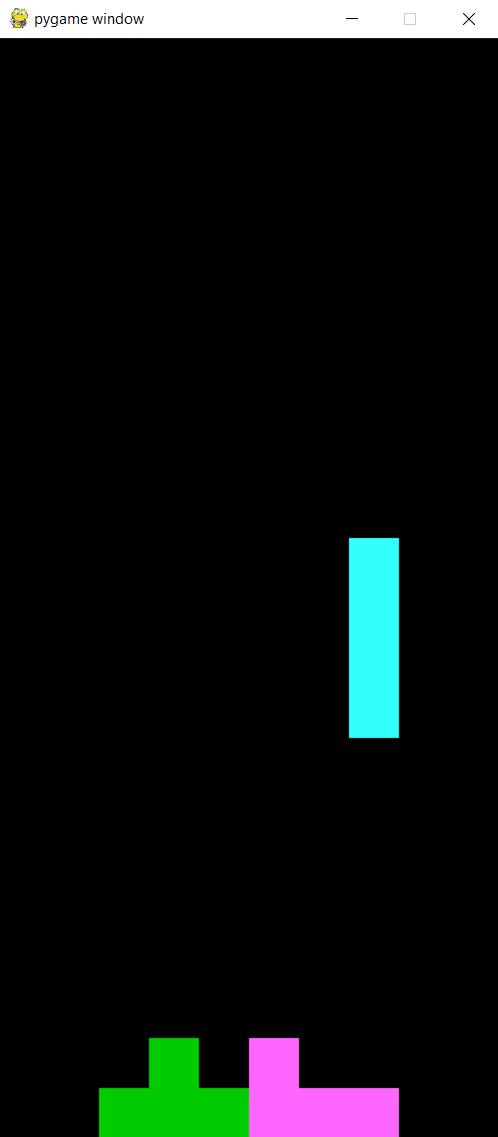
# Introduction

Tetris est un jeu 2d où le but est d’empiler des pièces de différentes formes. L’objectif est de compléter des lignes pour faire des points et pour supprimer la ligne en question. Le jeu s’arrête une fois que la grille de Tetris est complète et que l’on ne peut plus poser de pièce.

En étant le 3ème jeu vidéo le plus vendu de l’histoire, Tetris s’impose comme un incontournable dans son domaine. D’apparence plutôt minimaliste et simple Tetris est un vrai challenge pour les joueurs compétitifs. C’est pourquoi dans ce travail, nous allons essayer de réaliser le meilleur algorithme possible, pouvant ainsi battre les meilleurs joueurs au monde. La création de cet algorithme est un réel défi, car le jeu Tetris utilise l’aléatoire pour générer les pièces. Par conséquent, l’algorithme doit être capable de s’adapter à n’importe quelle configuration. Nous allons commencer par recréer le jeu entièrement en python puis construire un algorithme génétique qui joue au jeu. Finalement, nous allons créer une interface web pour voir l’algorithme en action.

# Création du jeu Tetris



J’ai codé le jeu en python à l’aide de Visual Studio. Mon jeu est basé sur un tableau de valeur 2d. Chaque valeur du tableau correspond à une case dans la grille de Tetris.

Dans un premier temps, j’ai codé le Tetris sans interface graphique pour qu’il soit le plus performant possible avec mon algorithme génétique. J’ai ensuite rajouté une interface graphique très simple pour visualiser la grille de Tetris à l’aide de la librairie « pygame ». Cette librairie, inclue dans python, permet de récupérer les touches claviers du joueur ainsi que de créé une fenêtre de visualisation.

**Actions :**

J’ai créé la librairie « Tetriminos » qui me permet de gérer toutes les pièces possibles du jeu. Cette librairie permet notamment d’instancier, de translater et d’effectuer une rotation de la pièce.

La pièce apparaît au milieu en haut de la grille. Grâce à un timer la pièce descend jusqu’à rentrer en collision avec le sol ou une autre pièce.

Le joueur peut effectuer 3 actions pendant qu’une pièce est en train de descendre :

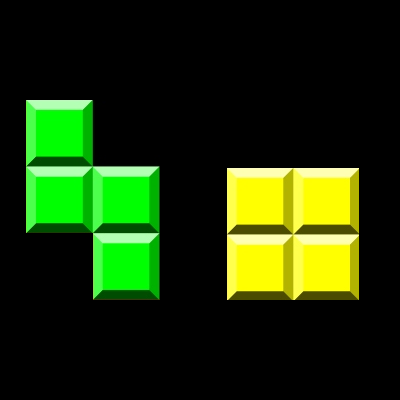
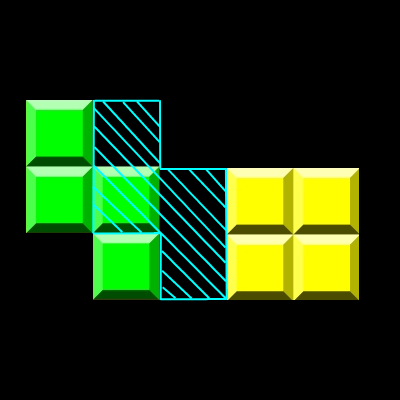
* Translater la pièce à gauche ou à droit s’il n’y a pas collision avec un mûr.
* Effectuer une rotation de 90° de la pièce si l’espace le permet.
* Accélérer la vitesse de descente de la pièce

Avant qu’une pièce n’effectue une action sur la grille, l’action est soumise à un test de collision. Il y a deux types de collision :

* Celles avec les limites de la grille comme le sol ou encore les parois.
* Celles avec les autres pièces déjà posées.

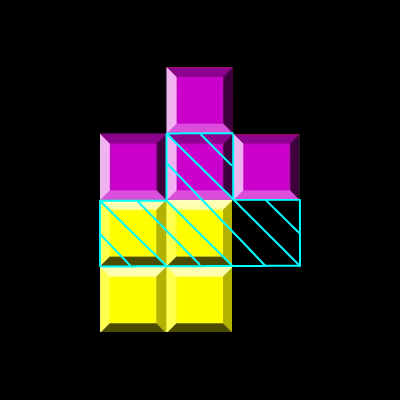
Pour effectuer ce test, la pièce va être projetée sur sa position d’arriver. Le jeu va alors tester si la projection de cette pièce va chevaucher une autre pièce ou être en dehors des limites de la grille.

Exemple avec un déplacement sur la droite :



Dans ce cas ci-dessus, la projection en bleu rayée ne chevauche pas d’autre pièce quelle même. L’action peut donc être validée et effectuée, car il n’y a pas de collision.

Une pièce ne s’arrête que complétement quand il y a une collision en dessous de celle-ci.

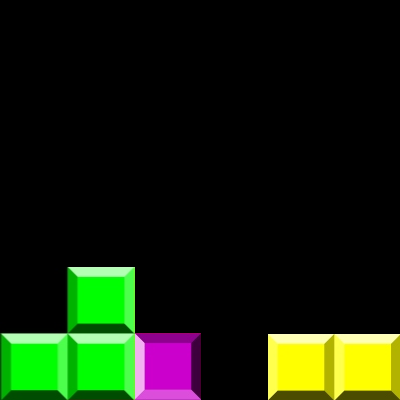


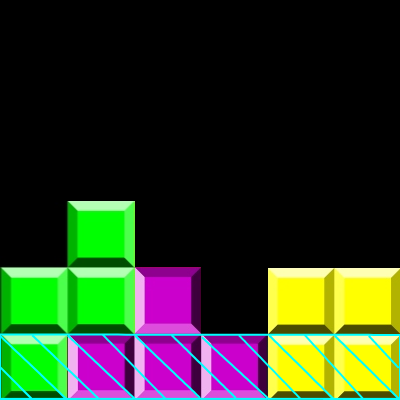
Dans l’exemple à gauche, il y a collision par le bas. Cela veut dire que l’action ne peut pas être effectuée. En plus de cela, la pièce va être figée. Une autre pièce va être instanciée en haut de la grille.

Le processus se répète en boucle jusqu’à que le joueur perde.

## **Score :**

Sur Tetris, on augmente son score de 1 en remplissant une ligne complète avec des pièces. Chaque fois qu’une pièce rentre en collision avec le sol, le jeu va tester si une ligne a été complétée. Si une ligne a été complétée, alors elle va être supprimée et le score du joueur va augmenter de 1.





+1 de score

## **Fin du jeu :**

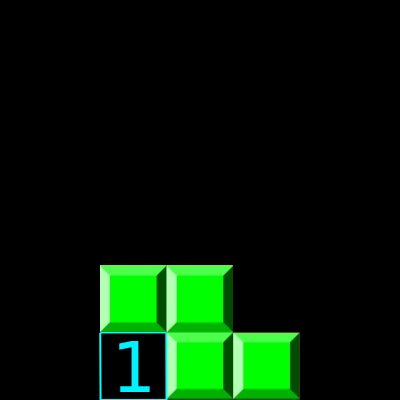
Le jeu s’arrête quand une pièce ne peut plus être instanciée car elle chevauche une autre pièce. Le score est enregistré et le joueur peut recommencer une partie.

# Création de l’algorithme

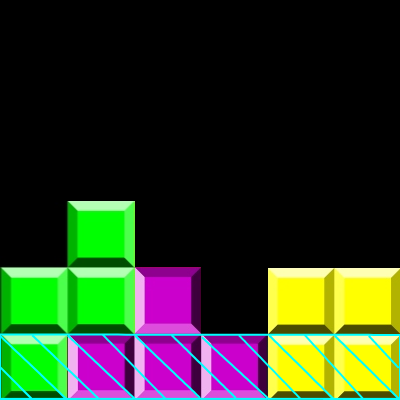
## **Algorithme de scoring**

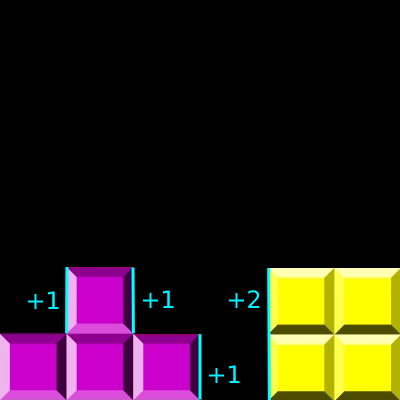
En premier lieu, nous allons implémenter un algorithme de scoring au Tetris. Un algorithme de scoring est une méthode qui permet de déterminer le meilleur positionnement d’une pièce dans la grille.

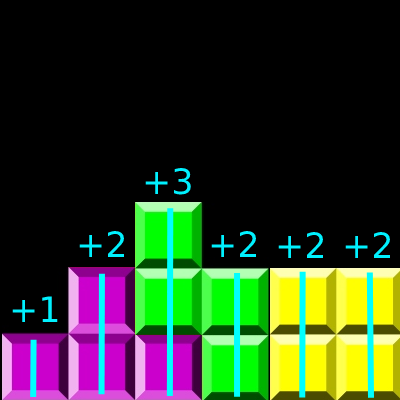
Pour cela, il faudra tester toutes les possibilités de placement d’une pièce et leur attribuer un score. Le score est attribué en fonction de différents paramètres qui permette de réduire le risque de perdre et d’augmenter la chance de compléter des lignes.

Les quatre paramètres ci-dessous sont le résultat de plusieurs recherches afin de ne garder que les paramètres pertinents :

* Le nombre de trous généré par la pièce. L’algorithme doit minimiser les trous pour pouvoir complété des lignes. Dans l’exemple, ci-contre, la pièce engendre 1 trou donc le paramètre « trou » est de 1.



* Le nombre de ligne compléter avec un coup. L’algorithme doit essayer d’augmenter le score. Exemple à gauche, la pièce jaune et violette sont déjà placé sur la grille et la verte est testé. Cette pièce aura un score de 1 dans « ligne complété ».
* La rugosité du terrain. Dans Tetris, il est plus simple de placer des pièces si le terrain est le plus plat possible. On calcule la rugosité en faisant la somme des différences de hauteur entre chaque colonne. Dans l’exemple à droite, le paramètre « rugosité » est de 5.



* La hauteur globale. Pour éviter de perdre dans Tetris, il faut essayer de rester le plus bas dans la grille. On calcule la hauteur globale en faisant le somme de la hauteur totale de chaque colonne. Dans le cas ci-contre où le paramètre « hauteur globale » est de 12.

L’algorithme en itérant toutes les possibilités de mouvement va trouver une valeur score pour chaque mouvement. Pour avoir le meilleur coup possible, on doit minimiser les paramètres « trou », « rugosité » et « hauteur globale ». On veut maximiser le paramètre « ligne complétée » pour que l’algorithme augmente le score général.

On se retrouve donc avec une fonction a minimisé :

Score par pièce = (trou + rugosité + hauteur globale) - ligne complétée

En appliquant cet algorithme de scoring sur notre jeu Tetris, nous obtenons sur 100 parties jouées une moyenne de 86 lignes complétées.

## **Connaissance de la prochaine pièce :**

Dans un jeu Tetris classique, le jeu affiche la prochaine pièce qui va être instanciée. Pour améliorer notre algorithme de scoring nous pouvons tester toutes les possibilités de positionnement en prenant compte cette prochaine pièce. Ceci augmente considérablement le nombre d’itération de test, car les deux-pièces doivent être placés dans toute les positions.

Cette méthode améliore grandement l’algorithme qui passe de 86 lignes complétées à 359 en moyenne.

## **Algorithme génétique**

Le problème avec l’algorithme de scoring est que chaque paramètre on le même poids. Tous les paramètres ne sont pas égaux, par conséquent, il faut pondérer plus les paramètres les plus importants.

Score par pièce = (trou\*poidstrou + rugosité\*poidsrugosité + hauteur globale\*poidshauteurglobal) - ligne complété\*poidslinecomplété

C’est là que l’algorithme génétique rentre en jeu. Cet algorithme va déterminer les poids de nos paramètres. Il va optimiser ces valeurs pour que l’algorithme atteigne le score le plus élevé.

L’algorithme génétique permet d’optimiser des valeurs en s’inspirant des principes de l’évolution décrit par Charles Darwin. Cet algorithme permet de s’approcher de valeur qui réponde à un problème sans connaitre de méthode précise pour résoudre ceux-ci. Cet algorithme génétique fait partie de la classe des algorithmes évolutionnistes qui permet d’approximer des réponses à certains problèmes.

Dans la nature, les différentes espèces animales sont constamment confrontés à leurs environnements. Par exemple, une population de rongeurs est menacée par des rapaces. Dans la population de rongeurs ce sont les plus lent et les plus faibles qui vont se faire manger. Par conséquent les rongeurs les plus forts et rapides vont se reproduire entre eux et à leur tour créé une descendance rapide et forte.

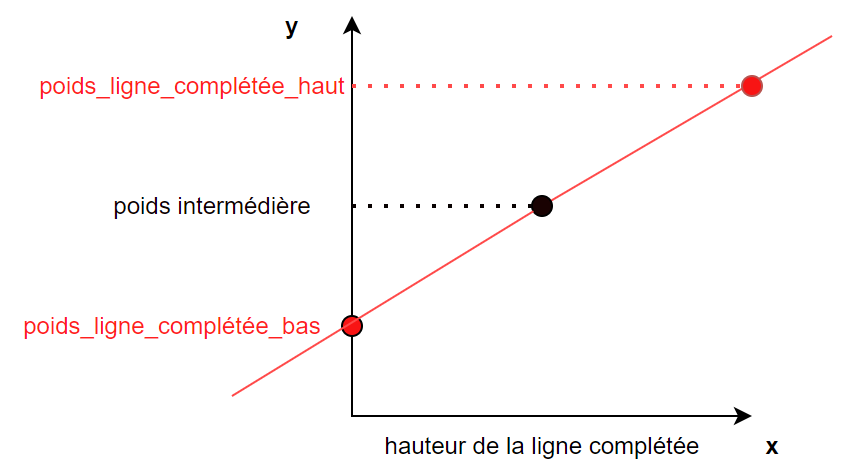
Nous allons transposer ce principe à notre jeu Tetris et à notre algorithme de scoring. Les rongeurs seront des parties de Tetris, nous déterminons si une partie de Tetris est bien en fonction du nombre de ligne quelle a pu compléter.

Chaque rongeur de notre population a des aptitudes différentes. Par exemple, certains rongeurs peuvent être plus rapide, plus légers ou encore plus intelligents. Tous ces aptitudes sont responsables de leurs survies, si elles ne sont pas performantes le rongeur se fait alors mangés.

Nous transposons les aptitudes des rongeurs en paramètre pour notre algorithme de scoring. Chaque individu, est égale à une partie de Tetris, aura ses propres paramètres de pondération.

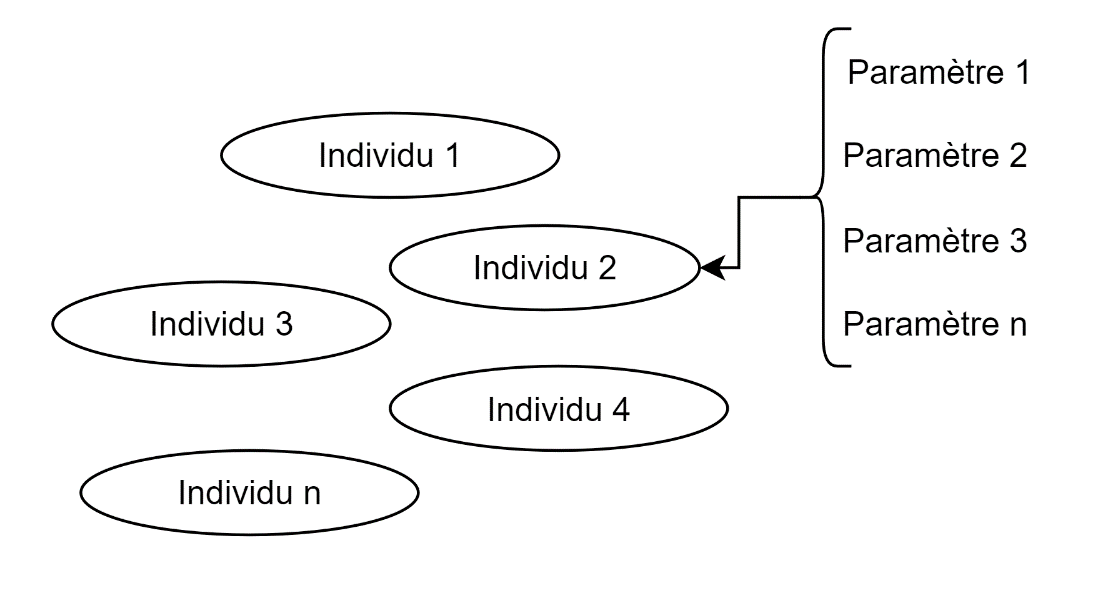
Pour aider l’algorithme et le rendre plus performant, nous constatons que compléter des lignes selon la hauteur de la structure est important. Par exemple, compléter une ligne est très important lorsque la grille de Tetris est quasiment remplie car si on ne complète pas de lignes l’algorithme perds. Pour que l’algorithme accorde plus d’importance au ligne complété en hauteur que celle en bas de la grille nous allons changer le paramètre « poids\_ligne\_complétée ».

Nous allons séparer ce paramètre en 2. Un poids pour les lignes complété dans le bas de la grille et un poids pour les lignes compléter en haut de la grille. Pour les lignes complétées entre ces 2 points (tout en haut et tout en bas de la grille de Tetris), le poids sera une valeur entre valeur entre nos deux paramètres (poids\_ligne\_complétée\_bas, poids\_ligne\_complétée\_haut).

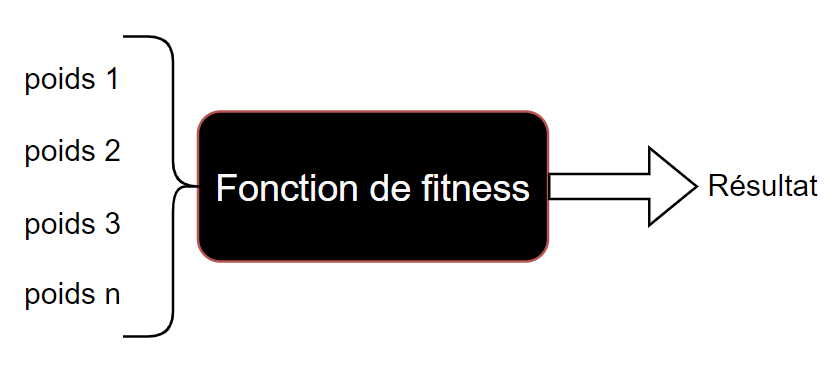
Ce poids sera calculé de la manière suivante :

On tire une droite entre 2 points. Point 1, en y nous avons « poids\_ligne\_complétée\_bas » et en x nous avons 0 car la hauteur pour la ligne complété la plus basse de la grille est 0. Point 2, en y nous avons « poids\_ligne\_complétée\_haut » et en x nous avons la hauteur maximale de notre grille.

Par exemple, si la hauteur de la ligne complétée est de 0 le poids de cette ligne complété est de « poids\_ligne\_complétée\_bas » si la hauteur de la ligne complétée est de x alors notre poids sera égal à y (voir graphique si dessus).

Cet algorithme fonctionne de la manière suivante, nous avons une population d’individu avec différents paramètres. Dans notre cas les individus sont les parties de Tetris qui possède chacune leurs propres pondérations. La population est l’ensemble des parties de Tetris joués pour des paramètres différents entre chaque partie.

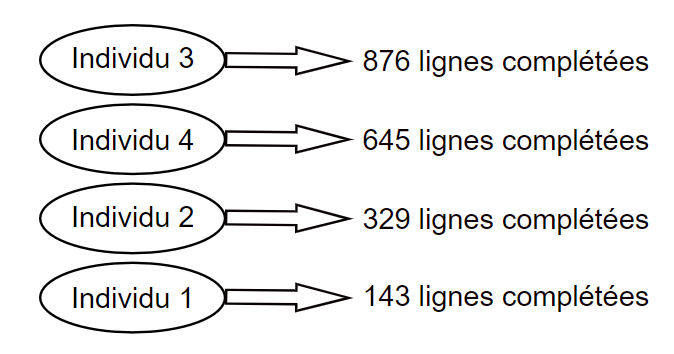
On génère une population de n individu qui possède les paramètres à optimiser. Dans notre cas, les différents poids (poidstrou, poidsrugosité, poidshauteurglobal, poidslinecomplété). Les poids sont attribués aléatoirement dans une plage de nombre entre 0 et 1.

Une fois les pondérations attribuées à chaque individu, nous allons passer cette population dans une fonction de fitness. Une fonction de fitness permet d’évaluer chaque individu pour savoir lesquels sont les mieux adaptés. Dans notre cas, quels individus font les meilleurs scores à Tetris. Nous allons itérer chaque individu dans cette fonction de fitness pour déterminer quel individu est le plus fort à Tetris.

Nous allons représenter la fonction de fitness par F(a,b,c,d) = y. A,b,c,d étant les paramètre de l’algorithme de scoring. Y étant le nombre de ligne complété par l’algorithme. Et la fonction F() étant une partie de Tetris joué par l’algorithme avec les paramètres a,b,c,d.

Comme Tetris est un jeu aléatoire la fonction de fitness ne retourne pas tout le temps la même valeur pour un individu. C’est pour ça qu’on effectue plusieurs fois la fonction de fitness sur un même individu. On en fait une moyenne qui va être plus précise.

Une fois que tous les individus sont passés dans la fonction de fitness, ils sont classés par leurs scores du plus petits au plus grands :



Avec le classement de nos individus, nous allons effectuer l’étape de la sélection. La sélection consiste à sélectionner des individus et à faire reproduire certains individus. Dans le but de crée une nouvelle population d’individu mieux adaptés et plus performant. Pour reproduire deux individus entre eux, nous faisons la moyenne de leurs paramètres pour créer un individu hybride entre ces deux parents.

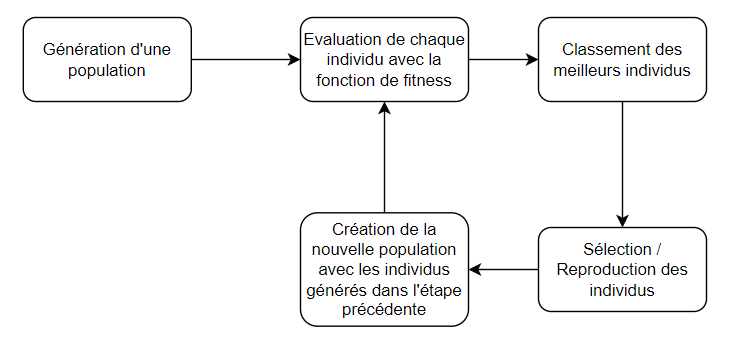
- Nous sélectionnons les 10% meilleurs individus de la population pour la population 2.

- Nous reproduisons 10% des meilleurs individu avec l’individu le mieux évolué de la population pour la population 2

- Nous reproduisons des individus aléatoirement de la population. Nous faisons en sorte qu’il y ait plus de chance qu’un individu avec un haut score soit sélectionner en pondérant la chance de tirage. Avec les individus issus de cette reproduction nous ajoutons 30% d’individu en plus à la population 2.

- Nous sélections 10% des individus de la population aléatoirement en pondérant la chance de tirage pour que les individus possédant un haut score ce face tiré plus facilement. Et nous ajoutons ces individus à la population 2.

-La population 2 est complétée à 60%. Pour ajouter de la variété dans notre population 2 et espéré qu’elle apporte une amélioration nous allons générer complétement aléatoirement les 40% individus restant.

La population 2 une fois générée, nous pouvons recommencer ce processus depuis la fonction de fitness. Pour à nouveau évalué chaque individu de la population 2 puis les sélectionner et reproduire. Nous répétons ce processus autant de fois qu’il faut pour que les individus s’améliorent de plus en plus.

Chaque fois qu’une nouvelle population est générée, nous appelons ça une génération. L’algorithme génétique va itérer plusieurs générations. Chaque génération va en théorie être meilleure que la précédente donc les individus s’améliorent de générations en générations. L’algorithme génétique va alors nous retournez l’individu et ses paramètres qui ont fait à le plus de de lignes complétées en moyenne.

C’est ces paramètres optimisés que nous allons utiliser dans notre algorithme de scoring.

# Création du serveur web

Pour pouvoir visualiser facilement et voir l’algorithme de scoring avec les paramètres optimisés par l’algorithme génétique en action, j’ai créé un serveur web.

J’ai recrée entièrement mon jeu Tetris et l’algorithme de scoring en JavaScript avec un support visuel en html et CSS. J’ai fourni à l’algorithme de scoring en JavaScript les valeurs trouvées par l’algorithme génétique en python car python est bien plus performant pour ce genre de processus.

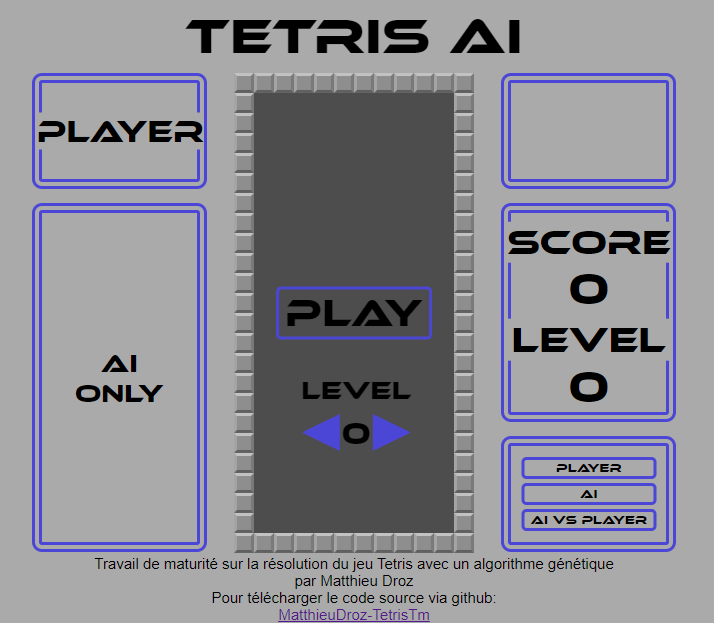
Après avoir tout recodé et optimisé pour le langage JavaScript, j’ai implanté mon algorithme de scoring de manière à en faire un jeu.

Pour cette version web, j’ai créé 3 mode de jeu Tetris :

* Mode « PLAYER », permet au joueur de joué une partie de Tetris classique. Dans ce mode de jeu, le score est le nombre de lignes complétées. Il y a plusieurs niveaux qui augmente selon le score. Plus notre score est élevé plus notre niveau augmente. Chaque fois que le niveau augmente, le jeu augmente de vitesse. C’est-à-dire que la pièce de Tetris va tomber plus vite. Par conséquent, il faut réfléchir plus vite pour mieux la placée sur la grille.

Affichage de la prochaine pièce

Nom du mode de jeu



Score (lignes complétées)

Commencer la partie

Niveau qui augmente en fonction du score. En se basant sur les niveaux de Tetris officiel1.

Niveau de départ du jeu. Pour ce challenger, on peut commencer à n’importe quel niveau.

Sélection des modes de jeu

\*1 https://tetris.fandom.com/wiki/Tetris\_(NES,\_Nintendo)

# Conclusion

# Source

<https://app.diagrams.net/>

<https://fr.wikipedia.org/wiki/Charles_Darwin>

<https://codemyroad.wordpress.com/2013/04/14/tetris-ai-the-near-perfect-player/>

<https://www.youtube.com/watch?v=l_KY_EwZEVA>

<https://www.youtube.com/watch?v=pXTfgw9A08w>

<https://fr.wikipedia.org/wiki/Algorithme_g%C3%A9n%C3%A9tique>

<https://www.youtube.com/watch?v=ncj_hBfRt-Y>

<https://www.youtube.com/watch?v=Lu0QoPFfhnE>