

Checklist itération 1 (non-officielle)

“ **Attention** cette checklist n'est pas officielle, elle est seulement basée sur le PDF de l'itération 1 et général. Pour la checklist officielle cliquez [ici](#)

Contraintes générales

- ☒ La structure des fichiers correspond à celle montrée dans le PDF
- ☒ Le projet utilise la version 11 de Java et intègre l'interface graphique Swing
- ☒ Le projet est compatible avec la version d'Eclipse de référence (voir ressources informatiques sur Learn)
- ☒ Les tests unitaires sont effectués à l'aide de JUnit5
- ☒ Le projet utilise le plug-in PMD avec le ruleset donné sur Learn

Acquis d'apprentissages

- ☒ Respect des principes de la programmation orientée objet
- ☒ Les comportements du programme sont testés à l'aide de tests unitaires
- ☒ Le programme est bien documenté à l'aide de la JavaDoc (et calcul de la CTT si demandée)
- ☒ Les utilisations des interfaces de collections et de ses implémentations sont justifiées

Fonctionnalités

AI-1 : Créer une partie

En tant que joueur, je souhaite démarrer une nouvelle partie afin de l'afficher à l'écran

- ☑ L'application démarre en affichant un menu principal composé de deux items : « Nouvelle partie » et « Quitter »
- ☑ L'activation de l'item « Nouvelle partie » débute la création d'une partie.
- ☑ La création d'une partie commence par le chargement de la carte depuis le fichier `resources/maps/map-sample.txt`. Nous vous fournissons à cette fin la classe `treasurequest.io.CharArrayFileReader`.
- ☑ Une fois la carte chargée, l'application y place des trésors sur les cases creusables. Une case est creusable si elle n'est pas de type Eau.
- ☑ Le nombre de trésors à placer correspond à 10% du nombre de cases creusables, avec au moins 1 trésor à trouver.
- ☑ La valeur d'un trésor est un nombre aléatoire allant de 10 à 20 pièces.
- ☑ Toutes les cases sont non-creusées.
- ☑ Une case active est désignée. Elle correspond à une case du centre de la carte.
- ☑ Le joueur possède une bourse correspondant à 2 fois le nombre de trésors. Cette bourse est exprimée en pièces (s'il y a 20 trésors à placer, la bourse de départ du joueur sera de 40 pièces).

Tests d'intégration

- ☑ Quand je lance l'application, celle-ci me présente le menu principal composé de deux items « Nouvelle partie » et « Quitter »
- ☑ Quand je sélectionne l'item « Quitter », l'application termine son exécution.
- ☑ Quand je sélectionne l'item « Nouvelle partie », l'application change d'écran

AI-2 : Voir une partie

En tant que joueur, je souhaite voir l'état d'une partie afin de décider des actions à entreprendre

- ☑ Une seconde vue présente l'état de la partie.
- ☑ Une carte montre les cases. En début de partie, aucune case n'est creusée.
- ☑ La case active de départ est au « centre » de la carte.
- ☑ Un panneau affiche la bourse du joueur, le nombre de trésors à trouver et le coût à payer pour creuser la case active.
- ☑ Creuser une case Sable coûte 1 pièce. Le coût des autres types de cases est déduit de cette valeur.

Tests d'acceptation

Important : pour les tests d'acceptation, nous utiliserons des cartes autres que celles fournies.

- ☒ Quand je lance une nouvelle partie, l'application me présente la carte donnée dans l'exemple.
- ☒ Quand je lance une nouvelle partie, la case active est celle au « centre » de la carte
- ☒ Quand je lance une nouvelle partie, l'application me dit qu'il y a XX trésors à trouver, que le joueur a une bourse de 2*XX
- ☒ Quand je lance une nouvelle partie, l'application m'affiche le cout correct de la case active

AI-3 : Revenir au menu principal

En tant que joueur, je peux abandonner une partie et revenir au menu principal, afin d'en relancer une nouvelle.

- ☒ Appuyer sur la touche « Esc » quand une partie est en cours permet de revenir à l'écran principal.
- ☒ Demander une nouvelle partie relance la création d'une nouvelle partie. En particulier, le rechargement de la carte.

Tests d'acceptation

- ☒ Soit une partie lancée avec une case creusée, quand je reviens au menu principal et que je relance une nouvelle partie, le jeu revient à son état de départ.
- ☒ Soit une partie lancée, quand je change le fichier à charger et que je reviens au menu principal, alors l'application présente la carte du nouveau fichier

Phase de conception et problèmes à résoudre

Diagramme de conception générale

- ☒ Les Candidats et les Responsabilités ont été identifiées
- ☒ Les cartes CRC et les liens entre elles ont été faites
- ☒ Le diagramme de séquence/collaboration a été créé
- ☒ Le diagramme de classe a été fait

La classe de fabrique (Factory)

- ☑ Une classe de fabrique (Factory) est créée dans le package `domains` comme les autres classes et sert à synchroniser le jeu sur les différents superviseurs
- ☑ La classe de fabrique est passée aux superviseurs au travers d'une interface
- ☑ Les superviseurs se synchronise avec la classe de fabrique dans les méthodes `onEnter` et `onLeave`
- ☑ La classe de fabrique est insanciée dans `Program` et est passée au constructeur des superviseurs concernés

La représentation de la carte de cases

- ☑ Pas de tableau 2D pour preprésenter l'association des coordonnées aux cases et choisir la collection appropriée pour mémoriser la carte et les cases

Questions supplémentaires d'algo et de POO

Questions algorithmiques

- ☑ Ecrire dans l'entête de la méthode `PlayGameSupervisor.onEnter` les post-conditions les classes `Player`, `CaseMap` et les `Case` de la carte doivent respecter après la création de la partie pour pouvoir commencer à creuser/jouer.
- ☑ Indiquer dans l'entête Javadoc de la classe `CaseMap` l'interface de collection a été utilisée pour représenter la carte et justifier son choix en expliquant les différentes opérations à y effectuer
- ☑ Indiquer dans cette même Javadoc quelle implémentation de la collection a été utilisée pour représenter la carte et justifier son choix en déterminant la CTT des principales opérations utilisées dans l'itération 1
- ☑ **Indiquer la CTT du placement des trésors sur les cases creusables dans la javadoc en entête de `TreasureQuestGameFactory`** (Pour répondre à cette question, examinez la partie de votre code concernée, et identifiez les boucles, les imbrications, mais aussi les collections utilisées et leurs opérations ou les appels de sous-méthodes ou de méthodes d'autres objets. Quand vous répondez, n'oubliez pas d'indiquer à quoi correspondent vos libellés N,M, etc)
- ☑ Les Superviseurs collabore avec la fabrique (Factory) de Game au travers d'une interface

Questions POO

- ☒ Les classes sont dans `treasurequest.domains` et n'ont pas de lien avec les vues (tout ce qui pourrait être en dehors de `domains`)
 - ☒ L'encapsulation (attributs private) et les copies défensives sont bien effectuées sur toutes les classes du domains
 - ☒ Ecrire des méthodes courtes et peu complexes (et tenter de respecter la règle de Demeter)
 - ☒ Ecrire des classes timides (c'est à dire qui font le boulot et qui ne nécessite pas de tout le temps leur demander des choses depuis d'autres classes)
 - ☒ Bon usage du polymorphisme
 - ☒ Bonne utilisation des interfaces
 - ☒ Aucune alerte PMD
 - ☒ Tests unitaires JUnit5 dans le dossier `tests` qui donne un coverage d'au moins 90% sur chaque classe du domains (pour un degré de maitrise supplémentaire → coverage de 100% de ces classes)
-

Revision #23

Created 27 April 2023 17:06:13 by SnowCode

Updated 21 September 2023 17:40:34 by SnowCode