DIEP Richard 2I013

EL GHAZZI Sarah

**Rapport final**

**Projet de Vie Artificielle et Jeux Systémiques**



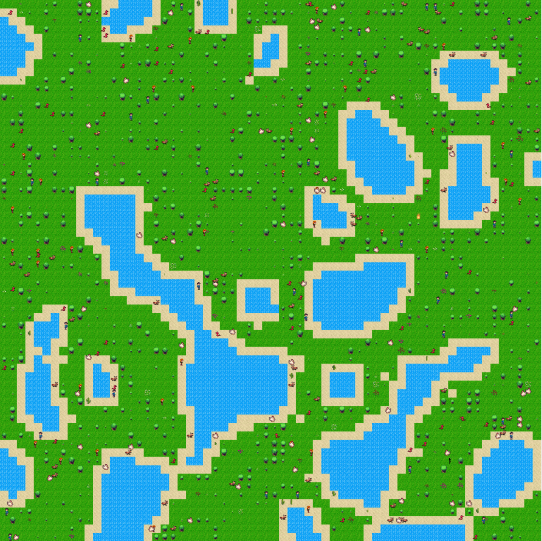
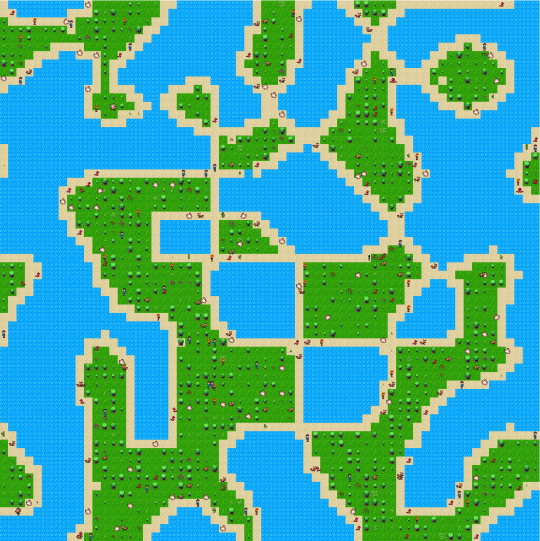
BREDECHE Nicolas

2018-2019

Le projet de Vie Artificielle et Jeux Systémiques que nous avons réalisé est fait sous un rendu 2D avec images, codé en JAVA. Nous avons fait ce choix car nous n’avons pas trouvé l’utilité d’utiliser un rendu 3D pour les implémentations des options. Le choix du langage JAVA est lié à notre niveau car nous sommes plus à l’aise avec JAVA.

Nous allons commencer par parler du monde, du terrain et de son évolution. Puis nous parlerons des agents, ainsi que leur façon de se déplacer.

Lors de plusieurs exécutions du programme, il est possible de constater que le terrain créé n’est pas identique. En effet, nous avons implémenté le bruit de Perlin que nous avons vu en cours pour générer automatiquement des paysages. Ainsi, nous avons un monde ayant un terrain avec de la cohérence et de manière ordonné.

Voici quelques illustrations des terrains générés aléatoirement :

Ainsi, on peut observer :



* l’herbe en vert

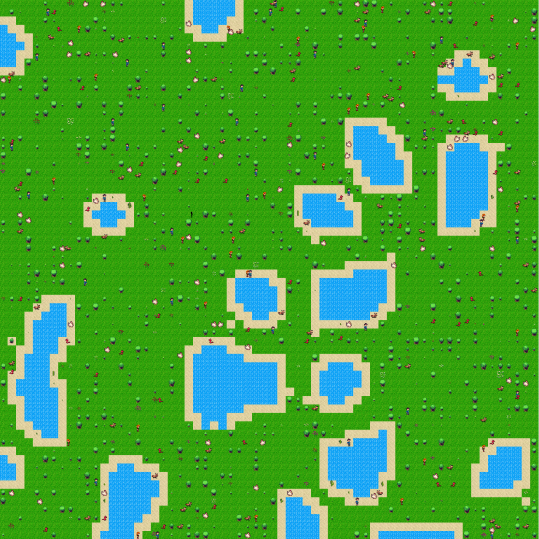
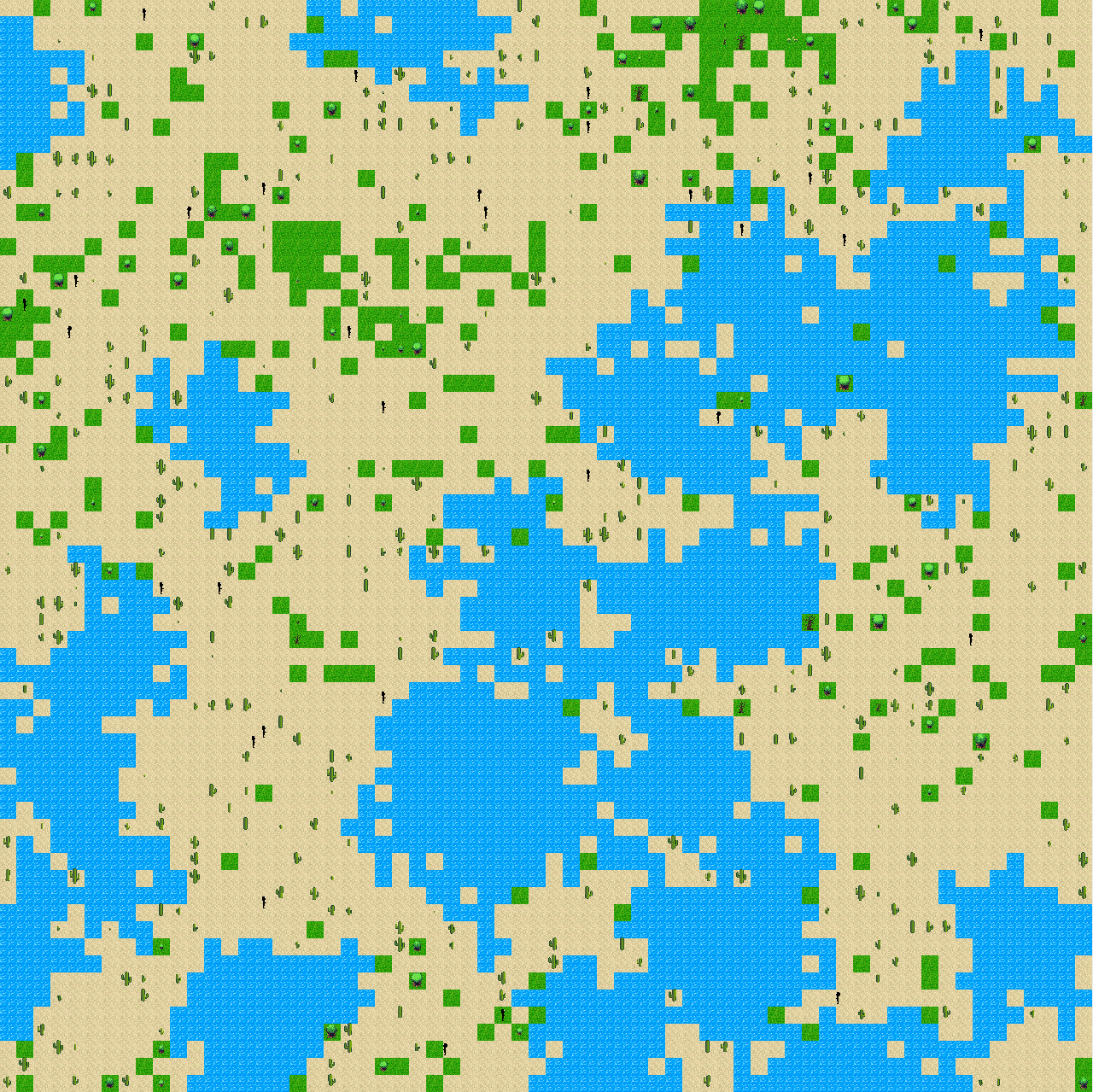
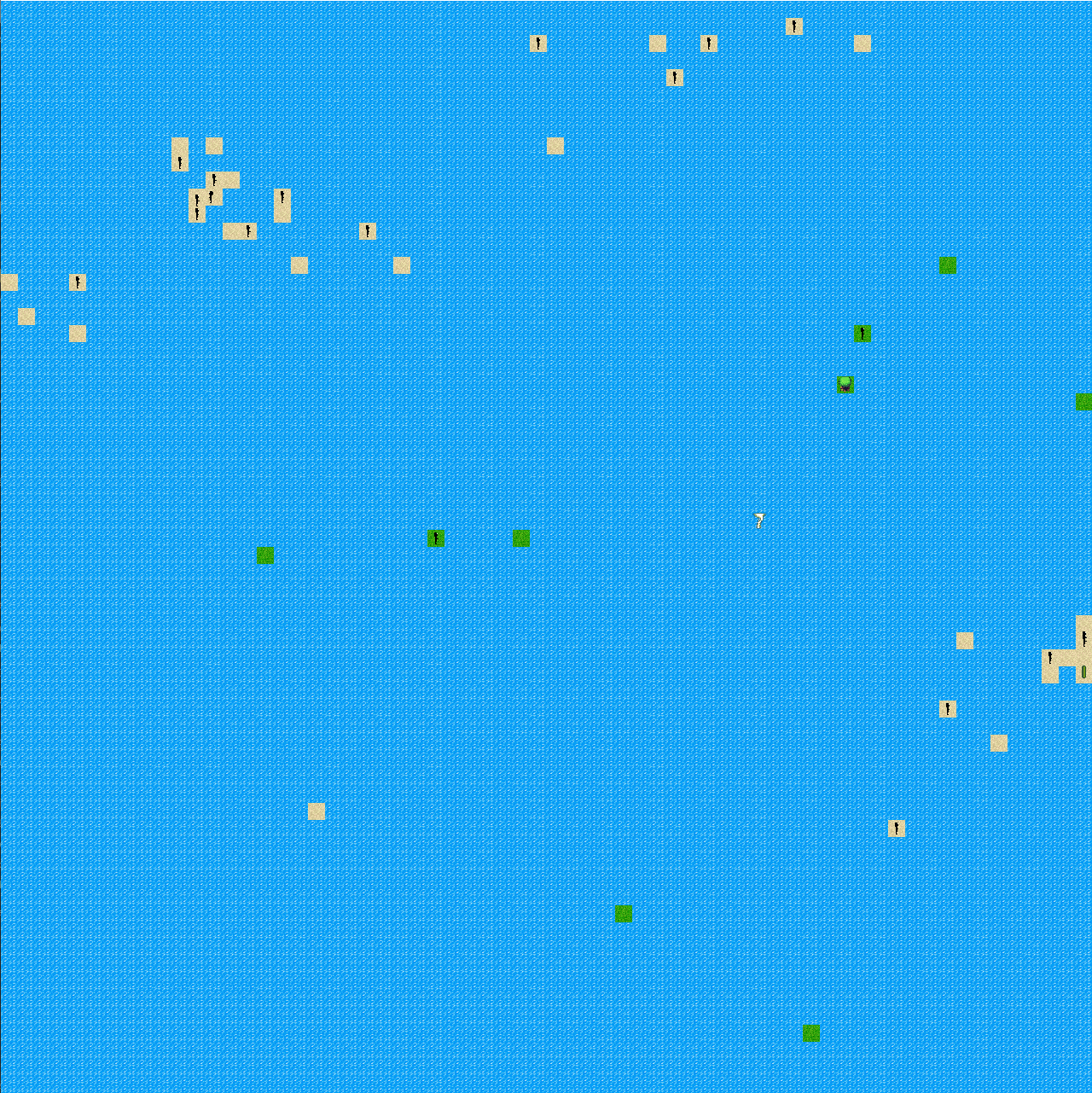


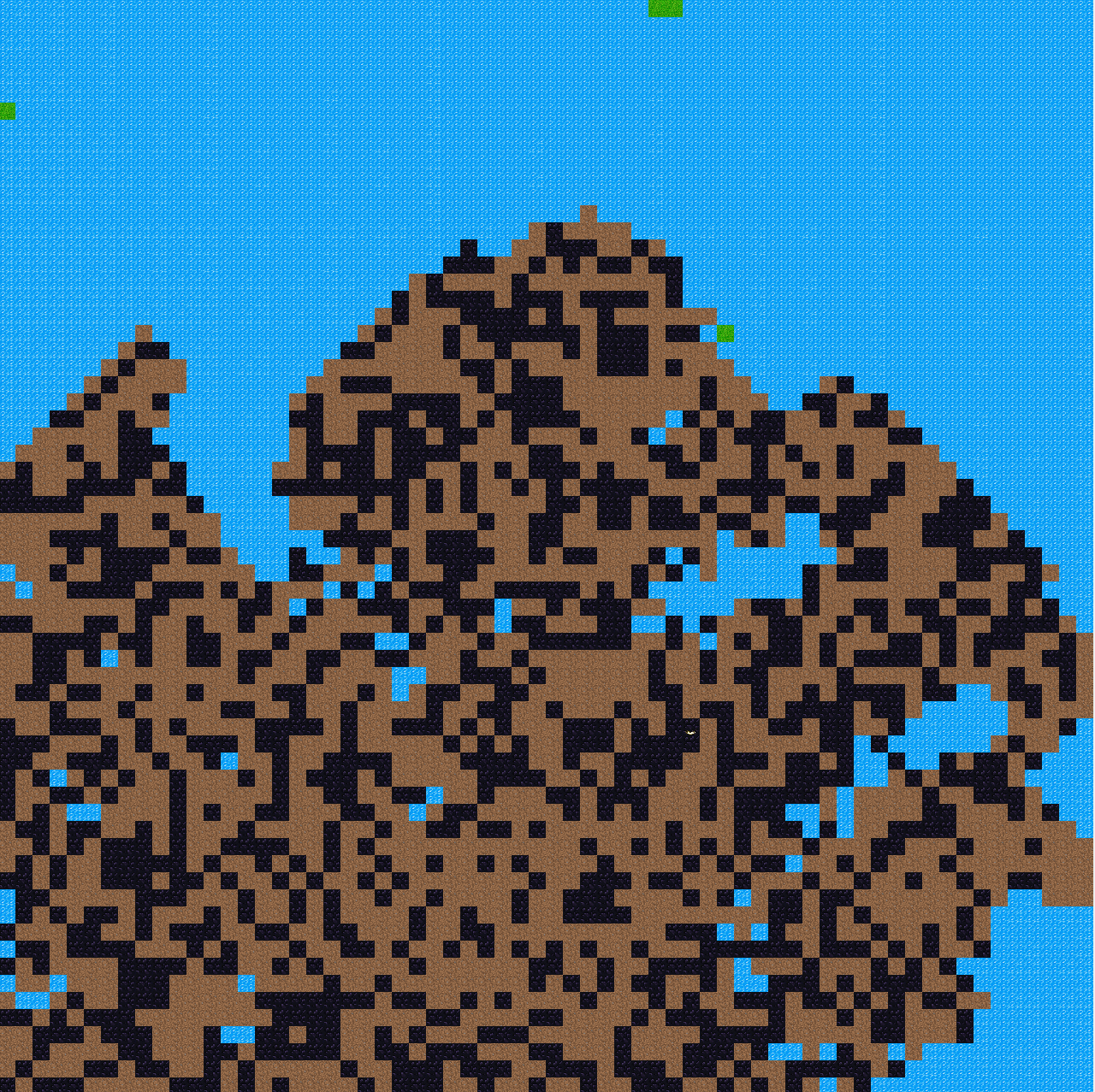
* le sable en beige

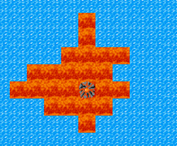


* l’eau en bleu (avec deux images pour créer les animations)

Nous avons fait un choix assez simple puisque nous avons eu l’idée de créer un monde dans lequel le terrain se renouvelle et notamment grâce à la tectonique des plaques où une herbe ayant du sable dans un voisinage de Moore, peut devenir du sable. De même que le sable peut être remplacé, dans un voisinage de Von Neumann, de l’eau. Au fur et à mesure, le terrain réduit jusqu’à ne laisser quelques zones d’herbes sembable à des petites îles. Voici des images représentant la tectonique des plaques :



 Cependant, cette disparition de terrain doit être compensée par une apparition de nouveau terrain pour permettre de créer un monde sans fin et qui se renouvelle toujours. De ce fait, nous avons choisi de créer un volcan lorsqu’il n’y a plus de sable dans le monde. Dans le cas où il reste de l’herbe, donc des îles, un volcan apparaît aléatoirement sur l’un des îles. Sinon il apparaît au centre du monde. Lorsque le volcan apparaît, on peut observer un écoulement de lave (Avec un voisinage de Von Neumann avec du bruit et avec le bruit de Perlin pour éviter une forme de losange) jusqu’à une certaine distance aléatoire et où la lave au contact de l’eau, crée de l’obsidienne. Ce dernier devient de la terre, qui deviendra ensuite de l’herbe. A noter que tout doit avoir transformé avant de passer à un autre changement. Voici les étapes de la création du nouveau terrain :





* Lave
* Obsidienne
* Terre

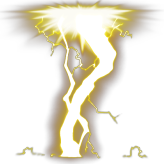
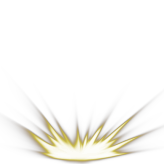
Volcan avec propagation de lave autour

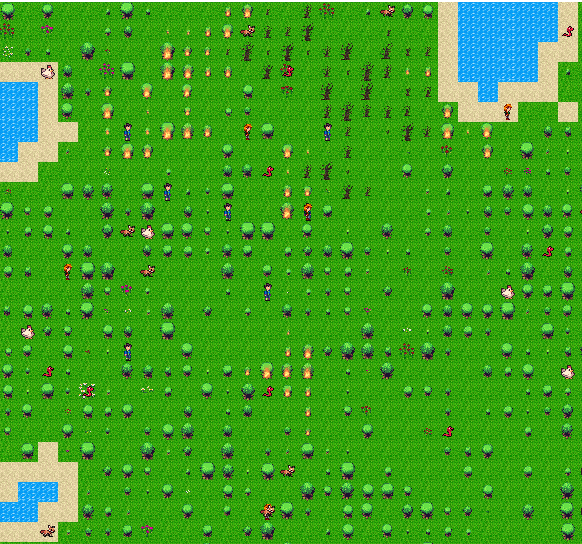


Lorsqu’il n’y a que de l’herbe, on ajoute du sable aux extrémités. Enfin, on remet des agents, des objets comme les arbres, les cactus aléatoirement sur le terrain. Le nouveau monde est terminé et tout recommence. Nous sommes ainsi parvenus à créer un monde où le terrain est renouvelé à chaque fois.

A présent, nous allons parler des objets présents sur le terrain, ainsi que le feu, qui apparaissent tous aléatoirement. En effet, nous avons ajouté des arbres et des arbres brulés, des cactus, des fleurs et des coups de tonnerre. Chaque objet possède une durée de vie, une taille différente et leur taille peut augmenter jusqu’à une certaine limite. Le feu s’adapte à la taille de chaque objet et est affiché au dessus de chaque objet pour réduire le nombre d’images.

Le feu apparaît suite à un coup de tonnerre à une zone du terrain sauf si le coup de tonnerre impacte l’eau. Le feu reste pendant une durée définie par fireStop dans le programme. Le coup de tonnerre se réalise en deux étapes, la foudre et l’impact :



 Les arbres et les cactus sont des obstacles pour les agents, qui apparaissent respectivement sur de l’herbe et du sable. Lorsque l’arbre est touché par un coup de tonnerre, l’arbre prend feu et lorsque le feu se dissipe, l’arbre devient un arbre brulé et vieillit plus vite. Si ce dernier redevient en feu, il disparaît lorsque le feu se dissipe. Nous avons aussi profité de ce qu’on a vu en cours pour implémenter des feux de forêts, qui ne fonctionne uniquement sur les arbres non brulés.

Quant au cactus, lorsque le feu se dissipe, il disparaît. Voici une illustration des arbres en arbres brulés :



Voici les différentes images du cactus :



Les fleurs sont aussi présentes dans le monde, et possèdent un rôle pour les agents où nous en parlerons plus tard. Elles ont un age, et lorsque l’age maximal est atteint, elles disparaissent. Il y a trois types de fleurs : les roses, les tulipes et les marguerites. Le choix d’avoir trois types de fleurs correspond aussi à un choix pour les agents que nous avons sélectionné et que nous verrons plus tard. Voici des images de la rose :



Maintenant que nous avons parlé du monde et du terrain, nous allons voir ce qui a été réalisé pour les agents. Voici la liste des agents :

* L’humain, qui cherche à vivre le plus longtemps possible
* Le zombie, qui est à la constante recherche d’un humain afin de le manger
* La poule qui cherche, dans un rayon que l’on peut modifier, sa proie, la vipère. Cependant, elle fuit lorsqu’un renard se trouve dans le voisinage de Von Neumann
* Le renard qui cherche sa proie, la poule. Il fuit lorsqu’une vipère se trouve dans le voisinage de Von Neumann
* La vipère cherche sa proie, le renard. Elle fuit lorqu’une poule se trouve dans le voisinage de Von Neumann.

A noter que la fuite est prioritaire à la chasse. Cependant, nous avons remarqué une limite de créer un algorithme de chasse et de fuite car le prédateur ne peut pas atteindre la proie sans que la proie soit bloqué. De ce fait, nous avons décidé de réduire le rayon de la fuite en utilisant un voisinage de Von Neumann, alors que la chasse se fait au minimum avec un voisinage de Moore. Cela permet au prédateur d’avoir une chance supplémentaire d’attraper la proie.

Un cas où le prédateur peut manger la proie :

Supposons que les agents peuvent aller dans les quatres directions. Les déplacements ne se font que verticalement ou horizontalement.

Le renard mange la poule. Comme il possède un voisinage de Moore, il trouve la poule et choisi l’un des deux chemins en bleu.

Quant à la poule, elle ne voit pas le renard qui la chasse avec un voisinage de Von Neumann. Alors quatres directions sont possibles, représentée par les flèches rouges.

Ainsi, le renard a une chance sur deux de tomber sur la même case que la poule et de la manger. Lorsque le prédateur mange sa proie, il gagne de la santé (nous en parlerons ci-dessous) et la proie meurt.

Nous avons aussi mis une limite de temps de chasse, lorsque le prédateur chasse une proie pendant trop de temps, il abandonne et ne chasse plus pendant un moment.

Dans le cas où l’agent ne chasse pas et ne fuit pas, il se déplacement aléatoirement sur les positions possibles.

Les agents, mis à part le zombie, possèdent les caractéristiques suivantes :

* un âge avec une limite d’âge avant de mourir. L’âge augmente à chaque itération.
* une santé. Si l’agent a une santé de 0, il meurt. La santé diminue à chaque itération si une probabilité ploseHealth est atteinte.
* un sexe, qui permet l’accouplement entre agents de même type. Donc si deux humains avec deux sexes différents sont sur la même position, et que la durée avant de pouvoir avoir un enfant est atteinte (representée par la variable stime dans le code), un enfant apparaît sur la même position que les deux humains. En effet, nous avons mis une limite d’enfant pour chaque agent. Si aucune limite n’était posée, il y aurait des enfants à l’infini et les agents ne disparaissent pas.

Comment les agents gagnent ou perdent donc de la vie ? Nous avons ainsi vu qu’ils peuvent en gagner en mangeant leur proie respective. Mais ils peuvent aussi en gagner en mangeant des fleurs. En effet, les humains mangent les roses, les poules mangent les tulipes et les renards mangent les marguerites. Quant aux vipères, elles ne mangent que leur proie, le renard. Nous avons fait en sorte que les vipères gagnent substantiellement plus de santé lorsqu’elles mangent leur proie et que la santé soit plus élevée que les autres agents.

Nous avons parlé des agents mis à part le zombie qui est un agent aussi, mais il possède des différences par rapport aux autres agents :

* Il n’a pas de santé, donc il n’a pas besoin de regagner de la santé
* Il meurt instantanément lorsqu’il est touché par du feu
* Il a une durée d’apparition refletée par l’âge
* Aucun agent ne chasse les zombies

A présent, revenons un court instant sur les feux et les coups de tonnerre. En effet, les agents qui se font toucher par un coup de tonnerre subissent des dégâts conséquents (variable thunderDamage dans le programme) sur le moment. De même qu’un agent en feu subit des dégâts de feu pendant toute la durée du feu (variable fireDamage). Le feu est affiché par-dessus les images de chaque agent pour permettre avoir un déplacement synchronisé et fluide du feu et de l’agent.

Le déplacement de chaque agent, donc d’une position à une autre est aussi affichée de manière fluide. En effet, nous avons fait en sorte qu’au lieu de voir un saut d’une position à une nouvelle position d’un agent, l’agent se déplace jusqu’à la nouvelle position.

Voici une vipère qui se déplace de la position (x,y) à (x,y-1) : 

Aucun agent ne peut se déplacer dans l’eau, et lorsque le sable se transforme en eau, l’agent a un temps limité pour pouvoir sortir de l’eau et aller sur une plateforme. Si aucune des quatres positions autour de lui ne sont pas des positions où l’agent peut se déplacer, alors il ne bouge pas et se noie au bout d’un certain temps et meurt.