PROJET TUTEURÉ

Rapport intermédiaire

Novembre 2020

Ecrit par: Saad El Din AHMED, Yessine BEN EL BEY, Salim DIB, Youssef HACHIMI, Hugo NORTIER

Tuteurs projet : Audrey OCELLO, Philippe RENEVIER GONIN

Sommaire

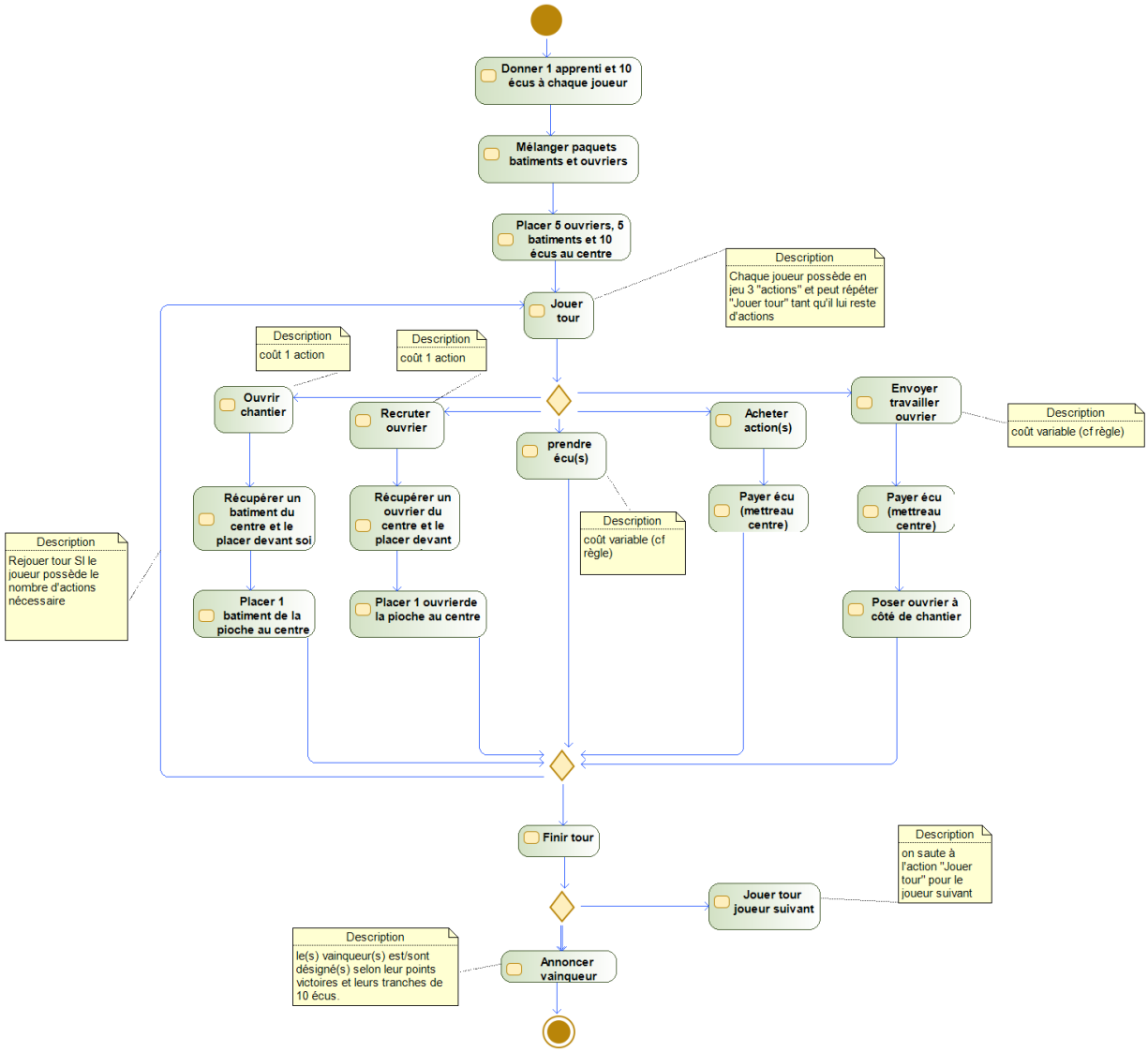
1) Point de vue général de l'architecture et des fonctionnalités

a. Glossaire

Ecus : monnaie de jeu

Points de victoire : points obtenus après avoir fini un chantier et issue du décompte final déterminant le gagnant

b. Représentation générale



c. Fonctionnalités traitées

A l'état actuel de la solution, il est possible de lancer une partie unique avec 4 joueurs tous au même niveau de difficulté IA.

Les fonctionnalités prises en compte dans ce jeu sont :

- Démarrage avec la création d'une pioche de cartes ouvriers et bâtiment avec des ressources aléatoires (pas celles des vraies cartes de jeu).
- Une fois les pioches constituées, réception d'une carte apprenti de la part de chaque joueur recevra aléatoirement une carte apprenti.
- Possibilité pour chaque joueur d'effectuer une action issue de l'IA par rapport à l'état actuel de la partie. Toutes les actions du jeu sont disponibles : ouvrir un chantier, recruter un ouvrier, envoyer un ouvrier travailler, prendre des écus, acheter une action.
- Les cartes machines sont prises en compte, si on termine une machine on recevra l'ouvrier correspondant.
- A la fin de chaque chantier, chaque joueur reçoit les points victoire et écus associés.
- Affichage textuel de toutes les fonctionnalités citées.

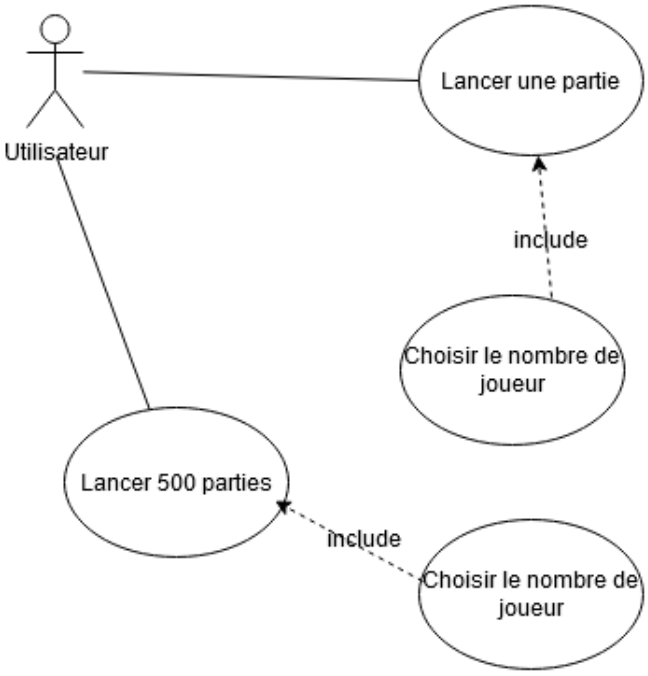
On n'a pas pris en compte une fin de partie réelle pour l'instant, la partie finie arbitrairement à l'issue de 10 tours.

2) Modélisation de l'application

a. Analyse des besoins (exigences) : Cas d'utilisation

Diagrammes de Cas d'utilisation

Nous avons qu'un seul acteur dans notre application, celui-ci est celui qui lance l'application, le commanditaire.



Scénarios

Il peut uniquement lancer 2 types de scenarios pour l'instant : lancer une partie unique ou lancer 500 parties. Il est possible de mettre en argument le nombre de joueurs s'il ne veut pas qu'ils soient 4.

Scenario : lancement de partie unique

1. L'utilisateur lance le logiciel sans spécifier le nombre de joueurs

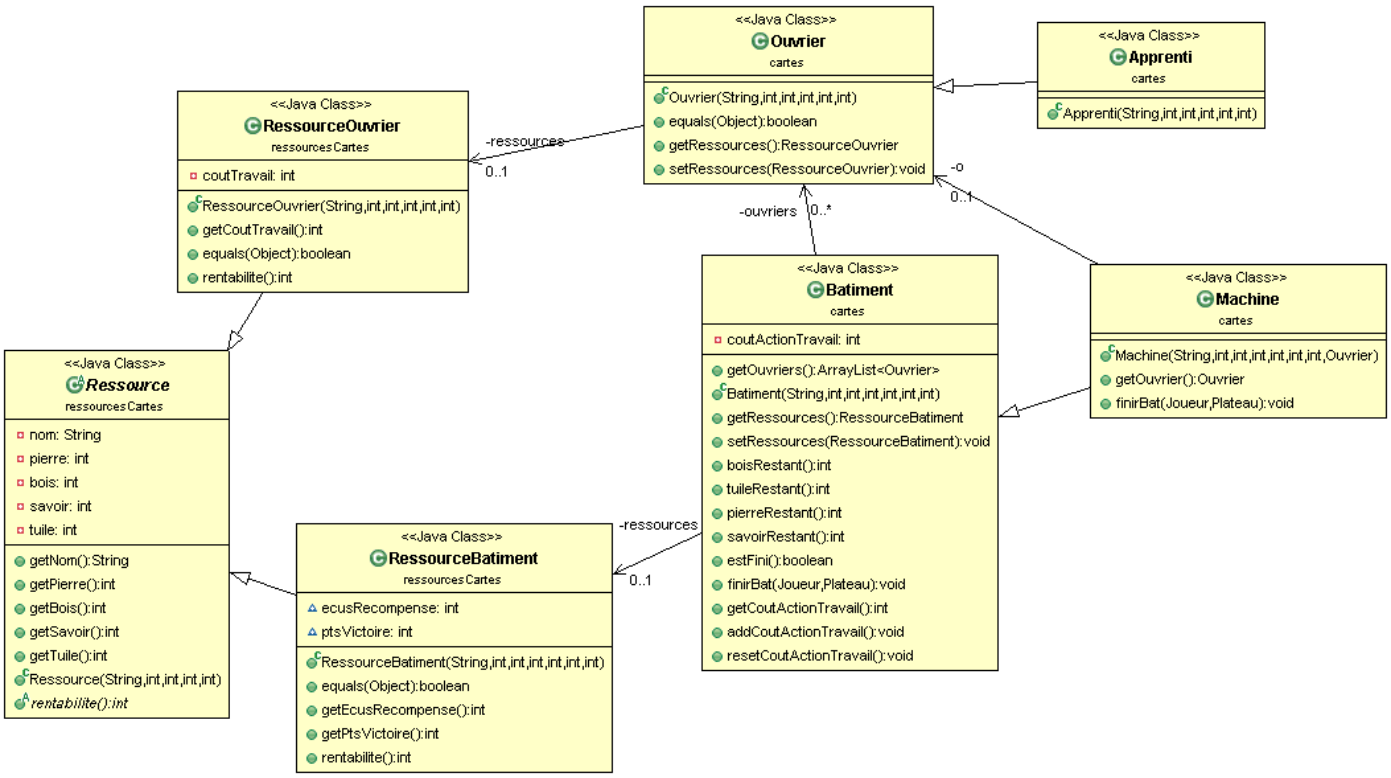
2. Le système démarre le jeu avec 4 joueurs
3. Le système affiche textuellement le déroulement du jeu à l'utilisateur
4. Le système affiche les statistiques du gagnant à la fin du jeu

Scenario : lancement de 500 parties

1. L'utilisateur lance le logiciel sans spécifier le nombre de joueurs
2. Le système démarre 500 jeux avec 4 joueurs
3. Le système affiche les statistiques et pourcentages de victoire pour chaque joueur

Dans les 2 scénarios, il y a la possibilité pour l'utilisateur de spécifier le nombre de joueurs souhaités, si rien n'est saisi, le système fera des jeux à 4 joueurs par défaut.

b. Conception logicielle

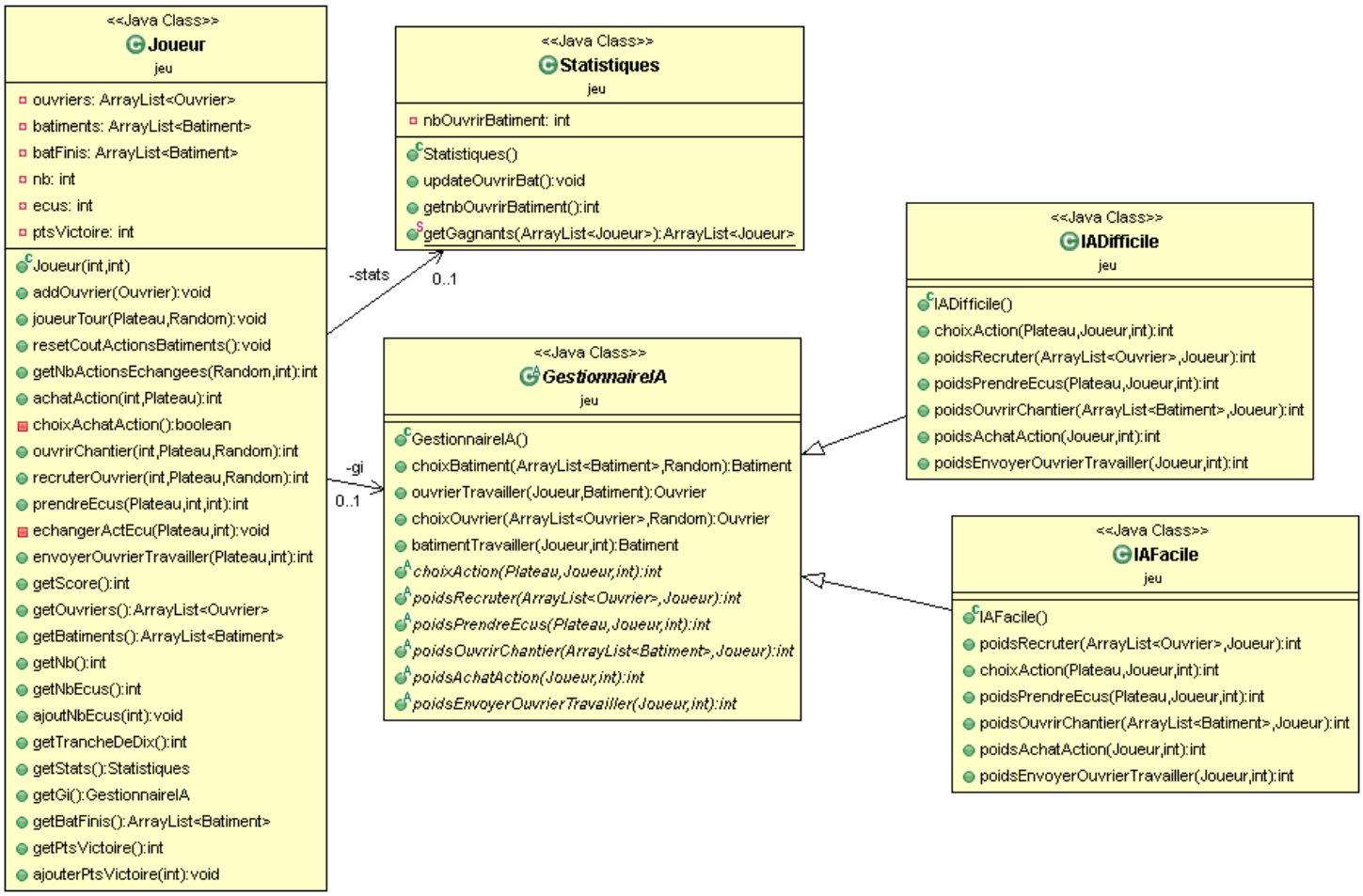


Nous avons créé une classe pour chaque type de carte, les cartes qui sont des extensions d'autres cartes ont été modélisées de la sorte. Ainsi, un Apprenti qui est un ouvrier avec comme différence qu'il est distribué aléatoirement à chaque joueur au début du jeu sera un extend d'Ouvrier. Une Machine sera un extend d'un Batiment car on le traite comme tel jusqu'à son complètement, à l'issue de son chantier elle deviendra un Ouvrier qui appartiendra au joueur qui l'a complété. Conceptuellement, la Machine sera un Batiment qui possède un ouvrier renvoyé à la fin du chantier.

Chaque Batiment possède une liste des ouvriers qui travaillent sur le chantier actuellement, cette liste sera vidée lorsque le chantier est complété.

Chacune des cartes possède des ressources que nous avons modélisé par des classes, ceci pour éviter la redondance car certaines ressources sont partagées, modélisé ici par le fait que les RessourceOuvrier et RessourceBatiment extend les 2 de Ressource qui est abstraite.

C'est à partir de ces classes Ressource que l'on pourra calculer la rentabilité des cartes pour ce qui concerne les choix de l'IA.



La classe Joueur correspond à chaque joueur d'une partie, c'est une classe assez conséquente qui possède une méthode pour chaque action qu'elle peut effectuer pendant son tour. Elle possède aussi une liste de ses Batiment ouverts, Ouvrier et Batiment complétés donc pour éviter de l'alourdir nous avons détaché tout ce qui est choix d'IA et action de joueur dans la partie.

Nous avons donc un GestionnaireIA qui possède des méthodes ayant comme but d'analyser l'état actuel du plateau et faire le choix de la meilleure action à effectuer pendant le tour. Cette action sera renvoyée au joueur su forme de int qui sera traduite en une action concrète exécutée par les méthodes de Joueur.

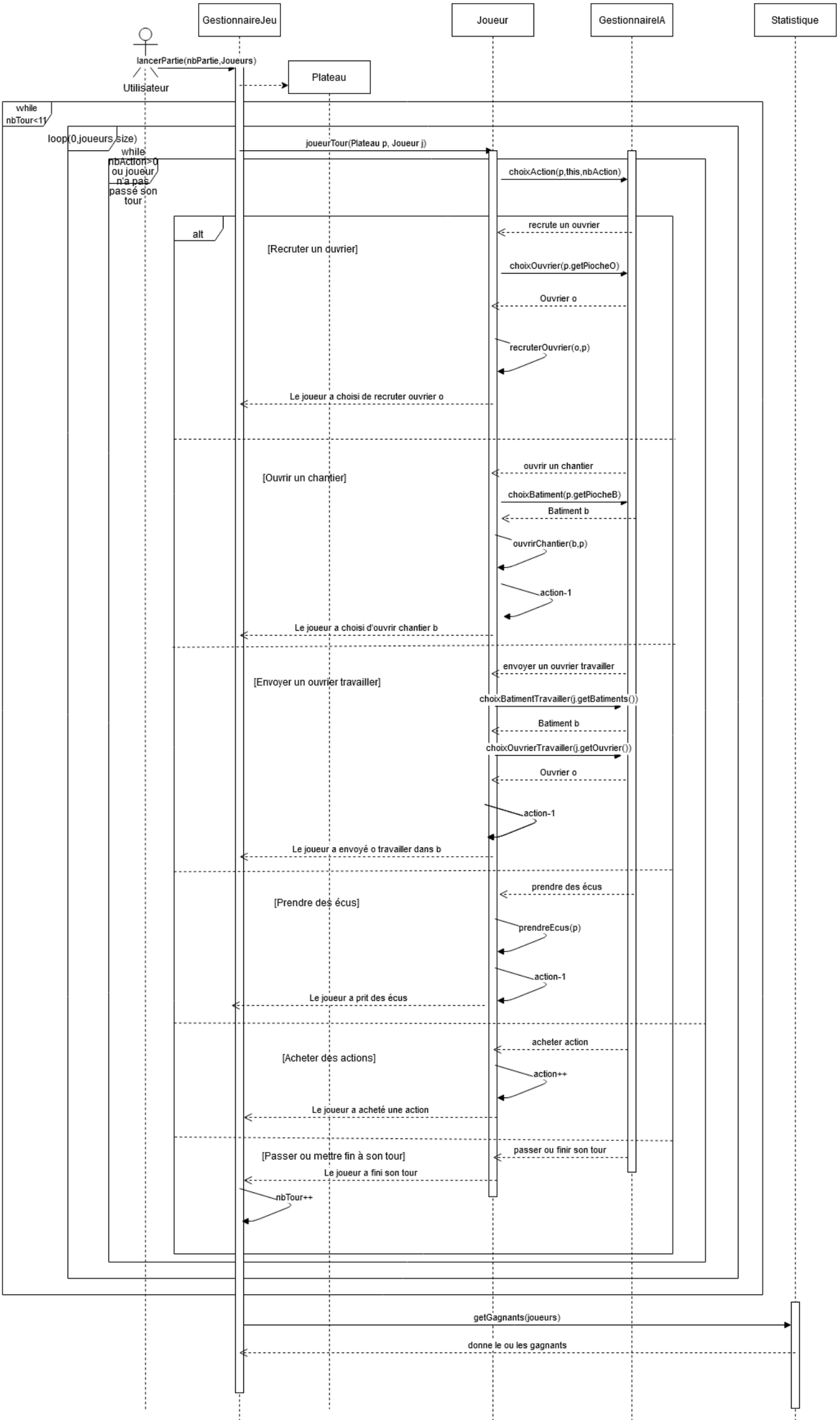
On peut noter aussi que gestionnaireIA est abstraite car on l'initialise uniquement avec IADifficile ou IAFacile qui correspondent aux niveaux de difficulté disponibles. Cependant, actuellement il y a que IADifficile qui est vraiment fonctionnelle, IAFacile est uniquement à ses stades initiaux de développement.

Chaque Joueur possède aussi une classe Statistiques, cette classe a pour rôle de « log » toutes les actions du Joueur en question pour pouvoir les afficher à la fin de la partie. Elle a aussi une méthode statique (car on n'a pas besoin d'objet spécifique pour l'utiliser) qui reprend toutes les statistiques de tous les joueurs en arguments pour pouvoir par la suite annoncer un gagnant. Il faut noter que cette classe est incomplète, on n'a pas un log de toutes les actions encore.

Nous avons modélisé le plateau de Jeu par la classe Plateau, cette classe contiendra les écus, les paquets de cartes respectives de Batiment et Ouvrier qui seront manipulées par les joueurs. Elle aura aussi comme tâche d'initialiser toutes les cartes en question.

La classe correspondant à l'Affichage textuel est une classe statistique que l'on appellera à chaque fois que l'on veut afficher textuellement quelque chose, cela nous permet de séparer tout ce qui est sortie textuelle et affichage du moteur de jeu.

La classe GestionnaireJeu est la classe abstraite possédant le main, elle analysera les arguments de l'utilisateur et lancera une partie avec lancerPartie qui correspondra aux arguments sélectionnés. Elle aura aussi une méthode correspondant au verbose du jeu qui sera un service que l'on pourra appeler dans n'importe quelle classe pour vérifier s'il faut faire une sortie textuelle ou pas (donc si appeler Affichage ou pas). Pour l'instant, elle n'a pas besoin d'être non statique, donc instanciable car nous n'avons pas encore pris en compte les arguments de l'utilisateur qui pourraient induire à une différenciation entre les parties lancées (on lance toujours la même partie entre 4 joueurs).


GestionnaireJeu
Joueur
GestionnaireIA
Statistique
lancerPartie(nbPartie,Joueurs)
Utilisateur
Plateau
while
nbTour<11
loop(0,joueurs.size)
joueurTour(Plateau p, Joueur j)
while
nbAction>0
ou joueur
n'a pas
passé son
tour
choixAction(p,this,nbAction)
alt
recrute un ouvrier
[Recruter un ouvrier]
choixOuvrier(p.getPiocheO)
Ouvrier o
recruterOuvrier(o,p)
Le joueur a choisi de recruter ouvrier o
ouvrir un chantier
[Ouvrir un chantier]
choixBatiment(p.getPiocheB)
Batiment b
ouvrirChantier(b,p)
action-1
Le joueur a choisi d'ouvrir chantier b
envoyer un ouvrier travailler
[Envoyer un ouvrier travailler]
choixBatimentTravailler(j.getBatiments())
Batiment b
choixOuvrierTravailler(j.getOuvrier())
Ouvrier o
action-1
Le joueur a envoyé o travailler dans b
prendre des écus
[Prendre des écus]
prendreEcus(p)
action-1
Le joueur a prit des écus
acheter action
[Acheter des actions]
action++
Le joueur a acheté une action
passer ou finir son tour
[Passer ou mettre fin à son tour]
Le joueur a fini son tour
nbTour++
getGagnants(joueurs)
donne le ou les gagnants

Ceci est le diagramme de séquence du déroulement d'une partie type :

En règle générale :

1. Une partie est lancée par le gestionnaire de jeu.
2. Un plateau est initialisé avec des pioches de cartes générées aléatoirement.
3. Le plateau distribue des écus et un apprenti à chaque joueur.
4. Pendant 10 tours, chaque joueur appellera sa méthode jouerTour tant qu'il ait assez d'actions restantes (on commence avec 3 actions) qu'à partir de l'état courant du plateau appellera à son tour choixAction du gestionnaireIA qui fera le choix de l'action.
5. Une fois le choix reçu, le joueur fera la manipulation correspondant à l'action en question.
6. Une fois le tour terminé du joueur, on passe à celui suivant.
7. Ainsi, jusqu'à que les 10 tours s'écoulent.
8. Une fois ces 10 tours écoulés, on appellera la méthode statique getGagnants de Statistiques qui annoncera le gagnant de la partie.

3) Prise en compte de l'évolution du sujet

a. Évolution prévue

En ce qui concerne le sujet, nous avons 3 évolutions à prendre en compte :

1) **Affichage textuel en couler** : pour cela il suffira uniquement de changer les méthodes dans la classe Affichage pour qu'elles affichent du texte coloré.
2) **Prendre en compte les bâtisseurs de l'antiquité** : pour cela il faudra changer l'architecture pour avoir une division qui prend en compte les différences avec le jeu déjà implémenté.
    a. On crée des classes correspondantes aux nouvelles cartes : esclave, université, outils, emprunt et aussi les classes Ressources associées (si nécessaire).
    b. On va factoriser les classes qui possèdent certaines méthodes qui vont changer par rapport au jeu que l'on joue, ces classes vont hériter d'une classe abstraite qui possèdera ce qu'elles ont en commun.
        i. **Affichage** : 2 affichages distinctes correspondant chacune à un jeu
        ii. **Plateau** : On aura 2 plateaux possédant chacune des piles de cartes différentes sur la table, notamment celui de l'antiquité ayant 6 piles comparées à 2 de moyen-âge.
        iii. **IA** : on distinguera 2 IA différentes entre les jeux car le jeu de l'antiquité possède une action supplémentaire (très conséquente) comparée à celui du moyen-âge
        iv. **Statistiques** : il y aura 2 statistiques distinctes pour chaque jeu, de plus le décompte final sera différé aussi.
    c. Il faut aussi donner la possibilité à l'utilisateur de l'application de choisir en ligne de commande quel jeu lancer entre les jeux disponibles.
3) **Découpage en client-serveur** : pour cette évolution, il faudra découper la solution actuelle en joueur client et partie serveur. Niveau conception cela implique plusieurs changements.

a. La classe GestionnaireJeu ne va plus contenir le main avec le déroulement d'une partie. Celle-ci va se remplacer par une classe Client et un autre Serveur possédant 2 main distincts.
    i. Le main du serveur prendra en argument le type de jeu, le nb de joueurs et le nb de parties.
    Une fois lancé, il va attendre que des joueurs (les clients) se connectent à la partie. Ensuite, la partie cela lancée. A chaque tour, il va demander à chaque joueur son action en lui envoyant l'état actuel du plateau et il passera au joueur suivant une fois cette réponse reçue et traitée. Une fois le jeu terminé et le gagnant annoncé, le serveur envoyer un message de fin de jeu à chaque joueur pour leur demander de s'arrêter.
    ii. Le main du client prendra en argument le type de jeu auquel il va jouer et son niveau de difficulté, ensuite il essayera de se connecter à une partie en atteinte. Une fois reçu le signal de partie commencé par le serveur, il va attendre qu'il soit son tour pour jouer son action. Il continuera ainsi jusqu'à que le serveur lui envoie un signal lui annonçant la fin de partie.
b. Tout ce qui fait partie de ce qui est moteur et logique de jeu fera partie d'une package « commun » entre le client et le serveur permettant ainsi la sérialisation et possibilité d'envoyer en sockets les objets correspondants pour pouvoir échanger sur des informations à jour concernant l'état actuel du jeu. Si possibilité, on mettra des classes de logique uniquement dans le serveur ou dans le client si celles-ci ne sont utilisées que par l'une d'entre elles.

b. Organisation des 4 dernières itérations

Les itérations suivantes vont être organisées d'une façon telle qu'on avance sur ce qui reste du jeu actuel à finir et sur les évolutions.

Il reste plusieurs fonctionnalités actuelles à mettre en place (+ les évolutions déjà citées précédemment) :

- Prise en compte des ressources réelle des cartes de jeu (on parle ici d'implémentation d'une BD), pour l'instant on a des cartes générées aléatoirement.
- Prise en compte d'une fin de jeu réelle à partir des points victoire de chaque joueur avec décompte final, pour l'instant le jeu se termine après un nombre fixe de tours.
- Affichage de statistiques finales de jeu plus détaillée, pour l'instant on affiche que les points de victoire des gagnants et les tranches de 10 écus mais il faudrait plus de statistiques.
- Possibilité de commencer plusieurs parties (500) avec affichage de statistiques issues de toutes les parties, possibilité aussi de commencer la partie avec un nombre décidé par l'utilisateur de joueurs. Pour l'instant, on peut que lancer une seule partie avec 4 joueurs.
- Mettre à jour l'IA facile. Pour l'instant l'IA facile n'est pas fonctionnelle, on utilise que l'IA difficile à l'initialisation des parties.

Donc, les dernières itérations vont être organisées ainsi :

- **Itération 6** : Mise en place du décomptage final du jeu moyen-âge, prise en compte des retours des itérations précédentes, première mise en place d'un jeu bâtisseurs du moyen-âge sans

l'action « investissement », modifications des affichages textuels du jeu moyen-âge pour avoir des couleurs, implémentation des cartes et ressources réelles de jeu.

- **Itération 7** : Possibilité d'effectuer l'action « investissement » et mise à jour de l'IA pour la prendre en compte, mise en place d'un décompte final pour antiquité. Finition des affichages en couleur pour le jeu de l'antiquité. Mise en place d'un serveur et début de transition de la solution actuelle vers une solution client/serveur.
- **Itération 8** : Compléter l'IA facile partiellement implémentée, mise en place de statistiques de fin de partie (nombre de chantiers ouverts, nombre d'ouvriers recrutés etc.) pour les 2 jeux, possibilité de lancer le jeu en mode 500 parties et création de l'affichage associé. Création de la partie client de la solution client/serveur, mise en place d'une communication partielle entre le client et le serveur.
- **Itération 9** : Finition et amélioration de la solution client/serveur. Optimisation de la solution finale.

4) Conclusion

Nous avons une solution actuelle presque complète (sans compter les évolutions) qui nous permet d'implémenter le client serveur en théorie sans trop de refactoring à faire grâce à la séparation du moteur de jeu et le controlleur (GestionnaireJeu). Cependant, en ce qui concerne l'évolution qui consiste à implémenter les Bâtisseurs de l'Antiquité on constate qu'il sera nécessaire beaucoup de refactoring (on va presque dupliquer la majorité des classes) ce qui nous fait poser la question niveau conception si on pourrait faire mieux.