Manuel développeur

Table des Matières

Introduction

Le projet The Big Adventure est un jeu de type aventure en 2D développé en Java. Le joueur incarne un personnage qui doit explorer un monde ouvert (ou presque) et ramasser des objets pour progresser dans le jeu.

Le but de ce projet était de nous faire découvrir le développement d'un gros projet en Java (ici un jeu vidéo). Cela nous a forcé à apprendre beaucoup plus que ce qui était enseigné en cours, et nous a permis de réexploiter les connaisances acquises. Le but était aussi d'apprendre a structurer un projet de grande envergure avec un `Design Pattern` ici le `MVC`.

Dans ce manuel, nous allons détailler les différentes étapes de développement du projet, ainsi que les outils utilisés et les problèmes rencontrés. Vous découvrirez ainsi quelle a été notre approche pour mener à bien ce projet.

Contexte du Projet

Tout d'abord, ce projet nous a été proposé par notre enseignant de Programmation Orientée Objet, M. Forax. Vous pourrez trouver le sujet du projet ici.
Dans ce sujet, vous trouverez ainsi le cahier des charges du projet, ainsi que les différentes étapes de développement à suivre.
Le cahier des charges étant plutôt vague, nous avons dû nous-même définir les contraintes techniques liées aux fonctionnalités demandées.

Architecture du Projet

Organisation des répertoires

Il nous a été recommandé de suivre une architecture de projet de type `MVC` (Modèle-Vue-Contrôleur), c'est-à-dire de séparer les différentes parties du code en trois catégories distinctes.
Vous retrouverez donc dans le répertoire `src` les trois packages `model`, `view` et `controller`, qui contiennent respectivement les classes du modèle, de la vue et du contrôleur. Ces packages sont précédés par `fr.uge.thebigadventure` qui est le package racine du projet.

L'utilisation d'un modèle `MVC` était tout a fait adapté à notre projet comme a beaucoup de projets JAVA. En effet, cela permet de séparer les différentes parties du code et de les rendres indépendantes.

Architecture du code

Nous avons donc séparé notre code en trois parties distinctes, le modèle, la vue et le contrôleur.
Le modèle contient les classes représentant les différents éléments du jeu, ainsi que les classes permettant de les construire.
Vous trouverez donc dans le package `model` des classes tel que `InventoryItem` qui représente un objet de l'inventaire du joueur, ou encore `Entity` qui représente un type d'entité du jeu.

Il a aussi des classes tel que `ElementBuilder` qui permet de construire un élément du jeu à partir d'un `ElementBuilder` et `MapBuilder` qui permet de construire une map à partir d'un `MapBuilder`.

Ces classes sont utilisées par le `MapParser` pour construire une map à partir d'un fichier `.map`.

Ensuite, nous avons la vue (package `view`) qui contient les classes permettant d'afficher le jeu.
Vous trouverez ainsi dans celui-ci des classes tel que `PlayerView` qui affiche le joueur, ou encore `EntityView` qui affiche une entité du jeu.
Il y a aussi des classes tel que `InventoryView` qui affiche l'inventaire du joueur ou encore `NPCView` qui affiche un personnage non joueur.
Ces classes sont utilisées par le `GameInitializer` pour afficher le jeu.

Enfin, nous avons les contrôleurs (package `controller`) qui contient les classes permettant de contrôler le jeu.
Vous avez ainsi tous ce qui peut être controller par le joueur. Vous trouverez donc le `KeyboardController` qui permet la gestion des touches du clavier.
Vous trouverez aussi les `CommandController` qui permet de gérer les commandes du jeu ou encore `PlayerController` qui permet de gérer le joueur et de le faire interagir avec les éléments du jeu.

Difficultés liées à l'architecture

N'ayant jamais utilisé de tels `Design Pattern` auparavant, nous avons eu quelques difficultés à comprendre comment les utiliser.
Il est donc possible que certaines parties du code ne respectent pas complètement l'architecture `MVC`. Cependant, nous avons essayé de nous en rapprocher le plus possible.
Nous avons surtout eu du mal a bien définir nos contrôleurs et à les utiliser correctement.

Environnement de Développement et Collaboration

Outils de Développement

Nous avons eu certains soucis lié à l'utilisation de `Eclipse` pour le développement du projet. En effet, nous avons eu des problèmes avec les nouveaux switchs de `Java 21` à cause du compilateur `Eclipse` qui n'était pas à jour.
Cela n'a pas été un problème majeur, car nous avons pu utiliser `javac` pour compiler le projet.
Nous avons aussi utilisé `Ant` pour la compilation et la construction du projet comme demandé dans le sujet.

Collaboration

Nous avons utilisé `Discord` pour communiquer entre nous de manière fluide et rapide.
Nous faisions aussi des petits points régulier sur l'avancement du projet afin de se donner des objectifs à atteindre pour la prochaine fois.
Afin de garder une trace des modifications apportées au code et avoir un historique des versions, nous avons utilisé `Git`.
Nous avons donc créé un dépôt `Git` sur `GitHub` pour le projet. Vous pourrez d'ailleurs le retrouver ici si vous souhaitez voir l'historique des modifications apportées au code.

Développement du Projet

Organisation du Travail

Nous avons décidé de nous répartir le travail dès le début de la manière suivante :

- Florian : Développement du `Map Parser` et des builders associés
- Nathan : Développement du moteur de jeu et des contrôleurs

Une fois que nous avons estimé avoir fini nos parties respectives, nous avons commencé à travailler ensemble sur des implémentations plus complexes sur le jeu lui même.
Nous avons ainsi pu travailler chacun de notre côté sur des parties différentes du projet, tout en travaillant ensemble sur des parties plus complexes.

Difficultés liées au Développement

À de nombreuses reprises, nous avons eu des problèmes d'interprétation du sujet. En effet, certaines parties du sujet étaient assez vagues et nous avons dû faire des choix.
Par exemple, pour un élément `Food`, il ne nous est pas dit comment il devait être représenté. Nous avons donc décidé de lui ajouter un attribut `health` qui représente le nombre de points de vie que le joueur récupère en le mangeant.

Nous avons aussi préféré utiliser notre propre "Lexer" pour le `Map Parser` plutôt que d'utiliser celui fourni sur le Discord universitaire. Étant donné que nous avions déjà commencé à développer notre propre `Map Parser`, nous avons préféré le terminer plutôt que de l'abandonner pour utiliser le Lexer fourni.

Il nous a aussi été difficile de comprendre ce qui était attendu avec la commande `--add-elements` et nous avons donc décidé de ne pas l'implémenter.

Phases implémentées

Nous avons implémenté entièrement les phases 0 et 1.
Nous avons aussi commencer à implémenter la phase 2, c'est pourquoi nos `model` ont des méthodes qui ne sont pas toujours utilisées dans le jeu.
Nous avons aussi implémenté les différentes commandes demandées dans le sujet. Pour plus d'information sur le fonctionnement du jeu, vous pouvez consulter le manuel utilisateur ici.

Changements depuis la Soutenance Bêta (Décembre)

Nous avons apporté de nombreuses modifications au projet depuis la soutenance bêta.
Mais ce qui nous intéresse le plus ici, c'est les recommandations que nous avons reçu lors de la soutenance bêta.
Nous avons avec les conseils de Mme Béal décide de restructurer notre projet pour qu'il soit plus clair et plus facile à comprendre. Nous avons ainsi réduit notre fonction main qui était alors beaucoup trop longue et nous avons déplacé certaines méthodes dans d'autres classes.
Nous avons aussi décidé de renommer certaines classes et méthodes pour qu'elles soient plus claires et plus compréhensibles ainsi que des packages pour qu'ils soient plus cohérents.

Nous avons aussi documenté notre code pour le rendre plus compréhensible et pouvoir le maintenir plus facilement.

Map Parser

Notre `Map Parser` n'est pas réalisé par une analyse syntaxique linéaire. Celui-ci est composé de plusieurs expressions régulières successives appliqué sur tout le texte. On découpe d'abord tout notre texte en différentes `sections`, celle-ci sont ensuite divisées en plusieurs `attributs` et enfin le contenu des `attributs` sera analysé. Lors de l'analyse, on construit une map avec une classe `MapBuilder`. Lorsque l'on rencontre un élément, on le construit avec un `ElementBuilder` inclut dans le `MapBuilder`. Toute erreur de construction d'un nouvel élément renvoyé par l'`ElementBuilder` est récupéré par le MapParser afin d'afficher l'erreur en l'associant à la ligne la plus proche possible de l'erreur. Les erreurs d'analyse sont également affichés avec la ligne la plus proche de l'erreur.

Compilation et Construction avec Ant

Vous trouverez un fichier `build.xml` à la racine du projet.
Ce fichier contient les différentes cibles `Ant` utilisées pour la compilation et la construction du projet.
`Ant` est un outil de compilation et de construction de projets Java développé par la fondation Apache.
Il permet de compiler et de construire un projet Java de manière automatique.

Comme demandé dans le sujet, nous avons écrit à la main notre fichier `build.xml`.
Dans celui-ci vous trouverez les cibles suivantes :

- `compile` : Compile les fichiers `.java` du projet et génère les fichiers `.class` dans le répertoire `classes`
- `jar` : Génère un fichier `.jar` exécutable du projet à partir des fichiers `.class` générés par la cible `compile` ainsi que du dossier `resources`, `lib` et `resources/META-INF/MANIFEST.MF`
- `javadoc` : Génère la documentation du code dans le répertoire `docs/api`. Vous pourrez ensuite consulter la documentation du code en ouvrant le fichier `docs/api/index.html` dans votre navigateur
- `clean` : Supprime les fichiers `.class` générés par la cible `compile` ainsi que le repertoire `docs/api` généré par la cible `javadoc`.

Pour lancer ses cibles, il vous faut bien sûr avoir `Ant` d'installé sur votre machine. Vous pourrez l'installer ici.
Une fois `Ant` installé, vous pourrez lancer les cibles avec la commande `ant <cible>` dans le répertoire du projet.
Par exemple, pour compiler le projet, il vous suffit de lancer la commande `ant compile` dans le répertoire du projet.

Problèmes Connus

Seul la transformation d'un `TREE` en `BOX` avec une `SWORD` a été implémenté. Attention, ce `TREE` doit être un `element` et non simplement positionné dans la toile de fond `grid`.

L'attribut `phantomized` n'a pas encore été implémenté.

L'attribut `teleport`, permettant de charger d'autres map à partir d'une map, n'a pas été implémenté non plus.

Liste des problèmes non résolus

On ne peut pas mettre de `:` dans un attribut `text`, sinon l'analyseur le lira comme un attribut.
Cela est dû au fait que l'analyseur utilise `:` comme séparateur entre les attributs et leur valeur.
Nous nous somme dit que cela n'était pas très grave, car il est peu probable que l'on veuille mettre un `:` dans un attribut `text`.
Pour contourner le problème des `:`, il suffit de ne pas en mettre dans les attributs `text`.
Cela peut être contraignant et nous nous en excusons.

Nous avons aussi trouver des problèmes avec les dialogues. Malgré nos efforts, nous avons des soucis d'affichage avec certains dialogues mais nous n'avons pas réussi à trouver d'où cela venait.
Les dialogues étant instanciés à chaque fois que l'on parle à un personnage et remis à zéro, cela est très étrange.

Conclusion

Vous l'aurez compris, ce projet nous a donné du fil à retordre. Nous avons dû faire face à de nombreux problèmes et nous avons dû faire des choix d'imlémentations.
Nous avons aussi dû apprendre à utiliser de nouveaux outils et à travailler ensemble sur un projet d'envergure.
Le fait d'avoir un aussi gros projet à réaliser nous a permis de nous rendre compte de l'importance de la documentation et de la structuration du code.
Il est tellement plus facile de s'y retrouver lorsque le code utilise un `Design Pattern` et que celui-ci est bien documenté.
Nous n'avons pas pu implémenter toutes les fonctionnalités demandées dans le sujet, mais nous avons tout de même réussi à implémenter la plupart des fonctionnalités que nous voulions.
Ceci étant dit, nous sommes tout de même assez fiers du résultat obtenu et nous espérons que vous apprécierez notre jeu.

Nous avons encore beaucoup de choses à améliorer dans notre jeu.
Nous aimerions par exemple implémenter les phases 2 et 3 du projet, ainsi que les fonctionnalités manquantes.
Nous aimerions aussi améliorer l'interface graphique du jeu et ajouter des animations.
Évidemment nous aimerions aussi corriger les bugs que nous avons rencontré et améliorer la stabilité du jeu.
Mais cela demande beaucoup de temps et d'énergie. Suivre un projet de cette envergure est très prenant et nous avons d'autres projets à réaliser et d'autres cours à suivre.
Nous regrêtons que ce tout premier projet que nous réalisons en Java soit aussi complexe et imprécis et que nous n'ayons pas eu le temps de créer un programme complet et parachevé.

Vous pouvez retrouver le manuel utilisateur ici.