Rapport intermédiaire

Burel Paul
Choisy Sebastien
Khaldi Djade
Kunwar Aryamaan
Maïssa Axel

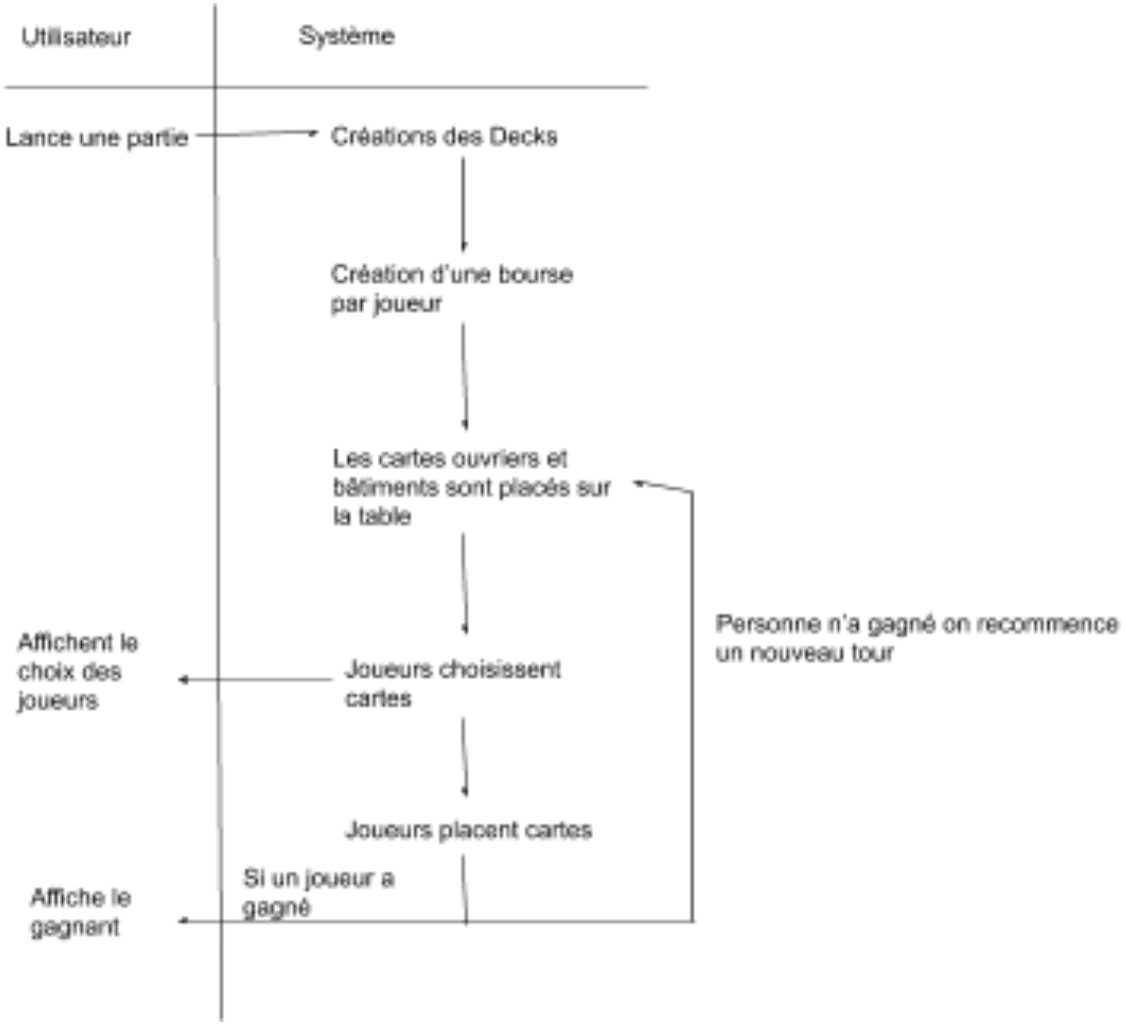
I-Point de vue général de l'architecture et des fonctionnalités

Glossaire :

On définit ici les méthodes dont le nom n'est pas explicite ou si la valeur de retour n'est pas triviale.

1. **Package asset:**
    1. Méthode getOuvriersDisponibles: renvoie une ArrayList des ouvriers disponibles.
    2. Méthode getOuvriersOccupés: renvoie une ArrayList des ouvriers occupés.
    3. Méthode getChantiers: renvoie une ArrayList des chantiers que le joueur possède.
2. **Package carteBatiments:**
    1. Méthode IsBuilt: celle-ci va comparer les ressources nécessaires pour la construction du bâtiment avec les ressources apportées par les ouvriers qui travaillent sur le bâtiment.
    2. Méthode sumRessources: renvoie la somme des ressources de tous les ouvriers présents sur le chantier
    3. Méthode carteSurTable: renvoie la pioche des 5 cartes.
    4. Méthode obtenirDeckJoueur: renvoie toutes les cartes que le joueur possède, celles dans sa main, sur le plateau et celles qui sont finies.
3. **Package carteOuvriers:**
    1. Méthode getApprenti: assigne aléatoirement un apprenti à un joueur.
4. **Package display:**
    1. displayOuvrierDuJoueur: affiche dans la console les ouvriers que le joueur possède.
    2. displayChantierDuJoueur: affiche dans la console les chantiers que le joueur possède.
    3. displayEtatChantiersDuJoueur: affiche l'état des chantiers du joueur.
5. **Package ia:**
    1. iaChoisitOuvrier: le joueur choisit une ou plusieurs cartes ouvriers présentes sur le plateau.
    2. iaChoisitChantier: le joueur choisit une ou plusieurs cartes chantiers présentes sur le plateau.
    3. iaAttributOuvrierAChantier: le joueur prend un ouvrier de son deck et le place sur un chantier qu'elle possède.
6. **Package moteurDeJeu:**
    1. Méthode déroulementJeux: cette méthode permet le déroulement de la partie.

Diagramme d'activité :



Fonctionnalités traitées :

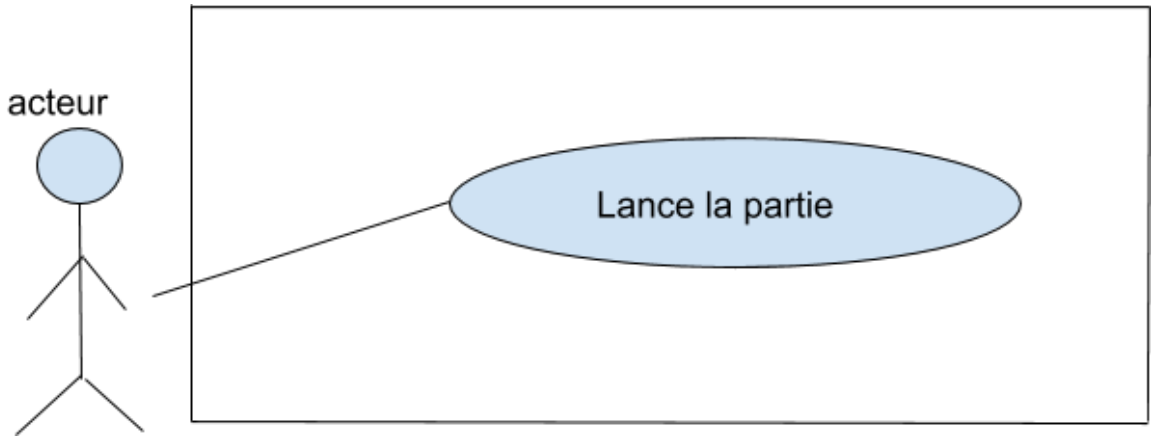
Il est possible de créer un deck ouvrier et un deck bâtiment. Les joueurs possèdent une bourse d'argent qui leur sert pour acquérir des bâtiments ou des ouvriers. Un joueur a la capacité de placer des ouvriers sur un bâtiment. Un affichage pour le client, afin de connaître le déroulement du jeu.

II-Modélisation de l'application

Analyse des besoins :

A ce stade du projet, le seul acteur est le commanditaire. Il n'y a qu'un seul scénario, celui où l'acteur lance une partie.

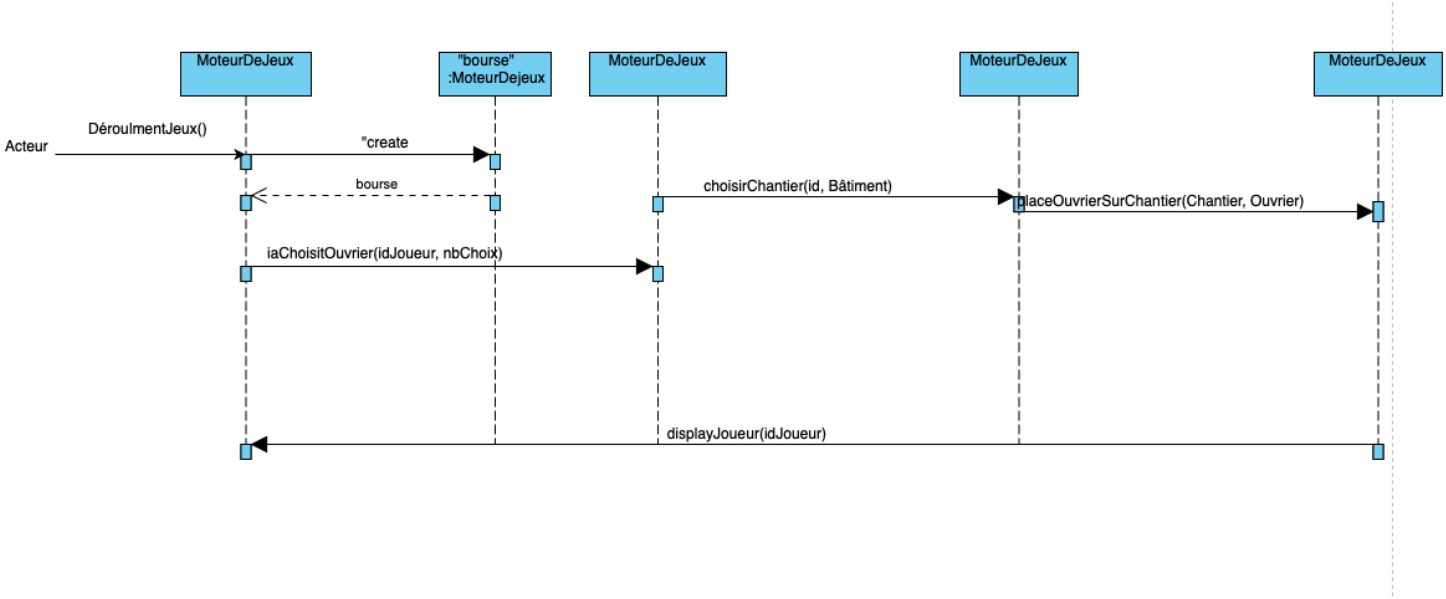
Use-case :



Conception logicielle :

Le diagramme de séquence est à retrouver en dernière page du rapport

Diagramme de séquence :



III-Evolution du projet

Evolution prévue :

Pour les évolutions demandés, nous ferons en premier lieu la création du modèle client-serveur et les modifications de couleur, puis la possibilité de lancer une partie à l'époque de l'antiquité quand tout le reste sera fonctionnel.

- Evolution A Possibilité de lancer une partie des bâtisseurs moyen-âge ou des bâtisseurs antiquité :

Possibilité de choisir au début d'une partie si on veut jouer avec les bâtiments/ouvriers du moyen-âge ou de l'antiquité, il faudrait donc adapter le code selon le jeu demandé, il y aura une partie commune aux deux jeux (la plupart des règles sont similaires), mais certains éléments diffèrent (un fichier Json par jeu par exemple, avec les cartes différentes), il y a l'action « réaliser un investissement » dans la version de l'antiquité, que nous n'avons pas au moyen-âge, les matériaux sont aussi différents.

- Evolution B Utilisez des couleurs pour le texte sur un fond blanc :

En relation avec l'évolution demandée plus haut, il y aura donc une couleur de texte différente, selon l'époque du jeu sélectionnée.

- Evolution C Le moteur est une application, chaque joueur est une application, les joueurs et le moteur communiquent sur le modèle client-serveur :

Tout le code qui définit le moteur de jeu devra donc être regroupé sur un même projet java qui définira le serveur, alors que tout ce qui concerne le joueur (IA, etc..) devra être regroupé sur un autre projet java et définira le client. Un premier client « commanditaire » pourrait envoyer une première requête au serveur, pour commencer l'échange, puis le serveur pourrait demander le nombre de joueur, le nombre de parties et le jeu demandé (Antiquité/Moyen-âge) puis attendre d'avoir communiqué avec un nombre de clients « joueur » identique au nombre de joueurs spécifié plus tôt par le premier commanditaire. Le serveur enverra ensuite des questions aux joueurs (quelles actions le client souhaite effectuer) et pourra aussi poser des questions grâce à des commandes définies au début de la partie par le serveur (voir les cartes en sa possession, éventuellement le nombre d'ouvriers disponibles, son score, etc. …).

Le serveur devra donc vérifier chaque réponse du client, pour être sûr qu'il n'y a pas de triche.

Organisation des 4 dernières itérations :

ITERATION 6 :

- Gestion des tours : fin d'un tour après avoir utilisé toutes les actions
- Création de l'action "Prendre de l'argent"/Passer

- Création du compteur "Point de Victoire"
- Fin de partie : atteindre un certain nombre point
- Mise en place du serveur
- Mise en place du client
- Création du Deck Antiquité

ITERATION 7 :

- Modification de la fin d'un tour : tout le monde a utilisé ses actions. Ordre : d'abord le joueur 1 utilise toutes ses actions, puis le joueur 2
- Améliorer l'IA (stratégie simple)
- Désignation du premier joueur avec le totem
- Gestion des machines, batiments => ouvriers
- Fin de partie : on fait 2-3 tour, celui qui a le plus de point gagne (égalité non gérée)
- Mise en place de plusieurs clients
- Lancer 500 parties
- nouveau pom.xml
- Proposer le choix du deck antiquité ou moyen-age

ITERATION 8 :

- Vraie condition de fin de partie + les conditions supplémentaires en cas d'égalité
- Création du classement en fonction des conditions de fin de partie
- Mise en place de 4 joueurs
- Proposer de rejoueur quand la partie est terminée

ITERATION 9 :

- Amélioration de l'IA

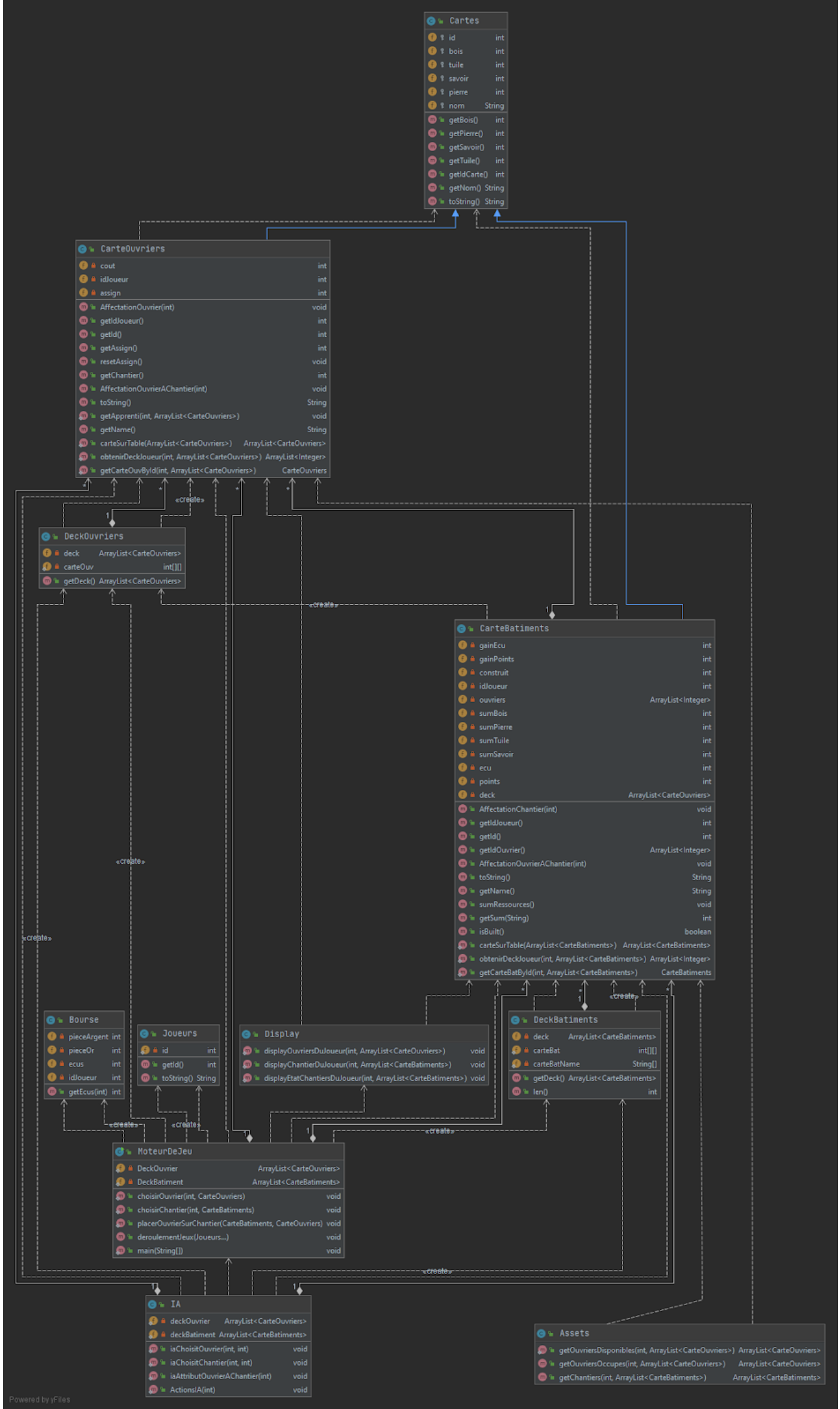
IV-Conclusion

Points forts :

- Notre java doc est claire pour le client

Points faibles :

- Notre code possède trop de packages ce qui entraine parfois une confusion dans l'utilisation des import.
- Certaines de nos classes se sont révélées inutiles.
- Lors de nos test nous ne testions pas toutes les conditions d'une méthode (boucle if).
- La conception du projet mérite d'être améliorer.


Cartes
id int
bois int
tuile int
savoir int
pierre int
nom String
getBois() int
getPierre() int
getSavoir() int
getTuile() int
getIdCarte() int
getNom() String
toString() String
CarteOuvriers
cout int
idJoueur int
assign int
AffectationOuvrier(int) void
getIdJoueur() int
getId() int
getAssign() int
resetAssign() void
getChantier() int
AffectationOuvrierAChantier(int) void
toString() String
getApprenti(int, ArrayList<CarteOuvriers>) void
getName() String
carteSurTable(ArrayList<CarteOuvriers>) ArrayList<CarteOuvriers>
obtenirDeckJoueur(int, ArrayList<CarteOuvriers>) ArrayList<Integer>
getCarteOuvById(int, ArrayList<CarteOuvriers>) CarteOuvriers
«create»
DeckOuvriers
deck ArrayList<CarteOuvriers>
carteOuv int[][]
getDeck() ArrayList<CarteOuvriers>
CarteBatiments
gainEcu int
gainPoints int
construit int
idJoueur int
ouvriers ArrayList<Integer>
sumBois int
sumPierre int
sumTuile int
sumSavoir int
ecu int
points int
deck ArrayList<CarteOuvriers>
AffectationChantier(int) void
getIdJoueur() int
getId() int
getIdOuvrier() ArrayList<Integer>
AffectationOuvrierAChantier(int) void
toString() String
getName() String
sumRessources() void
getSum(String) int
isBuilt() boolean
carteSurTable(ArrayList<CarteBatiments>) ArrayList<CarteBatiments>
obtenirDeckJoueur(int, ArrayList<CarteBatiments>) ArrayList<Integer>
getCarteBatById(int, ArrayList<CarteBatiments>) CarteBatiments
Bourse
pieceArgent int
pieceOr int
ecus int
idJoueur int
getEcus(int) int
Joueurs
id int
getId() int
toString() String
Display
displayOuvriersDuJoueur(int, ArrayList<CarteOuvriers>) void
displayChantierDuJoueur(int, ArrayList<CarteBatiments>) void
displayEtatChantiersDuJoueur(int, ArrayList<CarteBatiments>) void
DeckBatiments
deck ArrayList<CarteBatiments>
carteBat int[][]
carteBatName String[]
getDeck() ArrayList<CarteBatiments>
len() int
MoteurDeJeu
DeckOuvrier ArrayList<CarteOuvriers>
DeckBatiment ArrayList<CarteBatiments>
choisirOuvrier(int, CarteOuvriers) void
choisirChantier(int, CarteBatiments) void
placerOuvrierSurChantier(CarteBatiments, CarteOuvriers) void
deroulementJeux(Joueurs...) void
main(String[]) void
IA
deckOuvrier ArrayList<CarteOuvriers>
deckBatiment ArrayList<CarteBatiments>
iaChoisitOuvrier(int, int) void
iaChoisitChantier(int, int) void
iaAttributOuvrierAChantier(int) void
ActionsIA(int) void
Assets
getOuvriersDisponibles(int, ArrayList<CarteOuvriers>) ArrayList<CarteOuvriers>
getOuvriersOccupes(int, ArrayList<CarteOuvriers>) ArrayList<CarteOuvriers>
getChantiers(int, ArrayList<CarteBatiments>) ArrayList<CarteBatiments>
Powered by yFiles