Projet : un jeu d'aventure en mode texte

Consignes :

Le projet est à effectuer par trinôme.

Les livrables :

- une archive nommée `nom1_nom2_nom3.zip`, **à déposer sur UPdago**, contenant :
  - l'ensemble des fichiers source **organisé en packages**,
  - l'ensemble des fichiers de test (vous indiquerez de quels types de tests il s'agit),
  - un fichier `README.txt` indiquant comment installer et utiliser votre programme,
- un rapport (nbPages<20), en français, **à déposer au secrétariat**, dans lequel vous présenterez :
  - la partie documentation utilisateur indiquant comment utiliser votre jeu (version détaillée du README),
  - la partie documentation développeur, présentant la conception de votre projet. En particulier, vous présenterez le diagramme de classes, ainsi qu'un diagramme d'états et de séquences que vous jugerez pertinents.
  - L'organisation du groupe et la répartition des tâches (qui a fait quoi)

La soutenance **en anglais** :

- présentation du projet (10 minutes) :
  - un diaporama détaillant les choix de conception, les spécificités de votre projet, l'organisation du groupe, un bilan fonctionnel et une brève démo.
  - questions (10 minutes)

L'ensemble des documents est à déposer avant le vendredi 03/12/2018 11h00.

Le but de ce projet est de programmer un petit jeu d'aventure en mode texte à la manière de Colossal Cave Adventure [1]. Ce jeu, créé dans les années 70 est considéré comme le premier jeu d'aventure interactif où le programme décrit une situation (i.e. par un affichage texte) et où le joueur indique ce qu'il souhaite faire en saisissant du texte. Le programme analyse la phrase entrée par l'utilisateur et réagit à son tour en affichant la nouvelle situation, et ainsi de suite. Colossal Cave Adventure se déroule dans une grotte, mais vous pouvez créer l'univers de votre choix.

Les classes principales :

— **Des lieux (classe `Place`)**
Vous devez modéliser votre terrain de jeu (i.e. la carte de votre aventure) par un ensemble de lieux où le joueur peut se trouver (par exemple, si le jeu se déroule dans l'espace,

1. Des informations sur ce jeu sont disponibles aux adresses http ://www.rickadams.org/adventure/ et http ://en.wikipedia.org/wiki/Colossal_Cave_Adventure

chaque lieu peut correspondre à une planète ; alors que s'il se passe dans un bâtiment, chaque lieu pourra correspondre à une pièce). Au cours de l'aventure, le joueur se déplace d'un lieu à l'autre en franchissant des sorties.

— **Des sorties (classe `Exit`)**
Chaque lieu contient un ensemble de sorties permettant de se rendre dans les lieux voisins. C'est vous qui définissez le plan de votre terrain de jeu. Il peut bien évidemment exister différents types de sorties (des portes toujours ouvertes, des portes qui nécessitent une clé, un code secret...). Ainsi, chaque sortie connait son lieu voisin.
Notez que l'on l'on pourra tenter de franchir n'importe quel type de sortie, mais on ne franchira pas de la même manière une porte ouverte et une porte nécessitant une clé.
Notez également qu'une sortie est unidirectionnelle (i.e. permet éventuellement de se rendre d'une pièce A vers une pièce B, mais rien ne garantit qu'une fois dans la pièce B, on y trouvera une sortie vers la pièce A.
Depuis un lieu donné, il sera donc possible de se rendre à un lieu voisin à condition d'être en mesure de franchir la sortie correspondante. L'ensemble des sorties d'un lieu sera stocké sous la forme d'un dictionnaire de type `Map<String,Exit>` permettant d'associer les noms des lieux voisins avec leur sortie correspondante. Pour se rendre dans un lieu voisin, il faudra donc franchir la sortie associée à l'aide de la commande `GO` (cf. section Commandes).

— **Des objets et des personnages**
Afin de pimenter un peu votre jeu, chaque lieu pourra contenir des objets (des armes, de la nourriture, des coffres...) et/ou des personnages avec lesquels le joueur pourra éventuellment interagir (discuter, combattre, ...).

— **Pas d'aventure sans héros**
Le jeu consiste principalement à donner des instructions au héros de votre aventure, pensez donc à établir ses caractéristiques (e.g. points de vie, liste d'objets qu'il possède...) qui pourront/devront évoluer au cours du jeu.
Le joueur devra interagir avec le programme via une console. Vous devrez donc mettre en place un système d'analyse du flux de l'entrée standard. Plus précisément, le programme devra analyser mot par mot chaque ligne de texte entrée par l'utilisateur. Pour cela, vous utiliserez un objet `Scanner` instancié à partir de l'entrée standard `System.in`.
Le programme devra "comprendre" au minimum les commandes décrites dans la section Commandes, mais vous pouvez ajouter toutes les commandes nécessaires au bon déroulement de votre jeu.

— **Pas d'aventure sans objectif**
Il vous faut donc établir un but, une quête, ce que vous voulez qui permette de dire que la partie est gagnée, ou à l'inverse perdue.

— **Commandes**
Votre jeu doit permettre à l'utilisateur d'effectuer des actions en saisissant une suite de phrases. Dès que le caractère "fin de ligne" est saisi, la phrase est analysée entièrement. Une phrase bien formée sera de la forme `COMMAND [ARGS]`, où COMMAND correspond à une commande du jeu, et `ARGS` est une liste de mots (éventuellement vide).
Votre jeu doit comprendre au minimum les valeurs `GO`, `HELP`, `LOOK`, `ATTACK`, `TAKE`, `USE`, `QUIT`. Vous êtes bien entendu libre d'en ajouter autant que vous le souhaitez. Votre conception doit permettre d'ajouter simplement une nouvelle commande. Voici un rapide

descriptif du fonctionnement des commandes de base dans la console :

- `GO location` : la commande `GO` est suivie du nom du lieu voisin où l'utilisateur souhaite se rendre. Si le lieu `location` existe et que la sortie est bien franchissable, alors le personnage s'y rend, sinon il ne change pas de lieu et un affichage indiquera à l'utilisateur ce qui se passe.
- `HELP` : indique l'ensemble des commandes disponibles
- `LOOK [observables]` : affiche un descriptif du lieu courant si aucun argument n'est ajouté. Si une liste d'arguments est indiquée, un descriptif de tout ce qui peut être observé sera affiché.
- `TAKE takable` : ajoute (si cela est possible) l'objet à la liste d'objets du joueur.
- `QUIT` : quitte la partie.
- `USE item1 [item2]` : utilisation de l'objet désigné par le premier argument. Si un second argument est renseigné, le premier objet est utilisé avec le second. Par exemple, on peut imaginer que l'instruction `use gun bullet` permette de charger `gun`, ce qui pourra être utile pour une future utilisation.

Fonctionnalités supplémentaires

- Utilisation de compte à rebours (classes fournies). A vous de trouver une utilisation pertinente (déclenchement d'une bombe, temps limité pour terminer le jeu...)
- Gestion d'exceptions,
- Enregistrement/chargement de parties (Sérialization),
- ...