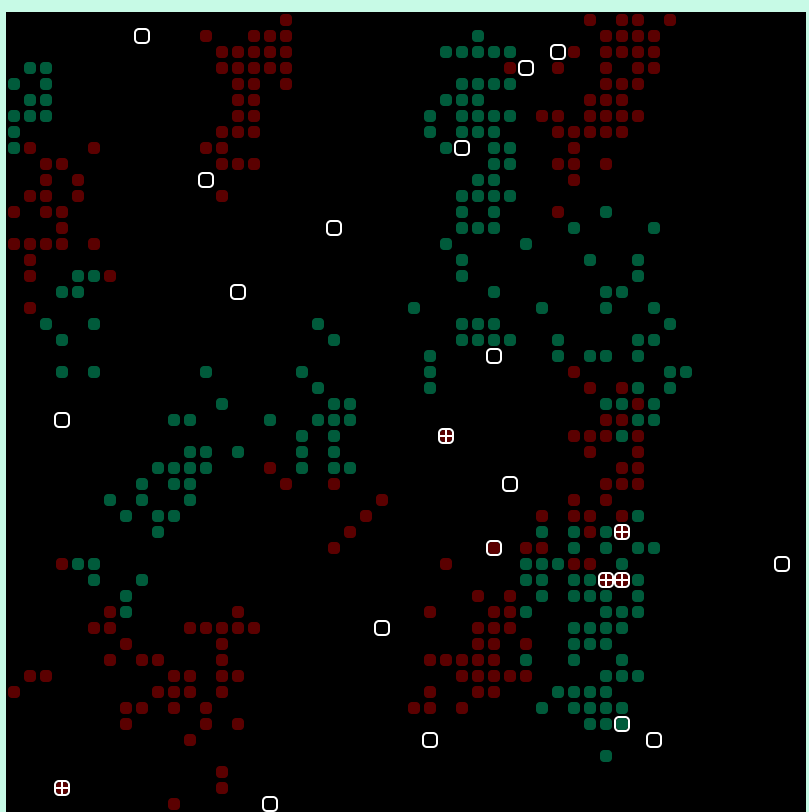
Systèmes Multi Agents TP2

Algorithmes de tri multi-agents autonomes



Le problème sur lequel nous allons travailler est le suivant :

Dans un environement sous forme d’une grille de taille N par M (ci-dessus 50 cases par 32 cases), nous allons répartir aléatoirement des objets, déclinés en deux variations (que nous allons appeler « Type A » et « Type B »).

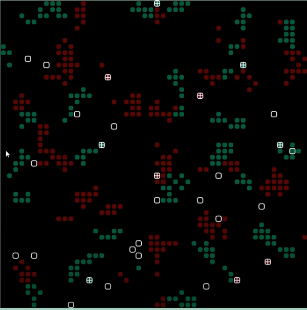
Nous souhaitons à l’aide d’agent capables de porter et de déposer des objets, regrouper géographiquement les objets. Les agents, sans communiquer entre eux, doivent former des tas d’objets, séparés par type (il ne peut y avoir qu’un objet par case).

Ci-dessus nous avons un exemple de bon résultat, avec des regroupements satisfaisants.

Les agents respectent des règles simples :

* S’ils se trouvent au dessus d’un objet (et qu’ils ne portent pas déjà un objet), ils vont essayer de le soulever suivant la probabilité suivante :
  + Pprise= (k+ /(k+ + f))2
  + k+ est une variable fixée arbitrairement
  + f correspond à la proportion d’objet du même type dans sa mémoire (si 15% des objets retenus sont du même type, alors f = 15/100)
* S’ils se trouvent au dessus d’une case libre en portant un objet, ils vont essayer de le poser suivant la probabilité suivante :
  + Pdépôt= (f /(k- + f))2
  + k- est une variable fixée arbitrairement
  + f correspond à la proportion d’objet du même type dans sa mémoire (si 15% des objets retenus sont du même type, alors f = 15/100)
* S’ils ne font aucune des deux actions si dessus, ils doivent se déplacer vers une case dans un voisinage bien défini. Les agents peuvent se déplacer jusqu’à ***i***cases de distance dans chaque direction.
* Ils vont ensuite regarder leur voisinage, et noter les différents types d’objets présents. La mémoire des agents est limitée, et ils ne peuvent retenir qu’un nombre fixé de cases. Quand la mémoire d’un agent est pleine, il oublie les cases parcourus les moins récentes pour mémoriser les nouvelles cases.
* Les agents ont une durée de vie limitée. Arrivé à la fin, ils disposeront de quelques mouvement supplémentaires pour déposer l’objet. Après avoir déposé leur objet (ou à la fin de ce temps impartis s’ils refusent de le poser), l’agent s’eteind.
* Le jeu s’arrête quand tous les agents sont eteind.

# Démonstration

Nous pouvons voir ici une démonstration de la simulation, sur une grille 50x50.  
30 agents ont pour tâche de ranger 400 objets (200 de chaque type).

<https://youtu.be/LYQmF8JwimU>

Le code est disponible sur ce dépôt github : https://github.com/BaptisteMagous/SMA-TP2.git

Pour executer le code, il suffit de lancer la classe **main.java** qui contien une simulation visuelle.  
Il est possible de changer les parametres de l’environement dans cette classe. Il est aussi possible de changer les parametres des agents en modifiant la classe **Agent.** Tous les résultats des simulations sont stockées dans le fichier **reports.csv**   
La classe **mainCollectData** contient les différents test sur les parametres présentés dans la suite (attention, l’execution peut prendre beaucoup de temps, mais nos résultats sont déjà présent dans le fichier de rapport).

# Quantification

Afin de mesurer l’impact de nos paramètres, nous devons définir une façon que quantifier l’ordonencement de nos objets. Nous avons décidé de compter pour chaque objets le nombre de voisins du même type. Nous appelerons cette somme le « résultat ». Afin de valoriser les grands tas, nous élévons au carré le nombre de voisins pour calculer le résultat. Ainsi, un objet entouré de 8 autres objets identiques rapportera plus de point que 4 paire d’objets.

# Collecte de données

Nous avons executé un ensemble de simulations avec ces paramètres :

Nombre d’agent : 10

Nombre de pas : 1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000

Taille de la mémoire : 5, 10, 15

K+ : 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35

K- : 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35

Distance de déplacement : 1 et 2

Distance de perception : 0, 1 et 2

Chaque combinaison de parametres possible a été testée, entre 10 et 50 fois.

# Résultat

Nous avons utilisé un tableur Excel pour afficher et analyser nos résultats. Le fichier est disponible avec le code.

Tout d’abord, nous allons étudier l’agent de base (déplacement de 1, et ne regarde que sa case pour la gestion de la mémoire).

## K+

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Moyenne de score | Nombre de pas |  |  |  |  |  |  |  |
| k+ | 1000 | 2000 | 4000 | 8000 | 16000 | 32000 | 64000 | 128000 |
| 0,05 | 1391,740952 | 1484,384762 | 1625,44381 | 1877,044146 | 2209,259048 | 2660,337143 | 3328,177143 | 4196,208545 |
| 0,1 | 1466,712381 | 1624,638095 | 1857,430476 | 2162,628788 | 2602,474286 | 3162,984762 | 3962,670476 | 5094,374765 |
| 0,15 | 1518,257143 | 1687,249524 | 1960,055238 | 2289,795802 | 2726,653333 | 3310,329524 | 4168,373333 | 5248,456744 |
| 0,2 | 1549,274286 | 1737,249524 | 2023,895238 | 2352,99334 | 2796,47619 | 3359,19619 | 4171,598095 | 5104,161353 |
| 0,25 | 1561,335238 | 1765,594286 | 2044,56381 | 2393,585413 | 2827,584762 | 3370,990476 | 4094,401905 | 5002,703633 |
| 0,3 | 1574,607619 | 1776,337143 | 2064,979048 | 2405,080416 | 2839,4 | 3302,946667 | 4021,288571 | 4830,058989 |
| 0,35000002 | 1571,952381 | 1781,737143 | 2067,462857 | 2403,377609 | 2808,56381 | 3281,569524 | 3930,813333 | 4783,39778 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Max. de score | Nombre de pas |  |  |  |  |  |  |  |
| k+ | 1000 | 2000 | 4000 | 8000 | 16000 | 32000 | 64000 | 128000 |
| 0,05 | 1844 | 2088 | 2496 | 3002 | 4000 | 5326 | 7406 | 11294 |
| 0,1 | 1874 | 2372 | 2618 | 3592 | 4406 | 6694 | 9712 | 13388 |
| 0,15 | 1978 | 2530 | 2876 | 3812 | 5008 | 5916 | 8730 | 12614 |
| 0,2 | 2052 | 2434 | 3028 | 3842 | 5076 | 5972 | 8434 | 13138 |
| 0,25 | 2118 | 2614 | 3352 | 4160 | 5174 | 6488 | 8662 | 12728 |
| 0,3 | 2058 | 2740 | 3220 | 3904 | 4566 | 6604 | 7554 | 12024 |
| 0,35000002 | 2086 | 2500 | 3548 | 4234 | 4690 | 6314 | 8120 | 10574 |

## K-

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Moyenne de score | Nombre de pas |  |  |  |  |  |  |  |
| k- | 1000 | 2000 | 4000 | 8000 | 16000 | 32000 | 64000 | 128000 |
| 0,05 | 1355,369524 | 1404,142857 | 1474,687619 | 1560,45977 | 1636,35619 | 1720,401905 | 1794,88381 | 1913,420597 |
| 0,1 | 1432,337143 | 1542,48 | 1688,809524 | 1846,998095 | 2029,967619 | 2203,773333 | 2403,184762 | 2620,184061 |
| 0,15 | 1512,624762 | 1673,318095 | 1897,300952 | 2147,093156 | 2465,245714 | 2786,721905 | 3209,67619 | 3765,451767 |
| 0,2 | 1555,721905 | 1750,2 | 2036,859048 | 2371,871893 | 2819,108571 | 3363,44381 | 4121,733333 | 5077,144084 |
| 0,25 | 1576,352381 | 1809,382857 | 2135,16381 | 2555,6673 | 3114,419048 | 3801,746667 | 4852,085714 | 6145,121327 |
| 0,3 | 1600,691429 | 1839,230476 | 2189,024762 | 2656,704198 | 3297,272381 | 4157,224762 | 5420,849524 | 7038,138031 |
| 0,35000002 | 1600,782857 | 1838,43619 | 2221,984762 | 2741,765595 | 3448,041905 | 4415,041905 | 5874,909524 | 7688,677871 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Max. de score | Nombre de pas |  |  |  |  |  |  |  |
| k- | 1000 | 2000 | 4000 | 8000 | 16000 | 32000 | 64000 | 128000 |
| 0,05 | 1670 | 1792 | 2096 | 2882 | 2182 | 2326 | 2398 | 6362 |
| 0,1 | 1820 | 2040 | 2268 | 2828 | 2956 | 3128 | 3698 | 8132 |
| 0,15 | 1930 | 2308 | 2954 | 3150 | 3564 | 4248 | 4794 | 7738 |
| 0,2 | 2044 | 2322 | 2902 | 3516 | 4174 | 4782 | 6900 | 10402 |
| 0,25 | 2052 | 2500 | 3188 | 3732 | 4676 | 5452 | 7888 | 11540 |
| 0,3 | 2086 | 2740 | 3548 | 4234 | 5076 | 5916 | 9712 | 13388 |
| 0,35000002 | 2118 | 2614 | 3220 | 4040 | 5174 | 6694 | 8934 | 13138 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Moyenne de score | Nombre de pas |  |  |  |  |  |  |  |
| Taille Mémoire | 1000 | 2000 | 4000 | 8000 | 16000 | 32000 | 64000 | 128000 |
| 5 | 1527,502857 | 1715,702857 | 1992,917551 | 2347,829457 | 2766,296327 | 3211,881633 | 3710,228571 | 4549,83075 |
| 10 | 1519,410612 | 1692,886531 | 1944,053061 | 2247,351814 | 2662,674286 | 3158,172245 | 3896,973469 | 4600,160997 |
| 15 | 1510,463673 | 1673,063673 | 1910,385306 | 2213,605887 | 2632,634286 | 3250,669388 | 4254,507755 | 5557,506366 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Max. de score | Nombre de pas |  |  |  |  |  |  |  |
| Taille Mémoire | 1000 | 2000 | 4000 | 8000 | 16000 | 32000 | 64000 | 128000 |
| 5 | 2086 | 2614 | 3548 | 4234 | 5174 | 6604 | 7168 | 12318 |
| 10 | 2118 | 2456 | 3018 | 4160 | 5076 | 6488 | 8468 | 10986 |
| 15 | 2052 | 2740 | 2832 | 3756 | 4676 | 6694 | 9712 | 13388 |

Nous remarquons que pris un par un, la taille de la mémoire, k+ et k- ne semblent pas avoir de grand impact. Il y a juste une baisse de performance notable quand k- est très faible. Cela peut facilement s’expliquer par le fait qu’avec un k- très faible, l’agent va très souvent reposer l’objet qu’il a pris au même endroit qu’il l’a pris. L’algorithme réalise en effet beaucoup de déplacement inutiles, et n’arrive pas à executer une tâche quelconque.  
Il semblerais qu’il y ai un phenomène similaire avec k+ (quand k+ est très petit, l’agent preferera très souvent ne rien porter). Mais nos plages de tests sont heureusement sufffisament loin de ce seuil. Il peut être intéressant de prendre des valeurs de k+ inférieures à 0.05 pour confirmer ou non cette hyopthèse.

Nous allons donc nous concentrer sur des valeurs de K- élevées et nous pencher sur les autres parametres :

*\*Par la suite, nous ne prendrons que les valeurs des simulations de 128000 pas*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Moyenne de score | k- |  |  |  |
| k+ | 0,05 | 0,25 | 0,3 | 0,35000002 |
| 0,05 | 1766,742857 | 5017,235294 | 6119,478571 | 6760,867133 |
| 0,1 | 1993,006897 | 6338 | 7119,685714 | 7995,388235 |
| 0,15 | 1931,44 | 6550,785235 | 7656,118343 | 8313,608108 |
| 0,2 | 1933,758389 | 6439,251748 | 7174,228571 | 8152,52349 |
| 0,25 | 1953,283784 | 6433,208054 | 7357,251613 | 7770,517986 |
| 0,3 | 1905,729032 | 6030,517241 | 7050,545455 | 7409,288591 |
| 0,35000002 | 1903,907895 | 6102 | 6620,246377 | 7394,959538 |

Nous pouvons voir ici, qu’avec un k- élevé et un k+ aux alentours de 0.15, les résultats semblent être meilleurs.

Si nous séparons nos résultats par taille de mémoire :

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Moyenne de score | taille de mémoire et k- |  |  |  |  |  |  |  |  |
|  | 5 |  |  | 10 |  |  | 15 |  |  |
| k+ | 0,25 | 0,3 | 0,35000002 | 0,25 | 0,3 | 0,35000002 | 0,25 | 0,3 | 0,35000002 |
| 0,05 | 4628,857143 | 4940,756757 | 5500,793103 | 4565,392857 | 5806,034483 | 6682,857143 | 5881,6 | 7492,644444 | 8536,697674 |
| 0,1 | 5462,622951 | 5749,288136 | 6380,580645 | 6043,321429 | 7312,146341 | 7439,076923 | 7794,25 | 8943,75 | 9392,731707 |
| 0,15 | 5408 | 6597,966667 | 6878,7 | 6282,689655 | 7692,392857 | 8008,518519 | 7898,479167 | 8815,698113 | 9681,592593 |
| 0,2 | 5274,75 | 5788,52381 | 6777,291667 | 6313,714286 | 7247,931034 | 8180,565217 | 7593,377358 | 8522,35 | 9329,272727 |
| 0,25 | 5844,385965 | 6470,655172 | 6752,153846 | 5992,708333 | 7395,208333 | 7799,956522 | 7676,545455 | 8369,510204 | 9029,073171 |
| 0,3 | 5785,347826 | 6201,16 | 6603,084746 | 5724,196721 | 6928,702128 | 7498,75 | 6819,026316 | 7896,087719 | 8439,571429 |
| 0,35000002 | 5681,515152 | 5961,294118 | 7018,75 | 5931,657143 | 6492,923077 | 7042,84 | 6605,19403 | 7406,923077 | 7870,48 |

Maintenant que nous pensons cerner une solution, nous pouvons essayer de faire plus de test avec une granularité plus fine sur nos paramètres :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| steps | 128000 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Moyenne de score | Mémoire et k- |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 10 |  |  |  | 15 |  |  |  | 20 |  |  |  | 25 |  |  |  |
| k+ | 0,33 | 0,36 | 0,39 | 0,42 | 0,33 | 0,36 | 0,39 | 0,42 | 0,33 | 0,36 | 0,39 | 0,42 | 0,33 | 0,36 | 0,39 | 0,42 |
| 0,1 | 12658 | 12336 | 12706 | 13408 | 10669 | 11298 | 11928 | 11761 | 11800 | 12865 | 13141 | 12974 | 12790 | 12399 | 13120 | 13118 |
| 0,125 | 11662 | 12855 | 13035 | 12674 | 10878 | 11464 | 11606 | 12289 | 11609 | 11265 | 13097 | 12999 | 12207 | 12766 | 12605 | 13035 |
| 0,15 | 12381 | 13267 | 13091 | 13616 | 11400 | 11332 | 11832 | 11843 | 12540 | 11997 | 11586 | 12711 | 12080 | 12454 | 12799 | 13254 |
| 0,17500001 | 11805 | 11923 | 13118 | 12972 | 10556 | 11752 | 12202 | 11083 | 11816 | 11669 | 11122 | 12487 | 12057 | 12131 | 12704 | 12547 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| steps | 128000 |  |  |  |
| memorySize | 10 |  |  |  |
| kMinus | 0,42000002 |  |  |  |
|  |  |  |  |  |
| Moyenne de score | k+ |  |  |  |
| Nombre d'agents | 0,1 | 0,125 | 0,15 | 0,17500001 |
| 10 | 8717 | 10004 | 9595 | 10858 |
| 20 | 13348 | 14071 | 15328 | 14595 |
| 30 | 15541 | 14609 | 14719 | 13675 |

Nous remarquons ici qu’augmenter le nombre d’agent accelère le tri. L’environnement étant assez grand, ils sont en sous population et ne se gène pas. Nous pouvons donc monter à 30 agents, même si on peut s’attendre à des resultats similaires avec 10 agents si nous leur laissons plus de temps.

|  |  |  |  |
| --- | --- | --- | --- |
| steps | 128000 |  |  |
| memorySize | 10 |  |  |
| nbAgent | 30 |  |  |
| kMinus | 0,42 |  |  |
|  |  |  |  |
| Moyenne de score | erreur |  |  |
| k+ | 0 | 0,05 | 0,1 |
| 0,1 | 15541 | 13234 | 13750 |
| 0,125 | 14609 |  | 12485 |
| 0,15 | 14454 | 13285 | 11948 |

Nous pouvons essayer d’ajouter un système d’erreur sur nos agents, mais cela ne semble pas avoir de résultats significatifs