

**420-V21-SF**  
**PROGRAMMATION DE JEUX VIDÉO II**  
**TRAVAIL PRATIQUE #1**  
**LE JEU DE TETRIS**  
**PONDÉRATION : 10%**

**OBJECTIFS PÉDAGOGIQUES**

Ce travail vise à familiariser l'étudiante ou l'étudiant avec les objectifs suivants :

- Procéder à la codification d'une classe (compétence 016T-2).
- Valider le fonctionnement d'une classe (compétence 016T-4)
- Générer la version exécutable d'un programme (compétence 016T-5)

De même, il permet à l'étudiante ou à l'étudiant d'appliquer les standards de conception, de documentation et de programmation.

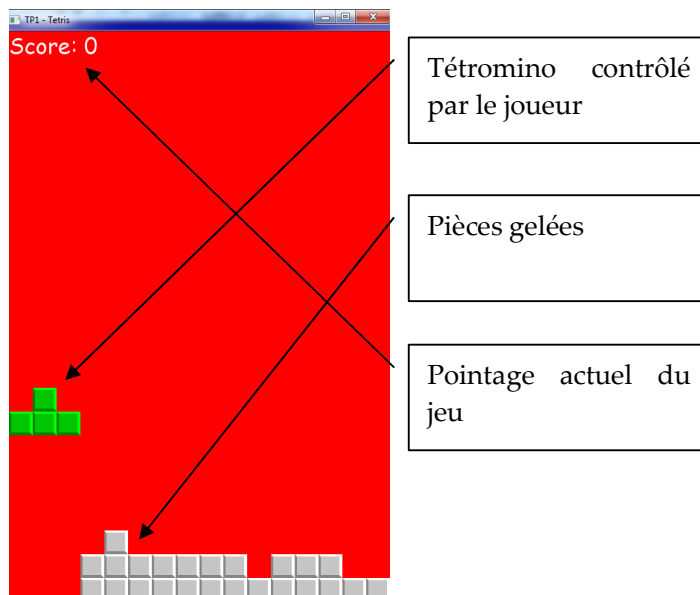
**MISE EN CONTEXTE ET MANDAT**

Pour ce premier travail, vous devez réaliser le jeu bien connu « Tetris ». Il existe une très grande variété de jeux similaires reprenant les mêmes fonctionnalités. Vous pouvez en trouver plusieurs exemples en recherchant sur Internet.


**SPÉCIFICATIONS DE L'APPLICATION**

Votre application doit notamment respecter les contraintes suivantes :

- Elle doit afficher au minimum un tétramino<sup>1</sup> contrôlé par le joueur, les pièces « gelées », i.e. qui ne peuvent plus être déplacées ainsi que le pointage actuel:



<sup>1</sup> C'est le nom officiel d'une pièce (<https://fr.wikipedia.org/wiki/T%C3%A9tromino> )

- Le jeu doit permettre de quitter la partie à l'aide du bouton habituel .
- A chaque tour, le joueur doit pouvoir faire déplacer ou tourner le tétramino courant. Il doit pouvoir utiliser une touche pour le faire descendre plus rapidement.
- Le tétramino courant doit descendre dans le jeu à un rythme régulier. Le rythme de descente doit s'accélérer au fur et à mesure que le joueur complète des lignes. Au début d'une partie, la pièce doit descendre à environ toutes les secondes.
- Lorsque le tétramino courant ne peut plus être déplacé, il doit être « figé » et ses sous-blocs doivent être par la suite dessinés en blanc à l'écran.
- Lorsqu'une ligne est complétée, la ligne doit être retirée du tableau de jeu et les lignes supérieures doivent être décalées vers le bas.
- Votre jeu doit comprendre un système de pointage dont vous définirez les modalités (la manière dont se calculeront les points). Le pointage doit être mis à jour à chaque ligne complétée et doit s'afficher dans l'écran à chaque rendu.
- Lorsqu'un tétramino est figé, un autre est généré au hasard en haut de l'écran (au milieu). Ce nouveau tétramino devient alors la pièce active.
- La couleur d'un tétramino dépend de son type. Chaque type possède une couleur différente (si possible car le nombre de couleurs est limité).
- La partie se termine lorsqu'un nouveau tétramino généré ne peut pas être descendu vers le bas.
- À la fin de la partie, le jeu doit afficher un message indiquant que la partie est terminée et demander au joueur s'il souhaite poursuivre ou quitter.
- Le jeu doit être programmé en utilisant les standards de programmation qui vous ont été communiqués en classe en début de session. En particulier, le code doit :
  - Être correctement indenté et ne doit contenir qu'une seule instruction par ligne.
  - Utiliser les structures de contrôles adéquates (`while`, `for`, `if`, etc).
  - Être séparé adéquatement en classes et méthodes pour faciliter la lisibilité et la réutilisation.
  - Utiliser les principes de programmation objet vus en classe, notamment en ce qui a trait à la visibilité des méthodes et propriétés.

Vous devez également commenter à profusion les instructions, les déclarations de variables et les entêtes de méthodes.

- Finalement, vous devez produire tous les tests pertinents demandés.

#### CONTEXTE DE RÉALISATION ET DÉMARCHÉ DE DÉVELOPPEMENT

Ce travail pratique doit être réalisé **seul**.

Les pages qui suivent résument les étapes du projet ainsi que les biens livrables devant être remis à la fin de chacune d'elles.

<b>Étape 1:      Analyse de la situation</b>
--

Dans cette étape, vous devez prendre connaissance du mandat qui vous est confié.

**Biens livrables :**

- Aucun.

**Méthode:**

- s/o

<b>Étape 2:      Programmation de l'application</b>
---

Vous devez procéder à la codification des classes contenues dans le diagramme de classes fourni (voir la section plus loin). Vous devez également procéder à la codification des tests unitaires pertinents demandés.

Pour cela, vous devez récupérer la base de code fournie sur LÉA et compléter le code des deux projets (TP1 et UnitTestsTP1).

**Biens livrables :**

- Code source de l'application documenté (selon les standards établis dans le cours) pour chacune des classes `Tetromino`, `TetrisGame` et `TetrominoLittleBlock` ainsi que pour le type énuméré `TetrominoType` que vous coderez.
- Pour les tests, utilisez la « coquille » du projet fourni et créez les différents tests requis. **Documentez adéquatement chaque test pour décrire son rôle.**

**Méthode:**

- Vous devez remettre votre code source dans une archive .zip identifiée en fonction de votre nom. Par exemple, pour l'étudiant nommé Pierre Poulin, l'archive devrait être nommée `TP1_PierrePoulin.zip`.

Cette archive doit être déposée sur LÉA **avant le 2 mars 2016 à 23h59**.

N'oubliez pas de documenter adéquatement vos entêtes de fichiers.

## MODALITÉS D'ÉVALUATION

Vous trouverez sur LÉA la grille de correction qui sera utilisée pour le travail. **Cette grille indique la pondération accordée à chacune des parties du projet.**

- Tous les **biens livrables** de chacune des étapes devront être **remis à temps** et selon les modalités spécifiées.
- Nous rappelons que la **présence à tous les cours (de corps et d'esprit)** est **FORTEMENT RECOMMANDÉE**.
- Je vous recommande aussi de profiter de toutes les périodes de disponibilité qui vous sont offertes pendant la semaine.

## STRATÉGIE DE DÉVELOPPEMENT.

Afin d'éviter de devoir gérer beaucoup de bogues en même temps, je vous suggère fortement de procéder à la codification de manière structurée.

Pour cela, procéder classe par classe, méthode par méthode en vous assurant de coder immédiatement les tests requis après chaque méthode (autant que possible). Ainsi, vous découvrirez rapidement les problèmes et serez en mesure de les régler rapidement et ce, sans que les problèmes en question ne se répercutent sur l'ensemble du programme.

Je vous suggère de procéder ainsi (dans l'ordre) :

- Prise de connaissance du projet « exemple » fourni. Portez une attention particulière à la gestion des entrées et à la fermeture de l'application.
- Prise de connaissance du code de la classe `Application`. Notez en particulier que le code de la méthode `OnKeyPressed` a été placé en commentaires. Vous pourrez enlever les commentaires lorsque les méthodes manquantes auront été codées.

**C'est la seule modification que vous aurez à apporter au fichier `Application.cs`.**

- Écriture du squelette de la classe `TetrisGame` :
  - Constructeur
  - Tableau logique de booléens.
  - Méthode `UpdateGame`. Faites en sorte d'incrémenter le pointage courant à chaque appel de la méthode.
  - Méthode `Draw`. Faites afficher le pointage à l'écran en enlevant les commentaires présents dans la méthode.
- Création du type énuméré `TetrominoType` selon les valeurs possibles. Vous pouvez inscrire toutes les valeurs qui apparaissent dans le diagramme de classes mais pour l'instant, seule la valeur `Square` sera considérée.
- Création de la classe `TetrominoLittleBlock`. Codez la classe au complet. Assurez-vous de tester l'affichage d'un bloc. Pour cela, créez-vous quelques

instances de la classe `TetrominoLittleBlock` dans la class `TetrisGame` et faites-les afficher à chaque appel de la méthode `Draw` de la classe `TetrisGame`. Passez différentes valeurs des paramètres `parentColumn` et `parentRow` pour bien en comprendre le rôle. Inspirez-vous du projet « Exemple » pour l’affichage.

**Ne passez pas à l’étape suivante tant que vous n’aurez pas terminé cette étape.**

- Création de la classe `Tetromino` :

- Constructeur. Par défaut, **ne considérez pour l’instant qu’un bloc carré 2x2**. A titre informatif, un tétramino carré 2x2 (celui correspondant au type `TetrominoType.Square`) devrait contenir 4 `TetrominoLittleBlock` `b1`, `b2`, `b3` et `b4` alloués ainsi (en supposant que la couleur du tétramino soit définie par la propriété `blockColor` :

```

▪ b1 = new TetrominoLittleBlock( 0, 0, blockColor );
▪ b2 = new TetrominoLittleBlock( 0, 1, blockColor );
▪ b3 = new TetrominoLittleBlock( 1, 0, blockColor );
▪ b4 = new TetrominoLittleBlock( 1, 1, blockColor );

```

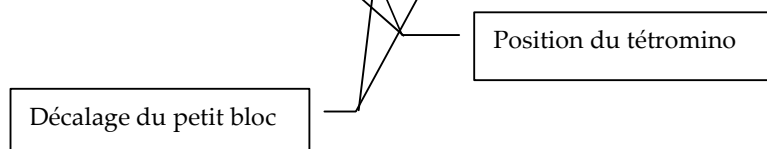
Ici, les paires  $(0,0)$ ,  $(0,1)$ ,  $(1,0)$  et  $(1,1)$  représentent les décalages des petits blocs par rapport à la position du tétramino (Voir schéma à la fin).

- Méthode `Draw`. La méthode `Draw` devrait appeler 4 fois la méthode `Draw` de la classe `TetrominoLittleBlock`, une pour chacune des propriétés `b1`, `b2`, `b3` et `b4`.
- Méthode `MoveDown`. La méthode `MoveDown` devrait seulement incrémenter le décalage en ligne par rapport au coin supérieur gauche du jeu.
- Méthode `CanMoveDown`. La méthode `CanMoveDown` devrait uniquement s’assurer que chaque `TetrominoLittleBlock` qui constitue le tétramino peut descendre. Par exemple, pour le `TetrominoLittleBlock` `b1`, il est possible de faire cela en faisant :

```

int potentialNextColumn = topLeftColumnOffset + b1.GetParentColumnOffset( );
int potentialNexRow = topLeftRowOffset + b1.GetParentRowOffset( ) + 1;
bool canMoveDown=!game.IsContentFrozen(potentialNextColumn, potentialNexRow);

```



où

`potentialNextColumn` représente la colonne où l’on veut essayer de descendre le petit bloc, i.e. celle qui tient compte de la position du tétramino ET du décalage du petit bloc `b1`

`potentialNexRow` représente la ligne où l’on veut essayer de descendre le petit bloc, i.e. celle qui tient compte de la position du tétramino ET du décalage du petit bloc `b1`

**Ne passez pas à l’étape suivante tant que vous n’aurez pas terminé cette étape.**

- Consultez les tests pertinents pour les méthodes `MoveDown` et `CanMoveDown`. Enlevez les commentaires.
- Méthode `MoveLeft`. La méthode `MoveLeft` devrait seulement décrémenter le décalage en colonne par rapport au coin supérieur gauche du jeu.
- Méthode `CanMoveLeft`. Pour la méthode `CanMoveLeft`, **inspirez-vous fortement du code de la méthode `CanMoveDown`**.
- Méthode `MoveRight`. La méthode `MoveRight` devrait seulement incrémenter le décalage en colonne par rapport au coin supérieur gauche du jeu.
- Méthode `CanMoveRight`. Pour la méthode `CanMoveRight`, **inspirez-vous fortement du code de la méthode `CanMoveLeft`**.
- Complétez les tests pertinents pour tester adéquatement les méthodes `MoveLeft`, `CanMoveLeft`, `MoveRight` et `CanMoveRight`. Ne codez que les tests utilisant un tétramino « carré ». Inspirez-vous de ceux fournis pour les méthodes `MoveDown` et `CanMoveDown`.

**Ne passez pas à l'étape suivante tant que vous n'aurez pas terminé cette étape.**

- Continuez l'écriture de la classe `TetrisGame` pour gérer la mise à jour du tétramino courant et la gestion du « gel » d'un tétramino qui ne peut plus bouger (i.e. la méthode `CanMoveDown` retourne faux). N'oubliez pas de modifier la méthode `Draw` pour dessiner les blocs qui auront été gelés.
- Écrivez ensuite le code nécessaire pour retirer les lignes complètes.
- Complétez ensuite les tests requis pour le retrait des lignes.
- Ajoutez la gestion de la fin de partie dans la méthode `UpdateGame`.

**Ne passez pas à l'étape suivante tant que vous n'aurez pas terminé cette étape.**

- Modifiez le code du constructeur de tétramino pour pouvoir faire aussi avoir des pièces « droites » (4 en ligne). Modifiez le code de la méthode `CreateNewTetrisBlock` pour rendre possible la génération de cette nouvelle pièce.
- Ajoutez le code requis pour pouvoir faire tourner la pièce courante. Il s'agit probablement de l'étape du travail qui vous générera le plus d'erreurs. Prenez le temps de les régler en totalité.
  - Méthode `CanRotateCW`
  - Méthode `RotateCW`
  - Méthode `CanRotateCCW`
  - Méthode `RotateCCW`
  - Complétez les tests pertinents pour tester adéquatement les méthodes `CanRotateCW`, `RotateCW`, `CanRotateCCW` et `RotateCCW`. Ne codez que

les tests utilisant un tétramino « carré ».

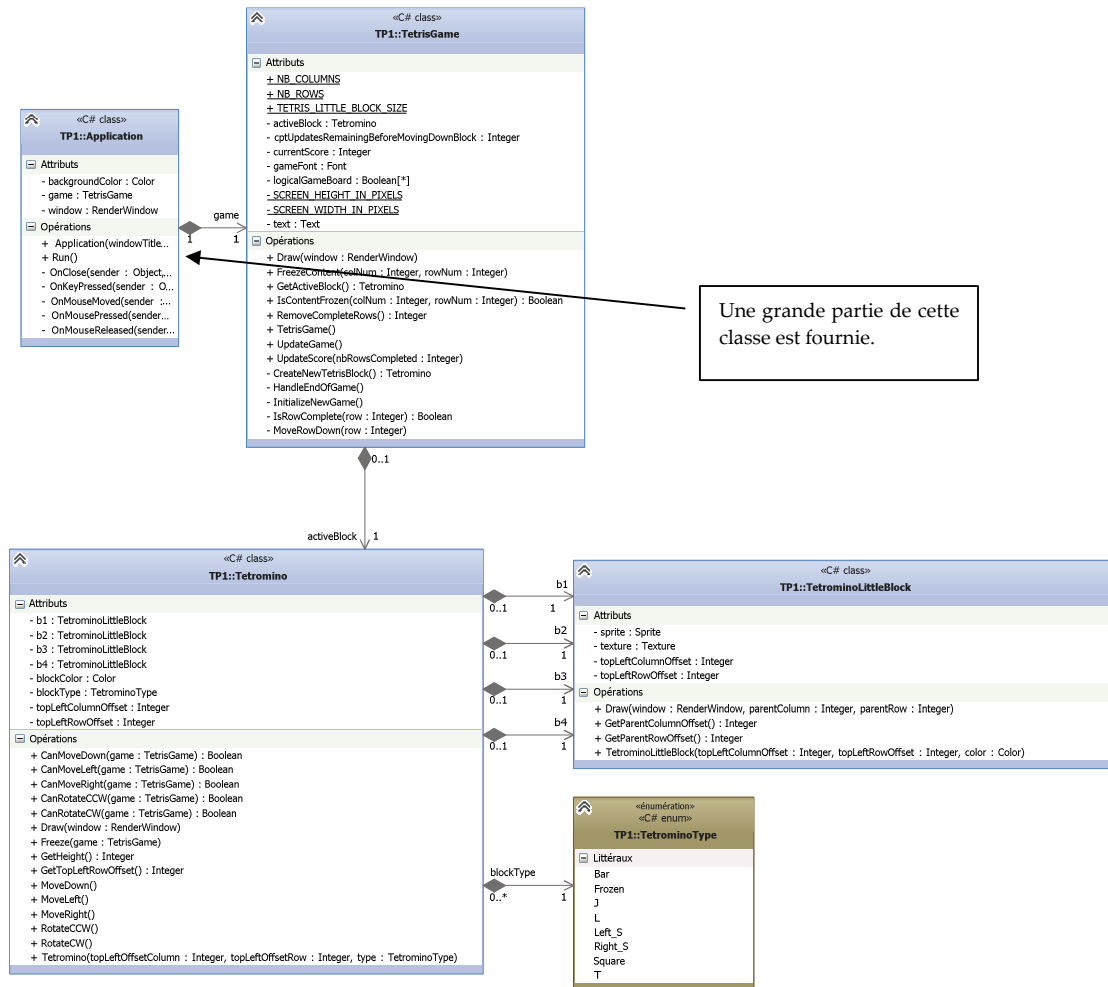
**Ne passez pas à l'étape suivante tant que vous n'aurez pas terminé cette étape.**

- Modifiez le code du constructeur de tétramino pour pouvoir gérer une nouvelle pièce (ex. en L) et modifiez le code de la méthode `CreateNewTetrisBlock` en conséquence.

**Si votre travail a été fait correctement auparavant, vous ne devriez pas avoir d'autres modifications à faire pour gérer cette nouvelle pièce!** Vous pouvez donc ajouter les if/else if nécessaires dans le constructeur pour gérer toutes les pièces que vous désirez.

- Relisez votre code et ajoutez la documentation pertinente.

## DIAGRAMME DE CLASSES ET EXPLICATIONS



### La classe TetrisGame.

Propriétés	Commentaires
SCREEN_WIDTH_IN_PIXELS	Constante entière. Largeur de la fenêtre de rendu. Fixé dans mon projet à 512.
SCREEN_HEIGHT_IN_PIXELS	Constante entière. Hauteur de la fenêtre de rendu. Fixé dans mon projet à 768.
TETRIS_LITTLE_BLOCK_SIZE	Constante entière. Largeur (et hauteur) d'un petit bloc (un <i>quart</i> de tétramino). Fixé dans mon projet à 32.
NB_COLUMNS	Constante entière. Nombre de colonnes



	dans le jeu.
NB_ROWS	Constante entière. Nombre de lignes dans le jeu.
cptUpdatesRemainingBeforeMovingDownBlock	Nombre de mises à jour visuelles à faire avant de forcer un déplacement vers le bas du tétramino actif.
currentScore	Pointage courant.
logicalGameBoard	Tableau 2D de booléens. Chaque booléen indique si la case est figée (remplie → true) ou non (non remplie → false).
activeBlock	Tétramino actif.
text	De type Text (de la bibliothèque SFML). Utilisé pour afficher le pointage à l'écran.
gameFont	De type Font (de la bibliothèque SFML). Utilisé pour afficher le pointage à l'écran.

Méthodes	Paramètres	Commentaires
TetrisGame		Constructeur de TetrisGame.
InitializeNewGame		Met en place les éléments nécessaires à une nouvelle partie.
CreateNewTetrisBlock		Crée un nouveau tétramino avec un type déterminé aléatoirement.
Draw	L'objet de type RenderWindow associé à la fenêtre de rendu	Dessine le jeu à l'écran : pointage, tétramino actif et pièces gelées. Voir les exercices en classe pour la manière de procéder.
GetActiveBlock		Retourne le tétramino actif.
UpdateGame		Met à jour le jeu. Cela inclut faire descendre le bloc actif au besoin, gérer la fin de partie et geler une pièce qui ne peut plus bouger (si applicable).
FreezeContent	Les coordonnées en colonne et ligne de la pièce à geler	Gèle dans le tableau logique la pièce spécifiée à la position reçue en paramètre.
RemoveCompleteRows		Retire du tableau logique les lignes complètes.
IsRowComplete	Numéro de la ligne à vérifier	Retourne true si la ligne est complète, false sinon.
MoveRowDown	Numéro de la ligne à descendre vers le bas	« Descend » toutes les lignes supérieures ou égales à celle reçue en paramètre dans le tableau logique.

IsContentFrozen	Les coordonnées en colonne et ligne de la pièce à vérifier	Retourne true si la pièce à la coordonnée reçue en paramètre est gelée, false sinon
UpdateScore	Nombre de nouvelles lignes complétées	Met à jour le pointage en fonction du nombre de lignes nouvellement complétées.
HandleEndOfGame		Gère la fin de partie en demandant au joueur s'il souhaite rejouer ou quitter le jeu.

## La classe Tetromino.

Propriétés	Commentaires
topLeftColumnOffset	Le décalage en colonne du tétramino par rapport au coin supérieur gauche
topLeftRowOffset	Le décalage en ligne du tétramino par rapport au coin supérieur gauche
blockColor	La couleur du tétramino. De type SFML.Graphics.Color
blockType	Le type du bloc. De type TetrominoType.
b1, b2, b3 et b4	Les quatre petits blocs qui forment le tétramino. De type TetrominoLittleBlock.

Méthodes	Paramètres	Commentaires
Tetromino	Le décalage par rapport au coin supérieur gauche et le type du bloc	Constructeur de Tetromino.
GetTopLeftRowOffset		Retourne le décalage en ligne par rapport au coin supérieur gauche.
Freeze	Le jeu dans lequel on doit geler les blocs.	Gèle le contenu du tétramino dans le tableau logique du jeu.
GetHeight		Retourne le nombre de blocs occupé par le tétramino en hauteur. Le résultat va dépendre de l'orientation de la pièce.
Draw	L'objet de type RenderWindow associé à la fenêtre de rendu	Dessine le tétramino à l'écran. Voir les exercices en classe pour la manière de procéder.
CanMoveLeft	Le jeu dans lequel on doit valider les cases gelées	Retourne true si le tétramino peut être déplacé vers la gauche ou false sinon
MoveLeft		Déplace le tétramino vers la gauche

CanMoveRight	Le jeu dans lequel on doit valider les cases gelées	Retourne <code>true</code> si le tétramino peut être déplacé vers la gauche ou <code>false</code> sinon
MoveRight		Déplace le tétramino vers la droite
CanMoveDown	Le jeu dans lequel on doit valider les cases gelées	Retourne <code>true</code> si le tétramino peut être déplacé vers le bas ou <code>false</code> sinon
MoveDown		Déplace le tétramino vers le bas
CanRotateCW	Le jeu dans lequel on doit valider les cases gelées	Retourne <code>true</code> si le tétramino peut être tournée dans le sens des aiguilles d'une montre ou <code>false</code> sinon
RotateCW		Fait tourner le tétramino dans le sens des aiguilles d'une montre.
CanRotateCCW	Le jeu dans lequel on doit valider les cases gelées	Retourne <code>true</code> si le tétramino peut être tournée dans le sens contraire des aiguilles d'une montre ou <code>false</code> sinon
RotateCCW		Fait tourner le tétramino dans le sens contraire des aiguilles d'une montre.

### La classe `TetrominoLittleBlock`.

Propriétés	Commentaires
<code>topLeftColumnOffset</code>	Le décalage en colonne du tétramino par rapport au tétramino parent
<code>topLeftRowOffset</code>	Le décalage en ligne du tétramino par rapport au tétramino parent
<code>texture</code>	La texture du bloc. De type <code>SFML.Graphics.Texture</code>
<code>sprite</code>	Le sprite associé au bloc. De type <code>SFML.Graphics.Sprite</code> .

Méthodes	Paramètres	Commentaires
<code>TetrominoLittleBlock</code>	Le décalage par rapport au coin supérieur gauche et la couleur du bloc	Constructeur de <code>TetrominoLittleBlock</code> .
<code>Draw</code>	L'objet de type <code>RenderWindow</code> associé à la fenêtre de rendu, la position du tétramino parent	Dessine le petit bloc à l'écran. Voir les exercices en classe pour la manière de procéder.
<code>GetParentColumnOffset</code>		Retourne le décalage en colonne par rapport au tétramino parent

GetParentRowOffset		Retourne le décalage en ligne par rapport au tétramino parent
--------------------	--	---

### Tableau logique de jeu : quelques exemples

