

Dossier projet

Deryne Four
TS4

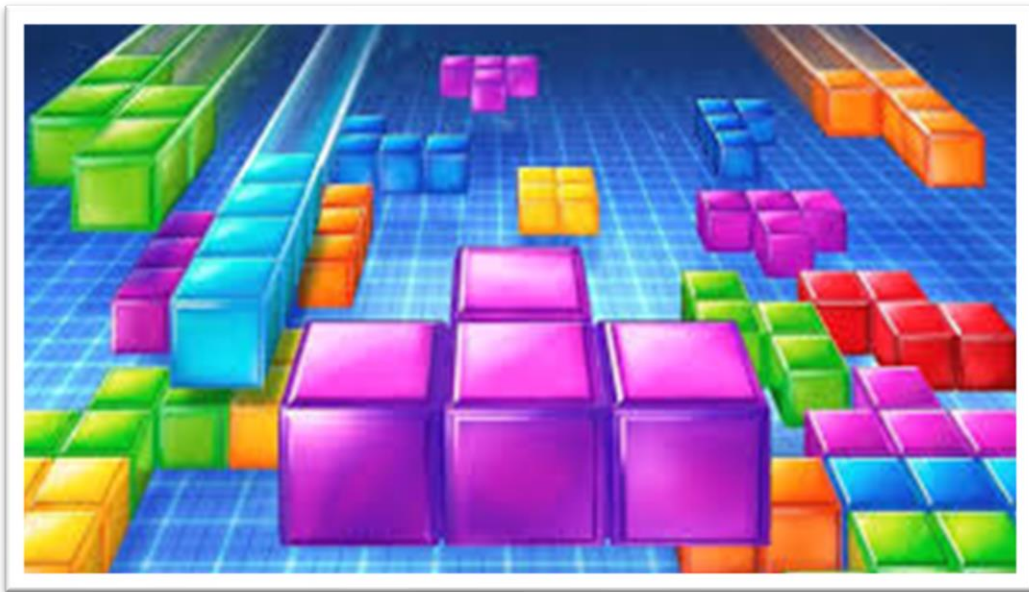


Table des matières

1. Introduction	3
1.1. Pourquoi un Tetris ?	3
1.2. Ce que je vais présenter.....	3
2. Qu'est-ce qu'un Tetris ?	3
1.3. Le concept et les règles.....	4
1.1.1. Le GamePlay	4
1.1.2. Les pièces	5
1.4. Le cahier des charges et la répartition des tâches	5
1.1.3. Conception des tâches à effectuer	5
1.1.4. La démarche collaborative	6
3. Développement.....	8
1.5. Ma réalisation	8
1.1.5. Afficher différents blocs aléatoirement : (3 séances)	8
1.1.6. Faire bouger les blocs : (2 séances)	9
1.1.7. Créer des niveaux (1 séance).....	10
1.1.8. Insérer de la musique (1 séance)	10
4. Mes impressions.....	10
1.6. Une démarche réflexive.....	10
1.7. Mes difficultés.....	11
5. Conclusion.....	11
1.8. Ce que j'ai appris.....	11
1.9. Perspectives d'amélioration	11
6. Annexes	12
1.10. Curriculum Vitae	Erreur ! Signet non défini.
1.11. Liens	12
1.1.9. Liens vers les images :	12
1.1.10. Liens vers les musiques:	12
1.12. Code du programme.....	13

1. Introduction

1.1. Pourquoi un Tetris ?

L'idée de travailler sur un Tetris nous est venue assez rapidement, avant même de commencer le projet. Mes camarades voulaient assembler leur projet de Science de l'ingénieur avec le codage de ce jeu afin de pouvoir afficher une partie de Tetris sur leur dispositif « HoloVision ». L'idée m'a parue intéressante car j'envisage de m'orienter dans le domaine des jeux-vidéos, après une formation en informatique en école d'ingénieur.

1.2. Ce que je vais présenter

Avec ce dossier, je vais vous présenter la démarche qui nous a permis de coder un jeu Tetris personnalisé. J'aborderai les concepts de bases du Tetris, faisant un petit point culturel puis je présenterai un cahier des charges. Ce dernier a été réparti et j'expliquerai les fonctions sur lesquelles j'ai travaillé, mentionnant les erreurs et les difficultés rencontrées. Je ferai part de mes impressions tout au long du projet et des perspectives d'amélioration envisageables si j'avais eu plus de temps. En annexe se trouve le Curriculum Vitae demandé ainsi que les liens vers les images utilisées et le code en entier avec, encadré en orange, la majeure partie de mon programme.

2. Qu'est-ce qu'un Tetris ?



Figure 1: Logo du jeu Tetris

Tetris est un jeu vidéo de puzzle conçu principalement par **Alekseï Pajitnov** en juin 1984 sur **Elektronika 60**.

Le jeu est édité par plusieurs sociétés différentes au fil des années, à la suite d'une guerre pour l'appropriation des droits à la fin des années 1980.

Après une exploitation massive par Nintendo, les droits reviennent depuis 1996 à la société The Tetris Company. Cette dernière vendit les droits d'exploitation sur mobiles à Electronic Arts et sur consoles à Ubisoft.

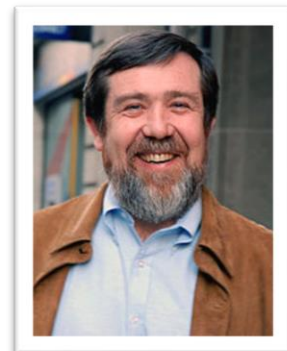


Figure 2 : Alekseï Pajitnov

Tetris met le joueur au défi de réaliser des lignes complètes en déplaçant des pièces de formes différentes, les *tétrominos*, qui défilent depuis le haut jusqu'au bas de l'écran. Les lignes complétées disparaissent et rapportent des points. Le joueur peut remplir une nouvelle fois les cases libérées.

Le jeu n'a pas de fin. Le joueur ne peut que perdre, c'est-à-dire lorsqu'un tétrmino reste bloqué en haut. Il doit donc résister le plus longtemps possible à la chute continue des tétrominos, pour faire le meilleur score.



Figure 3 : Une Elektronika 60

Selon le journaliste Bill Kunkel, « Tetris répond parfaitement à la définition du meilleur en matière de jeu : une minute pour l'apprendre, une vie entière pour le maîtriser ».



Figure 4: Image du jeu Pac-man

En effet, bâti sur des règles simples et exigeant intelligence et adresse, il est considéré comme un des grands classiques de l'histoire du jeu vidéo aux côtés de *Pong*, *Space Invaders* ou encore *Pac-Man*. Le jeu est adapté sur la plupart des supports de jeu, aussi bien sur ordinateurs que sur consoles de jeux et téléphones mobiles.

multijoueur.

Certaines versions apportent des variantes comme un affichage 3D, un système de réserve ou encore du jeu

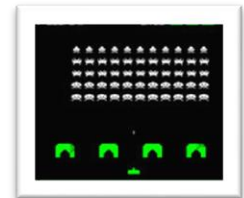


Figure 5 : Image du jeu Space Invaders

1.3. Le concept et les règles

1.1.1. Le Gameplay

Tetris est principalement composé d'un « champ de jeu » où des pièces de formes différentes, appelées « tétriminis », descendent du haut de l'écran.

Durant cette descente le joueur peut uniquement déplacer les pièces de manière latérale et leur faire effectuer une rotation sur elles-mêmes jusqu'à ce qu'elles touchent le bas du champ de jeu où une pièce est déjà placée. Le joueur ne peut ni ralentir la chute des pièces, ni l'empêcher, mais il peut dans certaines versions l'accélérer.

Le but pour le joueur est de réaliser le plus de lignes possibles, afin de garder de l'espace pour placer les futures pièces. Une fois une ligne complétée, elle disparaît, et les blocs placés au-dessus chutent d'un rang. Si le joueur ne parvient pas à faire disparaître les lignes suffisamment vite, l'écran peut alors se remplir jusqu'en haut. Lorsqu'un tétrmino dépasse du champ de jeu, et empêche l'arrivée de tétriminis supplémentaire, la partie se termine. Le joueur obtient un score, qui dépend essentiellement du nombre de lignes réalisées lors de la partie.

1.1.2. Les pièces


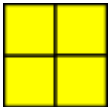
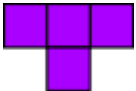


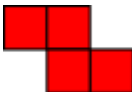
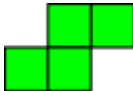
Les pièces de *Tetris*, sur lesquelles repose entièrement le jeu, sont des « tétriminos ». Le nom a évolué au cours du temps : alors que les mathématiciens préfèrent le terme « tétromino », les jeux *Tetris* des années 1990 utilisent différents termes : « blocs », « tétramino », etc.

Dans les années 2000, le terme officiel devient « tétrimino » dans les guidelines. Il en existe sept formes différentes, toutes basées sur un assemblage de quatre carrés, d'où le mot « Tetris » du préfixe grec *Tetra-* qui signifie « quatre ». Le joueur peut faire tourner plusieurs fois, à gauche et/ou à droite selon la version, de 90° n'importe quel bloc pour le poser de la façon désirée pendant qu'il descend.

1.4. Le cahier des charges et la répartition des tâches

1.1.3. Conception des tâches à effectuer

Grâce aux recherches effectuées, nous sommes en mesure de définir des fonctions principales qui formeront le corps de notre programme.

Forme	Appellation	Autres appellations	Construction
	Tétrimino I	« Bâton », « ordinaire », « barre », « long »	Quatre carrés alignés.
	Tétrimino O	« Carré », « bloc »	Méta-carré de 2x2.
	Tétrimino T	« Té »	Trois carrés en ligne et un carré sous le centre
	Tétrimino L	« L », « lambda »	Trois carrés en ligne et un carré sous le côté gauche.
	Tétrimino J	« L inversé », « gamma »	Trois carrés en ligne et un carré sous le côté droit.
	Tétrimino Z	« Biais »	Méta-carré de 2x2, dont la rangée supérieure est glissée d'un pas vers la gauche.
	Tétrimino S	« Biais inversé »	Méta-carré de 2x2, dont la rangée supérieure est glissée d'un pas vers la droite.

Le programme devra :

- Afficher de manière aléatoire un des sept blocs définis par le jeu
- Afficher une grille de jeu
- Permettre la rotation à 90° de chaque pièce
- Permettre la descente des pièces
- Permettre le déplacement des pièces dans l'espace défini
- Augmenter la vitesse avec l'augmentation de niveau
- Détruire chaque ligne complétée
- Afficher un score en fin de partie
- Faire tourner une musique en boucle

Afin d'apporter de l'originalité dans notre programme, nous mettrons en place :

- Une fin où le joueur gagne
- Des boss à chaque étape de difficulté avec réinitialisation à chaque palier
- Un affichage mixé avec un jeu plus récent, **Undertale**



Figure 6 : Image du logo Undertale

Matériel utilisé : langage python, extension Tkinter pour l'affichage et pygame pour le son

Qu'est-ce que *Tkinter* et *pygame* ?

Tkinter (Tk interface) est un module intégré à la bibliothèque standard de Python, bien qu'il ne soit pas maintenu directement par les développeurs de Python. Il offre un moyen de créer des interfaces graphiques *via* Python.

Pygame est une bibliothèque libre multiplate-forme qui facilite le développement de jeux vidéo temps réel avec le langage de programmation Python.



Figure 7 : Image logo Pygame

Construite sur la bibliothèque SDL, elle permet de programmer la partie multimédia (graphismes, son et entrées au clavier, à la souris ou au joystick), sans se heurter aux difficultés des langages de bas niveaux comme le C et ses dérivés. Cela se fonde sur la supposition que la partie multimédia, souvent la plus contraignante à programmer dans un tel jeu, est suffisamment indépendante de la logique même du jeu pour qu'on puisse utiliser un langage de haut niveau (en l'occurrence le Python) pour la structure du jeu.

1.1.4. La démarche collaborative

Dans un premier temps, nous avons réfléchi au projet final que nous allions proposer au baccalauréat. L'idée de reprendre un jeu simple et « codable » nous a paru une bonne idée pour plonger dans l'univers du jeu mais aussi de la programmation.

Après avoir décidé de faire un *Tetris*, nous avons élaboré un tableau de répartition des tâches en estimant le temps nécessaire pour coder chaque partie.

Tâche	Nombre de séances	Nom
Afficher une grille	1 séance	Lucas
Afficher différents blocs	2 séances	Deryne
Disposer des images	2 séances	Stéphane
Attribuer une couleur par bloc	1 séance	Stéphane
Tirer les blocs aléatoirement	1 séance	Deryne
Faire bouger les blocs	2 séances	Deryne
Détruire les lignes complètes	2 séances	Lucas
Afficher un score	2 séances	Lucas
Créer un logo	1 séance	Stéphane
Etablir des niveaux	1 séance	Deryne
Mettre une musique de fond	1 séance	Deryne

Moyens de communication : Afin de faire part de nos avancés mutuelles, nous avons créé un dossier Dropbox "ISN" et trois sous-dossiers individuels. J'ai organisé le mien en 3 parties distinctes :

- Les programmes en test : où j'expérimentais des fonctions diverses
- Les exemples : où je m'intéressais aux programmes de Tetris déjà existant
- Les étapes réussies : où je sauvegardais chaque version de mon programme au fur et à mesure

3. Développement

1.5. Ma réalisation

1.1.5. Afficher différents blocs aléatoirement : (3 séances)

Comme décrit dans l'introduction, je devais créer sept blocs différents. Sur plusieurs sites, j'ai remarqué que les pièces étaient définies suivant une liste de coordonnées. Elles avaient également quatre positions car la pièce doit pouvoir tourner de 90°. Chaque carré qui composait les différents blocs étaient un espace "plein" tandis que les autres cases, où la pièce n'apparaissait pas, étaient comme "vide".

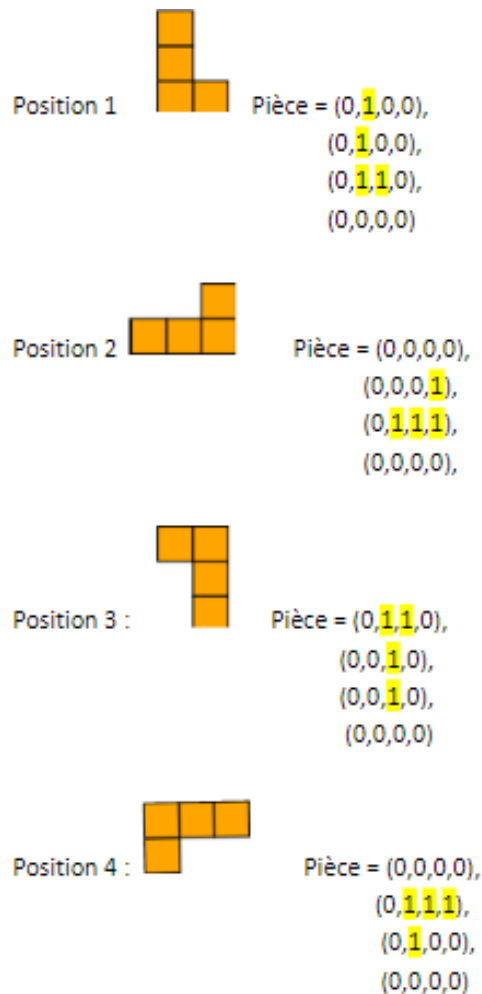


Figure 8 : Exemple de Tetrimino L

J'ai procédé ainsi pour les six autres pièces pour lesquelles j'ai réalisé un code afin qu'elles s'affichent. La fonction *début()* permettait ainsi de choisir un bloc à afficher. On crée une fonction *imprime_Piece(sens)* pour afficher le bloc suivant son sens, dû à ses coordonnées, dans un *tableau_grille*. Un problème survient alors : c'est la même pièce qui est affichée en boucle ! D'où la deuxième étape, insérer le caractère aléatoire des blocs. Pour cela on demande, par "for randrange import", une extension de python, et d'une variable « n » aléatoire, on demande à choisir un nombre entre 1 et 7 (correspondant au nombre de pièces disponibles) et d'afficher le bloc correspondant au chiffre choisi intitulé "index" :

```
tableau_Pieces=(Piece1,Piece2,Piece3,Piece4,Piece5,Piece6,Piece7)
```



```
index = randrange(0,len(tab_pieces))
Piece = tableau_Pieces[index]
```

1.1.6. Faire bouger les blocs : (2 séances)

Afin de faire pivoter et déplacer les différents blocs, un par un, on utilise la fonctionnalité `fen.bind()`. La fonction "bind" permet d'associer une fonction (à 1 argument) à un événement et à un widget. Les événements les plus courants sont les clics souris et la saisie au clavier. On note 5 actions possibles avec le clavier pour jouer :

La flèche directionnelle "droite" pour aller à droite dans la grille
La flèche directionnelle "gauche" pour aller à gauche dans la grille
La flèche directionnelle "haut" pour faire pivoter la pièce
La flèche directionnelle "bas" pour faire tomber la pièce
La flèche directionnelle "bas appuyé" lorsqu'on désire que la pièce tombe plus vite

On crée ainsi 5 fonctions :

```
fen.bind('<Right>', droite)
fen.bind('<Left>', gauche)
fen.bind('<Up>', tourne)
fen.bind('<Down>', tombe)
fen.bind('<KeyRelease-Down>', tombe_Relache)
```

Ces 5 informations différentes permettent alors 5 fonctions différentes qui ne s'activeront que lorsque les capteurs du clavier auront donné une valeur.

a) Les fonctions `droite(event)` et `gauche(event)`:

Le Tableau créé par Lucas comprend des 1 et des 0. Un 1 correspond aux limites de la grille et un zéro à la partie "disponible" pour faire bouger la pièce. Comme expliqué précédemment, chaque bloc fonctionne dans un carré de côté 4. La pièce va se modifier dans l'espace aussi a-t-on besoin "d'effacer" la pièce de sa position initiale en haut de la grille, de sa zone (en haut au centre) de "*Spawn*" (nom masculin singulier qui correspond à la réapparition de mobs, d'objets ou de joueurs et ici, de la pièce).

On crée donc un petit tableau où la pièce pourra pivoter dedans et on vérifie que la pièce s'affiche bien à ses coordonnées initiales. On vérifie par récursivité si une ligne au minimum a été construite. Cette vérification se réalise par le bas. Par défaut la ligne est considérée complète. Pour chaque ligne on vérifie chaque colonne. Si la valeur de la matrice à cette coordonnée vaut zéro la ligne n'est pas complète et on passe à la ligne suivante. Lorsqu'une ligne est complète, on la supprime, on rajoute une ligne de "zéro", une ligne de "vide" et on refait une vérification par le bas.

On utilise les coordonnées courantes pour déplacer la pièce. Aller à droite correspond à se déplacer, sur l'axe des abscisses, de manière positive "+1" tandis qu'aller à gauche dans la grille correspond à un "-1". On déplace ainsi le carré qui contient la pièce puis on affiche la pièce dans la même position qu'au début de la pression de la flèche directionnelle.

b) Les fonctions `tombe(event)`, `tombe_Relache(event)` et `tourne(event)`

Comme les deux fonctions précédentes, on a affaire à un événement dû à la pression d'une touche ici, soit la touche "bas" soit la touche "haut". La touche "bas" lorsqu'elle est sollicitée, augmente la vitesse de descente de la pièce. Cette valeur de vitesse diminue dans une autre fonction qui redéfinit la vitesse comme "normale" et plus "accélérée".

La fonction `tourne(event)` concerne la pression de la touche "haut" qui permet le pivotement à 90° de la pièce en jeu. A chaque pression de la touche, le sens est incrémenté. Si le sens est égal à 4 alors la

pièce vient de réaliser un 360° et se retrouve à sa place initiale. Enfin, on finit par afficher la pièce dans le sens demandé.

Une fonction pour descendre le bloc est également créée, elle incrémente la position du bloc dans le grille suivant la vitesse dite "normale".

1.1.7. Créer des niveaux (1 séance)

Le jeu fonctionnant sur le principe d'un boss, il fallait que ce dernier change en fonction de la progression du joueur. La première réflexion consistait à recréer une fonction `début()` avec de nouveaux paramètres comme l'image du boss, son texte, ses points de vie, le score du joueur, son nom. Cependant un tel travail n'engendre que des lignes de code qui existent déjà aussi ai-je défini dans la fonction de départ ces 5 paramètres. Avec une telle méthode, il suffisait d'insérer initialisation avec 5 nouveaux paramètres.

Afin que les niveaux se déclenchent au fur et à mesure, j'ai ajouté dans une fonction qui fait une vérification dans la boucle de la ligne complète : si les points de vie du boss sont à zéro et que le niveau correspond au premier, alors on appelle la fonction `levelup2()` qui contient une nouvelle initialisation avec de nouveaux paramètres. Lorsque le deuxième boss est vaincu, on appelle la fonction `levelup3()` pour une nouvelle initialisation.

1.1.8. Insérer de la musique (1 séance)

Afin d'incruster une musique pour rendre la partie plus "ludique", j'ai dû ajouter au programme une extension "pygame" qui permet d'ajouter au programme de la musique. On crée donc une fenêtre d'où la musique "sort". Comme je désirais une musique en boucle, j'ai ajouté, selon les tutoriels que j'ai cherché :

```
son.play(loops=-1, maxtime=0, fade_ms=0)
```

Dans un premier temps j'avais inséré cette commande dans l'initialisation mais lorsque l'on passait au niveau suivant et donc à une nouvelle initialisation, une deuxième fenêtre apparaissait et l'on se retrouvait avec deux musiques en boucle. J'ai réglé le problème en plaçant la musique en dehors de l'initialisation. Le type de fichier musical accepté étant du .WAV, j'ai converti la musique souhaitée "Undertale Megalovania".

Dans un deuxième temps, j'ai réalisé que cette musique possédait des droits d'auteur car elle était sortie en 2015. Il ne m'a pas semblé pertinent de garder cette musique, aussi sympa soit-elle, et j'ai opté pour une des premières musiques de Tetris, un extrait de l'acte 2 de *Casse-noisette*, la *Danse de la Fée Dragée*. Cette musique créée par Tchaïkovski, datant de la fin du 19ème siècle, appartenait désormais à tous.

A chaque ligne complétée, j'ai ajouté un son sortant d'une banque de son gratuits et libres de droits, un bruit de laser. J'ai réalisé la même chose avec un bruit d'épée pour chaque niveau passé.

Un dernier problème s'est posé lorsque le jeu se termine soit par une victoire soit un échec : la musique continue d'être diffusée. J'ai donc ajouté la commande `son.stop()` en fin de programme.

4. Mes impressions

1.6. Une démarche réflexive

Ce projet m'a permis de m'enrichir sur le plan « geek » car je ne connaissais pas vraiment l'origine du jeu *Tetris*.

Ma partie de programmation a nécessité un long travail de recherche permanent. N'ayant jamais codé auparavant en python, j'ai dû m'intéresser aux programmes de python existant déjà pour savoir comment procéder. Afin de le distinguer des autres Tetris python que j'ai vus, j'ai eu l'idée de créer des boss afin de rendre plus original le *GamePlay* et d'insérer des animations sonores.

J'ai trouvé ce travail enrichissant car il m'a permis d'exprimer ma créativité et mon imagination malgré la frustration que j'ai souvent eue lorsque le programme comportait des bugs ou ne faisait pas ce que je voulais. Il est néanmoins gratifiant de voir apparaître (enfin) des blocs aléatoires ou de voir le niveau 2 se déclencher sans qu'une seconde musique n'apparaisse en boucle.

Ce travail m'a sans aucun doute coûté plus de temps que déclaré précédemment, je n'ai eu cesse de revenir dessus et de modifier, de tester. Il m'a permis de découvrir de nouvelles fonctionnalités de python mais aussi de pygame et de Tkinter, une extension pour l'affichage car j'ai aussi touché par curiosité l'aspect graphique.

1.7. Mes difficultés

De nombreuses petites erreurs ont rendu mon avancée plus pénible. Au début, j'oubliais assez souvent l'indentation ou les « : » en fin de définition de fonctions, ce qui me faisait chercher pendant plusieurs minutes la source du problème avant de me rendre compte qu'il y avait un oubli de ponctuation. La diffusion de la musique, expliquée précédemment, m'a obligé à revoir sa position initiale dans la boucle de début() pour la placer avant même cette fonction, évitant ainsi d'avoir plusieurs fenêtres de sons et de jouer avec des fonctionnalités pour couper la musique.

5. Conclusion

1.8. Ce que j'ai appris

Grâce au projet, j'ai appris de nouvelles fonctions et j'ai touché à des extensions très intéressantes de python. Cela m'a contrainte à respecter une limite de temps et m'a fait toucher du doigt le rapport heure de travail et résultat visible, ce qui me permet aujourd'hui d'avoir une idée petite, mais plus précise, du temps consacré à créer des univers en 3D ou même en 2D.

1.9. Perspectives d'amélioration

Par manque de temps et de compétences, je n'ai fait qu'effleurer les possibilités inimaginables que j'aurai pu ajouter au programme. Même si la plupart des gens connaissent le principe du *Tetris*, j'aurai pu ajouter une fonctionnalité proposant un tutoriel écrit pour expliquer son fonctionnement.

La sauvegarde des scores, assez complexe, a été ignorée par manque de temps mais reste une idée majeure pour l'amélioration du programme car il est un élément important du jeu.

J'ai tenté en fin de conception d'afficher une autre image de chaque boss lorsque les points de vie étaient bas, une image de fureur mais je n'ai pas réussi à coder ce que je voulais.

6. Annexes

1.10. Liens

1.1.9. Liens vers les images :

- Fig1 : [https://fr.wikipedia.org/wiki/Fichier:The Tetris Company Logo.png](https://fr.wikipedia.org/wiki/Fichier:The_Tetris_Company_Logo.png)
- Fig2 : [https://commons.wikimedia.org/wiki/File:Alexey Pajitnov - 2575833305 \(crop\).jpg?uselang=fr](https://commons.wikimedia.org/wiki/File:Alexey_Pajitnov_-_2575833305_(crop).jpg?uselang=fr)
- Fig3 : <https://commons.wikimedia.org/wiki/File:E60M.JPG?uselang=fr>
- Fig4 : <http://gamezone.2001jeux.fr/play-00562-b-pacman.html>
- Fig5 : <http://gamezone.2001jeux.fr/play-01219-b-space-invaders.html>
- Fig6 : [https://fr.m.wikipedia.org/wiki/Fichier:Undertale vector logo on black borders.png](https://fr.m.wikipedia.org/wiki/Fichier:Undertale_vector_logo_on_black_borders.png)
- Fig 7 : <http://www.pygame.org/docs/logos.html>

1.1.10. Liens vers les musiques:

- Casse-noisette :
[https://commons.wikimedia.org/w/index.php?title=File%3ADance of the Sugar Plum Fairies \(ISRC USUAN1100270\).oga](https://commons.wikimedia.org/w/index.php?title=File%3ADance_of_the_Sugar_Plum_Fairies_(ISRC_USUAN1100270).oga)
- Bruitages :
<https://www.sound-fishing.net>

1.11. Code du programme

Ce programme est un Tetris combiné avec des boss de niveaux

```
from tkinter import *  
# Extension pour afficher des images
```

```
from time import sleep  
#Extension utile pour demander au programme de faire des pauses
```

```
from random import randrange  
#Extension servant à importer le caractère aléatoire
```

```
import pygame  
# Extension pour la musique
```

```
#-----  
# Code pour insérer la musique en boucle  
pygame.init()
```

```
#On initialise pour la musique  
fenetre = pygame.display.set_mode((3,3))
```

```
#Ouverture d'une fenêtre pour le son  
pygame.mixer.init()  
son = pygame.mixer.Sound('Casse-noisette.wav')  
son.play(loops=-1, maxtime=0, fade_ms=0)
```

```
#-----
```

```
### Entrée : niveau, score, img, nom_boss, vie, parole  
### Sortie : Affichage d'un tableau, des pièces, des images, des textes  
### Fonction : Elle a pour but de faire débiter la partie et d'afficher les\  
# différents éléments
```

```
def début(niveau=1,score=0,img='Toriel7.gif',nom_boss="Toriel",\  
vie=1200,parole="Je te briserai les os !"):
```

```
    global tableau_canvas,tableau_grille,tableau_couleur,coord_normales,Piece,label_vie,\  
    label_nom,score_score,Points_de_vie,level
```

```
    # On insère les variables des paramètres dans la fonction
```

```
    score_score = score  
    Points_de_vie = vie  
    level = niveau  
    image = img
```

```
    # Tableau de couleur pour les pièces
```

```
    tableau_couleur=['gray100','green3','goldenrod','red','dark turquoise','dark orchid',\  
                    'dark red','dark blue','dark green','dark cyan','dark magenta',\  
                    'dark brown','dark grey','dark blue','dark green','dark cyan',\  
                    'dark magenta','dark brown','dark grey']
```

'DarkOrange','pale violet red']

On crée d'un tableau de 20 lignes et 10 colonnes avec sa matrice
On définit les pièces et leurs positions en fonction de coordonnées

```
tableau_canvas=[]
tableau_grille=[]
for i in range(20):
    tableau_canvas.append([[]]*10)
    tableau_grille.append([[]]*10)

grille_jeu = Frame(fen)
for ligne in range(20):
    for colonne in range(10):
        couleur = tableau_couleur[0]
        tableau_canvas[ligne][colonne] = Canvas(grille_jeu,bg=couleur,height=20,
            width=20,borderwidth=1,relief=FLAT)
        tableau_canvas[ligne][colonne].grid(row=ligne,column=colonne)
        tableau_grille[ligne][colonne] = 0
```

#-----

On crée 4 positions pour chaque pièces qui sont au nombre de 7

```
Piece1 = [[(0,1,0,0),(0,1,1,0),(0,0,1,0),(0,0,0,0)], #0 1 0 0
            [(0,0,0,0),(0,0,1,1),(0,1,1,0),(0,0,0,0)], #0 1 1 0
            [(0,1,0,0),(0,1,1,0),(0,0,1,0),(0,0,0,0)], #0 0 1 0
            [(0,0,0,0),(0,0,1,1),(0,1,1,0),(0,0,0,0)]] #0 0 0 0
```

```
Piece2 = [[(0,2,2,0),(0,2,2,0),(0,0,0,0),(0,0,0,0)], #0 2 2 0
            [(0,2,2,0),(0,2,2,0),(0,0,0,0),(0,0,0,0)], #0 2 2 0
            [(0,2,2,0),(0,2,2,0),(0,0,0,0),(0,0,0,0)], #0 0 0 0
            [(0,2,2,0),(0,2,2,0),(0,0,0,0),(0,0,0,0)]] #0 0 0 0
```

```
Piece3 = [[(0,3,0,0),(0,3,0,0),(0,3,0,0),(0,3,0,0)], #0 3 0 0
            [(3,3,3,3),(0,0,0,0),(0,0,0,0),(0,0,0,0)], #0 3 0 0
            [(0,3,0,0),(0,3,0,0),(0,3,0,0),(0,3,0,0)], #0 3 0 0
            [(3,3,3,3),(0,0,0,0),(0,0,0,0),(0,0,0,0)]] #0 3 0 0
```

```
Piece4 = [[(0,0,4,0),(0,4,4,0),(0,4,0,0),(0,0,0,0)], #0 0 4 0
            [(0,0,0,0),(0,4,4,0),(0,0,4,4),(0,0,0,0)], #0 4 4 0
            [(0,0,4,0),(0,4,4,0),(0,4,0,0),(0,0,0,0)], #0 4 0 0
            [(0,0,0,0),(0,4,4,0),(0,0,4,4),(0,0,0,0)]] #0 0 0 0
```

```
Piece5 = [[(0,5,0,0),(0,5,5,0),(0,5,0,0),(0,0,0,0)], #0 5 0 0
            [(0,0,0,0),(0,0,5,0),(0,5,5,5),(0,0,0,0)], #0 5 5 0
            [(0,0,0,5),(0,0,5,5),(0,0,0,5),(0,0,0,0)], #0 5 0 0
            [(0,5,5,5),(0,0,5,0),(0,0,0,0),(0,0,0,0)]] #0 0 0 0
```

```
Piece6 = [[(0,0,6,0),(0,0,6,0),(0,6,6,0),(0,0,0,0)], #0 0 6 0
            [(0,0,0,0),(0,6,6,6),(0,0,0,6),(0,0,0,0)], #0 0 6 0
            [(0,6,6,0),(0,6,0,0),(0,6,0,0),(0,0,0,0)], #0 6 6 0
            [(0,0,0,0),(0,6,0,0),(0,6,6,6),(0,0,0,0)]] #0 0 0 0
```

```
Piece7 = [[(0,7,0,0),(0,7,0,0),(0,7,7,0),(0,0,0,0)], # 0 7 0 0
          [(0,0,0,0),(0,0,0,7),(0,7,7,7),(0,0,0,0)], # 0 7 0 0
          [(0,7,7,0),(0,0,7,0),(0,0,7,0),(0,0,0,0)], # 0 7 7 0
          [(0,0,0,0),(0,7,7,7),(0,7,0,0),(0,0,0,0)]] # 0 0 0 0

# On tire une première pièce de manière aléatoire entre 1 et 7
tab_Pieces =(Piece1,Piece2,Piece3,Piece4,Piece5,Piece6,Piece7)
n = randrange(1,7)
Piece = tab_Pieces[n]
```

```
#-----
```

#On crée un bandeau noir à gauche de la grille

```
grille_jeu.grid(row=1,column=0)
boss = Frame(fen,bg='gray14',width=200,height=605)
frame_suivante = Frame(boss,bg='green')
frame_suivante = Canvas(width = 1, height = 1, bg = 'gray1')
```

#On insère dedans l'image du boss "img" fixé dans les paramètres de départ

```
photo = PhotoImage(file = img )
item_boss = frame_suivante.create_image(70, 155, image = photo)
```

```
# On paramètre la taille et la place de l'image en format gif
frame_suivante.place(x=272,y=70,width=140,height=300)
boss.grid(row=1,column=1)
```

```
#-----
```

On ajoute le logo "UnderTris" de la même manière

```
frame_titre = Frame(boss, bg='red',width=200,height=200)
logo = PhotoImage(file = "Titre4.gif")
frame_titre = Canvas(width = 10, height = 1, bg = 'gray1')
titre_projet = frame_titre.create_image(125, 23, image = logo)
frame_titre.place(x=0,y=0,width=260,height=45)
```

```
#-----
```

On insère aussi le nom du boss

```
label_nom = Label(boss,text="BOSS\n"+str(nom_boss),
                  fg='white',font="Century 13 normal bold",
                  bg='gray1',relief=RIDGE)
label_nom.place(x=23,y=9,width=120,height=45)
```

```
#-----
```

On insère les points de vie

```
label_vie = Label(boss,text="PTS DE VIE\n"+str(Points_de_vie),
                  fg='white',font="Century 13 normal bold",
                  bg='gray1',relief=RIDGE)
label_vie.place(x=22,y=394,width=120,height=45)
```

```
#-----
```

On insère le niveau du joueur

```
label_niveau = Label(boss,text="NIVEAU\n"+str(niveau),
                    fg='white',font="Century 13 normal bold",
                    bg='gray1',relief=RIDGE)
label_niveau.place(x=23,y=444,width=120,height=45)
```

#-----

On insère les paroles du boss

```
label_parole= Label(boss,text="DIT: \n"+str(parole),
                    fg='white',font="Century 11 normal",
                    bg='gray1',relief=RIDGE)
label_parole.place(x=2,y=505,width=160,height=40)
```

#-----

On définit les touches utilisées pour jouer en ajoutant une fonction qui\

se déclenchera à chaque pression de la touche

```
fen.bind('<Right>',droite)
fen.bind('<Left>',gauche)
fen.bind('<Up>',tourne)
fen.bind('<Down>',tombe)
fen.bind('<KeyRelease-Down>',relache_bas)
```

```
maj_Grille()
nvelle_Piece()
mainloop()
```

#-----

Entrée : Piece,can_Piece_apres

Sortie : Pièce aléatoire suivante

Fonction : Elle a pour but de tirer la deuxième pièce aléatoire

def tirage():

global Piece,can_Piece_apres

```
Piece1 = [[(0,1,0,0),(0,1,1,0),(0,0,1,0),(0,0,0,0)], #0 1 0 0
           [(0,0,0,0),(0,0,1,1),(0,1,1,0),(0,0,0,0)], #0 1 1 0
           [(0,1,0,0),(0,1,1,0),(0,0,1,0),(0,0,0,0)], #0 0 1 0
           [(0,0,0,0),(0,0,1,1),(0,1,1,0),(0,0,0,0)]] #0 0 0 0
```

```
Piece2 = [[(0,2,2,0),(0,2,2,0),(0,0,0,0),(0,0,0,0)], #0 2 2 0
           [(0,2,2,0),(0,2,2,0),(0,0,0,0),(0,0,0,0)], #0 2 2 0
           [(0,2,2,0),(0,2,2,0),(0,0,0,0),(0,0,0,0)], #0 0 0 0
           [(0,2,2,0),(0,2,2,0),(0,0,0,0),(0,0,0,0)]] #0 0 0 0
```

```
Piece3 = [[[0,3,0,0),(0,3,0,0),(0,3,0,0),(0,3,0,0)], #0 3 0 0
           [(3,3,3,3),(0,0,0,0),(0,0,0,0),(0,0,0,0)], #0 3 0 0
           [(0,3,0,0),(0,3,0,0),(0,3,0,0),(0,3,0,0)], #0 3 0 0
           [(3,3,3,3),(0,0,0,0),(0,0,0,0),(0,0,0,0)]] #0 3 0 0
```

```
Piece4 = [[(0,0,4,0),(0,4,4,0),(0,4,0,0),(0,0,0,0)], #0 0 4 0
```



```

[(0,0,0,0),(0,4,4,0),(0,0,4,4),(0,0,0,0)], # 0 4 4 0
[(0,0,4,0),(0,4,4,0),(0,4,0,0),(0,0,0,0)], # 0 4 0 0
[(0,0,0,0),(0,4,4,0),(0,0,4,4),(0,0,0,0)] # 0 0 0 0

```

```

Piece5 = [[(0,5,0,0),(0,5,5,0),(0,5,0,0),(0,0,0,0)], # 0 5 0 0
[(0,0,0,0),(0,0,5,0),(0,5,5,5),(0,0,0,0)], # 0 5 5 0
[(0,0,0,5),(0,0,5,5),(0,0,0,5),(0,0,0,0)], # 0 5 0 0
[(0,5,5,5),(0,0,5,0),(0,0,0,0),(0,0,0,0)]] # 0 0 0 0

```

```

Piece6 = [[(0,0,6,0),(0,0,6,0),(0,6,6,0),(0,0,0,0)], # 0 0 6 0
[(0,0,0,0),(0,6,6,6),(0,0,0,6),(0,0,0,0)], # 0 0 6 0
[(0,6,6,0),(0,6,0,0),(0,6,0,0),(0,0,0,0)], # 0 6 6 0
[(0,0,0,0),(0,6,0,0),(0,6,6,6),(0,0,0,0)]] # 0 0 0 0

```

```

Piece7 = [[(0,7,0,0),(0,7,0,0),(0,7,7,0),(0,0,0,0)], # 0 7 0 0
[(0,0,0,0),(0,0,0,7),(0,7,7,7),(0,0,0,0)], # 0 7 0 0
[(0,7,7,0),(0,0,7,0),(0,0,7,0),(0,0,0,0)], # 0 7 7 0
[(0,0,0,0),(0,7,7,7),(0,7,0,0),(0,0,0,0)]] # 0 0 0 0

```

```

# On tire une première pièce de manière aléatoire entre 1 et 7
tab_Pieces =(Piece1,Piece2,Piece3,Piece4,Piece5,Piece6,Piece7)
index = randrange(0,len(tab_Pieces))
Piece = tab_Pieces[index]

```

```

#-----
### Entrée : coord_normales,sens,vitesse_normale,vitesse,level
### Sortie : Affichage score, vitesse
### Fonction : Elle a pour but de définir une vitesse en fonction du niveau\
               # et d'arrêter le jeu quand le joueur a perdu

```

```
def nvelle_Piece():
```

```
    global coord_normales,sens,vitesse_normale,vitesse,level
```

```
    vitesse_normale = 550 - (level*50)
```

```
    coord_normales = [0,3]
```

```
    sens = 0
```

```
    sleep(1)
```

```
    tirage()
```

```
    if not Mouvement_verif(0,0,0):
```

```
        print_Piece(sens)
```

```
        # On arrête la musique
```

```
        son.stop()
```

```
        def fairetoplevel() :
```

```
            top=Toplevel(root)
```

```
            lab=Label(top, text=score_score, bg="gray100")
```

```
            lab.place(x=0,y=0,width=260,height=45)
```

```
            top.geometry("%dx%d%d%d" % (424,560,26,26))
```

```
            lab.pack()
```

```
    root = Tk()
```

```
    root.geometry("%dx%d%d%d" % (24,60,26,26))
```

```
    go=Button (root , text="SCORE" , command=fairetoplevel)
```

```
go.pack()
root.mainloop()

else:
    print_Piece(sens)
    sleep(1)
    descente()

#-----
### Entrée : Piece_apres,vitesse,vitesse_normale
### Sortie : Affichage pièce avec sens
### Fonction : Elle a pour but de faire descendre ligne par ligne le bloc\
               #si c'est possible.Autrement on appelle nvelle_Piece avec Piece_apres\
               #en parametre.

def descente():

    global Piece_apres,vitesse,vitesse_normale

    Piece_Efface()
    vitesse = vitesse_normale

    if Mouvement_verif(1,0,0):
        coord_normales[0] += 1
        print_Piece(sens)
        fen.after(vitesse,descente)
    else :
        print_Piece(sens)
        verif_Ligne()
        nvelle_Piece()

#-----
### Entrée : \
### Sortie : Affichage pièce avec sens
### Fonction : Elle a pour but d'afficher une pièce quand elle est allée à gauche

def gauche(event):

    Piece_Efface()
    if Mouvement_verif(0,-1,0):
        coord_normales[1] -=1
        print_Piece(sens)
    else:
        print_Piece(sens)
    return

#-----
### Entrée : \
### Sortie : Affichage pièce avec sens
### Fonction : Elle a pour but d'afficher une pièce quand elle est allée à droite

def droite(event):

    Piece_Efface()
```

```
if Mouvement_verif(0,1,0):
    coord_normales[1] += 1
    print_Piece(sens)
else:
    print_Piece(sens)
return
```

```
#-----
### Entrée : sens
### Sortie : Affichage pièce avec sens
### Fonction : Elle a pour but d'afficher une pièce quand elle a tourné
```

```
def tourne(event):
```

```
    global sens
```

```
    Piece_Efface()
    if Mouvement_verif(0,0,1):
        sens += 1
        if sens == 4:
            sens = 0
        print_Piece(sens)
    else:
        print_Piece(sens)
```

```
#-----
### Entrée : vitesse
### Sortie : Affichage pièce avec vitesse rapide
### Fonction : Elle a pour but d'accélérer la descente de la pièce quand on\
    # maintient la touche "bas" enfoncée
```

```
def tombe(event):
```

```
    global vitesse
```

```
    vitesse = 10
```

```
#-----
### Entrée : \
### Sortie : Affichage pièce avec vitesse normale
### Fonction : Elle a pour but de faire ralentir la pièce quand on lâche la\
    #touche "bas"
```

```
def relache_bas(event):
```

```
    global vitesse,vitesse_normale
```

```
    vitesse = vitesse_normale
```

```
#-----
### Entrée : \
### Sortie : Afficher pièce
### Fonction : Elle a pour but d'effacer la pièce de la matrice par rapport à\
    # sa coordonnée courante
```

```
def Piece_Efface():

    for i in range(4):
        for j in range(4):
            if Piece[sens][i][j] == 0:
                continue
            tableau_grille[coord_normales[0]+i][coord_normales[1]+j] = 0

#-----
### Entrée : sens
### Sortie : Affichage pièce
### Fonction : Elle a pour but d'effacer la pièce de la matrice par rapport à\
# sa coordonnée courante et de passer les coordonnées dans la matrice à\
# la valeur de la pièce

def print_Piece(sens):

    for i in range(4):
        for j in range(4):
            if Piece[sens][i][j] == 0:
                continue
            tableau_grille[coord_normales[0]+i][coord_normales[1]+j] = \
                Piece[sens][i][j]

    maj_Grille()
```

```
#-----
### Entrée : dx, dy, pivot
### Sortie : Affichage pièce
### Fonction : Elle a pour but de vérifier si la pièce ne débordera pas de la\
# matrice. Si c'est le cas on renvoie un déplacement non possible à la\
# méthode appelante. Si non, on teste si elle ne chevauchera pas des\
# blocs déjà présent
```

```
def Mouvement_verif(dx,dy,pivot):

    rotation = sens + pivot
    if rotation == 4:
        rotation = 0
    for i in range(4):
        for j in range(4):
            if coord_normales[1]+(dy+j) > 9 or coord_normales[1]+(dy+j) < 0 or \
               coord_normales[0]+(dx+i) > 19:
                if Piece[rotation][i][j] != 0:
                    return False
            else:
                continue
            if (Piece[rotation][i][j] * tableau_grille[coord_normales[0]+(dx+i)] \
               [coord_normales[1]+j+dy]) != 0:
                return False
    return True
```

```
#-----
```

```
### Entrée : score, Points_de_vie,label_vie,label_nom,nom_boss,score_score,level
### Sortie : Affichage score et points de vie du boss, affichage si le joueur\
#gagne
### Fonction : Elle a pour but de vérifier par récursivité si au moins une\
# ligne à été construite
```

```
def verif_Ligne():
```

```
    global score,Points_de_vie,label_vie,label_nom,nom_boss,score_score,level
```

```
    for ligne in range(19,-1,-1):
```

```
        ligne_complete = True
```

```
        for colonne in range(10):
```

```
            if tableau_grille[ligne][colonne] == 0:
```

```
                ligne_complete = False
```

```
    if ligne_complete:
```

```
        del tableau_grille[ligne]
```

```
        tableau_grille[0:0] = [[0,0,0,0,0,0,0,0,0,0]]
```

```
        maj_Grille()
```

```
        verif_Ligne()
```

```
        score_score = score_score + 100
```

```
        Points_de_vie = Points_de_vie - 100
```

```
        label_vie.configure(text="PTS DE VIE\n"+str(Points_de_vie))
```

```
    # Animation sonore quand une ligne est détruite
```

```
    pygame.mixer.music.load("heat-vision.wav")
```

```
    pygame.mixer.music.play()
```

```
    #Retourne la valeur du volume, entre 0 et 1
```

```
    volume = pygame.mixer.music.get_volume()
```

```
    pygame.mixer.music.set_volume(1)
```

```
    # Passage au niveau 2
```

```
    if Points_de_vie == 0 and level == 1:
```

```
        levelup()
```

```
    # Passage au niveau 3
```

```
    if Points_de_vie == 0 and level == 2:
```

```
        levelup2()
```

```
    #Passage au niveau 4
```

```
    if Points_de_vie == 0 and level == 3:
```

```
        levelup3()
```

```
    # Fin du jeu
```

```
    if Points_de_vie == 0 and level == 4:
```

```
        print("Bravo")
```

```
        son.stop()
```

```
#-----
```

```
### Entrée : \
```

```
### Sortie : Initialisation avec de nouveaux paramètres
### Fonction : Elle a pour but de passer au niveau suivant

def levelup():
    début(niveau=2,score=1200,img="Undyne1.gif",nom_boss="Undyne",\
        vie=2000,parole="Capitulez !")

#-----
### Entrée : \
### Sortie : Initialisation avec de nouveaux paramètres
### Fonction : Elle a pour but de passer au niveau suivant

def levelup2():
    début(niveau=3,score=3200,img="Papyrus2.gif",nom_boss="Papyrus",\
        vie=3000,parole="Retournez au néant !")

#-----
### Entrée : \
### Sortie : Initialisation avec de nouveaux paramètres
### Fonction : Elle a pour but de passer au niveau suivant

def levelup3():
    début(niveau=4,score=6200,img="undertale3.gif",nom_boss="Sans",\
        vie=4000,parole="L'échec est inevitable !")

#-----
### Entrée : \
### Sortie : Initialisation avec de nouveaux paramètres
### Fonction : Elle a pour but mettre a jour des canevas en fonction des\
    # couleurs référencées par les valeurs de la matrice

def maj_Grille():
    for ligne in range(20):
        for colonne in range(10):
            couleur = tableau_couleur[tableau_grille[ligne][colonne]]
            tableau_canvas[ligne][colonne].configure(bg=couleur)

    fen.update()

#-----

fen = Tk()
fen.wm_geometry(newGeometry='+210+0')
fen.resizable(0,0)

# On modifie la taille de la fenêtre
fen.geometry("%dx%d%d%d" % (424,560,26,26))

# On ajoute un titre à la fenêtre
fen.title("*** UnderTris ISN ***")
# On débute le jeu
début()
fen.mainloop()
```