






EXTENSIONS

Falling Blox 2023 - CP00

par Antoine Leroux

Table des matières

 Échange de pièce	2
Description	2
Réalisation	2
 Déplacements clavier	4
Description	4
Réalisation	4
 Descente rapide	5
Description	5
Réalisation	5
 Score	6
Description	6
Réalisation	6
 Gravité accélérée	7
Description	7
Réalisation	7

Échange de pièce

Description

Lorsque le joueur appuie sur une touche, la pièce actuelle et la pièce suivante sont échangées.

Action

Appui sur la touche **R**.

Réalisation

Création d'une classe **PieceInteraction** qui étend de l'interface **KeyListener**. J'aurais pu la faire hériter de **KeyAdapter**, mais les classes **PieceDeplacement** et **PieceRotation** héritent déjà d'une classe mère. Cela aurait réduit les possibilités par la suite.

Dans la classe **VuePuits**, je crée une instance de **PieceInteraction** et je l'ajoute à la liste des observateurs du puits. Cela permettra de transmettre les événements (clavier) aux différents contrôleurs.

```
public class VuePuits extends JPanel implements PropertyChangeListener {

    private PieceDeplacement pieceDeplacement;
    private PieceRotation pieceRotation;
    private PieceInteraction pieceInteraction;

    public VuePuits(Puits puits, int taille) {
        super();
        ...
        // Create controllers
        this.pieceDeplacement = new PieceDeplacement(this);
        this.pieceRotation = new PieceRotation(this);
        this.pieceInteraction = new PieceInteraction(this);
        ...
        // Add listeners
        this.addMouseListener(this.pieceDeplacement);
        this.addMouseListener(this.pieceRotation);
        this.addMouseMotionListener(this.pieceDeplacement);
        this.addMouseWheelListener(this.pieceDeplacement);
        this.addKeyListener(this.pieceDeplacement);
        this.addKeyListener(this.pieceRotation);
        this.addKeyListener(this.pieceInteraction);
        ...
        // Make the window sensitive to focus (KeyListener)
        this.setFocusable(true);
    }
    ...
}
```

J'ai ensuite implémenté les méthodes **keyPressed**, **keyReleased** et **keyTyped** de l'interface. Dans **keyPressed**, je récupère le code de la touche appuyée, puis je teste si c'est un **R**. Dans ce cas, j'appelle la méthode **echangerPiece** du puits.

Cette méthode permet d'échanger la pièce actuelle et la pièce suivante si elles existent. À la fin de la méthode, je notifie les observateurs du puits que les pièces ont été changées.

```
public void echangerPiece() {
    if (this.pieceActuelle == null || this.pieceSuivante == null) return;

    // Swap the current piece with the next one
    Piece tempPiece = this.pieceActuelle;
    this.pieceActuelle = this.pieceSuivante;
    this.pieceSuivante = tempPiece;

    // Set the position of the current piece
    this.pieceActuelle.setPosition(
        tempPiece.getElements().get(0).getCoordonnees().getAbscisse(),
        tempPiece.getElements().get(0).getCoordonnees().getOrdonnee()
    );

    // Trigger the listeners
    pcs.firePropertyChange(MODIFICATION_PIECE_ACTUELLE, this.pieceSuivante, this.pieceActuelle);
    pcs.firePropertyChange(MODIFICATION_PIECE_SUIVANTE, this.pieceActuelle, this.pieceSuivante);
}
```



Déplacements clavier

Description

Permettre au joueur de déplacer la pièce avec les flèches du clavier.

Actions

Appui sur la touche ← pour déplacer la pièce à gauche.

Appui sur la touche → pour déplacer la pièce à droite.

Appui sur la touche ↓ pour déplacer la pièce vers le bas.

Appui sur la touche ↑ pour tourner la pièce.

Réalisation

J'ai étendu les classes **PieceDeplacement** et **PieceRotation** avec l'interface **KeyListener**, comme pour l'extension précédente. J'ai implémenté les méthodes **keyPressed**, **keyReleased** et **keyTyped** de l'interface.

Dans **keyPressed**, je récupère le code de la touche appuyée, puis je teste les quatre possibilités. Dans le cas d'une flèche droite ou gauche, je déplace la pièce avec la méthode **deplacerDe**. Pour la flèche du bas, j'appelle la méthode **gravite** du puits afin de faire descendre la pièce. Enfin, pour la flèche du haut, j'appelle la méthode **tourner** de la pièce actuelle.

J'appelle ensuite la méthode **repaint** du puits pour mettre à jour l'affichage.

Les fonctions relatives aux déplacements de la pièce sont dans la classe **PieceDeplacement**. Tandis que la gestion de la rotation est dans la classe **PieceRotation**.



Descente rapide

Description

Permettre au joueur de faire tomber sa pièce instantanément au fond du puits en cliquant sur sa molette de souris.

Actions

Appui sur **clic molette**.

Réalisation

J'ai utilisé la méthode **mousePressed** de la classe mère **MouseAdapter**, dans **PieceDeplacement**. Cette méthode est déclenchée lorsque l'utilisateur appuie sur un bouton de la souris.

J'ai donc testé si la touche correspondait à la molette de la souris, puis j'ai appelé la méthode **dropdown** du puits.

J'ai créé la méthode **dropdown** dans la classe **Puits**. Cette méthode permet de faire descendre la pièce actuelle jusqu'à ce qu'elle touche le fond du puits ou une autre pièce.

```
public void dropdown() {
    try {
        while (true) this.pieceActuelle.deplacerDe(0, 1);
    }
    catch (BloxException e) {
        // Check if the piece is out of the pit
        if (e.getType() == BloxException.BLOX_COLLISION) this.gererCollision();
        else throw new RuntimeException(e);
    }
}
```



Description

Afficher le score du joueur en fonction du nombre de lignes qu'il a fait disparaître.

Réalisation

J'ai tout d'abord créé une classe **Score**. Cette classe possède un attribut **score** qui est incrémenté à chaque fois que le joueur fait disparaître une ligne. J'ai également créé une méthode **getScore** qui permet de récupérer le score actuel.

```
public class Score {  
  
    private int score = 0;  
  
    public Score() {  
    }  
  
    public void ajouter(int points) {  
        if (points < 0) return;  
        this.score += points;  
    }  
  
    public int getScore() { return this.score; }  
  
    public void reset() {  
        this.score = 0;  
    }  
}
```

Pour afficher ce score, j'ai créé une classe **VueScore**.

```
public class VueScore {  
  
    private Score score;  
  
    public VueScore (Score score) {  
        this.score = score;  
    }  
  
    public void afficherScore(Graphics2D g2D) {  
        g2D.setColor(Color.BLACK);  
        g2D.drawString("Score: " + this.score.getScore(), 10, 20);  
    }  
}
```

afficherScore est appelé dans la méthode **paintComponent** de **PanneauInformation**.



Gravité accélérée

Description

Augmenter progressivement la vitesse de descente des pièces.

Réalisation

J'ai implémenté une gestion de la vitesse en fonction du score, dans la classe **Score**. La classe reçoit maintenant en paramètre une instance de **Gravite**. Cet objet voit la durée de son timer diminuer en fonction du score.

```
public class Score {  
  
    private int score = 0;  
    private float speed;  
  
    private Gravite gravite;  
  
    public Score(Gravite gravite) {  
        this.gravite = gravite;  
        this.updateSpeed();  
    }  
  
    public void ajouter(int points) {  
        if (points < 0) return;  
        this.score += points;  
        this.updateSpeed();  
    }  
  
    private void updateSpeed() {  
        this.speed = 1 + ((float)this.score / 1000);  
        this.gravite.setPeriodicite(Math.round(1000/speed));  
    }  
}
```