TÉTRISBOT

FRÉDÉRIC MULLER - LIONEL PONTON

Projet maths-infos du DU 2ème année - 2018-2019

INTRODUCTION

TABLE DES MATIÈRES

Première partie

Le moteur de jeu

Chapitre 1

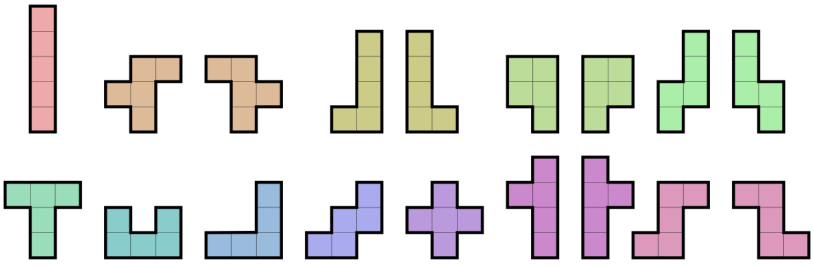
LE JEU TÉTRIS

1 L'origine du jeu

Tetris a été créé par Alekseï Leonidovitch Pajitnov au milieu des années 1980. Il était alors chercheur au centre informatique Dorodnitsyn de l'Académie des Sciences Soviétique, un laboratoire de recherche et développement de l'Union Soviétique. Il y travaillait à la reconnaissance de la voix humaine par les ordinateurs et, à ses heures perdues, s'amuser à programmer des jeux.

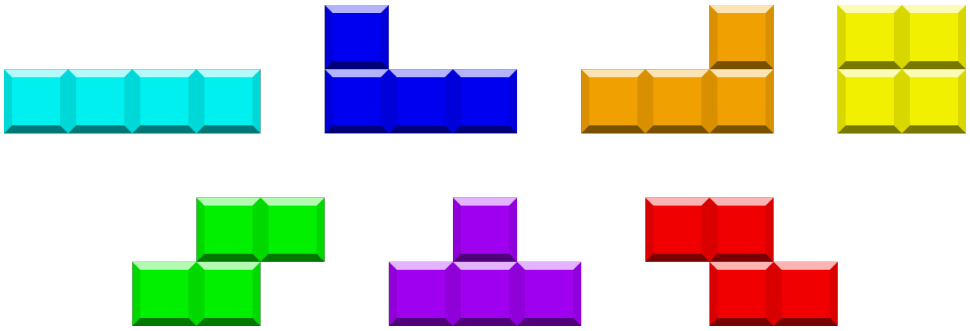
Alekseï Pajitnov
(Source :
`http://allrus.me/legendary-russian-game-programmer-alexey-pajitnov/`)

Il avait entendu parlé d'un puzzle créé par le mathématicien américain Salomon Golomb, le *pentomino*, dont le but était de recouvrir un rectangle avec des pièces de différentes formes, toutes obtenues par l'assemblage de 5 carrés identiques.



Les 18 pièces du Pentomino (en distinguant les pièces symétriques)
(Source : `https://fr.wikipedia.org/wiki/Pentomino`)

Il s'en procura un et, commençant à y jouer, découvrit que, malgré les apparences, ce n'était facile du tout ! Il eut alors l'idée d'en faire un jeu électronique dans lequel les pièces étaient choisies aléatoirement, l'une après l'autre, et à des intervalles de temps de plus en plus réduits. Il fit des tests mais devant le (trop) grand nombre de combinaisons qu'offraient les 18 pièces du pentomino, il décida d'opter pour une version simplifiée dans laquelle il n'y aurait que 7 types de pièces, toutes formées à l'aide de 4 carrés identiques. Ces pièces sont nommés les « tétrominos » [1], du grec *tetra* qui signifie *quatre*.



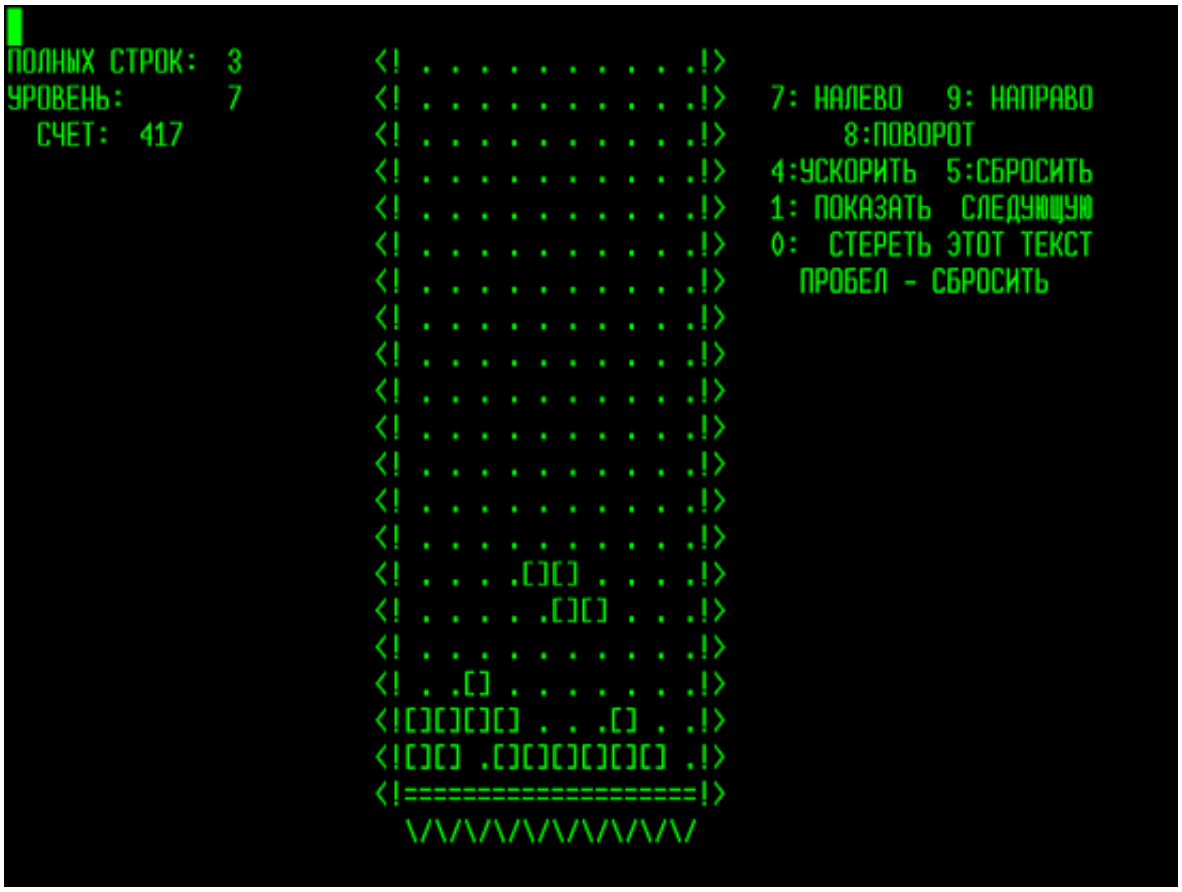
Les 7 tétrominos : I, J, L, O, S, T et Z.
(Source : `https://en.wiktionary.org/wiki/tetromino`)

Le jeu fut développé sur un Elektronica 60. Cet ordinateur ne disposait pas de fonctionnalités graphiques et les carrés formant les pièces furent alors représentés par un espace encadré de crochets : [ ].

2 Le principe du jeu

Le principe du jeu est le suivant : dans un champ de jeu rectangulaire, une pièce est choisie aléatoirement parmi les 7 tétrominos et se déplace du haut vers le bas. Le joueur peut faire tourner d'un, deux ou trois quarts de tour et déplacer horizontalement dans les deux sens la pièce afin de disposer le tétromino comme il le souhaite en bas du champ de jeu, sachant que les pièces successives s'empilent les unes sur les autres. Lorsque le joueur parvient à recouvrir une ligne complète du champ de jeu à l'aide de carrés des tétrominos, sans qu'il n'y ait plus aucun trou sur la ligne, celle-ci disparaît, rapportant un certain nombre de points, et la pile de tétrominos est décalée vers le bas, laissant ainsi plus de place dans le champ de jeu pour accueillir les pièces suivantes. Si, à un moment, le champ de jeu ne contient plus assez de place pour accueillir le tétromino suivant, la partie est perdue ! Ainsi, Tetris n'est pas un jeu dans lequel une partie se termine par la victoire du joueur. Il s'agit plutôt de faire le meilleur score possible.

1. On trouve aussi les appélations « tétraminos » ou « tétriminos ».



Première version de Tetris

(Source : `https://www.firstversions.com/2015/11/tetris.html`)

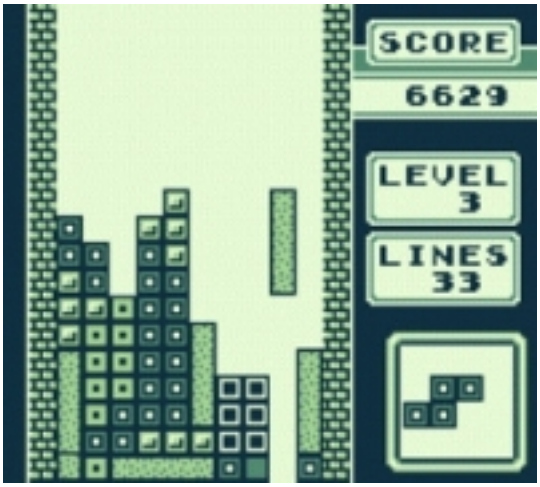
3 Diffusion du jeu

Sentant que le jeu serait plus attractif si les tétrominos apparaissaient « véritablement » à l'écran plutôt que leur représentation à l'aide des crochets, Pajitnov décida d'améliorer l'aspect visuel de son programme grâce à une version pour PC d'IBM intégrant une interface graphique. Pour cela, il fit appel à un jeune lycéen prodige de 16 ans, Vadim Gerasimov, qui lui avait été présenté par un autre programmeur du nom de Dmitri Pavlovsky. Gerasimov avait des compétences qui dépassaient largement celles de Pajitnov et Pavlovsky : il avait notamment appris seul à programmer dans un langage venu de l'Ouest : le Microsoft DOS. À l'issue de deux mois de travail, Gerosimov créa la première version « couleur » de Tetris qui intégrait également une table des meilleurs scores programmée par Pavlovsky. Selon Vadim Gerasimov, le nom Tetris résulte de la contraction des mots « tetramino » et « tennis », ce dernier étant le sport favori de Pajitnov.

Vadim Gerasimov
(Source : `http://www.stuff.co.nz/technology/games/2477794/Tetris-inventor-makes-waves-at-Google`)

Cette version se diffusa rapidement d'abord à Moscou puis dans les pays de l'ancien bloc soviétique. Une copie fut envoyée par Victor Brjabrin, le supérieur de Pajitnov, à l'Institut des Sciences Informatiques de Budapest où Robert Stein, un anglais d'origine hongroise travaillant pour une société de logiciels, découvrit le jeu. Stein vit le potentiel du jeu et envoya un fax au centre informatique Dorodnitsyn pour indiquer qu'il était intéressé par Tetris. Pajitnov lui répondit simplement qu'il était également intéressé et, à partir de cette simple réponse, Stein se mit à exploiter et vendre les droits de Tetris auprès de diverses compagnies occidentales, sans avoir signé aucun accord avec les russes. Par la suite, il voulut établir un contrat en bonne et due forme avec Pajitnov et ses supérieurs mais ceux-ci étant confrontés à un monde inconnu pour eux, l'économie de marché, se montrèrent très méfiants et exigeants et les négociations n'aboutirent pas, ce qui n'empêcha pas Stein de continuer à exploiter Tetris. L'une des premières versions commerciales de Tetris fut celle éditée par Spectrum Holobyte sur PC IBM en 1986.
Par la suite, Elektronorgtechnica (abrégé ELORG), l'agence russe chargée de gérer les importations et exportations informatiques, repris le contrôle des négociations pour la gestion des droits et du marketing de Tetris, reprochant même à Pajitnov d'avoir donné un accord, fut-il de principe, à Stein, étant donné que, pendant l'ère soviétique, les créations intellectuelles des chercheurs russes étaient la propriété de l'état. Finalement, ELORG confirma l'accord avec Stein pour la gestion des droits de Tetris mais seulement sur les ordinateurs et à l'exclusion de tout autre matériel électronique.
En 1989, les droits d'exploitation restants furent partagés entre Atari pour les bornes d'arcade et Nintendo pour les consoles. Minuro Arakawa, le président de la filiale américaine de Nintendo avait mandaté Henk Rogers, afin d'obtenir ces droits car il comptait faire de Tetris l'un des jeux phare de sa nouvelle console : la Game Boy. Celle-ci fut mise sur le marché en 1989 et connue un succès phénoménal avec des millions d'exemplaires vendus à

travers le monde. Si les ventes de la Game Boy participèrent à la diffusion de Tetris, celui-ci contribua également au succès de la console car le jeu était, dans un premier temps, implanté d'office sur la machine.



Le jeu Tetris sur Game Boy en 1989
(Source :
`https://de.wikipedia.org/wiki/Liste_der_erfolgreichsten_Computerspiele`)

Par la suite, Nintendo développa différentes variantes du jeu puis, après de l'effondrement de l'URSS en 1991, Pajitnov émigra aux États-Unis et récupéra finalement l'intégralité des droits de Tetris en 1996. Il fonda avec Henk Rogers The Tetris Compagny qui a depuis la charge exclusive de la gestion des droits du jeu. Ainsi, Pajitnov, qui n'a touché aucun droit d'auteur durant plus de 10 ans, peut aujourd'hui récolter les fruits de sa création.

4 Les raisons du succès

Tetris possède deux caractéristiques qui en font un modèle de jeu vidéo et qui peuvent expliquer son succès : d'une part, il est facile à prendre en main mais, dans le même temps, il possède un côté addictif et, d'autre part, malgré son apparente simplicité, il ne peut exister que sur un support électronique. Le principe du jeu fait qu'il procure une satisfaction quasi immédiate puisqu'on arrive très rapidement à supprimer une première ligne, puis une deuxième, etc... Il fait partie de la famille des « jeux occasionnels » (*casual gaming*) car une partie ne demande pas un gros investissement en terme de temps et on peut y jouer quelques minutes puis arrêter pour refaire une partie plus tard sans n'avoir rien perdu de significatif.
Le choix de Pajitnov de réduire le nombre de pièces à 7 est également un élément déterminant. En effet, 7 est le nombre d'« objets [2] » différents que l'esprit humain peut mémoriser rapidement et sans trop d'effort alors qu'à partir de 8, cela devient beaucoup plus difficile.

2. Il faut comprendre ici objet dans un sens très large : images, mot, idées, nombres, etc...

Ainsi, 7 est le nombre idéal de pièces puisqu'elles peuvent être mémorisées rapidement par le joueur.
Enfin, la simplicité de son programme et le peu de ressources qu'il requiert lui a permis d'être développé sur tous les supports de jeu successifs (ordinateur, borne d'arcade, console, smartphone...)

5 Peut-on gagner à Tetris ?

La principale difficulté dans Tetris est l'accélération de l'arrivée des nouvelles pièces qui devient vite difficilement gérable. Il est clair que si cette accélération continue indéfiniment, on atteint un palier au-delà duquel le jeu n'est plus jouable, ne serait-ce que parce que le temps de déplacement de la nouvelle pièce devient inférieur au temps matériellement nécessaire pour la déplacer à l'aide des touches de la console ou de l'ordinateur.
Imaginons qu'on mette de côté cet aspect du jeu et que les pièces arrivent à intervalle régulier. Dans ce cas, peut-on gagner à Tetris ? Telle quelle la question n'a pas beaucoup de sens puisque le jeu ne prend fin qu'avec la défaite du joueur. Dans son mémoire de master, J. Brzustowski étudie la question sous l'angle suivant : existe-t-il une stratégie qui permette de jouer à Tetris indéfiniment. Il montre que cela dépend essentiellement du choix aléatoire des pièces. En particulier, il prouve que, quelle que soit la stratégie adoptée, il existe une suite de tetrominos S et Z qui conduit inéluctablement à la fin de la partie.
Notons, cependant, que, dans le cas d'un choix au hasard des tetrominos, la probabilité qu'une telle suite sorte est très faible. De plus, Brzustowski met en évidence que le jeu n'est pas programmé pour faire perdre le joueur par le choix des tetrominos. C'est d'ailleurs le contraire puisque, dans la spécification officielle du jeu (*Guideline*) disponible sur le site de The Tetris Compagny [3], il est préciser qu'en fait les pièces sont choisies par vague de 7, une vague consistant en une permutation quelconque des 7 tetrominos : c'est le principe du « sac aléatoire » (*random bag*). Dans cette configuration, des stratégies gagnantes sont possibles mais Brzustowski constate également que face à la réalité du jeu (i.e. en tenant compte des contraintes temporelles), l'expérience acquise pas un joueur est plus efficace qu'une stratégie mathématique prédéfinie.

3. `http://tetris.wikia.com/wiki/Tetris_Guideline`

Chapitre 2

IMPLÉMENTATION DU MOTEUR DE JEU

1 Structures de données

Le moteur de jeu est construit autour de trois classes :

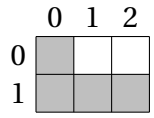
- `Tetramino` : les blocs
- `Board` : la grille de jeu
- `TetrisEngine` : le moteur de jeu qui fait le lien entre les deux classes précédentes

2 La classe Tetramino

La classe `Tetramino` est responsable de le gestion des blocs et de leurs rotations. Elle est implémentée dans le fichier `tetramino.py`.

2.1 Initialisation

La représentation d'un bloc est codée par la liste des coordonnées de ses cases occupées, le coin en haut à gauche ayant pour coordonnées `(0,0)` :
Par exemple, pour le bloc "J" placé horizontalement :



Ce bloc sera codé par la liste `[(0,0), (1,0), (1,1), (1,2)]`

Lors de l'initialisation d'un bloc, on va donner :

- Son `id` qui permettra d'identifier son type si besoin
- La liste de ses rotation dans le codage précédent
- Les coordonnées de son coin supérieur gauche et de son coin inférieur droit afin d'accélérer les tests de positionnement ultérieurs

2.2 Rotations

La rotation courante du bloc est mémorisée dans un attribut `glyph_index` qui correspond à l'indice courant dans la liste de ses rotations.
On peut donner l'ordre :

- Soit de tourner le bloc dans un sens avec la méthode `rotate(self, direction)`, où `direction` vaut `'H'` pour le sens horaire et `'T'` pour le sens trigonométrique.

- Soit de le placer directement dans une rotation désirée avec la méthode `setDirection(self, i)` où `i` est l'indice de la rotation voulue.

2.3 Méthodes utilitaires

Cette classe implémente une méthode `copy(self)` qui renvoie une copie du bloc ainsi qu'une méthode `toArray(self)` pour avoir une représentation matricielle dans une matrice carrée dans laquelle chaque case occupée vaut `id` et les case vides 0. Cette dernière méthode sera utile pour la pratie réseaux de neurones pour entrer la pièce sous la forme d'un vecteur 0/1.

2.4 Sacs de pièces

Après la définition de la classe `Tetramino` on définit les constantes correspondant à chaque bloc et on en fait la liste dans des constantes `BLOCK_BAG`.
Pour le moment il y a deux sacs de pièces :

- `RAPID_BLOCK_BAG` : Chaque bloc a un nombre limité de rotation suivant ses symétries. Par exemple le bloc "I" n'a que deux rotations. Ce sac est utile pour les placements directs des pièces et économise des calculs inutiles.
- `CLASSIC_BLOCK_BAG` : Chaque bloc (sauf le "O") a quatre rotations pour coller aux règles du Tétris dans lesquelles un bloc tourne autour du centre de sa matrice carrée. Ce sac sera utile pour simuler le comportement d'un vrai joueur humain.

3 La classe Board

La classe `Board` implémente la grille, ses méthodes de gestion (mise à jour des cellules, traitement des lignes,...) ainsi que les outils statistiques (nombre de trous, hauteur maximum,...). Elle est implémentée dans le fichier `board.py`.

3.1 Constructeur

Le constructeur de la classe `board` admet deux paramètres :

- Sa largeur : `width`
- Sa hauteur : `height`

La grille en elle-même est stockée dans la liste double `self.grid` qui a pour largeur `width` et pour hauteur `height+2` (avec les deux lignes invisibles du dessus).
Enfin, les différents indicateurs statistiques sont initialisés.

3.2 Gestion des cellules

La gestion des cellules de la grille sont gérées par des getters et setters qui présentent peu d'intérêt et dont les noms sont explicites.

Notons toutefois les méthodes suivantes qui seront utilisées lors de la suppression des lignes pleines :

- `isLineFull(self, i)` qui teste si la ligne `i` est vide
- `removeLine(self, i)` qui supprime la ligne `i` de la grille et rajoute une ligne vide en haut.

3.3 Récupération des caractéristiques de la grille

Les méthodes suivantes permettent de récupérer, dans des attributs spécifiques les différentes caractéristiques de la grille :

- `columnHeight(self, j)` : renvoie la hauteur de la colonne `j`. Le principe de l'algorithme est de partir du haut de la grille et de décrémenter cette hauteur tant que la cellule visitée est vide :

```
i = hauteur de la grille
Tant que i>=0 et la cellule (i,j) est vide :
 i = i-1
Renvoyer i+1
```

  Notons que cette fonction renvoie bien la hauteur de la colonne et non l'indice de la ligne de la case la plus haute.
- `getColumnHeights(self)` : renvoie la liste des hauteurs des colonnes.
- `getMaxHeight(self)` et `getSumHeights(self)` : renvoient respectivement la hauteur maximum et la somme des hauteurs des colonnes.
- `getBumpiness(self)` : renvoie la somme des valeurs absolues des différences de hauteur entre les colonnes consécutives.
- Pour la détermination du nombre de trous, on procède de la façon suivante : un trou est défini comme une cellule vide dans une colonne qui contient un cellule pleine au-dessus (cellule dominée).
  - `isDominated(self, i, j)` : teste si la cellule `(i,j)` est dominée.

```
Si la cellule (i,j) n'est pas vide :
 Renvoyer Faux
Sinon :
 Pour k allant de i+1 à (hauteur de la colonne j) - 1 :
   Si la cellule (i,j) n'est pas vide :
     Renvoyer Vrai
 Renvoyer Faux
```

  - `getNbHoles(self)` : renvoie le nombre de trous en comptant pour chaque cellule si elle est dominée.
- `updateStats(self)` : met à jour toutes les caractéristiques de la grille

3.4 Gestion des lignes

La méthode `processLines(self)` supprime les lignes pleines et renvoie le nombre de lignes supprimées :

```
nb_lignes = 0
hauteur_max = hauteur de la grille
i = 0
Tant que i <= hauteur_max :
 Si la ligne i est pleine :
   Enlever la ligne i
   hauteur_max = hauteur_max-1
   nb_lignes = nb_lignes +1
 Sinon :
   i = i + 1
Renvoyer nb_lignes
```

3.5 Méthodes utilitaires

- `copy(self)` : renvoie une copie de la grille.
- `__str__(self)` : renvoie une chaîne de caractère pour affichage de la grille.
- `printInfos(self)` : affiche les caractéristiques de la grille (pour les tests).

4 La classe TetrisEngine

Cette classe fait le lien entre les deux précédentes et implémente le déroulement de la partie. Elle est implémentée dans le fichier `tetris_engine.py`.

4.1 Constructeur et attributs

Les paramètres du constructeur sont les suivants :

- `width` et `height` : les dimensions de la grille
- `getMove` : c'est la fonction qui va renvoyer, à chaque tour, le coup à jouer. C'est cette fonction que vont implémenter les agents (fonction de callback).
- `max_blocks` : le nombre maximum de blocs à jouer pour limiter les temps des essais (0 pour jouer jusqu'à la fin de la partie)
- `base_block_bag` : le sac dans lequel on va tirer les pièce, comme défini dans le module `tetramino.py`
- `temporisation` : en secondes, un temps de pause entre deux mouvements
- `silent` : si `True`, ne produit aucun affichage (pour les essais sur plusieurs parties)
- `random_generator_seed` : la graine du générateur aléatoire pour reproduire des tests si besoin

- `agent_name` et `agent_description` : le nom et la description de l'agent, pour l'affichage.

Le constructeur va également définir les attributs suivants :

- Gestion de la grille :
  - `self.board` : la grille de jeu dans laquelle les pièces évoluent.
  - `self.fixed_board` : la grille de jeu statique dans laquelle les pièces sont placées. Cette grille est une copie de `self.board` à la fin de chaque coup.
- Gestion des blocs :
  - `self.block_bag` : la sac contenant les pièces à jouer
  - `self.block` : le bloc courant en jeu
  - `self.block_position` : les coordonnées du coin en haut à gauche du bloc courant
  - `self.next_block` : le bloc suivant
  - `self.nb_blocks_played` : le nombre de pièces déjà jouées
- Gestion des scores :
  - `self.score` : score total de la partie
  - `self.score_on_move` : score gagné sur le dernier coup (pour la fonction d'évaluation)
  - `self.total_lines` : le nombre total de lignes faites sur la partie
- Gestion de la partie :
  - `is_running` : Flag pour indiquer que la partie est en cours
- Gestion du temps :
  Différentes variables pour chronométrer différents aspects de la partie (temps par coup, temps moyen par coup, temps total pour les coups, temps de jeu,...)

4.2 Génération des pièces

La génération d'une nouvelle pièce dans le jeu se fait avec la méthode `getNewBlock()` et suit l'algorithme suivant :

- Si le sac de pièces est vide, on en recrée un nouveau que l'on mélange grâce à la méthode `generateNewBlockBag()`
- On tire une pièce du sac avec la méthode `generateNewBlock()`
- On place la pièce sur la ligne du dessus au centre de la grille avec la méthode `setBlockInitPosition()`

4.3 Déplacements et rotations

La méthode `isMoveValid(self, block, new_position)` teste si le bloc peut être placé dans une nouvelle position candidate.
Pour cela on teste d'abord qu'elle ne sort pas de la grille, puis que les cases qui seraient occupée dans la nouvelle position sont bien vides.

Pour déplacer un bloc, on commence par l'effacer de la grille avec la méthode `eraseBlock()`. Cette méthode se contente de vider les cellules de la grille occupée actuellement.
Ensuite on le place dans son nouvel emplacement en mettant l'id du bloc dans les cellules qu'il occupe.

Partant de là, il est facile de déplacer un bloc dans une direction, avec la méthode `moveBlockInDirection(self, direction)` où direction vaut `'L'` pour gauche, `'R'` pour droite et `''` pour un simple déplacement vers le bas.
La méthode `dropBlock()` fait tomber le bloc en le déplaçant vers le bas tant que c'est possible.

Pour les rotation, c'est le même principe sauf qu'avant de placer le bloc on lui fait subir la rotation désirée.

Enfin, il est possible de placer directement le bloc dans une colonne et une rotation donnée en bas de la grille.
Pour cela on commence par tester si le bloc peut bien être placé dans la colonne et la rotation voulue et on le fait tomber.
Notons la méthode `getPossibleMoveDirect(self)` qui renvoie tous les tuples de la forme `(colonne, rotation)` pour lesquels on peut placer le bloc directement. Cette fonction sera très utile dans la suite pour calculer le meilleur coup.

4.4 Commandes de jeu

L'interface entre le moteur et les agents se fait grâce à un système de commandes qui sont interprétées dans la méthode `playCommand(self, command)`. Ce sont des chaînes de caractères que l'agent va envoyer au moteur. Ces commandes sont :

- `'L'` et `'R'` : Déplacement à gauche et à droite
- `'D'` : Lâche la pièce en bas (hard drop)
- `'N'` : Ne fait rien (déplacement d'une case vers le bas)
- `'H'` et `'T'` : Tourne la pièce dans le sens horaire ou trigonométrique
- `P:j:r` : Place directement la pièce dans la colonne `j` et la rotation `r`

4.5 Déroulement de la partie

Le déroulement du jeu se fait dans la méthode `run(self)`. L'algorithme est le suivant :

- Tant que le jeu tourne
  - Met un nouveau bloc en jeu
  - Tant que la pièce peut descendre et que le jeu tourne :
    - Descendre la pièce d'un cran
    - Met à jour le score sur un mouvement
    - Met à jour l'affichage
    - Récupère le prochain mouvement

    - Joue ce mouvement
    - Traite les lignes faites
    - Met à jour le score sur les lignes
  - Fixe la grille
  - Teste la fin du jeu
- Retourne le score

4.6 Affichage

L'affichage n'étant pas encore définitif, cette partie sera complétée plus tard.

Chapitre 3

LES AGENTS

Les agents sont à la base du projet dont le but est, justement, de les programmer.

1 Généralités

Un agent est essentiellement une classe qui a accès à un moteur de jeu (et donc à tous les paramètres de jeu) et qui implémente une méthode `getMove()` qui renvoie la commande du prochain coup à jouer.
C'est de leur responsabilité de lancer la partie.
Les fonctionnalités de base des agents sont implémentées dans la classe `Agent` dont les agents héritent tous. cette classe permet de :

- Récupérer les paramètres de la grille après qu'un coup ait été joué via la méthode `getMoveStats(self, move)`. Cette méthode est utilisée dans la méthode `allMoveStats(self)` qui remplit un dictionnaire dont les clefs sont les différents placements possibles et les valeurs un dictionnaire content les différentes statistiques de jeu (nombre de lignes créées, nombre de trous,...)
- Créer une commande pour un placement direct via la méthode `commandFromMove(self, move)`.

2 Joueur humain en mode texte

Pour tester les différentes fonctionnalités du moteur, nous avons implémenté un agent, `AgetHuman`, qui se contente de recevoir les différentes commandes à jouer via l'entrée standard.
Grâce à cet agent nous avons pu tester le déplacement et la rotation des pièces, la suppression des lignes,...

3 Agent aléatoire

Ensuite le premier agent automatique qui joue de manière aléatoire.
`AgentRandom1` joue des coups aléatoires de type "aller à gauche", "tourner la pièce",... à la manière d'un joueur humain.
`AgentRandom2`, quant à lui, place directement les pièces dans des colonnes et des rotations aléatoires.

Évidemment ces deux agents sont catastrophiques en terme de performances mais ne demandent qu'à être améliorés (ces sont les "hello world" des agents).

4 Agent par filtrage

Le premier agent un tant soit peu efficace.
La stratégie utilisée est de filtrer la liste des coups jouables successivement selon plusieurs critères :

- D'abord il ne garde que les coups qui font le moins de trous
- Ensuite, parmi eux, il ne garde que ceux qui donnent une somme des hauteurs des colonnes de la structure minimal
- Puis, ceux qui minimisent le bumpiness
- Enfin sont qui créent le plus de lignes

Cet agent joue plutôt bien pour une heuristique aussi simple.

5 Agent par évaluation des coups

Cet agent est à la base de ce que nous ferons avec les algorithmes génétiques :
Pour chaque coup jouable notons L le nombre de lignes, H la somme des hauteurs des colonnes, T le nombre de trous créés et B le bumpiness, après que le coup ait été joué.
La qualité d'un coup peut être évaluée par une fonction

$$f(L,H,T,B) = a \times L - b \times H - c \times T - d \times B$$

où a, b, c et d sont des paramètres positifs que nous pouvons supposés dans $[0\ ;\ 1[$ (en effet on pourrait, sans perte de généralité les diviser par $a+b+c+d$ pour s'y ramener si ce n'était pas le cas).

Le coup à jouer est donc celui qui maximise cette fonction.

Tout le problème consiste à déterminer ces coefficients et c'est le but de l'optimisation par algorithme génétique.

Deuxième partie

Optimisation par algorithmes génétiques

Troisième partie

Optimisation par reinforcement learning