EFREI

Rapport Projet TI-101 "Tetris"

BENJAMIN JAEGLE
GREGOIRE ALPEROVITCH

TDE-GROUPE 10

2022-2023

Sommaire

Introduction

Nous avons eu 5 semaines pour réaliser un projet utilisant le langage python en lien avec notre cours suivi à l'EFREI-Paris. Le projet vise à faire le bilan de nos acquis d'apprentissage du premier semestre. Mais c'est aussi un moyen de savoir si dans un groupe on arrive à bien s'entendre, bien communiquer, bien se repartir le travail. L'aspect technique est tout aussi important que l'aspect collaboratif dans un projet de groupe.

Notre objectif est de réaliser un jeu qui s'apparente au jeu vidéo Tetris. Comme le Tetris, le but de ce jeu étant de faire le plus de points en remplissant les lignes et les colonnes à l'aide de blocs. Mais ici, les blocs ne tombent pas sous forme de gravité comme dans le jeu classique mais on les choisit parmi 3 blocs générés aléatoirement.

Fonctionnalité

Fonctionnalité attendu :

- Afficher les règles
- Choix de la grille du jeu (losange, triangle, cercle)
- Afficher la grille
- Proposer 3 blocs aléatoires
- Sélection des coordonnées du bloc choisi
- Vérification de la position du bloc et s'il s'y poser
- Ajout du bloc dans la grille
- Suppression de la ligne/colonne si pleine
- Réorganisation de la grille lorsqu'une ligne est supprimé
- Calcul des points et affichage du résultat

Fonctionnalité en + :

- Utilisation du module OS qui permet de lancer des commande depuis le fichier py. Notamment pour pouvoir effacer la console.
- L'utilisation du mot clef "match" disponible depuis python 3.10
- Sauvegarder la partie pour la reprendre plus tard

Réalisation

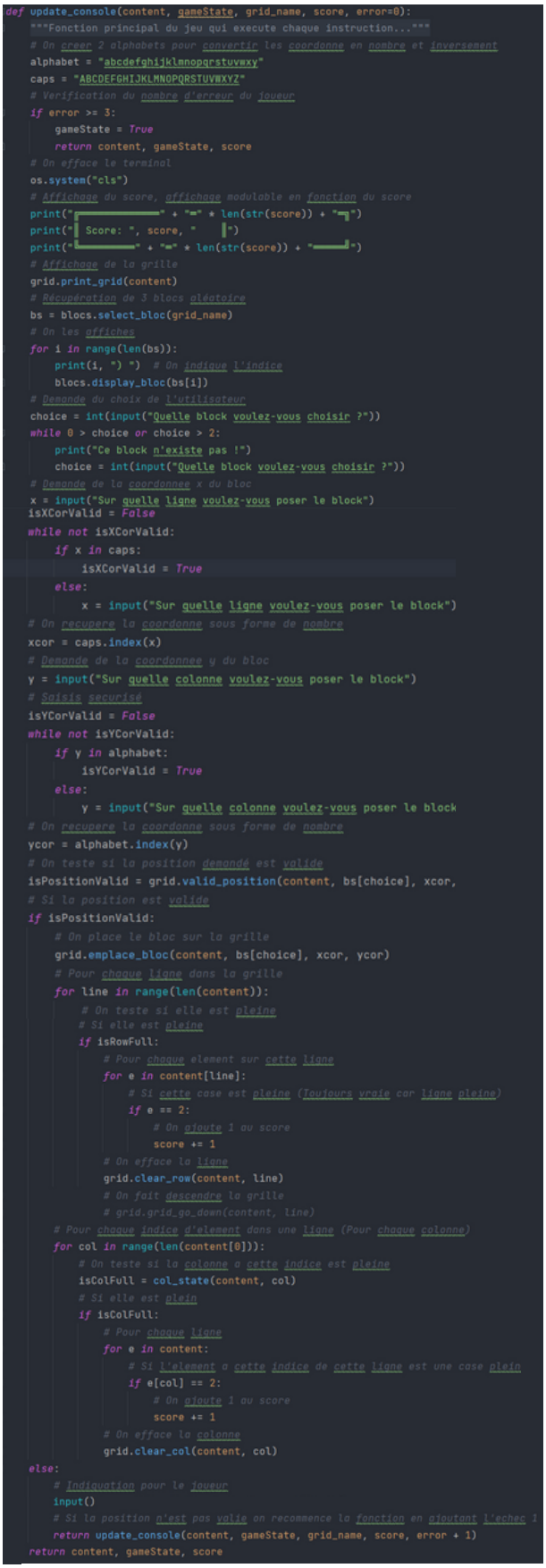
Fonction game_loop

```python
def game_loop(_grid):
    """
    Fonction principal qui execute en boucle la fonction d'instruction de jeu

        @:parameter:
            @:param _grid (str): Le nom du fichier de plateau

        @:returns :
            @:return void
    """

    isGameFinish = False  # Variable d'etat de la partie
    content = grid.read_grid(_grid)  # On recupere la matrice de la grille
    score = 100  # On creer la varible de score

    # Tant que la variable d'etat du jeu n'est pas vrai (jeu non fini) on execute la fonction
    while not isGameFinish:
        """
        La fonction update_console() renvoie un score, une nouvelle grille et un etat
        On reinsere ces elements dans la fonction en verifiant avant si la partie n'est pas finie
        """
        content, isGameFinish, score = update_console(content, isGameFinish, _grid, score)

    # On efface la console
    os.system("cls")

    # Affichage fin de partie
    print(" " * 10 + "╔═══════════════════════════════════════╗")
    print(" " * 10 + "║                                       ║")
    print(" " * 10 + "║              PARTIE TERMINE           ║")
    print(" " * 10 + "║                                       ║")
    print(" " * 10 + "╚═══════════════════════════════════════╝")
    print()
    print()
    print("╔═══════════" + "=" * len(str(score)) + "═╗")
    print("║ Score: ", score, "  ║")
    print("╚═══════════" + "=" * len(str(score)) + "═══╝")
```

Ce code définit une fonction appelée 'game_loop' qui prend en paramètre '_grid'. Le but de cette fonction est d'exécuter une boucle de jeu jusqu'à ce que le jeu soit terminé.
La fonction commence par définir une variable 'isGameFinish' sur *False*, ce qui indique que le jeu n'est pas encore terminé. Il lit ensuite le contenu de la grille à partir d'un fichier portant le nom '_grid' à l'aide de la fonction 'read_grid' et initialise une variable 'score' à 100.
La fonction entre dans une boucle qui continue tant que 'isGameFinish' est *False*. À l'intérieur de la boucle, la fonction appelle la fonction 'update_console' et transmet la valeur de 'content' de la grille, la valeur actuelle de 'isGameFinish', le nom du fichier de grille '_grid' et la valeur de 'score'. La fonction 'update_console' renvoie une nouvelle valeur pour le 'content' de la grille, une valeur mise à jour pour 'isGameFinish' et un mis à jour score. Ces valeurs mises à jour sont ensuite affectées aux variables 'content', 'isGameFinish' et 'score'.
Une fois la boucle terminée, la fonction efface la console et affiche un message indiquant que le jeu est terminé, suivi du score final.

Cette méthode est la plus couramment utilisé, elle a pour avantage d'être très simple et a mettre en place et très robuste. Le jeu tournera tout seul tant que la variable n'est pas changé

Fonction update_console



```python
def update_console(content, gameState, grid_name, score, error=0):
    """Fonction principal du jeu qui execute chaque instruction..."""
    # On creer 2 alphabets pour convertir les coordonne en nombre et inversement
    alphabet = "abcdefghijklmnopqrstuvwxy"
    caps = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    # Verification du nombre d'erreur du joueur
    if error >= 3:
        gameState = True
        return content, gameState, score
    # On efface le terminal
    os.system("cls")
    # Affichage du score, affichage modulable en fonction du score
    print("┌" + "─" * len(str(score)) + "──────────┐")
    print("| Score: ", score, "    |")
    print("└" + "─" * len(str(score)) + "──────────┘")
    # Affichage de la grille
    grid.print_grid(content)
    # Récupération de 3 blocs aléatoire
    bs = blocs.select_bloc(grid_name)
    # On les affiches
    for i in range(len(bs)):
        print(i, ") ")  # On indique l'indice
        blocs.display_bloc(bs[i])
    # Demande du choix de l'utilisateur
    choice = int(input("Quelle block voulez-vous choisir ?"))
    while 0 > choice or choice > 2:
        print("Ce block n'existe pas !")
        choice = int(input("Quelle block voulez-vous choisir ?"))
    # Demande de la coordonnee x du bloc
    x = input("Sur quelle ligne voulez-vous poser le block")
    isXCorValid = False
    while not isXCorValid:
        if x in caps:
            isXCorValid = True
        else:
            x = input("Sur quelle ligne voulez-vous poser le block")
    # On recupere la coordonne sous forme de nombre
    xcor = caps.index(x)
    # Demande de la coordonnee y du bloc
    y = input("Sur quelle colonne voulez-vous poser le block")
    # Saisis securisé
    isYCorValid = False
    while not isYCorValid:
        if y in alphabet:
            isYCorValid = True
        else:
            y = input("Sur quelle colonne voulez-vous poser le block
    # On recupere la coordonne sous forme de nombre
    ycor = alphabet.index(y)
    # On teste si la position demandé est valide
    isPositionValid = grid.valid_position(content, bs[choice], xcor,
    # Si la position est valide
    if isPositionValid:
        # On place le bloc sur la grille
        grid.emplace_bloc(content, bs[choice], xcor, ycor)
        # Pour chaque ligne dans la grille
        for line in range(len(content)):
            # On teste si elle est pleine
            # Si elle est pleine
            if isRowFull:
                # Pour chaque element sur cette ligne
                for e in content[line]:
                    # Si cette case est pleine (Toujours vraie car ligne pleine)
                    if e == 2:
                        # On ajoute 1 au score
                        score += 1
                # On efface la ligne
                grid.clear_row(content, line)
                # On fait descendre la grille
                # grid.grid_go_down(content, line)
        # Pour chaque indice d'element dans une ligne (Pour chaque colonne)
        for col in range(len(content[0])):
            # On teste si la colonne a cette indice est pleine
            isColFull = col_state(content, col)
            # Si elle est plein
            if isColFull:
                # Pour chaque ligne
                for e in content:
                    # Si l'element a cette indice de cette ligne est une case plein
                    if e[col] == 2:
                        # On ajoute 1 au score
                        score += 1
                # On efface la colonne
                grid.clear_col(content, col)
    else:
        # Indiquation pour le joueur
        input()
        # Si la position n'est pas valie on recommence la fonction en ajoutant l'echec
        return update_console(content, gameState, grid_name, score, error + 1)
    return content, gameState, score
```

Ce code définit une fonction 'update_console' qui prend cinq paramètres : 'content', 'gameState', 'grid_name', 'score', et 'error'. Le but de cette fonction est de mettre à jour l'affichage de la console avec l'état actuel du jeu et de permettre au joueur de faire une instruction.

La fonction commence par créer deux chaînes alphabétiques à convertir entre les lettres de coordonnées et les chiffres. Il vérifie ensuite si le joueur a fait plus de trois erreurs, et si c'est le cas, il définit 'gameState' et '*True*' renvoie les variables 'content', 'gameState' et 'score'.

La fonction efface ensuite la console, affiche le score actuel et appelle la fonction 'print_grid' pour afficher l'état actuel 'content' de la grille. Il appelle ensuite la fonction 'select_bloc' pour sélectionner au hasard trois blocs et les affiche avec leurs indices.

La fonction invite ensuite le joueur à choisir un bloc et à entrer les coordonnées de son emplacement. Il vérifie que les coordonnées saisies sont valides, et si elles le sont, il appelle la fonction 'emplace_bloc' pour placer le bloc choisi sur la grille aux coordonnées spécifiées.

La fonction vérifie ensuite chaque ligne de la grille pour voir si elle est pleine, et si c'est le cas, elle supprime cette ligne et incrémente le score. Il vérifie ensuite si le jeu est terminé en appelant la fonction 'game_over' et en passant dans la 'content' grille. Si le jeu est terminé, il définit 'gameState' et '*True*' renvoie les variables 'content', 'gameState' et 'score'. Si le jeu n'est pas terminé, il renvoie les mises à jour 'content', 'gameState'et 'score'.

Cette fonction rejoint l'idée de game_loop() C'est cette fonction qui s'occupe d'exécuter toute les instruction du jeu dans 1 tour

Fonction start

```python
def start():
    """=    Fonction de demarage, affiche le menu du jeu..."""

    # On efface la console
    os.system("cls")

    # On affiche le menu et les differents choix
    print(" " * 10 + "╔═══════════════════════════════════════════╗")
    print(" " * 10 + "║                                           ║")
    print(" " * 10 + "║                Bienvenue sur TETRIS       ║")
    print(" " * 10 + "║                                           ║")
    print(" " * 10 + "╚═══════════════════════════════════════════╝")
    print()
    print("1- Commencer le jeu")
    print("2- Afficher les regles du jeu")
    print("3- Quitter le jeu")
    print()
    print()

    # On demande a l'utilisateur son choix
    result = int(input("Reponse: "))

    match result:

        # 2 = Afficher les regles
        case 2:

            # On efface le terminal
            os.system("cls")

            # On execute la fonction regles()
            regles()

            # On attend tant que l'utilisateur n'appuie pas sur une touche
            input()

            # On efface la console
            os.system("cls")

            # On re affiche le menu
            start()

        # 1 = Commencer le jeu
        case 1:

            # On affiche les grilles
            choisir_grid()

        case 3:

            # On retourne 0, le programme s'arrete
            return 0

        # Sinon on re affiche le menu
        case _:
            start()
```

Ce code définit une fonction appelée 'start' qui sert à initialiser les variables du jeu et commencer la 'game_loop'. La fonction affiche un menu et demande à l'utilisateur de faire un choix.
La fonction commence par effacer la console et afficher un message de bienvenue et les options du menu. Il invite ensuite l'utilisateur à entrer une réponse. Le choix de l'utilisateur est stocké dans une variable appelée 'result'.
La fonction utilise ensuite une instruction 'match' pour effectuer différentes actions en fonction de la valeur de 'result'. Si 'result' vaut 2, la fonction appelle la fonction 'regles' pour afficher les règles du jeu, puis on attend que l'utilisateur appuie sur une touche, cela efface la console, puis affiche à nouveau le menu en appelant la fonction 'start'. Si 'result' vaut 1, la fonction appelle la fonction 'choisir_grid' pour afficher les grilles disponibles. Si 'result' vaut 3, la fonction renvoie 0, ce qui entraînera l'arrêt du programme. Si 'result' est une autre valeur, la fonction affiche à nouveau le menu en appelant la fonction 'start'.

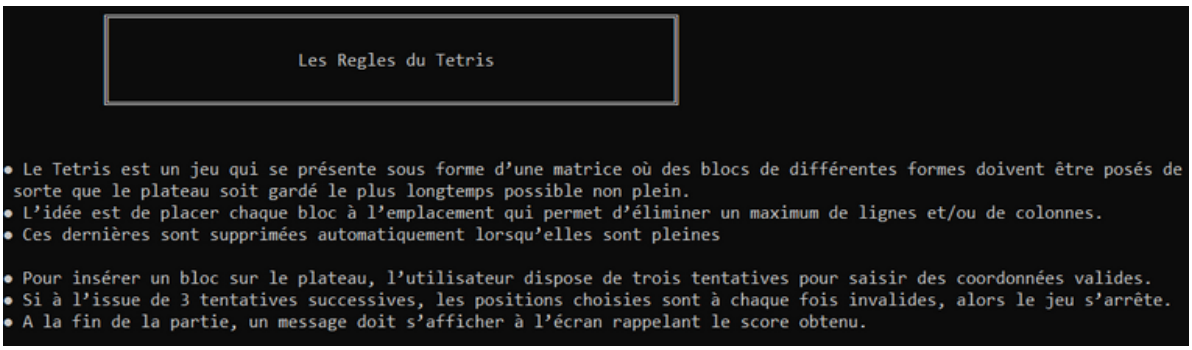
La fonction start() est séparé des autres car elle a pour unique but d'afficher le menu et commencer ou non la partie. Une fonction a part pour pouvoir l'exécuter a un moment précis

Affichage

Nous allons effectuer ici plusieurs tests afin de présenter notre code :

- Lorsque nous lançons le projet nous arrivons sur le menu principal qui nous propose 3 destinations

- Commençons par afficher les règles en entrant "2" dans le terminal

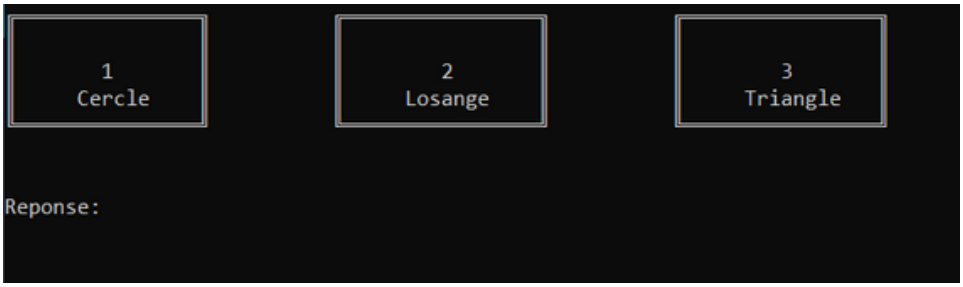


- Pour revenir au menu il suffit juste d'appuyer sur entrée, ensuite nous pouvons essayer de quitter le jeu mettant "3" depuis le menu principal

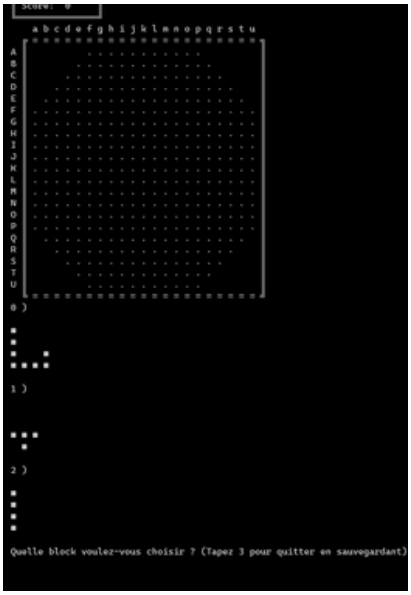
- On voit ici que je suis totalement sorti du jeu et que pour y revenir je dois mettre le le terminal "py main.py"

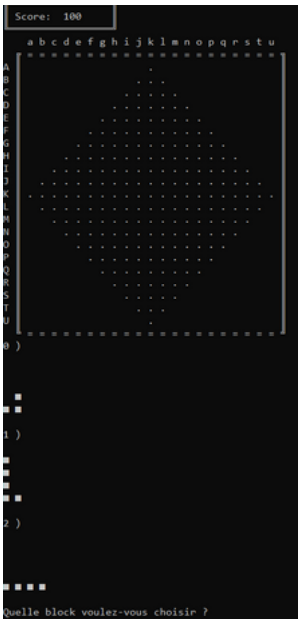
Nous avons tester le menu principal, essayons maintenant le jeu :

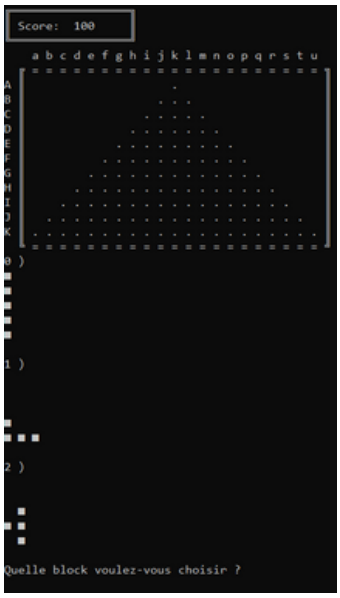
- Sur le menu principal pour lancer le jeu il faut mettre "1" dans le terminal de commande



- Il faut maintenant choisir quelle plateau/grille l'on souhaite pour notre partie
Entrer "1" pour le cercle, "2" pour le losange, "3" pour le triangle







- On remarque que le score est affiché en haut à gauche et que les 3 blocs générés aléatoirement sont proposés
- Pour la suite du test nous allons utilisés la grille triangle (ou n'importe laquelle). Maintenant cherchons à poser le bloc 1 en (E;k). ⚠ Nous disons que en (E;k) sera posé le blocs de la matrice en bas à gauche

- Maintenant essayons de remplir un ligne

- On observe que la ligne disparait, que les blocs contenus au dessus descendent de 1 ligne et que le score se mets à jour

- Essayons pour finir de perdre pour quitter le jeu (pour cela il faut placer que 3 fois un bloc soit mal-placé)

- On observe qu'au bout de 3 échecs le jeu s'arrête et le score s'affiche avant de quitter le jeu

Conclusion

Plan technique :

Ce projet nous fait utilisé les différentes fonctions apprises au cours du modules Programmation en Pyhton ("saisis sécurisé"). Parfois même des combinaisons de plusieurs notion. Mais aussi d'autres notions comme "match".
Le Readme est fait avec Markdown pour une meilleur experience

Organisation du travail :

Pour l'organisation du travail nous avons utilisé CodeWithMe lorsque nous étions en TD afin de pouvoir coder en même temps pour avancer plus vite.
Pour travailler depuis chez nous, nous avons créer un repository sur Github afin de pouvoir mettre à jour le code du projet à chaque modification.

Gestion du temps :

Nous nous sommes rendu compte très vite qu'il fallait prendre de l'avance chez soi mais surtout de mettre nos heures de TD prévu pour le projet à profit pour que nous puissions travailler les autres matières. Ainsi en privilégiant des heures pleines au projet, cela nous permettait de travailler moins le reste de la semaine