TP 1
Interaction multi-Agents

Aurélien CHEMIER, Arnaud DUHAMEL, Maëlyss FARGES

Université Lyon 1 - 2014

Table des matières

1 Introduction

Le but de ce TP est de gérer des interactions entre plusieurs Agents.

Pour cela les Agents sont placés sur une grille de taille n*n définissable par l'utilisateur et doivent atteindre une case "cible". L'objectif de l'Agent est donc d'atteindre la cible en prenant en compte les déplacements et les positions des autres Agents.

2 Implementation

Pour réaliser ce TP, nous avons utilisé Unity 4.3.2f1 qui permet de faire des représentations graphiques 3D très simplement.

Chaque Agent est un thread qui agit en parallèle des autres Agents. Chaque Agent peut être doté d'une vitesse propre. L'objectif de chaque Agent peut être n'importe quel Object de l'environnement. L'Agent peut ansi courir après un autre Agent par exemple.

2.1 V1

C'est la premiere version développée. L'Agent se déplace sur la case la plus proche de son objectif. Si elle est occupée, il envoie un message à l'occupant lui demandant de se déplacer, mais aucun Agent ne gère les messages reçus.

2.2 V2

L'Agent calcule toutes les cases à parcourir en utilisant l'algorithme A*.
Ensuite à chaque étape, le mouvement de l'Agent est défini selon les cases adjacentes disponibles et les messages reçus. L'Agent avance vers la première case disponible suivant cette ordre :

1. L'Agent prend le prochaine case calculée par A*. Si la case est occupée, il envoie un message demandant à l'occupant de se déplacer.
2. L'Agent ne bouge pas sauf s'il a reçu un message lui demandant de se déplacer.
3. Si aucun des deux cas de figure ci-dessus ne fonctionne, l'Agent se déplace donc de façon aléatoire sur l'une des cases adjacentes libre et non réservée.

2.3 V3

Après le calcul de A*, avant même de se déplacer, les Agents vont négocier en réservant leurs chemins grâce à un contrôleur global.

Chaque réservation peut engendrer un conflit sur une case. Dans ces cas là, les Agents comparent la longueur du détour à faire pour chacun d'entre eux en ne passant pas par la case contestée. L'Agent qui a le détour le moins long laisse la case à l'autre Agent.

Si la case contestée correspond à l'arrivée d'un Agent, celui laissera passer l'autre avant de retourner à sa place.

Cette algorithme est en place, mais n'a pas pu être débuggé.

3 Score

Une fois arrivé, chaque Agent obtient un score calculé en fonction de sa performance. Un score général est également calculé.

3.1 Score individuel

Le score individuel est calculé de la façon suivante : $\frac{nombredepasminimun}{nombredepasfait} * 100$

Le nombre de pas minimum est déterminé par le premier A* effectué en début de parcours. Ce nombre de pas minimum ne prend donc De plus, on peut déplacer les Agents pendant l'exécution du programme et les mettre en pause.pas en compte toutes les interactions à faire avec les autres Agents. Il s'agit du nombre de pas à faire si l'Agent n'était jamais bloqué.

3.2 Score général

Le score général est calculé de la façon suivante : $\sum scoresindividuels$

4 Exemples

4.1 Exemple Simple

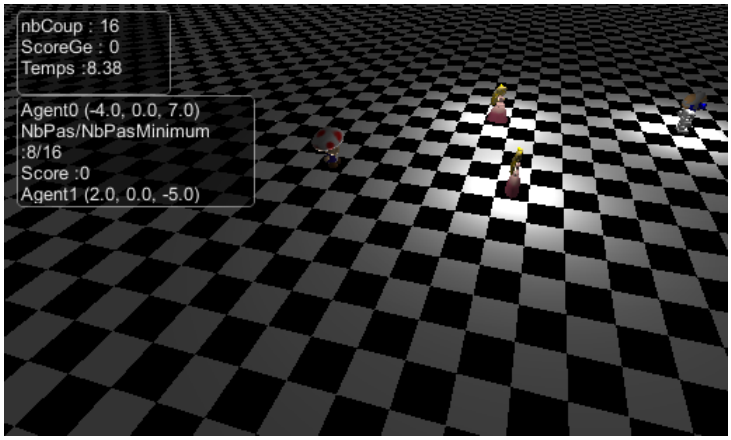

FIGURE 1 – Position de départ

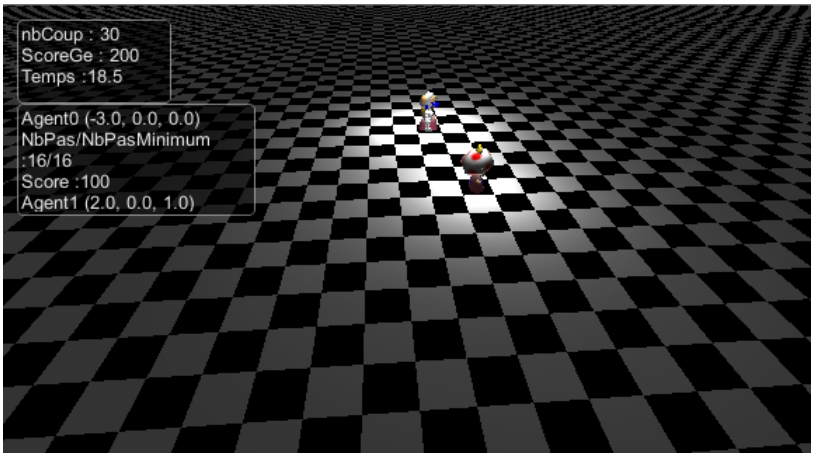

FIGURE 2 – Position d'arrivée

Sur cette première image, Toad et Mario sont deux Agents dont leurs objectifs est de rencontrer Peach. Ici, l'arène est volontairement immense (100*10) pour prouver que l'algorithme tient sur une grande grille. A chaque itération, Mario et Toad vont avancer vers leurs objectif respectif sans encombre. Dans la figure 2, les deux Agents ont complété leurs objectifs.

En haut à gauche est affiché le nombre de coup totaux, le score général et le temps depuis le lancement de l'application. En dessous est affiché pour chaque Agent : sa position, son nombre de

pas actuels / le nombre de pas minimum calculé par A* sans tenir compte des obstacles Agent en début de parcours ainsi que son score.

Dans notre premier exemple, nos deux Agents atteignent leur objectif sans encombre et obtiennent 100 points chacun pour un score général de 200 points en 30 coups et 18.5 secondes (l'imprime-écran n'a pas eu lieu au moment précis de la rencontre, sinon le temps aurai été de 16 secondes : 1 seconde par pas).

4.2 Exemple Échiquier

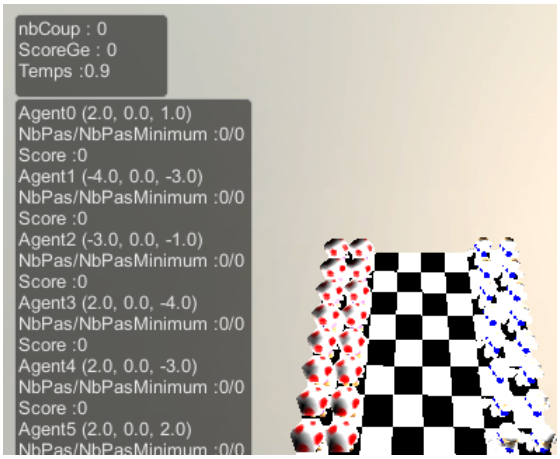

FIGURE 3 – Échiquier initial

Nous simulons ici un échiquier où les Toad et les Mario vont échanger leur place. Cette grande simulation est l'opportunité de montrer de très nombreuses négociations serrées. Dans la figure 3 qui se situe à t = 0 secondes, nous avons de chaque coté les deux équipes prêtent à échanger leurs places.

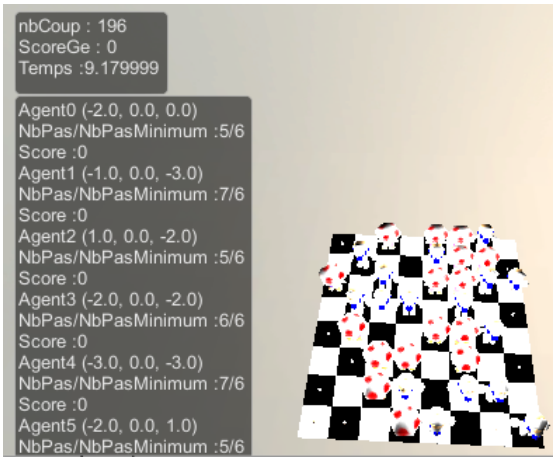

FIGURE 4 – Échiquier en cours d'échange

À t = 9 secondes , les deux camps sont en plein échange de place au milieu de l'arène (figure 4). Chaque Agent doit négocier avec son voisin pour avancer, tout en gérant les messages reçus de ceux qui ont réservé des places adjacentes.

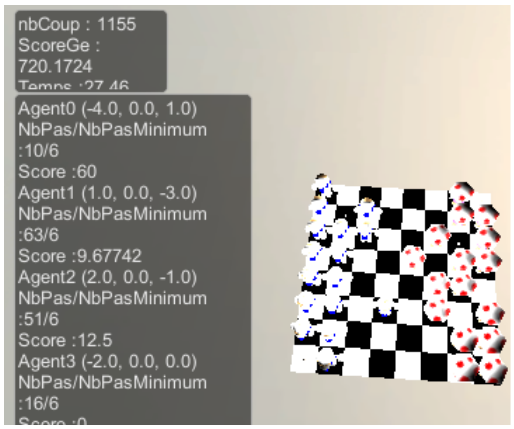

FIGURE 5 – Échiquier presque terminé

Après un peu plus de 20 secondes de négociation les Toads et les Marios on échangé leurs places (figure 5). Viens la deuxième épreuve : le placement de l'autre coté de la grille. En effet, si un Agent se place correctement il risque de bloquer le passage à un autre Agent. En se déplaçant, ce premier Agent risque à nouveau de bloquer un autre Agent et ainsi de suite. Si les Agents sont bloqués, ceux-ci se déplacent aléatoirement autour d'eux, ainsi au bout d'un certain temps il est certain que chacun trouvera sa place.

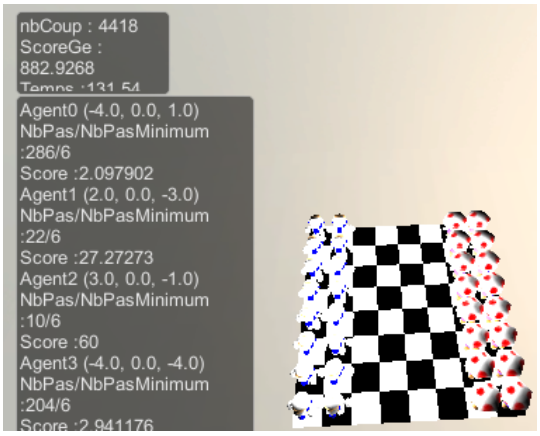

FIGURE 6 – Échiquier à la fin

A t = 130 secondes, tout les Agents ont échangé leurs positions. L'expérience est un succès ! On constate que quelques Agents ont trouvé leur emplacement en peu de déplacements (Agent 1 avec seulement 22 déplacements) tandis que d'autre ont dû être énormément sollicité (Agent 0 avec 286 déplacements). Les Agents peu sollicités doivent être ceux disposé aux coins de l'arène, tandis que les autres sont ceux posé en plein milieu, source de beaucoup de gènes pour les autres Agents et sont donc ceux qui ont besoin de beaucoup de négociations.

4.3 Autres exemples

Tout les exemples présenté et bien d'autres sont des scènes chargeable depuis Unity.

- Scénario 1 met en scène plusieurs Agents avec un même objectif.
- Scénario 2 oblige un Agent à éviter un mur pour passer.
- Scénario 3 met en scène un unique chemin que deux Agents doivent emprunter. Il vont devoir négocier pour savoir lequel des deux passera en premier.
- Scénario 4 met en scène un labyrinthe ou plusieurs Agents vont passer les uns après les autres pour rejoindre leur objectif personnel.

5 Conclusion

Ce système simple d'Agents semble marcher pour tous les cas de figure. Les nombreux paramètre des Agents sont facilement modifiable via l'interface d'Unity (figure 7). On peut ainsi modifier la vitesse d'un Agent et son objectif par exemple grâce à la fenêtre Inspector de droite.

On peut aussi ajouter des "murs" en rendant un Agent "sourd" au messages en cochant Stategie Agent Bloquant dans l'Inspector. Un tel Agent ne se déplacera pas en fonction des autres Agents, mais au contraire leurs répondra qu'il ne compte pas bouger, les forçant à trouver un autre chemin.

De plus, on peut déplacer les Agents pendant l'exécution du programme et les mettre en pause.

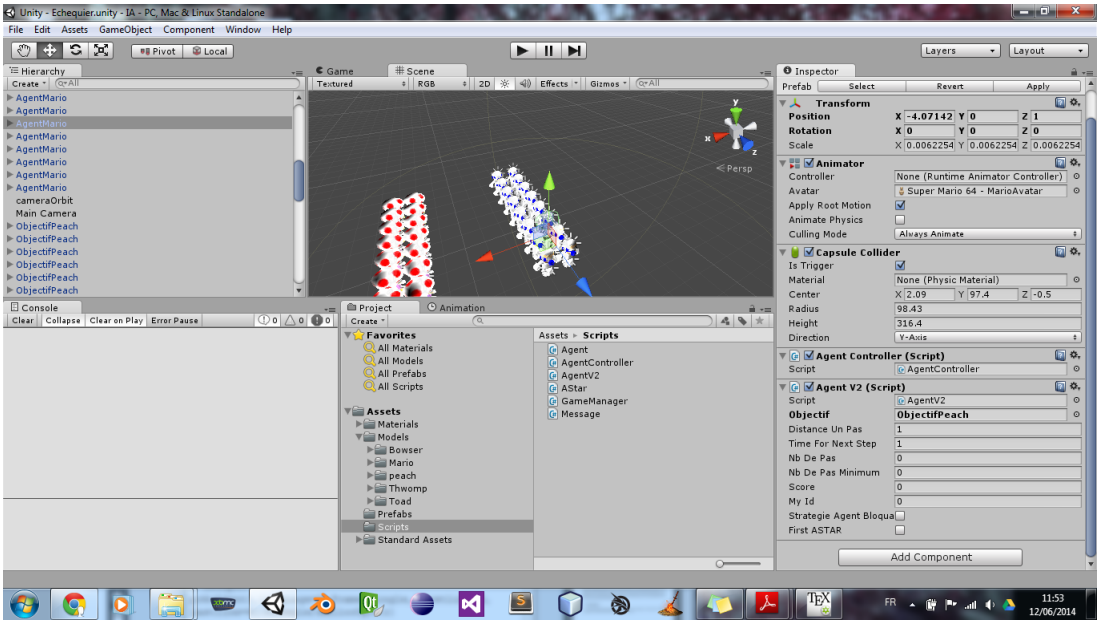

FIGURE 7 – Interface Unity

Même si l'Agent V3 n'a pas pu être terminé, cette implémentation offre d'ores et déjà une interface intuitive et une IA simple et efficace dans la résolution de chemin Multi-Agents.