Une image contenant texte, clipart, arts de la table

Description générée automatiquement

**EFREI Paris**

**L1 Groupe B**

**2022 - 2023**

Tétriste ? Arrête !

Une image contenant texte

Description générée automatiquement

**Par Maxime JACONELLI et Anaëlle POLLART**

Tables des matières

[I. Introduction 3](#_Toc122089179)

[II. Fonctionnalités 4](#_Toc122089180)

[III. Présentation technique 5](#_Toc122089181)

[IV. Présentation des résultats 5](#_Toc122089182)

[V. Conclusion 5](#_Toc122089183)

Une image contenant texte, clipart, arts de la table

Description générée automatiquement

# Introduction

Ce projet consiste à réaliser une variante du jeu Tetris dont l’objectif est de placer des blocs choisis par l’utilisateur dans un plateau de différente forme parmi un losange, un triangle et un cercle. Lorsqu’une colonne est pleine, celle-ci est supprimée. De même pour une ligne, faisant descendre les blocs placés au-dessus. Le jeu s’arrête lorsque le plateau est plein ou lorsque l’utilisateur échoues trois tentatives de suite lors du placement d’un bloc.

L’objectif de ce projet est de mettre en œuvre toutes nos connaissances acquises au cours du premier semestre en python et en particulier les matrices et les fonctions. Ce projet nous contraint et nous entraine également à se tenir à un cahier des charges ainsi qu’à des conditions obligatoires comme une date d’échéance par exemple. L’objectif est également de développer le travail en équipe et la répartition des tâches.

# Fonctionnalités

Les différentes fonctionnalités que nous proposons à l’utilisateur sont les suivantes :

* Trois fonctions *circle(dimension), rhombius(dimension)* et *triangle(dimension)* qui vont permettre de créer le plateau d’une forme spécifique utilisé lors du jeu.
* Une fonction *read\_grid(path)* qui retourne une grille valide lue à partir du contenu du fichier spécifié par path.
* Une fonction *save\_grid(path, grid)* qui sauvegarde la grille grid dans un fichier spécifié par path.
* Une fonction *print\_grid(grid)* qui affiche l'état de la grille grid.
* Une fonction *print\_blocks(grid)* qui prend en paramètres la forme du plateau choisi, et qui affiche la liste de tous les blocs qui lui sont associés.
* Une fonction *select\_block()* qui permet de sélectionner les blocs à proposer à l’utilisateur selon l’une des 2 politiques parmi mode bloc unique qui afficher à chaque tour du jeu l’ensemble des blocs disponibles et l’utilisateur en sélectionne un et le mode bloc aléatoire qui affiche uniquement trois blocs aléatoirement
* Une fonction *valid\_position(grid, bloc, i, j)* qui vérifie si le bloc peut être placé sur l'emplacement grid[i][j] de telle façon que la case inférieure gauche du bloc soit positionnée sur grid[i][j].
* Une fonction *emplace\_block(grid, bloc, i, j)* qui positionne le bloc bloc à la position (i, j) de la grille grid si et seulement celle-ci est une position valide, et modifie ainsi la grille.
* Une fonction *row\_state(grid, i)* qui vérifie si toute la ligne i d'une grille grid est pleine.
* Une fonction *col\_state(grid, j)* qui vérifie si toute la colonne j d'une grille grid est pleine.
* Une fonction *row\_clear(grid, i)* qui annule la ligne i de la grille grid en décalant toutes les lignes du haut d'une unité vers le bas. Faire attention car selon la forme du plateau, il se peut que certaines cases de la ligne précédente ne soient plus présentes dans le plateau.
* Une fonction *col\_clear(grid, j)* qui annule la colonne j de la grille grid
* Nous avons ajouté d’autre fonctions dans notre programme :
  + La fonction matrix\_conversion*(matrix\_line)* qui va effectuer la conversion d'une ligne d’une matrice en chaine de caractère
  + La fonction available\_blocks*(shape)* qui retourne une liste contenant la liste des blocs communs et la liste correspondant à la forme du plateau
  + La fonction clean\_console() qui permet de réinitialiser l’affichage de la console
  + La fonction invalid\_choice(error) qui incrémente et retourne le nombre d’erreur mais aussi qui affiche le nombre de tentatives restantes.
  + La fonction reformat\_lines(lines) qui remplace les 0 1 et 2 par des caractères ascii en retournant une chaine de caractère
  + La fonction start\_or\_rules() qui propose à l’utilisateur un menu ou il choisit entre jouer et afficher les règles
  + La fonction game\_configuration() qui va proposer à l’utilisateur les différents paramètres du jeu, comme la dimension du plateau, la forme ou encore le mode du jeu (politic)
  + La fonction rules() qui affiche les règle du jeu puis propose à l’utilisateur de revenir au menu ou de quitter
  + La fonction start\_game(shape,dimension,politic)  qui démarre le jeu en fonction des paramètre shape dimension et politic défini dans game\_configuration

# Présentation technique

Nous avons utilisé la fonction *conversion\_matrice(ligne\_matrice)* dans le but de convertir une matrice en chaine de caractère car cela facilitera la manipulation du programme notamment pour remplacer les 0 par un caractère ASCII grâce à la fonction *replace.*

Lors de la saisie de la largeur du plateau une fois la forme du plateau choisi, l’utilisateur peut uniquement choisir des largeurs impaires entre 21 et 25, soit 21, 23 ou 25. Nous avons fait ce choix car cela permettait d’avoir un milieu unique pour les plateaux.

Parmi le fonctions assez techniques, il y avait valid\_position, et row\_clear.

* valid\_position(grid, block,cord\_x, cord\_y) : doit vérifier si un certain block peut être placé aux coordonnées indiquées sur le plateau de jeu grid.

Le fonctionnement logique de la fonction est le suivant : comme les coordonnées indiquées sont les coordonnées du placement du coin inférieur gauche du block, la fonction va vérifié la validité en testant de la gauche vers la droite et du bas vers le haut.

Une case est dite invalide si elle est occupée, ou si elle ne fait pas partie du plateau, autrement dit si elle a la valeur 2 ou 0. Le programme va vérifier ligne par ligne et case par case cette validité, et il suffit qu’une case soit invalide pour qu’il retourne False.

Deux variable sont créees, temp\_x et temp\_y et prennent respectivement la valeur de cord\_x et cord\_y . On ne va vérifier la validité de la case temp\_x temp\_y uniquement si une case est à remplir, c’est-à-dire que le bloc occupera cette case après vérification de la validité. Temp\_x sera incrémenté de 1 le même nombre de fois que la taille d’un block pour se déplacer de la gauche vers la droite, et temp\_y sera décrémenté de 1 une fois que que toute une ligne est vérifié. A chaque fois qu’une ligne est vérifiée, temp\_x reprends la valeur initiale (cord\_x) pour repartir du début de la ligne et recommencer la vérification.

Si toute les boucle se sont terminées, c’est qu’elles n’ont rencontré aucun cas de cases invalides, ainsi le block peut être placé aux coordonnées indiquées, donc la fonction retourne True.

* Row\_clear(grid,i) : doit supprimer la ligne d’indice i et décaler tout les block du dessus de 1.

Pour supprimer la ligne i, il a suffit de de parcourir toute la largeur du plateau (grid), de vérifier si la valeur de la case est de 2, et si elle est de 2, la remplacer par un 1.

Mais la réelle difficulté ici était de descendre toutes les lignes du dessus de 1. Pour cela on fait une boucle k in range allant de i à 0, car on parcourt du bas en haut. Et chaque case du dessus va être copiée en dessous, si la case du dessus est un 2 et si la case du dessous n’est pas un 0 (car sinon cela modifie la forme du plateau).

Nous avons rencontré quelques difficultés lors de la réalisation de ce projet. En effet, l’affichage des blocks proposé dans l’invite des commande était compliqué, car nous tenions à ce que cet affichage occupe le moins de ligne possible car cela oblige l’utilisateur à remonter tout en haut pour voir le plateau. Nous avons donc décidé de coder la fonction print\_blocks() de manière à ce que en mode ensemble, 10 blocks sont affichés par lignes.

Une autre difficulté était la génération du plateau, car si les dimensions était paire, aucun réel milieu était défini. Nous avons donc fait le choix de contraindre le joueur de choisir uniquement une dimension impaire.

Aussi pour les saisies sécurisées, nous les avons codé de sorte à ce que si un caractères était rentré l ou un nombre était attendu, le programme ne plante pas. Nous avons aussi mis la ou nous le pouvions des sélections par « checkbox » que l’on choisi avec les flèches, pour que l’utilisateur puisse uniquement choisir ce que nous lui proposons.

Un autre imprévu à été confronté à cause du triangle. Les coin de celui-ci lorsqu’ils sont remplis forme une colonne / ligne complète et donc nos fonction clear supprimait immédiatement la case remplie. Alors nous avons décidé de considérer une colonne / ligne complète uniquement si sa taille est supérieur à 1.

# Présentation des résultats

Après de longues heures de codes et de corrections de beugues, nous sommes arrivés à un résultat graphiquement correcte et techniquement fonctionnel ! En voici quelques images

Menu principal :

Une image contenant texte

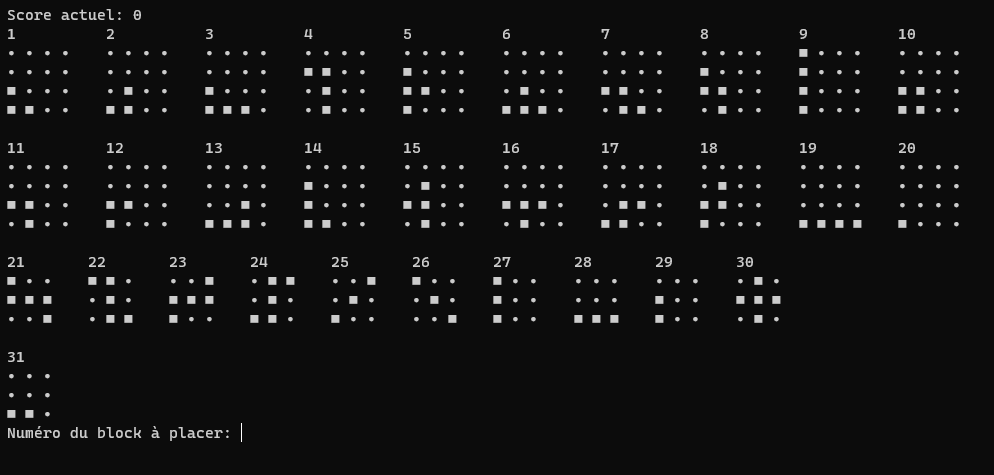
Description générée automatiquement

Paramétrage de la partie avec menu type « checkbox » :

Une image contenant texte

Description générée automatiquement

Affichage du plateau ainsi que des blocks disponibles en mode ensemble :

Exemple de saisies sécurisées et signalement de position invalide.

Une image contenant texte

Description générée automatiquement

# Conclusion

Ce projet nous a permis de nous familiariser avec le langage python et plus précisément avec les matrices et les fonctions, mais aussi avec la difficulté graphique car un jeu doit être agréable à regarder. Cela nous a aussi permis de manipuler des fichiers texte avec python et l’importation de fonction et de variable dans d’autre fichier python,.

Ce projet nous a également permis de découvrir une première approche sur ce qu’est le travail autour d’un projet en équipe, nous avons appris à se coordonner et à se répartir les tâches autour d’une même méthode de travail afin d’obtenir le résultat le plus efficace et productif possible. La date d’échéance nous a obligé à planifier et prévoir des dates de relectures pour vérifier la présence de bugs inattendus. L’obligation de coder certaines fonctions nous a appris à se tenir à un cahier des charges et a une logique qui n’était pas forcément la nôtre, ce qui a du aussi renforcer nos capacités d’adaptations.