

Projet

Mazing : une API graphique pour jeu d'aventure en projection isométrique

Introduction

Mazing est une API Java qui exploite javax.swing.* et java.awt.*. C'est-à-dire la bibliothèque de composants graphiques en Java.

Elle permet de dessiner un décor de jeu, de placer des objets et des personnages et d'animer les personnages. Le dessin est réalisé en isométrique 3D, c'est-à-dire une représentation de l'espace sans perspective.

Pour l'utiliser, il faut télécharger mazing.jar et lancer la compilation et l'exécution du projet :

```
javac -jar <chemin mazing>/lib/mazing.jar
      -d bin <chemin projet>/src/**/*.java
java -cp <chemin mazing>/lib/mazing.jar:bin
      <package projet>.<classe principale projet>
```

Les programmes minimaux project00, project01 et project02 illustrent par l'exemple ce qui est possible de faire avec cette API.

Fenêtre principale

fr.ubordeaux.ao.mazing.api.WindowGame est une classe qui implémente fr.ubordeaux.ao.mazing.api.IWindowGame et qui étend javax.swing.JFrame

On l'utilise ainsi :

```
import fr.ubordeaux.ao.mazing.api.IWindowGame;
import fr.ubordeaux.ao.mazing.api.WindowGame;
...
IWindowGame windowGame = new WindowGame();
```

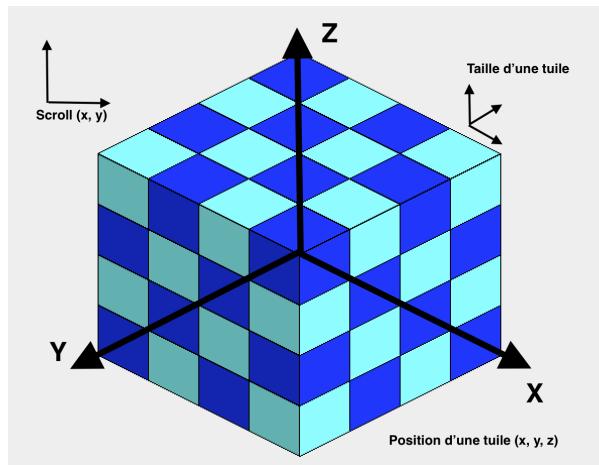
Toutes les méthodes de javax.swing.JFrame sont disponibles pour windowGame en retypant l'objet. On peut donc par exemple ajouter des éléments indépendants de **Mazing**, comme des images, des boutons, des menus, etc.

Exemple :

```
import fr.ubordeaux.ao.mazing.api.IWindowGame;
import fr.ubordeaux.ao.mazing.api.WindowGame;
javax.swing.JFrame;
...
IWindowGame windowGame = new WindowGame();
JLabel label = new JLabel("Un Label !");
((JFrame)windowGame).add(label);
```

Un objet WindowGame contient l'image du jeu. Toutes les coordonnées sont indiquées sous forme cartésienne x, y, z, et la projection dans l'espace isométrique est assurée par l'API automatiquement.

Il est possible de paramétriser la taille des tuiles et le placement de l'image relativement à la fenêtre. L'API se charge d'afficher les éléments visibles en prenant compte de leur topologie.



La classe WindowGame

WindowGame implémente IWindowGame :

- Pour gérer la scène :
 - **void add(ICharacter<? extends ICharacterMode> character)**
Ajoute un personnage à la scène ;
 - **void add(int code, int x, int y, int z)**
Ajoute un élément de décor code (voir plus loin) à la scène en (x, y, z).
 - **void add(int code, int x, int y, int z, float alpha)**
Même chose avec transparence (alpha=0f : transparent, 1f : opaque).
 - **void add(int [][] matrix)**
Ajoute tous les éléments de décor code = matrix[z][y][x] à la scène en (x, y, z).
 - **void add(int [][] matrix, float alpha)**
Même chose avec transparence (alpha=0f : transparent, 1f : opaque).
 - **void fillArea(int code, int x, int y, int z, int width, int height)**
Ajoute un élément de décor (voir plus loin) en x, y, z en le répétant sur width × height.
Cela permet de poser un sol avec un carrelage par exemple.
 - **void fillArea(int code, int x, int y, int z, int width, int height, float alpha)**
Même chose en ajoutant un facteur de transparence
 - **void clear()**
Supprime les objets de la scène.
- Pour gérer la fenêtre. Des valeurs raisonnables ont été fixées au démarrage. Il est inutile d'y toucher dans un premier temps, sauf setVisible (**true**) qu'il faudra ajouter.
 - **void setVisible(boolean b)**
La fenêtre est visible ou non.
 - **void setSize(int width, int height)**
Redéfinit la taille de la fenêtre en pixels. Défini à 1024 × 665 par défaut.
 - **void setTileSize(int tileSize)**
Redéfinit la taille des tuiles en pixels. Défini à 100 × 100 par défaut.
 - **void setFPS(int fps)**
Redéfinit la vitesse d'animation en nombre d'images par seconde. Fixé à 50 pfs par défaut.
 - **void scroll(int x, int y)**
Redéfinit la position du dessin dans la fenêtre. Défini à (0, 0) par défaut, c'est-à-dire au centre de la fenêtre.
 - **Point2D getIsoCoordinatesFromScreen(int x, int y)**
import java.awt.geom.Point2D;
Convertit un point de la fenêtre (coordonnées écran) en coordonnées isométriques sur la carte. Utile pour pointer avec la souris un personnage ou une tuile.
- Pour gérer les sons :
 - **void playSound(String soundId)**
Joue un son à partir de son identifiant (ex : "weapons/explosion") joue le son qui se trouve

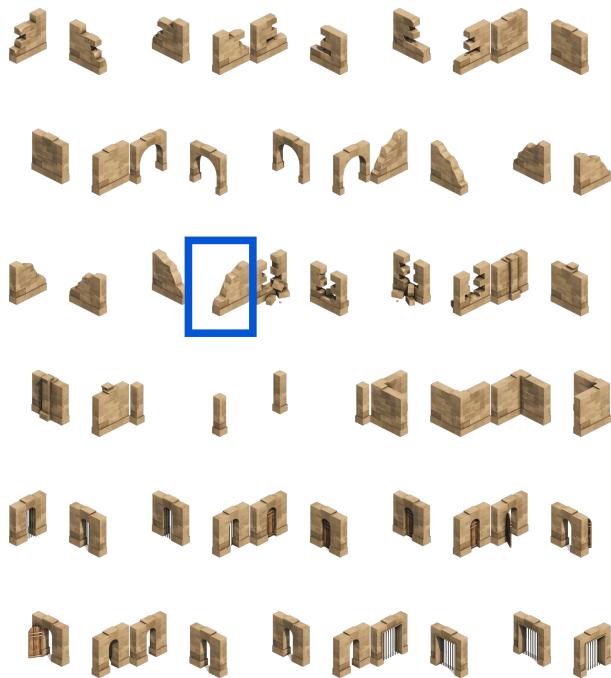
- dans sounds/weapons/explosion.wav.
- **void playShortSound(String soundId, int millis)**
Même chose, mais limite le son à millis ms.
 - Pour déboguer :
 - **void addCubeBackground(Color color, int x, int y, int z)**
Nécessite **import** java.awt.Color;
Dessine un cube de la grosseur d'une tuile en x, y, z
 - **void addTileBackground(Color color, int x, int y, int z)**
Nécessite **import** java.awt.Color;
Dessine une tuile de la couleur en x, y, z
 - **void fillCheckerboardBackground(int x, int y, int width, int height)**
Dessine un damier de \times height en x, y, 0

Éléments du décor

Les éléments du décor sont déjà dessinés (Dungeon Pack (2.3) Licence CCO) et sont identifiés par des nombres codés.

Les dessins se trouvent sur des planches 000, 100, 200, etc. (voir annexe)

Chaque planche est une planche d'au plus 100 dessins sur une grille de 10 lignes \times 10 colonnes.
La centaine du code indique la planche, la dizaine indique la ligne et l'unité indique la colonne.
Par exemple le code 123 correspond à l'image d'un mur cassé placé à l'Est d'une tuile :
`mazing/src/main/resources/Dungeon-Pack/Isometric/walls/all_100.png`



Personnages

Nous avons ajouté deux personnages :
Crusader (Bleed - remusprites.carbonmade.com CC-BY 4.0 License)



Spider (Remos - remusprites.carbonmade.com CC-BY 4.0 License)



Ces personnages (*characters*) sont animés, ont des modes (marche, court, saute, etc...), des directions et des sons qui permettent de leur donner vie dans le jeu.

Classes Crusader et Spider

Ces classes implémentent l'interface `ICharacter<S extends ICharacterMode>` où `ICharacterMode` est l'interface correspondant au mode du personnage.

Crusader et Spider implémentent :

- Le constructeur avec ou sans un identifiant de type `UUID`.
- **void setPosition(float x, float y, float z)**
Définit la position du personnage dans l'espace du jeu.
- **void setDirection(Direction targetDirection)**
Définit la direction vers laquelle le personnage est orienté sur le plan `x×y`.
Direction correspond aux 8 directions SOUTHEAST, EAST, NORTHEAST, NORTH, NORTHWEST, WEST, SOUTHWEST et SOUTH.
- **void setDirection(String targetDirection)**
Même chose, mais en passant le nom de la direction.
- **void setDirection(Direction targetDirection, int time)**
Même chose avec une animation de rotation du personnage pendant `time` ms pour prendre la direction.
- **void setMode(S mode)**
Définit le mode actuel du personnage. Le mode détermine l'animation à jouer.
Les mode de Crusader (Crusader.Mode) sont : WALK, RUN, JUMP, GOTHIT, ATTACK, FALL, DEATH, IDLE, TURN_LEFT, TURN_RIGHT, WALK_LEFT, WALK_RIGHT.
Les modes de Spider (Spider.Mode) sont : WALK, IDLE, ATTACK, FALL, DEATH, TURN_LEFT, TURN_RIGHT.

- **void setMode(String mode)**
Même chose, mais en passant le nom du mode
- **void setFrameRate(S mode, float fps)**
Définit la vitesse d'animation pour un mode donné en nombre d'images par unité d'animation du windowGame. Fixé à 1f, il est inutile de le modifier dans un premier temps, mais vous pouvez produire un effet de ralenti en le modifiant.
- **void setBeginAnimationTrigger(Predicate<ICharacter<?>> callback)**
void setMidAnimationTrigger(Predicate<ICharacter<?>> callback)
void setEndAnimationTrigger(Predicate<ICharacter<?>> callback)
void setTickAnimationTrigger(Predicate<ICharacter<?>> callback)
Définit un déclencheur appelé à chaque "tick" d'animation, ou au début de l'animation, etc. Un déclencheur est une implémentation d'une interface fonctionnelle `Predicate<ICharacter<?>>`. C'est-à-dire une lambda expression `c -> {}` où `c` est le personnage animé et qui renvoie un booléen.
Exemple d'utilisation :

```
Crusader crusader = new Crusader();
crusader.setBeginAnimationTrigger(
    c -> {
        System.out.printf("%s - s 'anime - !\n", c);
        return true;
    });

```

- **void setScale(float scale)**
Définit un facteur d'échelle pour le personnage. inutile dans un premier temps, les définitions par défaut sont correctes. Attention aux arachnophobes si vous grossissez la taille de l'araignée !
- **void setOffset(float x, float y)**
Définit un décalage du dessin pour le personnage. A utiliser conjointement à `setScale` pour recaler le dessin sur la tuile.
- **UUID getId()**
Donne l'identifiant du personnage (utile par exemple pour associer la vue d'un personnage à son équivalent dans le modèle).
Vous remarquerez que c'est le seul "getter" de cette API qui est utile pour construire une vue, mais pas pour contrôler le jeu. Par exemple la position des personnages doit être connue du modèle de votre application, mais n'exploite jamais la position de l'image.

Éléments supplémentaires du décor (inutile dans un premier temps mais bien pour aller encore plus loin)

Décor de nuit

Chaque code multiplié par 10000 donne le même élément de décor dans la pénombre. Multiplié par 20000 dans un décors encore plus sombre.

Décors supplémentaires

Il est possible à l'utilisateur de créer ses propres décors.

Voici comment faire :

1. Constituer un répertoires d'images *.png dont toutes les images sont de même dimension (par exemple 64x64)
2. Fusionner ces images en planches de 10x10 avec la commande `java -cp mazing/lib/mazing.jar fr.ubordeaux.ao.mazing.api.MergePng <repertory> <width> <height> 10 10 <code_base>` Où
 - `<repertory>` est le dossier contenant les images
 - `<width>` est la largeur des images
 - `<height>` est la hauteur des images
 - `<code_base>` est le début des codes (600 par exemple pour laisser les planches 000 à 500 libres)
3. Mettre ces fichiers (all_XXX.png) dans le répertoire lib

Pour exécuter votre programme, ajoutez à *class path* le répertoire où se trouvent les fichiers (all_XXX.png)
Puis dans le code de votre programme, chargez les images à partir de ce fichier.

Exemple complet :

En ligne de commande, pour créer "all_600.png et all_700.png" à partir des fichiers images/png/*.png :

```
java -cp mazing/lib/mazing.jar fr.ubordeaux.ao.mazing.api.MergePng images/png 64 128 10 10 600
> Total fichiers : 145
> Nombre d'atlas générés : 2
> Cellule : 64x128
> Images par ligne : 10, lignes par atlas : 10
> Image générée : all_600.png (640x1280)
> Image générée : all_700.png (640x1280)
> Fusion terminée !
```

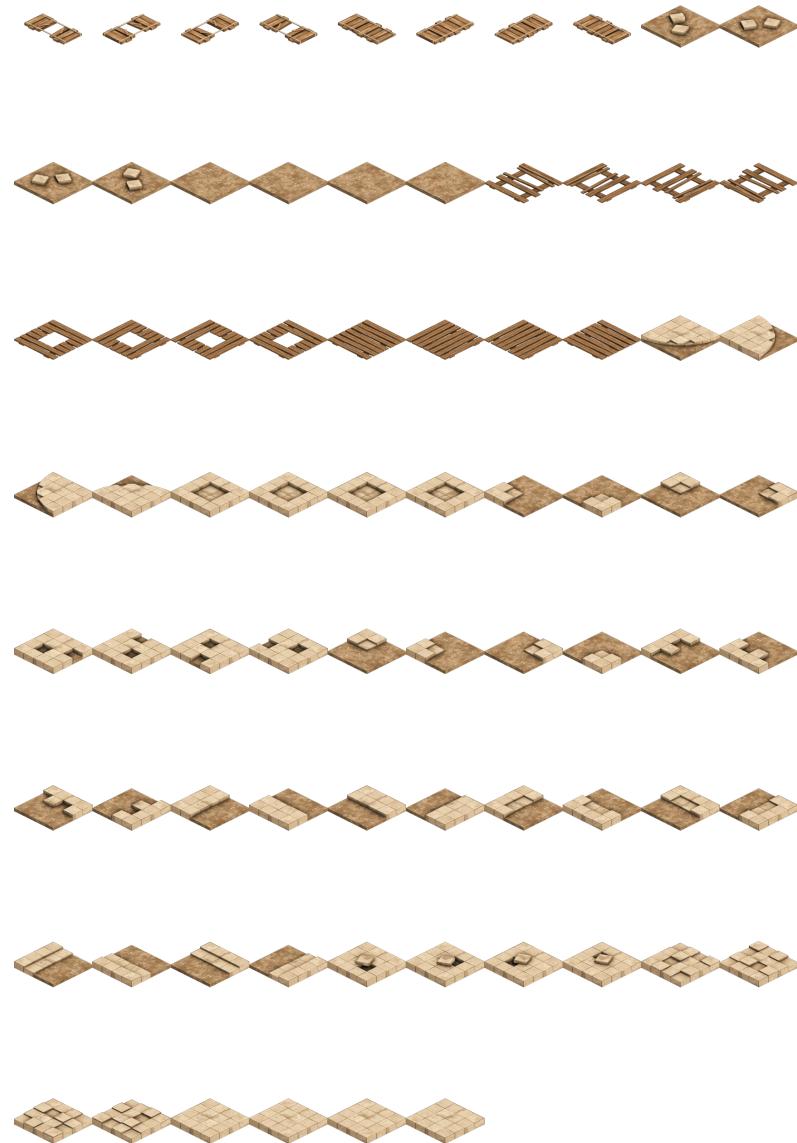
Dans le source du programme :

```
// 64x128 (mettre la taille des images)
// 0.6f : réduire la taille de l'image
// offsetX: 0, offsetY: 30 : décaler l'image pour qu'elle tombe pile sur une tuile
// 600: correspond à la série "all_600.png"
window.initImagesSheet("all_600.png", 64, 128, 0.6f, 0, 30, 600);

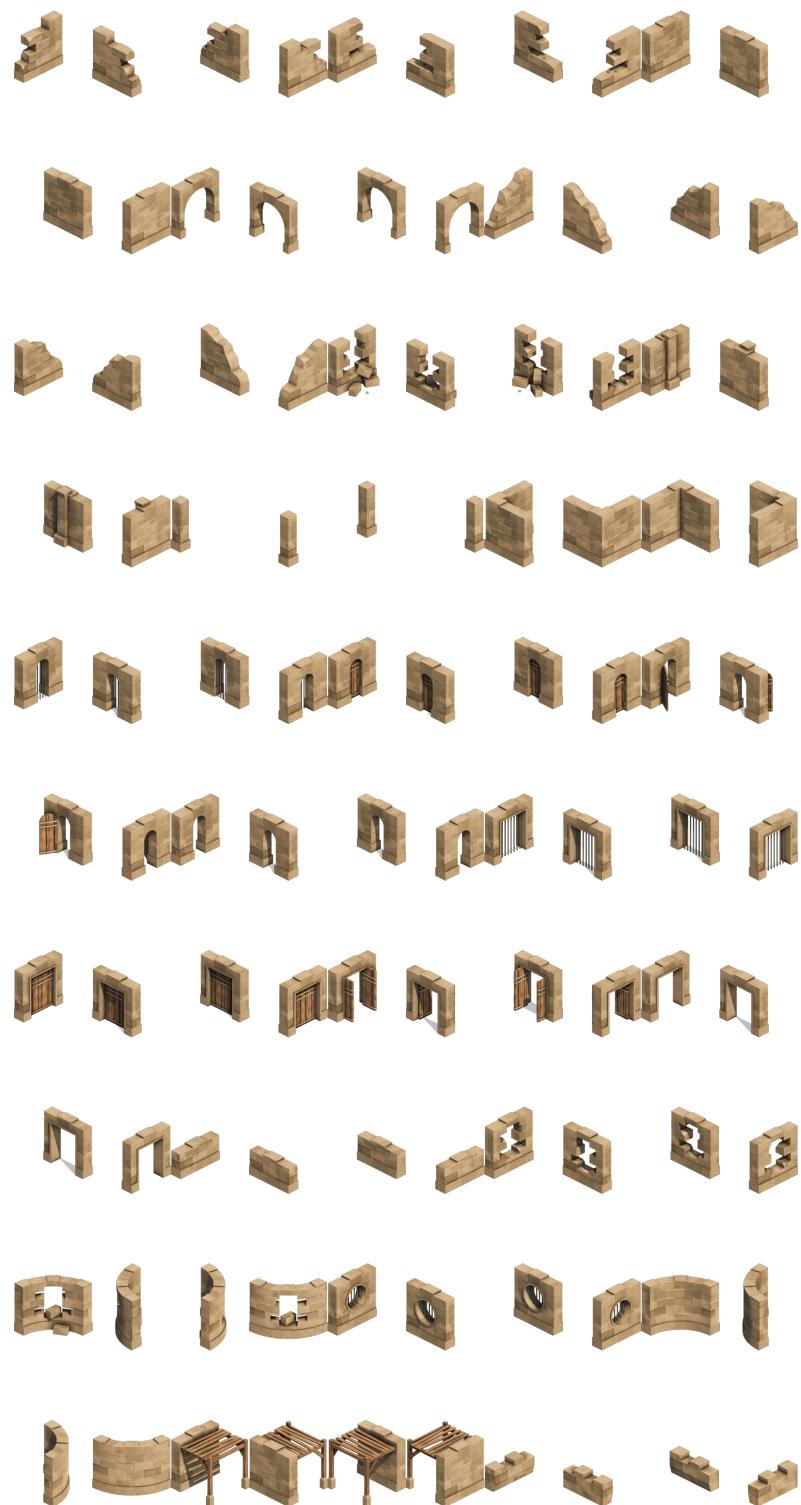
// On peut afficher l'objet 665 en (8, 8, 0)
window.add(665, 8, 8, 0);
```

Annexe – Tables prédéfinies des objets

— Table 000



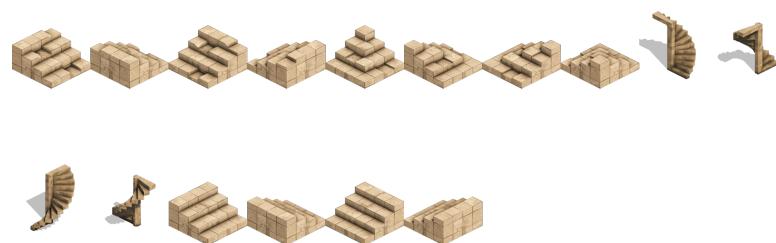
— Table 100



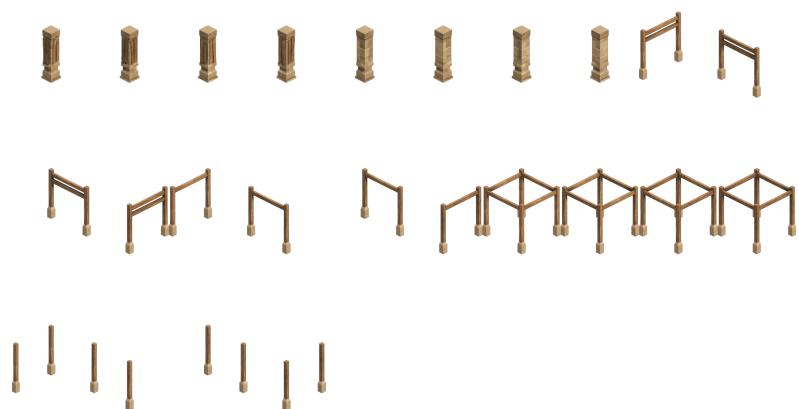
— Table 200



— Table 300



— Table 400



— Table 500

