

DIEP Richard

2I013

EL GHAZZI Sarah

Rapport final

Projet de Vie Artificielle et Jeux Systémiques



BREDECHE Nicolas

2018-2019

Sommaire

Sommaire	1
Le monde et sa création	2
La tectonique des plaques et le nouveau terrain (nouvelle vie)	2
Les objets présents sur le terrain	4
Les agents	5
Les différents agents	5
Les priorités des agents	5
Les déplacements, les prédateurs et les proies	7
Les déplacements vus au niveau graphique	8
Conclusion	8
Annexe	9
Guide d'installation	9
Méthode pour exécuter le programme avec un terminal	9
Voici un exemple d'exécution du programme avec illustrations sous LINUX	9
Guide de modification	11
Voici un exemple de modification de code avec illustrations	11

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 et leurs propriétés. Enfin, nous parlerons du graphique lié au déplacement des agents.

Le monde et sa création

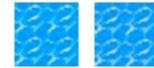
Le monde créé n'est pas torique, il est synchrone et stochastique. Lors de plusieurs exécutions du programme, il est possible de constater que le terrain créé n'est pas identique. En effet, nous avons implémenté le bruit de Perlin que nous avons vu en cours pour générer automatiquement des paysages. 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 :

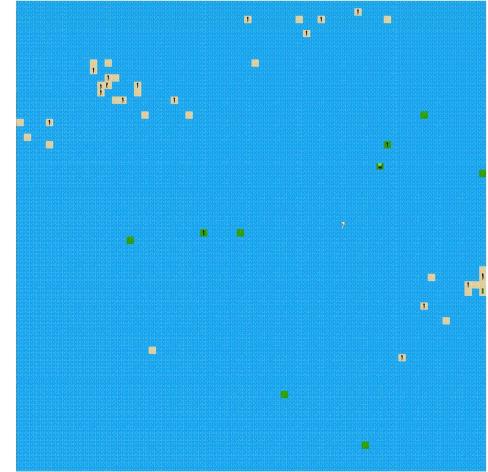
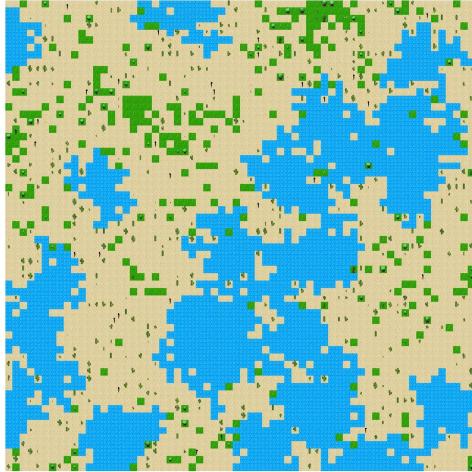
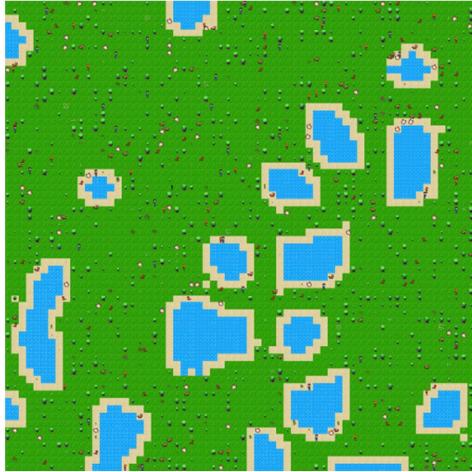
- l'herbe
- le sable
- l'eau (avec deux images pour créer les animations)



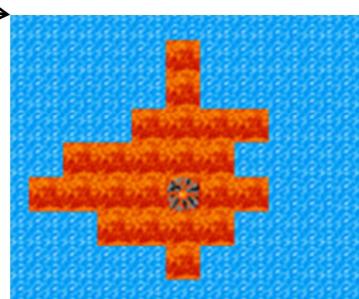
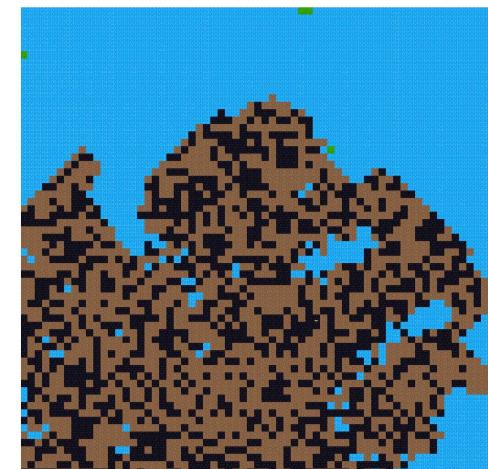
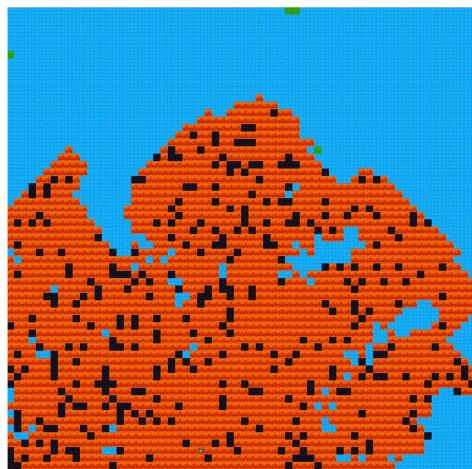
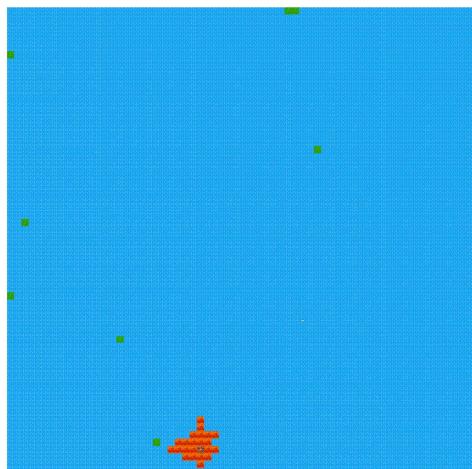
La tectonique des plaques et le nouveau terrain (nouvelle vie)

La tectonique des plaques a été implémentée aussi puisque nous avons eu l'idée de créer un monde dans lequel le terrain se renouvelle et 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 semblables à des petites îles.

Voici des images représentant la tectonique des plaques :



Cependant, cette disparition de terrain doit être compensée par une apparition d'un nouveau terrain pour permettre de créer un monde sans fin et qui se renouvelle toujours. De ce fait, nous avons choisi d'implémenter un volcan lorsqu'il n'y a plus de sable dans le monde. Dans le cas où il reste de l'herbe, que nous pouvons voir comme des îles, un volcan apparaît aléatoirement sur l'une 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 (Inspiré du TP1 avec les automates cellulaires 1D) et avec le bruit de Perlin pour éviter un terrain linéaire en forme de losange) jusqu'à une certaine distance aléatoire. La lave au contact de l'eau, crée de l'obsidienne, ce qui permet à la lave de se propager à de longues distances. L'obsidienne devient ensuite de la terre, qui deviendra ensuite de l'herbe. A noter que tout doit avoir transformé avant de passer à une autre transformation. Voici les étapes de la création du nouveau terrain :



- Lave
 - Obsidienne
 - Terre
-

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 noter que dans le cas où tout le monde est rempli d'herbe, le sable ne peut pas se créer puisqu'il n'y aura pas d'eau. Pour contrer cela, une flac d'eau se crée aléatoirement sur le terrain, dont du sable à proximité.

Les objets présents sur le terrain

A présent, nous allons parler des objets présents sur le terrain, ainsi que le feu, qui apparaissent tous aléatoirement. Certains des objets ont plusieurs états. En effet, nous avons ajouté des arbres et des arbres brûlés, 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. Toutes ses caractéristiques peuvent être modifiées dans le programme. Le feu s'adapte à la taille de chaque objet et est affiché au dessus de chaque objet pour réduire le nombre d'images dans le dossier.

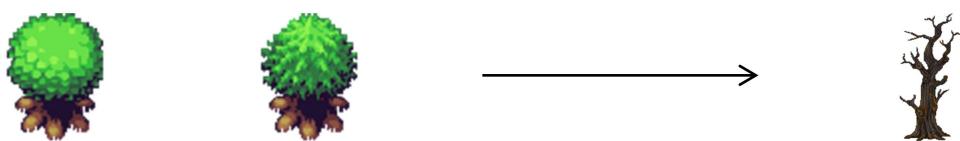
Le feu apparaît suite à un coup de tonnerre à une zone du terrain sauf si le coup de tonnerre impacte dans l'eau. Le feu reste pendant une durée définie par la variable fireStop dans le programme. Le coup de tonnerre se réalise en deux étapes, la foudre et l'impact :



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 brûlé et vieillit plus vite. Cette idée est dérivée du TP3 sur les feux de forêts que nous avons fait dans lequel l'arbre devenait du cendre. Lorsque l'arbre brûlé 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 brûlés.

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



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 :



Les agents

Les différents agents

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 le prédateur de l'humain
- 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 lorsqu'une poule se trouve dans le voisinage de Von Neumann.

A noter que la fuite est prioritaire à la chasse.

Les propriétés des agents

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. Cela évite qu'un agent vive jusqu'au renouvellement du terrain
- une santé. Si l'agent a une santé de 0, il meurt. Avec une probabilité ploseHealth, la santé diminue.
- un sexe, qui permet l'accouplement entre agents de même type. Cela permet d'avoir de nouveaux agents de chaque type et permet d'éviter l'extinction des agents avant le renouvellement du terrain. 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 (représentée 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. De même que nous avons mis en place une limite

d'agent par case (uniquement pour la naissance d'agent, pas pour l'initialisation au départ)

Comment les agents gagnent ou perdent donc de la vie ? Nous verrons dans la section des prédateurs et agents que les agents peuvent chasser leur proie pour gagner de la santé. 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. Chaque fleur rapportant une certaine quantité de santé en plus à l'agent respectif. 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 reflétée par l'âge
- Aucun agent ne chasse les zombies
- Il apparaît n'importe quand alors que les autres agents apparaissent soit par naissance, soit au début de chaque nouveau terrain
- Il se déplace bien plus rapidement que les autres agents
- Il peut traverser l'eau (et non se déplacer directement dans l'eau)

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.

De même que si un cactus ou un arbre apparaît à une position où il y a des agents, les agents meurent instantanément. Cela permet d'éviter qu'un agent reste bloqué dans un arbre ou un cactus.

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).

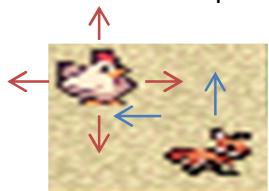
Les déplacements, les prédateurs et les proies

Les déplacements (attention, on parle de déplacement et pas de l'affichage) ne se font que verticalement ou horizontalement et jamais en diagonal.

Un algorithme de chasse, un algorithme de fuite et un algorithme de déplacement aléatoire ont été implémenté pour les agents. Les agents peuvent aussi se déplacer à une position en feu. Ce choix a été fait pour permettre de faire mourir plus d'agents et d'avoir plus de nouveau-né.

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 puisqu'elle cherche toujours le chemin où le prédateur n'est pas présent. De ce fait, nous avons décidé de réduire le rayon de la fuite en utilisant un voisinage de Von Neumann, comme le TP4 que nous avons fait en cours, 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, sans pour autant que le prédateur l'atteigne à coup sûr.

Un cas où le prédateur peut manger la proie :



Supposons que les agents peuvent aller dans les quatres directions.

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.

Une limite de temps de chasse a été aussi implémentée. Lorsque le prédateur chasse une proie pendant trop longtemps, il abandonne et il ne chasse plus pendant un moment. Cela permet au prédateur de faire une pause au lieu de poursuivre à l'infini.

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

Nous avons vu ci-dessus que le zombie se déplace plus rapidement que les autres agents. La recherche d'une nouvelle position se fait avec un voisinage de Von Neumann. Mais comme on recherche plusieurs fois une nouvelle position dans une même itération, le zombie se déplace ainsi plus rapidement que les autres.

Les déplacements vus au niveau graphique

A présent, nous allons discuter sur l'affichage des agents et la partie graphique des agents. Chaque agent est mis à jour en même temps que ce soit la position d'un agent ou du déplacement fluide au niveau graphique.

Premièrement, le feu est affiché par-dessus les images de chaque agent pour permettre d'avoir un déplacement synchronisé et fluide du feu et de l'agent.

Deuxièmement, 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. Mais aussi, l'agent qui se déplace horizontalement uniquement voit son image s'orienter vers le sens de la direction.

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



Voici lors qu'un humain se déplace à gauche et à droite respectivement :



Enfin, les déplacements diagonaux ne sont pas autorisés mis à part le zombie qui peut voir uniquement son image se déplacer diagonalement. Mais le changement de position reste uniquement avec un voisinage de Von Neumann

Attention, un agent peut disparaître subitement lorsqu'un prédateur mange une proie. C'est-à-dire que la proie et le prédateur ne sont pas encore sur la même case. Mais comme au niveau des coordonnées les deux agents sont sur la même case, la proie meurt et disparaît.

Conclusion

Pour conclure, nous sommes satisfaits de notre programme car la majorité des fonctionnalités du programme ont été réalisées. Le monde fonctionne en continu sans fin, les terrains créés sont toujours différents grâce aux volcans et à la tectonique des plaques. Les objets, bien qu'il n'y en a pas beaucoup, ont une utilité, que ce soit pour faire obstruction ou aider les agents. Les agents sont divers et variés et notamment au niveau des prédateurs et proies, seul le zombie ne possède pas de prédateur et l'humain ne possède pas de proie. Les autres agents ont tous un prédateur et une proie. Cependant, il reste des points à améliorer, par exemple ajouter de l'altitude, ajouter des prédateurs et proies dans l'eau.

Annexe

Guide d'installation

- Décompresser le Programme-DIEP_EL GHAZZI.zip à l'emplacement souhaité
- Accéder au programme décompressé qui s'appelle WorldOfSprites. Vous verrez plusieurs dossiers, notamment le dossier src.
- Deux façons d'exécuter le programme se présente : Avec le terminal ou avec Eclipse IDE (Les deux méthodes sont disponibles dans le fichier AIDE.txt)

(Dans le cas où le programme est trop grand ou trop petit ou trop lent, une aide se trouve dans AIDE.txt)

Attention, chaque image est susceptible d'être légèrement différent dans le programme.

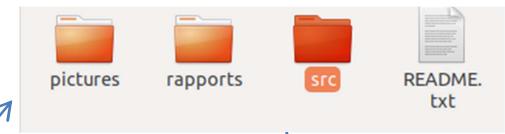
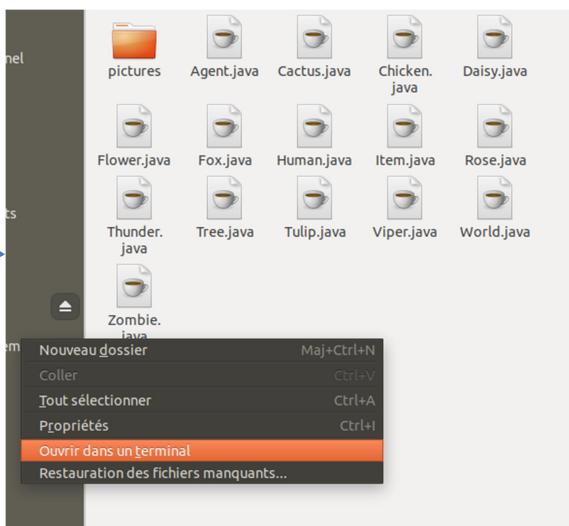
Méthode pour exécuter le programme avec un terminal

- Ouvrir le terminal
- Parcourir jusqu'au dossier src dans WorldOfSprites dans le terminal (avec la commande cd, par exemple cd Documents/2I013/WorldOfSprites/src)
- Compiler le programme avec la commande suivante : javac *.java
- Exécuter le programme avec la commande suivante : java World
- Le programme s'exécute

Voici un exemple d'exécution du programme avec illustrations sous LINUX

- Extraire/Décompresser le Programme-DIEP_EL GHAZZI.zip
- Un dossier du nom de Programm-DIEP_EL GHAZZI apparaît, accéder au dossier
- Accéder au dossier WorldOfSprites
- Vous verrez plusieurs fichiers .java et un dossier pictures
- Faire un clique droit sur une partie de la fenêtre ne contenant ni fichier, ni dossier et ouvrir un terminal (Sinon ouvrir un terminal et accéder au dossier avec la commande cd)
- Taper dans la console les commandes suivantes pour exécuter le programme : javac *.java
- Puis java World. Le programme apparaît et s'exécute





```
rich@ubuntur:~/Documents/Programme-DIEP_EL_GHAZZI/WorldOfSprites/src$ javac *.java
rich@ubuntur:~/Documents/Programme-DIEP_EL_GHAZZI/WorldOfSprites/src$ java World
```

A screenshot of a terminal window on an Ubuntu system. The command 'javac *.java' is run to compile the Java source code, and then 'java World' is run to execute the application. The terminal window has a dark background with white text.



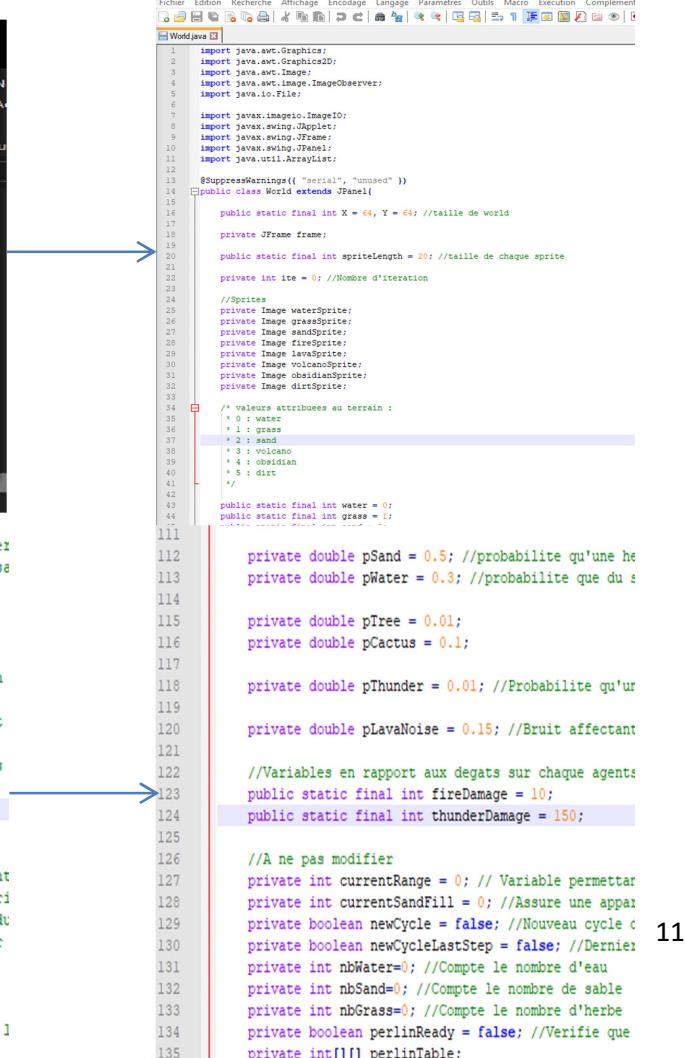
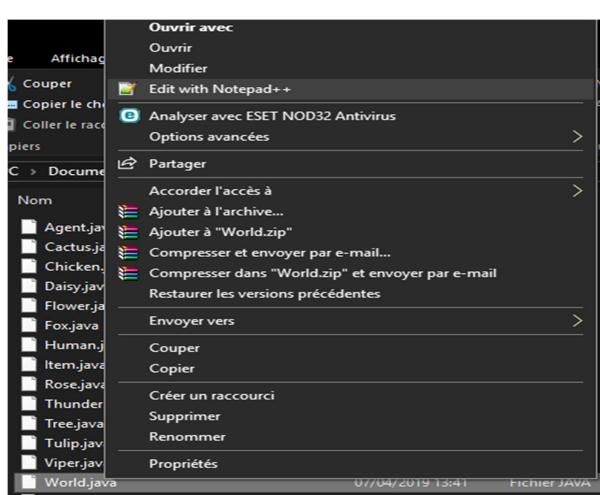
Guide de modification

- Ouvrir un fichier JAVA à l'aide d'un éditeur texte, mis à part Item.java
- Ignorer les lignes avec import et @SuppressWarnings...
- A partir de la ligne public class, la valeur de chaque variable peut être modifiée jusqu'à la ligne où vous verrez écrire : //A ne pas modifier

La majorité des variables ont un commentaire associé pour vous permettre de comprendre au mieux à quoi sert chaque variable.

Voici un exemple de modification de code avec illustrations

- Ouverture de World.java avec notepad++
- Notepad++ s'ouvre, et on peut observer les lignes de code se trouvant dans World.java
- Ignorer les lignes avec import et @SuppressWarnings
- En descendant légèrement, il est possible de voir la ligne //A ne pas modifier (ligne 126 sur l'image)
- Nous souhaitons par exemple modifier les dégâts dûs à un coup de tonnerre à 150. Ainsi, nous pouvons voir qu'à la ligne 124, la variable nommée thunderDamage, représente les dégâts du coup de tonnerre. Ainsi, on modifie la valeur de thunderDamage = 50 à thunderDamage = 150
- Sauvegarder, recompiler à l'aide du fichier AIDE.txt, et l'exécuter



```
import java.awt.Graphics;
import java.awt.GridLayout;
import java.awt.Image;
import java.awt.image.ImageObserver;
import java.io.File;
import java.awt.image.ImageIO;
import javax.swing.JApplet;
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.util.ArrayList;
import java.util.List;
@SuppressWarnings({"serial", "unchecked"})
public class World extends JPanel{
    public static final int X = 64, Y = 64; //taille de world
    private JFrame frame;
    public static final int spriteLength = 20; //taille de chaque sprite
    private int ite = 0; //Nombre d'itération
    //Sprites
    private Image waterSprite;
    private Image grassSprite;
    private Image sandSprite;
    private Image fireSprite;
    private Image lavaSprite;
    private Image obsidianSprite;
    private Image dirtSprite;
    /* Valeurs attribuées au terrain :
     * 1 : water
     * 2 : grass
     * 3 : sand
     * 4 : volcano
     * 5 : obsidian
     * 6 : dirt
     */
    public static final int water = 0;
    public static final int grass = 1;
    private static final int[] terrain = {0, 1, 2, 3, 4, 5, 6};
    private double pSand = 0.5; //probabilite qu'une herbe soit sable
    private double pWater = 0.3; //probabilite que du water soit dans le terrain
    private double pTree = 0.01;
    private double pCactus = 0.1;
    private double pThunder = 0.01; //Probabilite qu'un agent fasse un coup de tonnerre
    private double pLavaNoise = 0.15; //Bruit affectant les agents
    //Variables en rapport aux degats sur chaque agents
    public static final int fireDamage = 10;
    public static final int thunderDamage = 50;
    //A ne pas modifier
    private int currentRange = 0; // Variable permettant de savoir si l'agent est dans une zone de danger
    private int currentSandFill = 0; //Assure une appariance réaliste pour les zones sableuses
    private boolean newCycle = false; //Nouveau cycle du jeu
    private boolean newCycleLastStep = false; //Dernier pas du cycle
    private int nbWater=0; //Compte le nombre d'eau
    private int nbSand=0; //Compte le nombre de sable
    private int nbGrass=0; //Compte le nombre d'herbe
    private boolean perlinReady = false; //Vérifie que la table de perlin est prête
    private int[][] perlinTable;
```