**Programmation Objet**

**Travail 3 partie 2**

**Pondération : 16%**

**Remise : lundi 9 décembre avant minuit via Léa**

**(n’oubliez pas d’enlever les dossiers bin et obj et le dossier caché .vs)**

Le but de la partie 2 du travail est d’intégrer, dans un même projet, plusieurs des notions apprises dans les 2 cours de programmation objet tout en mettant l’accent sur l’implémentation des différents types de relation entre les classes ainsi que les liens avec une base de données relationnelle à l’aide des ‘repositories’. Il s’agit donc d’une bonne pratique en prévision de l’examen final.

Exigences

* Utilisez la base de données que je vous ai remise. Il est à noter qu’une seule table existe pour l’ensemble des cartes et de leur antidote.
* Vous devez évidemment programmer les ‘repositories’ permettant de manipuler les données de la base de données.
* Il est important d’appliquer les règles du document 1.11 ‘Schéma de classes et base de données relationnelles’ lors de la programmation de ces repositories.
* Tel que demandé par ce modèle, aucune manipulation de la base de données ne doit se retrouver dans les classes métiers, dans la vue et dans le contrôleur.
* **Vous devez utiliser la notion de polymorphisme** lorsque possible. N’oubliez pas que le polymorphisme évite de faire des if lors des traitements.
* Vous pouvez également utiliser les interfaces s’il est pertinent de le faire.
* Vous devez continuer à appliquer une architecture MVC pour structurer votre code.
* Faites attention pour dédier les responsabilités aux bonnes classes et à appliquer le principe « Tell don’t ask ».
* Vos classes doivent déclencher des exceptions lorsqu’il est pertinent de le faire.
* Vous devez respecter le standard de programmation tel que présenté au cour.

Schéma implémentation

Faites un schéma d’implémentation des classes. Le schéma doit :

* montrer les attributs du type d’une autre classe
* les méthodes nécessaires pour manipuler ces attributs
* comporter les méthodes virtuelles que vous voyez présentement

Faites-moi approuver votre schéma avant la fin du cours.

Programmation des classes

* La partie à programmer concerne la manipulation des cartes (carte et antidote), de leur catégorie ainsi que des joueurs.
* Programmez les classes qui correspondent à la partie du projet à programmer en utilisant :
  + Votre schéma d’implémentation des classes
  + Les commentaires que je vous ai donnés concernant ce schéma
  + Les explications concernant l’implémentation des différents types de relations (doc 1.10 sur Léa)
* La classe ‘Carte’ doit être une classe abstraite.

Fonctionnement à programmer

Le but du jeu est de faire le plus de points possible tout en essayant d’en faire perdre aux autres joueurs. Le jeu se joue à 4 joueurs.

Gestion des joueurs

* Au début de l’exécution, les 4 joueurs doivent être choisis parmi ceux qui se trouvent dans le fichier. Soit que vous entrez le nom du joueur et vous le recherchez dans la table ou soit que vous mettez tous les noms de joueurs dans une liste pour effectuer votre choix.
* Pour chaque joueur choisi, il faut afficher son nom et son nombre de points.
* Au début du jeu, chaque joueur reçoit 50 points.
* Les joueurs jouent à tour de rôle.
* Il faut toujours savoir qui est le joueur courant.
* Les noms et points de tous les joueurs doivent être affichés en tout temps.

Tour d’un joueur

* Au début du tour, le joueur courant doit choisir un joueur adverse.
* Un numéro de carte est généré au hasard (pas idéal mais…). Ce numéro indique la carte que le joueur doit jouer.

Note : Ne gardez pas toutes les cartes en mémoire mais seulement celle qu’il faut jouer.

* Toutes les informations de la carte générée doivent s’afficher dans des zones de texte. S’il n’y a aucune donnée à afficher pour une zone de texte alors il suffit d’afficher 0 ou du vide.
* La catégorie de la carte doit s’afficher : potion, sortilège, …
* Le type de carte doit s’afficher : carte d’attaque ou carte de défense.
* Si la carte générée est une carte d’attaque :
  + Le joueur courant attaque le joueur adverse.
  + Les points du joueur adverse diminuent donc d’un nombre généré au hasard par la carte. La formule magique à utiliser doit donner :
    - un chiffre entre le min et max de la carte
  + Heureusement, l’attaque possède un antidote. Les points du joueur adverse augmentent donc de la valeur des points de l’antidote \* le chiffre 1 ou 2 (choisi au hasard).
* Si la carte générée est une carte de défense :
  + Le joueur courant doit se défendre contre le joueur adverse.
  + Les points du joueur courant augmentent alors d’un nombre généré au hasard par la carte. La formule magique à utiliser est :
    - points de la carte de défense \* le chiffre 2 ou 3 (choisi au hasard)
  + Le joueur adverse peut devenir plus faible lorsqu’on se défend contre lui. Ses points doivent donc diminuer de la moitié des points de la carte de défense.
  + Heureusement, il y a un antidote ! On ajoute donc les points de l’antidote au joueur adverse.

Présentation du travail

* Vous devez obligatoirement me présenter votre travail une fois qu’il sera remis.
  + Fonctionnement
  + Explication du code selon les questions posées

Critères de correction

* Respect du schéma commenté
* Mise en place adéquate de la couche d’accès aux données (‘repositories’)
* Respect des principes de la POO
* Utilisation appropriée du polymorphisme
* Utilisation appropriée des autres notions entourant la POO
* Structure MVC et principe « Tell don’t ask »
* Fonctionnement
* Respect des exigences
* Respect des bonnes techniques de programmation et des standards