Université Caen Normandie

Rapport UE Projet 1
(L2 informatique)

Soumis par :
David Tom
Pépin Florian
Huron Emilien

12 avril 2024
Campus 2 CAEN

Table des matières

1 Introduction

1.1 Description générale du projet

Dans le cadre de l'UE Projet 1, de deuxième année de Licence Informatique, notre groupe, constitué de Tom David, Florian Pépin et Émilien Huron, s'est investi dans le développement d'un analyseur de livres dont vous êtes le héros (LDVEH). Nous avons utilisé le langage de programmation Java et les principes de la programmation orientée objet (POO). Cela nous a permis de mettre en pratique nos compétences en interface graphique et en algorithmique, acquises au cours des deux premières années de notre formation.

Les livres dont vous êtes le héros (LDVEH) offrent une expérience interactive où le lecteur influence le déroulement de l'histoire en fonction des choix qu'il effectue. Notre objectif principal dans ce projet est de concevoir un outil permettant d'analyser la structure de ces livres afin de les représenter sous forme de graphes.

Ce qui a particulièrement retenu notre attention dans ce projet est la possibilité de combiner une représentation graphique des LDVEH avec des éléments interactifs tels que les combats, ajoutant ainsi une dimension ludique à notre travail. En plus de l'analyse de la structure narrative et de la création du graphe, nous avons développé des classes visant à enrichir les fonctionnalités autour du graphe pour permettre une analyse complète des livres au format JSON.

1.2 Présentation du plan du rapport

Dans ce rapport, nous présenterons les fonctionnalités implémentées, les éléments techniques utilisés, ainsi que l'architecture globale du projet. Nous conclurons en récapitulant nos principales réalisations, en proposant des pistes d'amélioration et en partageant notre ressenti par rapport à ce projet.

2 Objectifs du Projet

2.1 Description de la problématique

La problématique centrale de notre projet réside dans l'analyse des LDVEH à travers une représentation graphique, tout en intégrant des éléments de jeu tels que les combats et la gestion des personnages. L'un des premiers défis auxquels nous avons été confrontés était la compréhension du format JSON utilisé pour stocker les informations du livre. Ce format de fichier organise les données en sections, ce qui nous a poussés à réfléchir à la manière optimale d'y accéder et de les utiliser efficacement.

Nous devons également tenir compte de la diversité des chemins possibles dans ces livres interactifs, avec de nombreuses spécificités propres à chaque section, pour certaines uniques dans le livre, ainsi que de leur impact sur la progression de l'histoire. De plus, l'introduction d'éléments de jeu tels que les combats a ajouté une dimension supplémentaire à notre analyse, exigeant une adaptation pour créer un analyseur aussi complet que possible.

2.2 Description de travaux existants sur le même sujet

Plusieurs projets d'analyseurs de livres dont vous êtes le héros (LDVEH) existent déjà, cela montre la diversité dans ce domaine. Parmi les travaux existants :

Premièrement il y a le projet "Fabled Lands Engine" développé par Megara Entertainment, une société spécialisée dans les jeux de rôle et les livres interactifs. Ce projet est sur une plateforme en ligne et sert à explorer et jouer aux célèbres livres. L'analyseur développé par Megara Entertainment combine une interface graphique interactive avec une représentation visuelle des choix disponibles pour les joueurs.

D'autres projets, tels que "LDVEH-Analyzer" sur GitHub, proposent des outils d'analyse et de visualisation des LDVEH, souvent accompagnés de fonctionnalités permettant d'explorer les différents choix possibles dans les livres. Ces projets permettent de voir comment étudier la structure des LDVEH et comprendre ces histoires interactives de l'interieur.

3 Fonctionnalités implémentées

3.1 Differentes fonctionalités

3.1.1 Lecture des Jsons

La première chose que nous avons faite après avoir choisi ce sujet était de rendre accessible les parties du JSON. Nous avons donc immédiatement commencé par créer une classe JsonReader, qui permet à n'importe qui de fournir un fichier livre.json en paramètre, puis de récupérer ce livre sous forme de JsonNode, similaire à une Map. Nous avons conçu des fonctions rudimentaires pour permettre à toutes les personnes impliquées dans le projet de comprendre rapidement le fonctionnement global et la manière d'initialiser un livre afin de pouvoir le lire simplement.

3.1.2 Création de points

Au début du projet, en examinant le LDVEH, nous nous sommes rapidement rendu compte que le livre représentait un graphe, avec chaque section représentée par un point. Nous avons donc décidé immédiatement de créer une classe Point, qui permet de compartimenter tous les nœuds du livre pour une lecture et des parcours plus simples. Nous avons ainsi développé diverses fonctions, plus ou moins simples, tout au long du développement, afin d'améliorer la lisibilité du code pour les autres membres du projet. Chaque Point représente une section du livre avec des listes pour les enfants, les parents, etc.

Cependant, pour une meilleure gestion des points, qui commençait à devenir compliquée compte tenu du nombre important de sections, nous avons tous convenu de créer une classe PointManager. Cette classe contient toutes les fonctions utiles pour la gestion des points, telles que la recherche de points via un ID, la gestion des parents, etc. Cette classe joue un rôle majeur dans le projet, car elle constitue l'axe principal de toute manipulation de points et de lecture de sections.

3.1.3 Gestion du graphique

Le début de la création de cette partie a été compliqué car nous avons voulu nous baser sur une bibliothèque Swing qui offrait des algorithmes de graphes parfaits. Nous avons essayé de suivre leur exemple, mais la tâche était beaucoup trop importante, et nous avions déjà perdu beaucoup de temps dessus. Nous avons alors suivi les conseils de notre professeur, qui nous a recommandé de consulter un PDF expliquant le fonctionnement des graphes de force dirigée de Fruchterman-Reingold, ce qui était plus simple à comprendre que la bibliothèque Swing. Nous avons ensuite implémenté le graphe, qui était créé à partir de la classe PointManager, elle-même utilisant la classe Point. Nous avons amélioré les graphismes et ajouté une sorte de pop-up qui affichait les valeurs d'un Point lorsque l'on cliquait dessus. Nous avons également décidé d'organiser les Points sur le graphe de sorte que les Points avec le plus de passages soient plus gros et d'une couleur différente. Cela permet de visualiser clairement l'ensemble du livre, en mettant en évidence les points les plus importants et les plus fréquentés.

3.1.4 Création des parcours

Après avoir terminé la partie concernant le graphe, nous avons décidé de nous attaquer à une tâche qui semblait rapide et simple à réaliser, mais qui s'est révélée bien plus complexe que prévu. La raison principale de cette complexité réside dans le fait que les points peuvent avoir des enfants qui ont déjà été parcourus par eux-mêmes. Autrement dit, les parcours peuvent revenir en arrière.

En conséquence, nous avons réussi à mettre en place des parcours aléatoires complets, ainsi que le parcours le plus court sans utiliser de combat ni de méthodes aléatoires.

Nous n'avons pas eu d'autre choix que d'adopter cette approche pour le parcours le plus rapide, car nous avons réalisé qu'il existe une infinité de chemins possibles pour une infinité de livres, de héros et de choix aléatoires possibles. Ceci est principalement dû au fait que le héros peut avoir une force nulle ou une vie infinie. Ainsi, si la seule condition de victoire du livre est la mort du héros (un exemple peu logique mais possible), alors si le héros a une vie infinie, le parcours devient impossible.

3.1.5 Création de la fenetre de statistique

Dans ce projet, nous avons décidé de créer une fenêtre de statistiques sur le livre choisi. Cela permet à l'utilisateur de se rendre compte de la difficulté du livre et de sa taille. Dans cette fenêtre, nous affichons tous nos parcours avec des statistiques en fonction du parcours, tout en prenant en compte le héros actuel (généré aléatoirement).
Nous pouvons retrouver sur cette fenêtre :

1. Nombre de chemins faits aléatoirement
2. Nombre de fins aléatoires
3. Taille du chemin le plus long fait aléatoirement
4. Taille du chemin arrivé à la fin (vert)
5. Chemin aléatoire
6. Chance de finir le livre de manière aléatoire
7. Chemin le plus rapide (héros et combat compris)
8. Chemin le plus rapide (absolu, sans combat et héros)
9. Mort la plus rapide
10. Nombre de chemins (trim, alternate, randomChoice)
11. Nombre de points
12. Nombre de combats

3.1.6 Gestion des combats

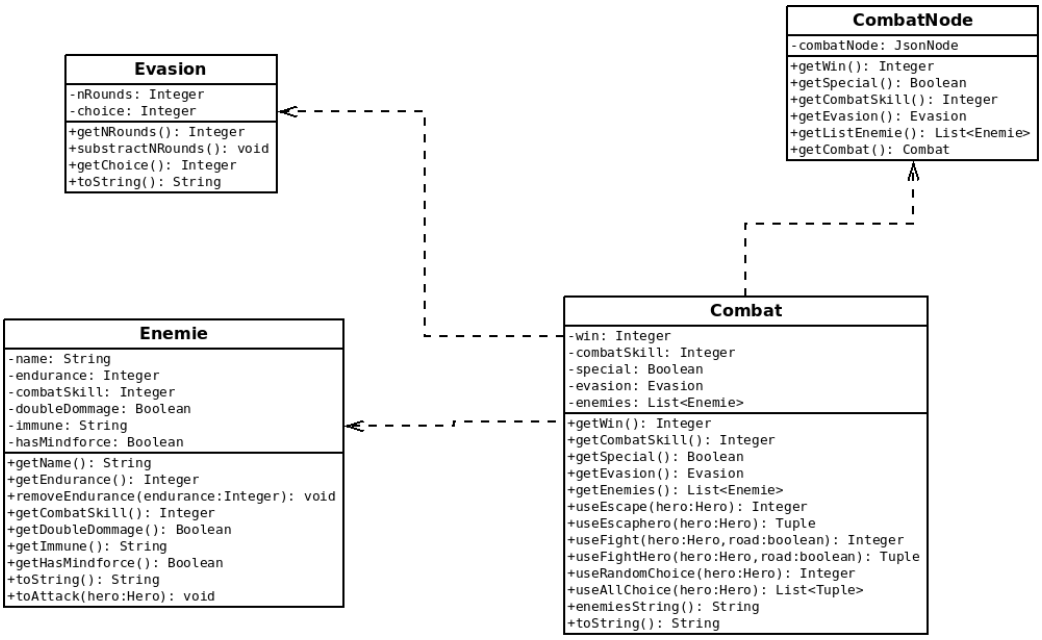

FIGURE 1 – UML : package ldveh.combat

1. La classe CombatNode représente le traitement des informations d'un combat en JsonNode se situant dans une section du LDVEH. Elle contient des informations telles que le nombre d'ennemies, une potentielle évasion ou bien les différents choix de section qui s'offre au hero pendant ou avant le combat.
2. La classe Combat représente la simulation d'un combat dans une section du LDVEH. Il est possible de simuler un combat avec des décisions aléatoires. Mais il est également possible de simuler un combat avec des décisions prédéfinies. Il est également possible de simuler le combat et la fuite afin d'obtenir une liste contenant toutes les possibilités d'un combat.
3. La classe Evasion permet de simuler une evasion lors d'un combat.
4. La classe Enemie permet de simuler un Enemie lors d'un combat.

3.1.7 Gestion des Choix

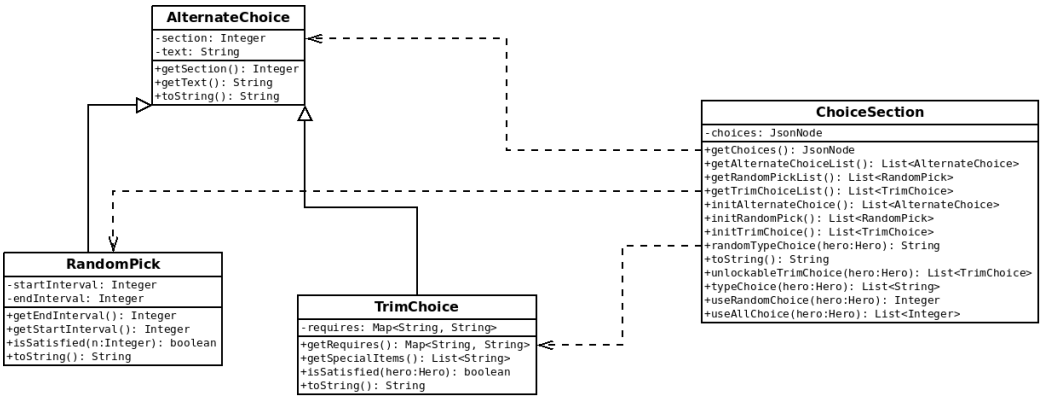

FIGURE 2 – UML : package ldveh.choiceSection

Pour permettre une gestion simple du parcours dans le programme, nous avons créé une classe ChoiceSection qui permet de gérer les différents types de choix d'une section.
Dans cette classe, nous avons 3 listes contenant chacunes 1 type de choix représenté par des classes uniques :

1. TrimChoice : Choix avec "contrainte" qui regarde dans l'inventaire du hero pour voir s'il a les items necessaires pour passer à la section suivante.
2. RandomChoice : Choix "aléatoire" qui choisit aléatoirement une section.
3. AlternateChoice : Choix "alternatif" sont les choix basiques, sans inconvenient.

3.1.8 Création du hero

Nous avons également intégré dans notre projet la possibilité d'avoir un héro créable aléatoirement ou personnalisable. Cela permet d'avoir des parcours plus réalistes. Ce Héro possède des parametres (vie, force, etc...) qui peuvent être modifiées en fonction de son avancement dans le livre.
Grâce à cela, le héros peut perdre de la vie et de l'argent, et peut accéder à certaines sections en fonction de son inventaire. Cela ajoute une nouvelle dimension car une personne qui lance le livre avec des compétences différentes aura un parcours différent. Notez également que la classe Setup est celle qui "initialise" le héro en fonction des règles du LDVEH.

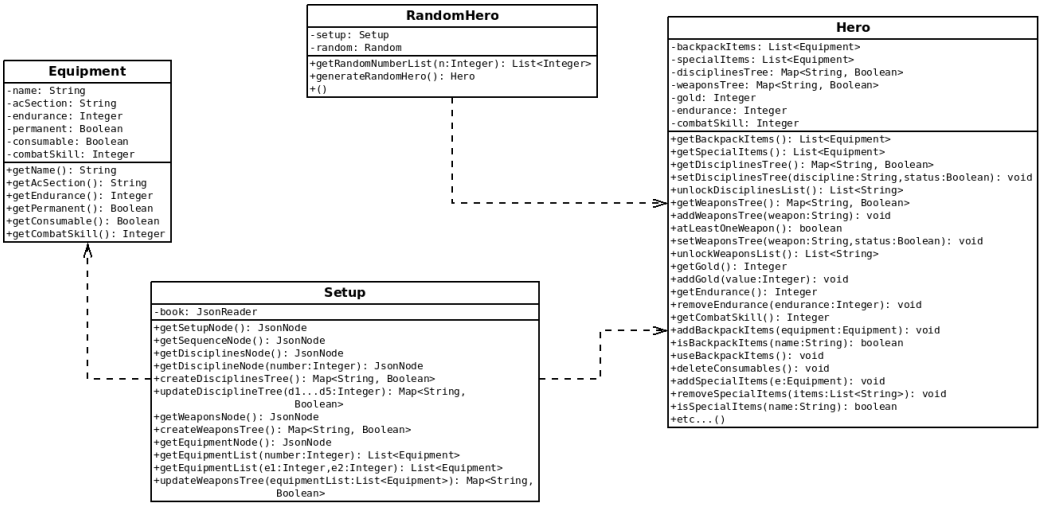

FIGURE 3 – UML : package ldveh.hero

3.1.9 Récolte des items

Il est également possible de récolter des items tout au long de l'histoire. Cet récolte est représentée par la classe Item qui permet de créer un item et de le donner au Hero.

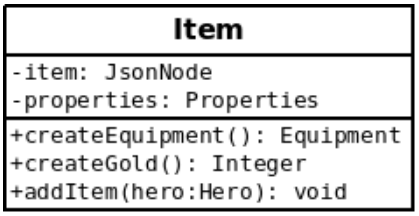

FIGURE 4 – UML : package ldveh.item

3.2 Organisation du projet

Pour assurer une gestion efficace du projet, nous avons adopté une approche collaborative et méthodique. Voici les principales facettes de notre organisation :

1. **Répartition des tâches :** Dès le début du projet, nous avons identifié les différentes composantes et fonctionnalités à développer. Nous avons ensuite réparti ces tâches entre les membres de l'équipe en fonction des class qui n'avait pas ded lien entre elle, pour évité de se marcher sur les pieds. Cette répartition s'est faite de manière équilibrée pour assurer une progression rapide.

2. **Communication régulière :** Nous avons fait des réunions régulières pour discuter de l'avancement du projet, partager les idées et résoudre les éventuels problèmes rencontrés. Ces réunions ont été important pour éviter des incomprehension lors e la fusion de nos partie.
3. **Utilisation d'outils de gestion de projet :** Nous avons etait trés rigoureux sur ce point la, nous avons crée un serveur discord, qui nous a permit de discuter et de transmettre nos informations le plus rapidement possible. Nous avons par exemple fait nos rapport et patchnote pour chaque push que l'on faisait.

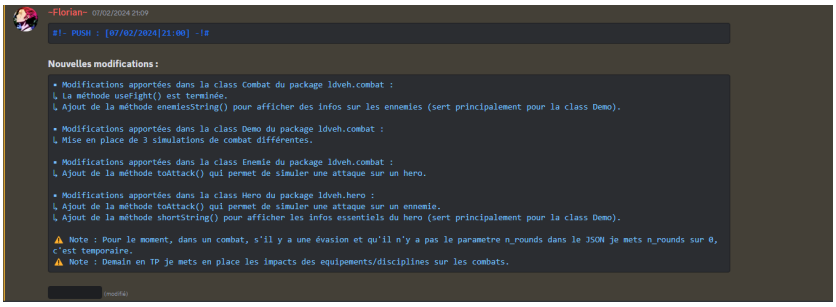

FIGURE 5 – Exemple de PatchNote fait sur Discord

4. **Collaboration et partage des connaissances :** Nous avons encouragé la collaboration entre les membres de l'équipe pour permettre le partage des connaissances et des compétences. Pendant les sessions de travail en groupe nous nous sommes principalement concentré sur la resolution e bug que d'autre n'arriver pas a résoudre. Ce qui a permit a tout le monde de ne pas rester bloqué trop longtemp sur un bug qui etait évident pour une autre personne.
5. **Révisions et ajustements :** Tout au long du projet, nous avons régulièrement révisé notre planification et notre organisation pour nous adapter aux changements et aux imprévus. Nous avons identifié les éventuelles lacunes ou difficultés rencontrées et pris des mesures correctives pour garantir la réussite du projet. Nous nous sommes donc auto-évalués sur les points demandés pour le rendu, ce qui nous a permis de nous concentrer sur les aspects que nous considérions comme les plus importants.

FIGURE 6 – Exemple des taches faite sur Discord

4 Élements techniques

4.1 Paquetages non standards utilisés

Pour ce projet, nous avons été contraints d'utiliser des bibliothèques externes pour effectuer certaines tâches complexes. Nous avons principalement utilisé deux grandes bibliothèques.

Tout d'abord, nous avons utilisé l'ensemble des bibliothèques Jackson, qui nous a permis très simplement de lire et d'ouvrir les fichiers JSON du livre LDVEH.

Ensuite, nous avons utilisé la bibliothèque JGraphX, qui nous a permis de créer le graphe de manière très simple et avec des graphismes plutôt agréables. Nous nous sommes bien évidemment interdits d'utiliser cette bibliothèque pour effectuer automatiquement le traitement du graphe par force dirigée. Elle nous a seulement permis de gérer la partie graphique du graphe et d'établir les liens entre les points plus rapidement.

Enfin, nous avons utilisé une petite bibliothèque nommée JUnit, qui nous a permis de réaliser des tests unitaires plus facilement.

Swing est une bibliothèque d'interfaces graphiques (GUI) pour Java, offrant un ensemble complet de composants graphiques et d'outils pour le développement d'applications avec une interface utilisateur (UI) interactive.

4.2 Algorithmes

4.2.1 Fruchterman-Reingold

L'algorithme utilisé pour la visualisation du graphe est celui de Fruchterman-Reingold, un algorithme connu pour la visualisation de graphes. Il repose sur deux fonctions principales : l'attraction entre les points et l'expulsion des points qui ne sont pas directement liés entre eux.

Pour l'attraction entre les points, nous avons utilisé la loi de Hooke. Cette loi est utilisée pour modéliser les forces d'attraction entre des objets liés par un ressort. Dans notre contexte, elle permet de rapprocher les nœuds du graphe qui sont reliés entre eux, favorisant ainsi une disposition plus ordonnée des éléments du graphe.

Algorithm 1 forceAttraction

```
    Input =  (node1,node2)
    dx ← node1.getX() − node2.getX()
 3: dy ← node1.getY() − node2.getY()
    Distance ← sqrt(dx ∗ dx − dy ∗ dy)
    k ← 0.9
 6: force ← k ∗ Distance
    forceX ← force ∗ (dx/Distance)
    forceY ← force ∗ (dy/Distance)
 9: geometry ← getCellGeometry(node2)
    geometry.setX(geometry.getX() − forceX)
    geometry.setY(geometry.getY() − forceY)
```

Cet algorithme effectue une simple translation d'un point par rapport à un autre point. De plus, il y a une constante qui permet de manipuler la puissance de l'attraction.Cette algorithme utilise la lois de Hooke.

Algorithm 2 forceExpulsion

```
    Input =  (node1,node2)
    dx ← node1.getX() − node2.getX()
 3: dy ← node1.getY() − node2.getY()
    Distance ← sqrt(dx ∗ dx − dy ∗ dy)
    if Distance > 0 then
 6:     k ← 70
        force ← k/Distance ∗ Distance
        forceX ← force ∗ (dx/Distance)
 9:     forceY ← force ∗ (dy/Distance)
        geometry ← getCellGeometry(node2)
        geometry.setX(geometry.getX() + forceX)
12:     geometry.setY(geometry.getY() + forceY)
    end if
```

Pour l'expulsion des points qui ne sont pas liés entre eux, nous avons utilisé la loi de Coulomb. Cette loi décrit les forces de répulsion entre des charges électriques de même signe. Dans notre cas, elle est appliquée pour éloigner les nœuds du graphe qui ne sont pas directement connectés, évitant ainsi la superposition et favorisant une meilleure lisibilité du graphe.

Nous avons ajouté un algorithme personnalisé visant à éviter la superposition des points, ce qui améliore la lisibilité du graphe. Cet algorithme parcourt chaque point du graphe et inflige une force d'expulsion à tous les points qui se trouvent dans sa zone d'influence, c'est-à-dire les points situés au-dessus ou en-dessous de lui.

En résumé, l'utilisation combinée de la loi de Hooke pour l'attraction entre les points et de la loi de Coulomb pour l'expulsion des points non liés, ainsi que l'ajout d'un algorithme personnalisé pour éviter la superposition, permet d'obtenir une représentation visuelle claire et lisible du graphe, améliorant ainsi l'analyse et la compréhension des données.

4.2.2 Chemin le plus court

L'algorithme utilisé pour trouver le chemin le plus court dans le graphe est basé sur une recherche en largeur. Cet algorithme est utilisé pour trouver le chemin le plus court entre deux nœuds dans un graphe.

Nous commençons par définir un point de départ et un point d'arrivée dans le graphe. Ensuite, nous utilisons une file (queue) pour parcourir le graphe de manière itérative à partir du point de départ. À chaque étape, nous visitons un nœud et nous explorons tous ses voisins non visités. Nous maintenons également une carte des distances, qui stocke la distance entre le point de départ et chaque nœud visité.

L'algorithme se termine lorsqu'il atteint le point d'arrivée ou lorsque la file est vide, ce qui signifie qu'il n'y a pas de chemin possible entre le point de départ et le point d'arrivée dans le graphe. Si le point d'arrivée est atteint, l'algorithme retourne le chemin trouvé, qui est le chemin le plus court entre le point de départ et le point d'arrivée.

Nous avons ajouté une fonctionnalité pour vérifier si le chemin trouvé est réalisable. Cette vérification consiste à s'assurer que chaque point dans le chemin a comme enfant le point suivant dans la séquence du chemin. Cela garantit que le chemin trouvé est valide pour le graphe.

Mais nous avons aussi ajouté un parcours le plus rapide qui prend en compte le hero, c'est a dire que chaque parcours a sa propre instance de hero, ce qui permet d'avoir un parcours le plus rapide qui est le plus realiste.

Algorithm 3 findPath

```
1: start ← points[0]
2: end ← points[dernier element de points]
3: queue ← file vide
4: distance ← dictionnaire vide
5: initialPath ← liste vide
6: Ajouter start à initialPath
7: Ajouter initialPath à queue
8: distance[Identifiant(start)] ← 0
9: while queue ≠ vide do
10:     currentPath ← retirer le premier element de queue
11:     currentPoint ← dernier element de currentPath
12:     if currentPoint = end then
13:         return currentPath
14:     end if
15:     for chaque enfant dans currentPoint.obtenirPointsEnfants() do
16:         nouvelleDistance ← distance[Identifiant(currentPoint)] + 1
17:         if Identifiant(enfant) ∉ distance or nouvelleDistance <
    distance[Identifiant(enfant)] then
18:             newPath ← copie de currentPath
19:             Ajouter enfant à newPath
20:             Ajouter newPath à queue
21:             distance[Identifiant(enfant)] ← nouvelleDistance
22:         end if
23:     end for
24: end while
25: return liste vide
```

4.2.3 Mort la plus rapide

Pour rajouter plus de diversité sur les parcours, nous avons décider de crée un parcours qui simule une mort la plus rapide possible, pour cela nous avons utilisé la récurssivté, qui nous a permit trés simplement de faire le chemin vers la mort du hero en question.

Algorithm 4 findDeath

```
    Ajouter point à listePoints
 2: if point = vide then
        return listePoint
 4: end if
    child ← nextSection()
 6: for child dans childs do
        id ← child.getChoice()
 8:     if child.getChoice() = null et point.getId() ≠ 350 ou id = −1 then
            return listePoint
10:     end if
        childPoint ← Pointlist[id − 1]
12:     childHero ← getHero()
        path ← findSmallPathDeath(childPoint, childHero, points)
14:     if path et pas vide then
            return points
16:
        end if
18: end for
```

4.2.4 Parcours aleatoire

Nous avons eux aussi l'idée de faire des parcours aleatoire qui prendrais en compte l'instance du hero,ce qui nous a permis de faire des tas de statistique, comme la probabilité sur 1000 parcours de finir le jeux. Cette fonction a un grand impacte car elle renvoie une liste de liste de tout les parcours, ce qui permet de manipuler les resultat tres facilement, et sa nous permet aussi de faire des etudes large sur les listes.

4.3 Structures de données

Lors de notre projet, nous avons créé plusieurs structures de données afin de faire des listes plus simples à utiliser. Ces listes de données sont implémentées et adaptées seulement pour notre code, on les a défini dans les classes.

1. La première, c'est la liste de points. Elle vient nous permettre de stocker les points de manière efficace pour les réutiliser après et avoir accès à la donnée qu'elle contient très facilement.
2. La deuxième structure de données non-triviale, c'est la liste choice. Il y en a plusieurs pour gérer chaque forme de choix dans le JSON. C'est sous trois formes, RandomPick, TrimChoice et l'Alternate-Choice cette dernière sert de classe de mère pour les différentes formes de choix, chacune avec ses propres méthodes de sélection.
3. Le troisième type de liste que l'on a créé, c'est la liste d'Equipement. Instancier dans la classe Equipement elle permet à celle-ci de gérer tous les objets que le héros va pouvoir rencontrer durant l'histoire de JSON.
4. Enfin le dernier type, c'est la liste d'ennemies. Elle vient faire la même chose que pour l'équipement, mais elle vient prendre tout ce que les ennemies possède.

   Nous avons aussi instancié des classes non-triviales. La première, c'est la classe tuple qui vient stocker l'instance du hero à ce moment-là. Pour les gérer et instancier les points en plus de faire des type de liste nous avons fait la classe Point et PointManager. Elles vont venir créer et gérer les points entre eux de manière plus simple grâce aux fonctionnalités implémentées à l'intérieur de ces classes.

5 Expérimentations et usages

5.1 Cas d'utilisation

Le logiciel est conçu pour offrir une expérience interactive et fluide aux utilisateurs, en les plongeant dans l'univers des livres dont ils sont le héros à travers une interface intuitive et complète.

1. **Traitement des livres JSON :** L'application commence par importer les livres sous forme de fichiers JSON. Ces fichiers sont ensuite traités pour permettre une meilleure lecture et une manipulation simplifiée des différentes sections du livre.
2. **Gestion des choix et des parcours :** Chaque choix dans le livre offre une variété de possibilités, telles que des combats, des choix aléatoires, des choix simples ou des évasions.
3. **Fenêtre statistique :** Une fonctionnalité importante de l'application est la fenêtre statistique, qui affiche un graphique représentant l'ensemble des parcours aléatoires possibles dans le livre. Cette représentation permet aux utilisateurs de visualiser la taille moyenne des parcours, la probabilité de finir le livre de manière aléatoire et les détails des différents parcours, tels que leur taille et leur fréquence.

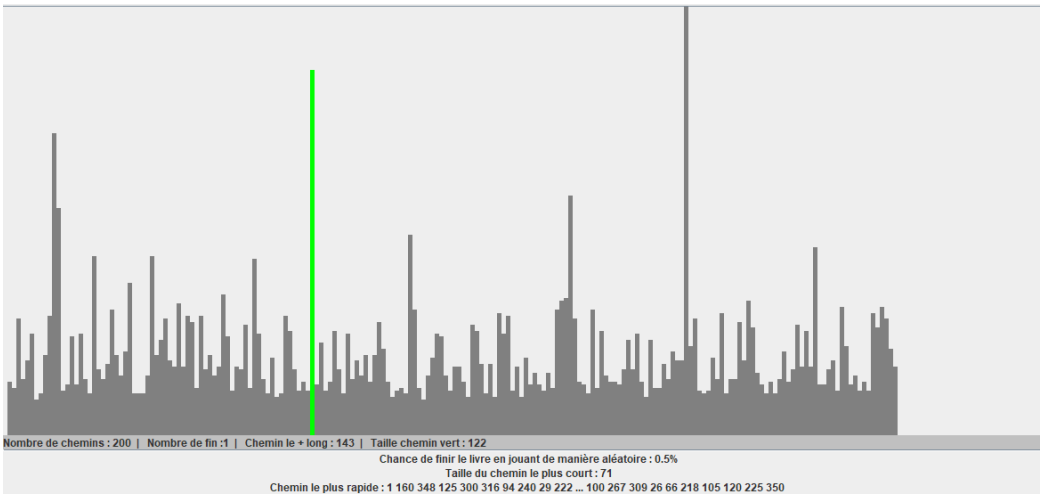

FIGURE 7 – Exemple de la fenètre de statistique

4. **Gestion de l'inventaire :** La gestion de l'inventaire est essentielle pour respecter l'histoire du livre. Certaines sections du livre nécessitent des objets spéciaux ou un certain nombre de points de vie pour progresser. L'application permet aux utilisateurs de gérer leur inventaire de manière à pouvoir avancer dans le livre en respectant les conditions requises.
5. **Création du héros :** Les utilisateurs ont la possibilité de créer leur propre héros de manière aléatoire ou de choisir un héros prédéfini. Cette fonctionnalité offre une personnalisation supplémentaire.
6. **Personnalisation du livre :** L'application offre la possibilité de personnaliser les livres en permettant de changer les messages d'erreur, les noms de fenêtres, etc. Cette fonctionnalité permet aux utilisateurs de créer une expérience unique et adaptée à leurs préférences personnelles.
7. **Affichage du graph :** L'application permet aussi dde faire un affichage le plus claire et simple du LDVEH.En fonction de l'histoire, le graph aura une représentation differente.De plus, si l'utilisateur souhaite voire le text d'une section, un click suffi sur le point en question pour afficher toutes les donnée de cette section.

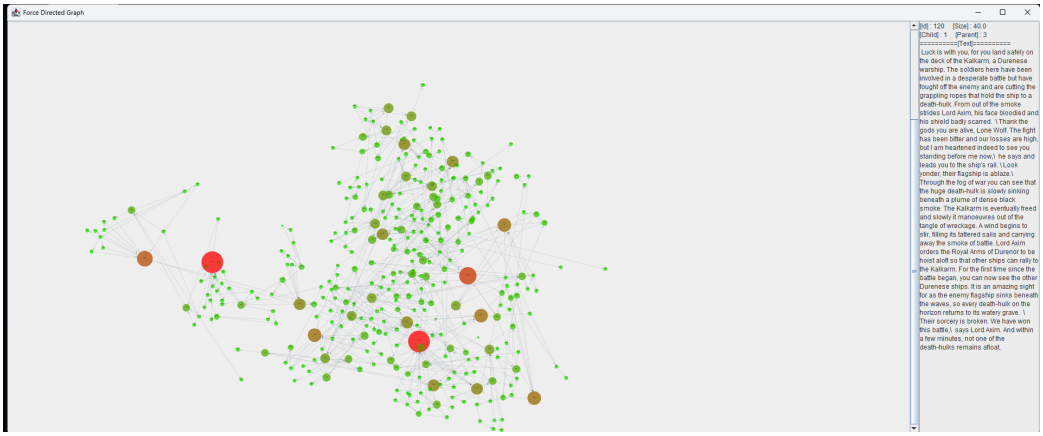

FIGURE 8 – Exemple de la fenètre du graphique

5.2 Résultats quantifiables

Classement	Parcours	Temps
1	SmallPathDeath	<1ms
2	SmallPath	<2ms
3	RandomPath	<4ms

Table 1 – Classement des parcours

Les parcours représentent les différentes méthodes utilisées relier les sections du LDVEH, et leur efficacité joue un rôle important dans l'expérience globale.

SmallPathDeath est le plus rapide, prenant moins de 1 milliseconde pour trouver le chemin le plus court vers la mort dans le LDVEH. SmallPath nécessite moins de 2 millisecondes pour trouver le chemin le plus court. En revanche, RandomPath prend un peu plus de temps, moins de 4 millisecondes, en raison de son approche aléatoire pour naviguer à travers les sections du LDVEH, nécessitant plus de calculs et prenant en compte le héros.

Classement	Parcours	Temps
1	HowManyFight	<0ms
2	TrimChoice	<1ms
3	RandomPath	<1ms
4	AlternanteChoice	<1ms

Table 2 – Classement des compteurs

Dans cette section, nous évaluons les différents compteurs utilisés pour mesurer divers aspects du LDVEH, tels que le nombre de combats ou de choix disponibles.

HowManyFight, avec son temps d'exécution faible, il offre une évaluation rapide du nombre de combats, tandis que TrimChoice et AlternanteChoice fournissent des mesures précises du nombre de choix, respectivement, en moins d'une milliseconde. Cette efficacité dans la recherche des donnée montre la capacité des compteurs à fournir des informations sur la structure du LDVEH rapidement.

6 Conclusion

6.1 Récapitulatif des fonctionnalités principales

En résumé, ce projet comprend environ dix fonctionnalités principales. Tout d'abord, nous avons mis en place la gestion du JSON avec Jackson. Ensuite, nous avons développé la création des points à l'aide de la classe Point, suivie de la gestion de ces points avec PointManager. Nous avons également implémenté la gestion des choix aléatoires, des combats, des fuites et des décès. La création du graphique et des menus s'avère également essentielle dans notre projet.

Nous avons également inclus la manipulation des parcours et l'affichage de statistiques avancées via des menus graphiques simples. La gestion de l'inventaire et des objets constitue une autre composante majeure du projet. Enfin, nous avons développé la gestion et la création d'un héros pour une simulation plus réaliste du livre.

Pour conclure, nous avons ajouté un fichier de configuration permettant à chacun de créer sa propre histoire avec un nouveau livre, complétant ainsi l'ensemble des fonctionnalités de notre projet.

6.2 Propositions d'améliorations

Bien que le projet ait atteint ses objectifs initiaux, des axes d'amélioration demeurent. Une intégration plus poussée des mécanismes de rencontres et de combats pourrait ajouter une meilleure interaction et des choix differents.

De plus nous pourrions augmenter le nombre de parcours et de statistique présent pour avoir une meilleur représentation du LDVEH.

Mais nous pourrions aussi rajouter, un systeme plus performant pour l'affichage du graph, qui est loin d'etre parfais, comme une fonction qui permet d'eviter que les edges se croise ou que les point se superpose.

En conclusion, bien que le projet soit déjà fonctionnel et accompli, l'exploration de ces pistes d'amélioration pourrai le perfectionner davantage.