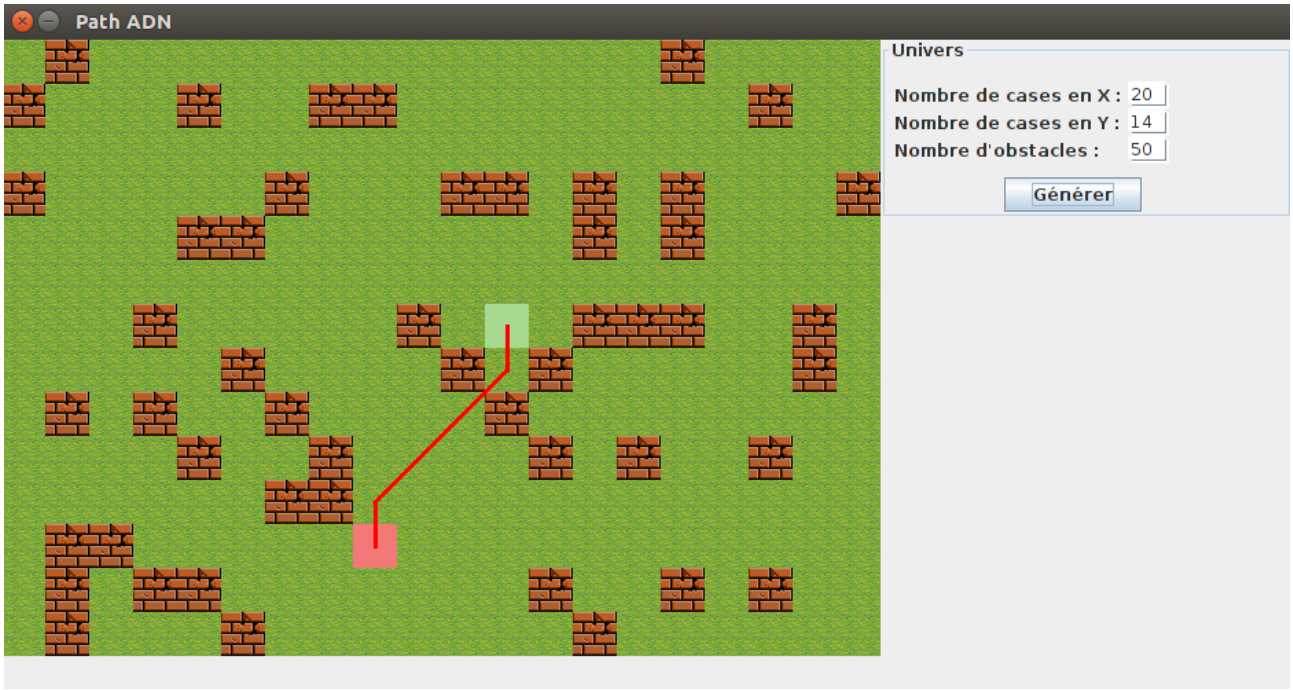
ABDESSELAM Gaïa
BERTRAND Lucas

## Introduction




L'objectif du projet est de trouver le chemin le plus court entre un point d'arrivé et un point de départ en utilisant les algorithmes génétiques.

Tout d'abord, un algorithme génétique se base sur les principes de Darwin :
→Un groupe d'individus représente une population qui va essayer de survivre à son environnement. Chaque individu de cette population ne possède pas les mêmes caractéristiques et donc certains arriveront mieux à s'adapter que d'autres.
→ Les plus faibles meurt tandis que les plus forts survivent.
→ Les survivants se reproduisent entre eux pour donner une génération encore meilleure.

## Modélisation

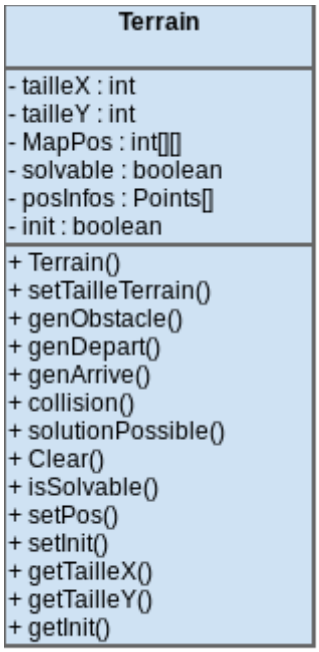
### 1. Univers

Notre univers (terrain), sera constitué d'un point d'arrivé, d'un point de départ et d'obstacles.
Nous avons choisit d'utiliser des sprites de taille 32*32 pour représenter chacun des éléments cités ci-dessus :

 Point de départ de l'Adn  Point d'arrivé  Obstacle

Cet univers est représenté par une matrice 2D qui contiendra tout les éléments de notre terrain. Il sera donc facile de détecter la présence d'obstacles aux alentours de notre ADN.

ABDESSELAM Gaïa
BERTRAND Lucas




## 2. Adn

Plusieurs modélisations étant possibles, nous avons choisit celle-ci :

→ Un adn représente un individu qui sera caractérisé par ses gènes. Ceci est stocké dans une liste d'entier
→ Un gène correspond à un déplacement entre 0 et 7 (c'est son orientation)
→ La capacité de l'ADN correspond à la taille de son chemin, c'est à dire le nombre de déplacement qu'il effectue pour arriver du point de départ au point d'arrivé.

La génération d'ADN se fait aléatoirement par la création d'une suite de séquence de gènes.

Ensuite il faut évaluer un ADN, il faut établir une fonction qui permettra de différencier les meilleurs adn de notre population.

Cette fonction prendra en compte le chemin de l'adn, les obstacles de l'univers et le point d'arrivé. A l'instant T, si lors de l'évaluation l'adn n'est toujours pas arrivé au point final il faut établir une estimation.

ABDESSELAM Gaïa
BERTRAND Lucas

Fonction d'évaluation :

*Initialisation : tailleChemin = 0, GenesMax = 100*

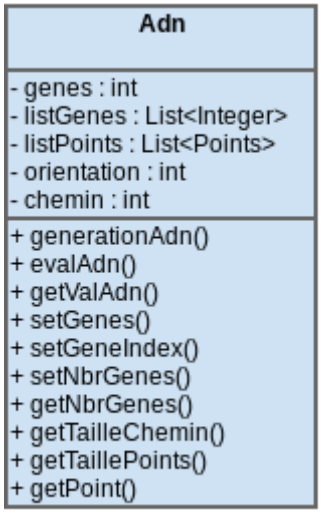
```
EvalAdn( Adn, Univers, PointArrive)
    Tant que Adn.Coordonnes != Point Arrive et tailleChemin < GenesMax Faire

        Si ADN.gene(i) == HAUT Alors
            Adn.DeplacementHaut () ;
        Si ADN .gene(i) == BAS Alors
            Adn.DeplacementBas() ;
        Si ADN .gene(i) == GAUCHE Alors
            Adn.DeplacementGauche() ;
        Si ADN .gene(i) == DROIT Alors
            Adn.DeplacementDroit() ;
        Si ADN .gene(i) == HAUTGAUCHE Alors
            Adn.DeplacementHautGauche() ;
        Si ADN .gene(i) == HAUTDROIT Alors
            Adn.DeplacementHautDroit() ;
        Si ADN .gene(i) == BASGAUCHE Alors
            Adn.DeplacementBasGauche() ;
        Si ADN .gene(i) == BASDROIT Alors
            Adn.DeplacementBasDroit() ;
        FIN SI


         Si Adn.Position == HorsUnivers ou Adn.Position == Obstacle Alors
            Adn .Position = PositionPrecedente
        Fin si
        Incremente tailleChemin de +1

        Si tailleChemin == GenesMax Alors
            tailleChemin += nombreCaseEntre(Adn.Position, Point Arrive)
        Fin tant que

    Renvoyer tailleChemin
```



ABDESSELAM Gaïa
BERTRAND Lucas

## 3. Population

Une population correspond à un groupe d'adn, nous l'avons donc représenté par une liste d'ADN.

Celle-ci doit se reproduire et s'évoluer pour engendrer une génération plus performante. Une reproduction entraîne aussi une mutation de l'ADN pour créer de nouveau individu et donc d'éviter une convergence dès le début de l'algorithme.

Nous avons implémenter deux méthodes différentes pour la sélection et la reproduction d'une population :

## 4. Sélection par survivants

Pour la sélection, on se fixe d'abord un nombre de survivants. On évalue chaque individu de la population et on les trie selon leurs aptitudes à résoudre la solution, les meilleurs se trouveront en tête de liste. Ensuite les survivants se reproduiront entre eux jusqu'à atteindre le nombre limite d'adn de la population.

### 4.1. Reproduction par coupe

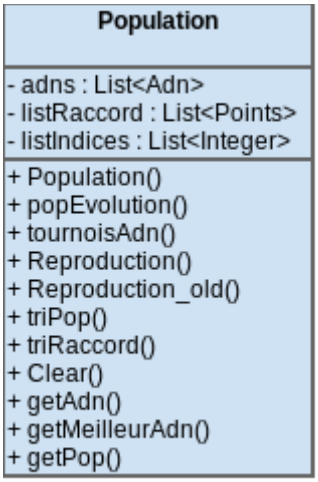
Le principe est simple le fils aura une partie de l'adn du père et une partie de l'adn de la mère. On se fixe que le fils aura l'adn du père avant le point de rencontre et l'adn de la mère après ce point.
Il faut donc trouver un point de coupe de façon a ce que le fils obtienne le meilleur chemin. On peut très bien le choisir aléatoirement mais nous avons opté pour un système de raccord. En effet, on stocke chaque point de rencontre entre le père et la mère dans une liste de point puis après on trie cette liste de façon a obtenir le meilleur point de rencontre. Ce trie etablie la distance entre le point actuel et le point d'arrivé.

```
Reproduction( Univers, Pere, Mere)
        Pour i allant de 0 à min(nombreAdn(Pere),nombreAdn(Mere) faire
                Si Pere(i) == Mere(i) alors
                        ajouter Point(i) dans listPoint
                fin si
        fin pour
        pointdeCoupe = meilleurPoint(listPoint)
        Si (!vide(listPoint)) alors
                pour i allant de 0 à pointdeCoupe faire
                        fils.adn(i) = pere.adn(i)
                fin pour
                pour i allant de pointdeCoupe à nombreAdn(Mere) faire
                        fils.adn(i) = mere.adn(i)
                fin pour
        fin si
```

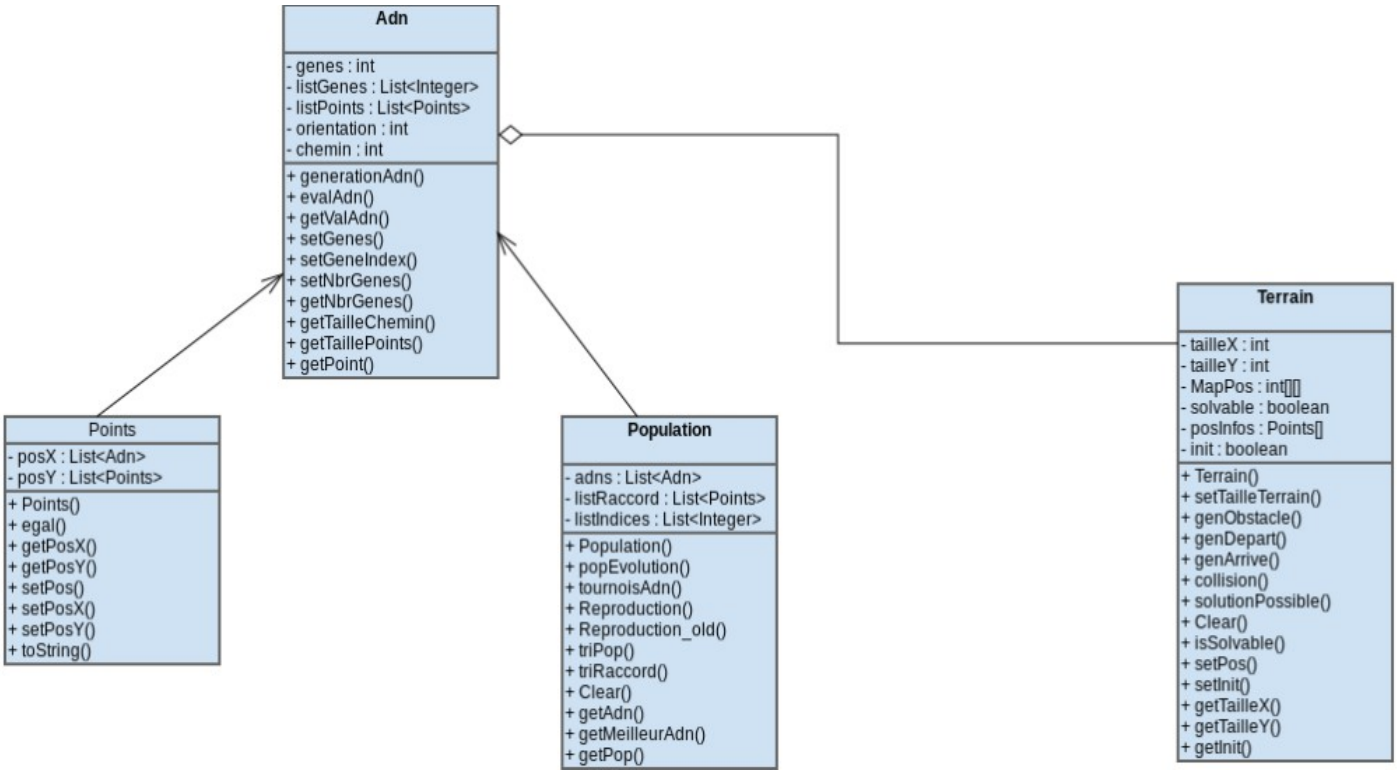
ABDESSELAM Gaïa
BERTRAND Lucas

## 4.2. Sélection par tournois

On sélectionne aléatoirement X personnes de la population et parmi ces individus on en sélectionne le meilleur. On réitère deux fois cette opération pour obtenir un père et une mère potentielle.




## 5. Modélisation UML



ABDESSELAM Gaïa
BERTRAND Lucas

## 6. Partie graphique

Concernant l'interface graphique, nous avons choisit d'utiliser la librairie SWING.
On peut choisir le nombre de case du terrain ainsi que le nombre d'obstacle directement depuis la fenêtre, il est aussi possible de générer autant d'univers que voulue.
L'interface se rafraîchit a intervalle de temps régulier et affiche par conséquent le chemin le plus court toutes générations confondues.

## 7. Tests

- L'univers est de dimension 20*14
- Les points de départs et d'arrivés sont définit aléatoirement
- Le nombre de gène d'un individu est fixé à 100

| Départ | Arrivé | Nombres Obstacles | Taille Chemin | Nombre Générations |
|---|---|---|---|---|
| (18;6) | (10;13) | 50 | 12 | 42 |
| (5;5) | (4;0) | 10 | 5 | 24 |
| (2;0) | (19;6) | 90 | 28 | 48 |
| (15;0) | (10;9) | 50 | 11 | 211 |
| (1;12) | (10;7) | 150 | 9 | 844 |

Dans la majorité des cas, la convergence vers la meilleure solution se fait en un cinquantaine de générations mais il existe tout de même des configurations ou un millier de génération ne suffisent toujours pas.

Il est quand même préférable d’utiliser un algorithme déterministe pour ce type d'application car la solution est en général trouvée plus rapidement.

## 8. Réparation du travail

Le projet est assez court en terme de code en effet, a part l'évolution et la reproduction, le reste est tout à fait trivial. Nous avions donc choisit de chacun implémenter sa propre vision des choses au sujet des deux principales fonctions (évolution / reproduction).

Lucas :
- Interface graphique + Affichage du chemin
- Evolution par système de tournois
- Reproduction

Gaia :
- Gestion de l'univers
- Gestion de l'adn
- Evolution par survivants
- Reproduction