**JAUROYON Maxime**
**KINDEL Hugo**

# Pet Rescue Saga

L'objectif de ce projet était de réaliser un moteur permettant de jouer à Pet Rescue Saga et comportant une architecture vue/modèle permettant une vue textuelle et une vue graphique, certaines fonctionnalités au cœur du jeu étaient aussi imposées par le sujet.

## 1 – Architecture du projet

```
/
├── src/ : Code source du projet.
│   ├── common/ : Code commun vue/modèle.
│   │   └── print/ : Système d'entrée et sortie en ligne de commande.
│   ├── entity/ : Classes des entités (joueur et robot).
│   ├── level/ : Système de niveau.
│   ├── njson/ : Analyseur de JSON (sérialisation avec lecture/écriture).
│   ├── option/ : Système d'options en ligne de commande.
│   ├── power/ : Classes des pouvoirs (fusée et sabre).
│   ├── util/ : Code utilitaire pour certaines parties du code.
│   ├── view/ : Code du système de vues.
│   │   ├── cli/ : Vue textuelle.
│   │   └── gui/ : Vue graphique.
│   ├── PetRescueSaga.java : Classe principale du jeu.
│   └── Program.java : Classe d'exécution du jeu.
└── data/ : Fichiers de donnée du jeu.
    ├── assets/ : Images, police d'écriture et fichier audio.
    ├── levels/ : Fichier JSON contenant les différents niveaux.
    ├── logs/ : Fichiers de log du jeu (généré après chaque exécution).
    └── settings.njson : Contient les paramètres de l'utilisateur (généré en jeu).
```

## 2 – Modèle du jeu

Le modèle du jeu a été réalisé comme demandé, il existe donc au cœur du jeu un système de niveaux jouables reprenant les mécaniques du jeu d'origine comme base.

C'est-à-dire que nous avons un plateau de jeu, composé de blocs de couleurs (5 couleurs possibles), d'animaux (à sauver), d'obstacles (équivalant à des blocs vides mais non déplaçables) et d'emplacements vides pouvant être remplis. Bien sûr nous avons reproduit le système de tour, qui permet lors de supprimer un bloc, d'enchaîner d'abord une mécanique de gravité qui est suivie lorsque c'est nécessaire d'un décalage des blocs vers la gauche avec finalement la suppression des animaux en bas du plateau. Cela permet donc d'obtenir un niveau jouable qui peut toujours être gagné après un certain nombre de tours. Il est intéressant de noter qu'un bloc de couleur seul ne peux pas être supprimé, ils doivent être groupé au minimum par 2 ! Mais l'usage des pouvoirs est illimité.

Le cœur du modèle est aussi composé d'une classe joueur, qui permet de définir un joueur qui possède un nom, c'est ce joueur en question qui va interagir avec le jeu et ses données seront sauvegardées. Il existe aussi un joueur robot, qui est capable de déterminer le bloc qui lui semble le plus avantageux à supprimer à l'aide d'un système de score. Pour être plus précis, il détermine pour chaque bloc de couleur un score basé sur trois paramètres, le nombre de blocs adjacents qu'il pourrait supprimer, le nombre d'animaux qu'il pourrait sauver et finalement une petite valeur de score aléatoire pour rajouter une certaine complexité à son intelligence. Si le bloc choisi par le robot ne peut pas être

supprimé (car il n'est pas groupé au minimum par 2) alors il utilisera aléatoirement l'un des deux pouvoir existant dans le jeu. Cela donne comme résultat un robot plutôt avancé, qui est capable de déterminer sur le coup un bon mouvement pour avancer sa partie (même s'il ne détermine bien sûr pas plusieurs coups à l'avance).

Il existe aussi deux pouvoirs utilisables, un sabre qui supprime toute une ligne de blocs et une fusée qui supprime toute une colonne de blocs.

Nous avons réalisé 6 niveaux (reproduction des 6 niveaux d'origine du jeu), qui sont tout d'abord très faciles mais dont l'apparition de contrainte permet d'introduire des mécaniques de jeu plus intéressantes. Les niveaux sont enregistrés sur le disque dur au format JSON, permettant d'éditer, charger et sauvegarder des niveaux facilement.

## 3 – Vues du jeu

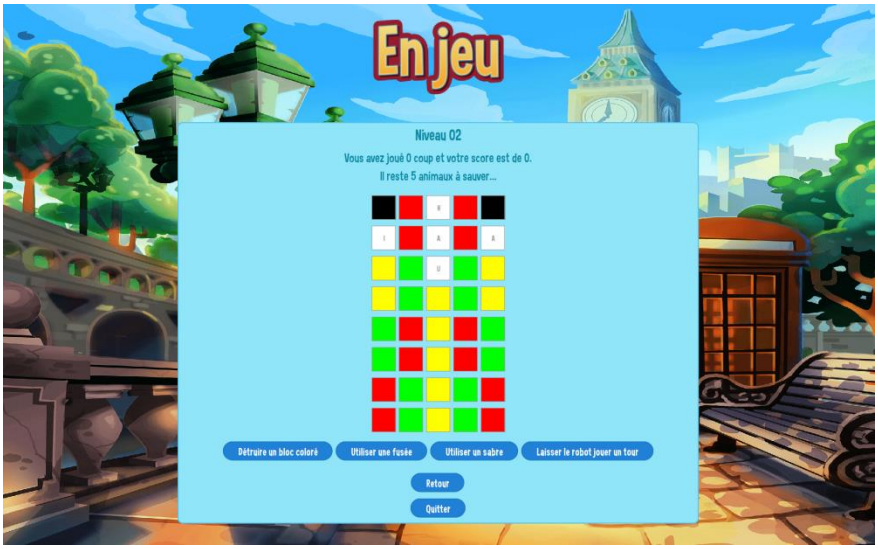
Comme demandé, le projet est composé d'une architecture vue-modèle, donnant une sorte d'indépendances aux vues, qui elles relaient bien sûr énormément sur le modèle. Nous avons donc réalisé deux vues, la vue en ligne de commande (cli) et la vue en interface graphique (gui). Les deux systèmes sont hiérarchisés par le même parent, leur permettant d'avoir un fonctionnement similaire pour garde un code clair, mais si un certain nombre de modifications est nécessaire pour faire fonctionner le programme sous forme de ligne de commande comme sous forme d'interface graphique.

L'interface en ligne de commande fonctionne directement dans le thread principal, un menu est affiché et une fois son affichage terminé on attend la réponse de l'utilisateur, une fois cette réponse envoyée, elle est traitée et le procédé est répété. Une pile est utilisée pour se souvenir de l'ordre des menus et donc pouvoir revenir en arrière.


Comme vous pouvez le voir sur cette image, l'affichage propose le support de l'Unicode et des couleurs, mais cela sera expliqué avec plus de détails dans les bonus du projet.

Nous avons aussi fait en sorte que les menus soient simples à coder à l'aide d'une classe parente qui est capable de simplifier l'affichage du projet et la demande de catégorie à l'utilisateur de manière à ce qu'une classe de menu est juste à posséder l'affichage à faire et la gestion du choix de l'utilisateur.

L'interface graphique est un peu plus complexe même si son procédé est au final le même, il est exécuté par l'Event-Dispatch Thread, ce qui veut dire que le thread principal va finir bien avant la fin du programme si l'on lance l'interface graphique. Ensuite, on affiche le menu en cours et on attend d'avoir un évènement à traiter ce qui est tout à fait similaire à l'interface en ligne de commande et le code est basé sur les mêmes classes parentes.




Cette interface possède quelques spécificités, il est programmé de telle manière qu'il soit possible d'afficher le thème que Swing possède par défaut, ou d'obtenir un thème plus intéressant comme vous pouvez le voir sur l'image précédente, de manière à obtenir un style qui définit mieux notre jeu.

Pour chacune des vues, les menus suivants ont été développé :

- Menu principal : Le premier menu lancé, permet d'accéder à la majorité des autres menus.
- Choix du niveau : Permet de choisir un niveau à jouer.
- En jeu : Permet d'effectuer des actions (supprimer un bloc, utiliser un pouvoir) sur le niveau en cours de jeu.
- Jeu gagné : Permet d'indiquer que la partie est gagné et quel est votre score final.
- Règles : Affiche la liste des règles.
- Crédits : Affiche les crédits du jeu.

Certaines spécificités entre les deux vues existent tout de même, par exemple l'interface en ligne de commande utilise couramment un sous-genre de menu appelé pop-up qui permet non pas de choisir une catégorie de menu mais à indiquer une valeur précise, comme un nom d'utilisateur ou une valeur de case de jeu. L'interface graphique possède lui un menu plus complet de paramètres, permettant de modifier le nom de l'utilisateur, désactiver le son ou changer le thème de l'interface.

Pour finir, le modèle n'est donc absolument pas dépendant des vues, le modèle s'occupe seulement de créer et lancer la vue demandée. Ensuite la vue s'occupera d'appeler elle-même les classes et fonctions du modèle dont elle a besoin.

## 4 – Un exemple de fichier de niveau

```
{
    "name": "Niveau 01",
    "authors": [
        "Hugo Kindel",
        "Maxime Jauroyon"
    ],
    "version": "1.0",
    "columns": 7,
    "rows": 7,
    "backgroundGrid": [
        [0, 1, 0, 0, 0, 1, 0],
        [1, 1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1, 1]
    ],
    "backgroundImagePath": "level_01.png",
    "initialBlocks": [
        [7, 1, 7, 7, 7, 1, 7],
        [2, 2, 3, 3, 3, 4, 4],
        [2, 2, 5, 5, 5, 4, 4],
        [3, 3, 5, 5, 5, 6, 6],
        [3, 3, 5, 5, 5, 6, 6],
        [2, 4, 4, 3, 2, 2, 3],
        [2, 4, 4, 3, 2, 2, 3]
    ]
}
```

```
{
    "name": "Niveau 06",
    "authors": [
        "Hugo Kindel",
        "Maxime Jauroyon"
    ],
    "version": "1.0",
    "columns": 9,
    "rows": 9,
    "groups" : [
        {
            "id": 8,
            "blocks": [2, 3, 4]
        },
        {
            "id": 9,
            "blocks": [4, 5],
            "canRefill": true,
            "refillCondition": {
                "name": "amongColumns",
                "values": ["8"]
            }
        }
    ],
    "backgroundGrid": [
        [1, 1, 1, 1, 1, 1, 1, 0, 1],
        [1, 1, 1, 1, 1, 1, 1, 0, 1],
        [1, 1, 1, 1, 1, 1, 1, 0, 1],
        [1, 0, 0, 0, 0, 0, 0, 0, 1],
        [1, 0, 1, 0, 1, 1, 1, 1, 1],
        [1, 0, 1, 0, 1, 1, 1, 1, 1],
        [1, 0, 1, 0, 1, 1, 1, 1, 1],
        [1, 0, 1, 0, 1, 1, 1, 1, 1],
        [1, 0, 1, 0, 1, 1, 1, 1, 1]
    ],
    "backgroundImagePath": "level_06.png",
    "initialBlocks": [
        [8, 8, 8, 8, 1, 8, 8, 7, 9],
        [8, 8, 8, 1, 8, 8, 8, 7, 9],
        [8, 8, 1, 8, 8, 8, 1, 7, 9],
        [2, 7, 7, 7, 7, 7, 7, 7, 1],
        [1, 7, 0, 7, 9, 9, 9, 9, 9],
        [3, 7, 0, 7, 9, 9, 9, 9, 9],
        [1, 7, 0, 7, 9, 9, 9, 9, 9],
        [3, 7, 0, 7, 9, 9, 9, 9, 9],
        [2, 7, 0, 7, 9, 9, 9, 9, 9]
    ]
}
```

Voici deux exemples de niveau sous format JSON utilisé pour notre projet. Comme vous pouvez le voir, un certain nombre de données sont utilisées, on peut y voir les noms des auteurs ou du niveau, ainsi qu'un numéro de version. Mais surtout, on peut ensuite voir que le nombre de colonnes et de lignes est indiqué, et que cela permet de créer deux tableaux qui vont sur l'un contenir seulement la valeur 0 ou 1 pour indiquer si c'est un bloc du jeu ou si c'est juste un décor. Et sur l'autre obtenir une valeur précise du contenu de la case en question (est-ce un bloque de couleur, un animal ou autre).

Sur le niveau 6, on peut aussi remarquer le système de groupe et de conditions avec la régénération de niveau, qui est une fonctionnalité en plus inspirée des niveaux officiels de Pet Rescue Saga et qui sera expliquée avec plus de détails dans la partie suivante.

Finalement, il est aussi intéressant d'indiquer que ce format de fichier permet d'obtenir un résultat visuel lisible et surtout relativement simple à éditer, permettant comme on pourrait l'imaginer de facilement supporter une sorte de « modding » du jeu d'origine en créant et ajoutant ses propres niveaux au jeu (ce qui d'ailleurs possible en créant un nouveau niveau et en l'ajoutant au fichier json du dossier level qui contient la liste des niveaux à afficher en jeu).

Nous avions décidé d'utiliser ce format de fichier assez tôt dans le développement pour ne pas avoir à réécrire notre code durant le développement pour chaque niveau et donc augmenter notre productivité.

## 5 – Les « bonus » ou codes réalisés non demandé par le sujet

Nous allons expliquer en détail le fonctionnement de chaque système bonus que nous avons implémenté :

- Le système d'options de lignes de commandes :

Nous avons implémenté un système suffisamment complexe à l'aide d'annotations Java et de l'API réflexion nous permettant de rajouter des options au programme très facilement, les « args » fourni à l'exécution du programme seront ensuite directement traités par une fonction qui permettra d'affecter des valeurs à toutes les options qui ont été appelé si c'est nécessaire.
Cela nous permet une certaine flexibilité dans l'ajout de nouvelles options, facilitant notre travail sur le long terme et donnant une impression de code utilisable en dehors du cadre de ce projet pour d'autres projets personnels.

- Le système de log avancé :

Un système de log a été créé, c'est-à-dire qu'au lieu de passer directement par la sortie et l'entrée système proposé par Java pour tout ce qui résulte d'écriture de texte (dans un terminal). Nous avons créé notre propre système de sortie, qui bien sûr fonctionne en réalité avec celui de Java, mais qui permet aussi de créer des fichiers de logs qui vont pouvoir contenir tout ce qui est écrit, ces logs seront enregistrés dans un dossier dans le dossier data du jeu, et ce dossier pourra contenir 8 fichiers de logs maximums à la fois (choix arbitraire), auquel cas les plus anciens seront supprimés. En plus de cela, nous avons dû nous assurer que ce fichier et la sortie supporte bien l'Unicode (plus précisément l'UTF-8) pour nous permettre d'utiliser des caractères spéciaux dans nos écrits au terminal, de manière à utiliser un rendu plus original et plus sympa sur certains menus (principalement le menu en jeu). Un dernier élément plutôt intéressant réalisé dans ce système est que notre programme est capable de réaliser s'il est en train d'être exécuté dans IntelliJ IDEA (l'IDE que nous avons tous les deux utilisé durant tout ce projet) et si c'est le cas le rendu nous aurons une sorte d'affichage hybride entre ce que propose un terminal et ce que propose le fichier de log dans notre projet, nous verrons exactement comme un terminal mais sans jamais que le terminal soit clear.

- Support des codes ANSI de couleur et de clear :

Le support des codes ANSI a été ajouté (seulement pour le support des couleurs et de la commande clear) en association avec notre système de log de manière à ce que je chaque écrit puisse contenir des couleurs et qu'un clear puisse t'être appelé à tout moment. Cela n'était pas toujours facile, surtout avec Windows qui apporte son lot de soucis à ce niveau-là (même si normalement le projet devrait fonctionner avec les codes ANSI sous toutes les versions récentes de Windows, en tout cas avec cmd.exe ou Powershell). Une chose à noté est que bien sûr les fichiers de log ne supportent pas les codes ANSI, alors avant d'écrire nos logs dans le fichier, nous utilisons une expression régulière qui s'occupe de supprimer les codes ANSI d'un texte.

- L'analyseur de JSON (sérialisation, lecture et écriture) :

Nous voulions avoir un format de données puissant, relativement lisible et simple d'utilisation. Le système de sérialisation de Java est simple à utiliser et plutôt pratique mais nous aurions aimé quelque chose de plus utilisé de manière générale. C'est pourquoi nous nous sommes tournés vers le format JSON qui nous le pensons est un format de données plutôt standard aujourd'hui, pas pour toute sortes de tâche certes, mais qui ici peuvent être très utiles. Mais le sujet est fait de telle manière que nous n'avions pas le droit d'importer une librairie. C'est pourquoi nous avons codé notre propre système de sérialisation avec lecture et écriture de JSON. Le nôtre n'est pas le plus complexe et ce n'était pas le but, mais il reste suffisamment puissant pour avoir un système relativement simple qui répond à tous nos besoins et fait bien le travail demandé. Réaliser la lecture de JSON n'est pas si dur étant donné que le JSON ne possède que deux réels types de conteneurs, un dictionnaire (ou une map) dont la clé sera toujours une chaîne de caractères et un tableau. Nous savons aussi que les valeurs possibles n'auront

que quelques types, entier, caractère, chaîne de caractères, etc. Nous avons donc écrit notre propre analyseur de JSON c'est pourquoi nous avons réalisé un algorithme assez commun pour ce genre de programme, un analyseur descendant récursifs dont chaque procédure d'analyse sont mutuellement récursive pour gérer chaque partie possible de la grammaire du langage. L'écriture est bien plus simple, grâce à la facilité d'obtenir un type sous forme de chaîne de caractères que propose Java, il n'était pas compliqué de directement l'écrire de manière assez brute. Finalement, la sérialisation est aussi possible à l'aide de l'API réflexion de Java et des annotations.

- Le thème de l'interface :

Pour l'interface graphique, le style par défaut proposé par Swing ne nous satisfaisait pas, bien que la logique de l'interface fonctionnât, nous avions donc décidé d'implémenter notre propre sorte de thème. Java Swing n'est pas par définition personnalisable, en tout cas ce n'est pas le but recherché de la librairie. Nous avons donc dû réaliser nos propres boutons, label et panel qui possédait quelques spécificités pour pouvoir entre autres changer la police d'écriture et arrondir les bords. Nous avons aussi ajouté des images de fond aux fenêtres pour styliser. Finalement, pour une question d'ambiance et parce que cela rajoutait un petit quelque chose, nous avons pris une musique d'ambiance du jeu d'origine que nous jouons en boucle dans l'interface graphique (elle peut bien sûr être désactivée dans le menu paramètres, qui arrêtera la musique et sauvegardera vos données pour ne pas relancer la musique au prochain démarrage non plus). La majorité des images utilisées (les images de fond de fenêtre) ainsi que la musique dépendent bien sûr de droits d'auteur de King et sont utilisées seulement à but éducatif. La police d'écriture est elle libre d'accès mais nous préférons précisé que nous ne l'avons pas réalisé. Le reste des images, comme les titres de pages ou le logo du jeu ont été réalisé par nous mais avec toujours une très grande reprise du style du jeu d'origine.

- Le système de groupe pour les niveaux :

Après avoir analysé les niveaux d'origines du jeu pour en comprendre la logique, nous avons pu voir que certains blocs de couleurs étaient générés aléatoirement mais que des paternes étaient toujours visibles. C'est-à-dire que des groupes de blocs était créé, donc si un ID de groupe était utilisé pour définir un bloc dans un niveau alors un bloc de ce groupe serait choisi de manière aléatoire. Cela permet de rajouter un côté aléatoire dans les niveaux, pour ne pas avoir toujours les mêmes niveaux lorsqu'ils sont lancés mais tout en gardant la possibilité de définir certaines parties du niveau comme voulu, et nous avons donc créé un système similaire.

- Le système de régénération de niveaux :

Si nous reprenons l'exemple du niveau 6 que nous avons fait, il y a une chose qui le différencie des autres niveaux, que ce soit sur notre version ou celle du jeu d'origine, il est capable de se « régénérer » en faisant tomber des blocs par le haut du niveau lorsque c'est possible pour faire en sorte qu'il y a toujours des blocs à utiliser. Ce système pourrait être aussi simple, mais comme nous avons vu dans le jeu d'origine il existe un système de condition, sur le niveau 6 par exemple, un bloc peut être généré seulement sur une colonne précise, et il ne peut pas être généré sur une autre colonne si le haut est vide. C'est pourquoi nous avons ajouté à notre système de groupes un système de conditions qui permettent de définir si un groupe peut être régénéré et si oui quelle condition de régénération il possède (actuellement le jeu possède seulement une seule condition qui est de vérifier un numéro de colonne précis).

## 6 – Problèmes rencontrés et pistes d'extensions

Durant le développement du projet nous avons bien sûr rencontré quelques problèmes, tout d'abord nous avons dû apprendre à travailler ensemble et cela s'est heureusement très bien passé alors nous avions vite pu nous mettre au travail en nous répartissant les tâches.

Ensuite, l'interface n'a été implémenté que très tardivement, en majorité dû au fait que le cours nécessaire ainsi que les TP sur le sujet ne sont arrivés que très tard dans l'année mais nous nous sommes adaptés en conséquence.

Mais encore, certaines parties du code n'étaient vraiment pas claires avec l'avancement du projet, un gros travail de réécriture a été nécessaire en cours de route, et nous a permis au final d'avoir un code dont nous sommes plutôt contents et qui respecte bien les choses demandées par le sujet.

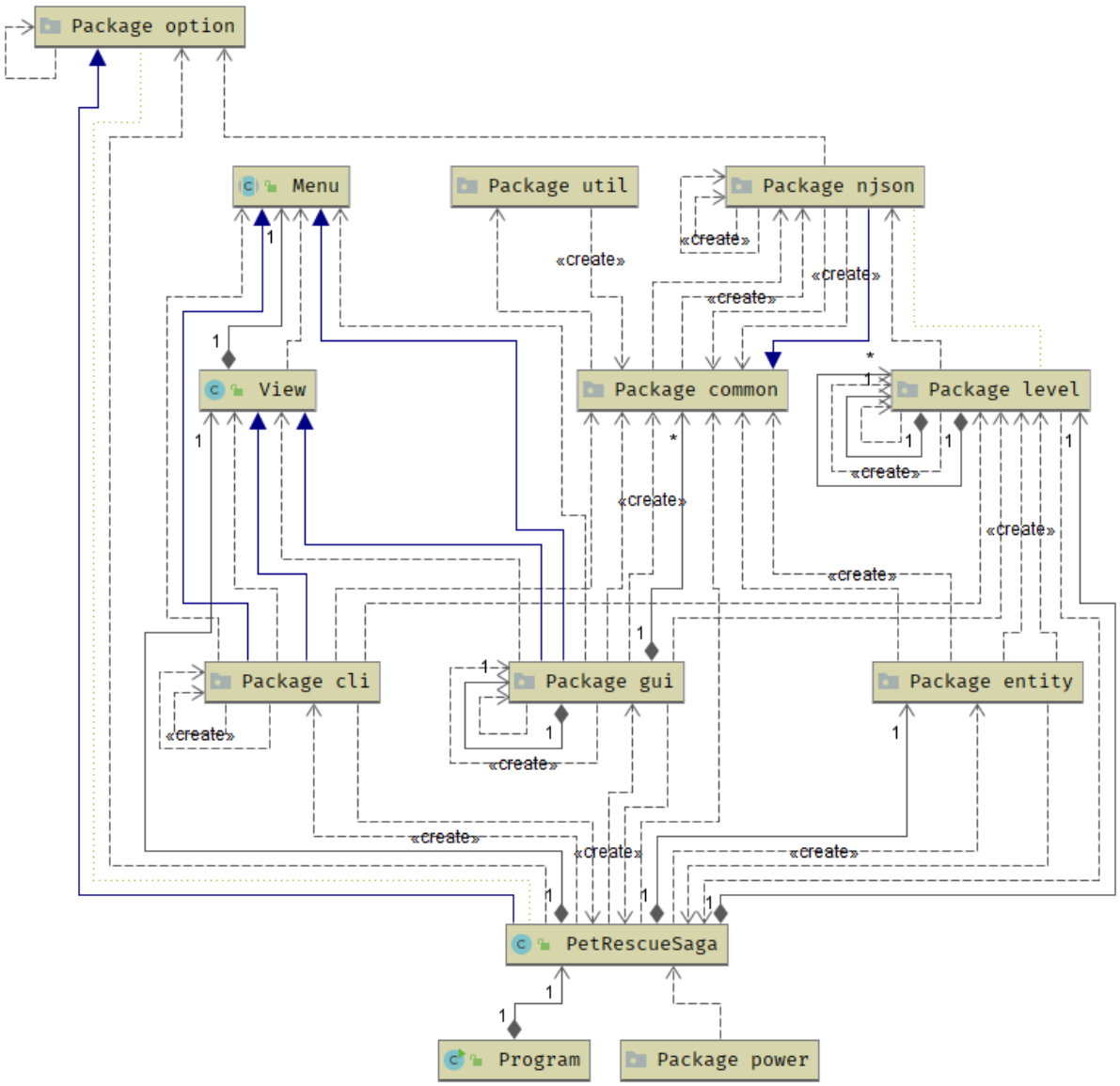
Le nombre de pistes d'extensions qui n'est pas implémenté dans le projet est sans limite ! Ce type de projet permet vraiment de laisser aller son imagination et d'en tirer un grand nombre d'idée, voici quelques-unes des idées que nous avions en tête :

- Complexifier le système de destruction des blocs en ajoutant des limitations (comme améliorer le système de gravité pour le rendre unique ou plus proche du jeu d'origine, améliorer le système de groupe, d'aléatoire et de conditions qui est toujours très simpliste pour le moment).
- Améliorer les différentes vues, en ajoutant des menus utiles ou en permettant le support de plus de thème (en plus d'améliorer les thèmes déjà présents), ainsi que de posséder uniquement des fichiers de données non sujets à des droits d'auteurs.
- Ajouter la possibilité d'avoir des niveaux qui vont vers le bas (des longs niveaux), comme il est présent dans le jeu d'origine.
- Ajouter un réel éditeur de niveau, ce qui pourrait être très intéressant avec une bonne interface graphique.
- Améliorer les mécaniques de jeux avec un système d'étoiles ou de sauvegarde des scores et niveaux joués.
- Améliorer la gestion d'erreur ou introduire un système de test unitaire.

Nous n'allons pas continuer d'étendre cette liste, mais vous l'aurez compris ce n'est pas les idées qu'il manque mais comme vous le concevez bien, en tant que programmeur le temps est une ressource rare et même si nous aurions aimé développer plus de fonctionnalités, nous ne sommes que trop limités par le temps.

## 7 – Schéma UML du projet

Voici un schéma UML du projet (relativement simplifié par paquet pour éviter qu'il soit trop gros), qui permet de rendre assez clair la séparation entre le modèle les deux vues.
Une version plus complète est fournie avec le projet si nécessaire.




Légende :

| | |
|---|---|
| ——→ | Relation d'extension d'une classe. |
| - - - - - - - - → | Relation d'importation de code. |
| - - - - - - - - | Relation d'usage d'annotations Java. |
| ◆——→ | Relation de propriété complète (une classe sert de propriété à une autre). |
| «create» | Relation de création d'instance d'une classe. |