RÉSOLUTION DU JEU TETRIS AVEC UN ALGORITHME GÉNÉTIQUE

DROZ MATTHIEU
2022-2023
3MG04

MENTOR : BRUCHEZ CHRISTOPHE
LYCÉE DENIS-DE-ROUGEMONT

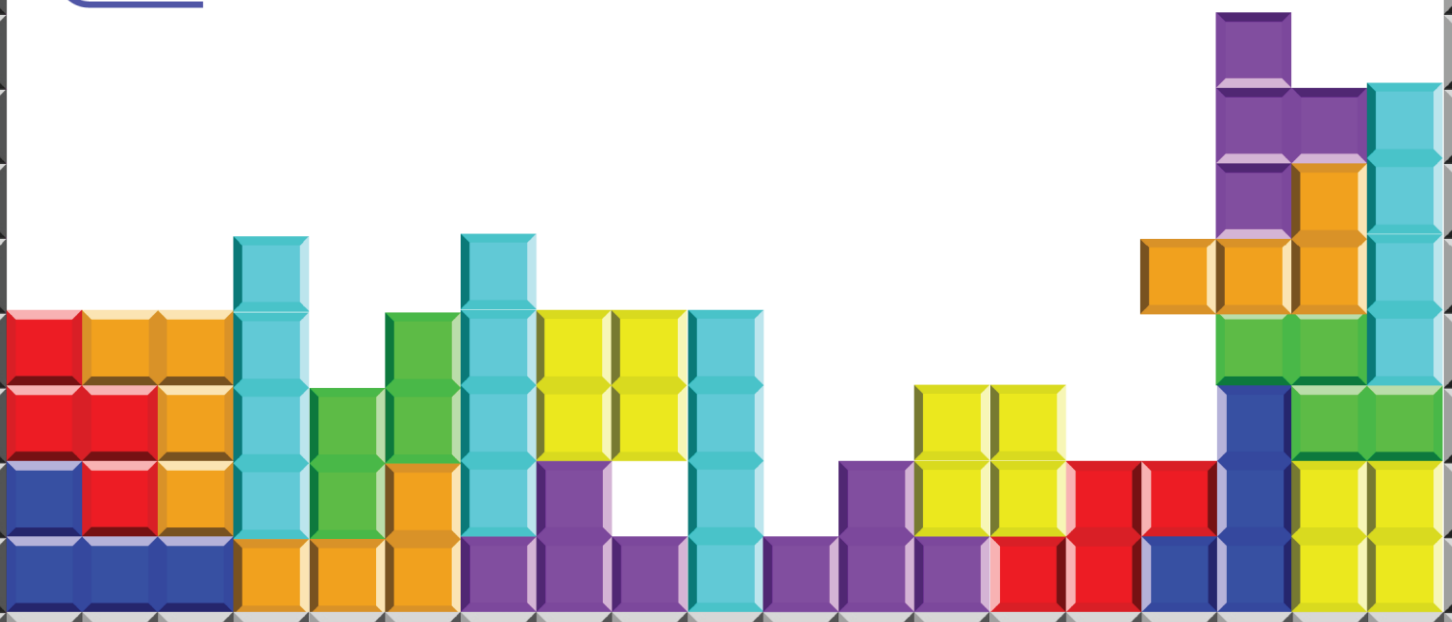


Table des matières

Lien du site internet :
http://tetris.matthieudroz.com/

Lien pour télécharger le travail de maturité :
https://github.com/BuseThai/MatthieuDroz-TetrisTm

1 Introduction

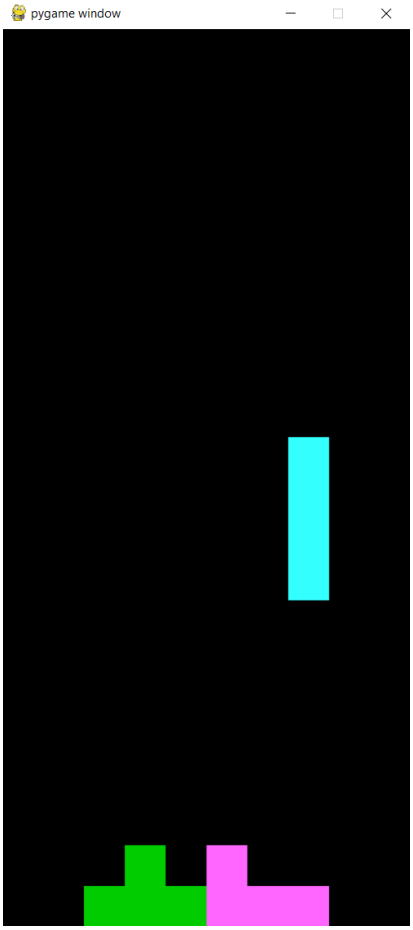
Tetris est un jeu 2D dont le but est d'empiler des pièces de formes différentes. L'objectif du jeu est de compléter le plus de lignes possibles. Chaque ligne complétée permet d'obtenir un point et ensuite la ligne est supprimée. La suppression de cette ligne permet de créer un nouvel espace afin de pouvoir poser les pièces suivantes. Le jeu s'arrête une fois que la grille de Tetris est complète et que l'on ne peut plus poser de pièce. Etant 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 représente toutefois un vrai challenge pour les joueurs compétitifs.

Ce travail a pour but de réaliser le meilleur algorithme pouvant ainsi battre les joueurs de Tetris les plus performants au monde. La création de cet algorithme est un réel défi puisque 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.

Etant amateur de jeux vidéo et de programmation depuis plusieurs années, j'ai décidé de choisir un travail de maturité pouvant allier ces deux domaines. Ainsi, je vais recréer le jeu Tetris et le coupler avec un programme me permettant de battre les records établis par les humains pour ce jeu.

Pour commencer, je vais entièrement recréer le jeu Tetris en utilisant le langage Python. Ensuite, je construirai un algorithme de scoring qui évaluera chaque position possible d'une pièce sur la grille. Pour compléter cette deuxième phase, je mettrai en place un algorithme génétique qui optimisera le résultat de l'algorithme de scoring. Finalement, je vais concevoir une interface web pour observer l'algorithme en action.

2 Création du jeu Tetris



J'ai codé le jeu en Python à l'aide de Visual Studio Code. 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 jeu sans interface graphique pour qu'il soit le plus performant avec l'algorithme génétique. J'ai ensuite ajouté une interface graphique très simple pour visualiser la grille de Tetris à l'aide de la librairie « pygame [1]». Cette librairie, inclue dans Python, permet de récupérer les touches clavier du joueur ainsi que de créer une fenêtre de visualisation.

2.1 Actions

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

Il faut noter que la pièce apparaît toujours en haut et au milieu de la grille. Grâce à un timer, la pièce descend jusqu'à entrer 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 droite pour autant qu'il n'y ait pas collision avec un mûr ou une autre pièce
- Effectuer une rotation de 90° de la pièce si l'espace le permet
- Accélérer la vitesse de descente de la pièce (nommé « hard drop »)

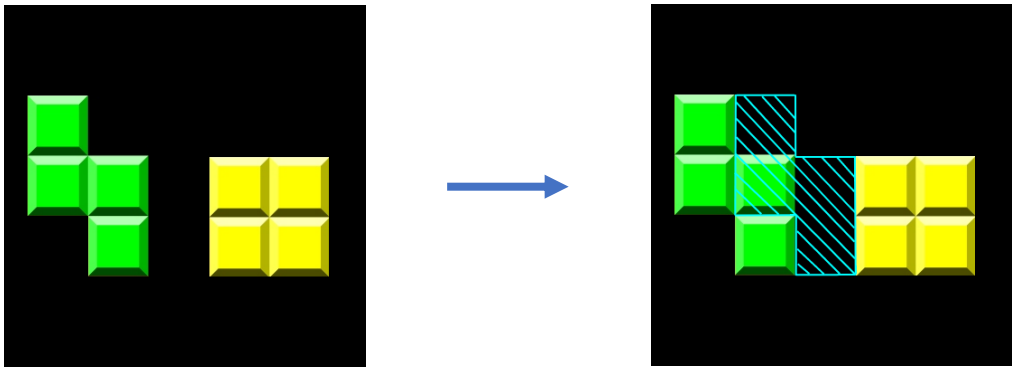
Avant qu'une pièce n'effectue une action sur la grille, l'action est soumise à un test de collision. Il existe deux types de collision :

- 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'arrivée. 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.

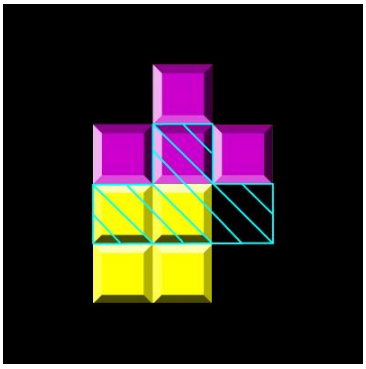
[1] PYGAME, Librairie pygame, https://www.pygame.org/news, consulté le 10 mai 2022

Exemple avec un déplacement sur la droite :



Dans le cas ci-dessus, la projection (en bleu hachuré) 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 s'arrête définitivement lorsqu'il y a une collision en dessous de celle-ci.

Dans l'exemple à ci-contre, il y a collision par le bas pour la pièce violette. Cela veut dire que l'action ne peut pas être effectuée et que la pièce va être figée. Dès lors une autre pièce va être instanciée en haut de la grille. Le processus se répète en boucle jusqu'à que le joueur perde.



2.2 Score

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

2.3 Fin du jeu

Le jeu s'arrête lorsqu'une pièce ne peut plus être instanciée car elle chevauche une autre pièce. Le score est enregistré et la partie se termine. Le joueur peut alors recommencer une autre partie.

3 Création des algorithmes

A ce stade, je vais créer deux algorithmes qui se compléteront pour essayer de jouer indéfiniment à Tetris.

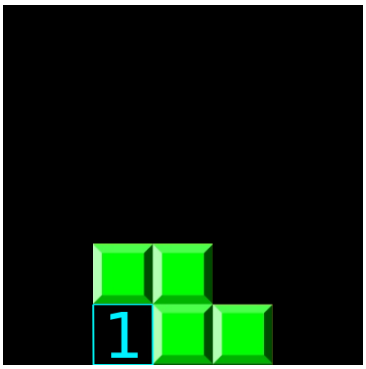
3.1 Algorithme de scoring

En premier lieu, je vais implémenter un algorithme de scoring au Tetris. L'algorithme de scoring est une méthode qui permet de déterminer le meilleur positionnement d'une pièce dans la grille.

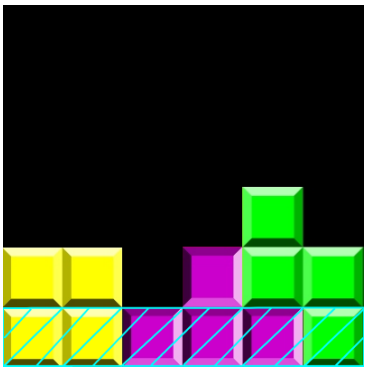
Pour cela, il faudra tester toutes les possibilités de placement d'une pièce et lui attribuer un score. Le score est attribué en fonction de différents paramètres qui permettent d'une part de réduire le risque de perdre et d'autre part d'augmenter les chances de compléter des lignes.

Après plusieurs recherches, j'ai sélectionné quatre paramètres qui me semblaient les plus pertinents étant donné qu'ils permettent de déterminer si le positionnement de la pièce est optimal.

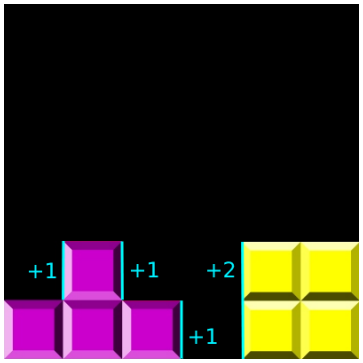
- 1er paramètre :
  Le nombre de trous généré par la pièce
  L'algorithme doit minimiser les trous pour pouvoir compléter des lignes. Dans l'exemple, ci-contre, la pièce engendre 1 trou dont le paramètre « trou » est de 1.



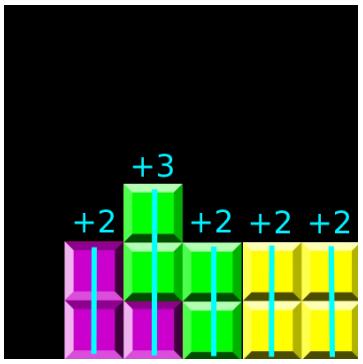
- 2e paramètre :
  Le nombre de lignes complétées avec un coup
  L'algorithme doit essayer de maximiser le score. Dans l'exemple à droite, les pièces jaunes et violettes sont déjà placées sur la grille et la verte est testée. Cette pièce aura un score de 1 dans « ligne_complétée ».



- 3e paramètre :
  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.



- 4e paramètre :
  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 la somme de la hauteur totale de chaque colonne. Dans le cas ci-contre, le paramètre « hauteur_globale » est de 11.



En itérant toutes les possibilités de position d'une pièce, l'algorithme va trouver une valeur de score pour chaque mouvement. Pour obtenir le meilleur positionnement, il faut minimiser les paramètres « trou », « rugosité » et « hauteur_globale » et maximiser le paramètre « ligne_complétée » afin que l'algorithme augmente le score général. Il faut ainsi minimiser la fonction :

$$Score\ par\ pièce = (trou + rugosité + hauteur\_globale) - ligne\_complétée$$

En appliquant cet algorithme de scoring au jeu Tetris, sur 100 parties jouées, on obtient une moyenne de 86 lignes complétées.

3.2 Connaissance de la prochaine pièce

Dans un jeu Tetris classique, la prochaine pièce qui va être instanciée est affichée. Pour améliorer l'algorithme de scoring, nous pouvons tester toutes les possibilités de positionnement en tenant compte de la pièce actuelle et de la pièce suivante. En effet, ceci augmente considérablement le nombre d'itération de tests, car les deux pièces doivent être placées dans toutes les positions.

Cette méthode améliore grandement l'algorithme puisque le jeu passe, sur une moyenne de 100 parties jouées, de 86 à 359 lignes complétées.

3.3 Algorithme génétique

Le problème rencontré avec l'algorithme de scoring est que chaque paramètre a le même poids. Cependant, tous les paramètres ne sont pas égaux. Dès lors, il faut pondérer en conséquence les paramètres les plus importants.

$$Score\ par\ pièce = (trou * poids\_trou + rugosité * poids\_rugosité + hauteur\_globale * poids\_hauteur\_globale) - (ligne\_complétée * poids\_ligne\_complétée)$$

C'est à ce stade que l'algorithme génétique entre en jeu pour déterminer les poids des paramètres. Il va optimiser ces valeurs pour que l'algorithme de scoring atteigne le nombre de lignes complétées le plus élevé.

L'algorithme génétique s'inspire des principes de l'évolution décrit par Charles Darwin. Il permet de s'approcher de valeurs qui répondent à un problème sans connaître de méthode précise pour le résoudre. Il fait partie de la classe des algorithmes évolutionnistes qui permettent d'approximer des réponses à certains problèmes.

On trouve ce principe dans la nature où les différentes espèces animales sont constamment confrontées à leur environnement. A titre d'exemple, on peut prendre une population de rongeurs menacée par des rapaces. Dans la population de rongeurs, ce sont les plus faibles et les plus lents qui vont se faire manger. Par conséquent, les rongeurs les plus forts et les plus rapides vont se reproduire entre eux et ainsi créer, à leur tour, une descendance plus résistante. De plus, on constate que chaque rongeur de notre population a des aptitudes différentes. Par exemple, certains rongeurs peuvent être plus rapides, plus légers ou encore plus intelligents. Toutes ces aptitudes sont responsables de leur survie ou non. Si elles ne sont pas performantes, alors le rongeur se fait attraper et manger. Ces aptitudes sont définies par les gènes du rongeur.

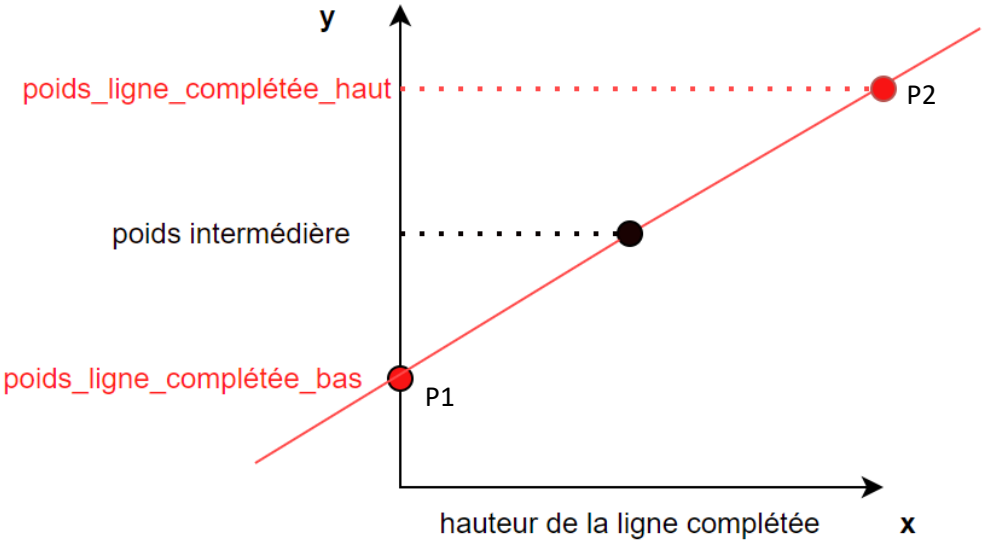
On peut transposer ce principe à une partie de jeu Tetris. Plus la pondération des paramètres de l'algorithme de scoring est optimale plus le nombre de lignes complétées est élevé. Une partie de Tetris possède ainsi des paramètres qui peuvent être assimilés aux gènes des rongeurs. On détermine alors si une partie de Tetris est performante en fonction du nombre de lignes complétées.

3.3.1 Poids variable pour les lignes complétées

J'ai constaté que plus la structure est haute plus il est important de compléter des lignes. Dès lors, il est primordial de compléter une ligne lorsque la grille de Tetris est quasiment remplie sinon l'algorithme perd. Pour que l'algorithme accorde plus d'importance aux lignes complétées en hauteur que celles du bas de la grille, il faut changer le paramètre « poids_ligne_complétée ».

Nous allons diviser ce paramètre en deux parties. Un poids pour les lignes complétées du bas de la grille et un poids pour les lignes complétées en haut de la grille. Pour les lignes complétées entre ces deux points, le poids sera une valeur comprise entre nos deux paramètres (poids_ligne_complétée_bas, poids_ligne_complétée_haut).

Ce poids est calculé de la manière suivante :



Voici comme il faut procéder. On tire une droite entre 2 points (P1, P2). Le point P1 sur l'axe y est égal à « poids_ligne_complétée_bas » et sur l'axe x à 0 car la hauteur pour la ligne complétée la plus basse de la grille est 0. Le point P2 sur l'axe y est égal à « poids_ligne_complétée_haut » et sur l'axe x à la hauteur maximale de notre grille.

Cette droite correspond à notre fonction pour déterminer le poids des lignes complétées :

$$F(h, \mathrm{pb}, \mathrm{ph}) = poids\_ligne\_complétée$$

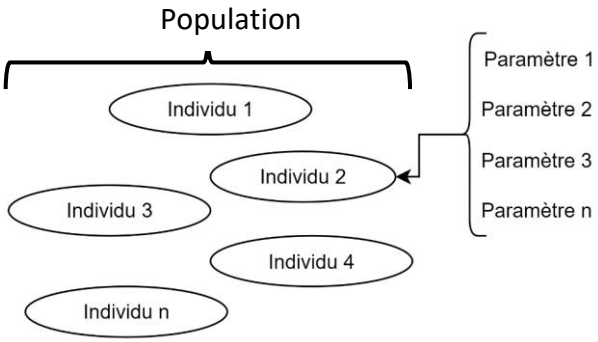
h = hauteur de la ligne complétée

pb = poids_ligne_complétée_bas

ph = poids_ligne_complétée_haut

3.3.2 Génération de la population

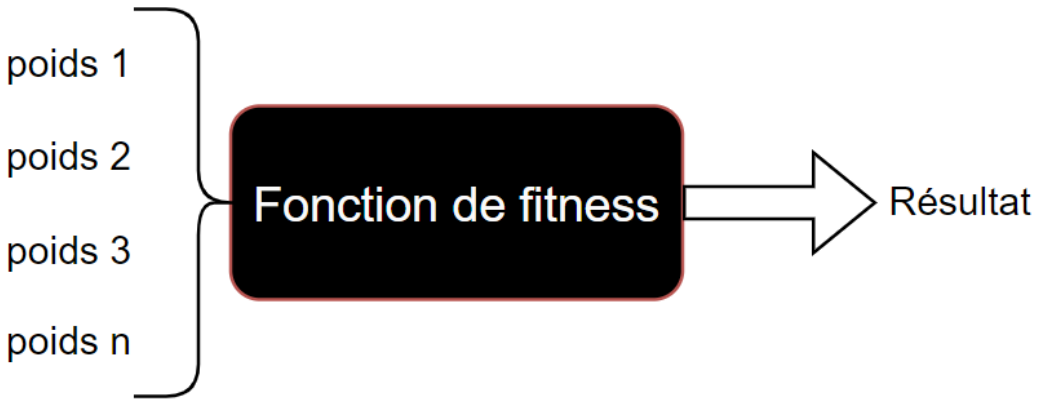
Voici comment l'algorithme génétique fonctionne. Nous avons une population d'individus avec différents paramètres. Dans notre cas, les individus sont les parties de Tetris avec l'algorithme de scoring qui possède chacun leur propre pondération. La population correspond à l'ensemble des individus.



On génère une population de n individus qui possède les paramètres à optimiser. Dans notre cas ce sont les différents poids (poids_trou, poids_rugosité, poids_hauteur_globale, poids_ligne_complétée_bas, poids_ligne_complétée_haut). Les poids sont attribués aléatoirement dans une plage de nombre compris entre 0 et 1.

3.3.3 Fonction de fitness

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, cela revient à connaître quels individus font les meilleurs scores à Tetris. Nous allons ensuite itérer chaque individu dans cette fonction de fitness pour déterminer quel individu est le plus performant à Tetris.



Nous allons représenter la fonction de fitness par :

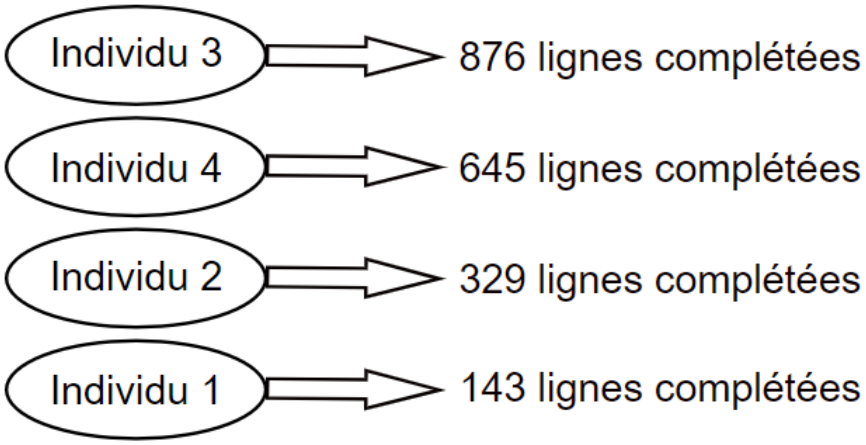
$$F\ (a, b, c, d, e)\ =\ y$$

- a, b, c, d, e étant les pondérations de l'algorithme de scoring
- Y étant le nombre de lignes complétées par l'algorithme
- La fonction F() étant une partie de Tetris jouée par l'algorithme de scoring avec les paramètres a, b, c, d, e

Comme Tetris est un jeu aléatoire, la fonction de fitness ne retourne pas tout le temps la même valeur pour un individu. C'est la raison pour laquelle il faut effectuer plusieurs fois la fonction de fitness sur un même individu. Ensuite pour obtenir un résultat plus précis on calcule une moyenne des résultats.

3.3.4 Classement des individus

Une fois que tous les individus sont passés dans la fonction de fitness, ils sont classés, selon leur score, du plus petit au plus grand.



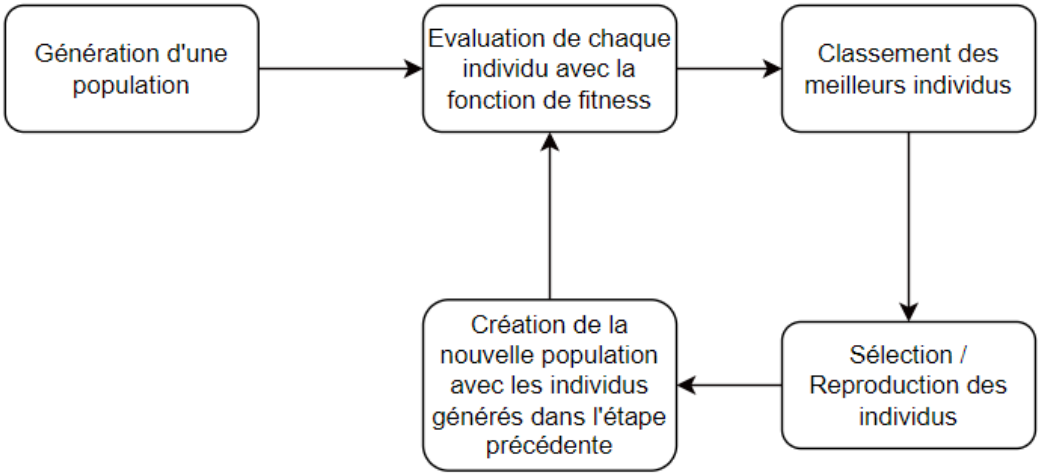
3.3.5 Sélection et croisement des individus

Le classement de nos individus permet de passer à l'étape de la sélection. La sélection consiste à choisir des individus et à les faire se reproduire dans le but de créer une nouvelle population d'individus mieux adaptés et plus performants. Pour croiser deux individus, il faut calculer la moyenne de leurs paramètres. Ainsi, une nouvelle population est générée à partir de la première et a pour objectif de posséder des individus plus performants. Pour ce faire, il faut croiser entre eux les individus les mieux classés. A cela, il faut ajouter de manière aléatoire certains individus pour apporter du brassage génétique. En effet, le brassage génétique sert à ajouter de la nouveauté dans la population.

Pour générer une nouvelle population, les méthodes suivantes sont utilisées :

- Sélectionner 10% des meilleurs individus de la population primaire
- Reproduire l'individu le plus performant avec le 10% des meilleurs individus de la population primaire
- Reproduire aléatoirement des individus de la population primaire. Il faut faire en sorte qu'il y ait plus de chance qu'un individu avec un haut score soit sélectionné en pondérant la chance de tirage. Avec les individus issus de cette reproduction nous ajoutons 30% d'individus à la nouvelle population
- Sélectionner aléatoirement 10% des individus de la population primaire en pondérant la chance de tirage pour que les individus possédant un haut score soit choisi prioritairement. Ajouter ces individus à la nouvelle population
- Générer aléatoirement les 40% d'individus restant pour obtenir le 100% la nouvelle population

Dès que la nouvelle population est générée, il faut recommencer le processus à partir de la fonction de fitness pour évaluer à nouveau chaque individu de la nouvelle population puis les sélectionner et les reproduire. Ce processus est ainsi répété autant de fois qu'il faut pour que les individus s'améliorent de plus en plus.



Chaque nouvelle population générée est appelée génération. L'algorithme génétique va itérer plusieurs générations. Chaque génération va en théorie être meilleure que la précédente puisque les individus s'améliorent. L'algorithme génétique va alors retourner le meilleur individu et ses paramètres. Ce sont justement ces paramètres optimisés qui sont utilisés dans l'algorithme de scoring.

3.3.6 Entrainement de l'algorithme génétique

Pour entrainer notre algorithme génétique, j'ai choisi les paramètres suivants :

- Population de 100 individus
- Répétition 10 fois de la fonction de fitness pour avoir une moyenne fiable
- Entrainement de l'algorithme génétique avec la connaissance de la prochaine pièce
- Arrêt de la partie après 1000 pièces posées sur la grille. Ce paramètre est présent pour réduire le temps de calcul de l'algorithme

L'algorithme génétique a tourné pendant 43 heures avec une moyenne de temps par génération d'environ 2 heures et demie. Au bout de 18 générations les paramètres optimisés par l'algorithme sont les suivants :

$$poids\_trou = 0{,}38006$$

$$poids\_rugosité = 0{,}23295$$

$$poids\_ligne\_complétée\_haut = 0.58981$$

$$poids\_ligne\_complétée\_bas = 0{,}67119$$

$$poids\_hauteur\_globale = 0{,}47393$$

Ces pondérations sont injectées directement dans l'algorithme de scoring qui a réussi à jouer huit heures d'affilée, sans perdre, atteignant un score de 32'000 points. Théoriquement l'algorithme pourrait jouer à l'infini sans perdre.

4 Création du serveur web

Pour pouvoir visualiser facilement l'algorithme de scoring avec les paramètres optimisés par l'algorithme génétique en action, j'ai créé un serveur web. Cependant, j'ai dû recréer entièrement mon jeu Tetris et l'algorithme de scoring en utilisant le langage 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. En effet, Python est bien plus performant pour ce genre de processus.

Après avoir tout recodé et optimisé pour le langage JavaScript, j'ai implémenté mon algorithme de scoring de manière à en faire un jeu. J'ai également designé une toute nouvelle interface pour rendre le jeu Tetris facile d'utilisation. J'ai finalement recréé une nouvelle charte graphique pour que le jeu soit agréable à jouer.

Mon jeu Tetris est disponible sur le site suivant : tetris.matthieudroz.com

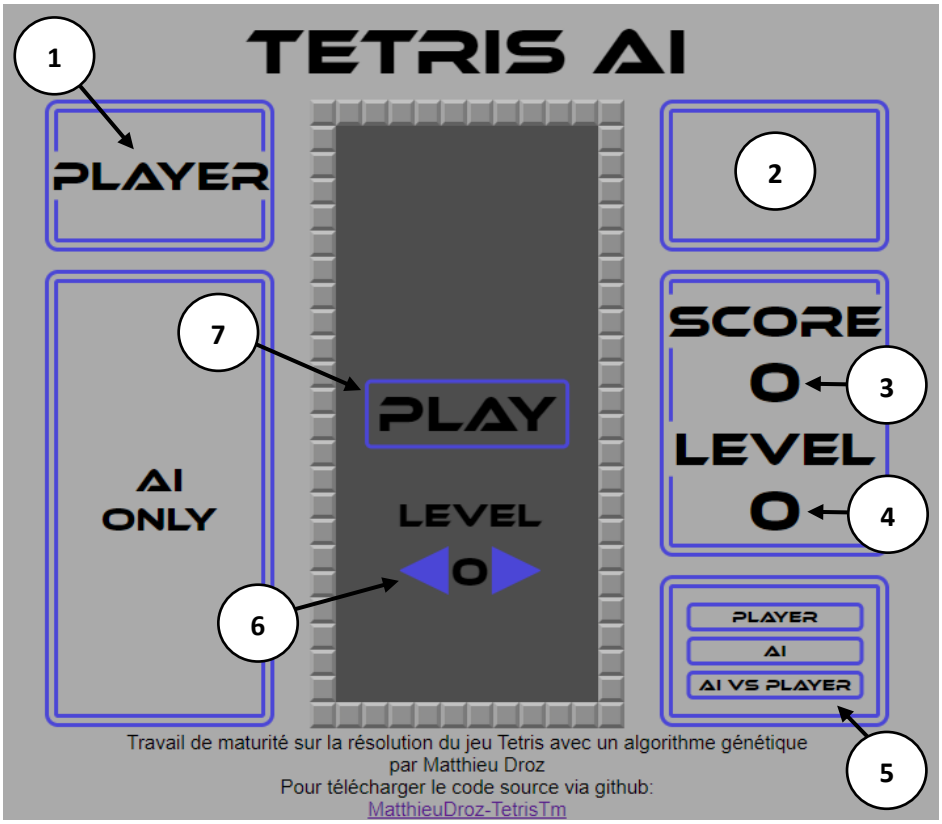
4.1 Modes de jeu

Pour cette version web, j'ai créé 3 modes de jeu Tetris qui se présentent de la manière suivante.

4.1.1 Mode « PLAYER »

Ce mode de jeu permet au joueur d'effectuer une partie de Tetris classique. Le score correspond au nombre de lignes complétées. Plusieurs niveaux sont disponibles. Plus le score est élevé plus le niveau ainsi que la vitesse de jeu augmentent. Cela signifie que la pièce de Tetris tombe plus vite et, par conséquent, qu'il faut réfléchir plus rapidement pour la placer sur la grille.

Description de l'interface :



1. Nom du mode de jeu
2. Affichage de la prochaine pièce
3. Score (= lignes complétées)
4. Niveau qui augmente en fonction du score en se basant sur les niveaux de Tetris officiel[2]
5. Sélection des modes de jeu
6. Niveau de départ du jeu. Pour se challenger, on peut décider de commencer à n'importe quel niveau
7. Commencer la partie

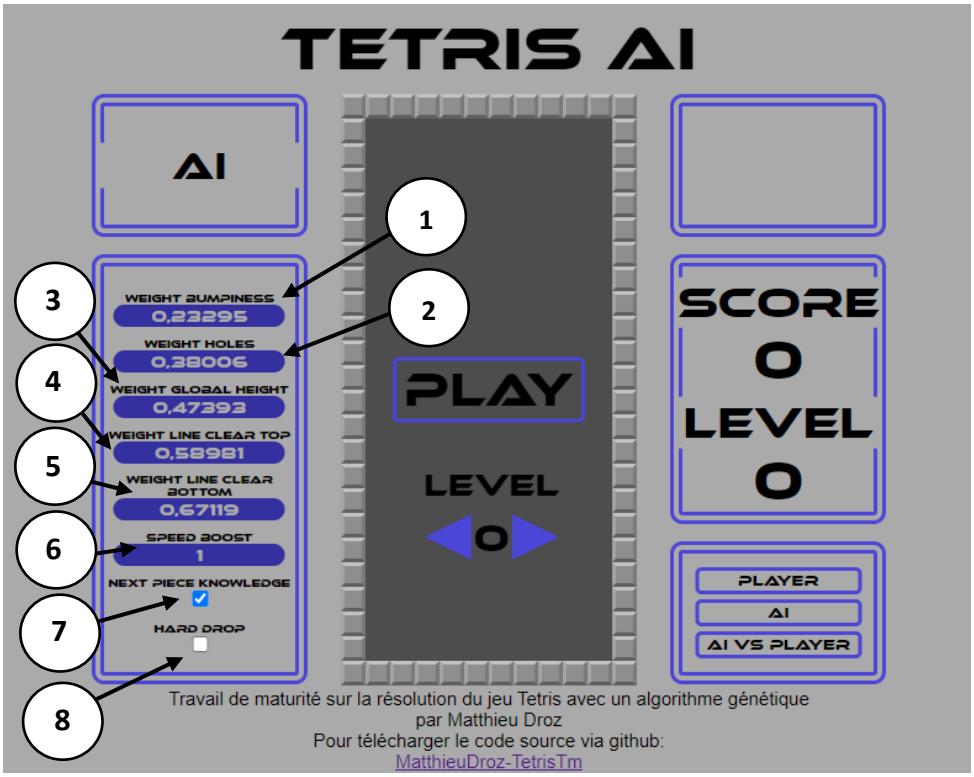
4.1.2 Mode « AI »

Le mode Intelligence artificielle est ici un abus de langage. Dans le cas présent, nous parlons plutôt d'algorithme évolutionniste plutôt que d'une réelle intelligence artificielle. Mais pour le bien du design et de la sobriété, nous avons choisi l'appellation « AI » (Artificial Intelligence). Ce mode permet de visualiser l'algorithme de scoring avec les paramètres optimisés par l'algorithme génétique. Le joueur peut alors voir en temps réel l'algorithme de scoring jouer.

[2] TETRIS FANDOM, Tetris (NES, Nintendo), https://tetris.fandom.com/wiki/Tetris_(NES,_Nintendo) , consulté le 23 septembre 2022

Le joueur peut, avec la sélection des modes de jeu, passer du mode « PLAYER » au mode « AI » et vice-versa tout étant dans la même partie de Tetris. Ce mode débloque une interface qui permet de paramétrer l'algorithme de scoring.

Description de l'interface :

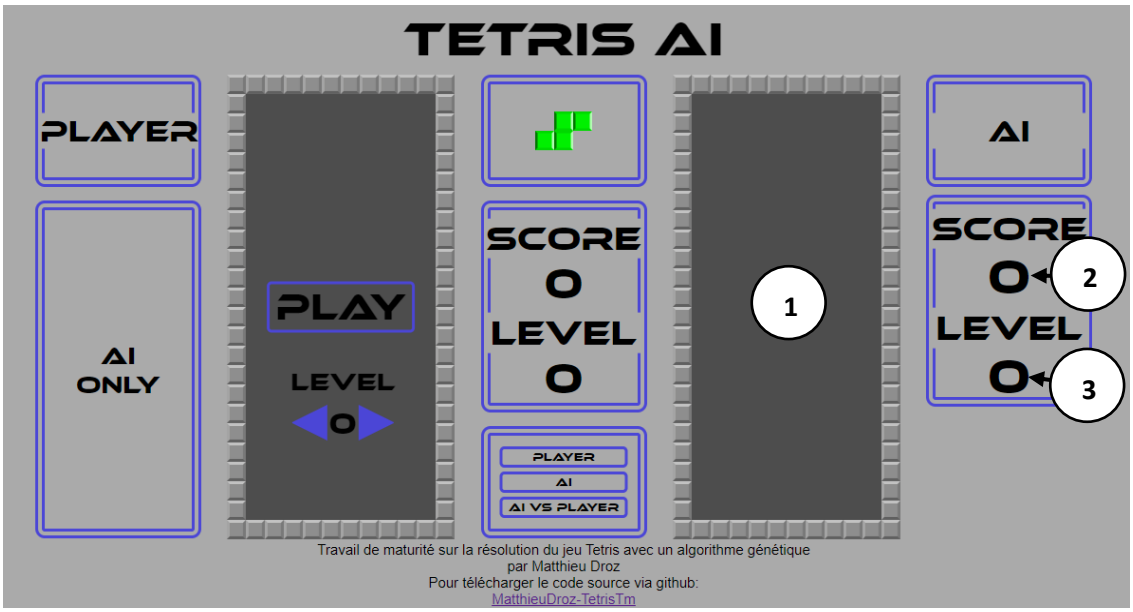


1. Poids_rugosité
2. Poids_trou
3. Poids_hauteur_global
4. Poids_ligne_completée_haut
5. Poids_ligne_completée_bas
6. Multiplicateur de vitesse pour la descente des pièces
7. Si l'algorithme de scoring prend en compte la prochaine pièce ou non
8. Si la pièce descend directement à vitesse maximum

L'algorithme de scoring est paramétré de base avec les valeurs optimales. Cependant, le joueur peut changer toutes ces valeurs pour voir comment l'algorithme de scoring se comporte. Le joueur peut également accélérer la vitesse de descente des pièces. De plus, il peut activer ou désactiver la fonction pour connaître la prochaine pièce.

4.1.3 Mode « AI VS PLAYER »

Comme son nom l'indique, dans ce mode de jeu, le joueur se bat contre l'algorithme de scoring. Ils partagent les mêmes pièces et la même vitesse de descente de ces dernières. Ce mode de jeu a la même interface que celle du mode « PLAYER » et possède en plus une grille Tetris et un tableau des scores.



1. Grille Tetris de l'algorithme de scoring
2. Score de l'algorithme de scoring
3. Niveau de l'algorithme de scoring

4.2 Déploiement

Pour le backend du serveur, j'ai utilisé le langage Python. Avec Python j'utilise la librairie « Flask[3] » de Armin Ronacher qui permet de construire une application web. Flask permet de créer un serveur web sur une port IP défini.

L'application web est hébergée sur un de mes serveurs physiques personnels. L'adresse http de mon site est directement reliée à l'adresse IP où est hébergé mon serveur web.

L'application web a été spécialement conçue pour un ordinateur. A noter que les mobiles n'ont pas accès au site internet.

Le site internet est optimisé pour le moteur de recherche Google Chrome. Les autres moteurs de recherche fonctionnent également mais l'optimisation n'est pas garantie. De plus, l'application web pourrait ne pas fonctionner si la version du moteur de recherche est trop ancienne.

[3] Flask, Flask project, https://flask.palletsprojects.com/en/2.2.x/, consulté le 19 septembre 2022

5 Conclusion

Pour entreprendre mon travail, j'ai débuté par comprendre comment fonctionne le jeu Tetris pour le recréer en Python. Ensuite, j'ai construit un algorithme de scoring qui permet d'évaluer la meilleure position d'une pièce sur la grille du jeu Tetris. Cet algorithme se base sur quatre paramètres : trou, rugosité, hauteur_globale et ligne_complétée. Ces paramètres ne sont pas égaux car certains sont plus importants que d'autres, dès lors il faut les pondérer. Pour pondérer ces paramètres, j'ai utilisé un algorithme génétique qui permet avec un grand nombre de parties jouées d'optimiser chaque poids pour chaque paramètre. Avec ces poids de paramètres, l'algorithme de scoring devient quasiment imbattable.

Pour rendre ce travail plus accessible, j'ai décidé de créer un site web (tetris.matthieudroz.com) qui contient un Tetris avec trois modes de jeu différents. Tout d'abord, le premier mode permet au joueur de jouer à Tetris normalement. Ensuite, le deuxième montre l'algorithme de scoring avec les paramètres optimisés par l'algorithme génétique en action. Et, finalement, le troisième permet de défier l'algorithme de scoring en jouant contre lui.

Dans ce travail, la principale difficulté que j'ai rencontrée à consister à transposer des informations théoriques (notamment pour l'algorithme génétique) en code. De plus, ce genre d'algorithme évolutionniste demande énormément de ressources et de temps de calcul. C'est pour cette raison que j'ai fixé une limite de pièces maximum par partie jouée par l'algorithme génétique. Les résultats que j'ai obtenus sont moins précis mais beaucoup plus rapide. Cela étant dit, les pondérations optimisées par l'algorithme génétique sont quasiment parfaites car l'algorithme de scoring n'a pas perdu une seule fois.

Finalement, ce travail m'a permis de créer un projet conséquent notamment grâce à mes connaissances informatiques acquises en autodidacte durant de nombreuses années et à l'intérêt que je porte à la programmation. J'ai ainsi pu mettre à contribution diverses compétences telles que le développement Web, l'ingénierie réseau et le développement de jeu vidéo afin de réaliser mon travail de maturité.

6 Sources

APP DIAGRAMS, https://app.diagrams.net/, consulté le 19 aout 2022

BAKIBAYEV T, How to write Tetris in Python, 2020, https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

CANNON G, Github, StackRabbit, https://github.com/GregoryCannon/StackRabbit, consulté le 22 juin 2022

CANNON G, Youtube, AI BREAKS NES TETRIS ! – 102 MILLION and level 237, 2021, https://www.youtube.com/watch?v=l_KY_EwZEVA

ELLINGWOOD J et al., How to Serve Flask Applications with Gunicorn and Nginx on Ubuntu 18.04, 2021, https://www.digitalocean.com/community/tutorials/how-to-serve-flask-applications-with-gunicorn-and-nginx-on-ubuntu-18-04

FLASK, https://flask.palletsprojects.com/en/2.2.x/, consulté le 19 septembre 2022

GUILLAUME, Algorithme génétique : Darwin au service de l'intelligence artificielle, 2018, https://toiledefond.net/algorithme-genetique-darwin-intelligence-artificielle/

LAROCHELLE H, Youtube, Intelligence Artificielle [3.4] : Recherche locale – algorithme génétique, 2013, https://www.youtube.com/watch?v=9tCySY6TLk8

LOONRIDE, Github, tetris-ai, https://github.com/knagaitsev/tetris-ai, consulté le 16 octobre 2022

LOONRIDE, Youtube, AI learns to play Tetris, 2018, https://www.youtube.com/watch?v=pXTfgw9A08w

MALLAWAARACHCHI V, Introduction to Genetic Algorithms – Including Example Code, 2017, https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

NEVEU T, Github, introduction, https://github.com/thibo73800/aihub/tree/master/genetic/introduction, consulté le 15 juillet 2022

NEVEU T, Github, zombies, https://github.com/thibo73800/aihub/tree/master/genetic/zombies, consulté le 15 juillet 2022

NEVEU T, Youtube, Comprendre les algorithmes génétiques #1, 2019, https://www.youtube.com/watch?v=ncj_hBfRt-Y

NEVEU T, Youtube, Sélection naturelle contre les zombies – Les algorithmes génétiques #2, 2019, https://www.youtube.com/watch?v=Lu0QoPFfhnE

PHOTOPEA, https://www.photopea.com/, consulté le 2 septembre 2022

PYGAME, https://www.pygame.org/news, consulté le 10 mai 2022

SEYMANUR, Deploying Flask Application on VPS Linux Server using Nginx, 2021, https://medium.com/geekculture/deploying-flask-application-on-vps-linux-server-using-nginx-a1c4f8ff0010

TETRIS FANDOM, Tetris Wiki, https://tetris.fandom.com/wiki/Tetris_Wiki, consulté le 23 aout 2022

WIKIPEDIA, Algorithme Génétique, https://fr.wikipedia.org/wiki/Algorithme_g%C3%A9n%C3%A9tique, consulté le 27 juillet 2022

WIKIPEDIA, Charles Darwin, https://fr.wikipedia.org/wiki/Charles_Darwin, consulté le 20 juillet 2022

WIKIPEDIA, Tetris, https://en.wikipedia.org/wiki/Tetris, consulté le 15 mai 2022

YIYUAN L, Tetris AI – The (Near) Perfect Bot, 2013, https://codemyroad.wordpress.com/2013/04/14/tetris-ai-the-near-perfect-player/