Compte Rendu du Projet SAÉ

1. Explicitation des fonctionnalités réalisées :

1. Chargement des joueurs et des scores :

Le logiciel charge les pseudos des joueurs depuis un fichier et les scores associés dans un tableau dynamique pour être utilisés dans le jeu.

2. Menu principal et gestion des parties :

Le menu principal permet à l'utilisateur de démarrer une nouvelle partie, d'afficher les scores ou de quitter le jeu.

3. Mise en combat et calcul des points :

Lors d'un combat, le joueur et le monstre s'affrontent. Le logiciel calcule les points en fonction des dégâts infligés et des actions réalisées pendant le combat.

4. Sauvegarde et affichage des scores :

Les scores sont sauvegardés dans un fichier texte trié, permettant de garder une trace des meilleurs scores du jeu.

2. Structure des fichiers utilisés :

Fichier des joueurs (joueurs.don)

Format : Texte

Structure interne :

La première ligne contient le nombre de joueurs dans le fichier.

Les lignes suivantes contiennent les pseudos des joueurs, chaque pseudo sur une ligne séparée.

Fichier score des joueurs (« pseodo du joueur ».txt), chaque joueur à son fichier.

Format : Texte

Structure interne :

La première ligne contient le nombre de parties.

Les lignes suivantes contiennent les scores d'un joueur triés par ordre décroissant, chaque score sur une ligne séparée.

Fichier de partie :

Format : Texte

Structure interne :

La première ligne contient le nombre de groupe de monstre à affronté

Les lignes suivantes contiennent le nombre de monstre par lieu, le lieu d'affrontement et le niveau des monstres.

Choix du format :

Le format texte est simple à manipuler, lisible et facile à débogguer. Cela permet de facilement éditer le fichier et de charger ou sauvegarder les données.

3. Structures de données choisies pour stocker les données en mémoire

Joueurs

Structure utilisée : Tableau dynamique (pour stocker plusieurs joueurs).

Raison du choix : Un tableau dynamique permet d'ajouter des joueurs de manière flexible sans avoir à connaître à l'avance le nombre exact de joueurs. Chaque joueur peut être ajouté indépendamment du nombre d'autres joueurs.

Tableau dynamique de Joueurs:

[Joueur1, Joueur2, Joueur3, ...]

Chaque Joueur est une structure contenant des attributs comme pseudo, pdv, nbDeg, score, etc.

Scores

Structure utilisée : Tableau d'entiers (pour stocker les scores).

Raison du choix : Un tableau est adapté pour stocker les scores, car le nombre de scores est relativement fixe et il permet un accès rapide.

Schéma mémoire :

Tableau de scores:

[score1, score2, score3, ...]

Monstres

Structure utilisée : Liste chaînée (pour gérer les monstres à différents niveaux).

Raison du choix : Une liste chaînée permet d'ajouter et de supprimer des monstres de manière flexible sans avoir à déplacer ou redimensionner des tableaux. Cela est pratique car le nombre de monstres peut varier au fur et à mesure du jeu.

Schéma mémoire :

Liste chaînée de Monstres:

[Monstre1 -> Monstre2 -> Monstre3 -> ...]

Niveaux

Structure utilisée : Liste chaînée (FileN).

Raison du choix : Les niveaux sont représentés par une liste chaînée afin de faciliter l'ajout et la suppression des niveaux au fur et à mesure du déroulement de la partie.

Schéma mémoire :

Liste chaînée de niveaux:

[Niveau1 -> Niveau2 -> Niveau3 -> ...]

4. Comparaison des algorithmes de tri et de recherche

Algorithmes de tri

Tri par insertion (utilisé pour les scores des joueurs) :

Avantages : Facile à implémenter, performant sur des tableaux de taille petite à moyenne.

Inconvénients : Moins performant pour des tableaux de grande taille (complexité $O(n^2)$).

Application : Utilisé pour trier les scores des joueurs afin d'afficher les meilleurs scores à la fin de chaque partie.

Inconvénients : Peut être plus complexe à implémenter dans certains cas.

Application : Permet d'insérer un score dans un tableau déjà trié, tout en maintenant l'ordre des scores.

Algorithmes de recherche

Recherche linéaire (utilisée pour trouver une position d'insertion de score) :

Avantages : Simple à implémenter et efficace sur des petites tailles de données.

Inconvénients : Plus lent pour des ensembles de données très grands (complexité $O(n)$).

Application : Recherche d'une position d'insertion pour les scores dans un tableau non trié.

Recherche dichotomique (utilisée pour le tri dichotomique des pseudos) :

Avantages : Plus efficace que la recherche linéaire (complexité $O(\log n)$).

Inconvénients : Nécessite que les données soient préalablement triées.

Application : Recherche rapide de la position d'un pseudo dans un tableau trié de pseudos.