Université Gustave Eiffel

License Informatique

Rapport Technique de Développement

Java: Baba Is You

3ème Année

Janvier 2021

Auteurs

- Jimmy TEILLARD
- Lorris CREANTOR

Contents

1 Introduction

1.1 Français

Ce rapport technique a pour but principal de décrire le projet suivant ; c'est à dire une description de l'architecture du projet, nos choix d'implémentation ainsi que les difficultés que nous avons pu rencontrer au cours du développement.

Le but de ce projet à été de créer un clone du jeu Baba Is You de Arvi "Hempuli" Teikari en Java.

1.2 English

The main purpose of this technical report is to describe the following project; that is to say a description of the project architecture, our implementation choices as well as the difficulties that we may have encountered during development.

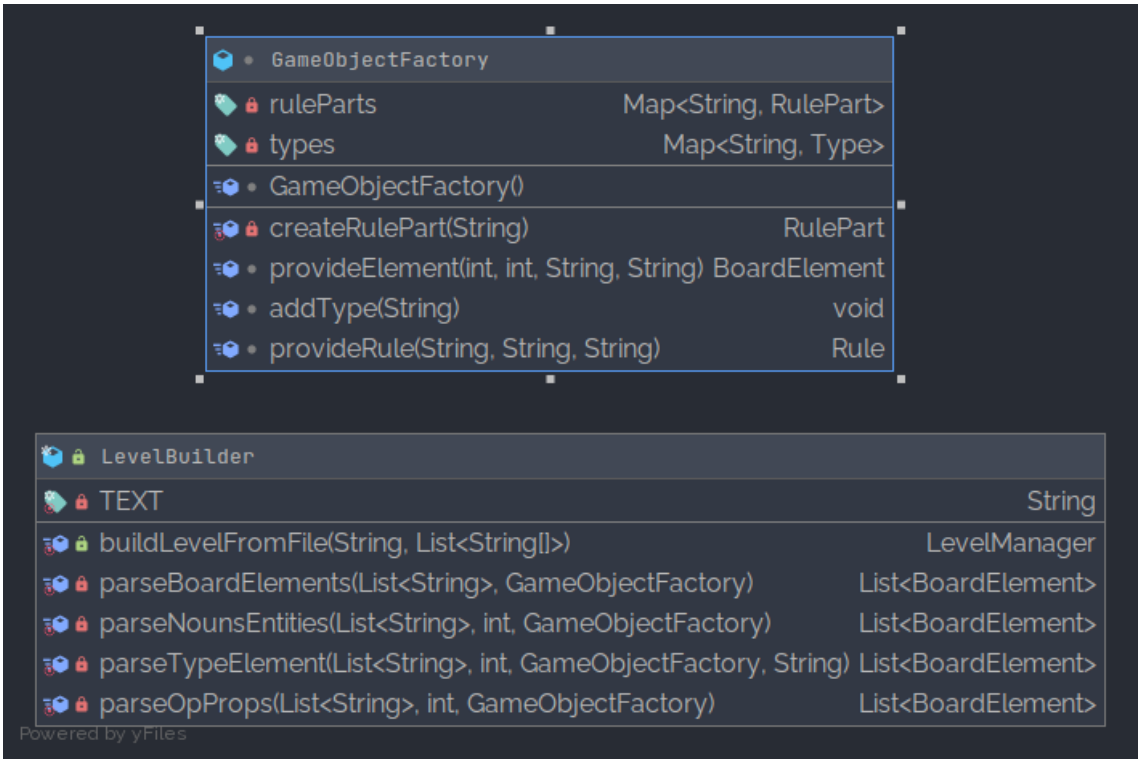
The goal of this project was to create a clone of the game Baba Is You by Arvi "Hempuli" Teikari in Java.

2 Architecture

2.1 Structure globale

- Les sources du programme se situent dans le package `com.notkamui.javaisyou` dans le dossier `src` :
  - le package `app` contient le point d'entrée de l'application ainsi que le contrôleur du jeu
  - le package `engine` contient le moteur principal du jeu dont l'état est contrôlé afin de faire se dérouler le jeu et faire les opérations nécessaires
  - le package `utils` contient des classes utilitaires à la création des niveaux et des entités de manière contrôlée et sécurisée.
- Les ressources du projet (niveaux et images) sont gardées dans un dossier `resources`

2.2 LevelBuilder et GameObjectFactory



`LevelBuilder` ne contient qu'une seule méthode publique qui se charge de construire un niveau à partir d'un nom de fichier ainsi que des règles (execute) données sous forme de String. En effet, l'intérêt d'avoir une classe qui se charge de cela, couplé à `GameObjectFactory`, permet d'être sûr que chaque type n'est créé qu'une seule fois pour tout le niveau. (Tout les BoardElement de même type ont la même instance de Type).

Le fichier donné est parcouru de haut en bas suivant des déclarations de type suivit de leurs instances

- `o NAME` correspond à un opérateur
- `p NAME` correspond à une propriété
- `n NAME` correspond à un "nom"

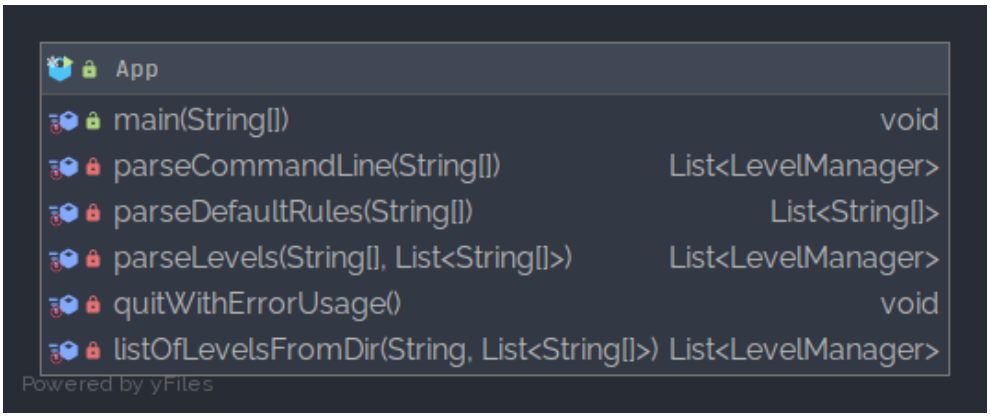
Chaque déclaration de type est suivit par une ou plusieurs lignes commençant par un chevron décrivant les coordonnées x et y d'une instance à créer. Dans le cas d'un "nom", deux options sont ajoutées: 't' (pour text) ou 'e' (pour entity) décrivant si l'instance à créer est un type de nom sous sa forme de mot ou sa forme d'entité, suivit de ses coordonnées.

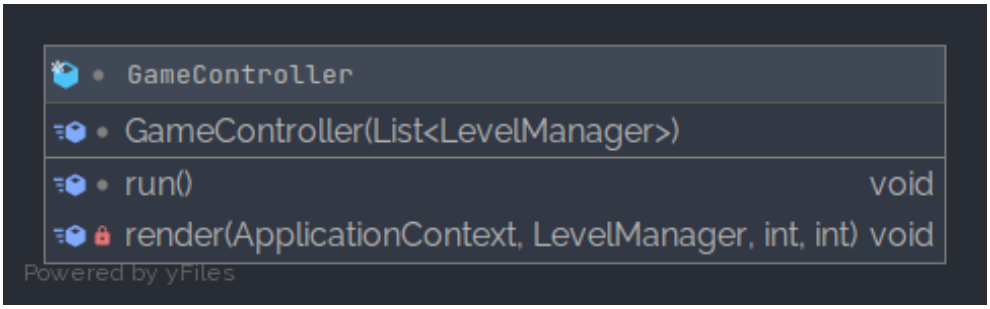
A chaque nouveau type rencontré qui n'est pas encore stockée dans le `GameObjectFactory`, il est créé, sinon il est relié afin de garder la cohérence des types. Ce système permet entre d'autres d'ajouter très facilement des nouvelles propriétés et opérateurs (il suffit de définir un nouveau type dans l'interface correspondante et de l'ajouter à la `Map` de `GameObjectFactory`). Il n'y a pas besoin de créer de classe pour les types d'entité ou de nom: elles sont définies par leur instance unique.

De la même facon, le `GameObjectFactory` peut créer des règles (qui seront les règles par défaut du niveau) avec les types déjà créés ou en en créant de nouveaux.

2.3 App et GameController

Ces deux classes servent à lancer le jeu ainsi que mettre à jour chaque à chaque frame l'état du niveau courrant.

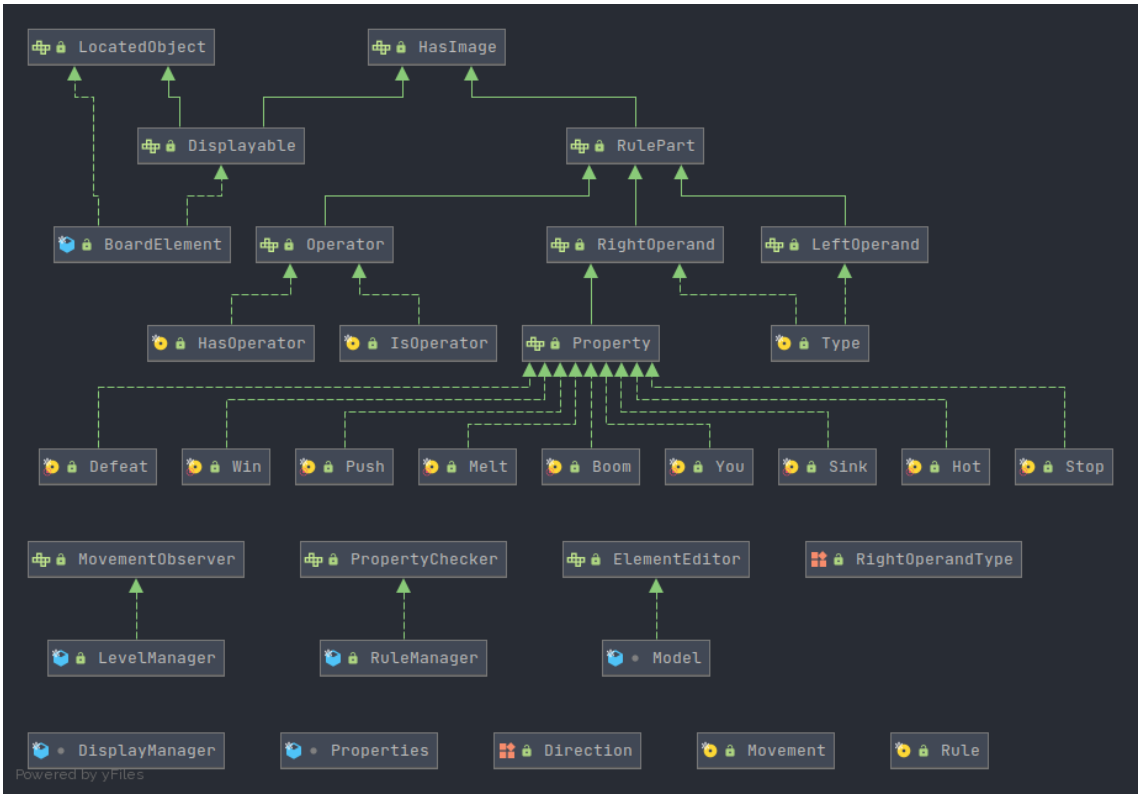




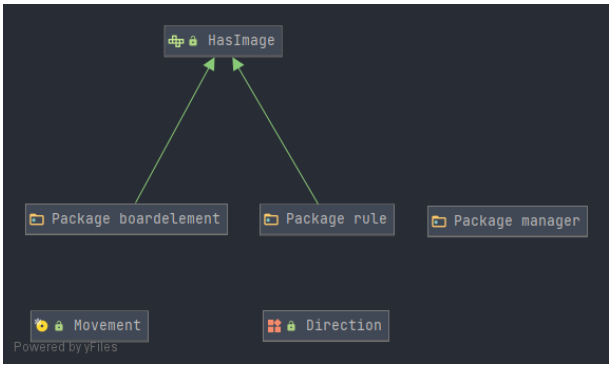
`App` lit les arguments en ligne de commande afin de demander au `LevelBuilder` de créer chaque niveau.

`GameController` attends à chaque frame l'appui d'une touche par l'utilisateur et charge de déplacer toutes les entités YOU ainsi que de mettre à jour les règles du plateau

2.4 Le package engine

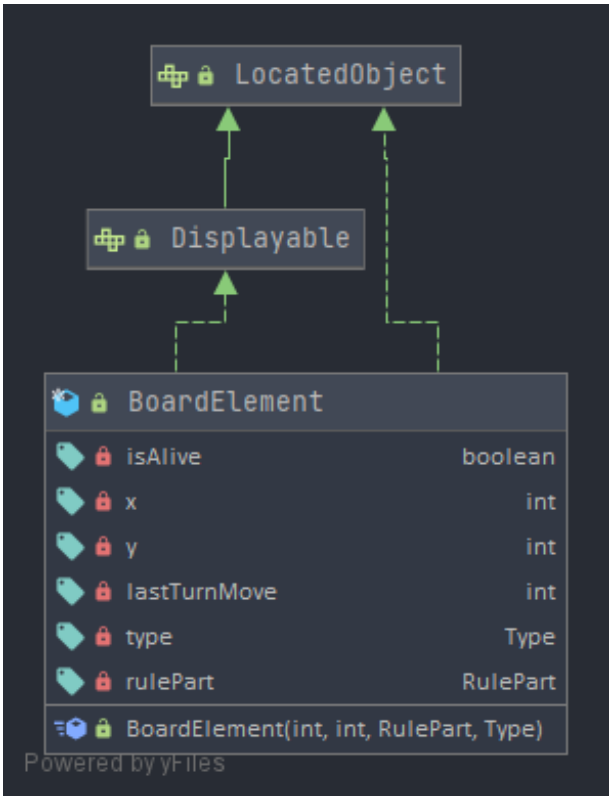


Le package `engine` est le package qui comme son nom l'indique, gère le déroulement d'une niveau de Baba is You. Il se divise en trois sous packages, le package `boardelement`, le package `rule` et enfin le package `manager`. Il contient également une Enum `Direction` représentant les 4 directions cardinales, un record `Movement` représentant un vecteur de mouvement et une interface `HasImage` représentant un objet qui possède une image.



2.4.1 Package board

Le package `boardelement` est le package qui contient les classes et interfaces servant à représenter les éléments du plateau d'un niveau de Baba Is You. Il contient ainsi deux interfaces, la première, `LocatedObject` représentant simplement un élément avec des coordonnées x et y, la seconde `Displayable`, héritant de `HasImage` et `LocatedObject`, qui représente donc un objet affichable. Ces deux interfaces sont implémentées par la classe `BoardElement`, classe représentant les éléments du plateau. Cette classe a pour responsabilité de connaître ses coordonnées, son état (vivant ou mort) ainsi que sont type (baba, wall, water ...). Il contient également un champ lastTurnMove servant à afficher les éléments du plateau dans l'ordre de dernier déplacement. Enfin le champ rulePart contient une instance unique du RulePart représenté par le `BoardElement`, propriété, opérateur, type ou dans le cas des entité, rien.



2.4.2 Package rule

Le package `rule` est le package contenant toutes les classes entrant gérant la représentation et le fonctionnement d'une règle (suite d'au moins 3 mots) de Baba Is You.

Le fichier principal du package est le record `Rule` représentant une règle, formée de 3 éléments, une opérande gauche `leftOperand` un opérateur `operator` et une opérande droite `rightOperand`. Une `Rule` possède également les méthodes servant à effectuer les actions engendrées par l'application de cette dernière.

Le package contient également l'interface `RulePart` représentant une partie de

règle, soit l'une des trois interfaces héritant de `RulePart` et entrant dans la composition d'une instance de `Rule` à savoir une `LeftOperand`, `Operator` et `RightOperand`. Ainsi, l'interface contient 3 méthodes servant à récupérer la représentation de la `RulePart` sous ses 3 formets possibles ainsi qu'une constante représentant la `RulePart` nulle et implémentant les comportements par défaut (utilisée par les **BoardElements** représentant une entité et ne contenant donc pas de rulePart effectif) et retournant pour chaque méthode les constantes "nulles" propres à chacune Interface et implémentant par défaut les méthodes des-dites interfaces.

Les trois interfaces héritant de `RulePart` ont donc un constante statique et des méthodes propres, `LeftOperand` qui représente l'objet qui suit la règle possède une unique méthode `getAsType` servant à interpréter l'objet en une instance de `Type`. L'interface `Operator`, qui est en charge de transmettre ou non les l'application de la règle à l'opérande droite. Enfin l'interface `RightOperand` représente la partie qui va appliquer la règle, elle possède donc les méthodes d'actions selon l'événement.

2.4.3 Package property

Le fichier principal de ce package est l'interface `Property` représentant les propriétés. En effet les propriétés sont aussi des `RightOperand`, dont la responsabilité est de connaître leur type et leur image. Chaque propriété peut alors implémenter une ou des méthodes parmi `onMove`, `onRuleCreation` ou `onSuperpositon` suivant l'effet qu'elle prend et quand (on parle alors de propriétés immédiates, passives et actives). Etant une sealed interface, les différentes implémentations des propriétés sont stockées dans ce même fichier en tant que records.

2.4.4 Operator et ses implémentations

`Operator` est l'interface qui se charge de décrire les différents opérateurs du jeu. Chaque opérateur possède ainsi pour responsabilité de connaitre la méthode à effectuer lors de leur mise en effet parmi `onMove`, `onRuleCreation`, `onSuperpositon` ou `onDeath`. Un opérateur doit aussi savoir si ses opérandes forment une règle interdite ou encore une interdiction (ROCK IS ROCK, par exemple, qui indique que ROCK ne peut jamais changer de type). HasOperator et IsOperator implémentent Operator

2.4.5 Record Type

`Type` est un record qui représente un type d'élément, aussi bien en tant qu'entité (pour ROCK, l'objet ROCK) que nom (pour ROCK, le mot ROCK). Chaque type est unique et est ainsi simplement déterminé par son instance (il ne possède que deux images, une pour le texte et une pour l'entité). Pouvant être à la fois une opérande gauche ou une opérande droite, un type est capable de se donner à un `BoardElement` pour modifier son type (BABA IS ROCK, par exemple).

2.4.6 Package manager

Le package `manager` contient les classes se répartissant la gestion du jeu au cours d'une partie, ainsi que le `Model` classe servant à accèder et modifier les éléments de la partie.

Le `Model` comme expliqué précédemment est la classe qui contient la liste des `BoardElement` de la partie ainsi que les méthodes servant à accéder et modifier cette liste.

La classe `DisplayManager` est le manager en charge de l'affichage. A chaque frame il récupère la liste des éléments affichables fournie par le `Model` et les affiche sur le plateau

Le `RuleManager` a pour responsabilité d'analyser le plateau, trouver les règles formées par les éléments dans la configuration actuelle de celui-ci et de fournir cette liste de règles au `LevelManager`.

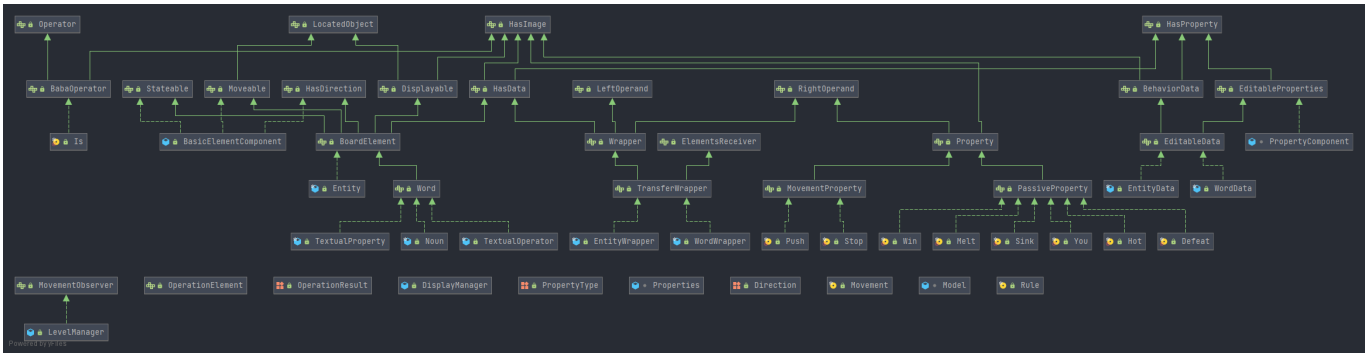
Enfin le fichier principal, le `LevelManager` est le manager coordonnant les informations du `Model` et celles du `RuleManager` afin d'appliquer les règles actuelles de la partie sur les éléments du plateau et déplacer les éléments ayant la propriété YOU selon les directives du joueur.

Le package contient également des interfaces servant à respecter le principe d'encapsulation afin de ne pas fournir en paramètre des Manager à des classes dont la responsabilité n'est

3 Améliorations et Difficultés

3.1 Améliorations post-bêta

Les principales remarques sur le projet lors de la bêta concernaient sa taille et sa complexité, jugées toutes deux trop importantes.

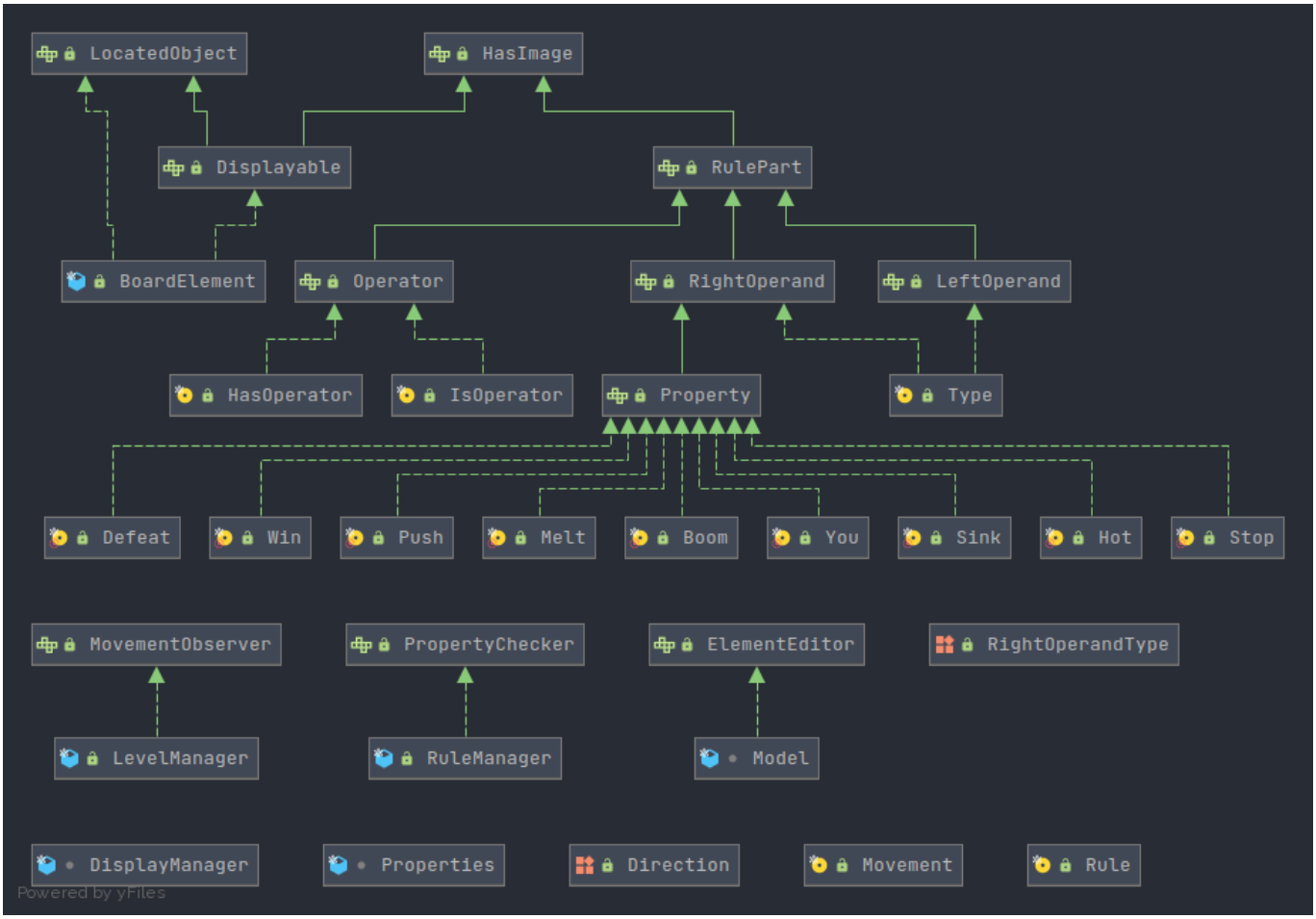


En effet, comme on peut le voir sur l'image, nous avions créé des interfaces inutiles et très nombreuses, et essayé de représenter différemment des éléments qui étaient sémantiquement identiques (séparations des `BoardElements` en plusieurs classes, séparation de `Property` en `MovementProperty` et `PassiveProperty` ...). De même un autre problème était la manière de stocker les `BoardElement`, ils étaient auparavant stocké dans des listes séparées, une liste par `Wrapper`, et ces listes étaient rassemblées en une unique liste pour l'affichage, ce qui nous a été décrit comme trop coûteux et illogique. Enfin, le dernier point abordé était notre algorithme de gestion des règles, que nous stockions entre chaque update et faisions une différence entre la liste des anciennes règles et les nouvelles, ce qui était plus compliqué que de simplement recréer les règles à chaque update.

Suites à la bêta, nous avons pris ces remarques en considération et accord avec ces dernières, apporté les changements suivants au projet:

- Rassembler les différents types de propriétés en une seule interface `Property`
- Rassembler les 4 sous-types de `BoardElement` en une seule classe, à savoir `BoardElement`
- Rassembler tous les `BoardElements` dans une unique liste afin de ne pas avoir à les rassembler pour l'affichage à chaque frame
- Modifier l'algorithme de gestion des règles afin de ne soient plus stockées mais simplement recréés

Ainsi, on obtient le diagramme suivant:



3.2 Difficultés rencontrées

La principale difficulté rencontrée a été l'obligation d'effectuer de nombreux refactor afin de pouvoir gérer et ajouter les nouvelles features du jeu, tout en respectant les principes de programmation orientée objet et cela sans tomber dans la redondance ou la surcharge inutile de classes dans le but de vouloir définir le maximum de concepts en vain .

4 Conclusion

Ce projet a pour nous été une excellente opportunité pour développer et appliquer des Design Patterns que l'on ne connaissait pas auparavant, et ainsi d'enrichir notre expérience en programmation orientée objet. En effet, en plus d'être un jeu intéressant, Baba Is You est très propice à la conception d'une architecture de code qui doit être rigoureuse et au polymorphisme. Nous avons donc favorisé en priorité la modularité tout en gardant une encapsulation et une sécurité maximale. En effet, il est maintenant assez facile d'ajouter de nouveaux opérateurs ou propriétés au jeu avec très peu de modification du projet.