Info-F-106 : Projet d'Informatique
Projet 1 – Tétraminos

Anthony Cnudde Gwenaël Joret Tom Lenaerts Robin Petit
Loan Sens Cédric Simar

version du 22 novembre 2023

Présentation générale

Le projet en trois phrases

L'objectif du premier projet est de réaliser une implémentation en Python 3 de *Tétramino*. C'est un jeu à 1 joueur sur une grille donnée. Le but du jeu est de placer toutes les pièces données, des tétraminos, dans la grille sans qu'elles se chevauchent.

Tétramino

La version du jeu que nous vous proposons est inspirée du jeu 2 joueurs *Gagne ta maman*[1] / *Gagne ton papa*[2] adapté pour 1 joueur (NB : ne vous laissez pas distraire par le nom maladroitement choisi de ce jeu, c'est un jeu très bien pensé) :

Chaque pièce colorée est un *tétramino*, composé de plusieurs blocs unitaires. Dans la version que nous vous proposons, une partie consiste à résoudre un problème décrit sur une *carte* donnée : celle-ci spécifie les tétraminos à placer, ainsi que la taille de la grille. Il faut arriver à placer chaque tétramino sur la grille sans qu'ils se chevauchent. Plus la grille et le nombre de pièces sont grands, plus la partie est difficile.

1. https://www.gigamic.com/jeux-pedagogiques/189-gagne-ta-maman-3421271314615.html
2. https://www.gigamic.com/jeux-pedagogiques/27-gagne-ton-papa-3421271314318.html

Le projet est décrit en long et en large plus loin dans ce document, mais avant d'entrer dans le coeur du sujet, voici d'abord quelques consignes générales pour la réalisation de ce projet. Elles seront également d'application pour le second projet au second semestre.

Consignes générales

- L'ensemble du projet est à réaliser en `Python 3`.
- Le module INFO-F106 est constitué de deux projets de programmation distincts, un par quadrimestre.
- En plus de ces projets à remettre, chaque étudiant(e) devra remettre un rapport final au second quadrimestre (Q2).
- La répartition des points est la suivante :
  - Projet 1 (Q1, Tétramino) : 40 points
  - Projet 2 (Q2) : 40 points
  - Rapport (Q2) : 20 points
  - **Total : 100 points**
- Les deux projets et le rapport devront être remis sur l'UV.
- **Il n'y aura pas de seconde session pour ces projets !**

Veuillez noter également que le projet vaudra **zéro** sans exception si :

- le projet ne peut être exécuté correctement via les commandes décrites dans l'énoncé ;
- les noms de fonctions (et de vos scripts) sont différents de ceux décrits dans cet énoncé, ou ont des paramètres différents ;
- à l'aide d'outils automatiques spécialisés, nous avons détecté un plagiat manifeste (entre les projets de plusieurs étudiant(e)s, ou avec des éléments trouvés sur Internet). Nous insistons sur ce dernier point car l'expérience montre que chaque année une poignée d'étudiant(e)s pensent qu'un petit copier-coller d'une fonction, suivi d'une réorganisation du code et de quelques renommages de variables passera inaperçu... Ceci sera sanctionné d'une note nulle pour l'entièreté du projet pour toutes les personnes impliquées, ainsi que d'éventuelles autres sanctions. Afin d'éviter cette situation, veillez en particulier à ne pas partager de bouts de codes sur des forums, Facebook, Discord, etc.

Soumission de fichiers

La soumission des fichiers se fait via la section ouverte sur l'UV.

Tests automatiques

Pour évaluer votre code, nous mettrons à votre disposition un fichier de test. Ce fichier contient des tests qui valideront (ou non) les fonctions que vous avez implémentées. Ces tests vous permettront de savoir si votre code réagit comme nous l'attendons ou si vous devez le retravailler. Cependant, cela implique que vous devrez respecter à la lettre les consignes qui vous sont données, sans quoi ces tests échoueront.

Nous exigerons de vous que votre code passe l'ensemble des tests fournis, sans quoi nous ne corrigerons pas votre projet, et votre note finale sera de 0.

Communication avec l'équipe des enseignants

Remarque préliminaire : Il y a 566 étudiant(e)s inscrits à ce cours (chiffres du 13 novembre 2023), il nous est donc simplement impossible de répondre individuellement aux questions par email. Ceci nous amène à la règle suivante :

Règle d'or : ne jamais envoyer d'email !

Des séances de questions / réponses seront organisées régulièrement, vous aurez donc de nombreuses occasions pour poser vos questions. De cette façon, tout le monde en profite ! Et cela nous évite de répondre 25x à la même question.

Afin d'insister sur l'importance de cette règle, voici quelques exemples de situations dans lesquelles un(e) étudiant(e) souhaiterait contacter un assistant ou le titulaire. Dans chacune de ces situations, écrire un email (ou message individuel Teams) est proscrit :

— « j'ai une question concernant l'énoncé du projet » → question à poser lors des séances Q/R organisées par les assistants ;
— « je ne comprends pas la note reçue pour mon projet, j'aimerais pouvoir en discuter » → allez à la séance de visite des copies organisée par les assistants en charge ;
— « j'ai eu zéro à mon projet pour non-respect des consignes à cause d'une faute de frappe dans le nom de mon fichier, je trouve cela trop sévère » → nous appliquons les consignes annoncées dans l'énoncé à la lettre, sans aucune exception. Apprendre à respecter les consignes est un des objectifs pédagogiques de ce projet d'année.

Notez que nous ne répondrons simplement pas aux emails concernant le projet d'année si jamais vous nous en envoyez. La seule exception à cette regle concerne les emails d'ordre administratif adressés aux titulaires (certificat maladie, réorientation, EBS, etc.)

Objectifs pédagogiques

Ce projet *transdisciplinaire* permettra de solliciter diverses compétences.

— *Des connaissances vues aux cours de programmation, langages, algorithmique ou mathématiques.* L'ampleur des deux projets requerra une analyse plus stricte et poussée que celle nécessaire à l'écriture d'un projet d'une page, ainsi qu'une utilisation rigoureuse des différents concepts vus aux cours.
— *Des connaissances non vues aux cours.* Les étudiant(e)s seront invité(e)s à les étudier par eux-mêmes, aiguillé(e)s par les *tuyaux* fournis par l'équipe encadrant le cours.
— *Des compétences de communication.* Les étudiant(e)s devront rédiger un rapport scientifique en LaTeX. Ce sera l'occasion pour les étudiant(e)s de se familiariser avec ce langage, utilisé pour la rédaction de documents scientifiques. Une orthographe correcte sera exigée.

En résumé, vous devrez démontrer que vous êtes capables d'appliquer des concepts vus aux cours, de découvrir par vous-même des nouvelles matières, et enfin de communiquer de façon scientifique le résultat de votre travail.

Bon travail !

1 Projet 1 : Tétramino

L'objectif du projet est de réaliser l'implémentation en Python 3 de Tétramino, un jeu un joueur demandant à celui-ci d'aligner des pièces données de manière à remplir une grille de taille déterminée.

1.1 Le jeu

Le jeu démarre sur une grille de taille $w \times h$. Le joueur reçoit un ensemble de pièces de forme prédeterminée. Ces pièces peuvent être tournées dans le sens horaire ou anti-horaire, mais ne peuvent pas être retournées. Le joueur doit placer ces pièces sur la grille de manière à ce que chaque espace de cette grille soit rempli. Concrètement, le joueur choisit une pièce et peut la déplacer librement, même par dessus une pièce déjà posée ou une bordure. Il ne pourra cependant déposer la pièce que dans une zone libre, sans bordure ni autre pièce. Une fois la pièce posée, il pourra choisir une autre pièce (ou la même), jusqu'à ce qu'il trouve une configuration gagnante.

Pour cette implémentation, la partie aura lieu sur une grille de taille $(3w + 2) \times (3h + 2)$ afin de permettre un déplacement plus aisé des pièces. La zone de jeu à remplir sera alors la partie centrale, et chaque pièce sera au départ positionnée dans une des grilles périphériques. Les *screenshots* ci-dessous (Figure 1) représentent l'aire de jeu initiale ainsi que la configuration à atteindre.

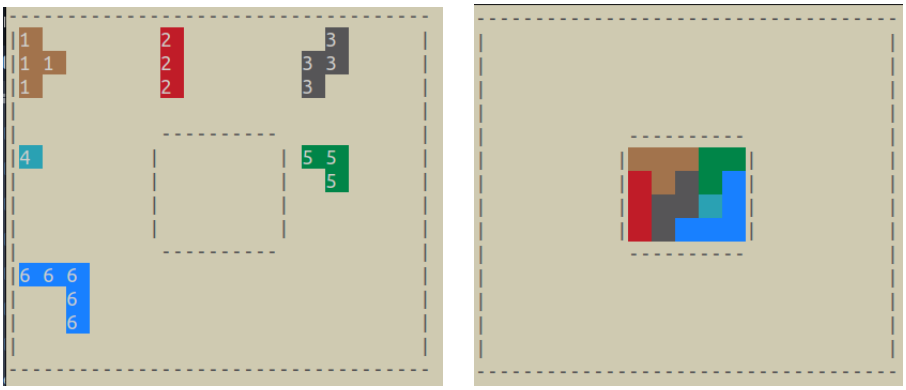

FIGURE 1 – A gauche, plateau de jeu au lancement de la partie. A droite, plateau de jeu en fin de partie

Le programme sera lancé en utilisant la commande suivante :

```
python3 tetramino.py carte
```

où `carte` est le nom du fichier texte correspondant à une des *cartes* qui vous seront fournies. Chaque carte encode un problème à résoudre pour notre joueur : la taille du plateau et les différentes pièces proposées pour la partie.

Afin d'utiliser des paramètres entrés en ligne de commande dans votre code, vous pouvez utiliser la liste `sys.argv` du module `sys`. Par-exemple, si nous lançons le programme via la commande

```
python3 tetramino.py carte_1.txt
```

alors `sys.argv` sera la liste de chaînes de caractères suivantes : `['tetramino.py', 'carte_1.txt']`. Le premier élément de la liste correspond au nom du script python, et le second contient le nom du fichier texte correspondant à la carte.

Veillez à ce que votre fichier puisse être importé, mettez donc l'appel à la fonction `main` dans un test conditionnel `if __name__ == '__main__':`.

1.2 Structure du programme

Le jeu est paramétré par un fichier "carte" que vous devrez lire et traiter au démarrage du programme. Ce fichier contient :

— en première ligne, les dimensions de la grille de jeu ;
— pour chaque ligne suivante, les positions et la couleur d'une des pièces à placer dans la grille.

1.2.1 Lecture du fichier de jeu

Chaque fichier qui vous est donné ressemblera à ceci :

```
5, 4
(0, 0);(0, 1);(0, 2);(1, 1);;0;37;43
(0, 0);(0, 1);(0, 2);;0;37;41
(1, 0);(1, 1);(0, 1);(0, 2);;0;37;45
(0, 0);;0;37;46
(0, 0);(1, 0);(1, 1);;0;37;42
(0, 0);(1, 0);(2, 0);(2, 1);(2, 2);;0;37;44
```

— La première ligne correspond aux dimensions du plateau de jeu ; le premier nombre correspond au nombre de colonnes, le second au nombre de lignes. N'oubliez pas que la taille réelle du plateau dans cette version doit être triplée et comprendre les bordures centrales (dans la cas ci-dessus, le plateau de jeu ferait donc 17 colonnes et 14 lignes).
— Les lignes suivantes décrivent les pièces à placer, une pièce par ligne décrite comme suit : d'abord les coordonnées des blocs unitaires constituant la pièce, et ensuite la couleur de la pièce. Plus précisément :
  — Les coordonnées correspondent à la position en (x, y) de chaque bloc unitaire constituant la pièce. Par exemple, un tétramino carré constitué de 4 blocs et décrit comme suit :

    ```
    (0, 0);(0, 1);(1, 0);(1, 1)
    ```

  — Le code couleur, séparé des coordonnées des blocs par deux point-virgules, donne une couleur lorsqu'il est intégré à une chaîne de caractères spécifique dans le terminal. Cette chaîne de caractères est la suivante :

    ```
    '\x1b[' + code_couleur + 'm' + texte + '\x1b[0m'
    ```

    où `code_couleur` est une chaîne de caractères contenant le code couleur, et `texte` est une chaîne de caractères contenant le texte qui doit être affiché. Par-exemple, un `print` de la chaîne de caractères suivante

    ```
    '\x1b[0;37;44mTetramino\x1b[0m'
    ```

    affichera :

`Tetramino`

Remarque : pour afficher une case de couleur pleine sans texte il suffit de mettre un espace (c-à-d : `' '`) comme texte.

A vous de transformer le contenu de ces fichiers en données interprétables.

Veuillez noter que les coordonnées correspondent à celles d'un repère orthonormé **avec l'axe des ordonnées inversé**, c'est-à-dire que la position $(0, 0)$ correspond au coin supérieur gauche de la matrice, et la position $(w-1, h-1)$ le coin inférieur droit, où w et h sont les dimensions de la matrice,

1.2.2 Plateau de jeu

Comme indiqué, le plateau de jeu doit être de dimension $(3w+2) \times (3h+2)$ afin de permettre de sélectionner et déplacer les pièces plus facilement.

- Les zones vides du plateau seront représentées par deux espaces (c-à-d `' '*2`), ce qui permet un affichage plus harmonieux des formes.
- Vous devrez également penser à placer les délimitations de la zone de jeu centrale, qui devra être totalement remplie pour gagner la partie. Pour faciliter la gestion du plateau, ces délimitations feront également partie de l'encodage du plateau, et seront représentées comme suit :
  - par la chaîne de caractères `' |'` (c-à-d `|` précédé d'un espace) pour les délimitations verticales du côté gauche, la chaîne de caractères `'| '` (c-à-d `|` suivi d'un espace) du côté droit ;
  - par la chaîne de caractères `'--'` pour les délimitations horizontales.

Ainsi, l'encodage du plateau final, délimitations de la zone centrale comprises, sera de dimension $(3w+2) \times (3h+2)$. Un exemple est donné en Figure 2.

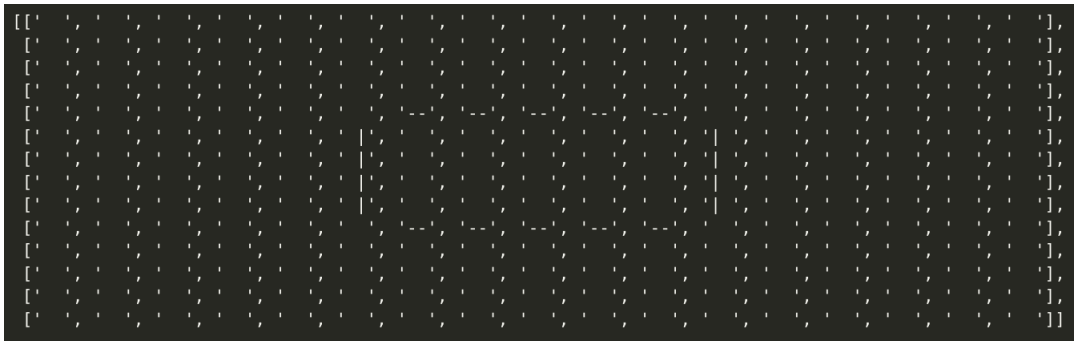

FIGURE 2 – Plateau de jeu 5×4 tel qu'il doit être représenté dans votre code

Pour la représentation de votre plateau lors d'une partie dans le terminal, nous vous demandons d'appliquer les consignes suivantes :

- Les pièces sont affichées dans la couleur indiquée dans le fichier importé
- Lors de la sélection de la pièce **uniquement** (pas lorsque le joueur a choisi sa pièce et la déplace), le numéro de la pièce est indiqué dans la partie gauche de chaque bloc unitaire constituant la pièce.

— Lorsqu'une pièce est positionnée sur une zone non-valide (au dessus d'une autre pièce ou d'une bordure), elle est affiché avec le texte `'XX'` sur le(s) blocs problématiques pour indiquer que poser la pièce n'est pas possible à cause de ce(s) bloc(s). Une illustration vous est donnée en Figure 3.

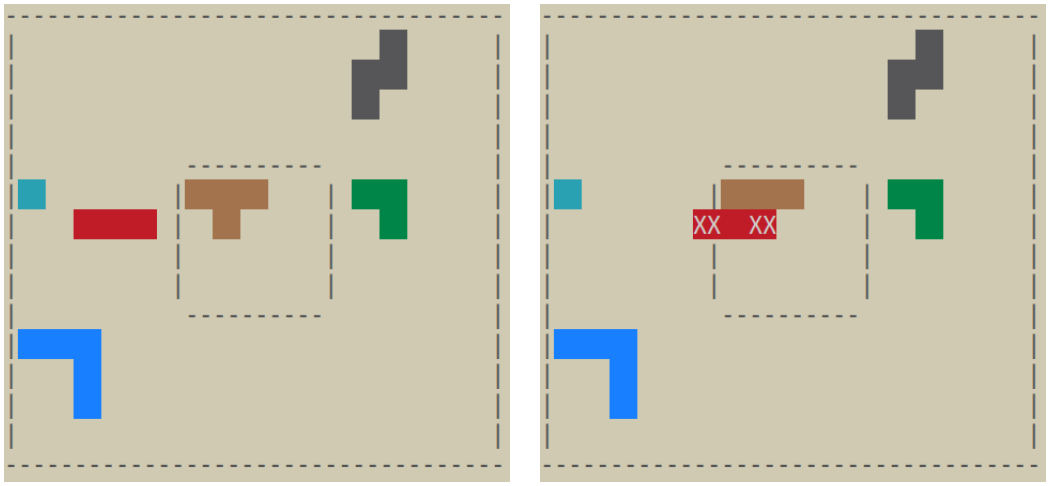

FIGURE 3 – Façon dont un positionnement invalide doit être indiqué dans le terminal. A droite, on peut voir que les blocs positionnés par dessus la bordure et l'autre pièce sont affichés avec le texte `'XX'`.

1.2.3 Déplacement des tétraminos

Un tétramino est un ensemble de blocs alignés pour composer une forme précise de couleur donnée. Comme expliqué plus tôt, ces informations vous sont fournies dans les cartes données. Pour permettre une gestion efficace et pratique des mouvements et rotations de ces pièces au cours du jeu, les tetraminos seront encodés sous forme de liste avec :

— en position 0 : les **coordonnées initiales** de la pièce, sous forme de liste de tuples. Chaque tuple correspond à la position d'un des blocs de la pièce. **Ces coordonnées ne seront modifiées que dans un seul cas : lorsqu'une rotation de la pièce aura lieu**. Elles ne changeront pas lorsque la pièce sera déplacée verticalement ou horizontalement.
— en position 1 : la **couleur** du tétramino, tel qu'indiquée dans les sections précédentes.
— en position 2 : un **décalage**, qui **correspond aux déplacements cumulés de la pièce**. Le décalage est un tuple (x, y) comptabilisant les déplacements totaux effectués sur la pièces à partir de ses coordonnées initiales. Par exemple, un bloc avec des coordonnées initiales en $(0, 0)$ et déplacé 2 fois vers la droite et 3 fois vers le bas aura un décalage de $(2, 3)$.

Toute mention aux tétraminos dans votre code à partir d'ici fera référence à cette structure.

1.2.4 Rotation des tétraminos

Pour permettre la rotation des tetraminos, vous devrez utiliser la formule suivante, étant donnés la position en abscisse et en ordonnée au moment t notées x_t, y_t :

— rotation horaire :

  - $x_{t+1} = -y_t$
  - $y_{t+1} = x_t$
- rotation anti-horaire :
  - $x_{t+1} = y_t$
  - $y_{t+1} = -x_t$

Cette formule simple vous permet de calculer la rotation de la pièce sur base des **coordonnées initiales** de la pièce. En d'autres mots, vous mettrez à jour les coordonnées initiales de la pièce en utilisant cette formule. (Si vous utilisez cette formule sur la position réelle de la pièce (coordonnées initiales incrémentées du décalage), votre rotation ne fonctionnera pas comme attendu).

1.2.5 Position initiale des tétraminos

Comme indiqué plus haut, les tétraminos devront, en début de partie, être répartis sur les 8 grilles entourant la grille centrale. L'ordre de remplissage à suivre est indiqué sur la première image de la Figure 4. Cet ordre correspond à celui dans lequel les pièces apparaissent dans le fichier texte. La pièce disposée dans sa sous-matrice sera alignée sur le coin supérieur gauche de celle-ci, comme indiqué dans la seconde image de la Figure 4.

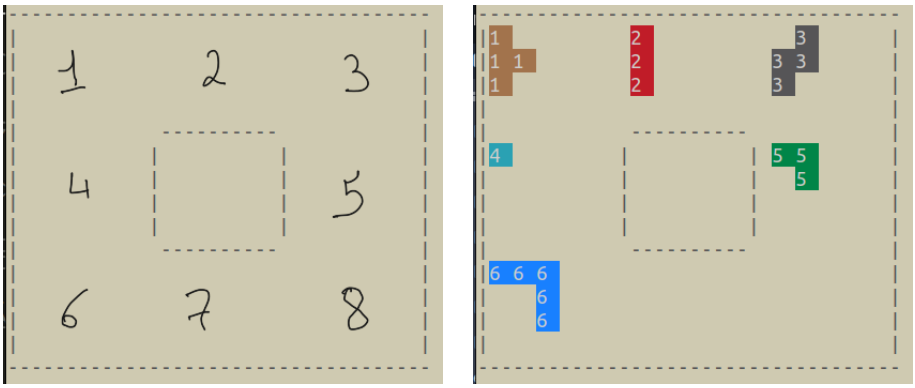

FIGURE 4 – Image de gauche : Position des tétraminos au départ du jeu. Image de droite : Position des tétraminos attendue en début de jeu pour la carte `'carte_1.txt'`

1.2.6 Gestion des déplacements

Les actions entrées par le joueur seront lues via la fonction `getkey` du fichier `getkey.py` qui vous est fournie sur l'UV. Cette fonction vous permettra de gérer les inputs de l'utilisateur facilement sans que celui-ci doive valider chaque mouvement avec la touche enter. Concrètement, la fonction `getkey` attend que l'utilisateur appuie sur une touche du clavier et renvoie ensuite le caractère correspondant (par rapport à la fonction `input`, l'avantage ici est qu'il ne faut pas appuyer sur enter).

Pour pouvoir utiliser la fonction `getkey`, mettez le fichier `getkey.py` dans le même répertoire que votre fichier `tetramino.py` et ajoutez la ligne suivante au début de ce dernier :

```
from getkey import getkey
```

Les inputs que l'utilisateur doit pouvoir rentrer sont :

— Lors du choix de la pièce à jouer : chiffres `1` à `8` pour choisir la pièce correspondante
— `i` pour déplacer la pièce vers le haut, `k` pour le bas, `j` pour la gauche et `l` pour la droite
— `o` permet de faire tourner la pièce dans le sens des aiguilles d'une montre ; `u` dans le sens inverse
— `v` permet de valider la position de la pièce (seulement si la position actuelle est valide). Si la position n'est pas valide, il ne se passe rien et le joueur doit entrer un nouveau mouvement.

1.3 Implémentation

Nous vous demandons d'implémenter les fonctions suivantes. Cette consigne doit être obligatoirement respectée, sans quoi les tests automatiques ne passeront pas, et votre code ne sera pas corrigé. Bien entendu, tant que toutes ces fonctions sont implémentées, vous pouvez tout à fait en ajouter d'autres qui vous permettraient d'écrire un code plus propre et lisible ; ceci est fortement encouragé.

Note : A partir d'ici, pour simplifier la définition des fonctions, le terme "tetramino" sera utilisé pour définir une liste contenant les informations d'un tétramino donné, à savoir (voir section 1.2.3) :

`[coordonnées initiales: list(tuple(x, y)), couleur:string, décalage: tuple(x, y)]`

Voici les fonctions qui devront impérativement être implémentées :

Fonction 1: `create_grid(w: int, h: int)`

Description:
Crée une grille de taille $(3w+2)\times(3h+2)$, comprenant les délimitations de la zone centrale.

Return:
list(list) : Matrice de taille $(3w+2)\times(3h+2)$ remplie tel qu'indiqué dans les consignes du point 1.2.2. (NB : Dans le cadre de ce projet, une matrice en python est encodée simplement comme une liste de listes).

Fonction 2: `import_card(file_path: string)`

Description:
Permet l'import d'un fichier de jeu. Cette fonction ouvre le fichier, en extrait le contenu et le traite de manière a générer les tétraminos définis ainsi que les dimensions du plateau.

Return:
(tuple`(x, y)`, list(tetramino)) → Tuple contenant les dimensions du plateau (sous forme d'un tuple `(x, y)`), ainsi qu'une liste de tetraminos

Fonction 3: `setup_tetraminos(tetraminos: list(tetramino), grid: list(list))`

Description:
Mets en place les tétraminos importés depuis le fichier chacun dans leur sous-matrice sur la matrice principale. Cette fonction est appelée une seule fois en début de jeu pour initialiser la grille de départ.

Return:
(list(list), list(tetramino)) → Nouvelle grille avec les tetraminos placés, et la liste de tétraminos avec leur nouvelle position mise à jour

Fonction 4: `place_tetraminos(tetraminos: list(tetramino), grid: list(list))`

Description:
Place les tétraminos sur la grille donnée. Cette fonction sert à effectuer un déplacement à tout moment du jeu.

Return:
list(list) → Nouvelle grille avec les tetraminos placés

Fonction 5: `rotate_tetramino(tetramino: tetramino, clockwise: bool (default=True))`

Description:
Permet d'effectuer la rotation du tetramino. Pour effectuer la rotation, seul le premier élément du tetramino, ses coordonnées initiales, est modifié. Attention à ne pas utiliser la position incrémentée par le décalage.

Return:
tetramino → Tetramino avec les coordonnées initiales modifiée par la rotation

Fonction 6: `check_move(tetramino: tetramino, grid: list(list))`

Description:
Vérifie si la position actuelle du tétramino est valide (celui-ci ne chevauche pas une autre pièce ou une bordure).

Return:
bool → `True` si la position actuelle est valide, `False` autrement.

Fonction 7: `check_win(grid: list(list))`

Description:
Vérifie si la position actuelle des pièces correspond à une configuration gagnante.

Return:
bool → `True` si la configuration est gagnante, `False` autrement.

Fonction 8: `main()`

Description:
Gère le déroulement du jeu. La fonction main est la fonction principale qui sera appellée au lancement du programme, et se terminera à la fin de la partie.

Return:
bool → `True` une fois la partie gagnée.

Fonction 9: `print_grid(grid: list(list), no_number: bool)`

Description:
Print le plateau de jeu. Pour améliorer la jouabilité, les limites du plateau doivent être affichées en utilisant les caractères '|' (bordures verticales) et (-) (bordures horizontales). Référez-vous aux figures des sections précédentes pour un exemple (par exemple : Figure 1). Le paramètre `no_number` indique si les pièces doivent être affichées avec (si sa valeur est `False`) ou sans (si sa valeur est `True`) les chiffres correspondants.

Return:
None

Comme dit précédemment, vous pouvez (et êtes encouragé(e)s) à implémenter d'autres fonctions afin de parvenir à une structure cohérente et lisible de votre code.

L'entièreté de votre code devra être mis dans un fichier appelé `tetramino.py`

1.4 Consignes de remise

Votre fichier `tetramino.py` (et uniquement ce fichier, il ne faut pas joindre `getkey.py` ni les cartes en particulier) est à remettre sur l'UV dans la section correspondante, avant le **dimanche 17 décembre à 22h**. Attention à bien respecter le nom du fichier : `tetramino.py`, tout en minuscule. (Le fichier doit être soumis tel quel, ne faites pas de `.zip`).

Remarque importante : **Aucun retard n'est possible.** (NB : n'envoyez pas votre code par email si vous êtes en retard).

Avis aux spécialistes du "dernière minute" : Nous vous conseillons fortement de soumettre une première version de votre projet sur l'UV dès que vous avez un code qui passe tous les tests, et ensuite de le mettre à jour sur l'UV après chaque session de travail. De cette façon, si jamais vous avez un problème technique de dernière minute, la version précédente de votre code sera toujours sur l'UV.

Comme indiqué plus tôt, la remise est conditionnée à la réussite de **TOUS** les tests automatiques qui vous seront fournis. Sans cela, votre code ne sera pas corrigé. Veillez donc bien à ce que votre code passe les tests.

Bon travail !