Groupe 5
Bertrand, Degeldt, Peeters, Raxhon
Octobre 2018

Pirrralchemy

Dans le cadre du laboratoire de Test driven development, il nous est demandé de réaliser un projet par groupe. L'objectif premier n'est pas la finalisation de ce dernier mais la réalisation planifiée et structurée conforme au développement en entreprise. Dans cette première étape, nous commençons par expliquer le concept et fonctionnement de notre idée. Ensuite, nous parlons des diagrammes de classes et de cas d'utilisation, des conventions de codage que nous avons choisi ainsi que les critères de qualité et la manière dont nous les mesureront au cours de réalisation du projet. Le contenu de ce document reprend nos premières idées de notre application mais celles-ci pourraient éventuellement évoluer lors de la réalisation. L'application sera codée en Java.

1. Notre projet Pirrralchemy

1.1 Idée

Notre application est inspirée du jeu existant *Alchemy*. Le p
rincipe d'Alchemy est le suivant : l'utilisateur a une 70^{aine} d'éléments à trouver (ex : bombe atomique, île, boue, etc.. Au début du jeu, il a à sa disposition quatre éléments : eau, terre, feu, air. Il doit combiner deux des éléments pour créer un nouvel élément et ainsi de suite pour arriver à créer les 70 éléments à trouver. Un exemple pourrait être de mélanger les éléments *eau* et *terre* pour trouver l'élément *boue.* Nous suivons le même concept de base.

1.2 Paliers

Nous pensons diviser le jeu en paliers afin d'avoir des 'étapes' à franchir et éviter que le jeu ne devienne trop redondant. A chaque palier, le jeu changera légèrement.

Premier palier : palier de base ; il faut combiner deux éléments pour débloquer un nouvel élément.

Deuxième palier : une troisième composante s'ajoute à la combinaison. Il est maintenant possible de débloquer des outils et il faut combiner deux éléments et un outil pour débloquer un nouvel élément.

Troisième palier : il faut combiner deux éléments, un outil et un personnage pour débloquer un nouvel élément.

Il est possible d'ajouter des paliers par la suite lors du développement mais ceux-ci sont nos premières idées.

1.3 Niveaux

Dans chaque palier, il y a des niveaux. Ceux-ci correspondent aux éléments débloqués. Pour chaque élément débloqué, l'utilisateur passe à un niveau supérieur. Nous avons certains éléments clés qui, lorsqu'ils sont débloqués, font passer

l'utilisateur au palier suivant. Nous choisissons ces éléments stratégiquement : le fait que ces deux éléments soient débloqués signifie que l'entièreté des éléments plus simples ont été débloqués. L'utilisateur dispose donc des éléments de base qui lui permettront de jouer dans le deuxième palier.
Exemple : Pour passer au deuxième palier, il faut débloquer les éléments *vie* et *énergie*.

1.4 Extensions

Une piste d'amélioration de notre application serait d'ajouter des extensions. Ces extensions (ex : Harry Potter, The incredibles) viennent avec de nouveaux éléments, outils et personnages. Lorsque l'utilisateur a fait le tour de l'application de base, il a la possibilité de télécharger une extension pour que le jeu continue. Il faut donc que les éléments soient facilement ajoutables au code du jeu.

1.5 Social

Pour jouer au jeu, l'utilisateur peut se faire un compte et peut créer un réseau d'amis sur l'application. Chacun a un profil public qui montre le pseudo, le niveau actuel et le nombre d'amis de l'utilisateur. Il est donc possible de jeter un oeil à l'avancement de ses amis dans le jeu.
Une idée serait de permettre aux joueurs de lancer des défis à des amis. Un défis pourrait être : trouver le plus d'éléments en un temps déterminé.

1.6 Interface

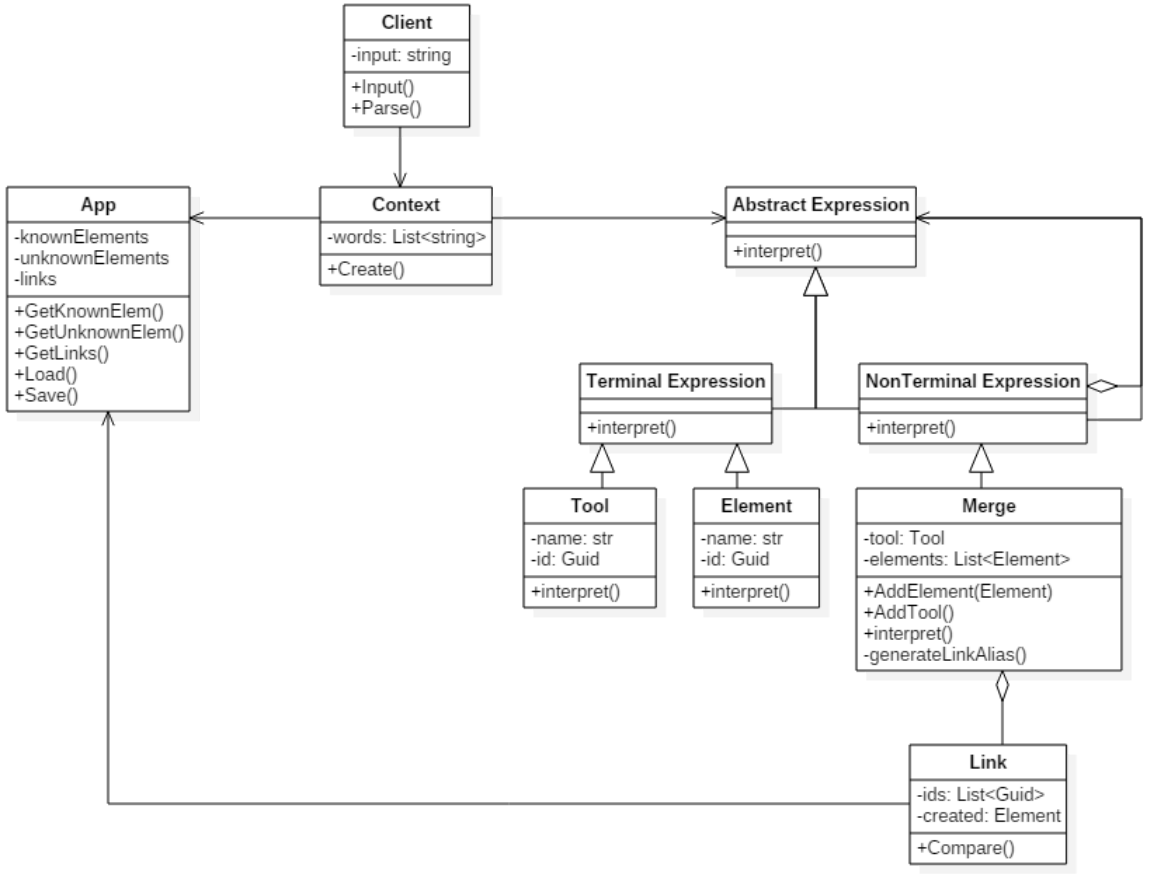
Nous n'avons pas encore défini d'interface mais elle sera simple et facile d'implémentation. Nous comptons faire une interface en ligne de commande. Celle-ci pourra bien-entendu être améliorée par la suite si nécessaire.

2. Analyse

2.1 Règles de combinaisons - choix du Design Pattern

La combinaison de base telle que nous l'avons introduite est encore simple. L'architecture de l'application doit permettre l'implémentation facile de nouvelles règles sur la combinaison d'éléments. Le design pattern interpreter correspond à ce critère. Des exemples de règles additionnelles sont décrites ci-dessus. Par exemple: ajout d'une classe personnage qui requiert une autre méthode de "merge".

2.2 Diagramme de classe



- App: Charge les données relatives au jeu: les éléments/outils connus et inconnus ainsi que la liste des liens entre tous les éléments.
- Client: Récupère les inputs entrés par l'utilisateur et les parse en expressions qu'il transmet au contexte.
- Context: Stock les expressions dans une liste et les interprète. Vérifie si les expressions appartiennent aux collections définies par l'application.
- AbstractExpression (abstract class): Conformément au design pattern interpreter, cette classe abstraite possède une méthode "interpret". Les classes qui hériteront de "AbstractExpression" devront implémenter "interpret".
- Terminal Expression (abstract class): En pratique, il s'agit des acteurs en bout de chaîne dans une expression. En quelque sorte, ce sont des éléments à manipuler, ils ont un sens à part entière, contrairement aux NonTerminalExpression.
- NonTerminalExpression(abstract class): Ce sont des expressions qui ont besoin d'expressions terminales pour pouvoir être interprétées.
- Element: Expression permettant de représenter les objets que l'on va combiner dans le jeu.

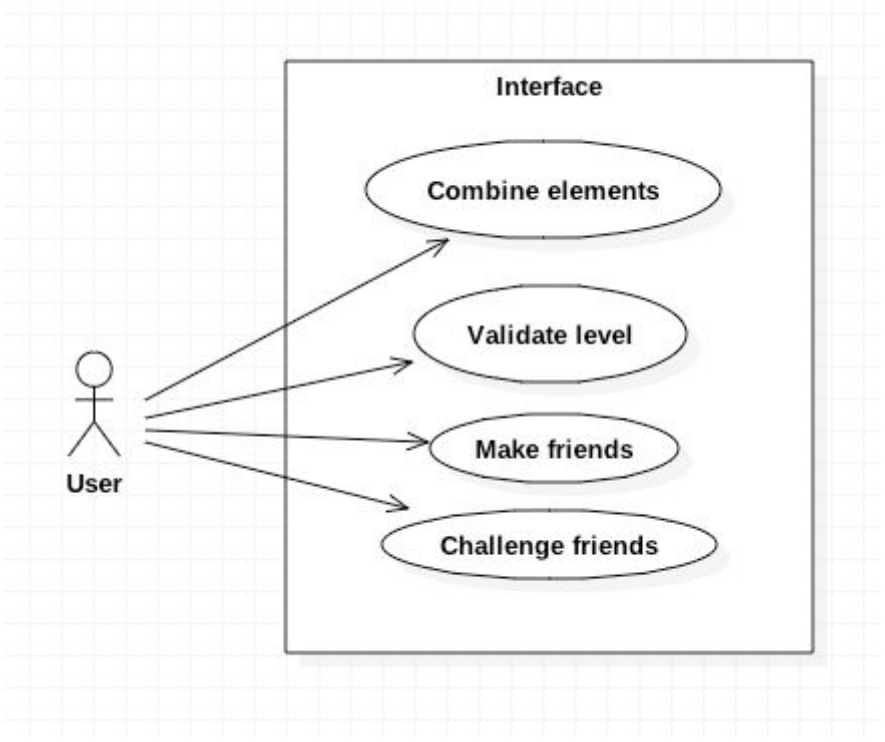
- Tool: Semblable aux éléments. Certains éléments ne peuvent être créés qu'avec des outils. Les outils sont débloqués via un système "d'objectifs".

NB: Dans un premier temps, nous ne nous concentrons pas sur l'implémentation de personnages.

- Merge: Opération permettant de combiner deux éléments, avec ou sans outils. Quand on souhaite créer deux éléments, la classe merge créer un "link" dont l'ID est créé en fonction de ses attributs.
- Link: Implémente les liens entre les éléments et les outils. Chaque link a un ID créé en fonction des éléments et des outils qui le composent. Cet ID est ensuite comparé avec les "links" dans l'instance de la classe "App". Si ce lien existe dans l'attribut de la classe "App", un nouvel élément est créé. Il suffit alors de vérifier s'il a déjà été découvert ou non.

Il est à noter que l'application correspondant au diagramme actuel ne représente que les aspects "primordiaux" du projet. Les différentes classes présentée ici devraient néanmoins permettre l'implémentation de toutes les fonction décrites plus haut dans de futures mises à jour.

2.3 Diagramme de cas d'utilisation



2.4 Conventions de codage

Les conventions de codage restent simples mais efficaces:

- Noms de Variables et noms de fonctions en écriture camelCase ; fonction minMaj() et variables minMaj.
- Pas de chiffre dans les noms, pas de “_”, pas plus de 3 mots dans les noms
- Noms de Classes en MajMaj.
- Les fonctions redéfinies (héritage) doivent être précédées par la balise “@Override”
- Accolades : sur la même ligne que la définition de la fonction.
- Laisser une ligne d’espace entre les fonctions.
- Mettre un espace après chaque virgule.
- Séparer les blocs d’instructions (if, else, for) par une ligne d’espace.
- Fonctions commentées en anglais de leur action par une phrase au dessus de la fonction. Dans la phrase : input, ce que fait la fonction et output.
- Classes commentées de leur raison d'être par une phrase simple au dessus du nom de la classe.
- Le respect global de la programmation orientée objet est aussi une convention implicite.

Toutes les conventions de codage sauf les trois dernières sont vérifiées par le checkstyle dans Jenkins. Ces dernières devront être vérifiées à la main.

2.5 Critères de qualité

Le but de notre jeu est d'être fluide à l'utilisation, clair, solide et simple à mettre à jour. C'est pour ces quatre raison que nous avons décidé des critères de qualités suivants:

- Test unitaire de chaque fonction sauf input() dans client.
- Exécution rapide (0.5s) des opérations de l’utilisateur. Càd après l’input une opération retourne une réponse à l’utilisateur.
- Tout ceci sans dépasser 30% de commentaires.
- Une extension du jeux doit se faire sans ajout de code. Deux de classes doivent être ajoutées ou changées au maximum pour changer ou ajouter une fonctionnalité au jeu.

3. Documentation

- Nous utilisons Trello pour fixer nos objectifs à court terme (https://trello.com/b/De0DTYL4/pirrralchemy).
- Github pour partager le code (https://github.com/Beheroth/Pirrralchemy.git). Chacun a sa branche que Sam mergera de temps à autre toutes les branches sur Master.
- Json parsing(https://www.tutorialspoint.com/json/json_java_example.htm).
- Nous utilisons Jenkins. Le nom de notre projet est Pirrralchemy.
- Environnement de développement : IntelliJ Community Edition 2018.2.5

- Version de SDK: 11.0.1

3.1 Manuel

Pour lancer l'application, suivez les étapes suivantes :

1. Dans IntelliJ, importer le projet depuis version control (Git).
2. Run Client.java (se trouve dans \Pirrralchemy\src\main\java)
3. Éventuellement, s'il s'agit de la première fois que le logiciel est lancé, il faudra configurer le classpath des modules. Dans ce cas, il suffit de spécifier d'utiliser ceux de Pirrralchemy.
4. Les test Junit sont créés.

A chaque build, un rapport est créé avec certaines informations : nom de Github de la personne qui a réalisé le commit, etc.

3.2 Etat du projet

Actuellement, le projet comprend les fonctionnalités suivantes :

- Palier 1 : mélange de deux éléments pour en créer un troisième
- Interface sur le terminal
- Mise à jour du Json servant à sauver la progression du joueur.

Les éléments qu'il reste à réaliser sont les suivants :

- Ajout des extensions (cf. 1.4)
- Implémenter les paliers 2 et 3
- Notion de niveau (cf. 1.3)
- Aspect social (cd. 1.5)
- Réaliser plus de test

4. Conclusion

En conclusion, nous avons créé un jeu basé sur le jeu existant Alchemy en le transformant avec des paliers et des niveaux. Pour l'implémentation, nous avons choisi le design pattern interpreter. Les critères de codage sont simple et stricts. les critères de qualité sont quant à eux concentrés sur la fluidité, la clarté, la solidité et l'extension.

---------- TO DO POUR 07/11 ----------

Travailler avec git obligé : tjrs message court
Faire le plus de commit possible : si un commit peut être divisé en deux c'est bon
Implémenter avec Java
Jenkins à développer par les informaticiens pour implémenter le test de code etc.

Pour la prochaine séance (07/11) : avancer un max sur le code : pas toutes les fonctionnalités mais toutes les fonctionnalités doivent être testées avec tous les test. Pouvoir montrer ça et montrer le processus qui nous permet de vérifier tout ce qu'on s'est fixé comme règles.

A la séance prochaine on check avec lui et il nous dit ce qu'on doit encore ajouter/modifier.

TO DO

Terminer checkstyle
tout le monde relit le rapport
Tests unitaires terminer (Arnaud)
Trouver comment (pom : balise test)