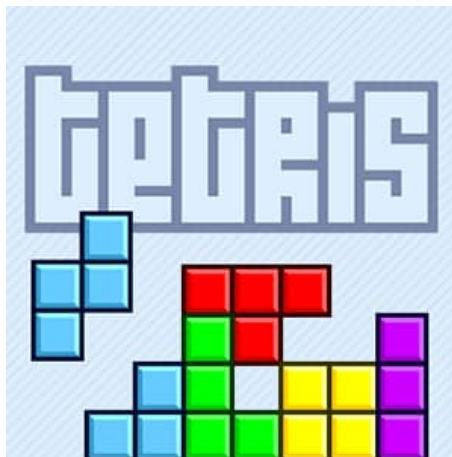


StandardLists=true

UNIVERSITÉ DE CAEN NORMANDIE

RAPPORT DU PROJET DE CONCEPTION LOGICIEL
GROUPE 1A
LICENCE INFORMATIQUE

Développement d'un Tetris multijoueur



Contents

1	Introduction	3
2	OBJECTIFS DU PROJET	3
2.1	But du projet	3
2.2	Présentation du jeu	3
2.2.1	Le plateau	3
2.2.2	Principe du jeu	3
2.2.3	Les pièces	4
2.2.4	Les Couleurs	4
2.2.5	Les mouvements latéraux	4
2.2.6	Les rotations	4
2.2.7	Le fonctionnement du jeu	5
3	FONCTIONNALITÉS IMPLÉMENTÉES & ALGORITHMES	5
3.1	DESCRIPTION DES FONCTIONNALITÉS	5
3.1.1	Phases d'analyse et de conception	5
3.2	Description de la classe GRILLE	6
3.2.1	Ajout de pièces dans la Grille	7
3.2.2	Vérification d'une ligne complète :	7
3.2.3	Suppression des lignes complètes	7
3.2.4	Affichage de la pièce dans la grille	8
3.3	Description de la classe Piece	8
3.3.1	Descente des pièces dans la grille	9
3.3.2	Validité de la descente d'une pièce	9
3.3.3	Rotation d'une pièce	10
3.3.4	Affichage de la pièce suivante	10
3.4	Description de la classe Solo	10
3.4.1	Déplacement des pièces	11
3.4.2	Configuration des touches de jeu	11
3.4.3	Mise à jour des pièces	12
3.5	Description de la classe Multijoueur	13
3.5.1	Mise en place des grilles	14
3.5.2	Mode confrontation	14
3.6	Organisation du travail	14

4	Éléments techniques	15
4.1	Bibliothèque Pygame	15
4.1.1	Importation de la bibliothèque	15
4.1.2	Ouverture d'une fenêtre	15
4.1.3	Chargement et collage d'une image dans une fenêtre	15
4.1.4	Gestion des évènements	16
5	Architecture du projet	17
5.1	Diagrammes des modules et des classes	17
6	Expérimentations et usages	17
6.1	Captures d'écrans	17
7	Conclusion	17
7.1	Récapitulatif des fonctionnalités principales	17
7.2	Propositions d'améliorations	18

1 Introduction

Au cours de ce second semestre de notre première année licence informatique, il était question pour nous de réaliser un projet dans le cadre du cours de Conception de logiciel. Le but de ce projet est de concevoir un jeu de bout en bout en langage python. Sur une liste de huit projets, nous avons choisis de travailler sur le jeu de Tetris que l'on pouvait réaliser avec la bibliothèque Pygame ou Tkinter. Une fois que notre choix a été fait, nous nous sommes documentés sur ce jeu afin de comprendre entièrement ses règles. Ensuite nous avons fait le choix de la bibliothèque avec laquelle nous voulions concevoir le jeu ; afin de pouvoir réaliser notre projet, nous avons fait le choix d'utiliser la bibliothèque **Pygame**. Et pour finir, nous avons établi sur papier un plan de travail avant d'aborder directement le codage en python.

2 OBJECTIFS DU PROJET

2.1 But du projet

Le but de ce projet est de réaliser une version multijoueur du célèbre jeu de Tetris. Dans celle-ci plusieurs joueurs s'affrontent, chacun ayant son propre champ de jeu, et réaliser des lignes. La première étape du projet étant d'implémenter le jeu avec une interface graphique et quelques améliorations classiques comme [niveau de difficulté progressif](#), [visualisation de la pièce suivante](#). Et dans un second temps, il s'agissait d'implémenter un mode multijoueur local contre une IA ou un autre joueur sur écran partagé.

2.2 Présentation du jeu

Le jeu du Tetris est un jeu dans lequel des pièces de différentes formes et couleurs descendent dans le puit qui est considéré comme espace de jeu ou la grille. Ainsi lors de la chute des pièces le joueur ne peut pas ralentir leurs descentes, mais peut les accélérer et peut choisir la position dans laquelle il souhaite placer la pièce dans la grille, ce qui équivaut à effectuer des rotations sur la pièce. Lorsqu'une ligne est complète c'est-à-dire lorsque toutes les cases à l'horizontal de la grille de jeu sont pleines, celles-ci disparaissent et les blocs supérieurs tombent. Si le joueur ne parvient pas à faire disparaître les lignes et que les pièces s'empilent jusqu'en haut de la grille, la partie se termine.

2.2.1 Le plateau

Le plateau de jeu est souvent appelé "puit" ou "matrice". Il s'agit de l'espace dans lequel tombent les pièces. Il dispose toujours d'une grille en arrière-plan, visible ou non, dont les cases sont de la même grandeur que les carrés des pièces, et que celles-ci suivent dans leur chute. Il est également entouré par une armature appelée "tétrion", infranchissable, qui pose les limites du champs de jeu.

2.2.2 Principe du jeu

Des formes géométriques, tombent depuis le haut de la grille. Elles peuvent prendre plusieurs formes pour certaines, et la rotation peut être faite à l'aide d'une touche quelconque du clavier selon la configuration effectuée. Le joueur doit ainsi placer les pièces dans la grille de jeu dans

le but de remplir des lignes. Les pièces tombent une par une, et fusionnent avec le reste des pièces déjà tombées une fois parvenues au bas de la grille. Le principe général du jeu de Tetris consiste à effacer des lignes en remplissant les cases horizontales de la grille de jeu avec les différentes pièces qui tombent lors du jeu.

2.2.3 Les pièces

Les pièces encore appelées les tétriminos, sont au nombre de 7 à savoir :



2.2.4 Les Couleurs

Les couleurs sont choisies aléatoirement de même que les pièces. Les différentes couleurs utilisées dans notre jeu sont le bleu, le vert, l'orange, le jaune, le rouge, le mauve, le bleu clair, le vert clair, l'orange clair, le jaune clair, le rouge clair, le mauve clair.

2.2.5 Les mouvements latéraux

Le joueur à l'aide de certaines touches (selon la configuration) déplace latéralement les pièces, c'est-à-dire vers la droite ou vers la gauche. Cependant quand la pièce touche les bords du jeu, le déplacement n'est plus possible.

2.2.6 Les rotations

Le joueur peut faire tourner plusieurs fois à gauche et/ou à droite, de 90 n'importe quel bloc pour le déposer de la façon désirée pendant que le bloc descend.

2.2.7 Le fonctionnement du jeu

Les pièces générées se déplacent puis se fixent dans notre grille. C'est une grille de 22 x 12 petits carrés. Initialement notre grille est uniquement composée de la chaîne de caractère "." contenue dans la variable *"VIDE"*. La taille de chaque petit carré est de 25. La vitesse de la chute des pièces varient selon le niveau atteint par le joueur. Plus le niveau est élevé, plus la vitesse augmente. Le joueur continue à placer ses pièces qui elles, tombent aléatoirement, jusqu'à ce qu'il réalise (forme) une ligne pleine qui sera effacée et fera évoluer son score. Lorsqu'une pièce est placée, une nouvelle est générée, c'est celle qui est annoncée. Si une pièce touche le sommet de la grille, par empilement, la partie s'arrête.

Travail attendu

Pour pouvoir concevoir le jeu qui a été expliquer ci-dessus, nous avons plusieurs étapes à réaliser dans notre travail, la première est d'implémenter le jeu avec une interface graphique, en ajoutant un niveau de difficulté, progressif. Ensuite nous allons améliorer notre jeu en l'adaptant pour un multijoueur. Nous avons besoin d'un quadrillage que l'on peut considérer comme notre terrain de jeu, qui sera donc dimensionné, afin de pouvoir l'afficher à l'écran. Sur ce quadrillage nous allons appliquer quelques propriétés comme l'ajout d'une manière aléatoire des pièces dans la grille, vérifier les lignes complètes et enfin effacer les lignes complètes. Il faudra tout de même ajouter des améliorations : Visualisation de la pièce suivante, déplacement des pièces, effectuer des rotations sur les pièces si possible. Une fois toutes ses parties terminées, nous allons devoir l'appliquer à un jeu de multijoueur. Celui-ci demande de posséder deux grilles, une pour chaque joueur. Chacun ayant le contrôle sur sa grille de jeu à l'aide des touches du même clavier, configuré différemment. Pour concevoir ce jeu, nous avons choisi Pygame car on s'était plus documenter sur cette bibliothèque.

3 FONCTIONNALITÉS IMPLÉMENTÉES & ALGORITHMES

3.1 DESCRIPTION DES FONCTIONNALITÉS

3.1.1 Phases d'analyse et de conception

La première phase a été de déterminer quelles allaient être les fonctions fondamentales dont nous aurions besoin dans notre programme. Aussi nous avons commencé par écrire les mots clefs du jeu tels que :

- Game over
- Gestion du clavier
- Affichage
- Effacer une ligne complète
- Dessiner une pièce
- Rotation d'une pièce
- Déplacer une pièce

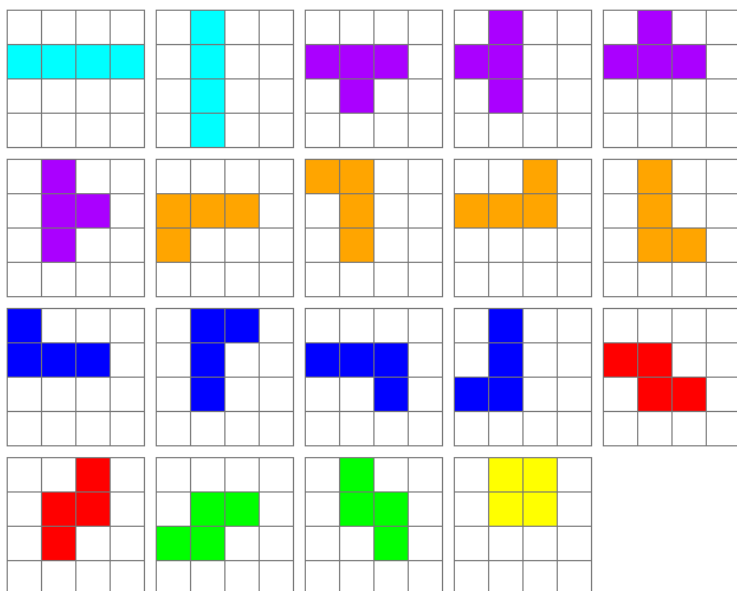


Figure 1: Différentes pièces du Tetris et leur configuration

- Supprimer une ligne
- Générer une ligne
- Score
- Plateau

La seconde consistait à la rédaction du programme principale, c'est-à-dire l'enchaînement des fonctions qui allaient constituer les différentes étapes du programme. Et la troisième étape fut de choisir le codage des données: la grille et les pièces.

Nous remarquons que chaque pièce est constituée de 4 cases et qu'une case repère qui permet de localiser la pièce et de gérer le déplacement de celle-ci.

- La grille définit l'espace où évoluent les pièces. Elle doit représenter la taille de l'espace d'évolution des pièces plus les contours pour pouvoir délimiter cet espace. Ceci nous amène à une grille de taille 12X22.
- L'interface et les touches directionnelles font aussi parti de nos choix : nous avons tout simplement choisi d'utiliser les touches directionnelles et la touche Ctrl pour faciliter l'utilisation du jeu, notamment pour déplacer les pièces dans le mode Solo.

3.2 Description de la classe GRILLE

Généralité : Dans cette classe, on commence par initialiser notre grille de jeu, et à implémenter les fonctions permettant d'ajouter les différentes pièces dans la grille, de vérifier la présence ou non des éléments sur les lignes et colonnes de la grille, d'effacer les lignes complètes c'est-à-dire les lignes où chaque case contient un élément, et aussi de dessiner nos tétrminos dans la grille de jeu.

Initialisation de la grille on multiplie la variable vide par le nombre de lignes en fonction du nombre de colonnes : cette variable vide correspondant aux blancs dans la grille finale c'est-à-dire les lieux où il n'y a pas de pièces.

3.2.1 Ajout de pièces dans la Grille

```
1 def AjoutELEMENT(self , piece):
2     FORME = PIECES[piece.getFORMEIndex()][piece.getPieceSuivante()]
3     L1 = len(FORME)
4     L2 = len(FORME[0])
5
6     for y in range(LONGUEUR):
7         for x in range(LARGEUR):
8             if FORME[y][x] != VIDE:
9                 self.m_grille[y + int(piece.getPosY() / TAILLE_BOITE)][
10                    x + int(piece.getPosX() / TAILLE_BOITE)] = piece.getColor()
```

Ici la variable forme prend en compte deux éléments c'est-à-dire: la pièce suivante et la pièce tombante.

On donne comme valeur aux variables : L1 le premier élément de la liste forme c'est-à-dire l'index des pièces et L2 le nombre d'items que contient la variable forme ensuite on crée une double boucle dans laquelle on vérifie si la grille n'est pas vide elle contient des 0 alors on remplace ces 0 tout en prenant compte la taille de la boîte, la position en x et en y des pièces et on lui attribue une couleur de façon aléatoire tout ceci à l'aide des fonctions utilisées dans la prochaine classe **PIECES** tel que (getposx, getcolorx, getposy) cette fonction ajoute juste les pièces a la grille. Mais la fonction qui dessine les pièces lorsqu'elle ne sont plus en chute et qu'elles sont posées dans la grille se nomme **dessinergrille** ,fonction qui sera expliquée par la suite.

3.2.2 Vérification d'une ligne complète :

Cette fonction est destinée à faire un test ou plutôt a vérifier si la grille contient ou non des éléments dans les différentes lignes et colonnes de la grille. C'est une fonction qui sera utilisée par la suite dans la fonction **effacelignecomplete** afin de pouvoir effacer les lignes totalement pleine de notre grille de jeu.

On signifie ici que la fonction **lignecomplete** est complémentaire à la fonction **effacelignecomplete** dans la mesure où c'est elle qui nous permet de vérifier les lignes complètes afin de pouvoir les effacer

3.2.3 Suppression des lignes complètes

```
1 def effaceLigneComplete(self):
2     y = COLONNES - 1
3     nombreLignecomplete = 0
4
5     while y >= 0:
6         if self.lignecomplete(y):
7             nombreLignecomplete += 1
```

```

8
9         for DansY in range(y, 0, -1):
10             for x in range(LIGNES):
11                 self.m_grille[DansY][x] = self.m_grille[DansY - 1][x]
12             for x in range(LIGNES):
13                 self.m_grille[0][x] = VIDE
14         else :
15             y -= 1
16
17     return nombreLignecomplete

```

Dans cette fonction on fait des tests. On initialise y égal à la longueur de la grille (-1) (donc 21 lignes) et le nombre de ligne complète à 0. On teste si le nombre de colonnes de la grille est supérieur ou égale à 0, puis on observe pour chaque colonne si elle contient ou pas des éléments. Si oui, une ligne complète s'ajoute. On parcourt le nombre de colonnes à l'aide d'une boucle **for** allant de la dernière colonne à la première en décrémentant de -1, puis on parcourt aussi le nombre de lignes.

Pour une ligne complète de la grille, les pièces des lignes en fonction des colonnes sont éliminées et remplacées par la variable VIDE; sinon si une ligne est déjà jouée c'est-à-dire si elle n'est pas entièrement pleine ou complète, on réduit de -1 le nombre de lignes restantes dans la grille de jeu.

Après cela il y aura une fonction **get()** qui retournera uniquement les éléments x et y de la grille. En bref, une fois que la condition de la fonction **lignecomplete()** est vérifiée, on ajoute une ligne au premier rang et on supprime la ligne complète.

3.2.4 Affichage de la pièce dans la grille

C'est la dernière fonction de la classe grille, elle prend en paramètre une surface que l'on a appelé screen. Cette fonction a pour but de dessiner les pièces dans la grille en fonction du nombre des colonnes et des lignes tout d'abord on vérifie si elles sont vides si ce n'est pas le cas alors elle contient des 0 étant donné que les pièces sont constituées de 0 ainsi que de la chaîne de caractère VIDE qui constitue notre grille. Ainsi une fois que les zéros constituant nos pièces sont placés dans la grille de jeu, on les dessine sur une surface à l'aide de la fonction `pygame.draw.rect` qui va pixéliser nos zéros en des petits cubes constituant les parties des pièces.

3.3 Description de la classe Piece

Généralité : Dans cette classe on gère le comportement de la pièce dans la grille de jeu comme sa position lors de sa chute dans la grille, qui a été centrée; on vérifie la possibilité de la descente de la pièce dans la grille, on permet à la pièce de bouger et d'effectuer des rotations au niveau de la grille. La pièce suivante est aussi dessinée dans cette classe.

On initialise les variables qui seront utilisées dans la classe on a :

- forme index qui représente les pièces utilisées dans le jeu on utilise la méthode `random.choice` pour choisir parmi les pièces initialisées en variable globale

- position qui représente ici la position de l’affichage de la pièce suivante dans la fenêtre
- couleur qui représente ici les différentes couleurs de pièces
- pièce_suivante on sélectionne aléatoirement les pièces entre 0 et l’index des des formes dans la listes des FORMES
- et enfin forme longueur_piece et largeur piece;

Quatre fonctions majeures composent la classe : la fonction **bouger**, la fonction **rotation**, la fonction **descenteValide** et la fonction **dessinerpieces**. Les autres fonctions retourne uniquement certaines variables qu’on utilisera souvent dans d’autres classes.

3.3.1 Descente des pièces dans la grille

Cette fonction permet la descente de la pièce dans la grille; ici on vérifie tout d’abord si la descente de la pièce est possible et cela par le biais de la fonction **descenteValide** puis a chaque fois on rajoute x ou y à ses coordonnées du moment où la descente n’est plus valide où que la pièce touche le fond de la grille ou quand elle rencontre une autre pièce on retourne **False**. Lors de la conception de notre jeu nous n’avons pas pensé à cette fonction lors de la phase des test nous nous sommes aperçus que les pièces restait tout en haut de la grille mais ne descendait pas, c’est par l’aide de madame Françoise Lambert, qui nous a suggéré de créer une fonction **bouger**, qu’on a pu faire bouger nos pièces. Il est aussi important de noter que cette fonction n’est active que si et seulement si la descente des pièces dans la grille est possible.

```

1 def bouger(self, x, y, GRILLE):
2     if self.descenteValide(x, y, GRILLE):
3         self.m_position['x'] += x
4         self.m_position['y'] += y
5         return True
6
7     else:
8         return False

```

3.3.2 Validité de la descente d’une pièce

Ici tout d’abord à l’aide, on vérifie la possibilité qu’une pièce ou que les futures pièces puisse bouger dans la grille. Dans cette double boucle parcourant les lignes et les colonnes de notre grille, on vérifie que notre pièce, essentiellement formée de la chaîne de caractère *VIDE* et de zéros, rencontre cette même chaîne dans la grille de jeu afin de continuer jusqu’à la prochaine itération. Une fois que notre pièce croise la chaîne de caractère *VIDE* dans la grille, à ses coordonnées x et y sont ajoutées des valeurs qui effectuerons pour ne pas dire qui mettrons à jour le mouvement de descente de la pièce au milieu de notre grille.

```

1 for y in range(self.LONGUEURPIECE):
2     for x in range(self.m_LARGEURPIECE):
3         if self.m_FORME[y][x] == VIDE:
4             continue
5
6         xPos = x + int((self.m_position['x'] + ajoutX) / TAILLE_BOITE)
7         yPos = y + int((self.m_position['y'] + ajoutY) / TAILLE_BOITE)

```

Après cette première itération on passe à la suivante qui est de voir, de vérifier si les coordonnées x de la pièce sont supérieures à zéro, ou supérieures/égales à celles des lignes ou encore si les coordonnées y de la pièce sont supérieures/égales à celles des colonnes; si tel est le cas on retourne **False**. De même on test aussi si les lignes et les colonnes ne contiennent pas la chaîne de caractère *VIDE* où plutôt si les parties de la grille sont occupées par d'autres pièces, on retourne aussi **False**. Et lorsque notre test de boucle vérifie toutes ces conditions on retourne **True**.

```

1 if xPos < 0 or xPos >= LIGNES or yPos >= COLONNES:
2     return False
3     if GRILLE.get(xPos, yPos) != VIDE:
4         return False

```

Ce test a été réalisé afin de s'assurer que notre pièce ne sort pas des limites de notre grille de jeu.

3.3.3 Rotation d'une pièce

Pour la rotation des différentes pièces, on vérifie tout d'abord si la place qu'occupe la pièce en fonction de sa rotation ne dépasse pas la largeur de la grille, et si ce n'est pas le cas on la pièce reste comme elle est donc on réaffecte aux variables les mêmes valeurs de façon à ce qu'elle restent dans la même position.

3.3.4 Affichage de la pièce suivante

Tout simplement cette fonction a pour but de dessiner les pièces en pleine chute dans la grille à l'aide de la méthode `pygame.draw.rect`. Elle est complémentaire à la fonction `dessinergrille()` car elle permet de dessiner les pièces posées dans la grille.

3.4 Description de la classe Solo

Généralité : Dans cette classe se trouve les fonctions essentielles pour le lancement de notre jeu; en effet c'est dans cette classe que la fenêtre de jeu est initialisée, que les mouvements des pièces à gauche et à droite dans la grille sont assurés, que les fonctions d'affichage de texte, que les touches de jeu ont été configurées, les niveaux ont été implémentés.

Dans cette classe on se sert des autres classes et de leurs attributs afin de faire tourner le jeu. Ainsi les fondamentaux de cette classe sont: les *mouvements des pièces*, la *fréquence de mise à jour* de la grille et bien sûr des *événements associés aux pièces* dans la grille. Dans cette classe, nous travaillons aussi sur l'aspect graphique, en dessinant tous les éléments du jeu aux endroits destinés. Elle ne comprend que 3 principales fonctions : `entreeDonnees` ; `logic` et `MiseaJour`; les autres fonctions telles que `dessinelements`, `texte` et `afficheScore` concernent uniquement la partie graphique puis d'autres fonctions concernant les niveaux et changements d'ambiance au cours du jeu.

3.4.1 Déplacement des pièces

C'est la dans fonction **logie** , qu'on a défini les différentes directions par rapport à la boîte par exemple si le joueur appuie la touche dédié a la direction gauche la pièce se déplace vers la gauche dans la grille et ça en est de même chose lorsque les touches dédiées aux directions haut et bas sont pressées. Pour faire cela nous avons initialisé un dictionnaire direction en début de la classe où nous donnons comme clé les directions et comme valeur, la valeur booléenne false. `Self.m_direction = 'left' : False, "right" : False, "down" : False` ensuite on régule le temps entre l'exécution des instructions de mouvement des pièces et le déplacement des pièces .

```
1 if time.time() - self.m_DernierTempsMOuv > 0.1:
2     if self.m_direction['left'] == True:
3         self.m_PieceCourante.bouger(-TAILLE_BOITE, 0, self.m_GRILLE)
```

Ici on vérifie d'abord si le temps entre ces deux événements est supérieur à 0,1 ce qui aura pour effet le fait qu'il y ait à chaque fois un temps de latence entre deux instructions, sinon lorsque le joueur appuiera sur une touche le mouvement de la pièce sera continu ce qui ne favorise pas les bonnes conditions de jeu . Puis on affecte en fonction des directions, la direction de la pièce en fonction dans la grille.

```
1 elif self.m_direction['right'] == True:
2     self.m_PieceCourante.bouger(TAILLE_BOITE, 0, self.m_GRILLE)
```

La clé *right* était au départ initialisée à False, et quand le joueur appuie sur la touche correspondant à la direction droite la pièce présente dans la grille (self, m_PieceCourante) se déplace vers la droite de la grille a l'aide de la fonction **bouger** présente dans la classe pièce qui permet aux pièces de se déplacer dans la grille

```
1 if self.descenteValide(x, y, GRILLE):
2     self.m_position['x'] += x
3     self.m_position['y'] += y
4     return True
5
6 else:
7     return False
```

Les deux arguments de la fonctions *x* et *y* de la fonction **bouger** correspondent aux coordonnées ajoutées en x et en y . Sachant que la variable *taille_boite* correspond ici a la taille de chaque petit carreau dans la grille. Ainsi lorsque la pièce va vers la droite on ajoute à la position initiale de la pièce +25 donc logiquement la pièce se dirige vers la droite on fera l'inverse pour la direction gauche, et pour la direction bas on rajoute en *y* +25 du coup la pièce va descendre d'un cran ainsi pour les directions gauche et droite on ne rajoute rien en *y*, on met donc 0.

```
1 self.m_PieceCourante.bouger(-TAILLE_BOITE, 0, self.m_GRILLE)
```

3.4.2 Configuration des touches de jeu

C'est dans la fonction **entreeDonnees** que nous avons attribué les touches qui seront utilisées dans le jeu pour certains événements. D'abord on lance une boucle **for**, qui parcourt tous les événements reçus grâce à la fonction **get()** du module *"event"* de la bibliothèque Pygame. Cette fonction retourne une liste d'objets Event, pour lesquels on peut connaître le type, la touche enfoncée si c'est au clavier, la position du curseur si c'est un clic. On teste si l'évènement est de type *QUIT*; et si la condition est satisfaite, on demande à la boucle de s'arrêter.

```

1 for event in pygame.event.get():
2     if event.type == pygame.QUIT:
3         self.m_gameOver = True

```

Puis on vérifie si le type d'évènement crée correspond à l'appuie sur une touche :

```

1 if event.type == pygame.KEYDOWN:

```

Ensuite on attribue des touches à chaque direction en fonction des clés données dans le dictionnaire des directions (self.m_direction)

```

1 if event.key == pygame.K_LEFT:
2     self.m_direction['left'] = True
3 elif event.key == pygame.K_RIGHT:
4     self.m_direction['right'] = True
5 elif event.key == pygame.K_DOWN:
6     self.m_direction['down'] = True
7 elif event.key == pygame.K_RCTRL:
8     self.m_rotate = True
9 elif event.key == pygame.K_ESCAPE:
10    self.m_gameOver = True

```

Après cela on vérifie le relâchement des touches c'est-à-dire si les touches appuyées sont relâchées la pièce continue à chuter sans aller à gauche où à droite mais continue de tomber a vitesse normale pour cela on redonne comme valeur au clé du dictionnaire correspondant aux directions gauche droite ou basse la valeur booléenne fausse :

```

1 if event.key == pygame.K_LEFT:
2     self.m_direction['left'] = True
3 elif event.key == pygame.K_RIGHT:
4     self.m_direction['right'] = True
5 elif event.key == pygame.K_DOWN:
6     self.m_direction['down'] = True
7 elif event.key == pygame.K_RCTRL:
8     self.m_rotate = True
9 elif event.key == pygame.K_ESCAPE:
10    self.m_gameOver = True

```

En résumé lorsque le joueur appuie sur la touche « direction gauche » du clavier par exemple, la clé correspondant à cette direction est égale à **True** et ainsi la pièce va se déplacer d'un cran vers la gauche il en est ainsi des autres cas.

3.4.3 Mise à jour des pièces

Cette fonction à pour but de faire tourner le jeu cette fonction est la plaque tournante du jeu entier.

Pour débiter nous avons initialisé la variable *self.m_dernierrechutetime* cette variable correspond au dernier temps de chute; on régle l'exécution des instructions de façon a ce qu'elle soit supérieure à 1.

```

1 if time.time() - self.m_DernierrechuteTime > 1:

```

Puis on définit la variable `self.m_piecechutes` en disant qu'au moment où la pièce est en chute, elle descend de façon continue sans que l'on ait besoin d'appuyer une touche.

```
1 if self.m_piecechutes:
2     if self.m_PieceCourante.bouger(0, TAILLE_BOITE, self.m_GRILLE):
3         pass
```

Dans le cas contraire on donne comme valeur à cette variable la valeur booléenne `False`, qui signifie qu'aucune pièce n'est en chute alors on ajoute à la grille une nouvelle pièce ensuite on ajoute 25 au score pour chaque ligne effacée en multipliant chaque ligne complète par 25. En fin de cette condition on réinitialise `self.m_Dernierrechutetime` à `time.time()` on fait ceci car si on ne le fait pas, les pièces tomberont sans cesse rendant impossible le jeu; cette réinitialisation assure donc une régulation de la chute et la vitesse de chute des pièces. Ensuite si aucune pièce n'est en chute la pièce courante est égale à la pièce suivante et elle-même la pièce suivante est égale à la future pièce courante et ainsi on la fait descendre tout au milieu de la grille. Dans ce cas l'état de chutes de la pièce devient vrai donc on réaffecte à la variable `self.m_piecechutes` la valeur booléenne `True` sinon la chute des pièces sera incontrôlée. Enfin si la descente n'est plus possible dans la grille le jeu se termine donc on attribue ainsi à la variable `gameover` la valeur `True`. En résumé la fonction mise à jour régule le passage de la pièce suivante à la grille et l'intervalle de mise à jour des éléments dans la grille, dans le cas où l'on réduit le temps de mise à jour les itérations s'exécutent plus rapidement, quant au joueur il aura constaté que la chute des pièces s'accélère car le temps entre chaque itération a baissé.

Fonctions supplémentaires

Niveaux

Dans cette fonction nous modifions juste l'ambiance et nous rajoutons à la liste de pièces des nouvelles formes assez difficile à jouer et le niveau change en fonction du score de l'utilisateur.

Changement d'ambiance

Dans cette fonction nous réutilisons juste la fonction mise à jour en diminuant la fréquence des mise à jour ce qui va accélérer la chute des pièces dans la grille selon le niveau atteint.

Concernant l'affichage et l'aspect graphique du jeu nous avons tout d'abord à l'aide des fonctions écrites antérieurement dans les classes grille et pièce, dessiné les pièces sur le plateau notamment grâce à `dessinerguille` qui dessine les pièces dans la grille créée sur les plateaux puis avec la fonction `dessinerpiece` on dessine pièce suivante. Ensuite on a dessiné les contours de la grille avec la méthode `pygame.draw.line` de pygame et ensuite on blit la grille sur la fenêtre.

3.5 Description de la classe Multijoueur

Tout comme la classe Solo la classe multijoueur est aussi une classe majeure de notre jeu, car comme son nom l'indique c'est dans cette classe qu'a été créée deux grilles indépendantes et donc c'est dans cette classe que nous avons mis en place notre Tetris multijoueur.

3.5.1 Mise en place des grilles

Tout d'abord nous avons initialisé dans la classe multijoueur la grille en lui donnant tous les attributs de la classe grille `self.m_grille2 = GRILLE()`. Puis on crée deux plateaux ayant comme dimensions, celle de la grille c'est sur ces plateaux que seront dessinées les deux grilles. Ensuite nous avons dupliqué tous les éléments utiles et permettant l'indépendance des deux grilles telle que les *pièces courantes*, les *pièces suivantes*, *l'état des pièces*, *le game_over*, *le sens de déplacement*, les *touches du clavier*, les *rotations de pièces* et enfin les deux différents plateaux de jeu car c'est ceci qui nous permet de distinguer bien les deux grilles.

Puis après nous avons dupliqué la classe pièce avant de modifier la variable position qui modifiera la position des éléments dans la boîte telle que l'affichage de la pièce suivante de la seconde grille. Après ceci nous avons modifié les touches directions pour chaque grille en dupliquant la fonction `logic`; après cela il ne nous restait qu'à afficher les deux grilles sur les plateaux de jeu initialisés en début de classe.

Dans la fonction dédiée à l'affichage et à l'aspect graphique du jeu nous avons tout d'abord à l'aide des fonctions écrites antérieurement dans les classes grille et pièce, dessiné les pièces sur les plateaux notamment grâce à `dessinerguille` qui dessine les pièces dans la grille créé sur les plateaux, puis avec `dessinerpiece` on dessine la pièce suivante. Comme dans la classe Solo, les contours de la grille ont été réalisés avec la méthode `pygame.draw.line` de pygame et ensuite on a blité la grille sur les deux plateaux à des coordonnées différentes; après ceci on constate que les deux grilles sont bien présentes dans la fenêtre de façon bien distingué il restera ensuite à mettre un peu d'action dans le jeu.

3.5.2 Mode confrontation

Notre mode multijoueur test le plus endurant des deux joueurs. En cassant des lignes des formes de pièces difficile à placer sont ajoutées au dictionnaire de pièce du joueurs adverse et ainsi de suite le joueur ayant cassé le plus de lignes c'est-à-dire ayant le plus grand score remporte la partie. Lorsque que l'un des deux joueurs perd avant son adversaire c'est-à-dire que sa grille est pleine il est impossible pour lui de continuer ainsi dans sa grille il n'y a plus d'évènement (chute de pièces) et l'adversaire peut tranquillement terminer sa partie ainsi le joueur ayant le plus grand score et ayant le plus duré dans la partie remporte celle-ci.

3.6 Organisation du travail

Concernant l'organisation du travail, nous nous ne nous sommes pas vraiment réparti les tâches pour concevoir le Tetris. Nous avons réalisé en grande partie ce projet en groupe, lors de nos séances de travaux pratiques mais aussi en dehors de ces séances afin de permettre à chacun d'entre nous de mieux comprendre le code et le fonctionnement du jeu.

4 Éléments techniques

4.1 Bibliothèque Pygame



Pygame est une bibliothèque Python permettant la réalisation simple de jeux interactif. Cette bibliothèque est principalement basée sur la *SDL (Simple Directmedia Library)* qui est une bibliothèque libre multi-plateforme permettant la gestion du multimédia dans la programmation. En résumé, cette bibliothèque sert principalement pour:

- L'affichage vidéo 2D
- La gestion de l'audio
- La gestion de périphériques de commandes tel que le clavier, la souris

Pygame est donc l'adaptation de la SDL au service de Python, mais est aussi constitué de quelques ajouts et modifications de son auteur. Cette bibliothèque est assez intuitive et constitue un très bon moyen de se lancer dans la programmation graphique avec Python.

4.1.1 Importation de la bibliothèque

Elle assez simple à installer et importer (`import pygame`); avant toute chose, cette bibliothèque nécessite d'être initialisée l'instruction `pygame.init()`.

4.1.2 Ouverture d'une fenêtre

L'ouverture d'une fenêtre se fait avec la fonction `set_mode()` contenue dans le module "display" de Pygame. Cette fonction prend en paramètre un tuple contenant la largeur et la hauteur de la fenêtre voulue.

```
pygame.display.set_mode((Width, Height))
```

4.1.3 Chargement et collage d'une image dans une fenêtre

Pour pré-charger des images dans notre fenêtre, on utilise la fonction `load()` du module "image"

```
pygame.image.load("nom_image").convert()
```

C'est avec la fonction `blit()` qu'on effectue le collage dans une fenêtre. Elle prend en argument l'image pré-chargée que l'on souhaite coller, et les positions sous forme de tuple de l'endroit où on souhaite afficher notre image. Les coordonnées sont celles de l'angle en haut à gauche de l'image.

```
Surface.blit(fond, (0,0))
```

À noter que l’affichage de l’image collée n’est fait qu’après avoir actualisé, rafraîchi la surface à l’aide de la fonction *flip* ou *update* du module "display" de pygame.

4.1.4 Gestion des évènements

Les touches du clavier Pygame est une librairie qui fonctionne avec des évènements. Un évènement est simplement une action que l’utilisateur effectuera, par exemple cela peut être « Appuyez sur la touche V » ou encore « Bouger la souris ». Pour cela, Pygame enregistre dans une liste tous les nouveaux évènements lorsqu’il y en a un et de manière asynchrone. Ces évènements sont alors enregistrés et récupérés avec la fonction *get()* du module "event" de pygame.

```
pygame.event.get()
```

La détection de l’appuie sur une touche se fait avec le mot clé **KEYDOWN** et la détection sur le relâchement d’une touche se fait **KEYUP**.

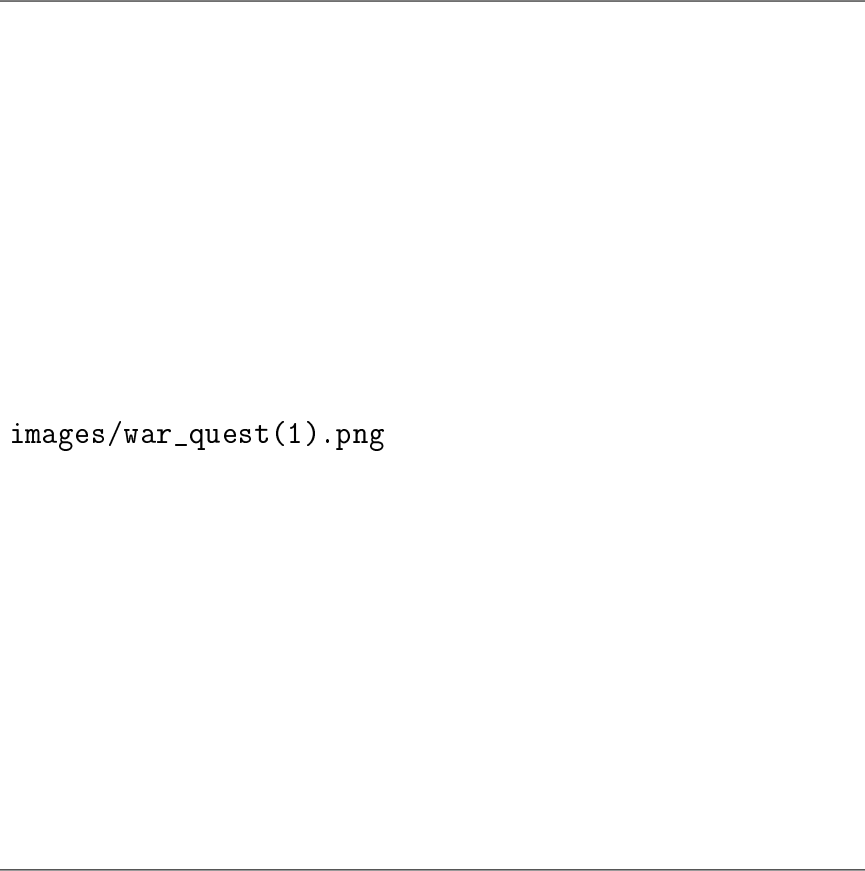
Les évènements liés à la souris Le type d’évènement crée lors d’un clic est **MOUSEBUTTONDOWN**, (ou **MOUSEBUTTONUP** au relâchement du bouton). Un évènement de souris possède deux attributs : le bouton (*event.button*) et la position du clic dans la fenêtre (*event.pos*). *event.button* peut prendre les valeurs suivantes :

1. Bouton gauche
2. Bouton milieu (ou droite + gauche)
3. Bouton droit
4. Molette haut
5. Molette bas

event.pos renvoie un tuple contenant l’abscisse et l’ordonnée à partir de l’angle haut-gauche, c’est-à-dire le bout de la pointe de la flèche.

5 Architecture du projet

5.1 Diagrammes des modules et des classes



images/war_quest(1).png

À travers ce graphique on peut donc voir les relations entre nos classes et leurs fonctions. En effet, les classes **PIECE** et **GRILLE** sont toutes reliées à la classe **GRILLE** car c'est dans la grille que nos pièces sont dessinées et que certaines actions (déplacement, rotation, accélération de chute de pièces) y sont effectuées. D'où une interdépendance de certaines fonctions des classes pour le bon fonctionnement du jeu.

6 Expérimentations et usages

6.1 Captures d'écrans

7 Conclusion

7.1 Récapitulatif des fonctionnalités principales

Au terme de ce projet, nous pouvons rappeler que les classes **Grille**, **Pieces** et **Solo** ont regroupé les principales fonctionnalités de notre jeu. Dans la classe Grille a été initialisée la grille de jeu, les pièces ont été ajoutées et dessinées, les fonctions de test de lignes complètes et d'effacement de ligne ont été implémentées.

Dans la classe Pieces les pièces suivantes ont été dessinées, déplacées (via la fonction **bouger**), que la fonction de rotation a été implémenté et que le test sur la possibilité de descente d'une

pièce a été effectuée. Et dans la classe Solo, les touches de jeu ont été configuré et que l'interface a été géré.

Toutefois comme dans tous les projets certaines difficultés ont été rencontré lors de la conception du jeu notamment pour faire un multijoueur local. Mais à force de persévérer à pouvoir le faire nous avons fini par réussir.

7.2 Propositions d'améliorations

Pour rendre notre jeu un peu plus performant, certaines améliorations peuvent être apportée ou proposée comme :

- ▶ La mise en place d'une pièce fantôme au niveau du fond de la grille c'est-à-dire une pièce qui reflétera au fond de la grille la position dans laquelle notre pièce va se poser;
- ▶ La mise en place d'un système de stockage de pièce, qui permettra au joueur de pouvoir changer de pièce courante quand ça ne lui arrange pas;
- ▶ Implémenter une intelligence artificielle (IA) locale et en réseau.