Rapport du jeu

‘Tu es le héros’

- Noms & Prénoms :
  - BOUDINAR LOUNES
  - MAOUCHI MELINA

- Formation :
  Licence 3 MIAGE Apprentissage

Sommaire

1-Introduction :

Dans le cadre de notre projet de développement logiciel, nous avons conçu un jeu d'aventure interactif inspiré par l'univers captivant de One Piece, un manga célèbre créé par Eiichiro Oda. Notre jeu, intitulé "Le monde des pirates", plonge les joueurs dans un voyage épique à travers des îles mystérieuses et des rencontres avec des personnages emblématiques de la série.

Ce rapport présente une analyse détaillée de la conception et de l'implémentation de notre jeu, mettant en lumière les choix technologiques, les structures de données, les algorithmes et les défis rencontrés tout au long du processus de développement. Nous décrivons également les fonctionnalités principales du jeu, telles que la navigation basée sur des nœuds, les scénarios interactifs, les énigmes à résoudre et l'intégration de médias tels que des images et des sons pour une expérience immersive.

2-Identifications des packages :

1. ***Representation*** : Ce package encapsule des classes essentielles pour la gestion et la représentation structurée des composants visuels, sonores et conceptuels du jeu "Le monde des pirates" telles que :

- **Sound :**

Cette classe représente un objet ou un utilitaire permettant la gestion des fichiers audio dans le jeu. Elle peut inclure des méthodes pour charger, jouer et arrêter des pistes audio.

- **ImageDisplay :**

Cette classe est responsable de l'affichage des images graphiques dans le jeu. Elle permet de charger des fichiers image depuis le système de fichiers local et de les afficher à l'écran. ImageDisplay peut intégrer des fonctionnalités comme le redimensionnement, la rotation ou d'autres transformations visuelles pour s'adapter aux besoins spécifiques des scènes du jeu.

- **NodeCombat :**

Ce type de nœud représente un point dans la structure du jeu où un combat ou une confrontation spécifique est géré. Il peut contenir des informations sur les ennemis, les personnages impliqués, les règles de combat, ainsi que les résultats et conséquences possibles de cette interaction.

- **Node, NodeN, NodeU, NodeZ :**

Ces classes représentent différents types de nœuds dans la structure de données du jeu. Chacune peut avoir un rôle spécifique dans la navigation, la prise de décision ou la gestion des événements narratifs. Par exemple :

1. Node peut être une classe abstraite de base pour d'autres types de nœuds.
2. NodeN, NodeU, NodeZ représentent des variantes ou des sous-types de nœuds avec des caractéristiques ou des comportements spécifiques dédiés à un scénario spécifique.

- **InnerNode, ChanceNode, DecisionNode, NodeTerminal :**

1. InnerNode : Un nœud interne qui peut contenir des sous-nœuds ou des informations supplémentaires pour la gestion de la structure de l'arborescence narrative ou de jeu.
2. ChanceNode : Un nœud où une décision est prise en fonction de conditions aléatoires.
3. DecisionNode : Un nœud où une décision doit être prise par le joueur, influençant ainsi le cours du jeu.
4. NodeTerminal : Un nœud qui marque la fin d'une partie où aucune autre action n'est possible, il contrôle également les résultats pour savoir si le joueur a gagné ou pas.

2. Univers :

Le package `Univers` regroupe plusieurs classes fondamentales pour la modélisation et la gestion d'un univers fictif dans un jeu ou une simulation. Voici une définition et une explication de chaque classe présente dans ce package :

1. Classe `Cle` :

- Définition : La classe `Cle` représente une clé avec un indicateur qui peut être activé ou désactivé.

- Explication : Cette classe est utilisée pour encapsuler une simple donnée booléenne (`flag`). Les méthodes `getFlag` et `setFlag` permettent respectivement de récupérer et de modifier l'état de cette clé. Elle est souvent utilisée pour activer ou désactiver des fonctionnalités ou des comportements dans le jeu en fonction de conditions spécifiques.

2. Classe `Monde` :

-Définition : La classe `Monde` représente un environnement dans lequel un personnage évolue.

- Explication : Cette classe maintient une référence à un objet `Personnage` et conserve également l'information sur le lieu actuel où se trouve ce personnage (`lieuActuel`). Les méthodes `getPersonnage` et `setPersonnage` permettent de manipuler l'objet `Personnage`, tandis que `getLieuActuel` et `setLieuActuel` sont utilisées pour obtenir et modifier la localisation courante dans le monde.

3. Classe `Personnage` :

-Définition : La classe `Personnage` représente un personnage avec un nom, un type et un nombre de points.

- Explication : Cette classe encapsule les caractéristiques d'un personnage fictif dans le jeu. Elle comprend des attributs tels que `nom` (le nom du personnage), `points` (les points de vie ou une autre forme de ressource), et `type` (le type de personnage, comme 'Luffy' ou 'Sakura'). Les méthodes comme `ajouterPoints` et `retirerPoints` permettent de modifier les points du personnage, tandis que les accesseurs (`getNom`, `getPoints`, `getType`) permettent d'obtenir ces informations.

Ensemble, ces classes du package `Univers` fournissent une base solide pour modéliser les entités principales d'un univers de jeu, incluant des fonctionnalités pour gérer des données cruciales telles que les personnages, leurs propriétés et les paramètres environnementaux comme la localisation actuelle.

3. ***Tests :*** C'est un package qui regroupe tout les tests unitaires des méthodes qui composent les différentes classes qui se trouvent dans les 2 packages précédemment cités :

***Tests Unitaires pour le Package representation :**

- Chaque classe du package representation possède des méthodes qui effectuent des opérations spécifiques, comme la manipulation de sons, d'images, ou la gestion des nœuds dans une structure de données mais également le passage entre node en utilisant choosenext().

- Les tests unitaires dans ce contexte s'assurent que chaque méthode de ces classes fonctionne comme prévu. Par exemple, pour la classe Sound, il pourrait y avoir des tests vérifiant que la lecture, l'enregistrement ou d'autres opérations audio se déroulent correctement et renvoient les résultats attendus.

***Tests Unitaires pour le Package univers :**

- Les classes comme Cle, Monde et Personnage du package univers représentent des entités et des fonctionnalités centrales du système.
- Les tests unitaires pour ces classes vérifient que les méthodes telles que getFlag et setFlag de la classe Cle, les méthodes de manipulation du personnage et de son environnement (Monde), ainsi que les opérations associées aux personnages (Personnage) sont correctement implémentées et renvoient les résultats attendus.

3-Déroulement du jeu :

Dans la classe `Jeu` du package `Representation`, le déroulement du jeu et le passage d'un nœud à un autre sont gérés de manière à créer un scénario interactif. Voici une explication détaillée de ce processus

1. Initialisation du Personnage et du Monde :

```
Personnage personnage = new Personnage("Monkey D. Luffy", "Luffy");
Monde monde = new Monde(personnage);
```

- Un personnage est créé avec le nom "Monkey D. Luffy" et le type "Luffy".

- Un monde est instancié avec ce personnage.

2. Création et Affichage de l'Introduction :

```
InnerNode intro = new InnerNode(1, "Introduction", monde);
intro.display(1);
```

- Un nœud interne (`InnerNode`) pour l'introduction est créé avec l'id 1 et le titre "Introduction".

- La méthode `display(1)` est appelée pour afficher le contenu de ce nœud introductif.

3. Navigation à travers les Nodes :

Chaque nœud (`Node`) contient des méthodes pour afficher son contenu (`display()`) et pour choisir le nœud suivant (`chooseNext()`). Voici comment cela se déroule :

- Premier Choix (`choix1`) :

```
Node choix1 = intro.chooseNext(1);

choix1.display(1);
```

- À partir de l'introduction (`intro`), le joueur fait un premier choix en appelant `chooseNext(1)`.

- Le contenu du nœud `choix1` est ensuite affiché avec `display(1)`.

- Suivi des Choix et Navigation :

```
Node d1 = choix1.chooseNext(2);

d1.display(2);

Node c1 = d1.chooseNext(50);

c1.display(50);

Node d2 = c1.chooseNext(3);

d2.display(3);

Node choix2 = d2.chooseNext(2);

choix2.display(2);
```

- À chaque étape, le joueur prend une décision représentée par un nœud (`Node`) spécifique.

- Chaque appel à `chooseNext()` permet de déterminer le nœud suivant en fonction du choix du joueur.

- La méthode `display()` est utilisée pour afficher le contenu pertinent à chaque étape du scénario.

4. Utilisation des Nodes Spécifiques :

- Certains types spécifiques de nœuds (`DecisionNode`, `NodeCombat`, `NoeudTerminal`, etc.) sont appelés à des moments clés du jeu.

- Par exemple, les nœuds `nodeN`, `nodeU`, `nodeZ` sont manipulés à partir du `DecisionNode` avec l'id=2, car ils dépendent du scénario choisi.

- Le `NodeCombat` est utilisé lors des scènes de combat, et le `NoeudTerminal` marque la fin d'un chemin narratif.

Alors :

- Chaque fois que le joueur interagit avec un nœud via `chooseNext()`, la logique interne de ce nœud détermine le prochain nœud à afficher en fonction des paramètres passés.

- Les nœuds peuvent être interconnectés de manière complexe, permettant ainsi de créer un arbre de décision narratif où les choix du joueur influencent directement le développement de l'histoire.

- L'appel à `display()` permet de présenter au joueur du texte, des images, des sons, ou toute autre forme de contenu multimédia nécessaire pour enrichir l'expérience du jeu.

4. Bibliothèques utilisées :

Dans le cadre de votre jeu, plusieurs bibliothèques Java sont utilisées pour divers aspects fonctionnels et d'interface utilisateur :

1. Java Standard Library (java.io):

- Utilisée pour la gestion des entrées/sorties, notamment la lecture et l'écriture de fichiers.

2. Java Sound API (javax.sound.sampled) :

- Permet la lecture de fichiers audio et la gestion des périphériques audio.

3. Swing (javax.swing):

- Framework d'interface utilisateur Java pour la création d'applications graphiques.

- Inclut des composants graphiques tels que les boutons, les étiquettes, les champs de texte, etc.

4. JPanel (javax.swing.JPanel) :

- Un composant Swing utilisé comme conteneur pour organiser et agencer d'autres composants graphiques.

- Permet de diviser l'interface utilisateur en sections logiques et de gérer la disposition des éléments à l'écran.

5. ArrayList (java.util.ArrayList) :

- Une implémentation de la structure de données de liste dynamique basée sur un tableau.

- Utilisée pour stocker et manipuler des collections d'objets, offrant des opérations efficaces d'ajout, de suppression et de récupération d'éléments.

6. Scanner (java.util.Scanner) :

- Utilisée pour analyser les entrées de texte utilisateur à partir de diverses sources comme le clavier ou des fichiers.

- Facilite la lecture de données entrées par l'utilisateur ou stockées dans des fichiers.

7. JFrame (javax.swing.JFrame) :

- Une fenêtre de haut niveau Swing utilisée comme conteneur principal pour toute l'interface utilisateur de votre application.

- Gère les opérations de base telles que l'affichage de fenêtres, la gestion des événements utilisateur, etc.

5. Collections & Exceptions :

Collections : Dans cette exemple on montre un bout de code qui utilise la classe `ArrayList` de la bibliothèque Java `java.util` pour gérer une collection dynamique de chaînes de caractères (`String`). Voici une explication détaillée de son fonctionnement en mettant en avant l'utilisation des collections :

1. Déclaration et Initialisation de l'ArrayList :

```
ArrayList<String> listeReponses = new ArrayList<String>();
```

- `ArrayList<String>` définit une liste dynamique contenant des éléments de type `String`. Cela permet d'ajouter, de supprimer et d'accéder aux éléments de la liste de manière flexible.

2. Ajout d'Éléments à l'ArrayList:

```
listeReponses.add("1: Répondre (1)");
listeReponses.add("2: Répondre (2)");
listeReponses.add("3: Répondre (3)");
listeReponses.add("4: Répondre (4)");
```

- Les lignes ci-dessus ajoutent quatre chaînes de caractères à `listeReponses`. Chaque chaîne représente une option de réponse numérotée (de 1 à 4) que l'utilisateur peut choisir.

3. Condition pour Ajouter une Option Supplémentaire :

```
if (!aideUtilisee) {
    listeReponses.add("5: Demander de l'aide à Nami");
}
```

- Cette condition vérifie si une aide n'a pas encore été utilisée (`!aideUtilisee`). Si c'est le cas, une cinquième chaîne est ajoutée à `listeReponses`, indiquant l'option de demander de l'aide à Nami.

4. Utilisation des Collections :

- L'utilisation de `ArrayList` permet de gérer dynamiquement la liste des réponses possibles. Contrairement aux tableaux Java traditionnels, `ArrayList` s'adapte automatiquement à la taille des données qu'il contient. Cela facilite l'ajout, la suppression et la manipulation d'éléments sans se soucier de la taille initiale ou maximale de la liste.

5. Accès aux Éléments de l'ArrayList :

- Une fois que toutes les options de réponse sont ajoutées à `listeReponses`, vous pouvez accéder à chaque élément en utilisant des méthodes telles que `get(index)` pour obtenir un élément spécifique, ou itérer à travers la liste avec une boucle `for` ou `foreach`.

****Exceptions :**

Dans ton code, j’ai mis en place plusieurs types d'exceptions pour gérer différentes situations exceptionnelles qui pourraient survenir. Voici comment chaque exception est utilisée :

1. InputMismatchException :

- Cette exception est généralement utilisée lors de la lecture d'entrées utilisateur à l'aide de `Scanner` ou d'autres méthodes de lecture. Elle se produit lorsque le type de données entrées ne correspond pas au type attendu. Par exemple, si tu attends un entier mais que l'utilisateur entre une chaîne de caractères, une `InputMismatchException` sera lancée. Cela permet de gérer les erreurs de saisie utilisateur de manière appropriée et d'empêcher le programme de planter.

2. IllegalArgumentException :

- Cette exception est lancée lorsque des arguments passés à une méthode sont invalides. Cela peut inclure des valeurs hors des plages autorisées, des formats incorrects, ou d'autres conditions qui rendent les arguments non acceptables pour la méthode en question. Par exemple, si une méthode attend un entier positif et reçoit un entier négatif, une `IllegalArgumentException` peut être lancée pour indiquer que l'argument est invalide.

3. NullPointerException :

- Cette exception est l'une des plus courantes en Java. Elle se produit lorsqu'une tentative est faite pour accéder à un objet dont la référence est `null`. Cela se produit généralement lorsqu'on essaie d'appeler une méthode ou d'accéder à un champ sur un objet qui n'a pas été initialisé correctement ou qui a été explicitement défini comme `null`. Par exemple, si tu essaies d'appeler une méthode sur un objet `null`, une `NullPointerException` sera lancée.

4. Exception dans SOUND :

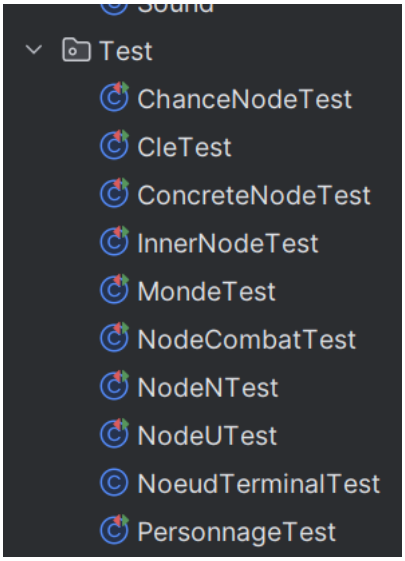
- Dans la gestion des opérations audio, j’ai implémenté la gestion d'exceptions spécifiques pour garantir une robustesse et une fiabilité accrues. Voici un résumé des exceptions utilisées :

1. IOException : Utilisée pour gérer les erreurs génériques liées à l'entrée/sortie, comme la lecture de fichiers audio depuis le disque.

2. UnsupportedAudioFileException: Lancée lorsqu'un type de fichier audio n'est pas pris en charge par Java Sound API.

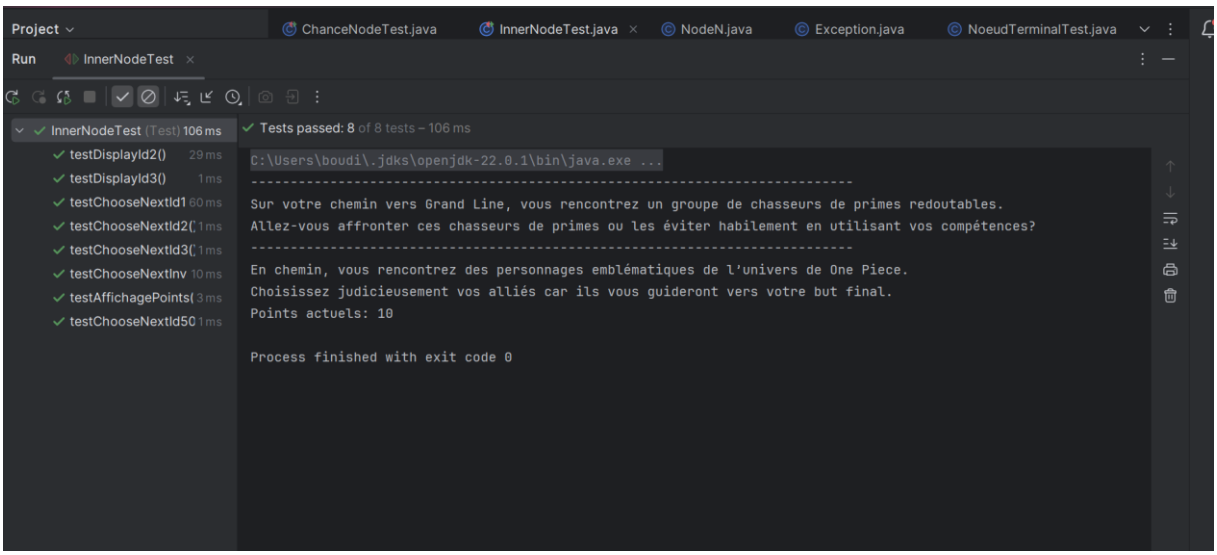
3. LineUnavailableException: Utilisée lorsque la ligne audio nécessaire pour la lecture ou l'enregistrement n'est pas disponible.

6. Tests Unitaires :



Dans chaque classe on teste les méthodes qui sont déjà existantes dans les packages representation et univers.

Par exemple le test de InnerNodeTest :



7. Conclusions :

Pour conclure ce projet, nous avons développé une application interactive basée sur des structures de données sophistiquées telles que les arbres de décision et les nœuds, permettant de créer des scénarios interactifs pour les utilisateurs. À travers l'utilisation de Java et des bibliothèques comme Swing, JPanel, et JFrame, nous avons construit une interface utilisateur intuitive et graphique qui permet à l'utilisateur de naviguer à travers divers choix et décisions, influençant ainsi le déroulement du scénario.

En résumé, ce projet nous a permis d'explorer et d'appliquer divers concepts de programmation orientée objet, de gestion d'interfaces utilisateur et de gestion d'exceptions dans le contexte d'un jeu interactif, enrichissant ainsi notre compréhension pratique des concepts théoriques étudiés en informatique.