Jeu d'aventure projet Zull
(A3P(AL) 2023/2024 G5)

I. a. Auteur(s)
   b. Thème (phrase-thème validée)
   c. Résumé du scénario (complet)
   d. Plan (complet, avec indication de la partie "réduit" si exercice 7.3.3)
   e. Scénario détaillé (complet, avec indication de la partie "réduit" si exercice 7.3.3)
   f. Détail des lieux, items, personnages
   g. Situations gagnantes et perdantes
   h. Éventuellement des énigmes, mini-jeux, combats, etc.
   i. Commentaires (ce qui manque, reste à faire, ...)
II. Réponses aux exercices (à partir de l'exercice 7.5 inclus)
III. Mode d'emploi (si nécessaire, instructions d'installation ou pour démarrer le jeu)
IV. Déclaration obligatoire anti-plagiat (*)

(*) Cette déclaration est obligatoire :
- soit pour préciser toutes les parties de code que vous n'avez pas écrites vous-même et citez la source,
- soit pour indiquer que vous n'avez pas recopié la moindre ligne de code (sauf les fichiers zuul-*.jar qui sont fournis évidemment).

I. Première partie

a. Auteurs :

Gnamien Marie Emilienne

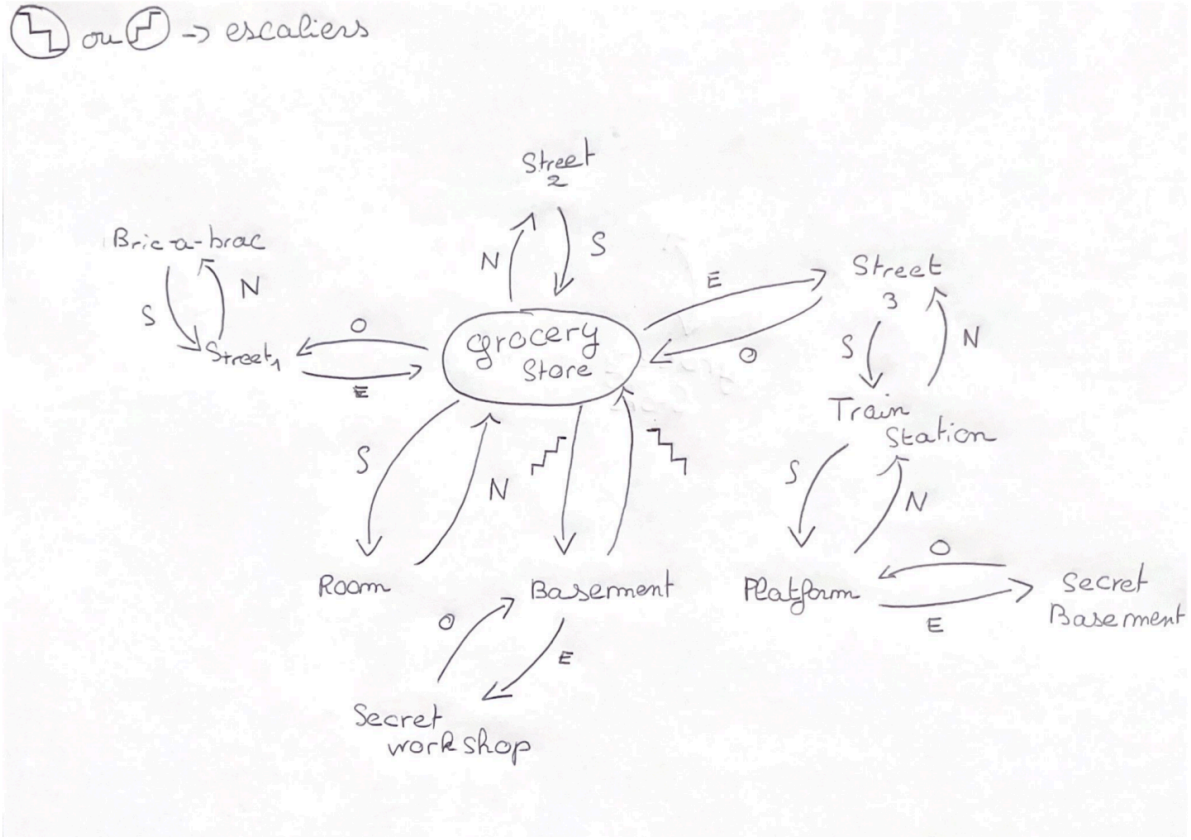
b. Thème :

En 3417, dans un univers cyberpunk : une jeune femme de 19 ans (moitié humaine/moitié androide) doit améliorer son système interne pour pouvoir participer à une compétition de combat.

c. Résumé du scénario :

Le but de l'héroïne est donc de réussir à obtenir des composants nécessaires à l'amélioration de son système. Dans cet univers, il est très compliqué de se procurer des composants de qualitatifs. Mais il y a une autre façon de les obtenir, et cette méthode est illégale. Une fois que cela sera acquis elle pourra donc participer à une compétition de combat très connue des banlieues, malgré le fait qu'elle soit interdite par les autorités.

d. Plan du jeu :



f. Détail des lieux, items, personnages :

- ➔ Lieux :
  - Chambre de l'héroïne
  - Sous-sol de la maison de son Grand-Père
  - Atelier secret du Grand-Père
  - épicerie tenue par le Grand-Père
  - 3 différentes rues (dont une avec un cul-de-sac)
  - Gare de train/métro
  - Quai
  - Sous-sol secret (compétition combats)
  - Brocante
- ➔ personnages :
  - Grand-Père de l'héroïne (lui cache des secrets, l'aide à trouver des composants plus tard dans le jeu, l'avertit mais reste quand même inquiet)
  - 3 amis à l'héroïne (l'un d'entre eux est un traître)
  - les autorités
  - passeur (pour pouvoir participer à la compétition
- ➔ items :
  - clés pour pouvoir entrer dans certaines pièces verrouillées
  - composants utiles à l'amélioration du système de l'héroïne
  - ...

g. Situations gagnantes ou perdantes :

- ➔ Situations gagnantes :
  - Réussit à se procurer les composants sans se faire remarquer (peut renoncer à son combat).
  - Réussit à améliorer son système, gagne son combat et réussit à échapper aux autorités lorsqu'elles découvrent le lieu de compétition secret (si elle n'a pas démasqué son ami traître).
  - démasque son ami.e traître.
- ➔ Situations perdantes :
  - Se fait remarquer en se procurant les composants nécessaires à l'amélioration de son système
  - Ne réussit pas à s'échapper et se fait arrêter par les autorités.
  - Ne réussit pas à se procurer les composants nécessaires.

i. Commentaires (ce qui manque, reste à faire, ...)

- Ajout d'une ou deux nouvelles pièces si besoin.
- réfléchir aux autres situations dans lesquelles il serait possible de perdre.
- ajouter des items en plus.
- rechercher des idées de mini-jeux et énigmes pour le jeu.

II. exercices :

Exercice 7.5 : dans la méthode goRoom de la classe Room
à chaque condition : suppression de l'affichage de la Current Room.
Puis suppression du code qui nous permet d'accéder aux exits.

Exercice 7.6 :
1. Il n'est pas nécessaire de vérifier si chaque paramètre est null. Il suffit simplement de vérifier la valeur de "direction".
2. Pour ma première classe Game, j'ai donc créé mon accesseur "getExits" me permettant d'accéder à mes attributs privés Exits.
J'utilise donc cet accesseur dans ma classe "Game" pour pouvoir attribuer à vNextRoom la pièce dans laquelle l'utilisateur veut se rendre. Grâce à cette ligne, je n'ai plus besoin de vérifier la valeur de vDirection à chaque fois quelque soit la valeur de vNextRoom (dont "null").
J'ai donc supprimé tous les tests vérifiant que vNextRoom est null en les remplaçant par deux tests.

Exercice 7.7 :
J'ai modifié ma méthode printLocationInfo en utilisant mon accesseur getExits sur mes comparaisons.
Pour définir "getExitString()", on retourne juste la liste des différentes sorties de notre pièce "Room".
Il est logique de demander à Room de produire les informations sur ses sorties puisqu'elle a accès aux attributs privés Exits contrairement à Game. Avant de définir "getExitString" il fallait utiliser un accesseur, en plus d'un test pour pouvoir afficher la liste de sorties. Mais maintenant ce n'est plus nécessaire.

Exercice 7.8 :
La méthode qui est désormais inutile est setExits puisque l'on peut maintenant définir les sorties de chacune des pièces une par une en utilisant la méthode setExit. Grâce à cette méthode, il est alors possible de définir d'autres directions dont haut et bas.
Pour modifier la méthode "getExitString", j'ai utilisé la liste de clés existantes dans mon tableau associatif. Chaque clé correspond à la sortie disponible, j'ajoute cette sortie à ma chaîne de caractères vExits. À la fin de cette boucle, vExits affichera donc toutes les sorties disponibles.

Exercice 7.9 :
(voir exo 7.8)

Exercice 7.10 :

Voici comment fonctionne la méthode "getExitString" :
Premièrement, une variable vExits contenant une chaine de charactères est déclarée.
Ensuite, on déclare un ensemble Set (le type des clés est String) contenant l'ensemble des sorties (clés) de notre HashMap exits.
Nous avons ensuite une boucle parcourant toutes les clés de ce tableau associatif.
Chaque clé est donc ajoutée à notre variable vExits déclarée auparavant.
Finalement, vExits est retournée.

Exercice 7.10.2 :
La javadoc de la classe Game est presque vide puisque la majorité des méthodes utilisées sont privées.