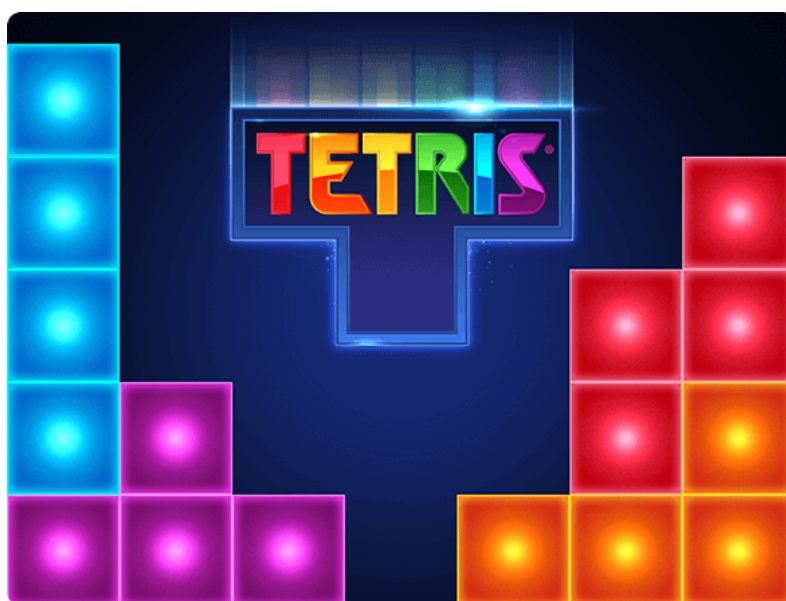


Polytech Clermont

Ingénierie Mathématiques et Data Science

Manuel développeur

Tetris



Présenté par :

Corentin Prigent

Felix Baubriaud

Année 2022/2023

1. Partie analyse et conception du logiciel	3
1.1. Présentation du jeu	3
1.1.1 Histoire du jeu	3
1.1.2 Règle du jeu de base	4
1.2. Les cas d'utilisation	5
1.3. Analyse (les diagrammes de haut niveau)	6
1.3.1. Diagramme de séquence d'analyse	6
1.3.2. Diagramme de classe d'analyse	8
1.4. Conception (les diagrammes de bas niveau)	11
1.3.1. Diagramme de séquences technique	11
1.3.2. Diagramme de classe technique	18
2. Partie programmation du logiciel	21
2.1. L'architecture du logiciel	21
2.2. Présentation des écrans	22
2.2.1. Écran de départ	22
2.2.2. Écran avec tétramino en réserve	24
2.2.2. Écran de fin du jeu	25
Bibliographie	26

1. Partie analyse et conception du logiciel

1.1. Présentation du jeu

1.1.1 Histoire du jeu

Tetris est un jeu créé en 1984, par Alekseï Pajitnov, ingénieur en informatique soviétique travaillant au centre informatique de l'Académie des sciences de l'URSS. En cherchant à reproduire l'un de ses jeux favoris, le pentomino, qui consiste à empiler des formes, il imagine le jeu Tetris et le programme.[1]

Tetris baptisé par la combinaison de “tetra” qui veut dire 4 en grec comme le nombre de carrés par tétramino et de “Tennis” qui était à l'époque le jeu vidéo favori de Alekseï Pajitnov. Le jeu eut un réel succès à l'international. Il est encore joué aujourd'hui par de nombreux joueurs.[2]


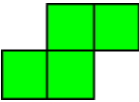
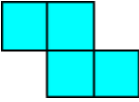


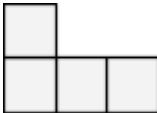
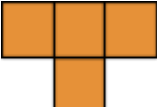
Ce jeu est connu pour être l'un des pionniers du jeu vidéo. Elle est encore aujourd'hui la deuxième série de jeux vidéo la plus vendue avec 494 millions de ventes derrière la licence Mario.[3]



figure 1: Alexey Pajitnov [4]

1.1.2 Règles du jeu de base

Le joueur manipule un tétramino à la fois parmi 7 formes différentes:

- Le I: 
- Le S: 
- Le Z: 
- Le O: 
- Le L: 
- Le J: 
- Le T: 

Il faut créer avec ces formes des lignes horizontales afin de faire un maximum de points. Le joueur peut pivoter et translater les pièces afin de les emboîter correctement. Plus le joueur supprime de lignes en un coup, plus il gagne de points. Supprimer 4 lignes en un coup s'appelle un tetrïs. Le joueur gagne des niveaux à chaque fois qu'il supprime un nombre donné de lignes. À chaque niveau la vitesse de descente des pièces augmente, rendant le jeu de plus en plus difficile. La version la plus classique et la plus connue est sur NES. Le nombre de niveaux s'élève à 29 sur cette version. Dans le jeu de base, le score augmente à chaque tétramino posé.

Quand les tuiles pleines arrivent jusqu'au niveau d'apparition du prochain tétramino le jeu s'arrête, c'est perdu. Une partie peut donc être jouée indéfiniment.

1.2. Les cas d'utilisation

Voici notre diagramme de cas d'utilisation :

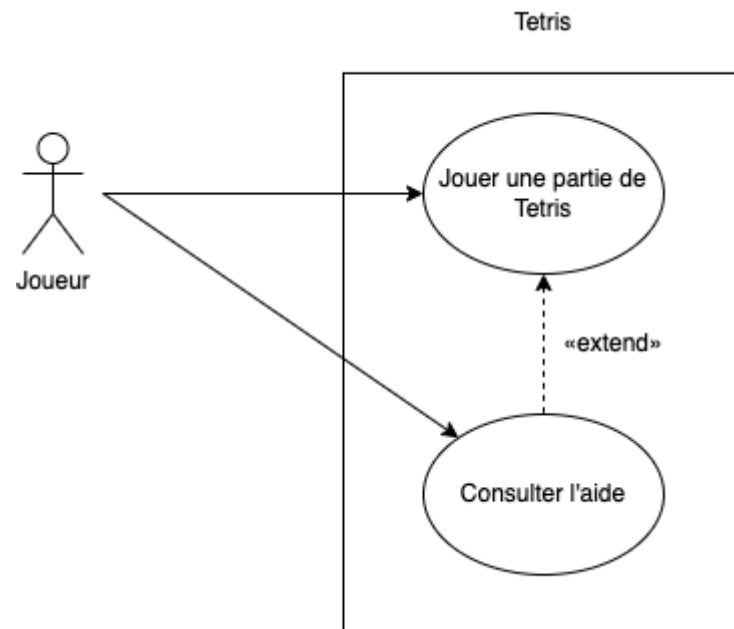


figure 2 : Diagramme de cas d'utilisation

Le principal cas d'utilisation du jeu Tetris est de jouer à une partie de Tetris et éventuellement d'accéder au manuel utilisateur. Les configurations restent les mêmes.

1.3. Analyse (les diagrammes de haut niveau)

1.3.1. Diagramme de séquence d'analyse

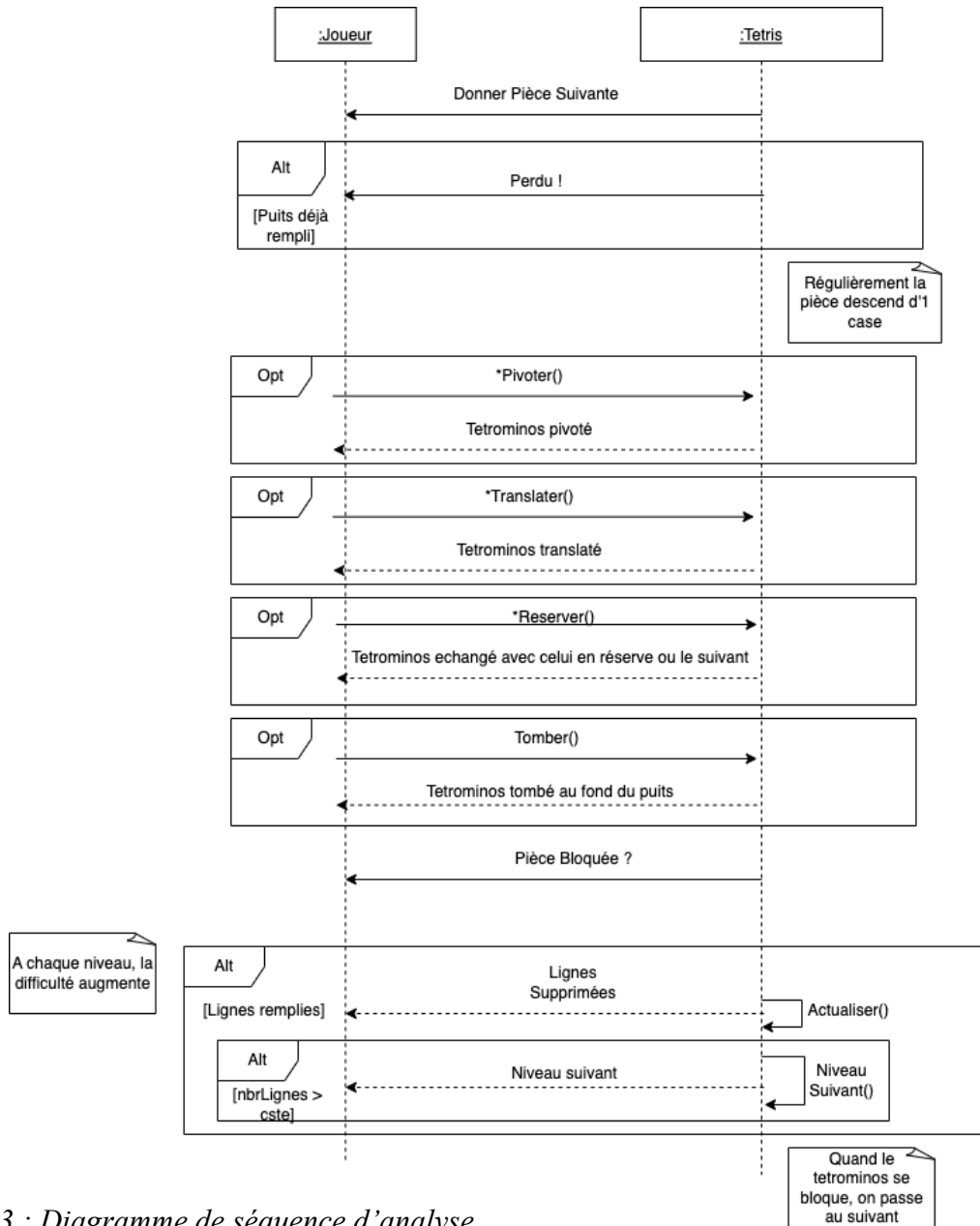


figure 3 : Diagramme de séquence d'analyse

Tant que le puits n'est pas rempli jusqu'en haut, le jeu fournit des tétraminoes un par un. Pendant que la pièce tombe, le joueur peut faire le choix de pivoter le tétramino, de le déplacer de gauche à droite ou de l'échanger avec le tétramino en réserve ou le tétramino suivant jusqu'à ce qu'il arrive en bas du puits et se bloque. Il peut aussi décider de faire tomber directement le tétramino au fond du puits. Une fois posé, le jeu va vérifier si le tétramino remplit une ligne. Si c'est le cas, la ligne est supprimée. Le score du joueur n'augmente que lorsque les lignes sont supprimées, et on passe au niveau suivant à partir d'un certain nombre de lignes complétées.

1.3.2. Diagramme de classe d'analyse

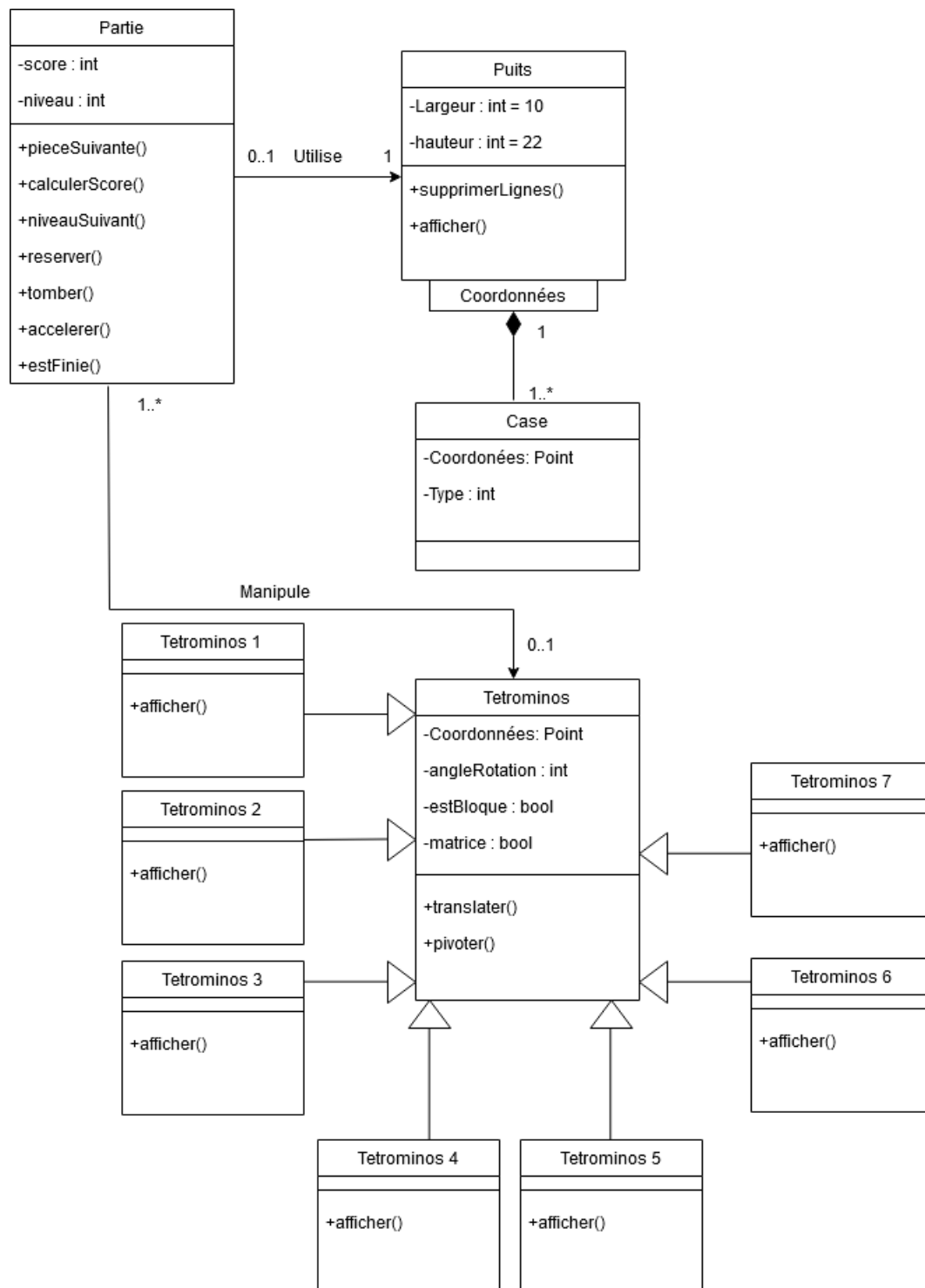


figure 4 : Diagramme de classe d'analyse

Classe Tetrominos :

Premièrement, il y a une classe par type de tétrominos (Tetrominos 1, ..., Tetrominos 7) (sans compter leurs rotations), et chacune d'entre elles est un héritage d'une classe Tetrominos plus générique. Cela permet une redéfinition de chaque méthode des tétrominos, qui seraient alors plus en accord avec leur forme physique.

Les attributs en question sont :

- leurs coordonnées ;
- leur angle de rotation ;
- un booléen (*estBloque*), qui permet de savoir, en tout temps, l'état de la pièce (et donc de savoir s'il est encore possible de la bouger ou non) ;
- une matrice de booléen où chaque coefficient vaut *true* si la case est pleine, *false* sinon, de taille différente selon le type de tetromino.

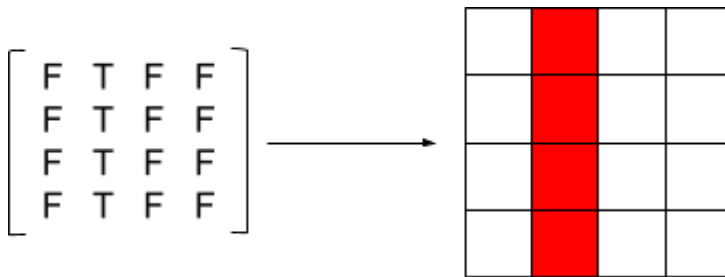


Figure 5 : matrice et affichage des tétrominos 1

De même, certaines méthodes sont communes à presque toutes les classes Tetrominos. Il s'agit de *translater()* et *pivoter()*, qui permettent respectivement, et comme leur nom l'indiquent, de translater (horizontalement ou vers le bas) les tétrominos et de les faire pivoter. La seule exception est la méthode *pivoter()* du tétromino carré car elle ne change pas sa forme. En revanche, la méthode *afficher()* est unique pour chaque tétromino car la couleur n'est pas la même.

Classe Puits + Case :

La classe Puits représente le puits contenant les tétrominos. Cette classe est définie par une largeur et une hauteur qui sont toujours les mêmes à toutes les parties. Comme une matrice de tétrominos peut avoir au plus 2 colonnes ou lignes vides d'un côté (voir tétromino 1 au-dessus), il est possible que le tétromino soit bien dans le puits mais que sa matrice dépasse du puits. On doit donc ajouter à la grille du puits deux colonnes vides de chaque côté et 2 lignes vides en bas de la grille. Ainsi, la grille a 10 colonnes + 4 transparentes et 22 lignes + 2 transparentes.

Ce Puits est rempli de Cases, représentées par des coordonnées et un type. Ce type, qui est un entier compris entre 0 et 8, permet à la fois de distinguer les types de tétrominos qui ont été bloqués par des couleurs différentes, et d'identifier s'il s'agit d'une case pleine ou vide.

- Le type 0 signifie que la case est vide ;
- Les types 1 à 7 représentent des cases pleines dont la couleur est celle du tétromino associé ;
- Le type 8 est réservé aux cases qu'on affiche pas mais qui sont considérées comme pleines. Elles se trouvent de chaque côté et en bas du puits.

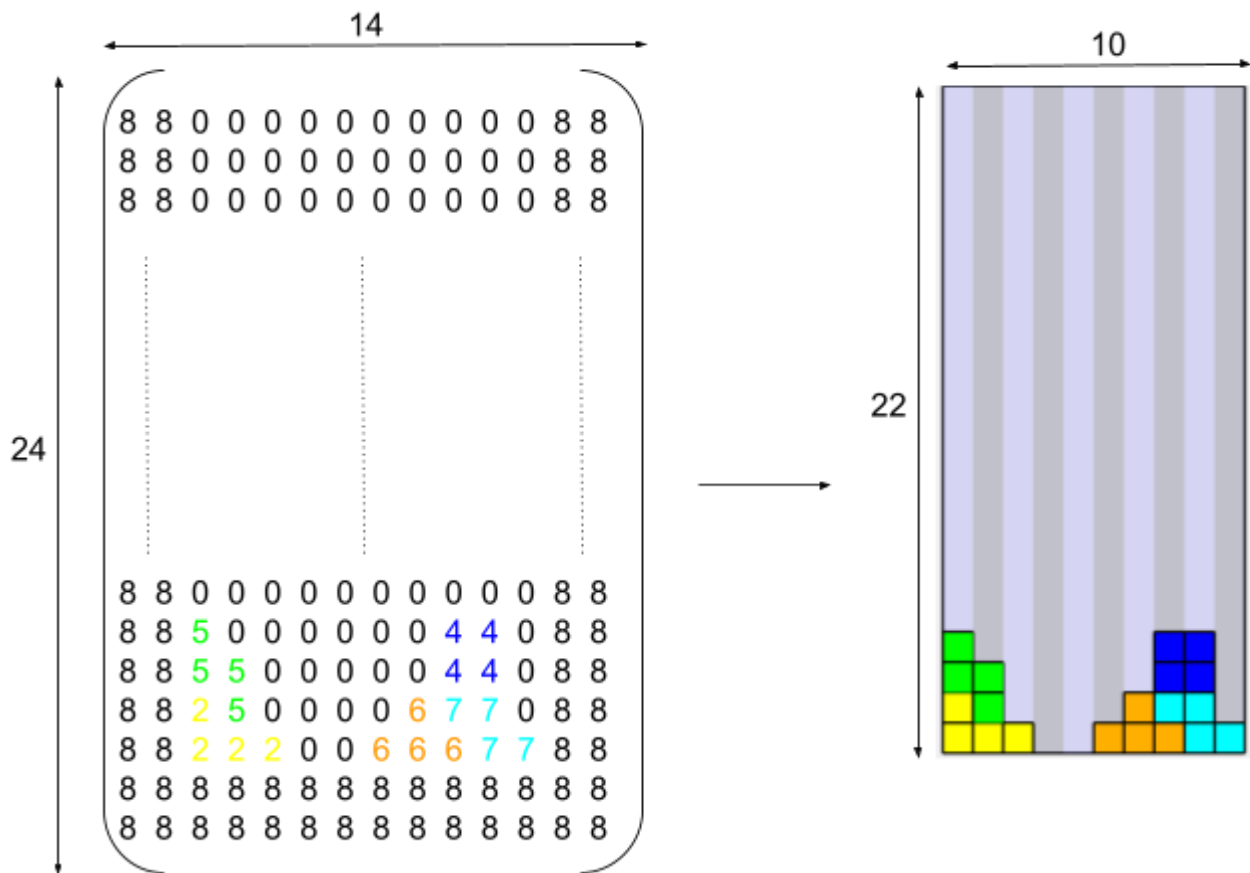


Figure 6 : matrice et affichage du puits

Classe Partie :

Une partie utilise un puits pour pouvoir être lancée car tout se passe à l'intérieur. Il y a une classe Partie définie par le score du joueur ainsi que le niveau, les deux augmentant pendant toute la durée de la partie (le niveau jusqu'à 10). Cette classe est reliée à la classe Tetrominos car la partie manipule les tétramino tout le long. Elle contient les méthodes :

- *pieceSuivante()*, permettant de passer d'un tétramino qui vient d'être posé au suivant ;
- *calculerScore()* et *niveauSuivant()*, qu'on appelle lorsqu'on complète des lignes pour le décompte des points ;
- *reserver()* pour mettre un tétramino en réserve et récupérer le suivant ou le tétramino qui est déjà en réserve s'il y en a un ;
- *tomber()*, permettant de faire tomber le tétramino directement au fond du puits ;
- *accellerer()*, qui fait descendre le tétramino d'une case.

1.4. Conception (les diagrammes de bas niveau)

1.3.1. Diagramme de séquences technique

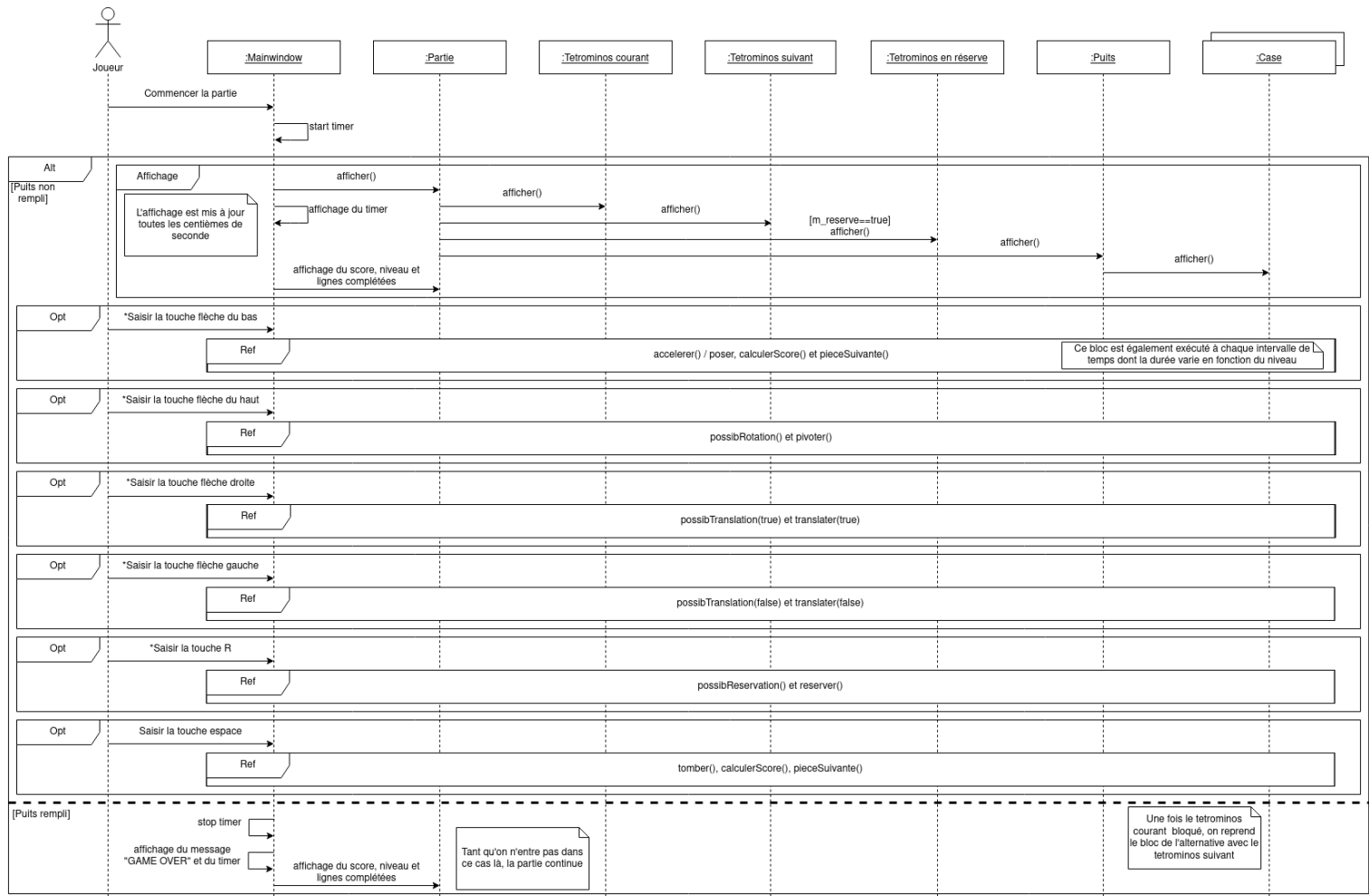


figure 7: Diagramme de séquence technique vu d'ensemble

Au début de la partie, le constructeur de MainWindow est appelé. Ce dernier appelle alors celui de Partie, qui appelle à son tour tous les autres. Dans MainWindow, on met en place un timer et on utilise sa fonction *start(10)* ce qui permettra d'appeler la méthode *update()* toutes les 10 millisecondes, c'est-à-dire toutes les centièmes de seconde.

Dans cette méthode *update()*, on effectue 3 actions distinctes:

- On incrémente les secondes de 0,01. Si les secondes arrivent à 60, on incrémente de 1 les minutes et on remet à 0 le compteur des secondes.
- A chaque intervalle de temps dont la durée est donnée par l'attribut *m_tempsDescente* de la classe Partie, on vérifie si le tétramino peut encore descendre avec la fonction *descentePossible()*. Dans ce cas on descend le tétramino d'une case en appelant la méthode *accélérer()* de Partie. Sinon on le bloque en appelant la méthode *poser()*, puis on calcule le score (si des lignes ont été complétées) et le tétramino suivant devient le courant. Pour effectuer ces actions à intervalle régulier, on vérifie que le reste de la division euclidienne du temps total par *m_tempsDescente* est nul. Or, l'opérateur % utilisé ne fonctionne qu'avec des

entiers. C'est pourquoi *m_tempsDescente* est exprimé en centième de seconde et que les minutes et secondes sont converties en centième de seconde.

- On met à jour l'affichage en appelant la méthode *paintEvent(e)* de *MainWindow*. Dans le cas où le puits n'est pas rempli, on affiche la partie (puits, tétramino courant, suivant et éventuellement celui en réserve) ainsi que le timer arrondi aux centièmes de seconde, le score, le niveau et le nombre de lignes supprimées à droite du puits. Si le puits est rempli, la partie est terminée donc on affiche uniquement le message "GAME OVER" et, au centre cette fois, le timer arrêté, le score, le niveau et le nombre de lignes supprimées.

Lors de la chute du tétramino, le joueur peut agir de différentes façons :

- En plus d'être appelé automatiquement toutes les *m_tempsDescente* centièmes de seconde, les méthodes *accelerer()* ou *poser()*, *calculerScore()* et *pieceSuivante()* de la classe *Partie* peuvent être appelées par l'utilisateur lorsqu'il appuie sur la flèche du bas du clavier ;
- L'utilisateur peut aussi pivoter le tétramino courant avec la flèche du haut à condition que la rotation soit possible ;
- Il peut appuyer sur les flèches droites ou gauches pour translater le tétramino à condition que la translation droite ou gauche respectivement soit possible ;
- Il peut également choisir de réserver si possible le tétramino en appuyant sur la touche R du clavier ;
- Enfin, il peut faire tomber directement le tétramino au fond du puits en appuyant sur la barre d'espace. Cette action appelle également les fonctions *calculerScore()* et *pieceSuivante()*.

Une fois que le tétramino courant est bloqué, on reprend avec le tétramino suivant, et ainsi de suite jusqu'à ce que le puits soit rempli.

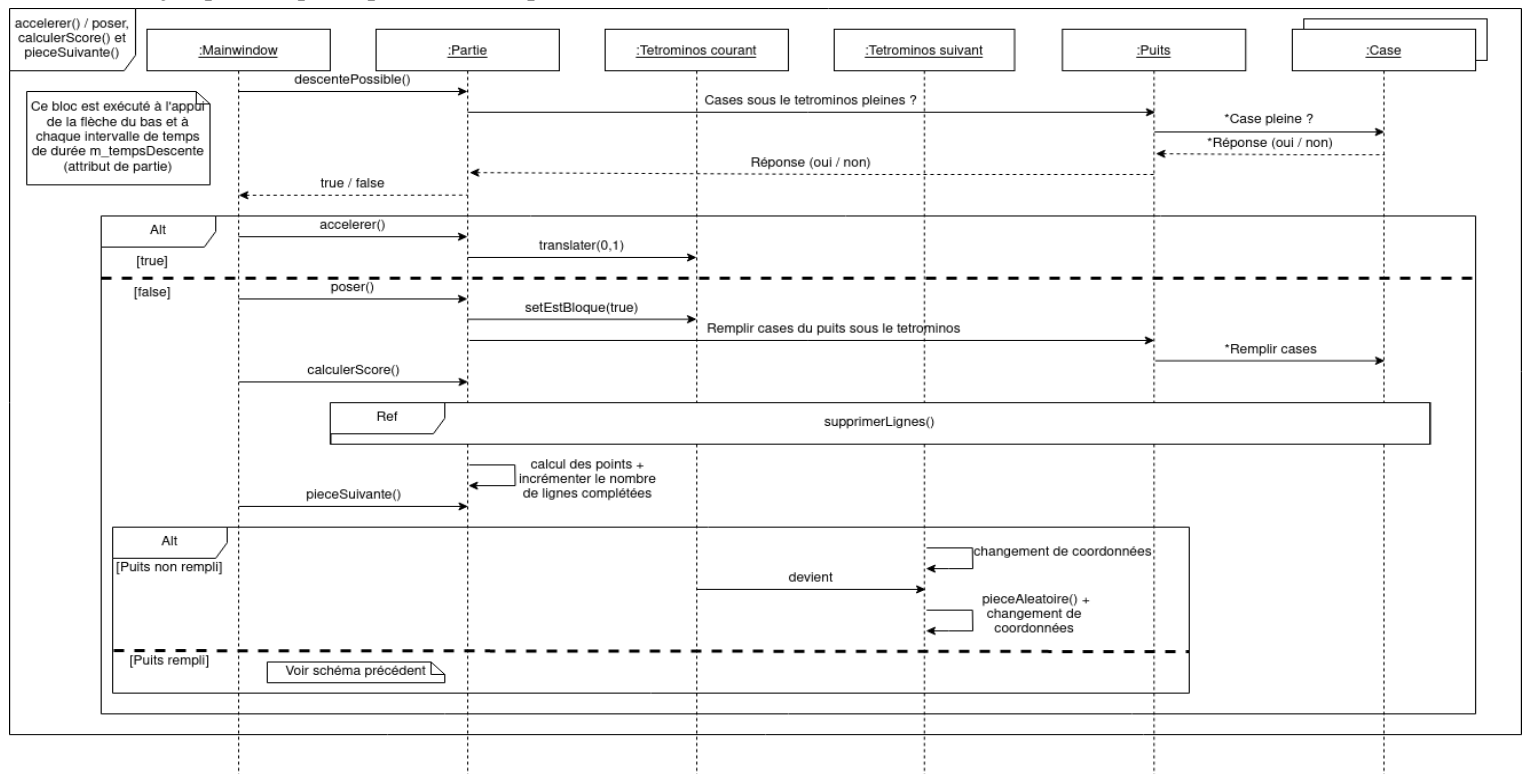


figure 8: Partie *accelerer()/poser*, *calculerScore()* et *pieceSuivante()* du Diagramme de Sequence Technique

A chaque fois que le tétramino courant doit descendre d'une case, on vérifie d'abord que le déplacement est possible. Pour cela, on vérifie pour chaque case pleine du tétramino courant si sa translation vers le bas mène sur une case vide du puits ou pas.

S'il n'y a rien qui bloque, on effectue le déplacement en appelant la méthode *accélérer()* de Partie qui à son tour appelle la méthode *translater(0,1)* du tétramino courant. Cette dernière prend en paramètre respectivement le nombre de cases à droite et en bas parcourues par le tétramino. Ici, il suffit d'incrémenter la coordonnée y du tétramino dans le puits de 1.

Dans le cas où le déplacement n'est pas possible, on effectue 3 actions :

- En appelant la méthode *poser()* de la classe Partie, on met l'attribut *m_estBloque* du tétramino courant à *true* et on remplit les cases du puits qui sont placées sous les cases pleines du tétramino, en changeant leur type par celui qui correspond au tétramino (pour la couleur).
- *calculerScore()* appelle la fonction *supprimerLigne()* de la classe Puits, et selon le nombre de lignes supprimées, on incrémente le nombre de lignes complétées et on incrémente le score : $40 \cdot (n+1)$ pour une ligne supprimée, $100 \cdot (n+1)$ pour deux, $300 \cdot (n+1)$ pour trois et $1200 \cdot (n+1)$ pour 4 avec $n = m_niveau$ le niveau. Dès qu'on dépasse les 5 lignes supprimées depuis le dernier niveau passé (on vérifie cela en regardant les restes de la division euclidienne du nombre de ligne complétées par 5 selon le nombre de lignes supprimées), on appelle également la fonction *niveauSuivant()* qui va diminuer le temps de chute du tétramino de 10 centièmes de seconde. Sachant qu'on démarre avec *m_tempsDescente* = 100 centièmes de secondes, le niveau maximum est 9.
- Si le puits n'est pas rempli (on vérifie avec la fonction *estPlein()* de Puits), dans la fonction *pieceSuivante()* on transporte le tétramino suivant en haut du puits et celui-ci devient alors le tétramino courant. L'ancien est supprimé puisqu'il est maintenant inclus dans le puits. On génère un nouveau tétramino suivant en appelant la méthode *pieceAleatoire()* et on le remet à sa place. En revanche, si le puits est rempli à l'insertion de la dernière pièce, on met l'attribut *m_finie* de Partie à *true* ce qui permet d'afficher directement l'écran de fin.

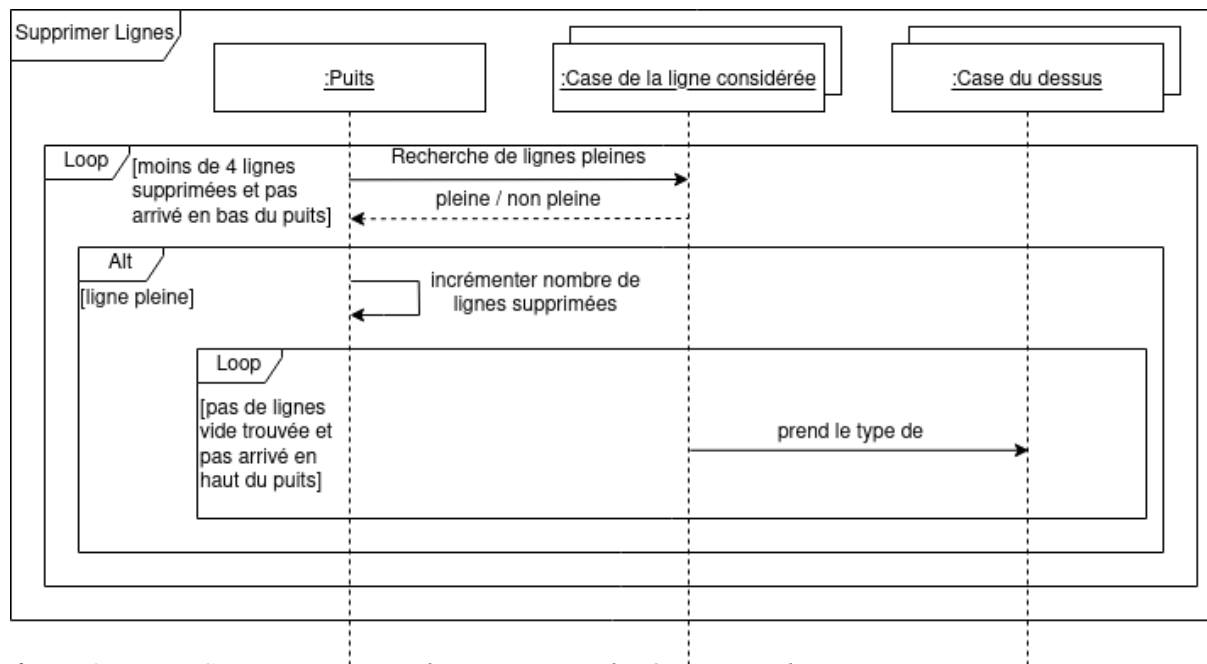


figure 9: Partie Supprimer Lignes du Diagramme de séquence techniques

Notre fonction *supprimerLignes()* fait à la fois office de procédure et de fonction. Voici les actions qui sont effectuées dans cette méthode :

- on cherche dans le puits des lignes pleines à partir du haut ;
- pour chaque ligne pleine i , on parcourt les lignes de la grille de manière ascendante à partir de la ligne i jusqu'en haut ou jusqu'à ce qu'on trouve une ligne complètement vide, et chaque case de cette ligne prend le type de la case d'au-dessus ;
- on arrête de chercher des lignes complétées lorsqu'on en a déjà trouvé 4 (c'est le maximum possible) ou si on arrive en bas du puits ;
- il ne faut pas oublier qu'il y a dans la grille 2 lignes invisibles en bas du puits et 2 colonnes invisibles à gauche et à droite du puits ;
- à la fin, on retourne le nombre de lignes qui ont été supprimées afin de pouvoir l'exploiter en dehors de la fonction.

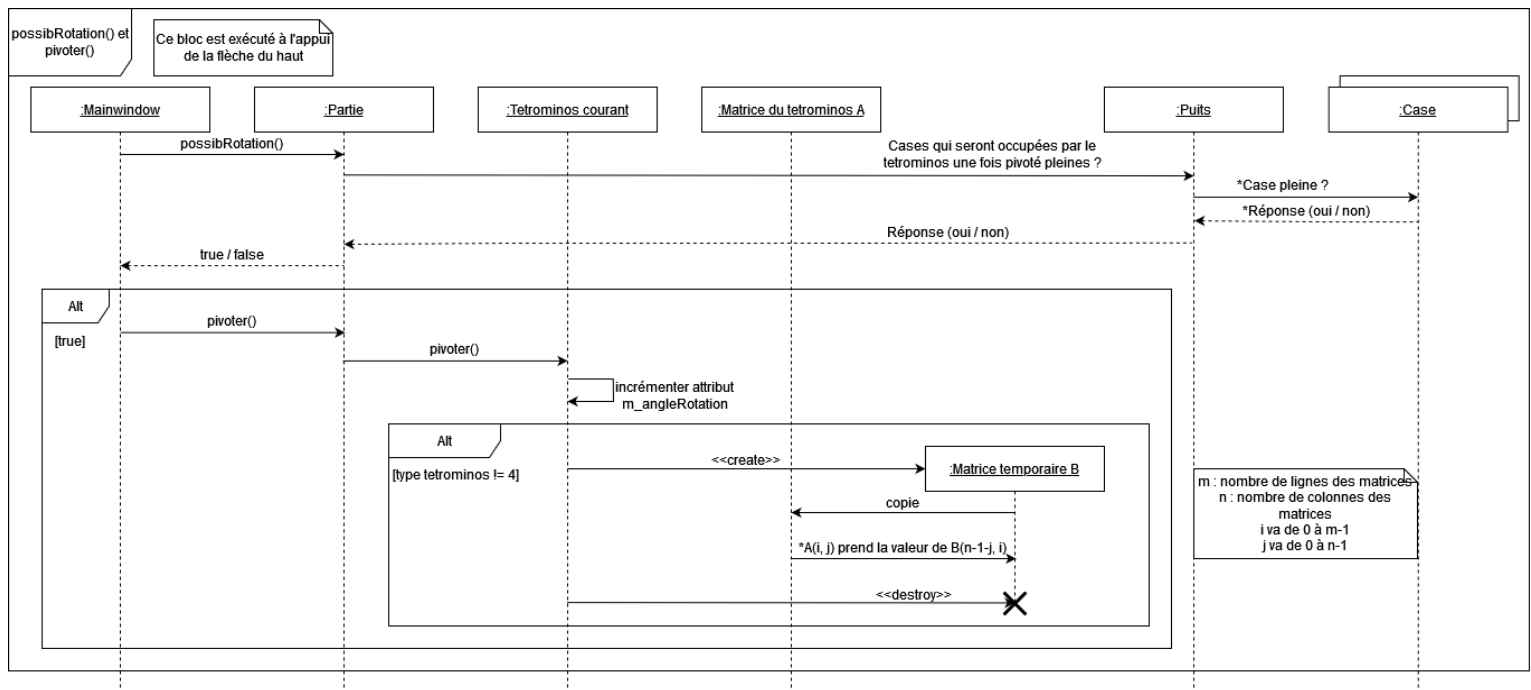


figure 10: Partie *possibRotation()* et *pivoter()* du diagramme de séquence technique

Avant de pivoter la pièce, il faut d'abord vérifier que sa rotation est possible dans le puits. Pour cela, on parcourt les cases du tétramino. Pour celles qui sont pleines, on regarde sur quelle case du puits mènerait leur potentielle rotation. Si toutes ces cases sont vides, on peut effectuer la rotation. Sinon, l'appui de la touche flèche du haut n'appelle pas la méthode *pivoter()*.

La fonction *pivoter()* de *Partie* appelle celle du tétramino courant. S'il s'agit du tétramino carré (*Tetrominos4*), on change uniquement l'attribut *m_angleRotation* puisque la rotation ne change rien. De plus, la méthode de pivot utilisée pose problème si la matrice du tétramino n'est pas carrée et c'est le cas pour *Tetrominos4*. Pour tous les autres tétramino, on doit pour chaque coefficient $a_{i,j}$ de leur matrice le remplacer par le coefficient $a_{n-1-j,i}$ avec n le nombre de colonnes de la matrice. Mais pour cela il est nécessaire de faire une copie de la matrice d'origine afin de ne pas remplacer certains coefficients par des coefficients déjà modifiés. On alloue donc une matrice temporaire et on la remplit case par case avec les mêmes coefficients que la matrice du tétramino courant, puis on fait les modifications de la matrice du tétramino avant de désallouer la matrice temporaire. En C++, les indices commencent à 0 donc un coefficient placé à la dernière colonne a pour deuxième coordonnée $n-1$.

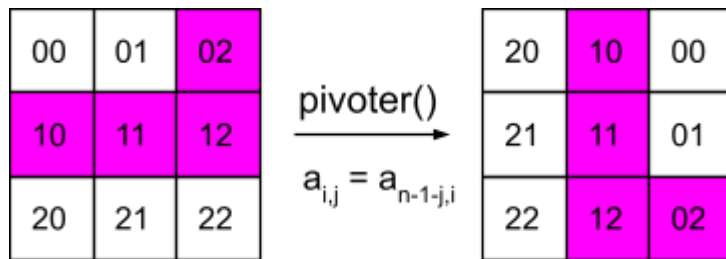


Figure 11 : rotation du tetromino 3

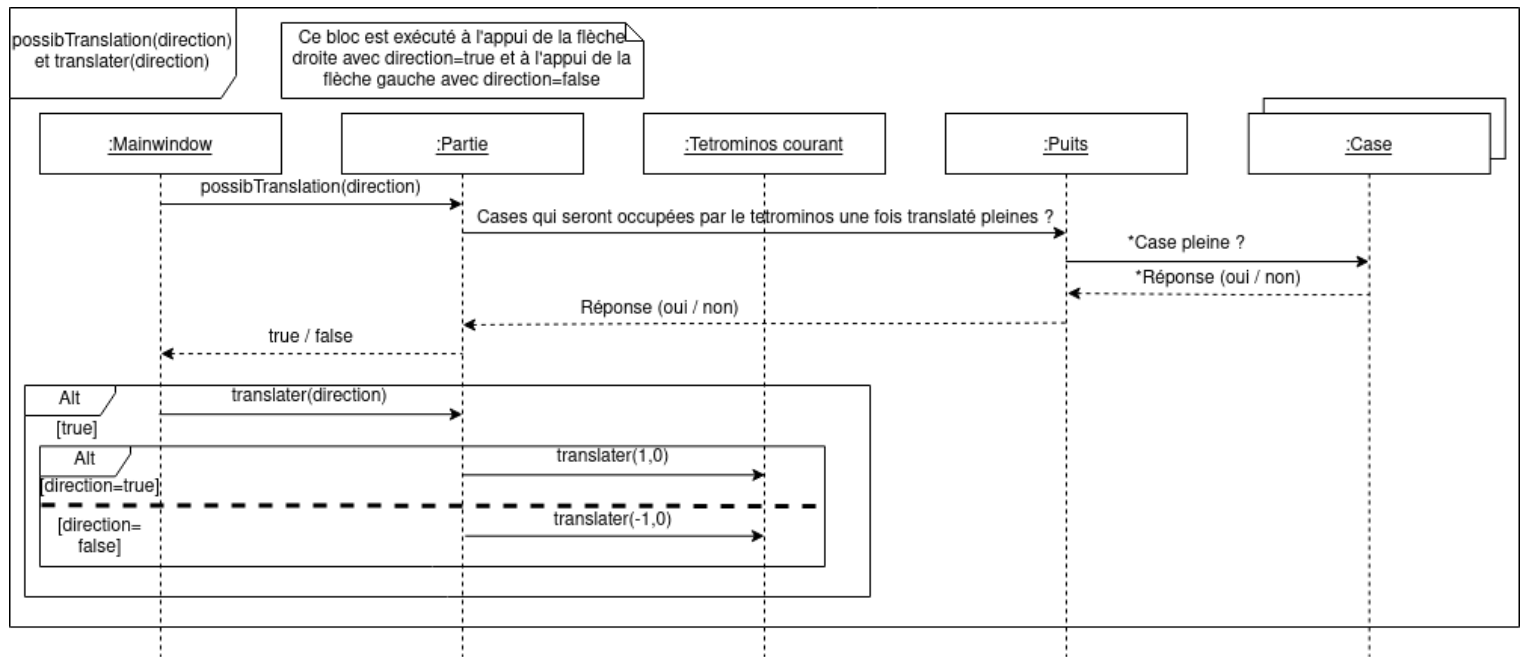


figure 12: Partie possibTranslation(...) et translater(...) du diagramme de séquence technique

Comme pour la rotation, on vérifie que les cases du puits situées à une case (à droite ou à gauche selon la direction) de chaque bloc composant le tetromino sont vides. Dans le cas où on trouve une case pleine, l'appui de la touche correspondante n'appelle pas la méthode *translater(direction)* de Puits. Sinon, cette dernière appelle *translater(x,0)* de Tetromino avec $x=1$ pour déplacer le tétramino d'une case vers la droite et $x=-1$ pour le déplacer d'une case vers la gauche.

Une fois les tétramino échangés, on doit les remettre à leur place d'origine dans le puits car les attributs correspondant aux coordonnées des tétramino ont aussi été copiés. Dans les deux cas, le tétramino en réserve a copié les coordonnées du tétramino courant, qui doit donc reprendre les coordonnées de celui-ci. Enfin, le tétramino, on change les coordonnées du tétramino en réserve de sorte qu'il soit à sa place à droite du puits.

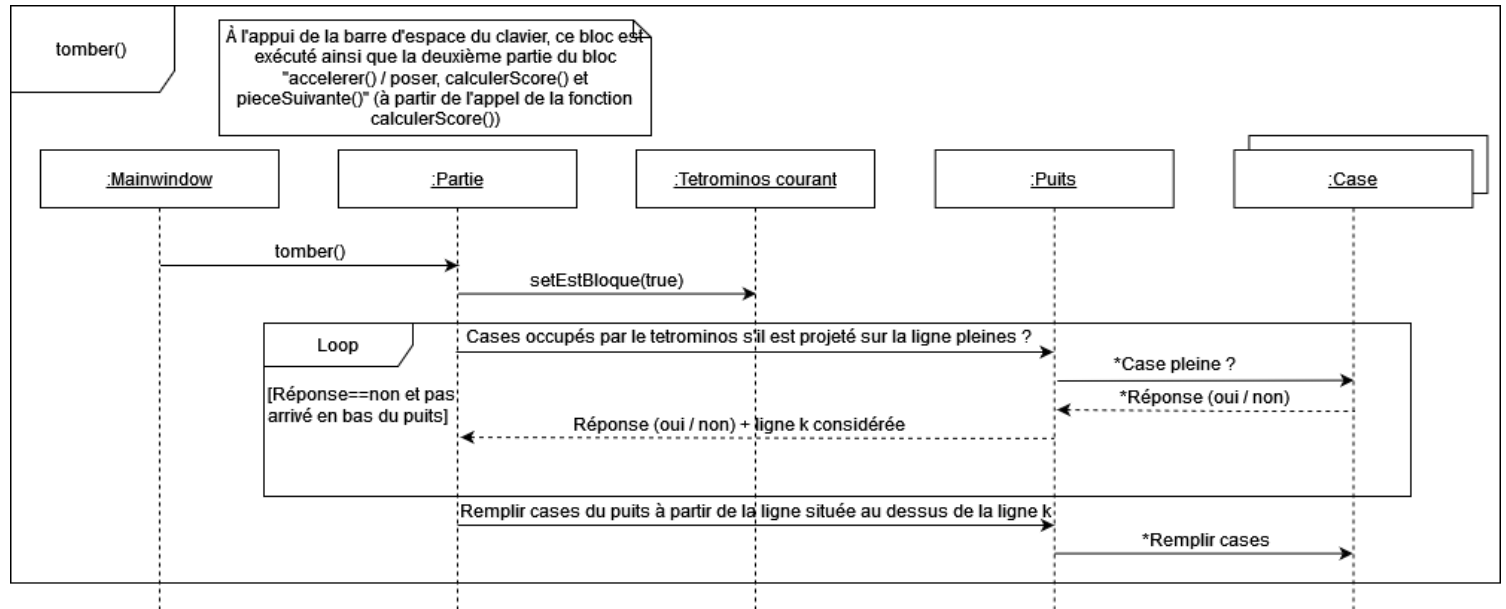


figure 14: Partie *tomber()* du diagramme de séquence technique

L'appui de la barre d'espace du clavier appelle tout d'abord la fonction *tomber()* de *Partie*. Pour cette méthode, il n'y a aucune vérification à effectuer : il est toujours possible de faire tomber un tétramino directement dans le puits. On commence par mettre l'attribut *m_bloque* de *tetromino* à *true* comme pour la méthode *poser()*. Ensuite on va rechercher dans le puits la ligne où va tomber le tétramino. Pour cela, on parcourt la grille du puits de haut en bas à partir de la ligne où est situé le tétramino (plus précisément la première ligne de la matrice du tetromino). Pour chaque ligne, on va alors vérifier si les cases du puits qui seront situées sous les cases pleines du tétramino une fois qu'on l'aura projeté sur la ligne considérée (de sorte que la première ligne de la matrice du tétramino soit sur la ligne considérée) sont vides. Si c'est le cas, on passe à la ligne suivante. Sinon, c'est que la projection n'est plus possible sur la ligne considérée. On projette donc le tétramino sur la ligne d'au-dessus : on remplit les cases du puits qui se trouve sous les cases pleines du tétramino une fois transposé sur cette ligne. Ensuite, les méthodes *calculerScore()* et *pieceSuivante()* sont appelées.

1.3.2. Diagramme de classe technique

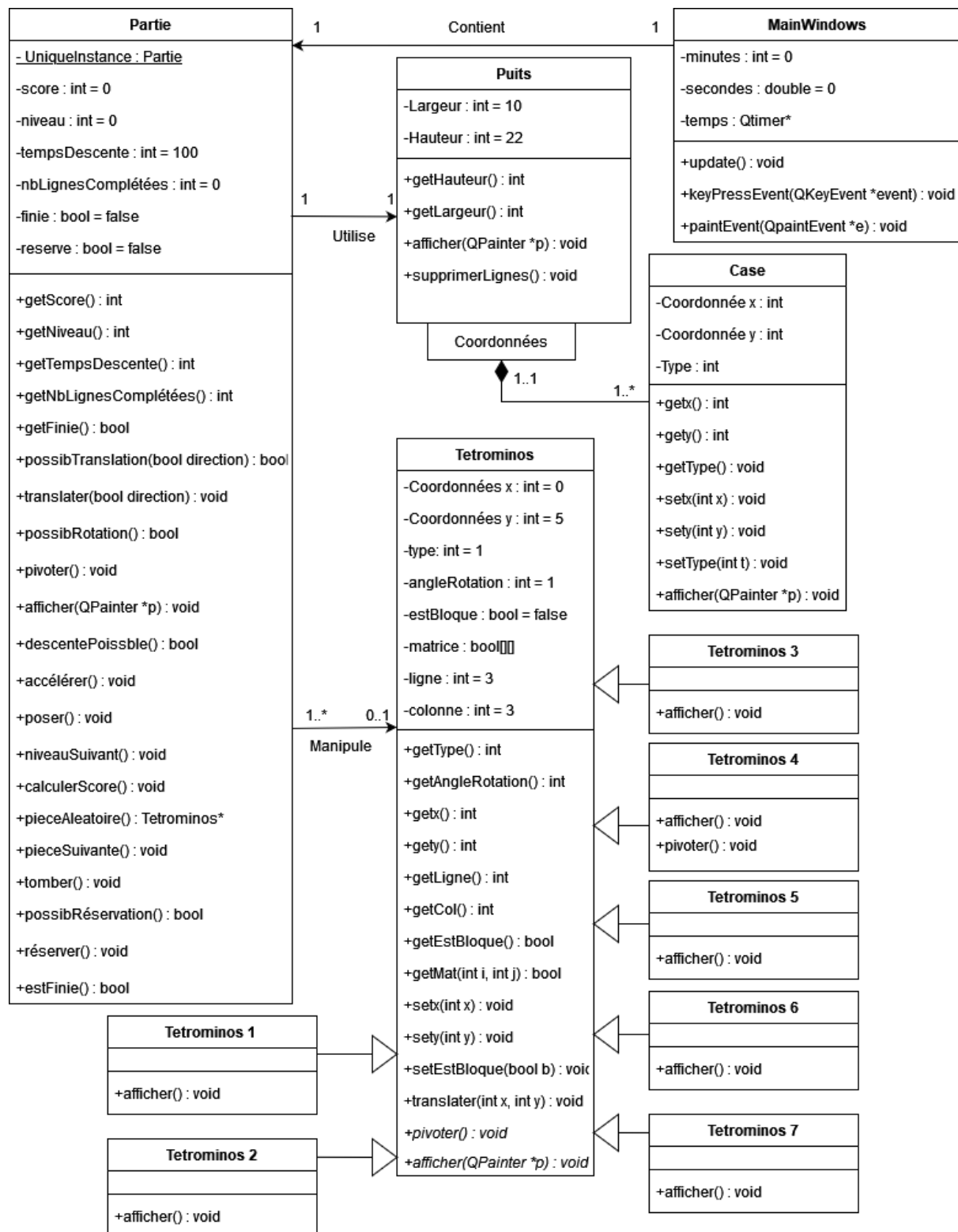


figure 15: Diagramme de classe technique

Notre tetris fonctionne avec des matrices, une pour le puits et une pour chacun des tétramino. Chaque coefficient de ces matrices représente une case, pleine ou vide pour les tétramino, et d'une couleur particulière pour la grille. On utilise ensuite ces matrices pour l'affichage : chaque coefficient représente alors un carré de côté T , qui est une constante.

Notre Tetris est composé de 12 classes :

- Les 7 types de Tetraminos, chacun hérite des attributs et méthodes de la classe mère Tetraminos. Les coordonnées x et y , initialisées respectivement à 0 et 5 pour tous les tétramino, sont les coordonnées dans le puits de la case située en haut à gauche de la matrice du tetramino. Les coefficient de cette matrice sont des booléens : *true* si la case est pleine, *false* sinon. Les attributs ligne et colonne représentent le nombre de lignes et le nombre de colonnes de la matrice du tétramino, qui sont égaux (3 ou 4) sauf pour le tétramino carré. Ils sont initialisés par défaut à 3 et sont modifiés si nécessaire dans les constructeurs des sous-classes. L'attribut *angleRotation* vaut 1 si le tetramino n'a pas été pivoté, 2 s'il a été pivoté à 90° , 3 s'il a été pivoté à 180° , et 4 s'il a été pivoté à 270° . Le type est un entier entre 1 et 7 permettant de différencier chaque type de tétramino.

Il y a des accesseurs pour tous les attributs et des mutateurs pour les coordonnées et *estBloque*. La méthode pour translater le tétramino prend en paramètre le nombre de case que le tétramino doit parcourir à droite et en bas, avec la possibilité de mettre des nombres négatifs pour translater le tétramino à gauche (ou en haut mais on ne le fait pas dans ce jeu). La méthode *pivoter()* est virtuelle car elle est redéfinie pour le tétramino carré, et *afficher()* est une méthode abstraite, faisant de Tetraminos une classe abstraite, car elle est redéfinie pour tous les tétramino.

Remarque : si matrice on note la matrice $(a_{i,j})$ la matrice du tétramino, les j sont les colonnes et correspondent à l'axe des abscisses tandis que les i sont les lignes et correspondent à l'axe des ordonnées. Il faut donc faire attention lors de l'affichage dans le puits des tétramino.

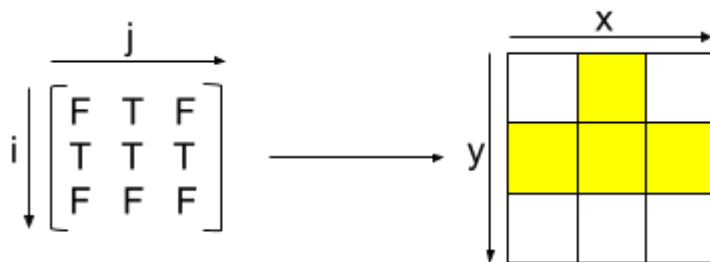


Figure 16 : matrice et affichage du tetromino 6

- Les cases du puits, ayant comme méthodes les accesseurs et mutateurs de chaque attribut (x , y et le type de la case) ainsi qu'une méthode *afficher()* qui dessine un carré de taille T à partir des coordonnées de la case, d'une couleur différente selon le type. Pour le type 0 (case pleine) on affiche des cases de deux couleurs distinctes pour les colonnes paires et impaires pour mieux visualiser où le tetramino se dirige, tandis que pour le type 8 (case invisible) on n'affiche rien. Pour tous les autres types on ajoute une bordure pour bien voir que la case est pleine.
- Le puits, dont la méthode *afficher()* appelle celle de chaque case en plus de dessiner le contour du puits.

- La classe *Partie*, contenant comme attribut en plus du score et du niveau :
 - le temps entre chaque appel de la fonction *accellerer()* en centième de seconde qu'on nomme *tempsDescente*. Cet attribut est initialisé à 100 et baisse de 10 dès qu'on passe un niveau, et de 5 pour le passage des deux derniers niveaux (9 et 10).
 - le nombre de lignes complétées au cours de la partie. Le joueur passe un niveau toutes les 5 lignes complétées.
 - l'attribut *finie* qui permet de savoir si la partie est terminée ou en cours.
 - l'attribut *reserve* qui permet de savoir si un tétramino est en réserve.

En ce qui concerne les méthodes, la classe *Partie* contient les accesseurs des attributs qu'on affiche dans la fenêtre du jeu (*MainWindows*). Étant donné que cette dernière contient uniquement une instance de la classe *Partie*, on doit pouvoir accéder aux méthodes nécessaires de toutes les classes dans *Partie*. C'est pourquoi elle contient les méthodes *pivoter()* qui appelle simplement la méthode *pivoter()* de *Tetrominos*, et *translater(direction)* qui appelle, selon la direction, *translater(1,0)* ou *translater(-1,0)* de *Tetrominos*. Quant à la méthode *accellerer()*, elle appelle la méthode *translater(0,1)* de *Tetrominos*. Pour certaines méthodes, il y a des pré-conditions à respecter :

- *possibTranslation(direction)* pour *translater(direction)* ;
- *possibRotation()* pour *pivoter()* ;
- *descentePossible()* pour *accellerer()* ;
- *possibReservation()* pour *reserver()* ;
- *!descentePossible()* pour *poser()*, *calculerScore()* et *pieceSuivante()*.

La méthode *pieceAleatoire()* renvoie un tétramino aléatoire parmi les 7. Pour cela on génère un entier complètement aléatoire à l'aide de la fonction *rand()* puis on regarde son reste dans la division euclidienne par 7. Selon le chiffre obtenu, on alloue un type différent de tétramino. La méthode *estFinie()* vérifie pour chaque case pleine du tétramino si elle est située sur une case pleine du puits. Cela ne peut arriver que lorsque le tétramino apparaît en haut du puits, et signifie alors que le puits est plein. Si c'est le cas pour une ou plusieurs case du tétramino, on met l'attribut *finie* à *true* ce qui affiche l'écran de fin.

- La classe *MainWindow* qui correspond à la fenêtre principale. Dans le constructeur, on alloue un timer qui se met à jour toutes les 10 millisecondes avec la méthode *start(10)*. La méthode *update()* est ainsi appelée toutes les centièmes de seconde. On met pour le type des secondes double pour éviter les erreurs d'arrondi, car on les incrémente de 0.01 toutes les centièmes de secondes. La méthode *keyPressEvent(*event)* permet d'effectuer des actions à l'appui des touches du clavier présentes dans le switch.

2. Partie programmation du logiciel

2.1. L'architecture du logiciel

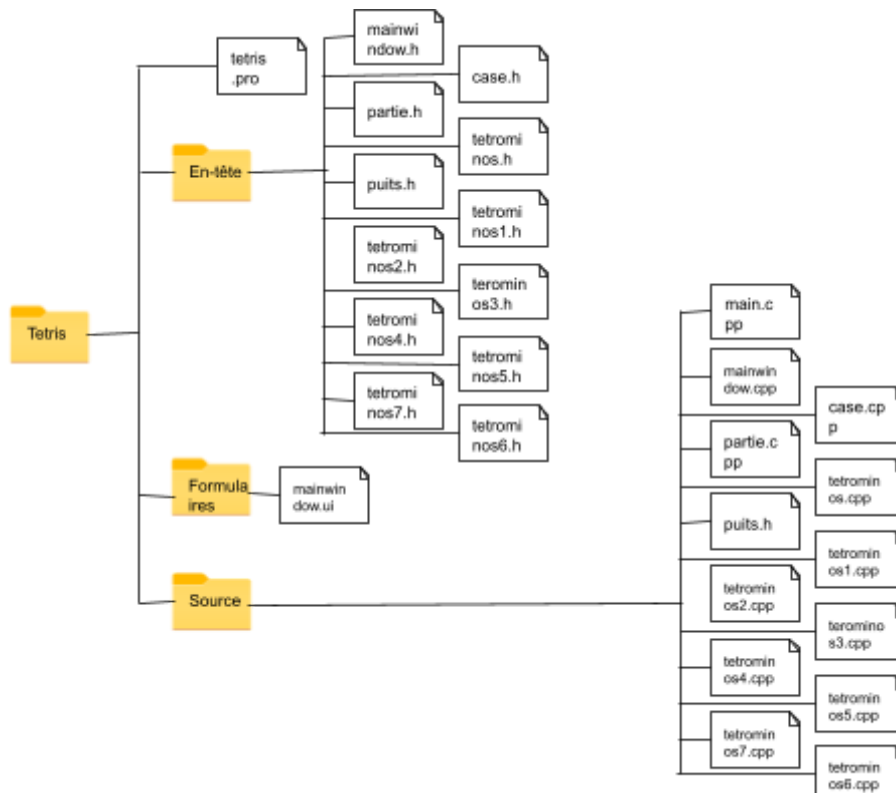


figure 17: Schéma de l'arborescence des fichiers et dossiers du projet

Ci-dessus le schéma de l'arborescence des fichiers et dossiers du projet lorsque nous l'ouvrons avec Qt. Dans En-tête les fichiers .h et dans Source les fichiers .cpp.

Les classes les plus importantes dans ce projet sont :

- MainWindow car sans cette classe nous ne pouvons pas ouvrir la fenêtre et il n'y a pas d'affichage ;
- Partie car elle permet de gérer toutes les règles du jeu ;
- Tetrominos car elle permet de générer les tetrominos.

Les méthodes les plus importantes sont :

- *possibTranslation()* et *possibRotation()* car elles permettent aux tetrominos de ne pas sortir du puits ;
- *pivoter()* et *translater()* car sans ces méthodes nous ne pouvons pas empiler correctement les tetrominos ;

- *poser()* car sans elles nous ne passons jamais au tétramino suivant ;
- *supprimerLignes()* car elle permet de vider le puits ;
- *estFinie()* car elle permet de finir la partie quand le tétramino suivant apparaît.

2.2. Présentation des écrans

2.2.1. Écran de départ

Au début de la partie, les tétramino courants et suivants sont générés aléatoirement. Toutes les caractéristiques de la partie sont initialisées à 0 (Niveau, Score et Nombre de lignes complétées).

Contrôles utilisés :

- flèche du haut pour pivoter le tétramino, à condition que ce soit possible ;
- flèche droite pour translater le tétramino à droite, à condition que ce soit possible ;
- flèche gauche pour translater le tétramino à gauche, à condition que ce soit possible ;
- flèche du bas pour accélérer le tétramino et éventuellement le poser ;
- barre d'espace pour faire tomber le tétramino dans le puits ;
- touche R pour mettre le tétramino courant en réserve et l'échanger avec le tétramino suivant.

Ces contrôles peuvent être utilisés autant de fois que le souhaite l'utilisateur.

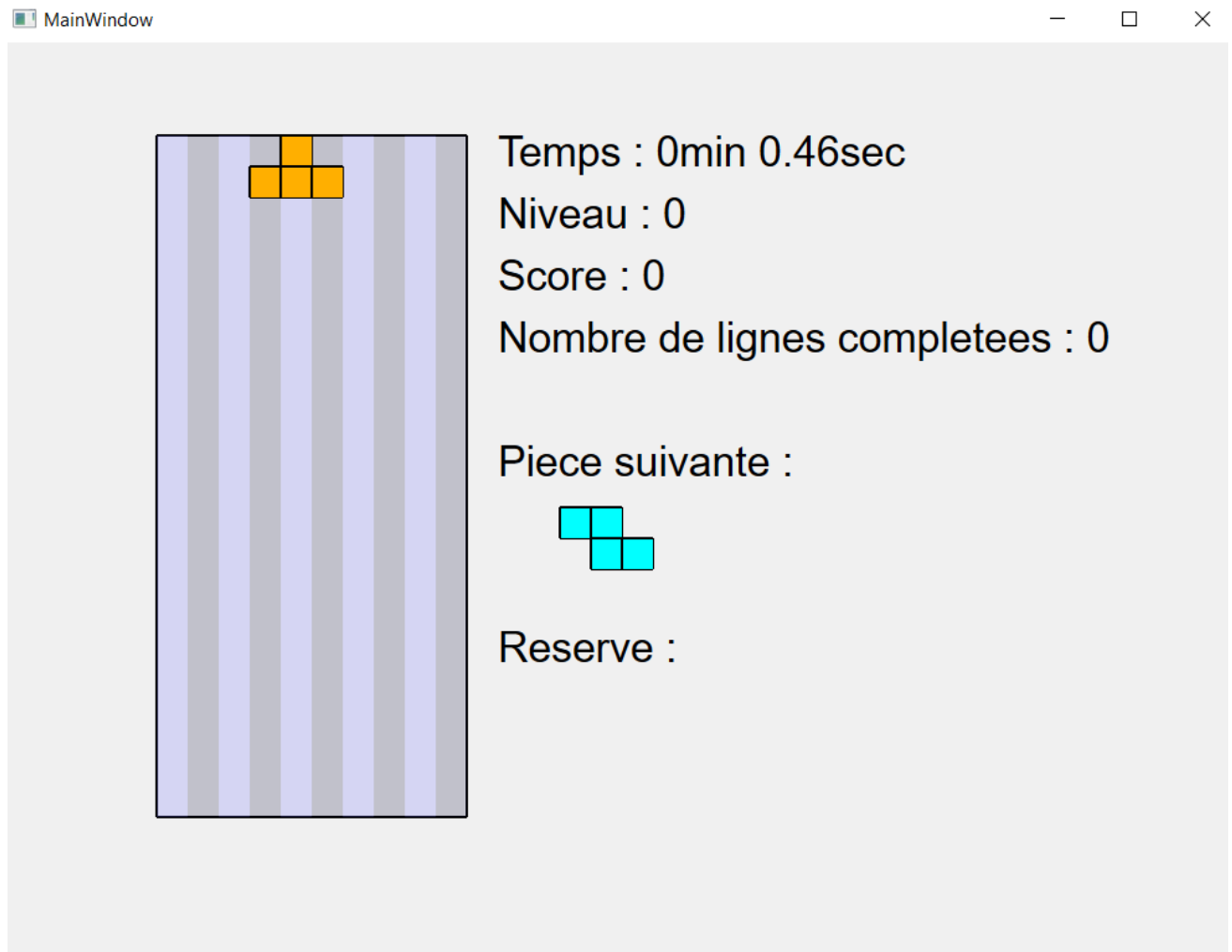


figure 18: Écran de départ

2.2.2. Écran avec tétramino en réserve

On passe à cet écran une fois qu'on a appuyé sur la touche R. Les contrôles sont les mêmes qu'à la partie précédente sauf pour la touche R, qui échange cette fois le tétramino courant avec le tétramino en réserve.

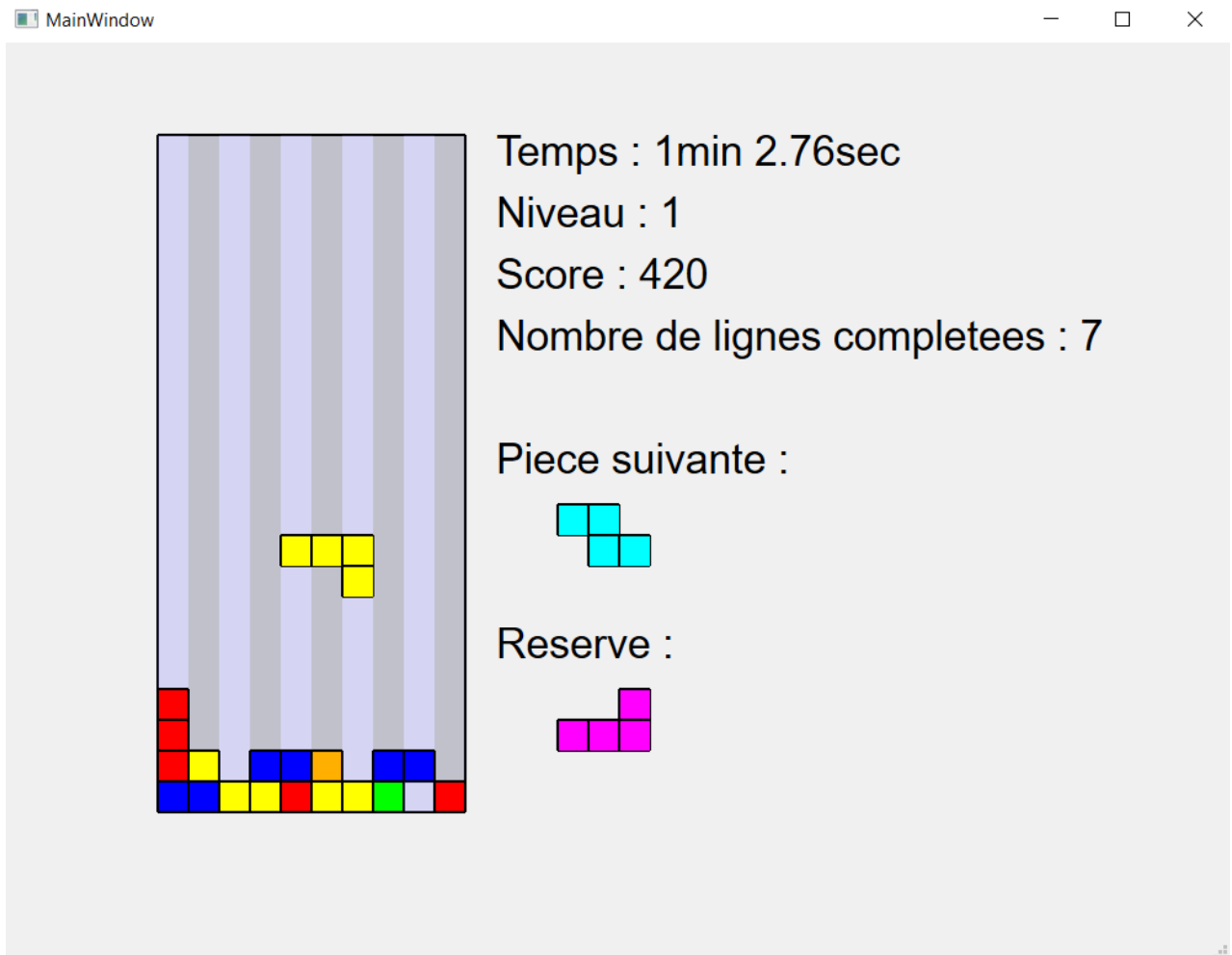


figure 19: Écran avec tétramino en réserve

2.2.2. Écran de fin du jeu

Une fois le puits rempli jusqu'en haut, la partie s'arrête et l'écran de fin apparaît. On affiche le chronomètre arrêté, le niveau atteint, le score et le nombre de lignes complétées.



figure 20: Écran de fin (Game Over)

Bibliographie

<https://www.playstudios.com/tetris/> [image page de garde]

<https://en.wikipedia.org/wiki/Tetris> [1]

<https://dailygeekshow.com/tetris-origines-histoire-creation-aleksei-pajitnov> [2]

https://fr.wikipedia.org/wiki/Liste_des_s%C3%A9ries_de_jeux_vid%C3%A9o_les_plus_vendues [3]

https://fr.wikipedia.org/wiki/Alekse%C3%AF_Pajitnov [4]

<https://openclassrooms.com/forum/sujet/tetris-besoin-d-infos-82930>

<https://tetris.com/play-tetris> [jeu sur lequel on s'est basé]