Manuel d'architecture

Architecture des différentes classes:

Main.java :

Pour commencer, nous avons notre classe *Main.java* qui va s'occuper d'exécuter l'application. Pour cela, nous créons, à partir des différents arguments que l'utilisateur nous donne, une liste des fichiers.txt qui sont des niveaux.

Nous envoyons le nom du fichier à la classe *LevelParser* qui va créer un nouveau *LevelManager*.
Puis nous créons l'interface graphique du niveau grâce à l'interface *GameView* qui appelle la classe *BabaGameView* pour afficher le niveau tant qu'il n'y a pas victoire ou défaite du joueur.

LevelParser.java :

Cette classe a pour but de parser le fichier (cf annexe) du niveau, et créer les classes qui vont permettre de représenter le niveau, c'est-à-dire un *LevelManager* et un *LevelDesign*.

LevelDesign.java :

Elle représente la banque d'image du niveau, elle va stocker dans une hashmap le BabaEntity comme clé, et son ImageIcon comme value.

GameView.java :

Interface appelée par le Main qui permet d'afficher une view (victoire, défaite, le plateau) du jeu.

BabaGameView.java :

Classe permettant de gérer l'affichage du jeu. Elle possède deux champs xorigin et yorigin pour définir le point de début de l'affichage du plateau. Deux autres champs lengthsize, widthsize pour le stockage des tailles de l'écran de l'utilisateur. Et un dernier champ squaresize, calculé en fonction des quatres autres champs pour définir la taille d'une case graphique dans le plateau de jeu.

LevelManager.java :

Elle permet de gérer le niveau. Elle stocke le nombre de colonnes et de lignes du niveau dans des champs. L'ensemble des BabaEntity sont stockés dans une liste. Et elle possède un *RuleManager.*
Elle va pouvoir ajouter, enlever des coordonnées à des éléments. Gérer le déplacement des éléments…

RuleManager.java :

Cette classe gère l'ensemble des règles du plateau de jeu. Elle possède une liste de *Rule.*
RuleManager peut détecter toutes les règles du plateau, et les faire appliquer, c'est-à-dire, envoyer la propriété de la règle à l'élément en question.

Rule.java :

Une règle a trois champs *Word*, un pour le nom, le deuxième pour l'opérateur, et le dernier pour la propriété ou pour le deuxième nom (Dans le cas d'une règle du type rock is lava).
Une règle peut récupérer l'élément associé au nom de la règle.

BabaEntity.java :

Cette classe est une interface, pour distinguer un *BabaElement* à un *Text*. Elle comporte des méthodes par défaut, car *BabaElement* et *Text* réagissent à peu près similairement. Notamment au niveau de l'application des règles.

BabaElement.java :

Elle représente un élément du niveau (comme Rock, Lava, Baba, Wall ….).
La classe *BabaElement* possède une liste de *Coordinate*, qui stocke où se situe l'élément, une liste de *PropertyEnum*, les propriétés qui définissent l'élément, et un *NounImgEnum* qui définit le type de l'élément.

Text.java :

Elle représente un élément du niveau qui est du texte. La classe *Text* possède une liste de coordonnées, qui stocke où se situe le texte, une liste de *PropertyEnum*, les propriétés qui définissent le texte (par défaut "push"), et un *Word* qui définit le type du texte.

NounImgEnum.java :

Enum des différents éléments possible dans le jeu. La classe a un champ string : le fichier qui correspond à l'affichage de l'élément et un autre champ int : qui correspond à l'opacité de l'image.

Word.java :

Sealed interface qui autorise uniquement *NounTextEnum*, *OperatorEnum* et *PropertyEnum*. Permet de définir ces trois classes comme un mot.

NounTextEnum.java :

Enum des différents noms possible dans le jeu. La classe a un champ string : le fichier qui correspond à l'affichage du texte.

OperatorEnum.java :

Enum des différents opérateurs possible dans le jeu. La classe a un champ string : le fichier qui correspond à l'affichage du texte.

PropertyEnum.java :

Enum des différentes propriétés possible dans le jeu. La classe a un champ string : le fichier qui correspond à l'affichage du texte.

Coordinate.java :

C'est un record qui permet de définir une coordonnée avec deux champs de type int i et j. Nous l'utilisons pour le plateau de jeu.

Améliorations/corrections depuis la soutenance bêta:

Avant la soutenance bêta, nous étions partis dans une toute autre direction architecturale. Nous n'avons pas amélioré, ni corrigé quelques parties de notre code, mais plutôt l'intégralité du projet.

En effet, nous avions qu'une classe pour définir un élément : BabaGameCell. Avec comme champs une multitude de boolean qui permettaient de définir si l'élément était de telle propriété, ou non. Nous avions autant de champs boolean que de propriétés dans le jeu.

De plus, une BabaGameCell était représentée comme une liste chainée, et donc avait un champ otherBabaGameCell. Pourquoi ? Car nous avions une seule classe BabaGameData qui permettait de gérer le niveau. BabaGameData avait pour champ un tableau à deux dimensions qui stockait un BabaGameCell… voilà pourquoi BabaGameCell était une liste chainée.

Cela entraînait une multitude de méthodes complexes et récursives, car nous traitions avec des éléments chaînés.

Nous avons repris l'intégralité de notre code afin d'essayer de respecter au mieux les principes de la programmation objet.

Annexe:

La première ligne sert à connaître le nombre de colonnes et de lignes.
Puis, une entité suivie de ses coordonnées ligne par ligne.
Voici l'architecture des fichiers que lit la classe *LevelParser* :

---

nombreDeLigne nombreDeColonne

(nt | ni | o | p) EnumValue
xCoord yCoord
xCoord yCoord
.. ..
(nt | ni | o | p) EnumValue
xCoord yCoord
xCoord yCoord
.. ..

---

nt = NounTextEnum
ni = NounImgEnum
o = OperatorEnum
p = PropertyEnum