Les déplacements des agents à travers l'environnement sont déclenchés par la fonction MoveTo du script DwarwBehaviour.cs. Si un nain est dans une mine, cette fonction l'en fait sortir. Ensuite, l'agent se met à marcher dans la direction souhaitée grâce à la fonction SetDestination applicable à un NavMeshAgent. Les processus amenant l'agent à décider d'une destination sont décrits plus bas (Modèle délibératif, Modèle réactif).

Les agents se déplacent sur un NavMesh que nous avons créé (« bake » dans le langage de Unity) à partir de notre terrain. Ce NavMesh consiste en la surface navigable par nos agents. Le NavMesh permet de plus de définir des zones, en changeant leur coût dans l'algorithme de pathfinding de Unity. Dans notre terrain, trois zones sont utilisées :

- Walkable : il s'agit de la surface sur laquelle les nains peuvent se déplacer, en marchant.
- Road : cette zone correspond à la route partant du village au centre de la carte et se dirigeant vers le bord. Le coût de déplacement sur cette zone est deux fois moins important que sur la zone Walkable. Les nains se déplaçants sur la route courent au lieu de marcher et se déplacent donc significativement plus vite.
- Not Walkable : comme son nom l'indique, cette zone correspond aux éléments sur lesquels les nains ne peuvent pas marcher. Il s'agit par exemple des éléments formant le décor des mines, de l'eau, et (bien entendu) de la lave du volcan.

Notons que les destinations sont nécessairement accessibles :

- lorsqu'il s'agit d'un nain, celui-ci se trouve déjà sur le navmesh ;
- lorsqu'il s'agit d'une mine, on cible en réalité un empty gameobject (mineEntrance) qui se situe devant la mine ;
- lorsqu'il s'agit d'une destination « aléatoire », celle-ci est générée via les fonction GetRandomDestination et FixDestination du script DwarfMemory.cs.

FixDestination permet d'assurer que la destination « aléatoire » se situe sur le NavMesh (si la destination aléatoire n'est pas accessible, Unity permet de récupérer la destination accessible la plus proche via NavMesh.SamplePosition). Des points du NavMesh n'étants cependant accessible par aucun itinéraire, nous vérifions au début de l'Update du comportement des nains que leur destination est accessible. Sinon, les nains recherchent une nouvelle destination.

B. Perception de l'environnement

La perception de l'environnement se résume ici au sens de la vue : elle est modélisée par les fonctions DwarvesInSight() et MinesInSight() du script DwarwBehaviour.cs. Ces fonctions listent les éléments qui sont « à portée de vue[5] » (en terme de distance) et « visibles » dans la mesure où aucun objet ne vient interférer dans le champ de vision (modélisé par un raycast envoyé vers les objets potentiellement visibles).

Par ailleurs, lorsque l'agent est arrivé à destination (dans la fonction Update() du script DwarwBehaviour.cs), il détecte la proximité du stock de bière ou de la forge et « interagit » avec eux - adaptant ses jauges en conséquences.

C. Communication

Les discussions (qui ont lieu dans la fonction Update() du script DwarwBehaviour.cs) permettent une mise à jour des connaissances d'un nain : lorsqu'un agent en voit un autre qu'il ne connait pas et que ces agents sont suffisamment proches[6] l'un de l'autre, ils discutent. On verra plus loin qu'un nain en apercevant un autre peut choisir de s'approcher de ce dernier.

[5] Il s'agit d'une variable globale configurable (dans la classe VariableStorage). Étonnamment, les nains de la population étudiée semblent avoir tous très exactement le même degré de myopie.
[6] Chez les nains comme chez les hommes, la distance sociale (c'est la distance propre à une relation professionnelle ou commerciale) se situe entre 1,20m et 3,60m. Dans le code, cette valeur est de 3 (en unités de distance Unity).

L'échange de connaissance consiste à ajouter des informations à propos de nains et de mines inconnus et réactualiser les données sur les nains connus et mines connues si et seulement si les informations apportées par l'interlocuteur sont plus récentes. On considère qu'un agent ne peut pas « en connaissance de cause » communiquer de fausses informations. Cependant, les données transmises concernent des éléments de l'environnement perçus à un instant t, et qui pour beaucoup ont probablement été modifiés depuis.

D. Action sur l'environnement

Les nains ont une influence directe sur certains éléments de leur environnement :

- S'ils sont dans une mine, ils creusent et influent sur la quantité de granite encore disponible
- S'ils sont vigile ou approvisionneur, leurs interactions avec les autres nains influent directement sur les jauges de leurs interlocuteurs : un nain appréhendé par un vigile voit son envie de travailler remontée au maximum ; un nain assoiffé va consommer tout ou partie de la bière transportée par l'approvisionneur.

E. Gestion des ressources

Un nain dispose des informations suivantes :

- Mine dans laquelle il se trouve (s'il est dans une mine) ;
- Destination qu'il a en tête (même s'il est interrompu) ;
- Activité actuelle ;
- Date du dernier changement d'activité ;
- Niveau d'apaisement de l'envie de bière ;
- Envie de travailler ;
- Quantité de bière transportée ;
- Etat de la pioche.

Notons que des informations sont également conservées au niveau de la simulation de part leur intérêt statistique, mais pas accessibles du point de vue de la *conscience* du nain :

- Quantité de granite miné ;
- Quantité de bière bue ;
- Quantité de bière distribuée ;
- Nombre de déviants appréhendés ;
- Temps passé en tant que mineur, approvisionneur, explorateur, vigile, deviant.

Modélisation des comportements :

Modèle délibératif

Considérant les dimensions de notre environnement, il semble nécessaire de permettre à nos agents d'être *proactifs* et de *mémoriser* ce qu'ils perçoivent de cet environnement.

A intervalles réguliers, nos agents raisonnent et décident d'une action à effectuer en fonction de leurs connaissances (ce qu'ils croient, ce qu'ils savent, ce qu'ils se souviennent). Ce raisonnement relevant du modèle délibératif a lieu en trois temps :

1) Premier arbitrage : ils se demandent dans un premier temps s'ils peuvent, veulent et/ou doivent changer d'activité.
2) S'ils décident de changer, ils se demandent alors quelle nouvelle activité prendre.
3) Cette activité choisie, ils décident enfin d'une destination.

Ainsi, chaque agent a l'initiative quant à son état (l'activité actuelle) et le comportement associé (généralement un déplacement).

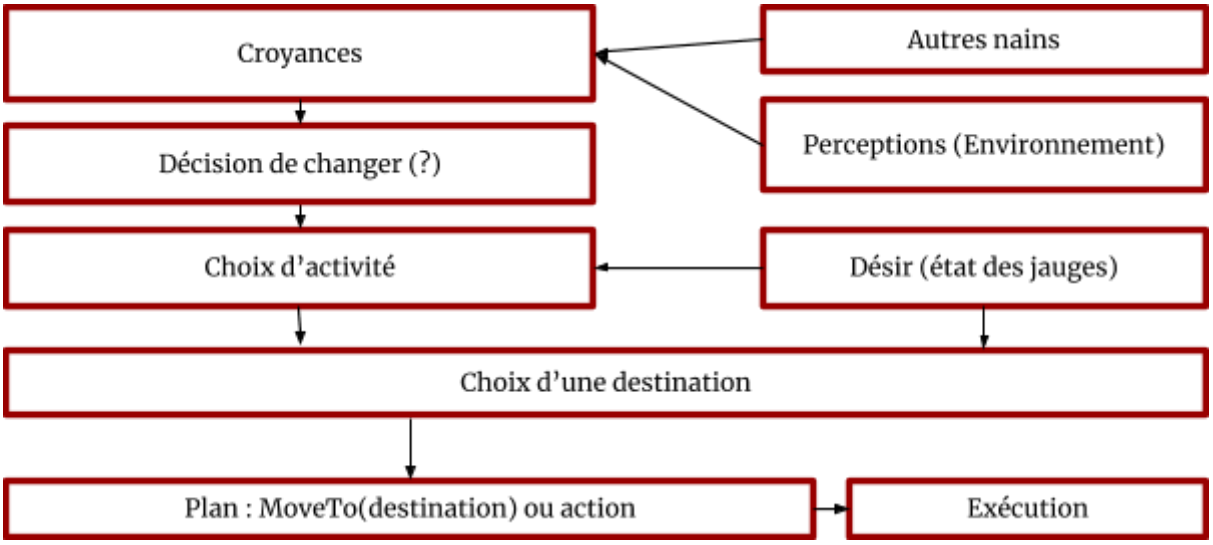

Figure IX : Fonctionnement de notre modèle délibératif

A. Le changement d'activité

A chaque interrogation, l'agent *peut* toujours changer d'activité. Le procès de décision (changer ou non d'activité ?) est affecté par les croyances (utilité de l'activité actuelle, informations à disposition, intérêt porté au temps passé sans changer d'activité, etc). Il s'agit ici de favoriser les actions susceptibles de réaliser les buts et de favoriser les actions adaptées à la situation courante.

Ce choix a lieu dans la première partie[7] de la fonction RethinkActivity() du script DwarfMemory. Les éléments influant sur ce choix sont de plusieurs ordres.

D'abords, des éléments généraux sont pris en compte :

- une activité qui dure depuis trop peu de temps ne changera pas, évitant ainsi à nos agents un comportement équin (ou plutôt canin[8]) des plus primaires ;
- un nain qui creuse perd toute notion du temps ;
- s'il n'est pas déjà déviant, l'envie de bière influe sur le désir de changer d'activité

Ensuite, des éléments corrélés à l'activité actuelle :

Activité	Eléments poussant à changer d'activité	Eléments entraînant un choix certain	Eléments poussant à poursuivre l'activité actuelle
Déviant	soif apaisée	-	soif haute
Explorateur	connaissance de mines pleines	-	pas suffisamment de mines pleines connues
Vigile	connaissance de peu de déviants	connaissance d'*aucun* déviant	-
Approvisionneur	connaissance de trop peu de nains	-	transporte de bière en quantité suffisante
Mineur	pas actuellement dans une mine, et connaissance trop faible de mines pleines ; ou actuellement dans une mine pratiquement vide	aucune mine connue	actuellement dans une mine encore bien remplie
En chemin vers la forge	pioche en état encore correct	-	pioche en mauvais état

Les éléments du tableau ci-dessus affectent la décision de changer ou non d'activité. De plus, le niveau d'influence de ces éléments est pondéré par le désir de travailler.

[7] dans la région STEP TWO du code (la région STEP ONE n'étant qu'une prise de conscience du temps écoulé depuis le dernier changement d'activité)
[8] contrairement l'opinion commune, Buridan dans son *Expositio in De caelo* met en scène, non pas un âne, mais un chien confronté au cruel dilemme connu sous le nom de *paradoxe de l'âne de Buridan*

B. Le choix d'activité

Si la décision de changer d'activité est prise, l'agent examine dans un premier temps les les opportunités qui s'offrent à lui (et leur affecte un poids) puis, dans un second temps, il effectue le procès de décision via un tirage aléatoire parmi les choix pondérés.

Ce choix a lieu dans la seconde partie[9] de la fonction RethinkActivity() du script DwarfMemory. Les éléments influant sur ce choix sont de plusieurs ordres.

Les poids sont calculés en fonction des croyances (utilité de l'activité potentielle, etc) et des désirs (état des jauges) de chaque agent.

Ainsi, s'il n'était pas précédemment en train d'effectuer l'activité en question, le poids de chaque activité est corrélé aux éléments suivants :

Activité	Influences
En chemin vers la forge	Mauvais état de la pioche, envie de travailler
Déviant	Envie de travailler, soif
Explorateur	Nombre de mines connues, état de la pioche, envie de travailler
Mineur	Nombre de mines (pleines !) connues, état de la pioche, envie de travailler, coefficient supplémentaire arbitraire[10]
Approvisionneur	Nombre de nains assoiffés connus, soif personnelle, quantité de bière transportée actuellement
Vigile	Nombre de nains déviants connus, soif personnelle, proximité d'un nain connu déviant (ou à risque car assoifé)

Le choix effectué, la fonction met à jour l'activité actuelle, et enregistre la date du changement et l'intitulé de l'activité précédente.

[9] l'analyse d'opportunités se trouve dans la région STEP THREE du code ; le procès de décision dans la partie STEP FOUR
[10] Il s'agit ici de tenir compte d'un élément naturel communs à tous les nains : les nains aiment le granite.

C. Le choix de destination

Il arrive que la fonction GetNewDestination expliquée dans cette section soit appelée alors que l'agent n'avait pas encore atteint sa destination précédente (typiquement : s'il a interrompu momentanément son activité suite à un appel du modèle réactif). Dans ce cas, la fonction renvoie la destination que le nain avait conservé en mémoire.

Dans tous les autres cas (choix de destination suite à un changement d'activité, nain arrivé à sa destination précédente), l'agent examine dans un premier temps les les opportunités qui s'offrent à lui (et leur affecte un poids) puis, dans un second temps, il effectue le procès de décision via un tirage aléatoire parmi les choix pondérés.

L'agent décide d'une destination en pondérant son choix en fonction de la distance et/ou de la pertinence de la destination relativement à l'activité choisie (en fonction de l'état de ses désirs et de ses connaissances). Les destinations potentielles sont les suivantes selon les activités :

Activité	Destinations possibles
Déviant	destination aléatoire dans un faible rayon de distance[11] ; réserve de bière
Explorateur	destination aléatoire suffisamment éloignée de toute mine connue e de lui-même
Mineur	mine connue et non vide (pondération relative au nombre de nains dans la mine concernée, à la distance de la mine)
Approvisionneur	réserve de bière (uniquement si la quantité de bière actuellement transportée est basse) ; nain connu assoiffé (pondéré par la proximité) ; mine connue où des nains ont soif (pondéré par la proximité) ;
Vigile	nain déviant (pondéré par la proximité)

Notons qu'un nain en chemin vers la forge n'hésite pas quant à sa destination. Cette dernière ne fait donc pas l'objet d'un tirage aléatoire. Notons également que, par sécurité, un nain qui ne serait pas parvenu à envisager une destination va automatiquement s'interroger de nouveau sur son activité (étape précédente).

D. A propos de la proactivité

De manière générale, on ne favorise pas les actions qui s'inscrivent dans une démarche à long terme à l'échelle d'un nain. Cependant, le comportement des vigiles et des approvisionneurs s'inscrit dans une démarche à long terme à l'échelle du système.

[11] Ainsi, on peut parfois observer des nains déviants qui tournent *littéralement* en rond

Modélisation des comportements :

Modèle réactif

A ce modèle délibératif s'ajoutent des comportements qui relèvent d'un modèle réactif. Ainsi, les agents modifient parfois leur comportement suite à la manifestation d'un élément externe, au risque de délaisser un moment leur tâche courante.

Les comportements issus de ce processus n'interfèrent pas avec la « vision à long terme » du nain dans la mesure où sa destination initiale est conservée en mémoire de manière à revenir à son occupation après la digression.

Ainsi, lorsqu'il est à proximité d'un élément susceptible de lui apporter une meilleure connaissance de son environnement (typiquement : une mine ou un autre agent), le nain peut décider de s'en approcher. Cette décision est pondérée par son activité et l'état de ses connaissances actuelles.

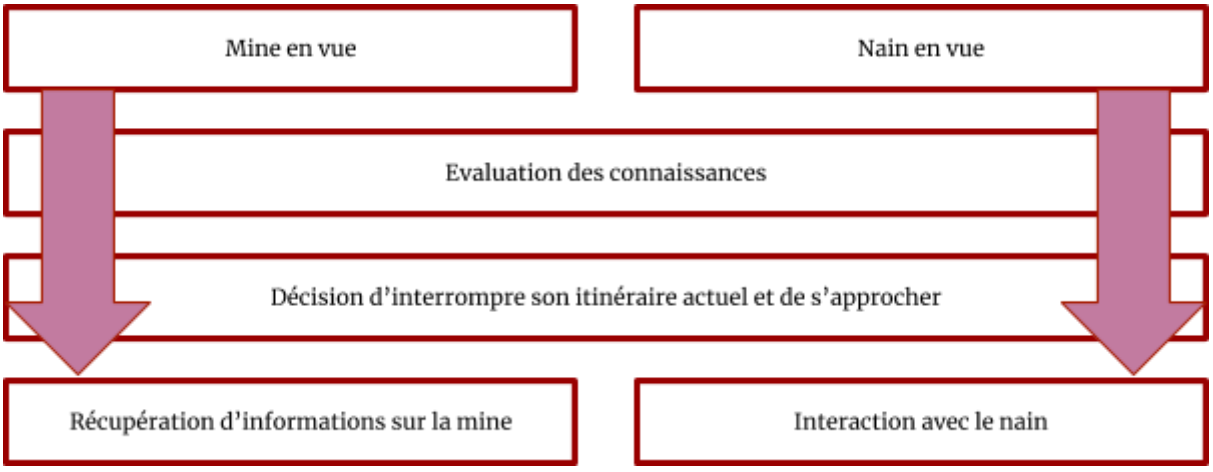

Figure X : Fonctionnement de notre modèle réactif

Les choix issus du modèle réactif ont lieu dans la fonction Update() du script DwarfBehaviour.cs. Ces choix concernent plusieurs éléments.

Tout d'abord, on y retrouve les différents types d'interaction :

- Communication ;
- Interaction vigile-déviant ;
- Interaction approvisionneur-assoifé.

Ces interactions ont lieu entre deux nains « suffisamment proches »[12].

[12] cf. Présentation des comportements, paragraphe C : Communication

Ensuite, on retrouve dans cette fonction les réactions à la vue d'un nain :

- Un **explorateur** ou un mineur s'approche pour parler à un nain qu'il ne connait pas ;
- Un **vigile** va se diriger vers un déviant qu'il voit ;
- Un **approvisionneur** va fournir à boire à un assoiffé ;
- A la vue d'un déviant et/ou d'un assoiffé, un **nain** ayant envie de travailler va spontanément s'interroger sur son activité et sa destination.

On retrouve également les réactions relatives aux mines :

- Un **explorateur** qui aperçoit une mine inconnue va s'en approcher pour en apprendre plus ;
- Un **mineur** va spontanément quitter une mine si elle est vide ;

Figure XI : Réaction du nain lors de l'arrivée à destination

Enfin, les réactions relatives à l'arrivée à destination :

- La forge répare la pioche ;
- La réserve de bière soulage les déviants ;
- La réserve de bière fournit 100 unités[13] aux approvisionneurs ;
- Un **explorateur** arrivé devant la mine ciblée va se placer sur l'entrée, prendre acte des informations relative à la mine, puis spontanément s'interroger sur son activité et sa destination ;
- Un **vigile** va appréhender le déviant qu'il ciblait, puis spontanément se diriger vers un autre déviant s'il en connait un (sinon, il s'interroge sur son activité et sa destination) ;
- Un approvisionneur, après avoir soulagé un nain assoiffé, va spontanément se rendre à la réserve de bière s'il estime la quantité qu'il transporte trop basse. Sinon, il se rend vers un autre nain assoiffé. Et s'il n'en connait aucun, il va s'interroger sur son activité et sa destination ;
- Un **mineur** arrivé devant sa mine entre dans ladite mine.

[13] Les études menées sur les rapports nains-bières suggèrent qu'il s'agit d'hectolitres de bière

Structure logicielle adoptée

Figure XII : Deux nains se croisent sur un pont, capture d'écran

Notre structure logicielle est tout d'abord structurée en quatre parties :

1. Caméras + script Camera Behaviour ;
2. UI + script UI Behaviour ;
3. Lumière ;
4. Environnement de jeu.

Ce dernier (le monde) comprend :

1. les décors (bâtiments, volcan et lave, etc)
2. les routes
3. les mines et leurs entrées + script Mine Behaviour
4. les nains créés à partir du préfab dwarf

Nos nains sont associés aux scripts Dwarf Behaviour et Dwarf Memory (avec les classes imbriquées KnownDwarf et KnownMine) et à un animator, permettant d'animer le modèle du nain.

En parallèle dans les scripts :

1. le script Weight nous permet de formaliser les choix pondérés
2. la classe VariableStorage fait office de **fichier de configuration** pour ne pas avoir a modifier le code lors de la modification de valeurs

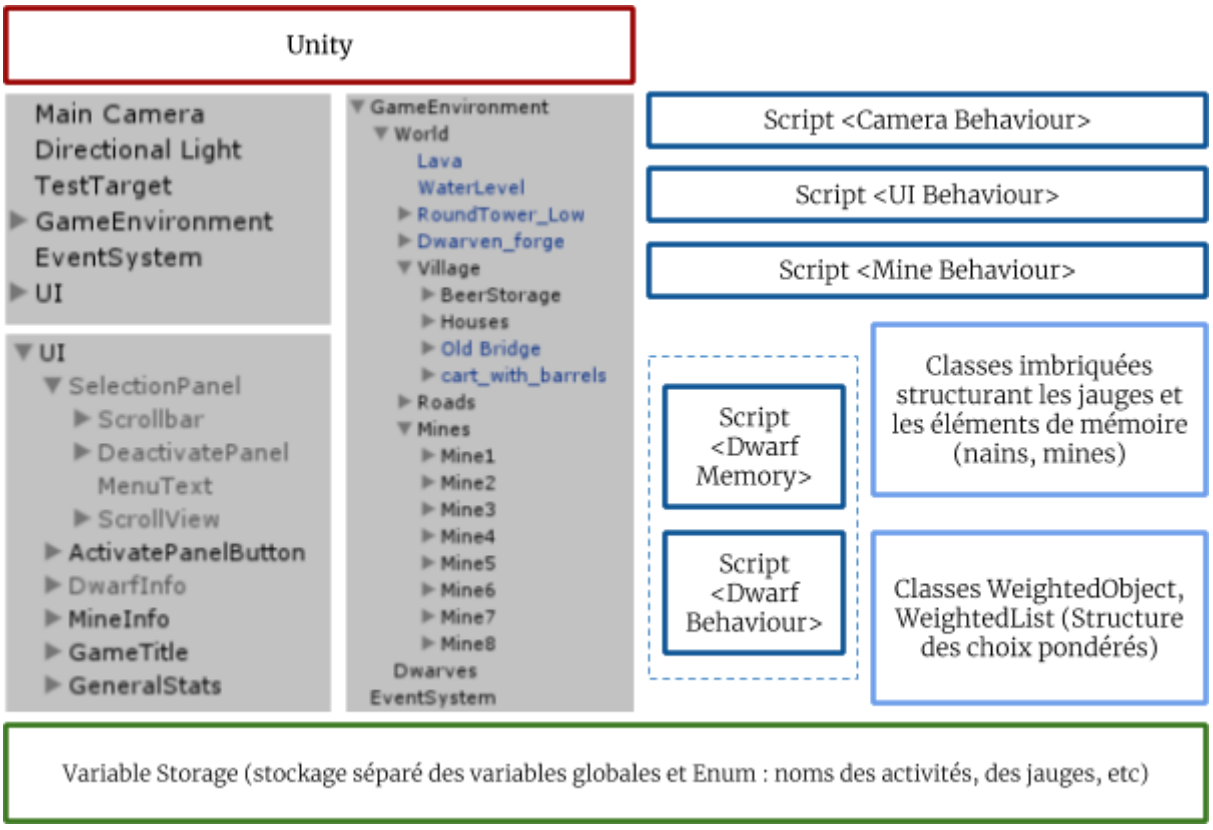

Figure XIII : Résumé de la structure logicielle adoptée (hiérarchie Unity et scripts C# associés)

Etude qualitative des résultats de simulations obtenus

L'utilisateur de notre simulation peut faire varier 3 paramètres, comme montré dans la partie description de l'UI. Nous avons donc réalisé des tests permettant de mesurer l'effet de ces paramètres sur les résultats de la simulation.

Nous avons réalisé 4 tests différents. Chaque test contient 10 essais, durant chacun 10 minutes. Afin de gagner du temps, les tests ont été réalisé avec un TimeScale de 10, et chaque essai a donc duré une minute. Les tests ont été faits avec les paramètres suivants :

- Test 1 : Nombre de nains = 5, empan mémoire = 180
- Test 2 : Nombre de nains = 40, empan mémoire = 180
- Test 3 : Nombre de nains = 5, empan mémoire = 60
- Test 4 : Nombre de nains = 40, empan mémoire = 60

Pour chacun de ces tests, nous avons récupéré au bout des 10 minutes la quantité totale de granite minée par les nains au cours de la simulation, ainsi que la quantité totale de bières bue par les nains. On obtient les statistiques descriptives suivantes :

Test n°	**Granite miné**		**Bière bue**	
	Moyenne	**Ecart Type**	**Moyenne**	**Ecart Type**
1 : 5 nains, 180s mémoire	2984	327	743	77
2 : 40 nains, 180s mémoire	2867	1021	3477	762
3 : 5 nains, 60s mémoire	2692	565	705	62
4 : 40 nains, 60s mémoire	2643.5	594	3818	311

On peut observer dans ces statistiques que la quantité de granite moyenne minée au bout de 10 minutes de simulation ne varie que très peu avec le nombre de nains dans la simulation. La quantité de bière bue par contre augmente fortement avec un plus grand nombre de nains. Les effets de la mémoire semblent assez faibles, les deux simulations avec un paramètre d'empan mémoire plus faible n'ayant que faiblement moins miné que les deux autres.

Il parait cependant surprenant que les nains en plus grand nombre ne soient pas capables de miner une véritablement plus grande quantité de granite. Face à ce constat, nous avons émis l'hypothèse que l'accélération du temps avait un impact négatif sur les comportements. Nous

avons donc relancé le test 2, mais avec un TimeScale de 5, et avons obtenu les résultats suivants :

Test n°	Granite miné		Bière bue	
	Moyenne	Ecart Type	Moyenne	Ecart Type
2 bis	4246	564	5308	341

La quantité de granite miné ainsi que la quantité de bière bue pour les mêmes paramètres augmentent donc de manière significative. On observe de plus que l'écart type des deux mesures a fortement baissé, indiquant une meilleure régularité des résultats des tests. L'échelle de temps modifie donc les comportements des nains, et rend plus imprévisibles les résultats.

De façon moins formelle, nous avons de plus pu faire les observations suivantes :

- Très peu de granite est miné en début de simulation : les nains étant tous occupés dans un premier temps à explorer la carte, aucun ne décide de devenir mineur.
- La mine numéro 1 est surexploitée : les nains la vident dès que de le granite y apparaît, et y rentrent même lorsqu'elle est vide. Cela s'explique par le pathfinding : en effet, la mine se trouvant à l'angle d'une falaise, tout les nains devant se rendre de l'autre côté de ladite falaise (ou revenant de la falaise vers le village) passent devant cette mine. Tous sont alors tentés d'explorer cette mine et/ou d'y entrer.
- De part le même problème de pathfinding, la zone entre le village et la mine numéro 1 est très fréquentée par les nains se dirigeant de l'autre côté de la falaise. La majorité des interactions (échanges d'informations, approvisionnement en bière) se déroulent donc dans cette zone - ce d'autant plus que la proximité avec le village entraîne la présence de nombreux approvisionneurs.

Figure XIV : Grand nombre de nains exploitant ou cherchant à exploiter la mine n°1, capture d'écran

Conclusion

Un tel projet s'inscrit parfaitement dans la formation d'ingénieur cogniticien, tant par ses apports techniques (la modélisation 3D est un véritable enjeu des modes d'interaction de demain tels que la réalité augmentée) que pour son aspect très libre qui nous a permis de nous familiariser avec Unity 3D dans un cadre nous confrontant à la fois à la conception, à la production de code et à l'analyse.

Par ailleurs, cette analyse de l'émergence des comportements résultant de la dynamique des interactions de nos nains - outre la satisfaction issue de notre aptitude à peupler un environnement 3D d'agents autonomes - nous a incontestablement permis de comprendre les enjeux de telles simulations et les nombreuses difficultés inhérentes à ce type de modélisation.

Plusieurs points de notre projet auraient pus être améliorés, et quelques compromis ont été faits. Par exemple, une erreur demeure dans nos fonctions de recherche de destination : celles-ci peuvent, dans des situations particulières, générer une exception StackOverFlow. Nous n'avons pas réussi à trouver l'origine de ce problème. Nous l'avons contourné en encapsulant l'exception dans un bloc Try / Catch. Lorsque l'exception apparaît, la destination est définie par défaut sur l'origine du monde. Une exception StackOverflow étant cependant assez lourde en mémoire, son apparition fait parfois s'arréter la simulation pendant une seconde. Une fois l'exception attrapée la simulation reprend sans problème.

Un autre défaut important de notre simulation réside dans le comportement des nains : en effet, comme montré dans les tests, un plus grand nombre de nains n'implique pas véritablement une plus grande efficacité de minage. Quoique l'échelle de temps semble en être une cause, il serait certainement possible d'améliorer la performance des agents en ajoutant ou modifiant certaines heuristiques.