CS-119(a) – ICC-C Mini-Projet
Séries 10 - 11

Pour faire passer le temps Bob aime bien jouer aux vieux jeux vidéo. Ça lui rappelle son enfance. Malheureusement, à cause de la volatilité des marchés financiers, le site web roumain où Bob trouve ses jeux est devenu payant. Puis, *colac peste pupăză*[1], il y a quelques semaines le gouvernement vient de mettre en place un tarif de 500% afin d'encourager la croissance et le retour de la prospérité. Bob a très envie de jouer à Tetris, mais il n'a pas le budget.

Il se souvient que dans son grenier se trouve un vieil ordinateur - c'est sur cet ordinateur qu'il jouait à Tetris quand il était petit ! L'ordinateur ne fonctionne plus, mais il arrive à récupérer le disque où se trouve le code C du jeu. En fait c'était un clone du jeu original que sa tante avait codé à l'époque (juste avant de s'enfuir avec un riche entrepreneur de l'Europe de l'Est, c'est une longue histoire). Bref, il suffirait de compiler le code du jeu. Mais zut ! Le disque est cassé et un des fichiers C est irrécupérable... Bob vous supplie de l'aider. Heureusement vous venez d'apprendre le langage C.

Le fichier en question s'appelle `engine.c` et doit contenir l'implémentation de quelques fonctions essentielles au déroulement du jeu. On sait ce qu'on doit coder, car il y a toujours le fichier entête `engine.h` qui contient les déclarations de ces fonctions, ainsi que quelques définitions de types.

Détails techniques

Les fonctions que vous devez implémenter sont décrites plus bas. C'est **essentiel** d'utiliser **exactement** la déclaration indiquée afin de faciliter l'évaluation automatique. Chaque fonction sera soumise à une suite de tests pour vérifier que votre code fonctionne correctement (on ne veut pas que Bob soit déçu). La note dépendra du nombre de tests réussis par vos fonctions.

1. expression roumaine que Bob a appris sur le forum du site, littéralement "bretzel par-dessus une huppe"

Le fichier `engine.c` ne doit pas contenir de fonction `main()`, car elle existe déjà ailleurs.

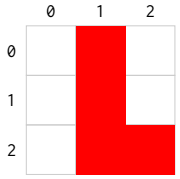
Une pièce est représentée comme un tableau bi-dimensionnel carré qui est stocké dans une `struct` adaptée :

```
struct piece
{
    int **shape;
    int size;
};
```

Vous pouvez supposer que toutes les fonctions ayant un paramètre “pièce” seront appelées avec une `struct` dont le pointeur `shape` est correctement initialisé et pointe vers un tableau bi-dimensionnel à `size` lignes et `size` colonnes. Autrement dit, le pointeur double `shape` pointe vers un tableau de `int*` (pointeurs vers `int`) de taille `size`, dont chacun pointe vers un tableau de `int` de taille `size` (la gestion de la mémoire dynamique est faite ailleurs).

La pièce est composée de blocs marqués par des `1`. Les cases vides du tableau contiennent des `0`.

Par exemple la pièce en “L” :



est décrite par le tableau de taille `3 x 3`

```
0 1 0
0 1 0
0 1 1
```

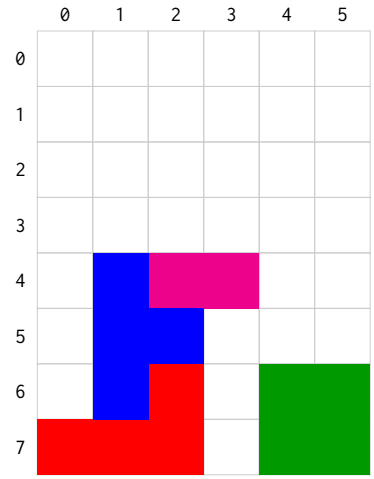
La grille de jeu est représentée comme un autre tableau bi-dimensionnel rectangulaire qui est stocké dans une autre `struct` adaptée :

```
struct board
{
    int **data;
    int n_rows;
    int n_cols;
};
```

Vous pouvez supposer que toutes les fonctions ayant un paramètre “grille de jeu” seront appelées avec une `struct` dont le pointeur `data` est correctement initialisé et pointe vers un tableau bi-dimensionnel à `n_rows` lignes et `n_cols` colonnes.

Comme dans le cas des pièces, une case vide dans la grille de jeu correspond à `0` dans le tableau `data` et une case occupée correspond à un entier positif. Cet entier représente la couleur du bloc de la pièce à laquelle il avait appartenu.

Par exemple, la grille `8 x 6` qu'on pourrait obtenir à une certaine étape du jeu



est décrite par le tableau de taille `8 x 6`

```
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 2 3 3 0 0
0 2 2 0 0 0
0 2 4 0 1 1
4 4 4 0 1 1
```

Exo1 Tourner une pièce

Lorsqu'on appuie sur  la pièce tourne de 90 degrés dans le sens horaire.

Implémentez une fonction `rotate_piece` qui prend un pointeur constant vers une pièce originale `piece` et un pointeur vers une nouvelle pièce `rotated` de la même taille que la première. Il ne faut pas allouer de mémoire, c'est le code qui appelle la fonction qui s'en charge. La fonction écrit dans `rotated->shape` un tableau qui correspond à la rotation de 90 degrés dans le sens horaire autour du centre du tableau de la pièce originale `piece`.

Pour assurer la compatibilité avec le reste du code, la fonction doit impérativement avoir la signature suivante :

```
void rotate_piece(
    const struct piece *piece,
    struct piece *rotated);
```

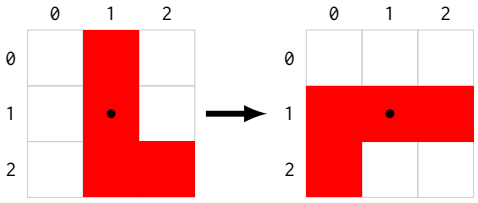
Par exemple, si le tableau `piece->shape` contient

```
0 1 0
0 1 0
0 1 1
```

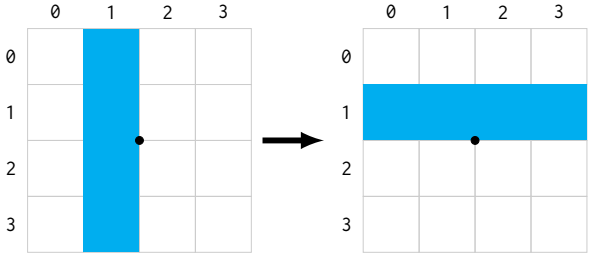
alors après l'appel à `rotate_piece` le tableau `rotated->shape` doit contenir

```
0 0 0
1 1 1
1 0 0
```

Pour le cas `size = 3` ceci correspond à une rotation autour de la case du centre du tableau (ligne 1, colonne 1) marquée ci-dessous par un point :



Si la pièce est contenue dans un tableau carré de côté de longueur paire, alors le centre de rotation est au milieu du tableau entre les cases. Par exemple, si `size` est `4` le centre de rotation est entre les lignes 1 et 2 et entre les colonnes 1 et 2 :



Exo2 Position valide

Le jeu doit être capable de savoir si une pièce peut se déplacer ou tourner sans heurter des cases occupées de la grille de jeu.

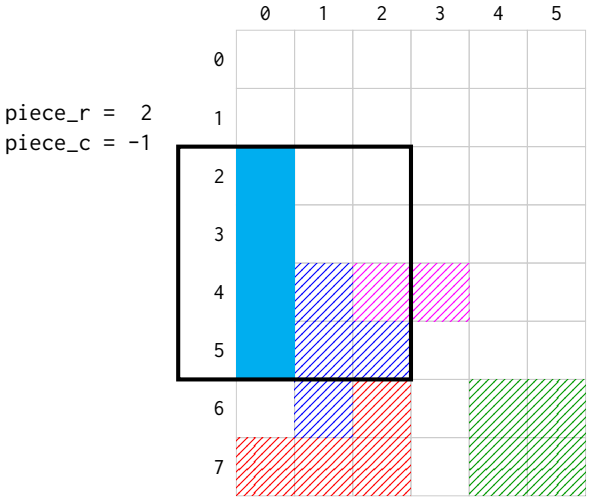
Implémentez une fonction

```
int is_valid_position(
    const struct board *board,
    const struct piece *piece,
    int piece_r,
    int piece_c);
```

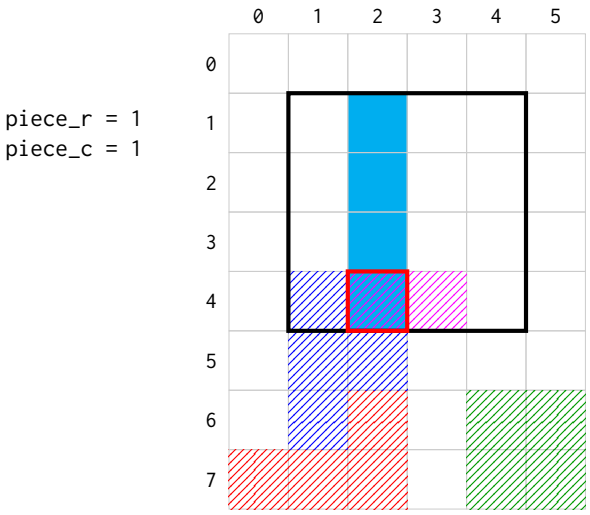
qui retourne `1` si on peut placer une pièce `piece` à la ligne `piece_r` et à la colonne `piece_c` sur la grille de jeu `board` et `0` sinon. Les paramètres `piece_r` et `piece_c` représentent les coordonnées où on veut placer la case `(0, 0)` du tableau `piece->shape` sur la grille de jeu.

Attention, il se peut que la case (`piece_r`, `piece_c`) ne soit pas sur la grille de jeu, mais que les blocs non-vides de la pièce soient quand-même placés sur la grille sans couvrir aucune case occupée. Cette situation est valide et la fonction doit retourner `1` dans ce cas.

Ci-dessous on voit un tel exemple. On représente la grille de jeu donnée en exemple auparavant et un tableau en position `piece_r = 2, piece_c = -1`, mais dont l'emplacement est valide, car chaque bloc de la pièce contenue dans le tableau occupe une case libre sur la grille de jeu.



Au contraire, l'exemple suivant montre une situation où la fonction doit retourner 0, car la pièce contenue dans le tableau placé à la case `piece_r = 1, piece_c = 1` occupe la case `(4, 2)` de la grille de jeu qui est déjà remplie.



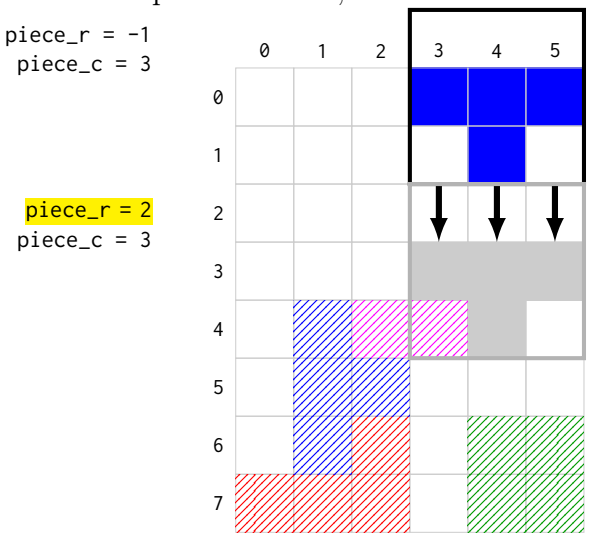
Exo3 Drop

Si on appuie sur la touche "espace"  la pièce active est envoyée vers le bas de la grille de jeu. Implémentez une fonction qui prend une grille de jeu `board`, une pièce `piece`, ainsi que sa position (`piece_r`, `piece_c`), et qui retourne **la ligne** où la pièce se retrouverait si Bob appuyait sur la touche “espace”.

```
int drop_piece(
    const struct board *board,
    const struct piece *piece,
    int piece_r,
    int piece_c);
```

On peut supposer que la position initiale est valide.

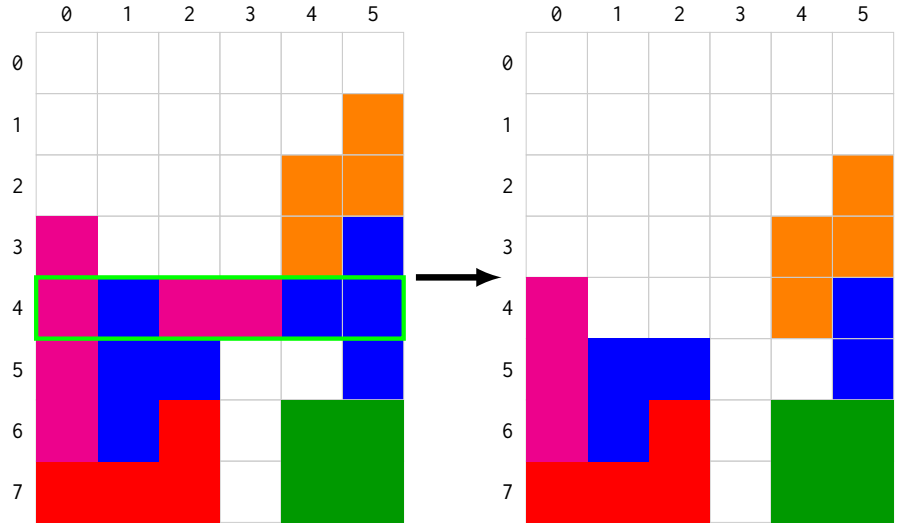
Pour l’exemple ci-dessous, la fonction doit retourner `2` :



Exo4 Éliminer des lignes

Le but de Tetris est de placer une pièce de telle manière à remplir une (ou plusieurs) lignes. Ceci nous fait gagner des points, élimine les lignes remplies, et fait descendre les cases placées au dessus de ces lignes.

Par exemple, la longue pièce à 4 blocs vient d'être placée verticalement tout à gauche de la grille de jeu ci-dessous. La ligne 4 sera éliminée, et toutes les cases qui se trouvent au dessus de la ligne 4 seront décalées vers le bas d'une ligne :



Écrivez une fonction

```
int clear_lines(struct board *board);
```

qui prend une grille de jeu, élimine les lignes remplies de la grille (il se peut qu'il y en ait plusieurs), décale vers le bas les blocs se trouvant au dessus de ces lignes, et retourne le nombre de lignes ainsi éliminées.

Pour l'exemple qu'on vient de montrer, la fonction reçoit en entrée la grille ci-dessous et la transforme comme indiqué

```
0 0 0 0 0 0          0 0 0 0 0 0
0 0 0 0 0 5          0 0 0 0 0 0
0 0 0 0 5 5          0 0 0 0 0 5
3 0 0 0 5 2   ==>    0 0 0 0 5 5
3 2 3 3 2 2          3 0 0 0 5 2
3 2 2 0 0 2          3 2 2 0 0 2
3 2 4 0 1 1          3 2 4 0 1 1
4 4 4 0 1 1          4 4 4 0 1 1
```

La fonction retourne `1` dans ce cas (une seule ligne éliminée).

Tetris AI

Nous voulons que notre programme sache jouer tout seul à Tetris en mode “autopilote”. Une façon de faire est de définir une fonction qui prend une grille de jeu, une pièce, et une position de cette pièce, et qui calcule un score qui reflète à quel point cette position de la pièce est désirable. Ensuite nous pouvons simplement évaluer ce score pour beaucoup de positions sur la grille et choisir celle qui a le meilleur score. Enfin, il s'agit de guider la pièce vers la position choisie en simulant les actions d'un joueur.

Exo5 Fonction score

Voici comment calculer le score d'une position :

Si dans la position considérée la pièce peut continuer à descendre, ou si la position est invalide, alors le score associé à cette position est de `-1000`.

Sinon, notre score a trois composants :

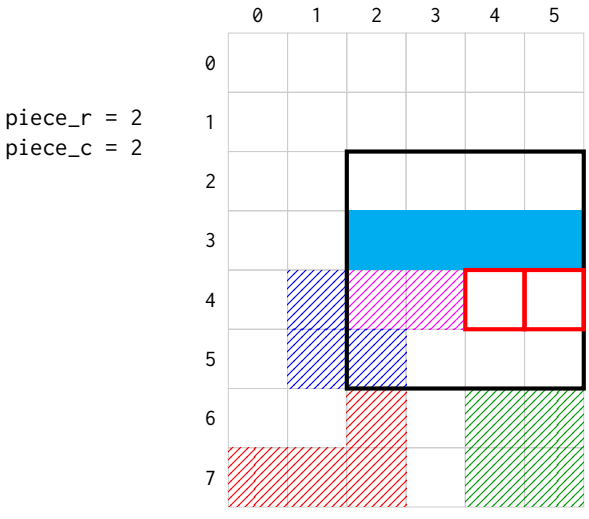
`L =` Le nombre de lignes qui seraient éliminées si on plaçait notre pièce dans la position considérée.

`H =` A quelle hauteur par rapport au bas de la grille se situerait le point le plus haut de la pièce.

`T =` Combien de “trous” sommes-nous en train de créer. Un “trou” est une case vide située **juste en dessous** d'un bloc qui appartient à la pièce.

La formule du score de la position est `10*L - H - 2*T`, car nous souhaitons éliminer des lignes dès que possible, mais une petite hauteur est désirable, et nous ne voulons pas créer des trous si possible.

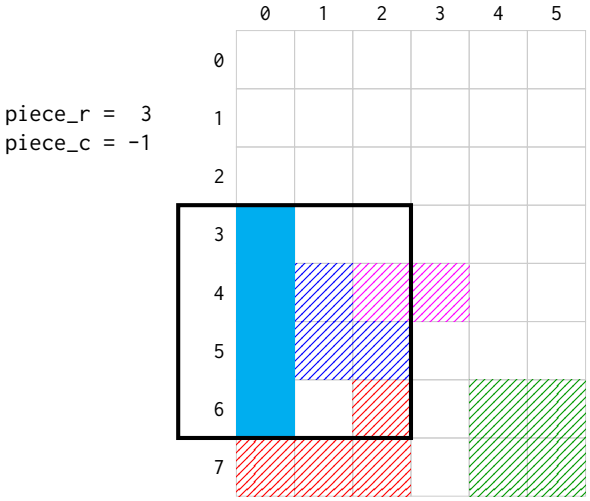
Par exemple, ci-dessous le score de la position `(2, 2)` de la longue pièce couchée est de `-9` :



Effectivement, dans ce cas `L = 0`, `H = 5` (c'est la 5e ligne si on compte depuis

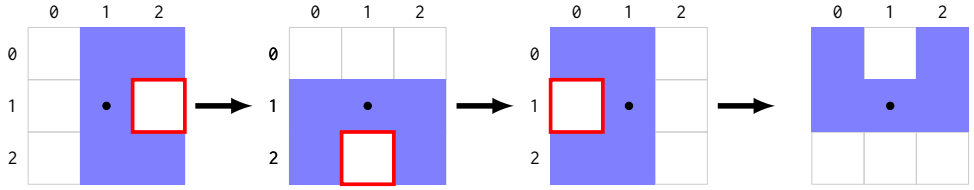
le bas), et `T = 2`, car la pièce repose bien sur 2 cases occupées de la grille, mais les deux cases libres `(4, 4)` et `(4, 5)` marquées en rouge sont des trous. Les cases `(5, 3)`, `(5, 4)`, etc. sont aussi vides, mais elles ne sont pas collées à la pièce et donc ne comptent pas comme des trous.

On aurait un meilleur score si on mettait la pièce verticalement à la position `(3, -1)`. Pour cette configuration on obtiendrait un score de `-5` :



Effectivement, dans ce cas `L = 0`, `H = 5` (le point le plus haut de la pièce est toujours à la 5e ligne si on compte depuis le bas), par contre `T = 0`, car il n'y a pas de trou – la case `(6, 1)` est vide, mais elle n'est pas **en dessous** d'un bloc appartenant à la pièce et ne compte pas comme un trou.

Notez également qu'une pièce en forme de "`C`" aura toujours au moins un trou à cause de sa forme peu importe où elle est placée, sauf si elle subit des rotations pour la transformer en forme de "`U`" :



Écrivez une fonction

```
int position_score(
    const struct board *board,
    const struct piece *piece,
    int piece_r,
    int piece_c);
```

qui prend une grille de jeu, une pièce et une position possible, et qui retourne le score de la position dans le contexte de la grille donnée.

Exo6 La meilleure position

Maintenant qu'on sait calculer le score d'une position, il faut déterminer le meilleur emplacement depuis lequel envoyer la pièce en bas de la grille.

Écrivez une fonction

```
int find_best_position(
    const struct board *board,
    const struct piece *pieces,
    int *best_c,
    int *best_rot);
```

qui prend une grille de jeu et un tableau représentant les quatre rotations possibles d'une même pièce. Cette fonction trouve la meilleure colonne `best_c` et la meilleure rotation `best_rot` selon le critère suivant : après avoir appliqué la rotation choisie à la pièce, on la place à la ligne 0 et à la colonne choisie, on l'envoie en bas de la grille (comme si on appuyait sur la touche espace), et on calcule le score de cette position possible de la pièce.

Le paramètre `best_rot` pointe vers un entier où la fonction place la valeur de la meilleure rotation, c'est à dire un chiffre entre 0 et 3 qui correspond à l'indice dans le tableau de pièces `pieces` reçu en paramètre. Le paramètre `best_c` pointe vers un entier où la fonction écrit la meilleure colonne. Cet entier peut être négatif (voir l'exercice 2).

La fonction retourne la valeur du meilleur score calculé. Si plusieurs réponses atteignent la valeur maximale du critère, n'importe laquelle est acceptée.

Dans l'exemple ci-dessous pour la pièce en "`T`" on obtient la colonne 4 et la rotation qui "couche" le "`T`". Le meilleur score est de `10 - 5 - 2 = 3` : on remplit une ligne (`+10`), on a une hauteur de 5 (`-5`), et on introduit quand même un trou (`-2 * 1`).