Projet tuteuré de désintégration (Caylus Magna Carta) : liste des tâches

Tâche 1. [/5] Créez un dépôt GIT.
Invitez votre enseignant pour qu'il puisse suivre chacun de vos multiples dépôts.

Tâche 2. [/3] Utilisez un outil d'intégration continue.
Dans ce cas, faites-le dès le début du projet !

Tâche 3. [/5] Reconstruisez, à partir du code, le diagramme de classes (UML).
Vous devez dessiner toutes les classes et les relations (avec leurs multiplicités et navigabilités) entre celles-ci, mais sans donner ni les attributs ni les méthodes ni les rôles ; ainsi, seules les classes et relations sont nommées.

Tâche 4. [/3] Reconstruisez, à partir du code, le sous-diagramme des classes (UML) `Player`, `ColorPlayer` et `MoneyResource`.
Vous devez préciser, pour ces classes et pour leurs relations, tous les attributs (sans visibilité et sans type) et méthodes (sans leurs paramètres et sans le type renvoyé), de classe et d'instance, ainsi que les rôles, multiplicités et navigabilités des relations.
Rappel : un attribut du code peut provenir d'un rôle d'une relation du diagramme de classes.

~~Tâche 5. [/0] Reconstruisez, à partir du code, le diagramme des cas d'utilisation (UML).~~

Tâche 6. [/4] Reconstruisez, à partir du code, le diagramme de séquence (UML) du scénario « Procéder à la phase de collecte des revenus dans le cas de la version standard » (méthode `play_phase_income` de la classe `Game`).
Vous devez ignorer toutes les instructions d'affichage (`print`).

Tâche 7. [/2] Reconstruisez, à partir du code, le diagramme de communication (UML) du scénario « Appliquer l'effet du bâtiment neutre du colporteur pour un travailleur d'un joueur « intelligence articielle » disposant de suffisamment d'argent et/ou de ressources» (méthode `apply_peddler_effect` de la classe `Building`).
Vous devez ignorer toutes les instructions d'affichage (`print`).

Tâche 8. [/3] Divisez le code en modules.
Rappel : un module Python ne correspond pas à une classe Java.

Tâche 9. [/2] Utilisez le patron de conception singleton pour la classe `Money`. Effectuez les tests unitaires correspondants.

Tâche 10. [/1] Quel patron de conception a été singé dans le code ? Indiquez quelle(s) classe(s) et méthode(s) sont concernées.

~~Tâche 11. [/0] Utilisez un autre patron de conception.~~

~~Tâche 12. [/0] Créez un motif d'architecture logicielle modèle-vue-contrôleur.~~

Tâche 13. [/2] Mettez en place les tests unitaires de la fonction `ordinal_number`.
Vous devez effectuer un maximum de tests pertinents (c.-à-d. sans tester tous les entiers !).

Tâche 14. [/3] Mettez en place les tests unitaires de la méthode `resource_all_payments` de la classe `Player`.
Vous devez uniquement effectuer les deux tests décrits dans les commentaires du code.

Tâche 15. [/4] Mettez en place les tests unitaires les plus complets possibles pour la méthode `remove_tokens_castle` de la classe `Game`.

Tâche 16. [/2] Mettez en place un (seul) test unitaire pour la méthode `choose_n_provost_movement` de la classe `HumanPlayer`.
Vous devez simuler la saisie de deux valeurs incorrectes (plus petite que la borne inférieure de l'intervalle et plus grande que la borne supérieure de l'intervalle) puis d'une valeur correcte (comprise dans l'intervalle), et vérifier tous les affichages et la valeur renvoyée.

Tâche 17. [/1] Concevez un seul test fonctionnel non automatisé pour la phase du château permettant de considérer simultanément les trois cas suivants : un même joueur obtient des jetons de points de prestige différents, deux joueurs ont donné le plus grand nombre de lots et un joueur pourrait proposer plus de lots que de jetons de points de prestige disponibles.
Vous présenterez ce test fonctionnel comme celui présenté dans l'exemple de la règle pour la phase du château (et aussi dans la tâche suivante).

~~Tâche 18. [/0] Automatisez le test fonctionnel de l'exemple de la règle pour la phase du château :~~
~~*On dispose de 2 jetons Donjon (à 4 PP), de 8 jetons Murailles (à 3 PP) et de 9 jetons Tours (à 2 PP).*~~
~~*L'ordre de passage (sur le pont) est Orange, Bleu, Vert, Rouge.*~~
~~*Orange donne 1 lot (1 cube de nourriture, 1 cube de bois et 1 cube de pierre) et obtient 1 jeton Donjon pour 4 PP.*~~
~~*Bleu donne 1 lot (1 cube d'or pour remplacer 1 cube de nourriture, 1 cube de bois et 1 cube de pierre) et obtient 1 jeton Donjon pour 4 PP.*~~
~~*Vert donne 2 lots (2 cubes de nourriture, 2 cubes de bois et 2 cubes de pierre) et obtient 2 jetons PP Murailles soient 6 points.*~~
~~*Rouge ne donne aucun lot (et n'obtient donc aucun point de prestige).*~~
~~*Vert a donné le plus de lots et gagne donc un cube d'or du stock.*~~

Erreur ! Référence de lien hypertexte non valide.

Tâche 19. [/1] Utilisez une fonction `filter` pour remplacer le code (de l'instruction `return`) de la méthode de classe `get_wild_resource` de la classe `Resource`.

Tâche 20. [/4] Programmez (simplement) la phase actions de l'IA avancée.
Vous devez uniquement prioriser chaque action : construire un bâtiment de prestige, construire l'un de ses bâtiments, placer un ouvrier sur un bâtiment neutre, placer un ouvrier sur l'un de ses bâtiments, placer un ouvrier sur un bâtiment d'un autre joueur, piocher une carte, remplacer toutes ses cartes, passer.

Tâche 21. [/3] Programmez (simplement) la phase de déplacement du prévôt de l'IA avancée.
Vous devez éloigner le prévôt du château (c.-à-d. le déplacer vers la fin de la route) le plus possible si l'un de vos travailleurs est dans chacun des bâtiments allant du bâtiment juste après l'ancienne position du prévôt jusqu'au bâtiment de la nouvelle position du prévôt ; sinon, laissez le prévôt là où il est.

~~Tâche 22. [/0] Programmez (simplement) la phase des effets des bâtiments de l'IA avancée.~~

~~Tâche 23. [/0] Programmez (simplement) la phase du château de l'IA avancée.~~
~~Vous devez acheter un maximum de lots.~~

Tâche 24. [/3] Programmez la possibilité de jouer une nouvelle partie.
Vous ne devez ni relancer l'application ni relire le fichier XML. Vous devez demander au joueur s'il veut rejouer ou non. Si l'utilisateur veut rejouer, il doit pouvoir choisir la version du jeu (initiation ou standard), l'ordre des joueurs, les couleurs des joueurs, le niveau (basique ou avancée) de chaque intelligence artificielle.

~~Tâche 25. [/0] Lancez des parties entre IA seules.~~
~~Il faudrait pouvoir lancer beaucoup de parties, sans joueur humain, entre 2, 3 ou 4 IA, chacune de niveau basique ou avancée, enregistrer les scores, de sorte à savoir si l'intelligence artificielle avancée dernièrement codée est effectivement meilleure que l'intelligence artificielle basique.~~

~~Tâche 26. [/0] Rendez l'interface utilisateur bilingue (anglais et français).~~

~~Tâche 27. [/0] Explicitez les types.~~
~~C'est a priori déjà fait pour les types primitifs (bool, int, str, etc.) et c'est mis sous forme de commentaires pour les types complexes.~~

~~Tâche 28. [/0] Rendez abstraites certaines classes.~~
~~Par exemple, les classes `MoneyResource`, `Phase`, `Player`, `AIPlayer`, `Building` pourraient être abstraites.~~

~~Tâche 29. [/0] Changez le type de certains attributs.~~
~~Par exemple, les attributs `color_players` et `buildings` de la classe `GameElement` pourraient être des ensembles plutôt que des listes.~~

~~Tâche 30. [/0] Programmez une interface graphique.~~

Tâche 31. [/2] Générez automatiquement la documentation de vos programmes dans des fichiers *HTML*.

Tâche 32. [/2] Dressez une liste de griefs (pertinents et donc argumentés, exceptés ceux pouvant correspondre à certaines des tâches demandées) que vous auriez envers le code.