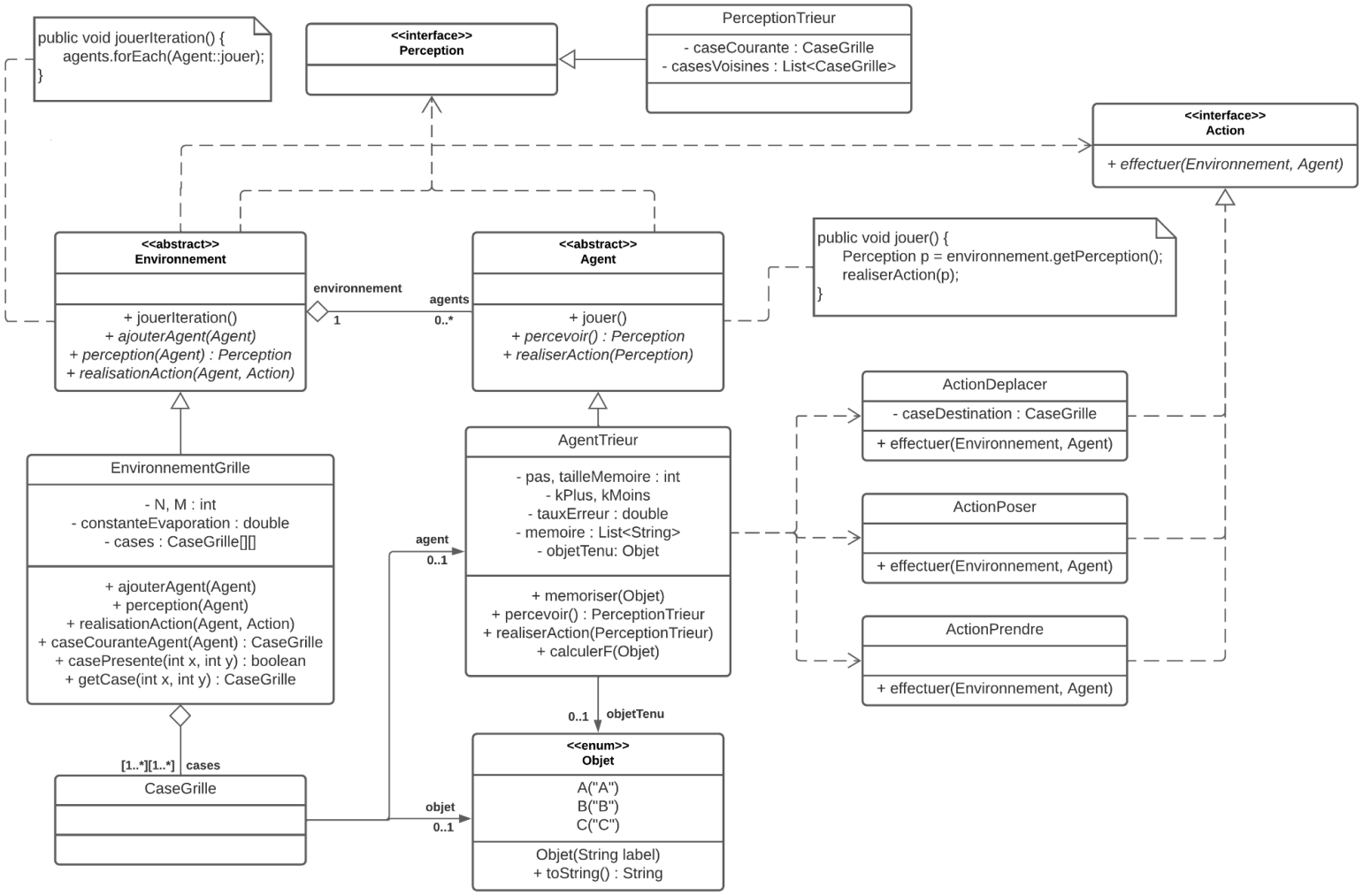
GROSJEAN Justin & Rémi PAILHAREY

Compte rendu TP2 SMA

Première version

Structure

Diagramme de classes UML



Lors du premier TP de simulation de blocs pouvant s'empiler, nous avions pris à cœur le fait de rendre notre code modulaire. Pour cela, nous avons notamment conçu une couche d'abstraction constituée des classes abstraites Environnement et Agent, ainsi que des interfaces Action et Perception. Nous avons fait en sorte que cette conception puisse encadrer correctement le développement de telles simulations d'agents, et elle a justement fait ses preuves puisque nous avons pu la réutiliser intégralement pour ce TP.

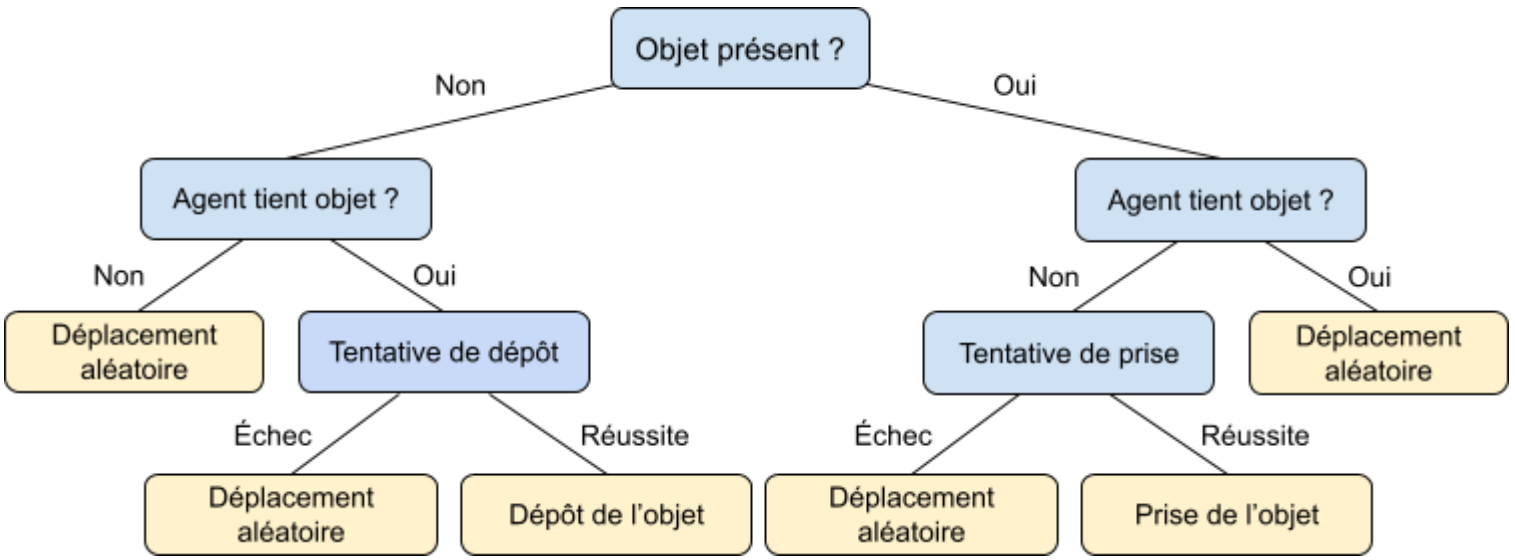
Nous avons également choisi le paterne stratégie pour l'implémentation des actions des agents, car ce dernier permet de décrire simplement le comportement d'une action sans impacter les classes des agents et de l'environnement, en plus de faciliter la conception de nouvelles actions. L'agent peut alors demander à l'environnement l'exécution d'une action précise sans que l'environnement n'ait à se soucier de l'implémentation de l'action en question.

Perceptions et actions des agents

La perception d'un agent consiste en l'ensemble des cases présentes dans son voisinage, ainsi que la case où se situe actuellement l'agent. Ces cases permettent alors à l'agent de prendre connaissance des autres agents et objets qui l'entourent, ainsi que de savoir s'il peut prendre ou poser un objet sur sa case courante. Après chaque perception, un agent enregistre dans sa mémoire (à taille limitée) l'objet présent sur sa case courante, ou alors à défaut l'absence d'objet.

De plus, chaque agent dispose d'un panel de trois actions différentes. Il s'agit de pouvoir prendre ou poser un objet, ou bien de se déplacer vers une case donnée.

Arbre de décision



Résultats et impact des paramètres

Nous nous sommes tout d'abord intéressés aux résultats produits avec les paramètres par défaut donnés dans l'énoncé, sans taux d'erreur et avec un pas de 1. Au bout de 30 000 itérations, des agrégats légèrement dispersés semblent déjà se former. Si l'on poursuit la simulation jusqu'à 100 000 itérations (voir Figure 1.1), ces agrégats deviennent plus prononcés et uniformes. Cet effet peut être impacté par la taille de la mémoire des agents, puisqu'un agent doté de moins de mémoire se souviendra moins bien des types d'objets des agrégats qu'il a traversé récemment et qui sont donc plus ou moins proches de lui.

Après 500 000 itérations les agrégats deviennent massifs et peu nombreux (voir Figure 1.2), ce qui est logique puisque les agents ont plus de temps pour les rassembler.

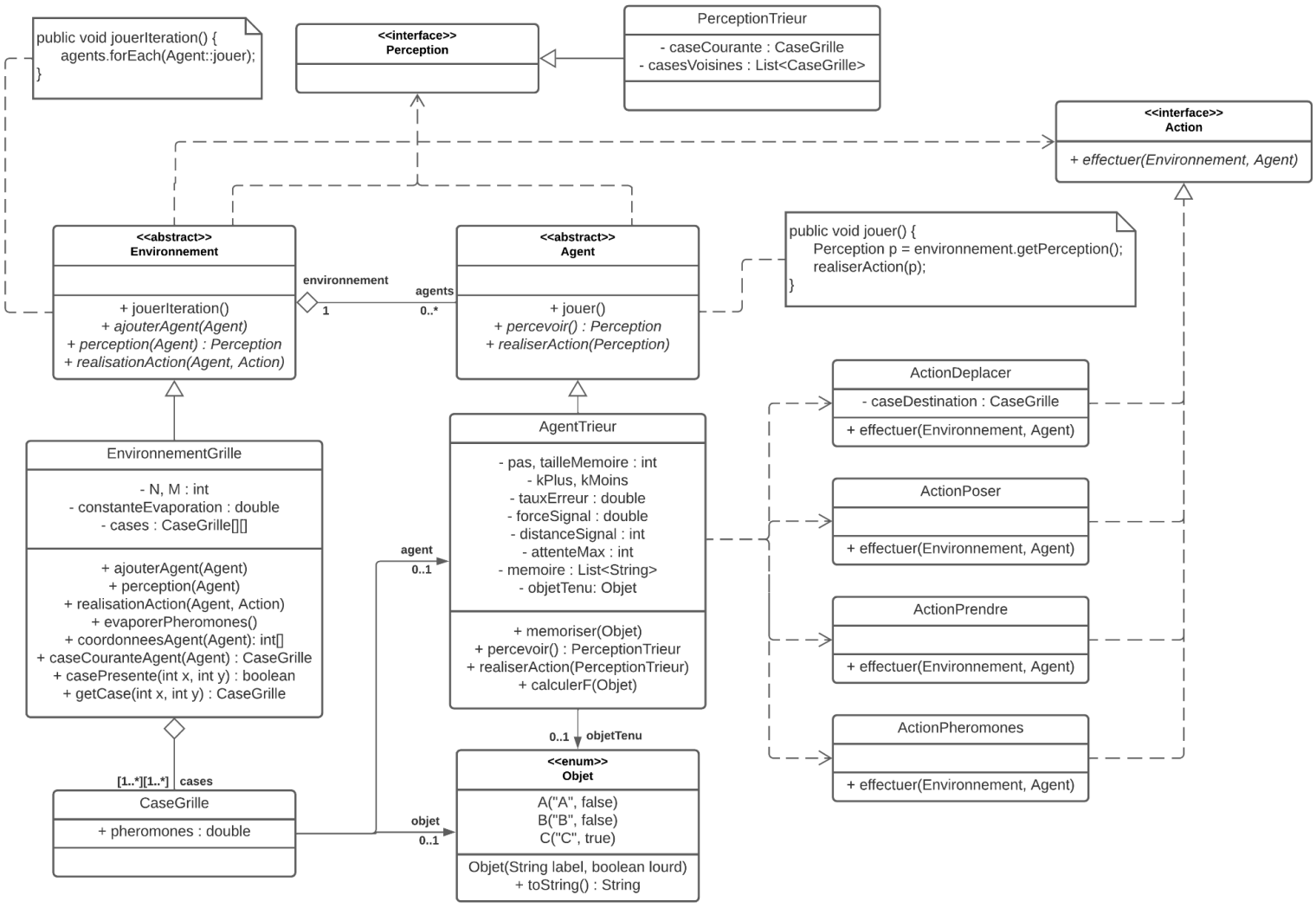
Si l'on réduit le nombre d'agents à 10, ou que la constante k^+ est plus grande que k^-, on peut constater que les agrégats sont plus dispersés puisque dans le premier cas les agents sont moins nombreux pour trier, ou dans le second cas ces derniers ont une plus forte tendance à prendre les objets plutôt qu'à les déposer. Un exemple de cet effet est visible sur la Figure 1.3.

Enfin, l'implémentation du taux d'erreur produit un effet intéressant : si ce dernier augmente trop, les agents ne sont plus capables de distinguer les objets entre eux. Des tas vont toujours se former, mais en mélangeant de façon plus ou moins homogène les objets de type A et B entre eux (voir Figure 1.4).

Deuxième version

Nouveautés du modèle

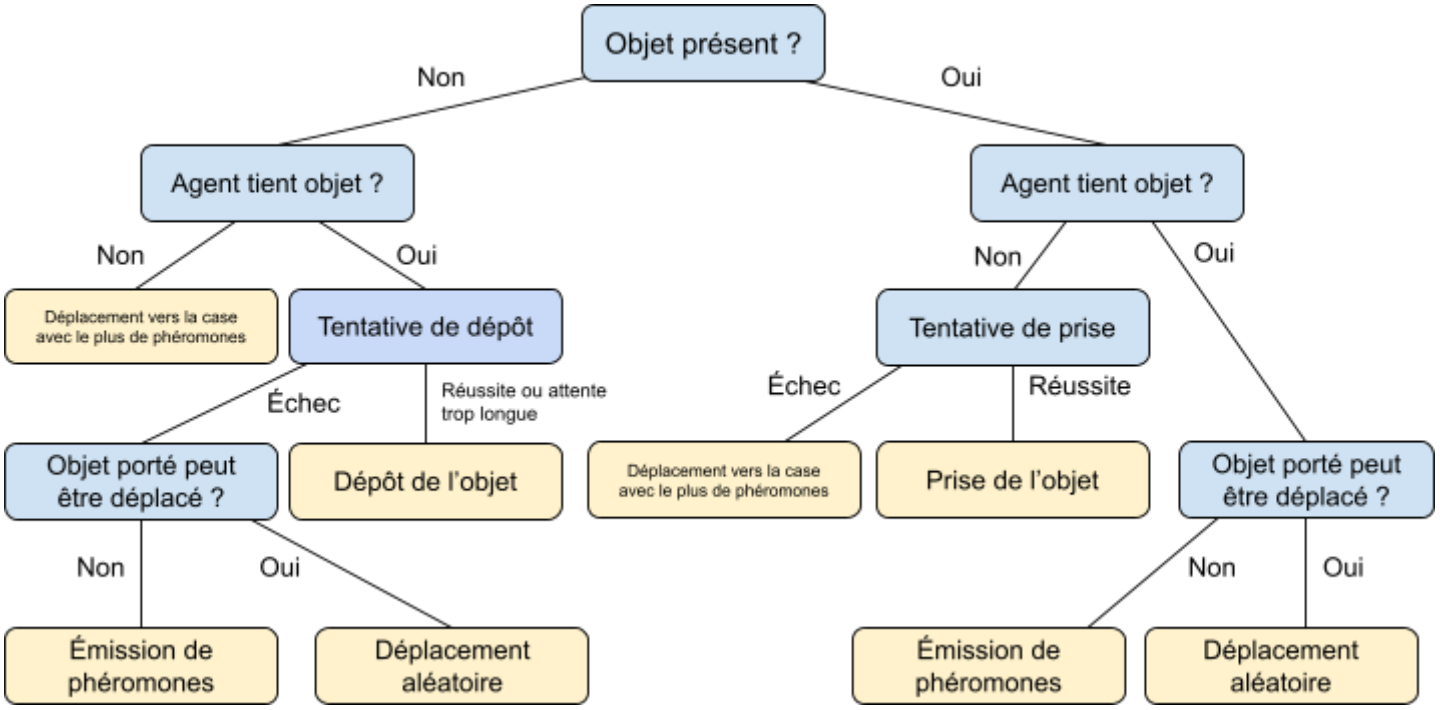
Mise à jour du diagramme de classes UML



On peut constater sur ce nouveau diagramme que les cases de l'environnement peuvent désormais contenir plusieurs agents. Nous avons fait ce choix afin de permettre à plusieurs agents d'aller sur une même case et ainsi contribuer ensemble au transport d'un objet. Les cases sont également dotée d'une nouvelle valeur représentant la quantité de phéromones déposée par les agents que ces derniers émettent lorsqu'ils ont besoin d'aide. Ces phéromones s'évaporent d'un montant fixe à chaque itération.

De la même manière, les agents disposent d'une nouvelle action leur permettant de déposer des phéromones dans une certaine zone de leur voisinage.

Modifications de l'arbre de décision



Lorsque l'agent souhaite se déplacer vers la case voisine ayant le plus de phéromones, si plusieurs cases voisines ont le même montant maximal, l'agent choisit une de ces cases aléatoirement.

Un objet porté peut être déplacé s'il n'est pas lourd. Si l'objet porté est lourd, ce dernier pourra être déplacé seulement si d'autres agents (au moins un) sont présents sur la même case que le porteur.

À chaque fois que l'agent porte un objet lourd et émet des phéromones pour potentiellement attirer d'autres agents, ce dernier reste sur place. Si l'agent reste sur place plusieurs tours sans aide, ce dernier décidera au bout d'un moment de poser l'objet porté quoiqu'il arrive pour pouvoir se débloquer (sauf si un objet est déjà présent sur sa case courante, auquel cas l'agent continuera d'émettre des phéromones car une case ne peut contenir qu'un seul objet).

Résultats et impact des paramètres

Pour tester cette deuxième version, nous avons à nouveau utilisé les paramètres par défaut proposés dans l'énoncé (notamment la portée des signaux émis fixée à 2), et nous avons ajouté 200 objets de type C. Nous avons également choisi de fixer la taille de la mémoire de nos agents à 20 afin de contrebalancer l'attente qu'un agent peut subir lorsqu'il tente de porter un objet de type C. Enfin, nous avons fait en sorte qu'un agent attende de l'aide jusqu'à 10 tours maximum avant d'abandonner.

De la même manière que pour la première version, 30 000 itérations suffisent pour commencer à observer de petits agrégats. Cependant, ces derniers sont beaucoup plus dispersés, notamment pour les objets de type C qui sont naturellement plus difficiles à déplacer. Au bout de 100 000 itérations, on peut obtenir des résultats plus satisfaisants, comme le montre bien la Figure 2.1.

Toutefois, cet environnement de taille 50 par 50 est assez grand, ce qui diminue les chances d'un agent d'être trouvé et aidé lorsqu'il tente de porter un objet lourd. Cela peut être contrebalancé en augmentant la distance et la puissance du signal émis comme le montre la Figure 2.2. Il est également possible d'augmenter le temps d'attente maximum des agents, mais cela risque de réduire leur efficacité.

En prenant en considération ces informations et en augmentant le nombre d'agents et d'itérations de la simulation, il est possible d'obtenir de bons résultats avec des agrégats très nets, comme le montre la Figure 2.3.

Conclusion et pistes d'amélioration

Ce projet a été pour nous l'occasion d'apprendre à implémenter de façon concrète un système multi agents, et nous en avons également profité pour mettre à l'épreuve nos capacités en programmation objet, notamment en mettant en avant quelques éléments appris l'année dernière en cours d'ISI3.

Avec plus de temps, nous aurions aimé mettre en place une interface graphique avec Swing ou JavaFX afin de rendre notre projet plus simple et agréable à visualiser et à déboguer. Nous aurions également pu tenter d'implémenter un pattern État pour nos agents, afin de gérer plus proprement leurs décisions. Concernant la prise de décision de nos agents, nous pourrions également remplacer le système d'attente basé sur un nombre fixe de tours par une probabilité d'abandon croissante en fonction du nombre de tours écoulés. Un agent aurait ainsi plus de chance d'abandonner le déplacement d'un objet lourd s'il a attendu longtemps, plutôt que d'attendre jusqu'à un nombre fixe de tours.

Enfin, nous pourrions envisager des améliorations dans la qualité de nos tests et dans l'évaluation de nos résultats, tout d'abord en implémentant une gestion parallélisée de nos agents pour accélérer la simulation, mais aussi en réfléchissant à la conception d'un indice de qualité sur l'état de l'environnement afin de pouvoir les comparer de façon plus objective. Cet indice pourrait par exemple être basé sur le nombre d'agrégats présents, leur dispersion et leur homogénéité.

Annexes

GitHub

Les sources de notre projet sont disponibles sur GitHub au lien suivant :
https://github.com/Justin-Gr/TP2-SMA

Vous y trouverez la version 1 et 2 de ce projet, avec une classe Main de démonstration pour chacune.

Première version

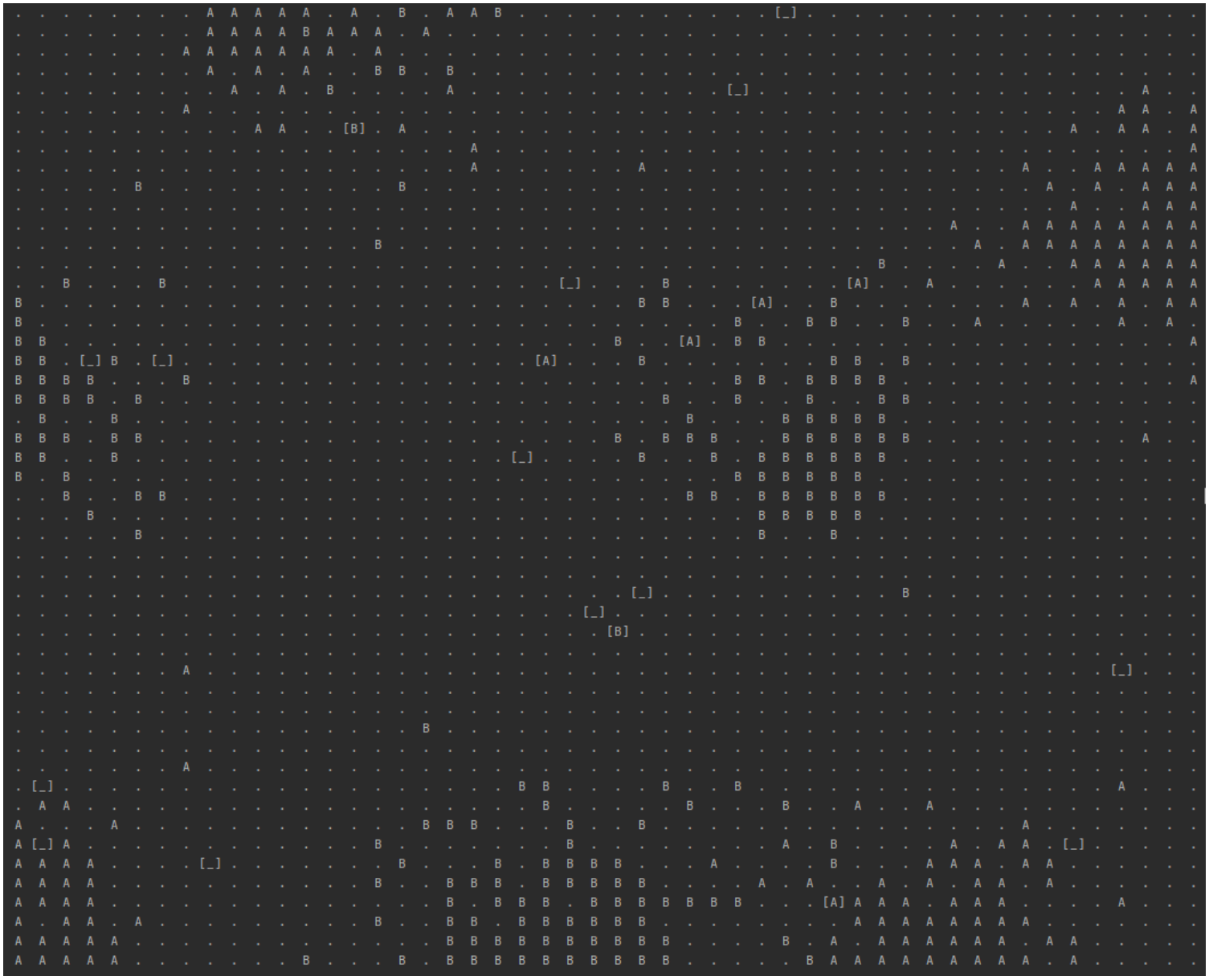


Figure 1.1 : paramètres par défaut, sans taux d'erreur, 100 000 itérations

Figure 1.2 : paramètres par défaut, sans taux d'erreur, 500 000 itérations

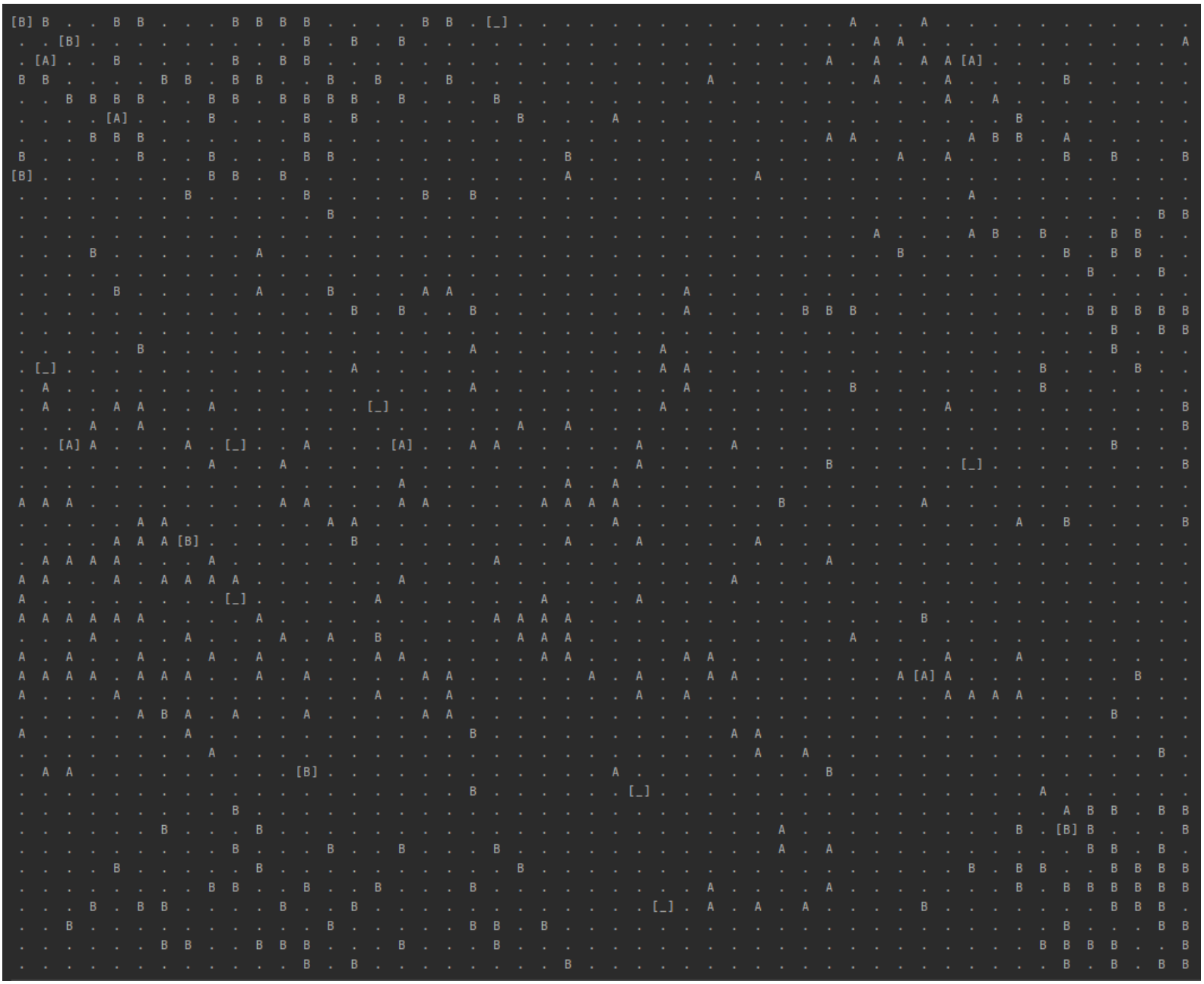


Figure 1.3 : 20 agents, k^+=0.8, k^-=0.2, 100 000 itérations

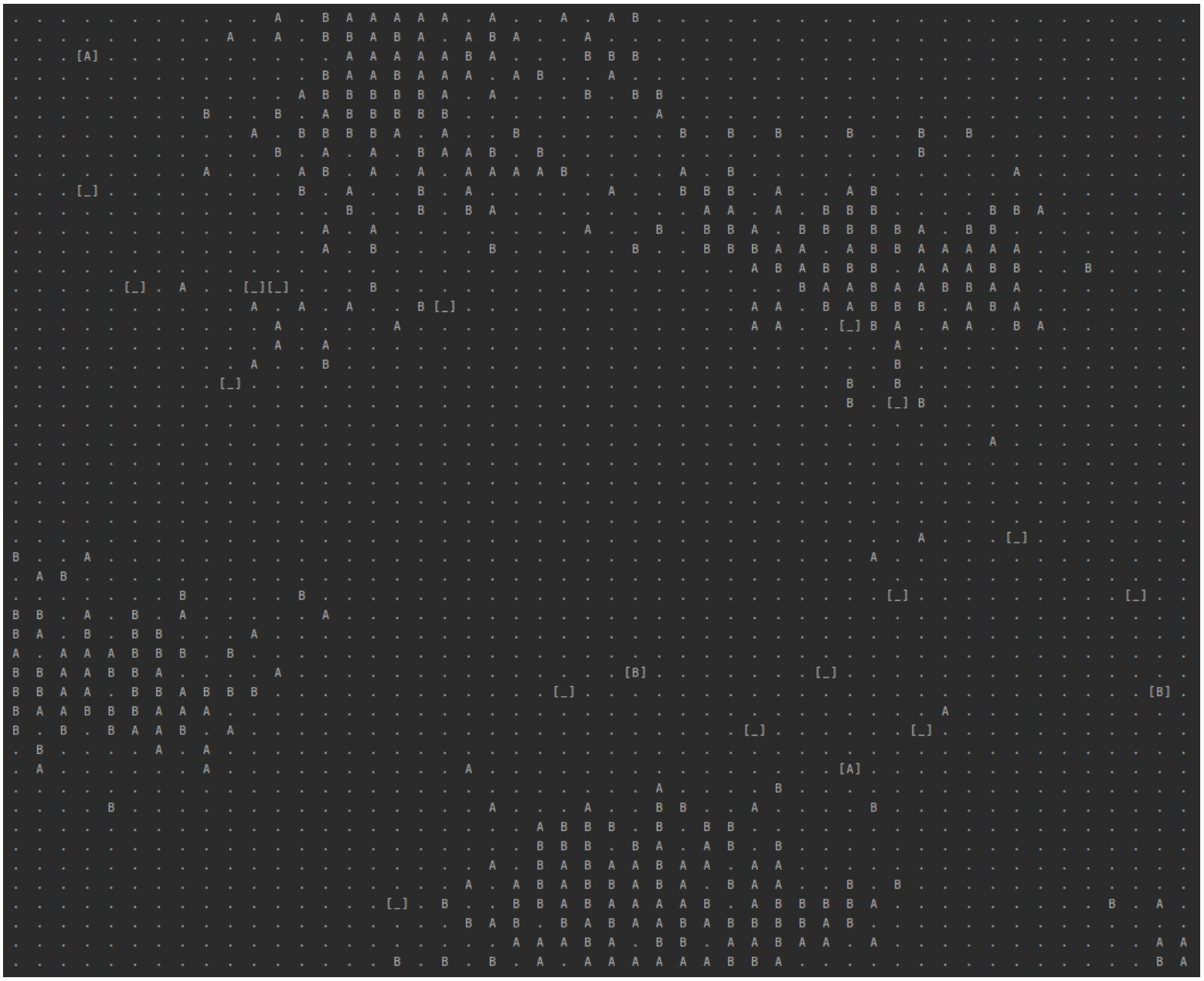


Figure 1.4 : paramètres par défaut, taux d'erreur à 70%, 100 000 itérations

Deuxième version

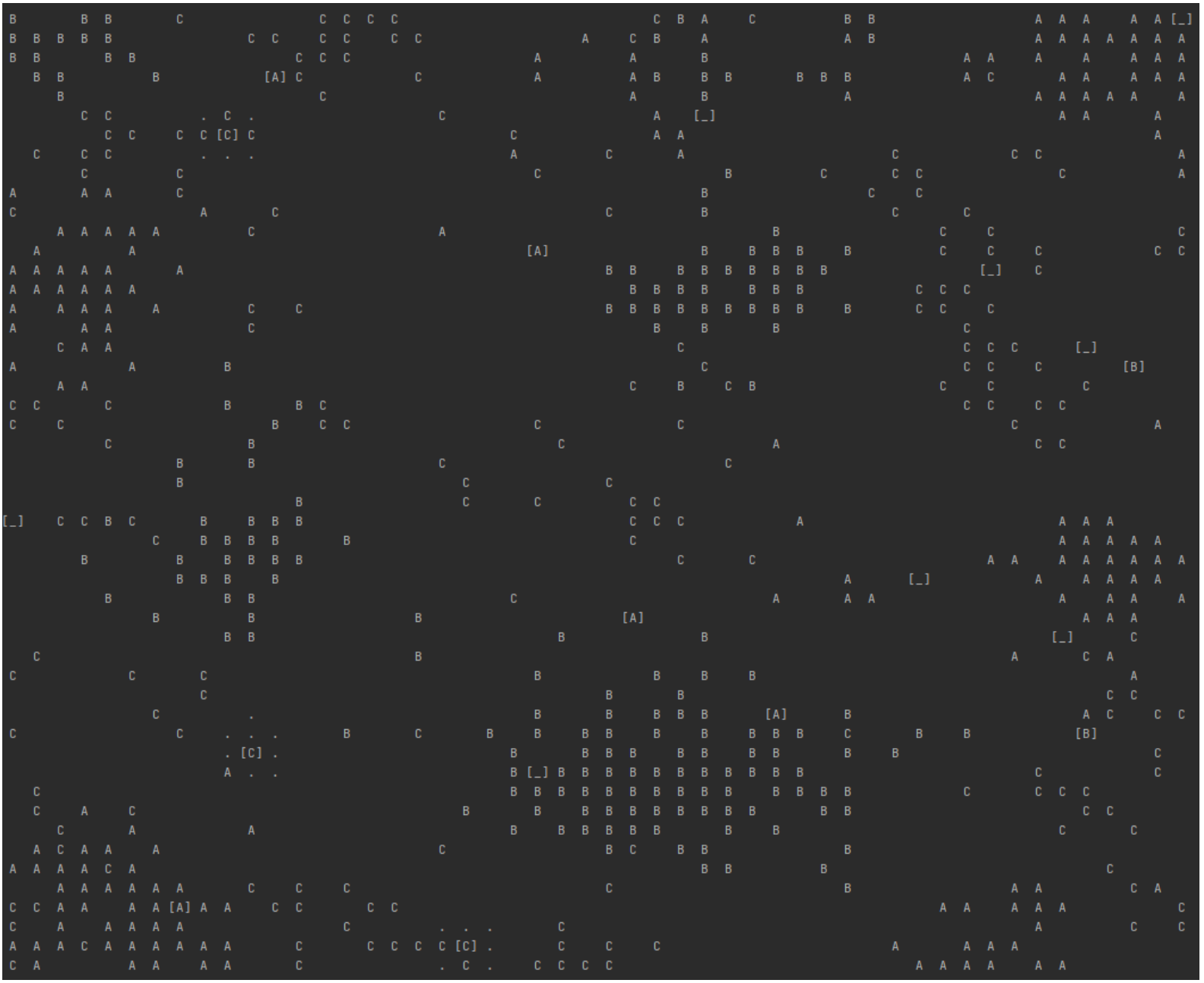


Figure 2.1 : paramètres par défaut, 100 000 itérations

NB : Les points présents sur la grille représentent la présence de phéromones

Figure 2.2 : paramètres par défaut, dS=3, puissance du signal à 6, constante d'évaporation à 0.5, 100 000 itérations

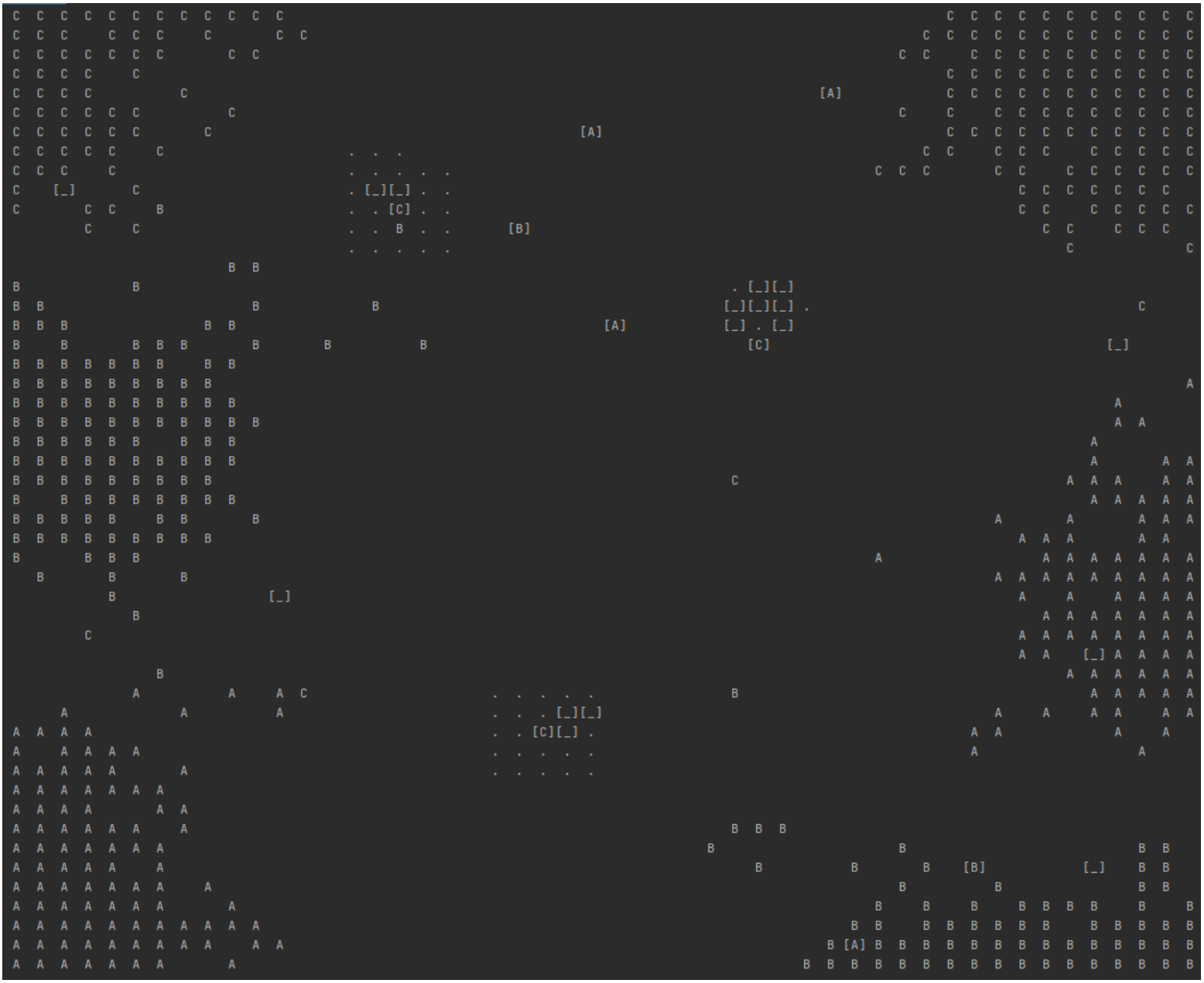


Figure 2.3 : 50 agents, dS=3, 1 000 000 d'itérations