|  |  |
| --- | --- |
| Image illustrant une route sinueuse et des arbres  Tetris AI  with python | Résumé  Le but de ce projet est d’avoir une première approche concrète du développement d’une IA. Pour cela nous allons créer un Tetris en python ainsi qu’une intelligence artificielle capable de jouer d’y jouer.  Maxence Vanhaezebroeke Maxime Medynska  Projet intelligence artificielle |



Table des matières

[I) Introduction 2](#_Toc58574649)

[1) Tetris 2](#_Toc58574650)

[2) Intelligence Artificielle 3](#_Toc58574651)

[II) Conception et développement 3](#_Toc58574652)

[1) Tetris 3](#_Toc58574653)

[2) Deep Q Network 4](#_Toc58574654)

[Les aspects importants de notre DQN (Deep Q Network) : 4](#_Toc58574655)

[Algorithme du DQN (simplifié) : 5](#_Toc58574656)

[III 5](#_Toc58574657)

# I) Introduction

Pour ce projet nous devons avoir le jeu Tetris en python afin de pouvoir développer notre Intelligence Artificielle en python, cela sera beaucoup plus facile pour nous. Dans cette introduction nous allons découvrir le jeu Tetris au cas où vous ne le connaîtriez pas (est-ce possible de ne pas le connaître ?) puis nous allons définir le terme Intelligence Artificielle.

## Tetris

Tetris est un jeu vidéo de "puzzle" inventer en 1984 par Alekseï Pajitnov et aidé de Dimitri Pavlovski et Vadim Guerassimov pour le développement (de nationalité russe). Le jeu est dans une grille de 10x20 cubes, des figures ou tétrominos apparaissent en haut de cette grille est chute lentement en bas. Le but pour le joueur est de disposer ces tétrominos dans la grille de façon qu’aucun tétrominos ne touche le haut de la grille. Pour cela lorsque qu’une ligne est entièrement remplie avec des tétrominos (ou une partie de ceux-ci) la lignes disparaît libérant de l’espace pour d’autre tétrominos. Le joueur ne peut pas empêcher la chute d’un tétrominos, il peut seulement déplacer la figure sur la droite ou la gauche ou la faire tourner autour d’un centre de rotation fixe pour chaque figure.

Sur cette image le tétrominos bleu en haut de la grille va être placé tout en bas à droite et va faire disparaître les 4 dernières lignes :

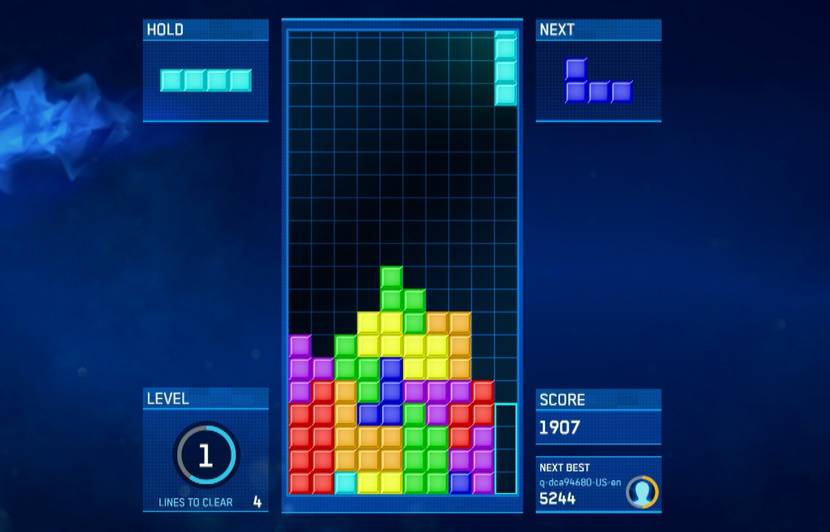


Figure  : Grille de jeu Tetris

## Intelligence Artificielle

L’intelligence artificielle consiste en la réalisation d’un programme ou d’une machine capable d’imiter une forme d’intelligence réelle.  
Il existe de nombreuses façons de créer une intelligence artificielle, et ses domaines d’applications sont très vastes.

Dans le jeu Tetris, créer une intelligence artificielle capable de jouer au jeu reviendrait à concevoir un programme, qui connaît les règles du jeu (déplacer une pièce à droite ou à gauche, faire une rotation de la pièce) avec le but du jeu (obtenir le score maximum sans que la pièce qui apparaît ne bloque les pièces existantes du plateau)

(Ce ne sont pas les règles du jeu mais plutôt comment jouer, l’IA ne connait pas les règles du jeu selon moi)

Nous souhaitons que l’intelligence joue à Tetris, et grâce à de nombreuses techniques, qu’elle s’améliore pour qu’elle puisse obtenir le score potentiellement le plus élevé du jeu.

Il y a différentes façon d’apprendre à une intelligence artificielle, il existe l’apprentissage supervisé, non supervisé, par renforcement, etc. Nous avons donc dû choisir une méthode d’apprentissage :

**Par renforcement** : cette méthode permet à un agent d’apprendre à partir de rien. L'agent est plongé au sein d'un environnement, et prend ses décisions en fonction de son état courant. En retour, l'environnement procure à l'agent une récompense, qui peut être positive ou négative. L'agent cherche, au travers d'expériences itérées, un comportement décisionnel (appelé stratégie ou politique, et qui est une fonction associant à l'état courant l'action à exécuter) optimal, en ce sens qu'il maximise la somme des récompenses au cours du temps.

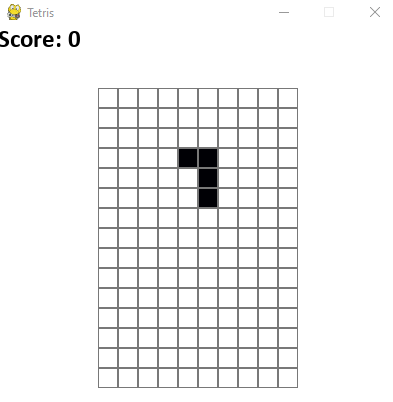
Pour notre Tetris nous utiliseront l’apprentissage par renforcement, nous allons créer un agent qui sera plonger dans un environnement (notre Tetris) et qui apprendra à partir de plusieurs parties de Tetris jouer.

# II) Conception et développement

## Tetris

Pour développer notre jeu Tetris, nous avons d’abord pensé le jeu au niveau des classes nécessaires au bon fonctionnement. Nous avons décidé de créer 5 classes :

* Une classe Coordinates, qui contiendra les coordonnées des blocs (leur emplacement sur la grille)
* Une classe Block, qui contient un ensemble de coordonnées
* Une classe Tetris block, qui hérite de la classe Block mais dans laquelle on ne peut construire que des tétrominos
* Une classe Grid, qui contient un tableau et qui va stocker la position de chaque bloc dans le jeu
* Une classe Tetris, qui s’occupe d’agencer des tétrominos dans une Grid en fonction de l’utilisateur (ou de l’IA)

Le plateau de notre Tetris ressemble à cette image :

On peut voir la représentation de la Grid d’une instance Tetris, dans lequel un Tetris Block est en train de tomber.

## Deep Q Network

Le Deep Q Learning est une technique d’apprentissage par renforcement, elle utilise une fonction qui mesure la qualité d’une action effectué, plus l’action est bien plus la récompense pour l’agent sera importante. Pour développer notre Deep Q Network nous allons utiliser la bibliothèque PyTorch.

### Les aspects importants de notre DQN :

**Train.py et DeepQNetwork.py :**

* Le réseau de neurones : notre classe DeepQNetwork qui hérite de Torch.module.
* La fonction "forward" du DQN qui permet de faire traverser les données à travers les couches de neurones de notre DQN.
* La fonction "train" qui entrainera notre DQN :
  + L’attribut env qui est l’environnement, ici on a : env = Tetris(). L’environnement nous permettra de récupérer les informations nécessaire pour jouer.
  + L’attribut net qui est notre DQN : net = DeepQNetwork(). C’est lui qui prendra une décision du coup à jouer.
  + Torch.cuda qui permet de réaliser des calculs simples sur le GPU au lieu du CPU.
  + Replay\_memory : la structure de données qui va nous servir de mémoire pour stocker nos ensembles (state, action, new\_state, reward). A chaque action, on va remplir cette mémoire au lieu d’entrainer, puis on va régulièrement piocher aléatoirement des samples dans cette mémoire, pour lancer l’entrainement sur un batch de données.
  + Notre boucle " while epoch < opt.num\_epochs” : on boucle tant qu’on a pas atteint le nombre de génération voulus.

**Tetris.py (l’environnement) :**

* Les fonctions get\_states, get\_holes, get\_bumpiness, etc, qui vont nous permettre de mesurer la qualité d’un coup jouer par notre IA.
* La fonction step qui prend une action en entrée (nombre de rotations de la pièce et position), la réalise, et retourne un score associé à cette action ainsi qu’un booléen pour gameover ou non.
* La fonction get\_next\_states qui va enregistrer tous les coups jouables à un instant t.
* La fonction get\_states : retourne le nombre de trous, ligné complétée, la hauteur et l’aplatissement des figures.
* Une fonction reset() qui nous permet de réinitialiser notre environnement.

### Algorithme du DQN (simplifié) :

Pour un certain nombre de génération :

1. On récupère tous les états possibles avec la pièce qu’on a actuellement.
2. On passe nos données à travers le réseau de neurones : notre prédiction.
3. On choisit soit une action au hasard, soit une action prédite (avec epsilon).
4. On exécute notre action sur l’environnement et on en récupère un score et un booléen gameover.
5. On enregistre notre action dans replay\_memory afin qu’elle fasse partie des actions choisies au hasard.
6. Si on est sur un gameover on réinitialise l’environnement et on enregistre les états finaux.
7. On calcul la perte et les nouveaux gradients.
8. On lance une étape d’optimisation.

# III) Avancement et conclusion