Projet POO-COO

Ce projet s'étend sur 5 séances de TP de 3h **: 2 séances en POO et 3 séances en COO**. Vous devez développerez le code de l'application en POO et créer la documentation de ce code en COO. Vous aurez deux livrables distincts à déposer sur UMTICE (le zip du code sur POO et la documentation sur COO) et obtiendrez donc 2 notes d'examen pratique distinctes.

Objectifs

- Concevoir et développer un projet complet selon la logique orientée-objet
- Savoir utiliser les diagrammes utiles pour concevoir et documenter le projet
- Comprendre les relations entre COO et POO

Sujet

Le sujet est celui du **donjon-RPG textuel** : il s'agit pour le joueur de parcourir les différentes pièces d'un donjon. Le but peut être de sauver et/ou de combattre quelqu'un, de trouver quelque chose, etc. => vous êtes libres de choisir ce qui vous intéresse en fonction de vos compétences/souhaits.

Voici une liste non exhaustive mais priorisée des fonctionnalités attendues que peut aborder votre projet et des propositions de fonctionnalités optionnelles supplémentaires :

- [OBLIGATOIRE] gestion des déplacements dans le donjon (les 4 directions cardinales)
  - [OPT] prise en compte de l'orientation du joueur (porte à « gauche » peut être la porte située à l'est si le joueur vient de la salle nord !)
- [OBLIGATOIRE] gestion du donjon : les salles sont agencées en dur dans le code
  - [OPT] le donjon est généré aléatoirement
  - [OPT] le donjon est construit à partir des données lues dans un fichier externe
  - [OPT] les portes entre les salles peuvent être fermées et nécessiter une clé
- [OBLIGATOIRE] gestion de la fin de partie (succès ou game over)
  - [OPT] possibilité de sauver/reprendre la partie (persistance des données)
  - [OPT] proposer un menu avec possibilités de relancer une partie, changer des options, etc.
- [OBLIGATOIRE] gestion d'un seul joueur (interactions au clavier pour préciser les actions souhaitées)
  - [OPT] les interactions proposées dépendent du contexte (exemple : on ne propose pas d'aller en face/nord s'il n'y a pas de porte dans cette direction)
  - [OPT] le joueur a des points de vie qui seront utilisés lors des combats/pièges
- [OPT] gestion d'ennemis/pièges
  - [OPT] les salles peuvent avoir un ennemi qu'il faut combattre pour passer

    - [OPT] possibilité de fuir un combat
    - [OPT] gestion du combat (tour, attaque, parade, coup critique, etc.)
    - [OPT] gestion des armes/sorts si le joueur en a plusieurs dans son équipement
    - [OPT] les ennemis morts ne ré-apparaissent pas si l'on revient dans leur salle
    - [OPT] les ennemis battus peuvent lâcher des objets
    - [OPT] les salles peuvent avoir des pièges qui s'appliquent automatiquement
    - [OPT] les pièges peuvent être désamorcés
- [OPT] gestion d'objets
    - [OPT] le joueur a un équipement (clés, armes, etc.)
    - [OPT] les salles peuvent proposer des coffres
    - [OPT] les coffres peuvent contenir 1 objet ou être vide (ou plusieurs objets)
    - [OPT] les coffres peuvent être verrouillés et nécessiter une clé
    - [OPT] proposer divers objets et actions associées (fioles d'HP, etc.)
- [OPT] autres

Au delà de ces fonctionnalités le véritable objectif de ce projet est de « penser » en objet votre application : **identifier les bons objets en interaction et les implémenter en encapsulant des structures de données pertinentes**. Ce point est essentiel et central : l'évaluation y sera très attentive !

Organisation et livrables

- Le projet sera réalisé **individuellement**.
- Le projet s'étend sur 4 semaines incluant 2 séances TP de POO et 3 séances TP de COO (le temps de travail personnel sera variable et dépendant du travail réalisé en TP).
- Vous aurez à rendre un **livrable logiciel** (le code complet – sources et binaires) et **la documentation de votre logiciel** (document PDF). Le tout doit être mis dans un fichier ZIP portant les noms de l'étudiant ou du binôme et **déposée sur l'espace-cours POO** au plus tard le **vendredi 22 mai 2020**. Pour ne pas vous obliger à faire deux dépôts, le prof de POO donnera une copie des fichiers à la prof de COO. Il y aura des pénalités de retard.

Évaluation pour POO

Il vous faudra mettre en avant votre bonne compréhension et application des principes de bases de la POO à travers votre code.

L'évaluation portera sur les aspects suivants :

- qualité du code (respect des conventions de codage, sources commentées) ;
- qualité de la POO ;
- qualité des algorithmes et codes ne rentrant pas le champ de la POO ;
- les fonctionnalités produites.

Il est possible que les personnes impliquées dans un même binôme n'aient pas la même note si une implication non homogène est détectée lors des séances encadrées. La réutilisation totale comme partielle de code entre groupes sera sanctionnée. Toutefois, il n'y pas d'inconvénient à ce que les différents groupes échangent oralement sur leurs intentions et techniques de POO.

Evaluation for COO

The documentation must be written **in English**.

The documentation should be in 1 **folder** containing:

¡ A **PDF file called "code documentation"** with the following information:
- **Game description**
  - Short explanation of the game.
  - List of the developed functionalities.
- **Code documentation**
  - The documentation must present **detailed information** of the important classes and methods. In other terms, all the information that is necessary for another development team to understand and continue your work.
  - You can use **diagrams** to facilitate the comprehension of the text. You can use any type of tool to make these diagrams.
  - You can **copy pieces of code** to explain them.

¡ The **Javadoc folder**, generated with Eclipse
- All the classes, attributes and methods need to be commented in English, with the Javadoc format.

The evaluation will take the following criteria into account:
- Quality of the textual explanations
  - the text is easy to understand
  - the parts are well organized
- Diagram relevance
  - the diagrams help understand
  - the diagrams cover all the complicated parts of the code
- Diagram correctness
  - the diagrams use the correct UML syntax and semantics
- Diagram presentation
  - the diagrams are readable (not too small)
  - the elements in the diagram are well organized
- Coherence between diagram and code
  - the elements in the diagrams represent the elements of the code accurately (names of classes, attributes, methods...)
  - the diagram and coherent with each other (use the same terms and syntax)

- Completeness of Javadoc
    - all the classes, methods and attributes have clear comments