Le jeu du chat et de la souris

Dans un labyrinthe, des souris cherchent à s'organiser afin de s'évader tout en évitant des chats affamés. Vous devez les aider ! Dans ce jeu, le labyrinthe est représenté sous la forme d'une matrice de cases. Chaque case peut-être de 8 types :

- Murs : Ils sont infranchissables par les souris et les chats.
- Entrées : Elles correspondent aux entrées du labyrinthe pour les souris.
- Sorties : Elles correspondent aux sorties du labyrinthe pour les souris.
- Chemin : Tout le monde peut se déplacer sur ces cases.
- 4 Flèches : haut, droite, bas, gauche : qui vont permettre au joueur d'aider les souris.

Les chats se déplacent soit sur des lignes soit sur des colonnes du labyrinthe. Ils ne peuvent pas se déplacer de plus d'une case à la fois. Lorsqu'ils rencontrent un mur, ils font bêtement demi-tour. Les souris quant à elles se déplacent en ligne droite, case par case. Pour les aider à sortir du labyrinthe, le joueur positionne des flèches haut, bas, gauche ou droite que les souris sont capables de suivre (elles sont vachement intelligentes !). Le joueur peut ajouter et supprimer des flèches quand il le désire mais il en possède un nombre restreint. Si une souris rencontre un mur, elle tourne de 90° dans le sens horaire.

Le but du jeu est de sauver un maximum de souris. En effet, un chat et une souris sur la même case entraine un combat peu équitable et la mort de la souris...

L'architecture de l'application est basée sur le design pattern MVC (modèle, vue, contrôleur). Dans le projet auquel vous avez accès se trouve la partie vue complète et les classes abstraites : Modèle et Contrôleur permettant le bon fonctionnement de la vue.

1. Faire le diagramme de classe du modèle.
2. Implémenter les classes de votre modèle.
3. Implémenter les méthodes héritées du modèle abstrait :
    a. TypeCase getTypeCase(int x, int y) :
    Retourne le type de la case du labyrinthe au coordonnée x, y. la classe TypeCase existe déjà et définie 8 types de case différente : MUR, IN, OUT, CHEMIN, FLECHE_HAUT, FLECHE_DROITE, FLECHE_BAS, FLECHE_GAUCHE.
    b. int getLargeur() et getHauteur() :
    Retourne la largeur et la hauteur du labyrinthe.
    c. int getNbSourisIn() :
    Retourne le nombre de souris qui ne sont pas en encore sortie du trou d'entrée dans le labyrinthe.
    d. int getNbSourisOut() :
    Retourne le nombre de souris qui sont parvenu à rejoindre l'arrivé.
    e. int getNbFlecheUtilisee() :
    Retourne le nombre de flèche utilisée.
    f. int getNbFlecheMax() :
    Retourne le nombre maximum de flèches disponible pour ce labyrinthe.
    g. boolean partieTerminer() :
    Retourne VRAI lorsque la partie est terminé (par exemple lorsqu'il n'y a plus de souris dans le labyrinthe etc …)
    h. Animal getAnimalPlusFort(int x, int y) :
    Retourne l'animal le plus fort de la case au coordonné x , y, cette **Animal** doit être un **Chat** ou une **Souris** et correspond à ce qui sera afficher à l'écran.

4. Implémenter un contrôleur héritant de la classe abstraite : AbstractControler.
5. Instanciez votre modèle et contrôleur dans le main.
6. Testez votre jeu !

Vous devez concevoir et implémenter cette application de la façon la plus générique possible. Une fois à la base précédemment conçue et développée, vous pouvez ajouter les options que vous désirez parmi les suivantes :

- Des chats plus intelligents qui chassent les souris les plus proches d'eux
- Des chiens défendent les souris en mangeant un chat chacun. Par contre, ils bougent aléatoirement et écrasent les souris.
- Des téléporteurs téléportent chats et souris d'un endroit à l'autre du labyrinthe.
- Un éditeur de niveau permet de produire facilement un niveau.

Nous sommes ouverts à toutes autres extensions !