POO - grand devoir 1

● Pour ce projet vous allez travailler par équipes de 2-3 personnes
● Le projet est à rendre sur la plateforme Moodle. S'il y a des problèmes lors du téléchargement vers ou depuis la plateforme, contactez sur Teams votre assistant des travaux pratiques : Iulia Stanica ou Alexandru Bratosin.
● Un seul upload par équipe suffit!
● Date limite: 8eme semaine (Lundi, 18 novembre 8:00, affichée sur moodle) et présentée pendant le TP de cette semaine-la
● Vous devez être présent à la séance de TP respective pour recevoir les points. Un devoir mis sur Moodle et pas présenté, la note sera 0!
● Pour toute question concernant le sujet du projet ou les exigences, utiliser le groupe de Teams de POO ou envoyer un message sur le chat

Travail avec: classes, relations, interfaces, comparateur/comparable, énumérations, exceptions, fichiers (lecture & ecriture)

Tâches:

Vous devez implémenter un jeu de survie. Le joueur peut se déplacer sur une carte en utilisant les touches W A S D.
Cette carte de jeu est basée sur une matrice carrée, chaque emplacement de la matrice peut être vide (défini sur 0) ou contenir un objet, vous pouvez décider de la représentation de la manière dont la matrice contient les objets ou les ennemis. Si le joueur rencontre un emplacement vide de la matrice, il peut fabriquer un objet sur cet emplacement vide. Si l'emplacement contient un objet, le joueur peut ramasser l'objet et vider l'emplacement. Si le joueur rencontre un ennemi, un combat commencera (le joueur frappe en premier, puis l'ennemi, cela continue jusqu'à ce que l'un d'eux meure).

Les objets peuvent être:
- des arbres
- des rochers
- des céréales.

Ils laissent tomber une certaine quantité de bois, de pierre et de nourriture respectivement. Avec le bois, la pierre et la nourriture, vous pouvez fabriquer des objets ou des bâtiments, mais vous avez besoin d'un emplacement libre sur la matrice pour fabriquer l'objet ou le bâtiment. Si vous créez un bâtiment, cet emplacement sera occupé.

Détails

Vous aurez besoin d'une **classe abstraite** pour généraliser les **objets récupérables** (grains, arbres, rochers).
- *Attributs:* quantité et la qualité (e.g. quantité de bois qu'un arbre laisse tomber lorsqu'il est coupé, la qualité influence la quantité comme multiplicateur)
- Vous devez utiliser une *énumération* pour la qualité (commune, rare, épique) des objets
- *Méthodes abstraites:* Gatherable(), toString().

Vous implémenterez cette classe et chacune des 3 sous-classes d'objets récupérables. Chacune des classes aura des getters, des setters et toString.

La 2ème classe abstraite sera Character, elle sera héritée par les classes Player et Enemy.

- *Attributs* : nom, attaque, statut (mort ou vivant), défense et santé, ensemble d'objets.
- *Méthodes abstraites:* damage(), takedamage(), die().

La classe du **joueur** doit avoir des attributs supplémentaires liés à la détention des ressources : bois, pierre, nourriture, méthodes pour les collecter et gérer les interactions et l'artisanat.
L'ennemi, une fois vaincu, peut laisser tomber un objet (épée, armure, casque, etc.) donc vous devez implémenter des méthodes d'attributs supplémentaires pour gérer cela.

Une autre classe pour les **objets**.

- *Attributs:* nom et attributs qui peuvent améliorer le joueur, ce qui signifie qu'ils peuvent augmenter la santé, l'attaque ou la défense du joueur. Les objets peuvent être fabriqués ou lâchés par les ennemis.

Implémenter un **comparateur** capable de trier les objets (par ex ceux de l'inventaire du joueur), en fonction des critères choisies par vous.

Une **classe pour les bâtiments** pouvant être fabriqués, ils ont une fonction spéciale ex : si vous êtes dans un emplacement avec un bâtiment « fontaine de vie », vous récupérez votre santé au maximum, si vous construisez un « monument d'épée », vous obtenez un boost permanent d'attaque de 10 %, vous pouvez imaginer d'autres effets. En bref, vous avez besoin de méthodes qui modifient les statistiques des joueurs lors de leur création ou de leur interaction. Les bâtiments ou les objets fabriqués coûtent du bois et de la pierre à fabriquer.

Test:

Vous devez travailler avec des fichiers externes (lecture et écriture) pour un scénario de votre choix, par exemple : ouvrir et lire des fichiers qui chargent une liste d'objets prédéfinis et leurs statistiques. Lire des fichiers qui chargent une liste de monstres disponibles. Sauvegarder l'état du jeu dans un fichier, etc. Choisissez ou imaginez une autre utilisation utile des fichiers et implémentez-la (elle doit avoir une fonction pertinente).

Vous pouvez ajouter des attributs/méthodes supplémentaires dans les classes précédentes ou créer de nouvelles classes si vous semble nécessaire.

Vous devez tester votre implémentation dans le main et permettre à l'utilisateur de se déplacer sur la matrice avec les touches. L'exécution du jeu doit être interactive.

Bonus : essayez d'implémenter d'autres comparateurs ou interfaces supplémentaires.

Points:

- Classes abstraites - 1p
- Classes Joueur, Ennemi - 1.5p
- Classes Objet, Bâtiment - 2p
- Sous-classes objets - 0.75p
- Test complexe WASD + affichages interactif: 1.5p
- Lecture et écriture fichier - 1p
- Comparable / Comparator - 1p
- Utilisation Enum - 0.25p
- Utilisation correcte de tous les vérifications, ajout méthodes nécessaires supplémentaires - 1p

Attention! Si les classes ne sont pas testées dans le main pour voir qu'elles fonctionnent, ça va vous faire perdre au minimum ½ de leurs points!