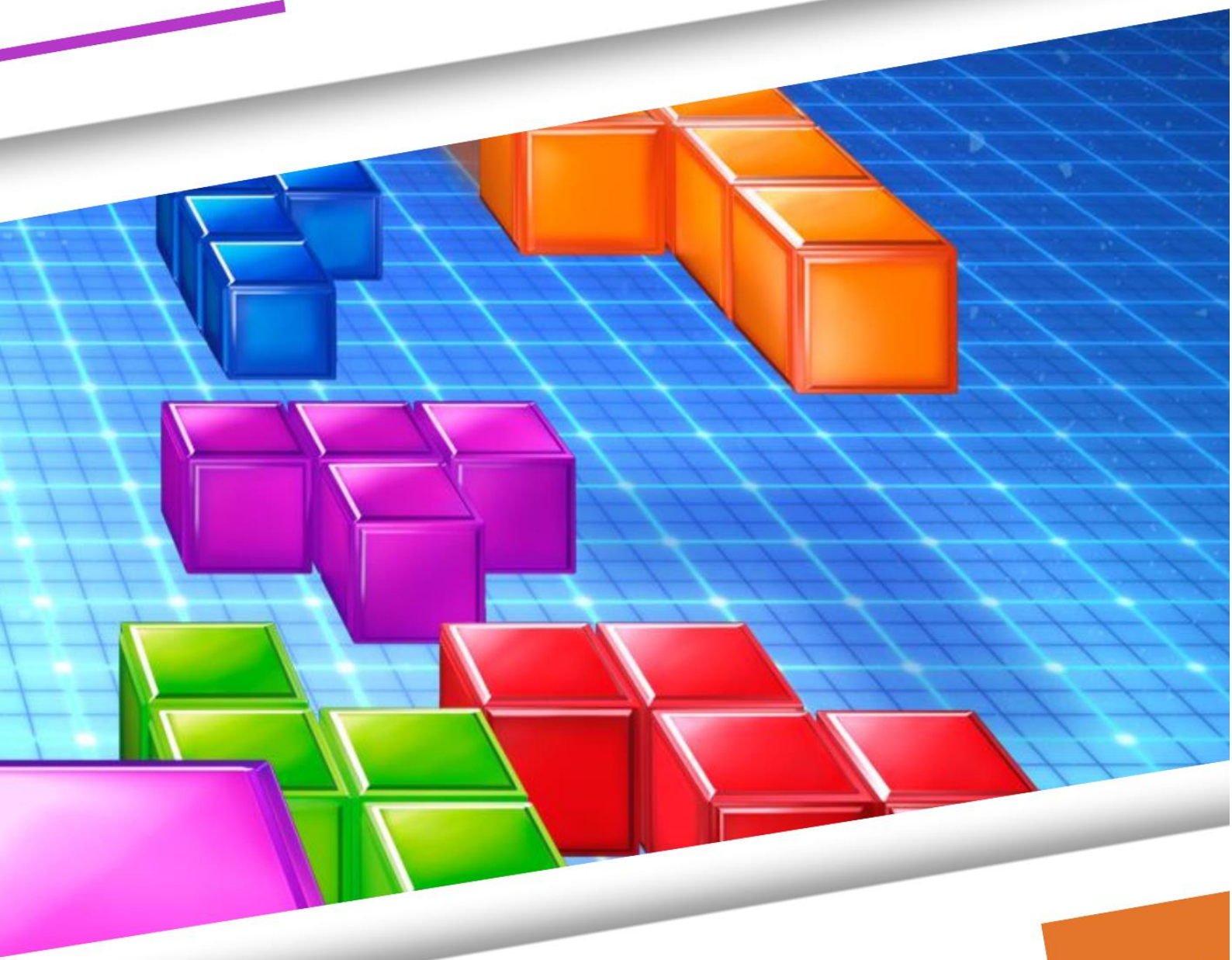


TESSIER CÉCILE



FALLING BLOX


PDLO

Table des matières

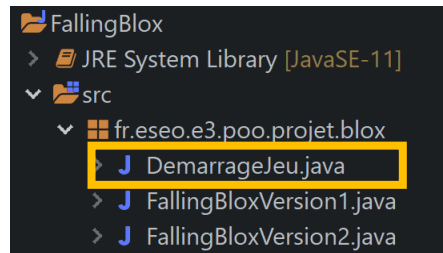
Compiler et exécuter le code.....	3
Extensions.....	4
1. Ajout des pièces manquantes	4
Une TPiece	4
Une LPiece	5
Une JPiece	5
Une ZPiece	6
Une SPiece	7
2. Retirer une ligne et descendre les pièces plus haut.....	7
3. Ajout du score	9
Gestion du score	9
Ajout de l'interface PanneauScore	9
4. Changer la pièce actuelle avec la pièce suivante.....	10
La classe ChangementPiece.....	10
Modification dans la classe VuePuits	11
5. Modification de la vitesse.....	12
6. Descente directe.....	12
7. Utilisation du clavier pour la rotation et le mouvement de la pièce	13
Mouvement de la pièce.....	13
Rotation de la pièce.....	13
8. Fin de partie.....	14
Fonction : fin de partie	14
Classe PanneauGameOver.....	14
9. Rejouer au jeu	15
10. Mettre le jeu en Pause.....	16
11. Image de démarrage.....	17
12. Ajout d'une bande son	18

Compiler et exécuter le code

Pour compiler le code et exécuter le code, il faut se rendre sur eclipse : *Project Explorer -> FallingBlox -> src -> fr.eseo.e3.poo.projet.blox -> DemarrageJeu.java.*

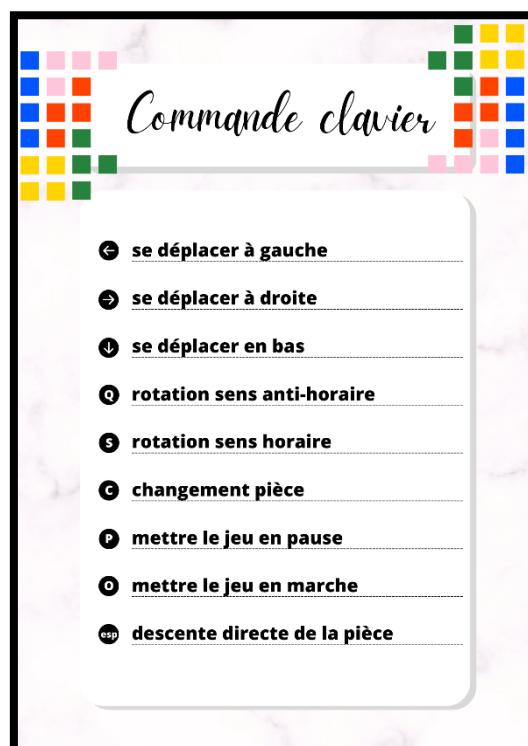
Pour l'exécuter, il faut que vous dirigez la souris vers le bouton run  en haut de l'application.

Le programme compile et se lance. Vous devriez voir apparaître cet onglet :



Tout est prêt pour jouer à mon Falling Blox.

Pour vous aider :



Extensions

1. Ajout des pièces manquantes

La première extension que j'ai réalisée est d'implémenter les différentes pièces manquantes. Il y a sept différentes pièces.

Une TPiece

La première pièce est la **TPiece**. Ci-dessous, la représentation d'une **TPiece** et son diagramme de classe.

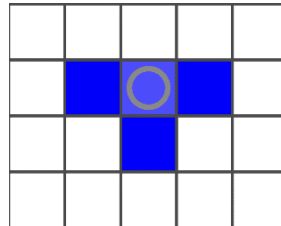


Figure 1 : Une TPiece

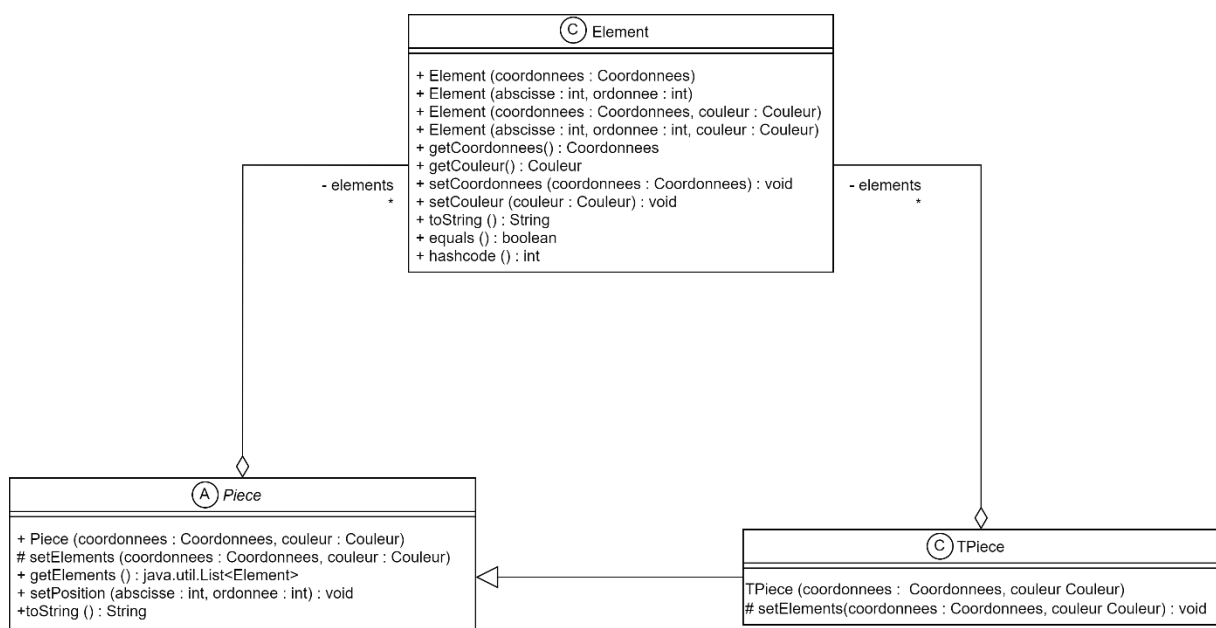


Figure 2 : Diagramme de classe de la classe TPiece

Une LPiece

La deuxième pièce est la **LPiece**. Ci-dessous, la représentation d'une **LPiece** et son diagramme de classe.

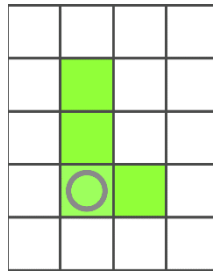


Figure 3 : Une LPiece

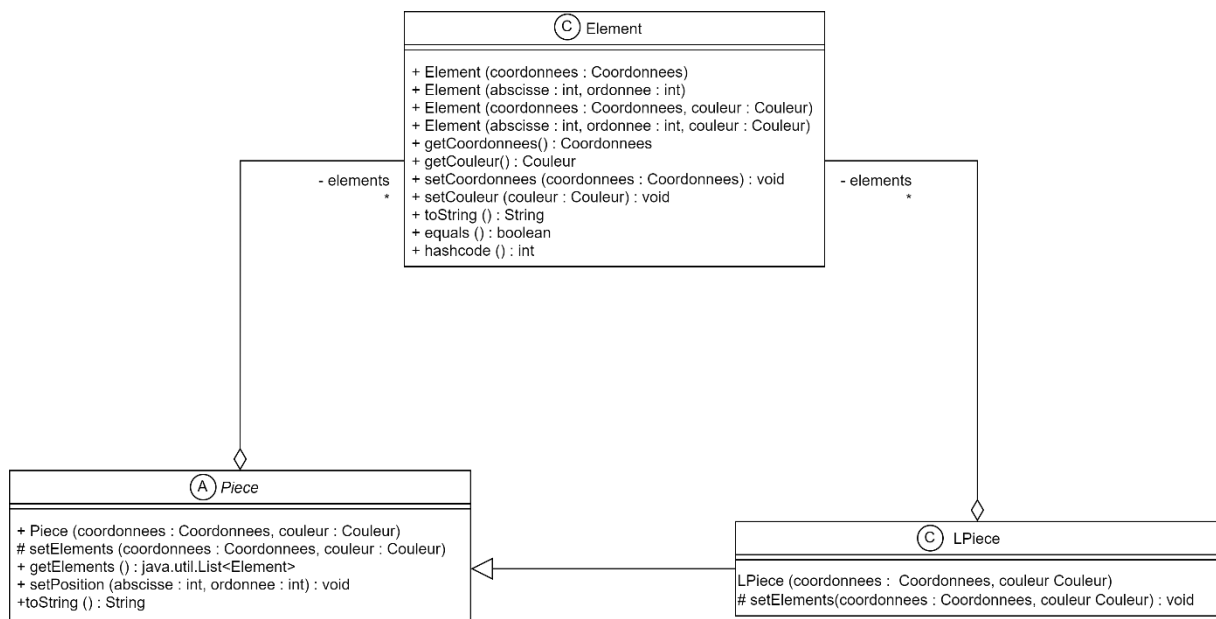


Figure 4 : Diagramme de classe de la classe LPiece

Une JPiece

La troisième pièce est la **JPiece**. Ci-dessous, la représentation d'une **JPiece** et son diagramme de classe.

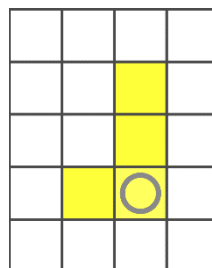


Figure 5 : Une JPiece

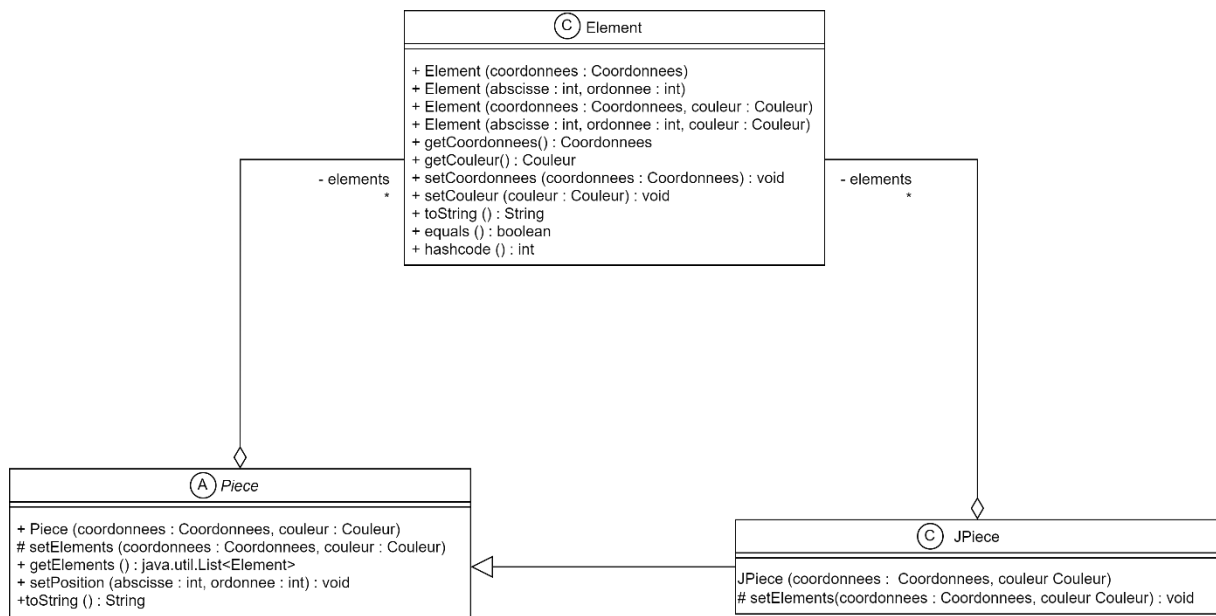


Figure 6 : Diagramme de classe de la classe JPiece

Une ZPiece

La quatrième pièce est la **ZPiece**. Ci-dessous, la représentation d'une **ZPiece** et son diagramme de classe.

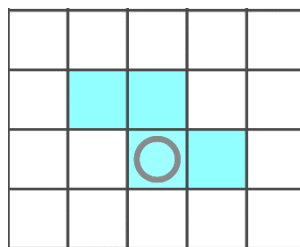


Figure 7 : Une ZPiece

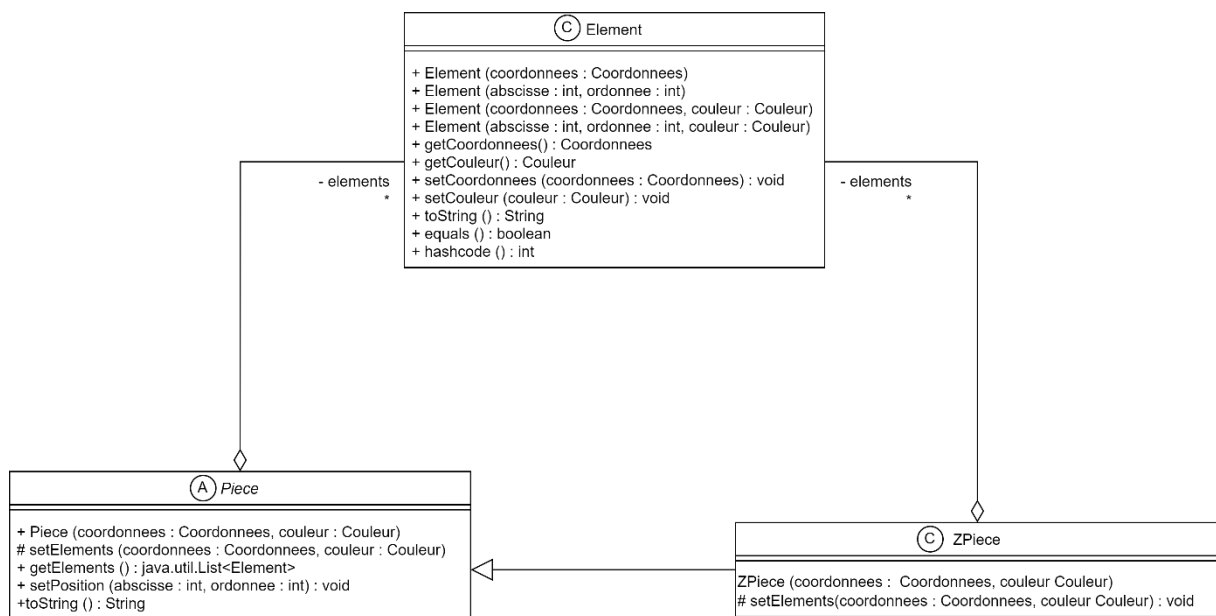


Figure 8 : Diagramme de classe de la classe ZPiece

Une SPiece

La cinquième et dernière pièce est la **SPiece**. Ci-dessous, la représentation d'une **SPiece** et son diagramme de classe.

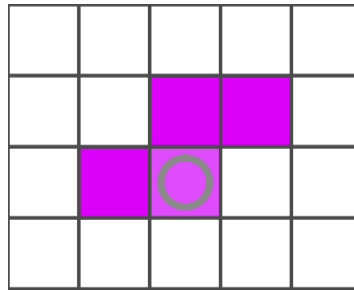


Figure 9 : Une SPiece

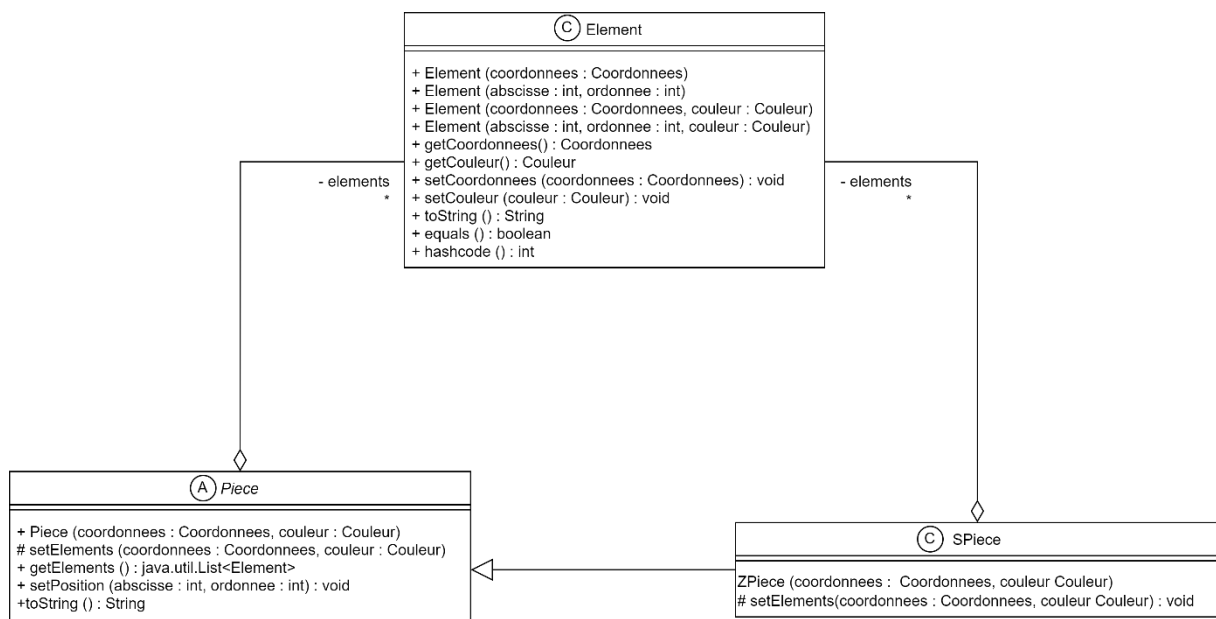


Figure 10 : Diagramme de classe de la classe SPiece

Les classes **TPiece**, **LPiece**, **JPiece**, **ZPiece** et **SPiece** sont créées dans le paquetage **fr.eseo.e3.poo.projet.modele.pieces** avec les autres pièces telle qu'elles sont représentées par les diagrammes de classe.

J'ai créé les cas de test JUnit5 **TPieceTest**, **LPieceTest**, **JPieceTest**, **ZPieceTest** et **SPieceTest** dans le paquetage **fr.eseo.e3.poo.projet.modele.pieces** qui se trouve dans le dossier source test, afin de tester l'implémentation des classes **TPiece**, **LPiece**, **JPiece**, **ZPiece** et **SPiece**.

Après avoir créé ces pièces, j'ai modifié les classes **UsineDePiece** et **UsineDePieceTest** afin de prendre en compte les sept différentes pièces. J'ai donc modifié la fonction **genererPiece()** en complétant le switch case.

2. Retirer une ligne et descendre les pièces plus haut

La deuxième extension que j'ai créée est de détecter quand une ligne est complète dans le tas, pour la supprimer et faire descendre les éléments plus haut pour réduire la hauteur du tas.

Afin d'implémenter cette extension, j'ai créé deux fonctions dans la classe **Tas** :

- **retirerLigne()** : méthode qui retire les lignes complètes de la matrice d'éléments du tas. Pour chaque ligne complète, cette méthode supprime tous les éléments de cette ligne, ajoute le score correspondant au score total du tas (voir extension sur le [score](#)) et descend les éléments de toutes les lignes au-dessus de la ligne supprimée en appelant la méthode **descendreLigne()**.
- **descendreLigne()** : méthode permettant de descendre les éléments de toutes les lignes au-dessus de la ligne spécifiée en paramètre.

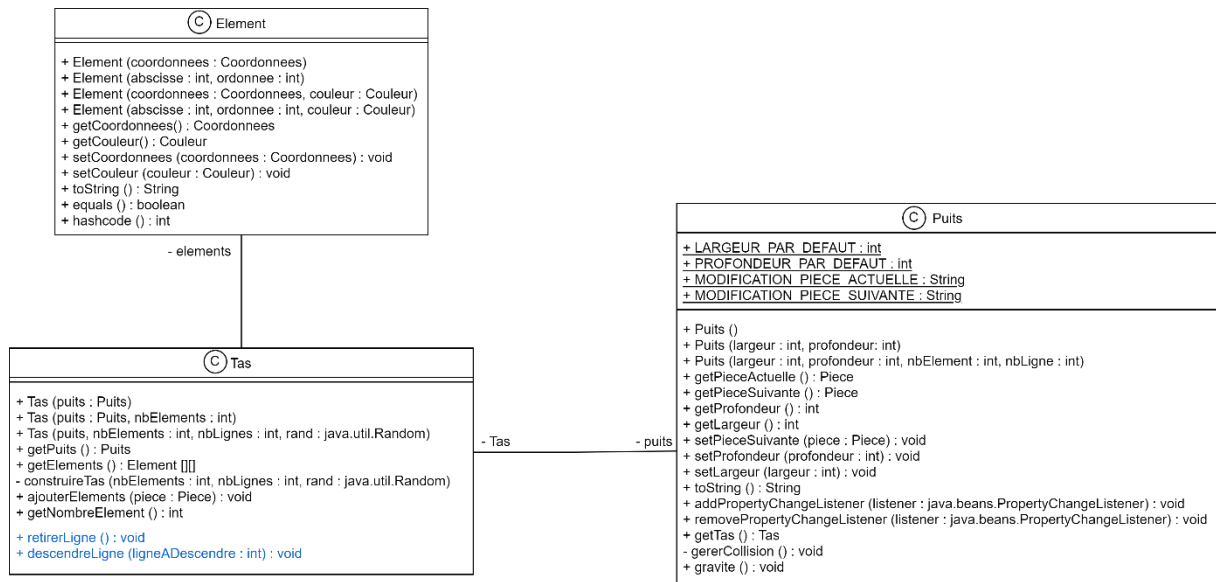


Figure 11 : Diagramme de classe du Tas

J'ai ensuite appelé la fonction **retirerLigne()** de la classe **Tas** dans la fonction **gererCollision()** de la classe **Puits**.

Ainsi, lorsqu'une ligne est complète, le programme supprime les éléments du tas et fait descendre les éléments plus haut.

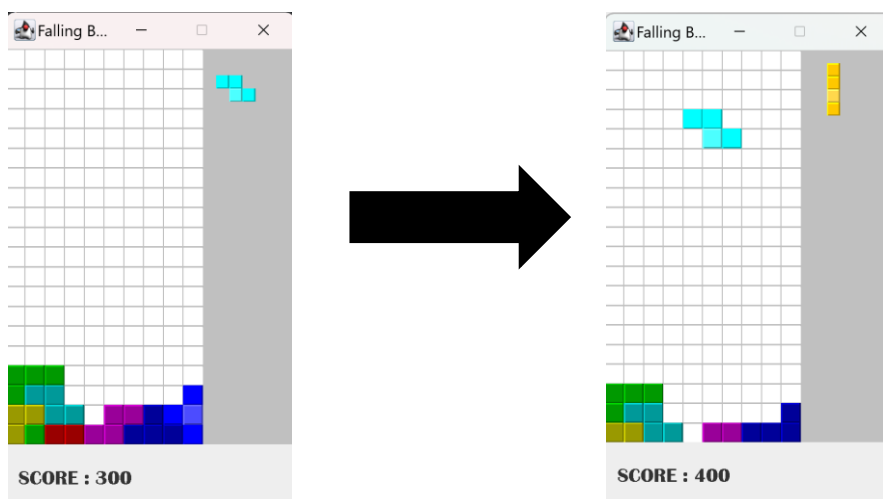


Figure 12 : Représentation graphique de la fonction retirer ligne et descendre les éléments plus haut

Remarque : lorsque le joueur complète plusieurs ligne en même temps, le programme les enlève en même temps.

3. Ajout du score

La troisième extension que j'ai créée est l'ajout d'un score.

Gestion du score

Afin d'implémenter cette extension, j'ai ajouté une instance privé **score** qui est un entier dans la classe **Tas**. Je l'initialise à 0 et je crée l'accesseur qui permet de manipuler la variable d'instance **score**.

Je gère le score dans la méthode **retirerLigne()**. Le score est mis à jour chaque fois qu'une ligne est retirée avec succès. Dans ce cas, le nombre de lignes retirées avec succès est stocké dans la variable **nbrLigneSup**. Le score est alors augmenté de 100 fois le nombre de lignes retirées avec succès, en utilisant l'opération d'addition composée (+). Il y a un bonus quand on retire plus d'une ligne. Ainsi pour :

- 1 ligne supprimée : 100 points ;
- 2 lignes supprimées : 300 points ;
- 3 lignes supprimées : 600 points ;
- 4 lignes supprimées : 1000 points ;

Ajout de l'interface PanneauScore

Afin que le joueur puisse voir le score qu'il a, j'ai décidé de créer une interface. J'ai donc créé la classe **PanneauScore** dans le paquetage **fr.eseo.e3.poo.projet.blox.vue**. Cette classe est une sous-classe de **JPanel** qui permet d'afficher le score. Elle implémente l'interface **java.bean.PropertyChangeListener**.

Son constructeur prend en paramètre une instance de la classe **Puits**. Il enregistre l'instance en tant que **PropertyChangeListener** pour le **Puits** passé en paramètre et règle la taille de préférence pour être 50 x 50 pixels.

J'ai redéfini la méthode **paintComponent()**, elle prend en paramètre un objet **Graphics**, dessine le score actuel sur le panneau en utilisant les graphiques 2D. Elle récupère le score actuel du **Tas** associé à l'objet **Puits** et l'affiche en tant que texte sur le panneau.

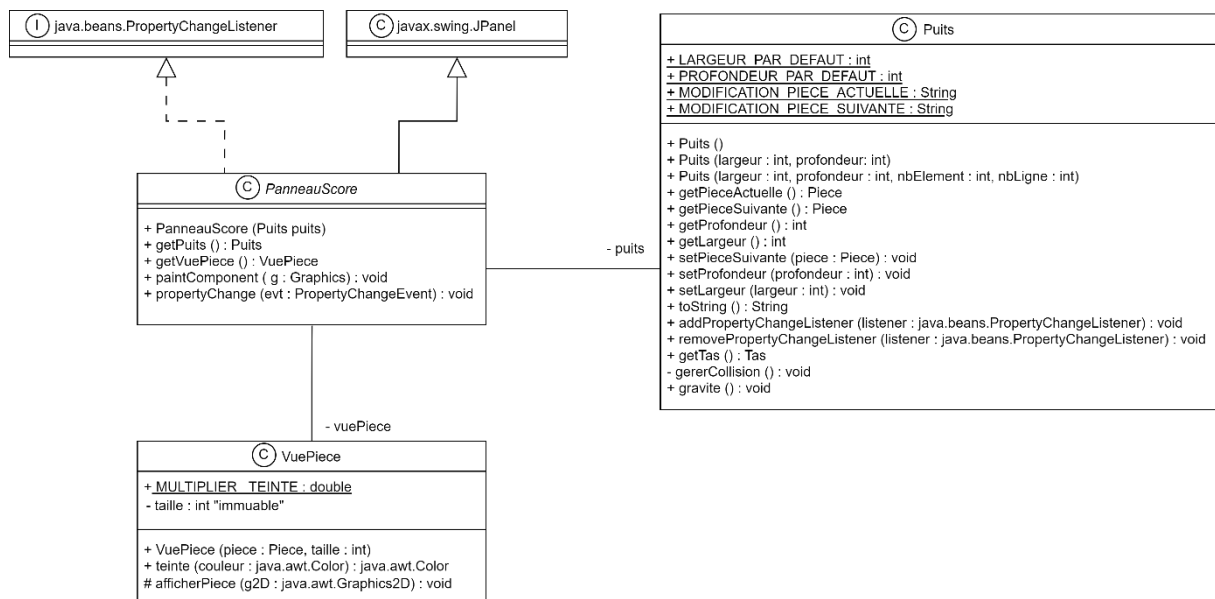


Figure 13 : Diagramme de classe de PanneauScore

J'ai aussi redéfini la méthode `propertyChange()`, c'est la même que pour la classe `PanneauInformation`.
J'appelle le `PanneauScore` dans la classe `FallingBloxVersion2`.

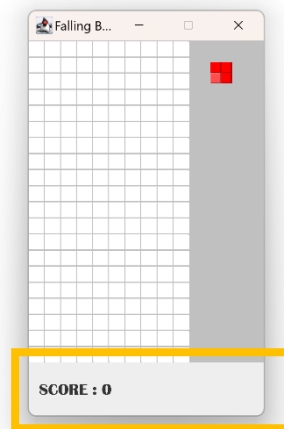


Figure 14 : Représentation graphique du score

4. Changer la pièce actuelle avec la pièce suivante

La quatrième extension est de permettre au joueur de faire un échange entre la pièce actuelle et la pièce suivante.

La classe `ChangementPiece`

Afin d'implémenter cette extension, j'ai créé la classe `ChangementPiece` dans le paquetage `fr.eseo.e3.poo.projet.blox.controleur`.

Cette classe implémente l'interface `java.awt.event.KeyListener`.

Elle est définie selon le diagramme de classe ci-dessous :

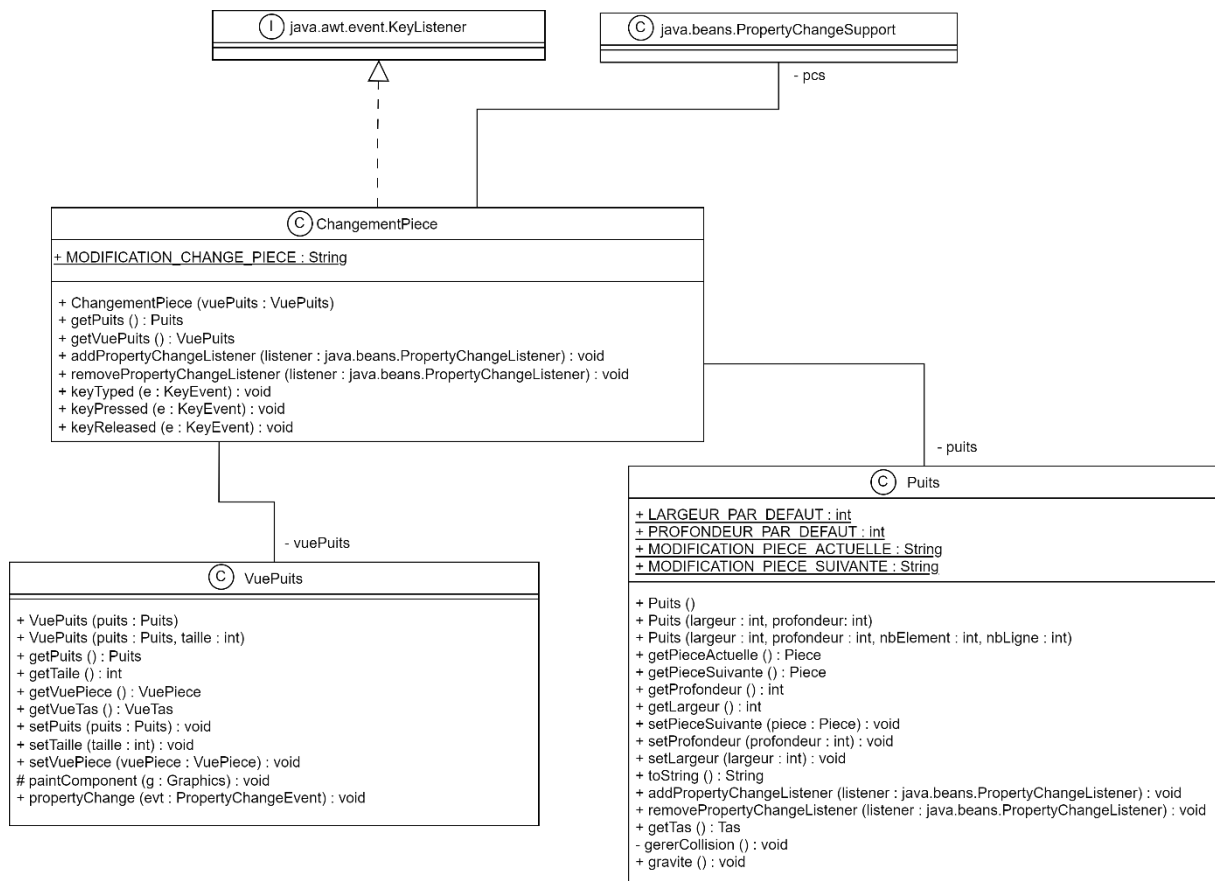


Figure 15 : Diagramme de classe de ChangementPiece

Le constructeur prend en paramètre une instance de la classe **VuePuits**. Après construction de la classe, la variable d'instance **pcs** est initialisé en passant l'instance de **ChangementPiece** comme paramètre vers le constructeur de **PropertyChangeSupport**.

Les méthode **addPropertyChangeListener()** et **removePropertyChangeListener()** sont les même que dans la classe **Puits**. Les méthodes **keyTyped()** et **keyReleased()** ne sont pas nécessaire dans cette classe, elle ne fait donc rien.

La méthode **keyPressed()** est appelée lorsqu'une touche de clavier est enfoncée. Si la pièce actuelle du puits est null, la méthode se termine. Sinon, si la touche « C » est enfoncée, la méthode invoque **firePropertyChange** sur l'objet pcs. Cette méthode permet de signaler à tous les écouteurs enregistrés sur l'objet pcs qu'un changement de pièce est survenu.

Modification dans la classe VuePuits

Dans la classe **VuePuits**, j'ajoute une variable d'instance **ChangementPiece**. Comme avec la classe **Puits**, j'enregistre l'instance de **VuePuits** comme listener de mon instance de **ChangementPiece** (avec la méthode **addPropertyChangeListener()**). J'ajoute aussi dans le constructeur de la classe **VuePuits**, une instance de **ChangementPiece** à la liste de ses gestionnaires d'événements de type **KeyListener** (avec la méthode **addKeyListener()**). Enfin j'utilise la méthode **setFocusable()** dans le constructeur de la classe **VuePuits** pour qu'il détecte les touches de clavier.

Dans la méthode **propertyChange()** de la classe **VuePuits**, j'ai rajouté une condition qui prends en compte le changement de pièce. Je déplace la pièce actuelle vers la position de la pièce suivante et je défini la nouvelle pièce suivante comme pièce actuelle puis je redessine la vue. Ainsi lorsque le joueur appuie sur la touche « C », la pièce actuelle devient la pièce suivante.

5. Modification de la vitesse

La cinquième extension est de modifier la vitesse de la gravité.

Afin d'implémenter cette extension, il a fallu d'abord implémenter l'extension sur le score car je voulais faire en sorte que la vitesse change en fonction du score et pas du nombre de ligne complété comme le suggérait le document du cours sur le projet.

J'ai donc changé la méthode **actionPerformed()** dans la classe **Gravite**. J'ai ajouté des conditions qui vérifie le score actuel du jeu et ajuste la fréquence de gravité en conséquence, en accélérant la chute des pièces. Dans ces conditions je stop le premier timer et j'en crée un autre, puis j'appelle la méthode **gravite()** du puits pour faire tomber la pièce actuelle d'un cran. La vue du puits est ensuite mise à jour avec la méthode **repaint()**.

Ainsi lorsque le score est égal à 200 la vitesse est doublée par rapport au timer précédent. Lorsque le score atteint 400, la vitesse est aussi doublée par rapport au timer précédent. Lorsque le score atteint 600, la vitesse est encore doublée par rapport au timer précédent et ainsi de suite jusqu'à ce que le joueur perde.

6. Descente directe

La sixième extension est d'ajouter la descente directe, quand l'utilisateur appuie sur la touche espace, la pièce se trouve en bas du puits en faisant bien attention aux collisions avec le tas.

Afin d'implémenter cette extension, j'ai modifié la classe **PieceDeplacement** afin qu'elle implémente aussi l'interface **KeyListener**.

Voici son diagramme de classe :

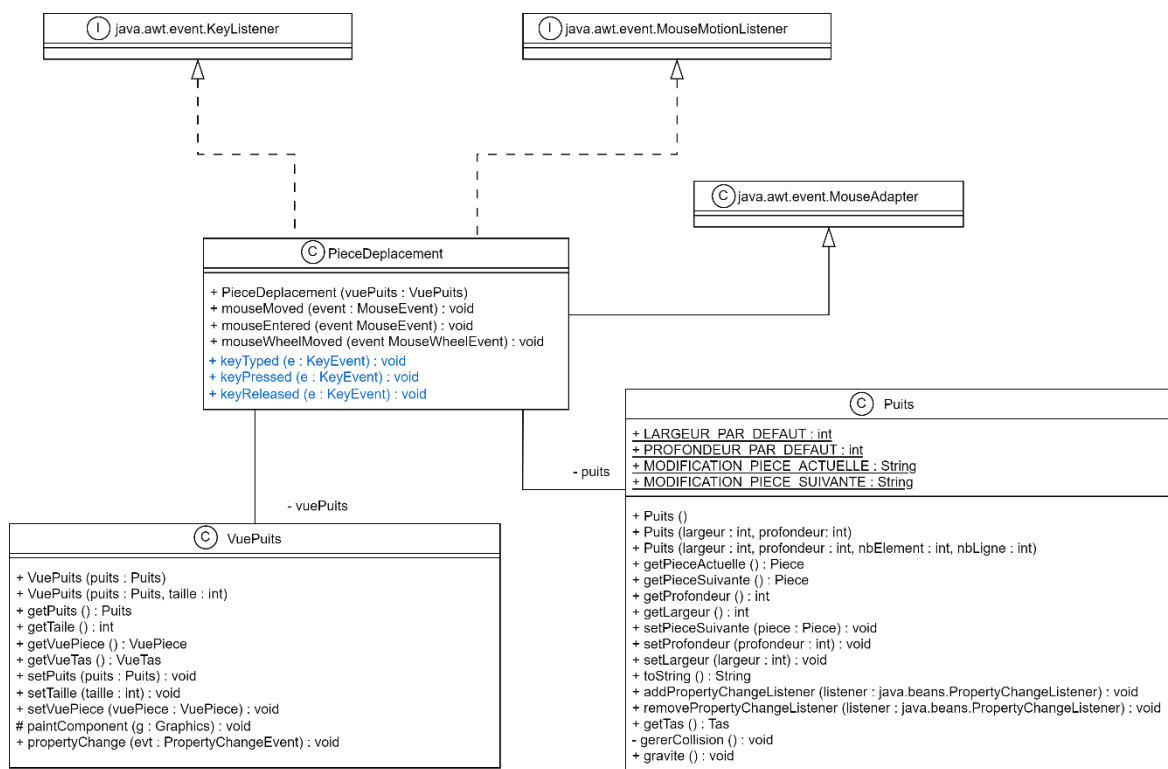


Figure 16 : Diagramme de classe de PieceDeplacement

Comme pour la classe **ChangementPiece**, les méthodes **keyTyped()** et **keyReleased()** ne sont pas nécessaire. La méthode **keyPressed()** est appelé lorsqu'une touche du clavier est enfoncée. Elle vérifie si une touche est actuellement sélectionnée et déplace la pièce en fonction de la touche enfoncé (dans notre cas, lorsque la touche espace est enfoncé) et gère les exceptions qui peuvent être levées lors du déplacement. J'ai donc fait une boucle et j'ai déplacer la pièce jusqu'à ce qu'il y ait une collision.

J'ai modifié le constructeur de la classe **VuePuits**, en ajoutant une instance de **PieceDeplacement** à la liste de ses gestionnaires d'événements de type **KeyListener** (avec la méthode **addKeyListener()**).

Ainsi lorsque le joueur appuie sur la touche « espace », la pièce descend directement au plus bas.

7. Utilisation du clavier pour la rotation et le mouvement de la pièce

La septième extension est d'utiliser le clavier pour la rotation et le mouvement des pièces.

Mouvement de la pièce

Afin d'implémenter cette extension j'ai modifié la fonction **keyPressed()** de la classe **PieceDeplacement**. Comme j'avait déjà fait les changements sur le classe **PieceDeplacement**, il a juste fallu rajouter trois conditions dans la méthode **keyPressed()** pour prendre en compte les déplacements vers la gauche, la droite et le bas. J'utilise ainsi la fonction **deplaceDe()** de la pièce actuelle pour faire déplacer la pièce en fonction des touches flèches vers la gauche, droite et vers le bas.

Rotation de la pièce

Afin d'implémenter cette extension j'ai modifié la classe **PieceRotation** suivant le diagramme de classe ci-dessous :

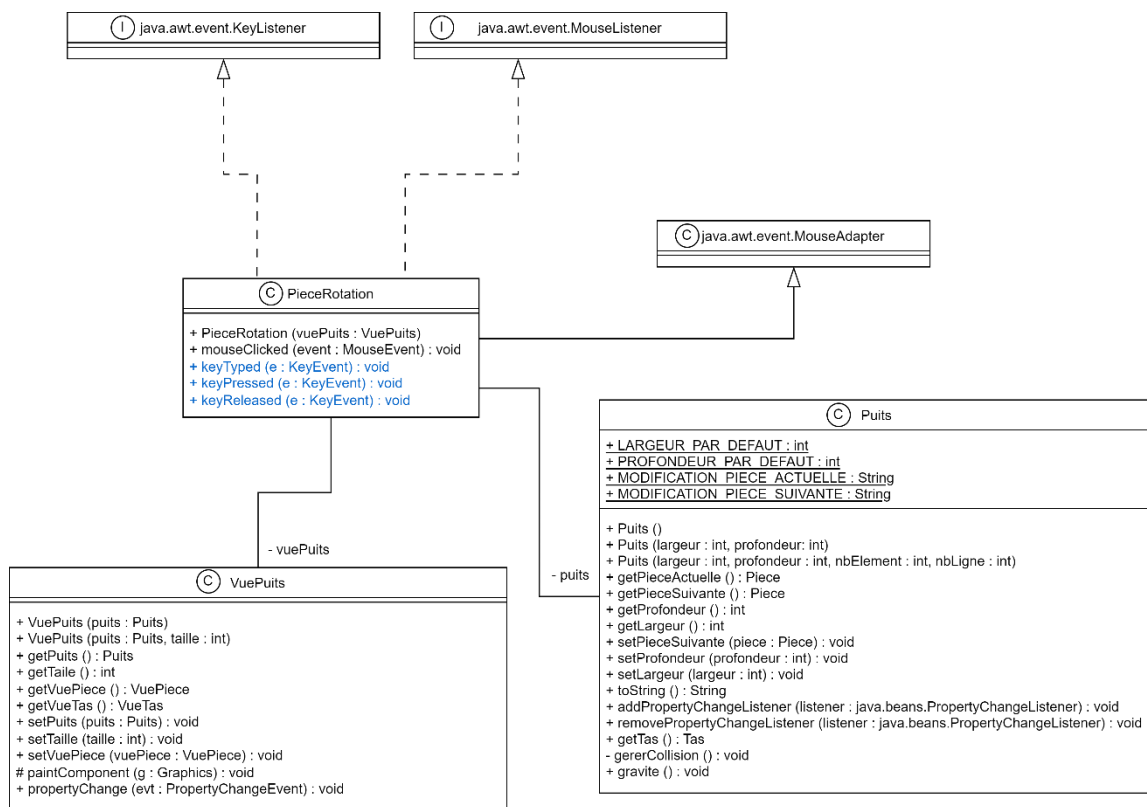


Figure 17 : Diagramme de classe de PieceRotation

Comme avec la classe **PieceDeplacement**, les méthodes **keyTyped()** et **keyReleased()** ne sont pas nécessaire dans notre cas. La méthode **keyPressed()** est appelé lorsque l'utilisateur appuie sur une touche du clavier. Elle vérifie quelle touche a été appuyée (S ou Q) et fait tourner la pièce actuelle dans la direction correspondante. Lorsque le joueur appuie sur la touche « S », la pièce tourne dans le sens horaire alors que si le joueur appuie sur la touche « Q », la pièce tourne dans le sens anti-horaire. Elle redessine ensuite la vue du puits pour afficher la pièce dans sa nouvelle position.

Comme avec la classe **PieceDeplacement**, j'ai modifié le constructeur de la classe **VuePuits**, en ajoutant une instance de **PieceRotation** à la liste de ses gestionnaires d'événements de type **KeyListener** (avec la méthode **addKeyListener()**).

Ainsi, le joueur peut utiliser le clavier pour faire tourner la pièce et la faire déplacer.

8. Fin de partie

La huitième extension est de gérer la fin de partie.

Fonction : fin de partie

Afin d'implémenter cette extension, j'ai créé une méthode **finDePartie()** dans la classe **Tas** afin de gérer la fin de la partie. Lorsqu'il y a un élément à la ligne 0 du tas, la fonction renvoie true, false sinon. Je crée une nouvelle instance **gameOver** qui est un booléen.

J'ai aussi modifié la méthode **ajouterElements()** en faisant ajouter une exception **ArrayIndexOutOfBoundsException**.

Ainsi, on va pouvoir récupérer la fonction **finDePartie()** pour dire au programme de s'arrêter.

Pour cela, dans la classe **VuePuits**, je modifie la méthode **propertyChange()**. La méthode va d'abord vérifier si la partie n'est pas finie, si elle ne l'est pas elle va exécuter la suite du programme sinon le programme va s'arrêter.

Classe **PanneauGameOver**

Afin que le joueur puisse savoir plus précisément qu'il a fini une partie, j'ai décidé de créer un panneau Game Over pour le notifier qu'il a perdu.

Pour cela je crée une classe **PanneauGameOver** suivant son diagramme de classe représenté ci-dessous :

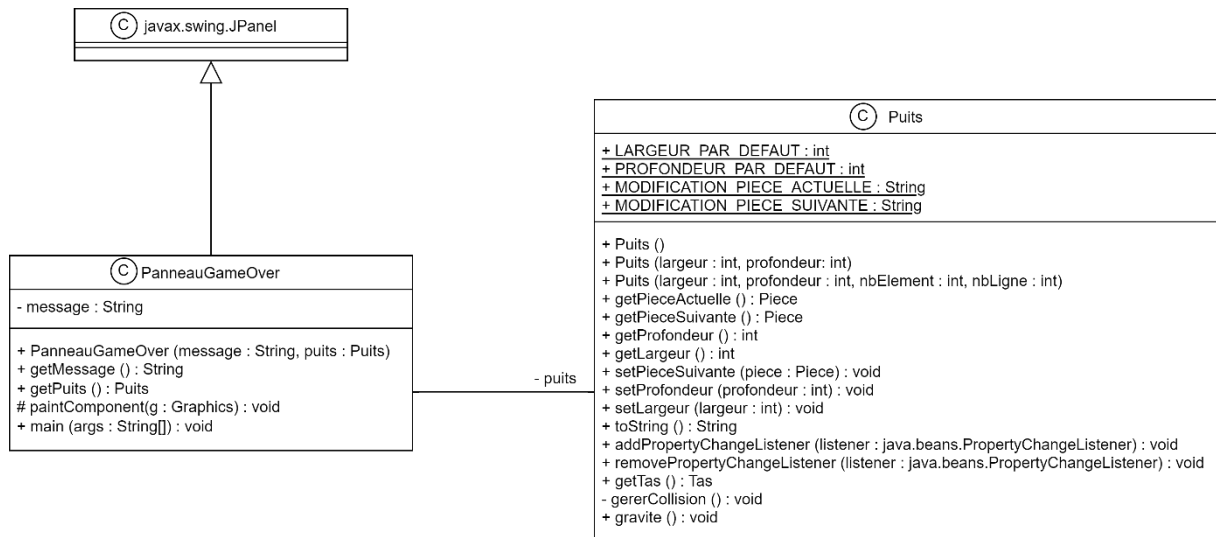


Figure 18 : Diagramme de classe de PanneauGameOver

J'ai redéfini la méthode `paintComponent()` pour faire en sorte qu'il y ait un message « Game over ». J'ai aussi affiché le score final dans cette méthode.

La fonction `main()` consiste à afficher un onglet Game Over. J'affiche le panneau Game over dans la méthode `paintComponent()` de la classe `PanneauInformation` en utilisant la condition avec la fonction `finDePartie()` de la classe `Tas`.

Ainsi lorsque le joueur a perdu sa partie de FallingBlox, il devrait y avoir cet onglet qui s'affiche :



9. Rejouer au jeu

La neuvième extension est de permettre au joueur de pouvoir rejouer au jeu en cliquant sur un bouton.

Afin d'implémenter cette extension j'ai modifié la classe de `PanneauGameOver` suivant son diagramme de classe représenté ci-dessous :

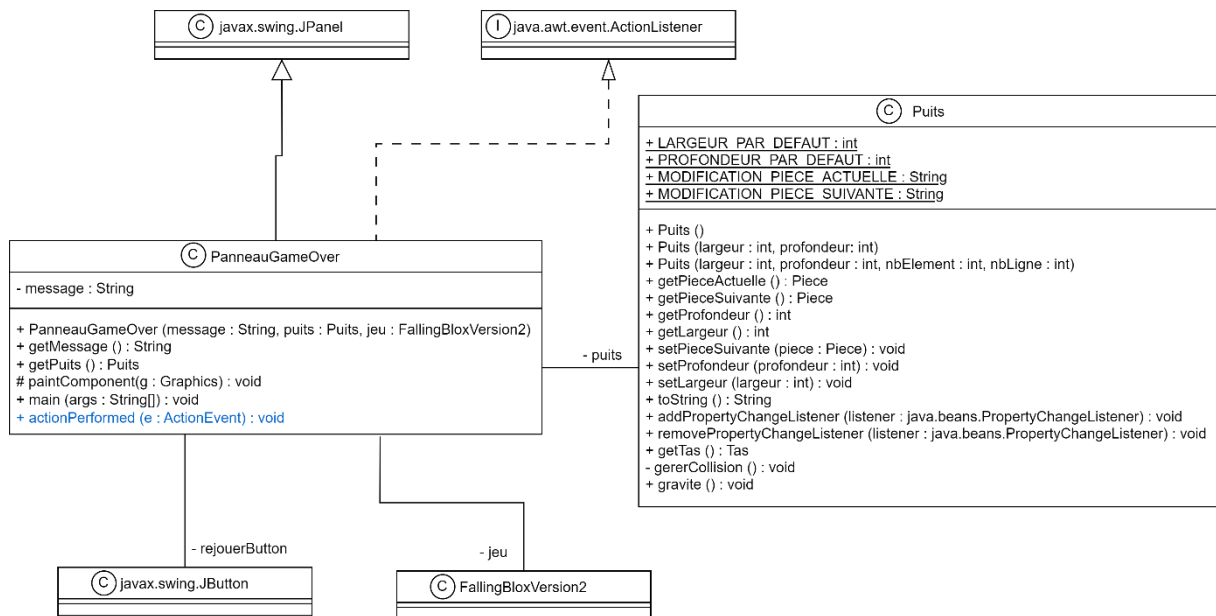
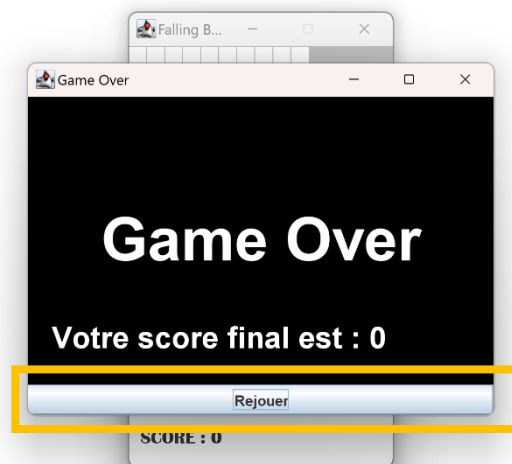


Figure 19 : Diagramme de classe PanneauGameOver

La nouvelle méthode `actionPerformed()` permet de réinitialiser le jeu. Je ferme d'abord les deux fenêtres « Game Over » et « FallingBlox » puis je crée une nouvelle instance de la classe **FallingBloxVersion2**.

J'ai aussi redéfini la classe **PanneauInformation** en rajoutant le constructeur `public PanneauInformation(Puits, FallingBloxVersion2)`.

Ainsi, lorsque le joueur a perdu sa partie il peut rejouer grâce à un bouton « rejouer », il devrait y avoir cet onglet qui s'affiche :



10. Mettre le jeu en Pause

La dixième extension est de mettre le jeu en pause.

Afin d'implémenter l'extension j'ai créé une classe **Pause** suivant son diagramme de classe représenté ci-dessous :

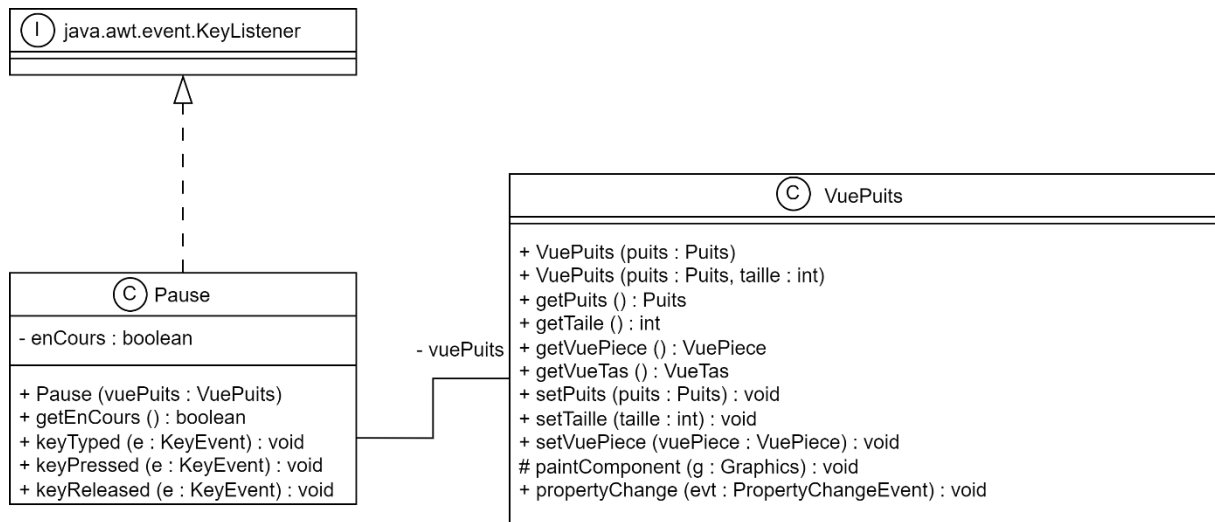


Figure 20 : Diagramme de classe Pause

La méthode **keyPressed()** est appelé lorsque l'utilisateur appuie sur une touche. On vérifie si la touche appuyée est la touche « P », si c'est le cas, on passe la variable enCours à false. Si la touche appuyée est la touche « O », on passe la variable enCours en true.

J'ai modifié les classes **VuePuits**, **PieceDeplacement**, **PieceRotation** et **Gravite** pour prendre en compte le jeu en pause. J'ai donc rajouté dans la méthode **propertyChange()** de la classe **VuePuits**, la condition si le jeu n'est pas en pause, le jeu peut fonctionner normalement. J'ai rajouté la même condition dans :

- Les méthodes **mouseMoved()**, **mouseWheelMoved()** et **keyPressed()** dans la classe **PieceDeplacement** ;
- Les méthodes **mouseClicked()** et **keyPressed()** dans la classe **PieceRotation** ;
- La méthode **actionPerformed()** dans la classe **Gravite**.

Ainsi lorsque le joueur appuie sur la touche « P », le jeu se met en pause, et si le joueur appuie sur la touche « O », le jeu se remet en marche.

11. Image de démarrage

La onzième extension est d'ajouter une image de démarrage du jeu puis de pouvoir cliquer sur un bouton pour que le jeu se lance.

Afin d'implémenter cette extension, j'ai créé une nouvelle classe **DemarrageJeu** suivant son diagramme de classe représenté ci-dessous :

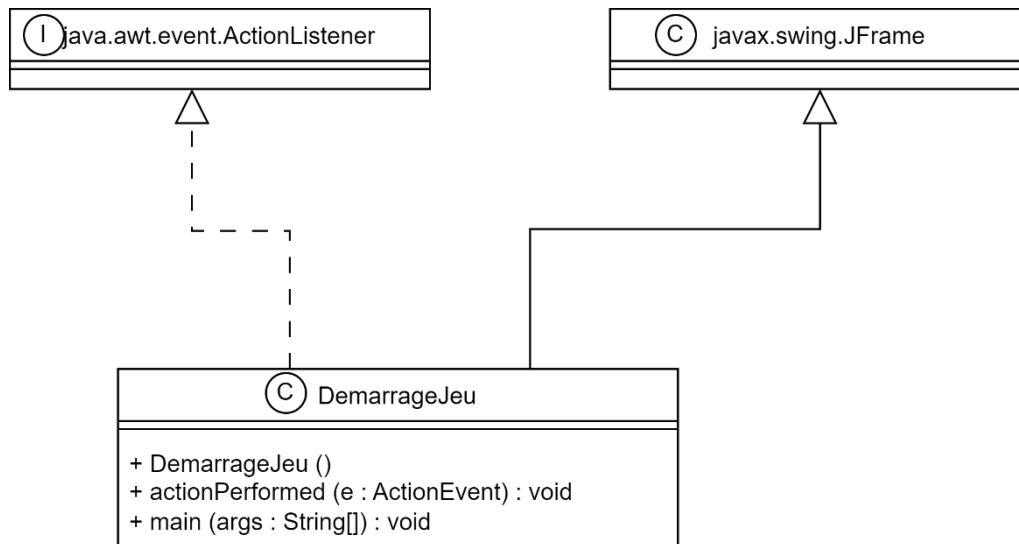


Figure 21 : Diagramme de classe DemarrageJeu

Dans le constructeur, je crée une fenêtre puis une image de fond. Je redimensionne l'image puis je l'ajoute à la fenêtre. Je crée ensuite le bouton « Jouer » puis je l'ajoute à la fenêtre. Je rends ensuite la fenêtre visible.

La méthode `actionPerformed()` est appelée lorsque l'utilisateur clique sur le bouton « Jouer ». Elle crée une nouvelle instance de la classe `FallingBloxVersion2`. Si une erreur se produit lors de la création de l'instance, elle affiche la pile d'erreurs. Elle cache ensuite la fenêtre de démarrage.

Ainsi lorsqu'on exécute le programme, il devrait y avoir cet onglet qui s'affiche :



12. Ajout d'une bande son

La douzième extension est d'ajouter une bande son lorsque le jeu se lance.

Afin d'implémenter cette extension, j'ai créé une classe `LecteurMP3` suivant son diagramme de classe représenté ci-dessous :

La méthode **play()** lit le fichier audio spécifié dans la variable fichier. Elle récupère un objet

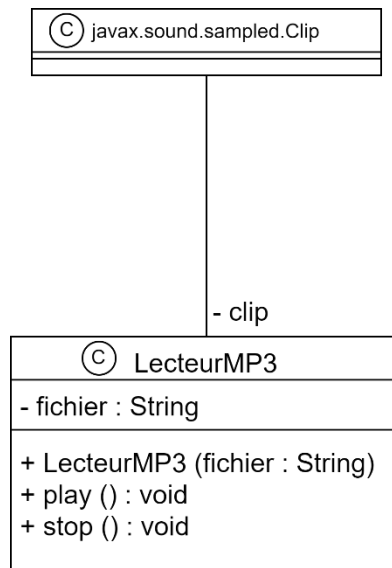


Figure 22 : Diagramme de classe LecteurMP3

AudioInputStream à partir de la bibliothèque **AudioSystem** de Java pour pouvoir lire le fichier audio. Elle récupère un objet **Clip** à partir de la bibliothèque **AudioSystem** de Java et l'ouvre avec l'objet **AudioInputStream** précédemment créé. Elle démarre la lecture du fichier audio à l'aide de la méthode **start()** de l'objet **Clip**.

La méthode **stop()** arrête la lecture du fichier audio en cours avec la méthode **stop()** de l'objet **Clip**.

Ainsi lorsque le jeu se lance une bande son commence à se lancer.