Manuel du développeur

Table des matières

Description

Ce fichier contient les différentes explications à avoir sur la partie développement du jeu. Il contient des explications sur les choix que nous avons fait et sur les différentes méthodes que nous avons utilisés. Il contient aussi les changements qu'il y a eu depuis la soutenance bêta que nous avons eu en décembre.

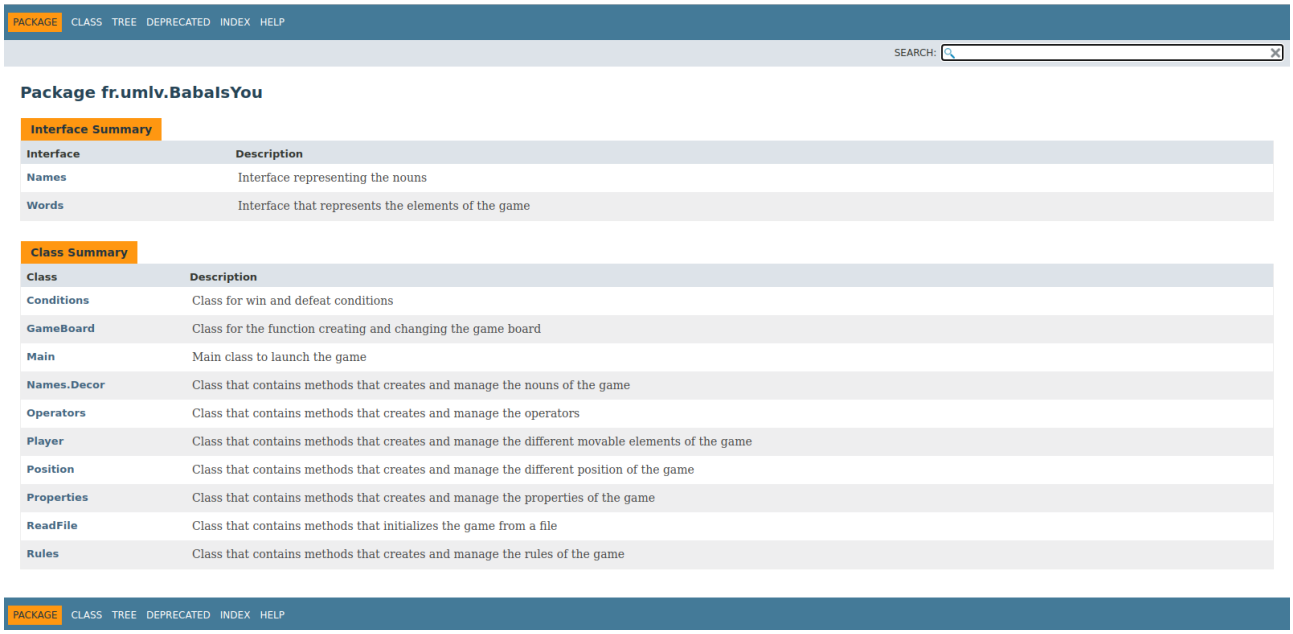
Javadoc

Je tiens à rappeler qu'une Javadoc existe pour ce projet et qu'elle peut être générer au souhait de l'utilisateur du programme. Nous allons vous rappeler la démarche à suivre pour cela.

Ouvrez un terminal et placez vous dans le dossier du projet. Tapez ensuite la commande **ant Javadoc** pour générer la documentation du projet dans le dossier doc du dossier docs.

Pour lire la documentation, ouvrez le dossier docs puis le dossier doc. Ensuite ouvrez le dossier du nom **fr,** puis **umlv** et **BabaIsYou.** Cliquez sur la documentation de la classe que vous voulez.

Une page HTML va s'ouvrir sur votre navigateur et vous pourrez parcourir la documentation à l'aide des différents menus.



Architecture

Notre projet se décompose en plusieurs classes et interfaces qui contiennent les différentes fonctionnalité du jeu. Les classes et interfaces se situent dans le package fr.umlv.BabaIsYou.

La classe Conditions

Cette classe contient quatre méthodes servant à déterminer quand un joueur a gagné ou perdu une partie.

Un joueur réussit un niveau lorsque le ou un des joueurs qu'il contrôle est sur l'élément du jeu permettant de gagner, celui associé à *Win*. La méthode *victoryCondition* permet de vérifier cette condition.

Un joueur échoue à un niveau si il ne contrôle plus aucun éléments du jeu, si plus rien n'est associée à *You* ou si tous les éléments contrôlables ont été détruit par la propriété *Defeat. La méthode defeatConditions* permet de vérifier cette condition.

La classe GameBoard

Cette classe est l'une des classes les plus importantes du projet. Elle contient en effet les méthodes représentant le codage des propriétés, la méthode qui applique les propriétés au jeu ainsi que d'autres méthodes nécessaires au jeu.

Cette classe représente le jeu en lui même qui est composé d'éléments jouables et non-jouables ainsi que les règles présentent dans le jeu.

Parlons rapidement de la méthode *applyRules* qui est une méthode essentielle au jeu, elle permet d'appliquer les règles valides présentes dans le jeu à l'instant t. Nous avons utilisé un switch afin d'appliquer la bonne règle au bon moment.

La classe Main

Cette classe est la classe permettant de lancer le jeu et de le dessiner. On y initialise toutes les variables nécessaires pour le bon déroulement du jeu.

L’interface Names

Cette interface définit le type Names qui correspond aux noms du jeu (Baba, Wall, …) et c’est une extension de l’interface Words.

La classe Decor

Cette classe représente les noms du jeu et implémente l’interface Names. Un élément de type Decor est composé d’un nom sous forme de chaîne de caractère (String), d’une position de type Position, d’un attribut *isMelt,* et d’une image de l’élément et celle de son texte associé.

Chaque élément est vérifié avant d’être crée afin d’éviter de créer un élément qui n’existe pas dans le jeu.

La classe Operators

Cette classe représente les opérateurs du jeu et implémente l’interface World . Un élément de type Operators est composé d’un nom sous forme de chaîne de caractère (String), d’une position de type Position et d’une image de son texte associé.

Chaque élément est vérifié avant d’être crée afin d’éviter de créer un élément qui n’existe pas dans le jeu.

Pour l’instant seul l’attribut **Is** est disponible mais à l’avenir d’autre opérateurs pourront venir s’ajouter à la liste.

La classe Player

Cette classe représente tous les éléments qui sont liés à la propriété *You* c’est-à-dire tous les éléments qu’on peut contrôler, ainsi que leur(s) positions respectives.

La classe Position

Cette classe représente une position d’un élément du jeu.

Une position est représenté par sa coordonnée horizontales et sa coordonné verticale.

On vérifie au préalable si une position est valide.

La classe Properties

Cette classe représente les opérateurs du jeu et implémente l’interface World . Un élément de type Properties est composé d’un nom sous forme de chaîne de caractère (String), d’une position de type Position et d’une image de son texte associé.

Chaque élément est vérifié avant d’être crée afin d’éviter de créer un élément qui n’existe pas dans le jeu.

La classe ReadFile

Cette classe sert à la lecture de fichier de configuration des niveaux ainsi qu'à la lecture des options *--level name* et *--levels name*.

Nous utilisons le type File dans cette classe pour manipuler les fichiers et pour lire leur contenu on utilise un type Scanner.

La classe Rules

Cette classe est aussi une classe importante, elle permet de définir une règle, de contrôler les règles du jeu en temps réel et de de récupérer une règle entrée avec l'option *--execute.*

Une règle est constitué d'un nom, d'un opérateur et d'une propriété. Une règle doit se lire de gauche à droite ou de haut en bas pour être valide.

Parlons rapidement de la méthode *pickRulesFromBoard* qui permet de récupérer les règles valides du jeu en temps réel. Elle change le champs correspondant aux règles du type GameBoard en y ajoutant les règles toujours valides et en enlevant les règles qui sont devenues invalides.

L'interface World

L'interface regroupe les types représentant les noms, les opérateurs et les propriétés en un seul « super » type ainsi que des méthodes communes à ces classes.

Améliorations/Corrections depuis la soutenance bêta

Depuis la soutenance, nous avons d'abord entièrement terminer de coder les propriétés du jeu tout en veillant à respecter les factorisation de code possibles que le professeur nous a fait remarquer.

Le jeu a été entièrement terminé tout en respectant les consignes de l'énoncé.

À l'issue de la soutenance, nous avons essayer au maximum de ne pas réécrire des méthodes qui font à peu près la même chose que d'autres méthodes déjà écrites.

Nous avons aussi optimiser certaines méthodes qui étaient un peu longue à l'exécution. Par exemple, la méthode *pickRulesFromBoard* a été clairement revu et optimiser. En effet, elle faisait certaines choses plusieurs fois et inutilement, nous avons régler ça afin qu'elle ne fasse que ce qui est vraiment utile au programme, c'est-à-dire mettre à jour les règles valides du jeu.

Nous avons aussi écrit toutes nos méthodes en 20 lignes ou moins.

Les dimensions du jeu ont été revu afin de pouvoir jouer en plein écran sur n'importe quelle écran.