



Les bâtisseurs du moyen-âge

*15 Novembre
Découverte Métier*

*Mathieu RETHERS, Yannick ALCARAZ,
Nicolas CORBIERE, Alexandre LEMOINE
& Lisa DESLANDES
Mr RENEVIER GONIN*

Table des matières

Table des matières	2
Introduction.....	3
Point de vue général.....	3
Diagramme d'activité.....	3
Introduction	3
Application dans le sujet	3
Méthodes fonctionnelles	4
Modélisation de l'application	6
Analyse des besoins	6
Introduction	6
Diagrammes de Cas d'utilisation.....	6
Scénarios (UseCase).....	6
Conclusion	8
Glossaire	9

Introduction

Ce rapport provient d'un projet réalisé dans le cadre de notre cursus Licence 3 Parcours MIAGE. Le sujet de ce projet est de réaliser une version informatique du jeu de cartes et de stratégie : Les bâtisseurs du Moyen-Âge. Le but du jeu est simple : nous avons des cartes ouvriers et bâtiments. Notre but est de finir un maximum de carte bâtiment jusqu'à 17 points de victoires.

Ce projet, fait en groupe, nous permet ainsi de développer nos compétences en équipes mais aussi en professionnel. Nous épanouissant ainsi dans beaucoup de domaine tel que la gestion du git, de l'organisation d'un groupe et de la mise en place de normes interne tel que celle de nommages par exemple.

Point de vue général

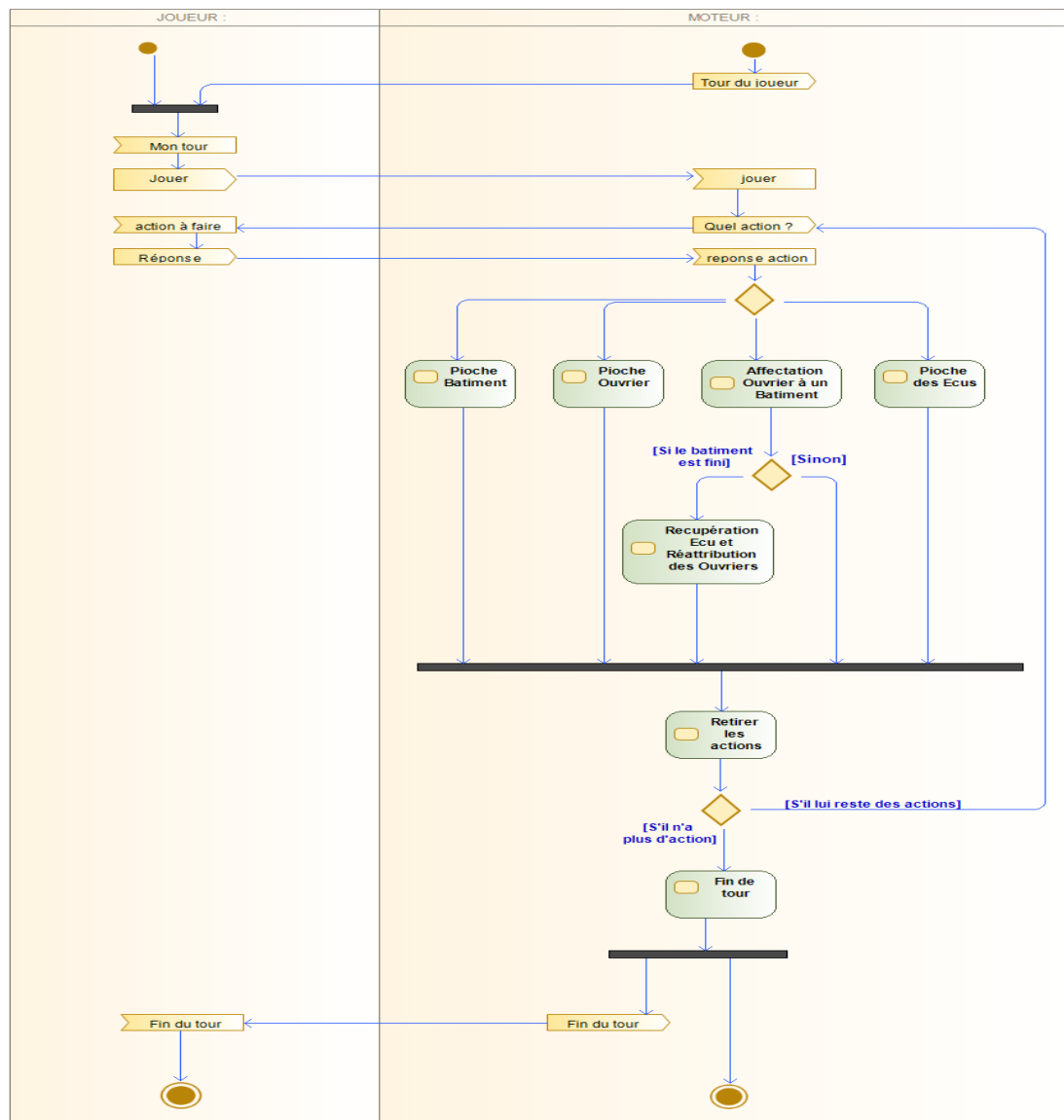
Diagramme d'activité

Introduction

Un diagramme d'activité permet de modéliser un processus interactif, global ou partiel pour un logiciel ou un programme. Il est recommandable pour exprimer une dimension temporelle sur une partie du modèle, à partir de diagrammes de classes ou de cas d'utilisation, par exemple mais cela peut être aussi à partir d'un texte.

Le diagramme d'activité est une représentation proche de l'organigramme ; la description d'un cas d'utilisation par un diagramme d'activité correspond à sa traduction algorithmique. Une activité est l'exécution d'une partie du cas d'utilisation, elle est représentée par un rectangle aux bords arrondis.

Application dans le sujet



Où en sommes-nous ?

Pour le projet initial, on a traité quasiment la totalité du sujet demandé hormis quelques détails et bugs qui nous reste encore à améliorer pour une meilleur utilisation et respect du jeu. Pour les tests, nous poursuivons notre effort d'en réaliser un maximum.

Méthodes fonctionnelles

Package carte

Ce package regroupe tout le code qui est en lien direct avec les cartes ou leur fabrication.

Pile de Carte

Nom de la méthode	Ce qu'elle fait	Attributs	Retour
prendrePremiereCarte	Elle prend la première carte de la liste des cartes visibles et la supprime de la liste	/	Carte
longueurArray	Elle donne la longueur du tableau	/	Integer

Package opération

Le package opération regroupe tout le code en lien directe avec les actions (ou opération) qu'un joueur peut demander d'effectuer.

Opération

Nom de la méthode	Ce qu'elle fait	Attributs	Retour
utiliseAction	Elle retire une ou plusieurs actions au joueur	Integer	/

OpAchatNouvelleAction

Nom de la méthode	Ce qu'elle fait	Attributs	Retour
achatNouvelleAction	Elle ajoute une action au joueur en lui retirant les écus correspondant	/	/

OpAffectationBatiment

Nom de la méthode	Ce qu'elle fait	Attributs	Retour
affectationBatiment	Elle permet de piocher une carte Bâtiment et de l'ajouter à sa main	/	/

OpAffectationOuvrier

Nom de la méthode	Ce qu'elle fait	Attributs	Retour
affectationOuvrier	Elle permet de piocher une carte Ouvrier et de l'ajouter à sa main	/	/

OpAffectationOuvrierBatiment

Nom de la méthode	Ce qu'elle fait	Attributs	Retour
affectationOuvrierBatiment	Elle permet d'affecter un Ouvrier au Bâtiment	/	/
batimentEstFini	Elle vérifie si le bâtiment est fini	/	Boolean
finDeChantier	Elle permet de remettre les ouvriers dans la main Ouvrier et de récupérer les Ecus et les points de Victoire	/	/

OpObtenirEcu

Nom de la méthode	Ce qu'elle fait	Attributs	Retour
obtenirEcu	Elle permet de prendre des Ecus en échanges d'actions	/	/

Package joueur

Le package joueur regroupe tout le code en lien directe avec les joueurs (ou bots actuellement) avec leur différent niveau d'intelligence

BotBasique

Nom de la méthode	Ce qu'elle fait	Attributs	Retour
jouer	Elle permet au bot de jouer	Liste de Carte Bâtiment Liste de Carte Ouvrier	Opération

BotIntelligent

Nom de la méthode	Ce qu'elle fait	Attributs	Retour
estMeilleurPointVictoireMain	Elle permet de trouver la meilleure carte bâtiment dans notre main	/	CarteBatiment

CartesMainBatimentFini	Elle permet de voir si tous nos Bâtiments sont fini ou pas	/	Boolean
estMeilleurPointVictoirePresente	Elle permet de trouver la meilleure carte bâtiment dans les cartes visibles	Liste de Carte Bâtiment	CarteBatiment
PositionEstMeilleurPointVictoirePresente	Elle permet de trouver la position de la carte dans la liste des cartes visible	Liste de Carte Bâtiment	Integer
estMeilleurRessource	Elle permet de trouver l'ouvrier qui a le plus de compétence selon la ressources choisi	Integer Liste de Carte Ouvrier	CarteOuvrier
positionRessourceLaPlusEleve	Elle permet de trouver la position de de la ressource la plus élevée sur une carte	CarteBatiment	Integer
positionEstMeilleurRessource	Elle permet de trouver la position de la carte ouvrier qui a le plus de Compétences pour la Ressources choisis	Liste de Carte Ouvrier Integer	Integer

BotMedium

Nom de la méthode	Ce qu'elle fait	Attributs	Retour
jouer	Elle permet au bot de jouer	Liste de Carte Bâtiment Liste de Carte Ouvrier	Opération

BotHard

Nom de la méthode	Ce qu'elle fait	Attributs	Retour
jouer	Elle permet au bot de jouer	Liste de Carte Bâtiment Liste de Carte Ouvrier	Opération

Joueur

Nom de la méthode	Ce qu'elle fait	Attributs	Retour
ecuNecessaire	Elle permet de savoir combien d'écu il manque au joueur pour pouvoir faire une action	/	Integer
toStringStats	Elle permet d'avoir tous les Statistiques du joueur en cours	/	Tableau de String

Package partie

Ce package regroupe le code lié à l'exécution de la partie ainsi que le moteur de jeu. Le rôle de ce dernier est de vérifier que le déroulement de la partie s'effectue sans tricherie.

Partie

Nom de la méthode	Ce qu'elle fait	Attributs	Retour
lancerPartie	Elle lance une partie	/	/
déroulementPartie	Elle gère le déroulement de la partie	/	/
afficheScore	Elle permet d'afficher les scores des joueurs	/	/
afficheStatistiques	Elle permet d'afficher les statistiques	/	/
estFini	Elle permet de voir si la partie est fini	/	Boolean
calculGagnant	Elle permet de voir quel joueur à gagner la partie	/	Joueur
remplirCarteBatimentPresent	Elle permet de remplir la liste des carte Bâtiment visible	/	/
remplirCarteOuvrierPresent	Elle permet de remplir la liste des carte Bâtiment visible	/	/

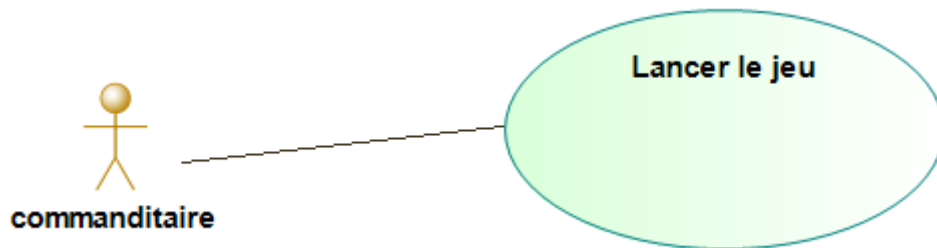
Modélisation de l'application

Analyse des besoins

Introduction

Pour pouvoir analyser comme il se doit les besoins que va nécessiter le sujet, le mieux est d'utiliser des cas d'utilisation (ou « use case »), qui sont leur format textuel ou des diagrammes de Cas d'utilisation qui sont leur format graphique. Ils servent à représenter les interactions qu'auront le programme avec l'utilisateur.

Diagrammes de Cas d'utilisation



Scénarios (Use Case)

Cas Bot Basique

Nom : Lancer le jeu

Acteur(s) : Commanditaire

Description : Le lancement du jeu doit être possible pour un commanditaire

Auteur : Yannick ALCARAZ

Date(s) : 10/11/2020

Préconditions : L'utilisateur doit avoir le jeu

Démarrage : L'utilisateur a demandé de lancer le jeu

-----**DESCRIPTION** : -----

Le scénario nominal :

1. Le système lance le jeu
2. Le système lance la partie
3. Le système affiche le déroulement de la partie.
4. Le Bot Basique attend son tour
5. Le système indique que c'est au Bot Basique de jouer
6. Le Bot Basique récupère une carte bâtiment
- 7 Le système lui retire une action
8. Le Bot Basique récupère une carte ouvrier
- 9 Le système lui retire une action
10. Le Bot Basique affecte l'ouvrier au bâtiment
- 11 Le système lui retire une action
12. Le Bot Basique gagne la partie
13. Le système affiche le gagnant de la partie.
14. Le système affiche les statistiques des 1000 parties
- 15 Le système vérifie le nombre de partie effectué, si < 1000 --> Retour 2
16. L'utilisateur quitte le jeu.
17. Le système éteint le jeu

Les scénarios alternatifs :

* L'utilisateur décide de quitter le jeu. Retour 17
10a. Le Bot Basique pioche des écus. Retour 10aa.
10aa. S'il pioche 1 écu, le système lui retire 1 action. Retour 12a
S'il pioche 3 écus, le système lui retire 2 actions. Retour 12a
S'il pioche 6 écus, le système lui retire 3 actions. Retour 12a
12a. Le Bot Basique ne gagne pas la partie et termine son tour. Retour 4

Postconditions : Aucun

-----**COMPLEMENTS :** -----

Ergonomie

L'affichage du déroulement d'une partie devra afficher l'ensemble des actions réalisées par les bots dans la partie

Performance attendue

Le déroulement de la partie doit se terminer par l'affichage du gagnant de la partie

Cas Bot Medium

Nom : Lancer le jeu
Acteur(s) : Commanditaire
Description : Le lancement du jeu doit être possible pour un commanditaire
Auteur : Yannick ALCARAZ
Date(s) : 10/11/2020

Préconditions : L'utilisateur doit avoir le jeu
Démarrage : L'utilisateur a demandé de lancer le jeu

-----**DESCRIPTION :** -----

Le scénario nominal :

1. Le système lance le jeu
2. Le système lance la partie
3. Le système affiche le déroulement de la partie.
4. Le Bot Medium attend son tour
5. Le système indique que c'est au Bot Medium de jouer
6. Le Bot Medium récupère la carte bâtiment qui a le plus de point de victoire
- 7 Le système lui retire une action
8. Le Bot Medium récupère la carte ouvrier avec la carte qui a le plus de bois/ ou de pierre /ou de savoir/ ou de tuile
- 9 Le système lui retire une action
10. Le Bot Medium affecte l'ouvrier au bâtiment
- 11 Le système lui retire une action
12. Le Bot Medium gagne la partie
13. Le système affiche le gagnant de la partie.
14. Le système affiche les statistiques des 1000 parties
- 15 Le système vérifie le nombre de partie effectué, si < 1000 --> Retour 2
16. L'utilisateur quitte le jeu.
17. Le système éteint le jeu

Les scénarios alternatifs :

* L'utilisateur décide de quitter le jeu. Retour 17
10a. Le Bot Medium pioche des écus. Retour 10aa.
10aa. S'il pioche 1 écu, le système lui retire 1 action. Retour 12a
S'il pioche 3 écus, le système lui retire 2 actions. Retour 12a
S'il pioche 6 écus, le système lui retire 3 actions. Retour 12a

12a. Le Bot Medium ne gagne pas la partie et termine son tour. Retour 4

Postconditions : Aucun

-----COMPLEMENTS : -----

Ergonomie

L'affichage du déroulement d'une partie devra afficher l'ensemble des actions réaliser par les bots dans la partie

Performance attendue

Le déroulement de la partie doit se terminer par l'affichage du gagnant de la partie

Conception logicielle

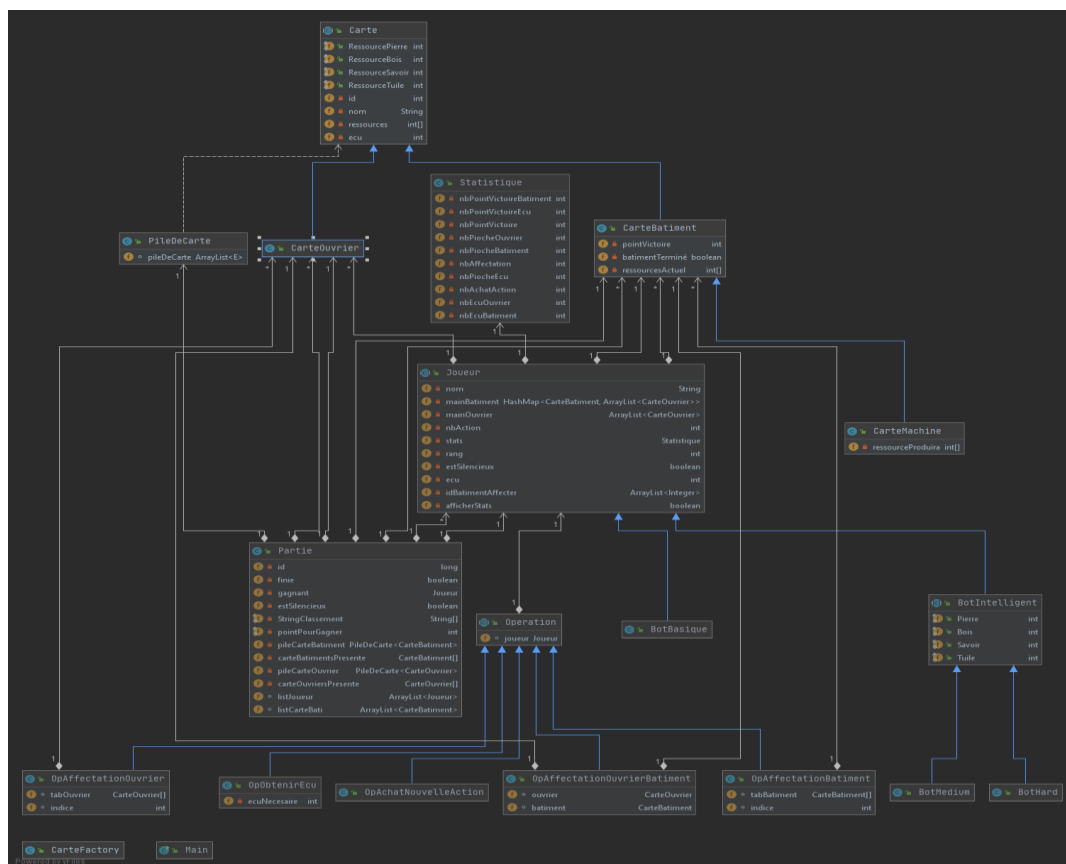
Introduction

Pour la conception de notre logiciel, nous avons effectués des diagrammes de classe qui permet de répertorier toutes les classes mais aussi méthodes et fonctions programmés. De plus, nous avons des diagrammes de séquence qui permettent de voir la durée de vie et l'utilité de nos méthodes. Toutes les méthodes citées dans les prochains diagrammes seront expliquées de manière détaillée dans [Méthodes fonctionnelles](#)

Diagrammes des Classes

Générale

Ce dernier ne possède pas les méthodes pour des questions de lisibilités mais ces dernières seront visibles dans les suivantes.

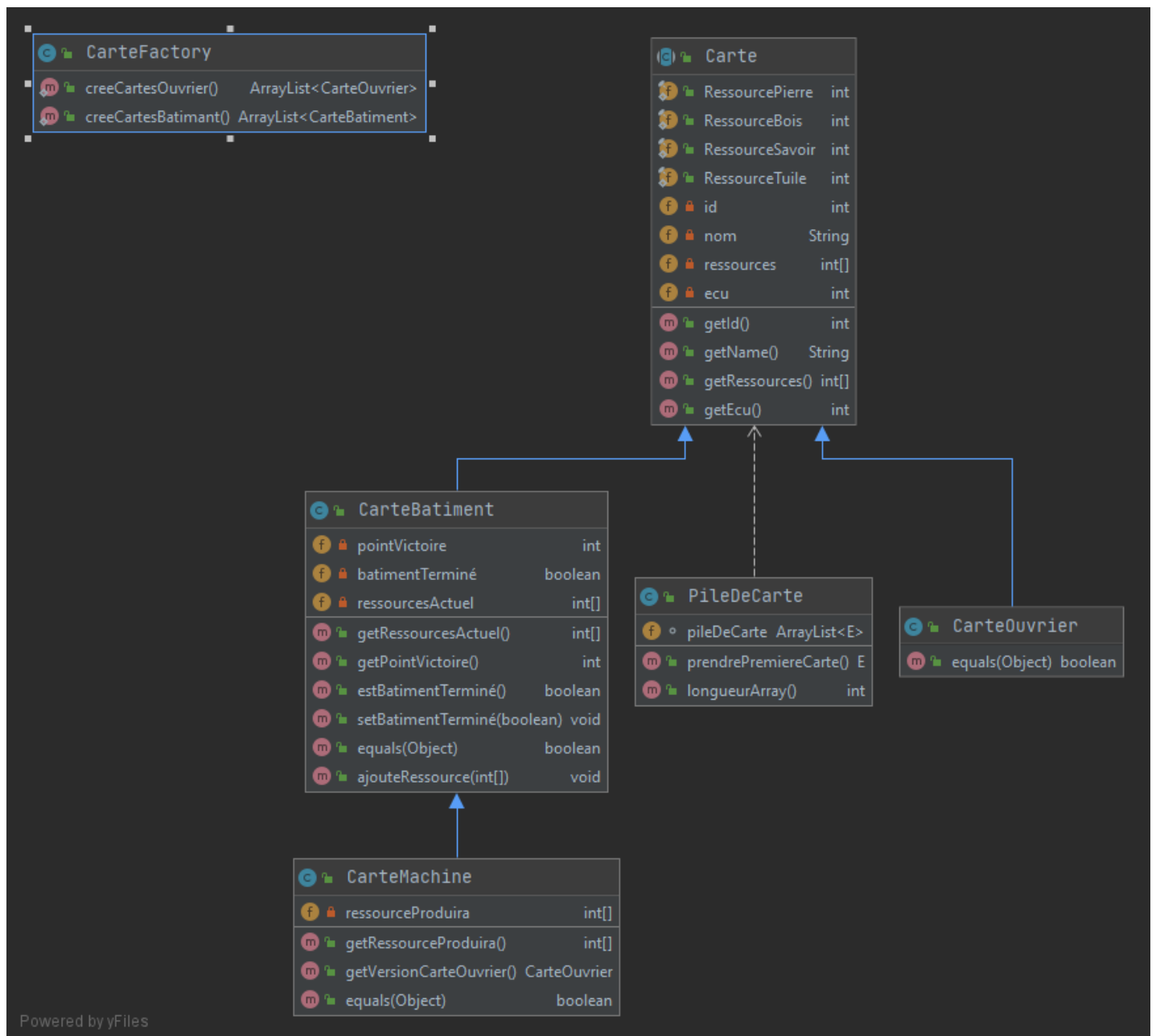


Chaque carte appartient à une catégorie de carte (Bâtiment, Machine, Ouvrier) et c'est catégorie se différencie par leur présentation mais surtout par leurs données et leurs fonctionnalités.

Il a donc été décidé de créer une class mère abstraite « Carte », qui ne sera pas instanciable car une carte doit obligatoirement dans une catégorie, elle sera donc instanciable via les classes filles « CarteOuvrier », « CarteBatiment » et « CarteMachine » (classe fille de Bâtiment et donc de Carte).

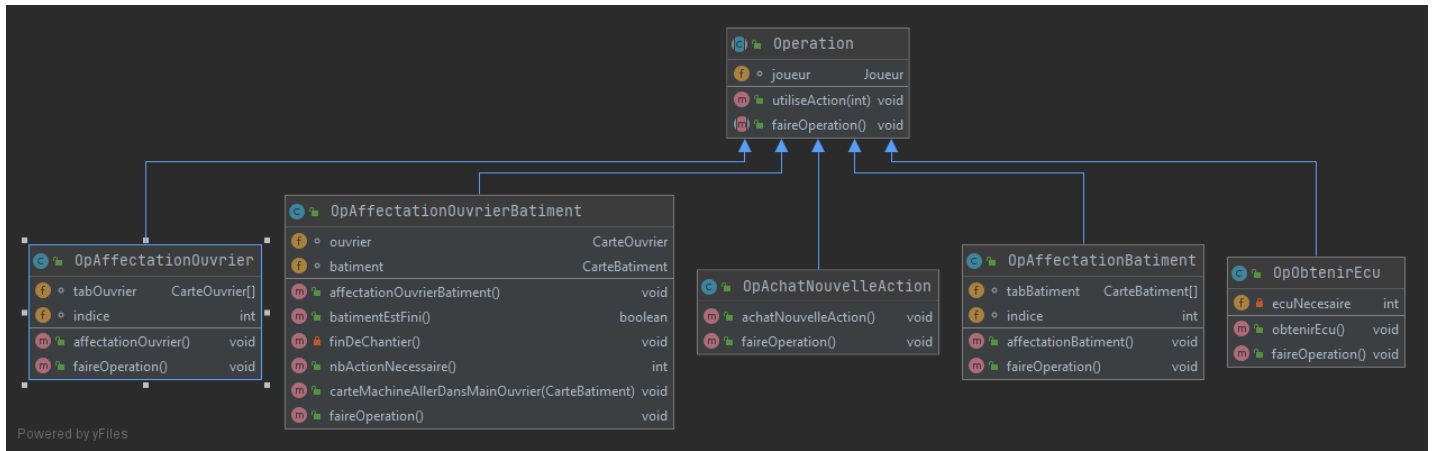
Afin de gérer une pile de carte (qu'elles soient une Carte Batiment, Machine ou Ouvrier), une classe générique « PileDeCarte » a été créé. Cette classe générique n'accepte seulement que les Objets de la classe « Carte » ou des classes filles.

La class CarteFactory possède seulement 2 méthodes statiques qui permet de retourner une collection de cartes ouvriers et de bâtiments (qui comprennent les cartes machines) prédéfinie. Cela évite une réécriture constante de l'instanciation des cartes.



Package Opération

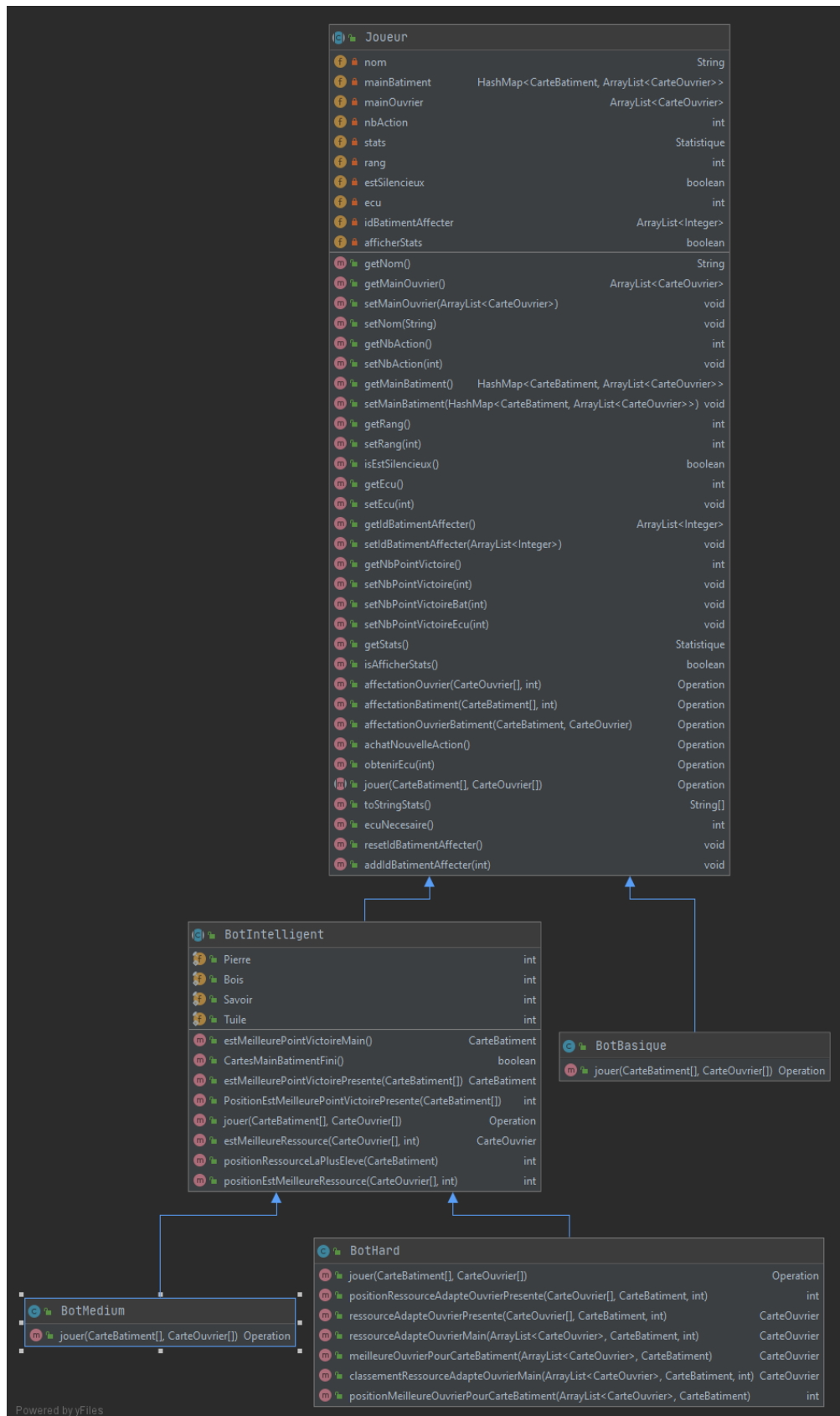
Afin de différencier les différentes opérations, une classe mère abstraite a été créée « Opération ». Cela permettra d'avoir dans chaque classes filles d'avoir la methode « faireOperation() » qui permettra d'exécuter l'opération. Ces classes filles sont « OpAffectationOuvrier », « OpAffectationBatiment », « OpObtenirEcu », « OpAffectationOuvrierBatiment » et « OpNouvelleAchat ».



Package Joueur

Chaque joueur aura différent niveau de stratégie, soit les même tâches mais pas les même résultats. De ce fait, il a été décidé de créer une classe abstraite « Joueur » afin que les différentes tâches soient les mêmes mais avec des résultats. Ces tâches sont définies dans des classes filles « BotBasique », « BotIntelligent », « BotMedium » et « BotHard ».

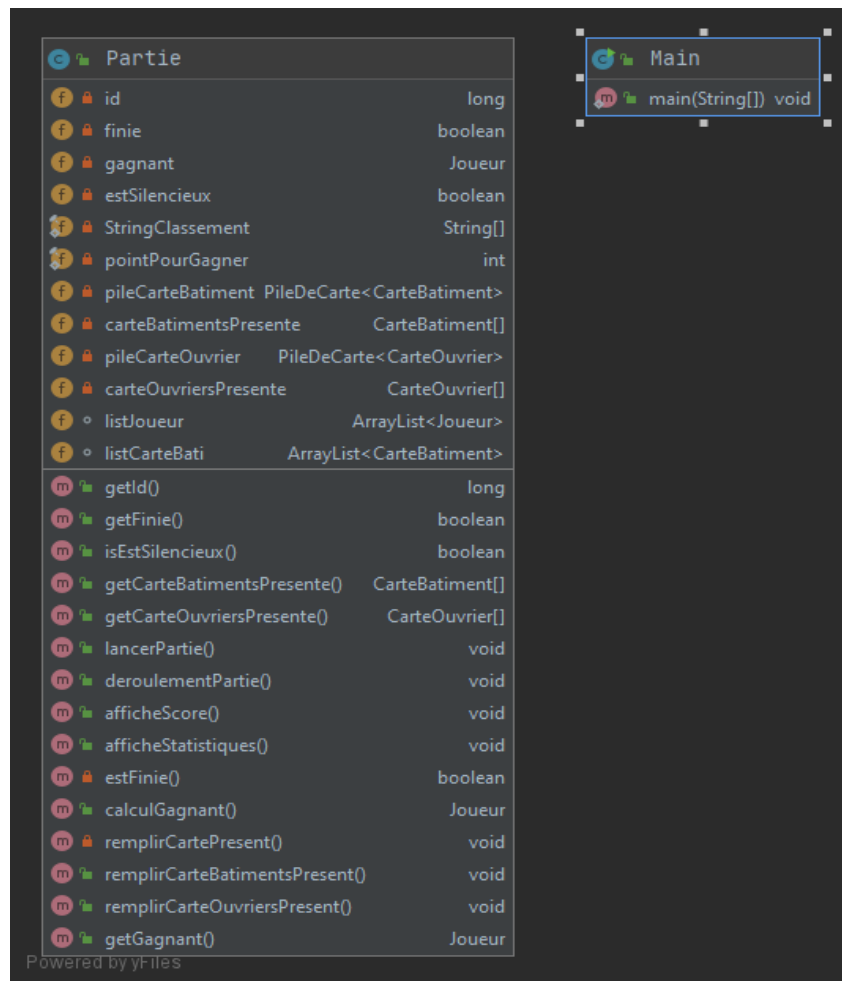
La classe abstraite « BotIntelligent » permet de rajouter des méthodes abstraites à ces classes filles « BotMedium » et « BotHard ».



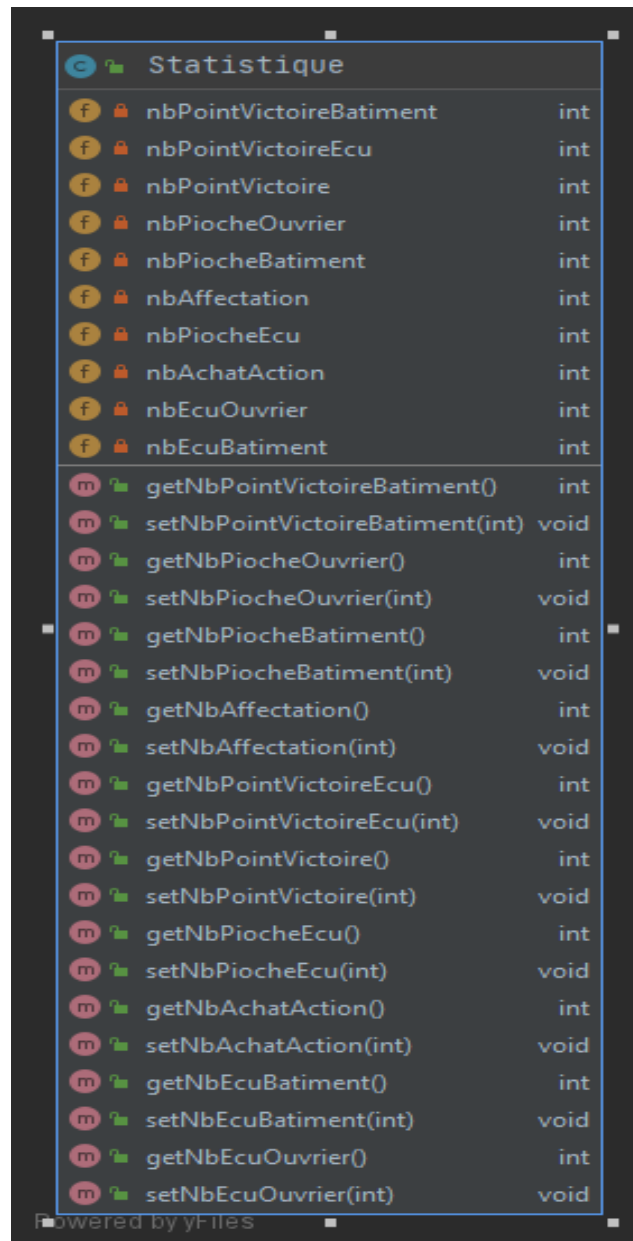
Package Partie

Afin de différencier les parties, une classe « Partie » a été créée. Chaque partie ne se finira pas de la même manière.

Afin d'exécuter et de tester les parties la classe « Main » possède la méthode « main ». Cela exécutera 1000 parties.



Afin que le client connaisse, les détails de la partie à la fin d'une partie, la classe « Statistique » permettra d'enregistrer certaine donnée. Ces données évolueront pendant la partie et se fixeront à la fin de partie.



The screenshot shows a Java IDE window titled "Statistique". The class contains 10 private integer attributes and 20 public methods (10 getters and 10 setters). The attributes are: nbPointVictoireBatiment, nbPointVictoireEcu, nbPointVictoire, nbPiocheOuvrier, nbPiocheBatiment, nbAffectation, nbPiocheEcu, nbAchatAction, nbEcuOuvrier, and nbEcuBatiment. The methods are: getNbPointVictoireBatiment(), setNbPointVictoireBatiment(int), getNbPiocheOuvrier(), setNbPiocheOuvrier(int), getNbPiocheBatiment(), setNbPiocheBatiment(int), getNbAffectation(), setNbAffectation(int), getNbPointVictoireEcu(), setNbPointVictoireEcu(int), getNbPointVictoire(), setNbPointVictoire(int), getNbPiocheEcu(), setNbPiocheEcu(int), getNbAchatAction(), setNbAchatAction(int), getNbEcuBatiment(), setNbEcuBatiment(int), getNbEcuOuvrier(), and setNbEcuOuvrier(int). The IDE interface includes a search bar at the top, a list of files on the left, and a status bar at the bottom that says "Powered by yFiles".

Statistique		
f	nbPointVictoireBatiment	int
f	nbPointVictoireEcu	int
f	nbPointVictoire	int
f	nbPiocheOuvrier	int
f	nbPiocheBatiment	int
f	nbAffectation	int
f	nbPiocheEcu	int
f	nbAchatAction	int
f	nbEcuOuvrier	int
f	nbEcuBatiment	int
m	getNbPointVictoireBatiment()	int
m	setNbPointVictoireBatiment(int)	void
m	getNbPiocheOuvrier()	int
m	setNbPiocheOuvrier(int)	void
m	getNbPiocheBatiment()	int
m	setNbPiocheBatiment(int)	void
m	getNbAffectation()	int
m	setNbAffectation(int)	void
m	getNbPointVictoireEcu()	int
m	setNbPointVictoireEcu(int)	void
m	getNbPointVictoire()	int
m	setNbPointVictoire(int)	void
m	getNbPiocheEcu()	int
m	setNbPiocheEcu(int)	void
m	getNbAchatAction()	int
m	setNbAchatAction(int)	void
m	getNbEcuBatiment()	int
m	setNbEcuBatiment(int)	void
m	getNbEcuOuvrier()	int
m	setNbEcuOuvrier(int)	void

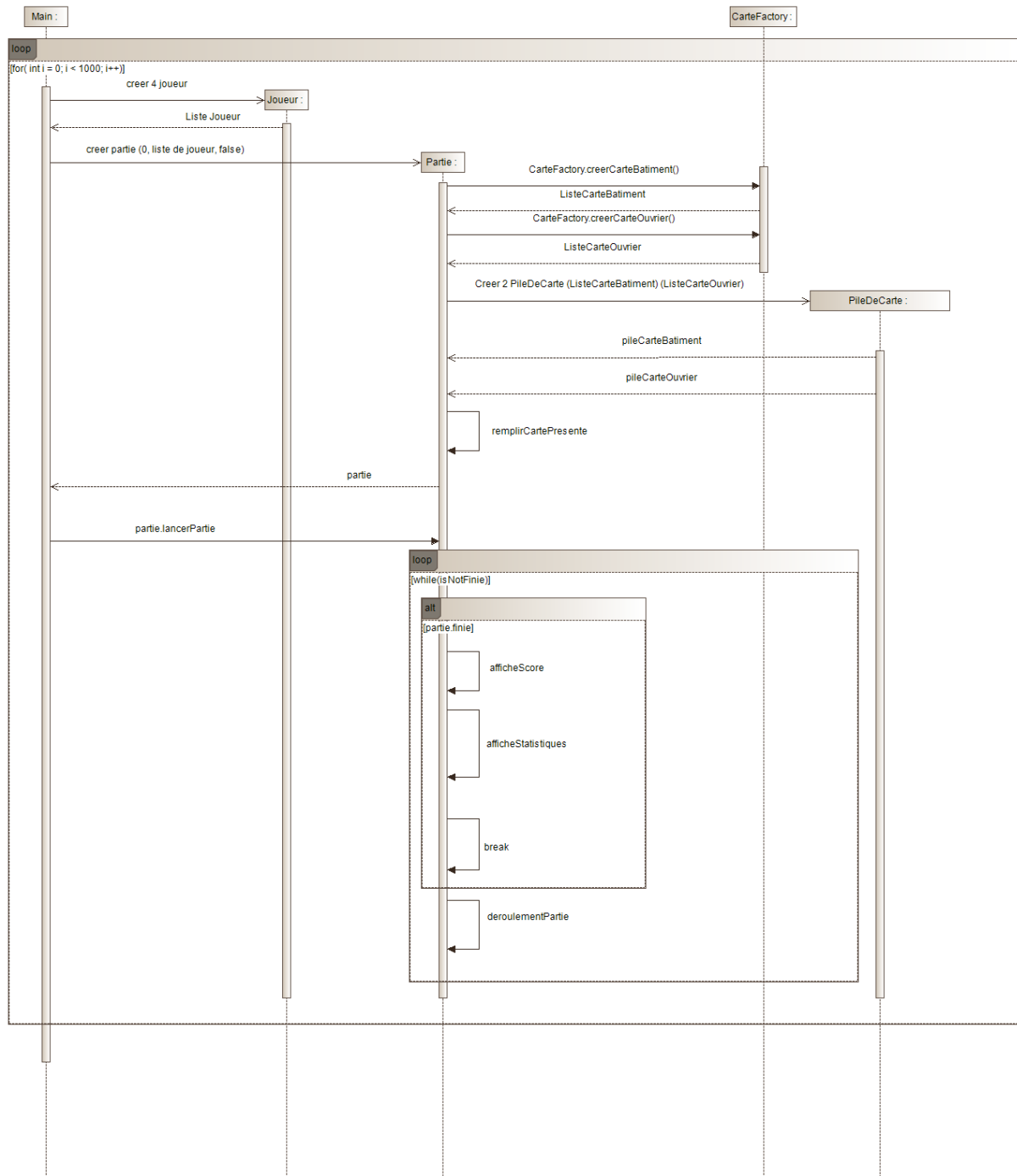
Powered by yFiles

Diagrammes des Séquences

Partie

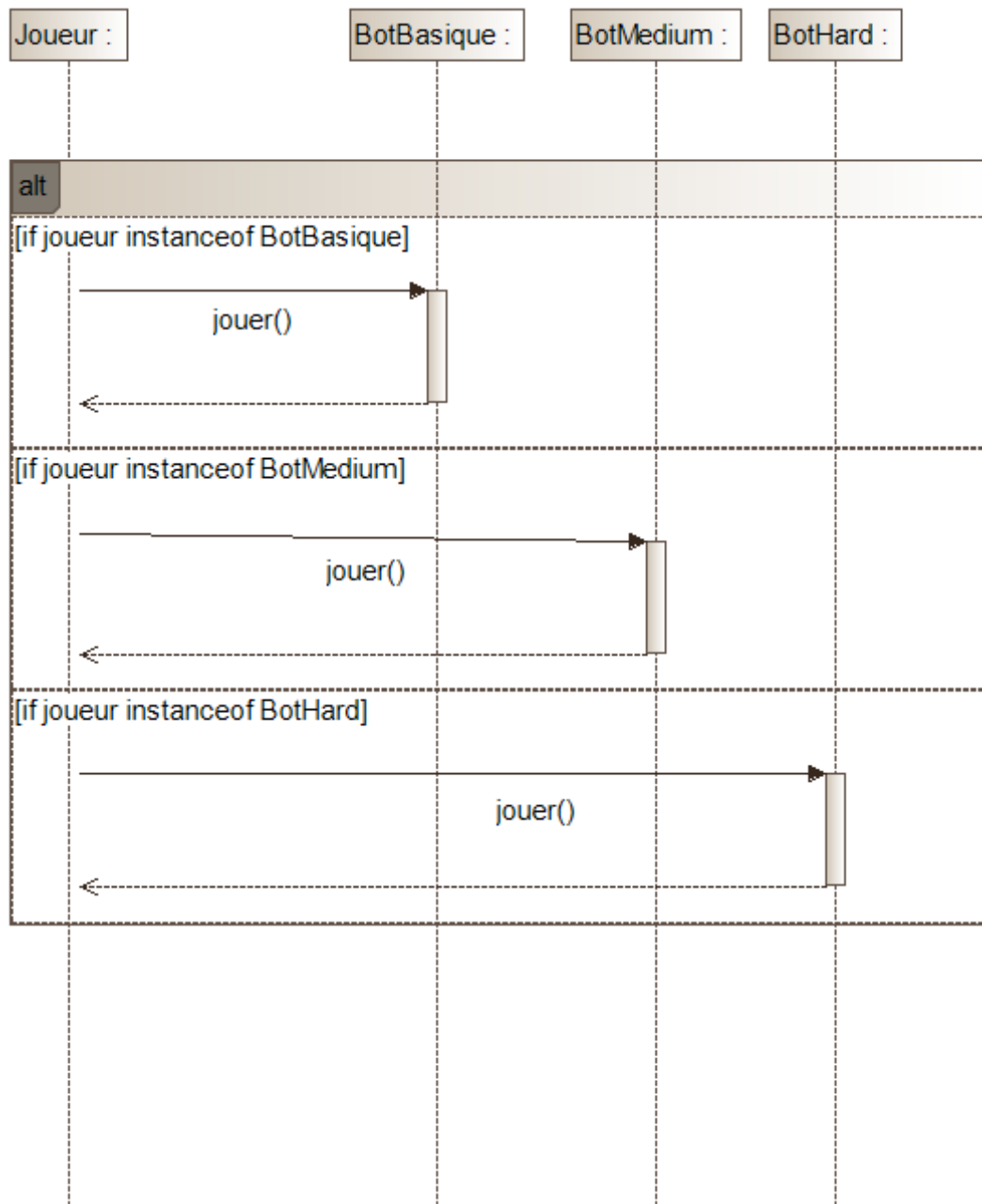
LancerPartie

Lorsque l'utilisateur exécute une partie, la classe main crée 4 joueurs ensuite la partie charge toutes les cartes du jeu en les mélangeant. Ensuite ces cartes sont ajoutées à une pile de cartes ouvrier et bâtiment ainsi que les cartes présentes sur le jeu. Ainsi une partie se lance et on affiche le déroulement de la partie. Une fois la partie terminée on affiche les statistiques de la partie.



jouer

Dans la classe Joueur, si le joueur est un bot basique alors on rentre dans la méthode jouer du bot et on return joueur. Sinon si le joueur est un bot medium alors on rentre dans la méthode jouer du bot et on return joueur. Sinon le joueur est un bot hard basique alors on rentre dans la méthode jouer du bot et on return joueur



jouerBotBasique

Dans la classe botBasique, on initie firstBatiment a null. On rentre dans la boucle for pour vérifier si la main du bot ne comporte pas de carte ouvrier. Ensuite on rentre dans le si et on vérifie si firstBatiment est égale a nul, alors on récupère la première carte bâtiment présente dans le jeu qu'on met dans la main du bot. Sinon on passe ou sinon si ou là on vérifie si la main de l'ouvrier et elle aussi vide, si c'est le cas on récupère la première carte ouvrière présente dans le jeu qu'on met dans la main du bot. Sinon on rentre dans une nouvelle condition si en vérifiant si le bot n'a pas assez d'ecu pour affecter le premier ouvrier sur un bâtiment alors on fait l'action ObtenirEcu. Sinon on affecte le premier bâtiment de la main du bot en construction et on y affecte des ouvriers dessus pour le finir



jouerBotMedium

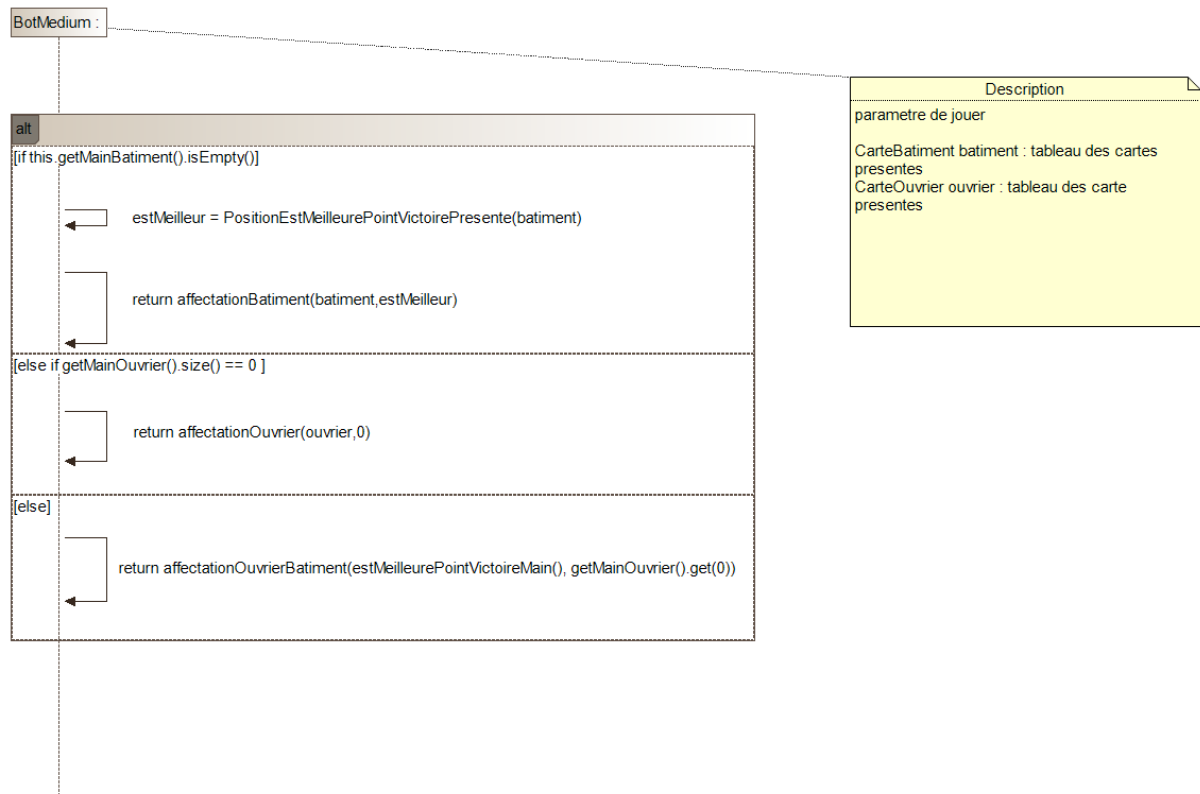
Dans la classe BotMedium

On regarde si la main bâtiment du bot est vide, si elle est vide on appelle la méthode PositionEstMeilleure avec comme paramètre les carte bâtiment présente.

Ensuite on affecte le meilleur bâtiment avec le plus de point de victoire a la main du bot.

Sinon si la main carte batiment contient une carte on regarde si la main ouvrier du bot est vide, est la on recupere la premiere carte ouvrier presente et on l'affecte a la main du bot.

Si la main du bot contient une carte bâtiment et ouvrier alors on affecte le bâtiment en construction et on affecte le premier ouvrier de la main au bâtiment pour le construire



estMeilleurePointVictoireMain

Dans la classe BotIntelligent

La méthode récupère la première carte bâtiment de la main du bot et la définit pour le moment comme la meilleure, elle rentre dans une boucle for de la longueur du tableau.

Pour chaque boucle qu'on fait dans le for on regarde la carte i (valeur de la boucle) et on regarde si la carte de la valeur i a plus de points de victoire que la carte meilleure définie dans la variable.

Si oui on l'enregistre comme nouvelle meilleure et on avance dans la boucle

Sinon on laisse l'ancienne carte meilleure et on avance dans la boucle

Une fois la boucle finie on retourne la meilleure carte bâtiment de la main avec le plus de victoire.



PositionEstMeilleurePointVictoirePresente

Dans la classe BotIntelligent la méthode initie la positionMeilleur à 0.

Ensuite on considère que le meilleur bâtiment avec le plus de victoire est le premier.

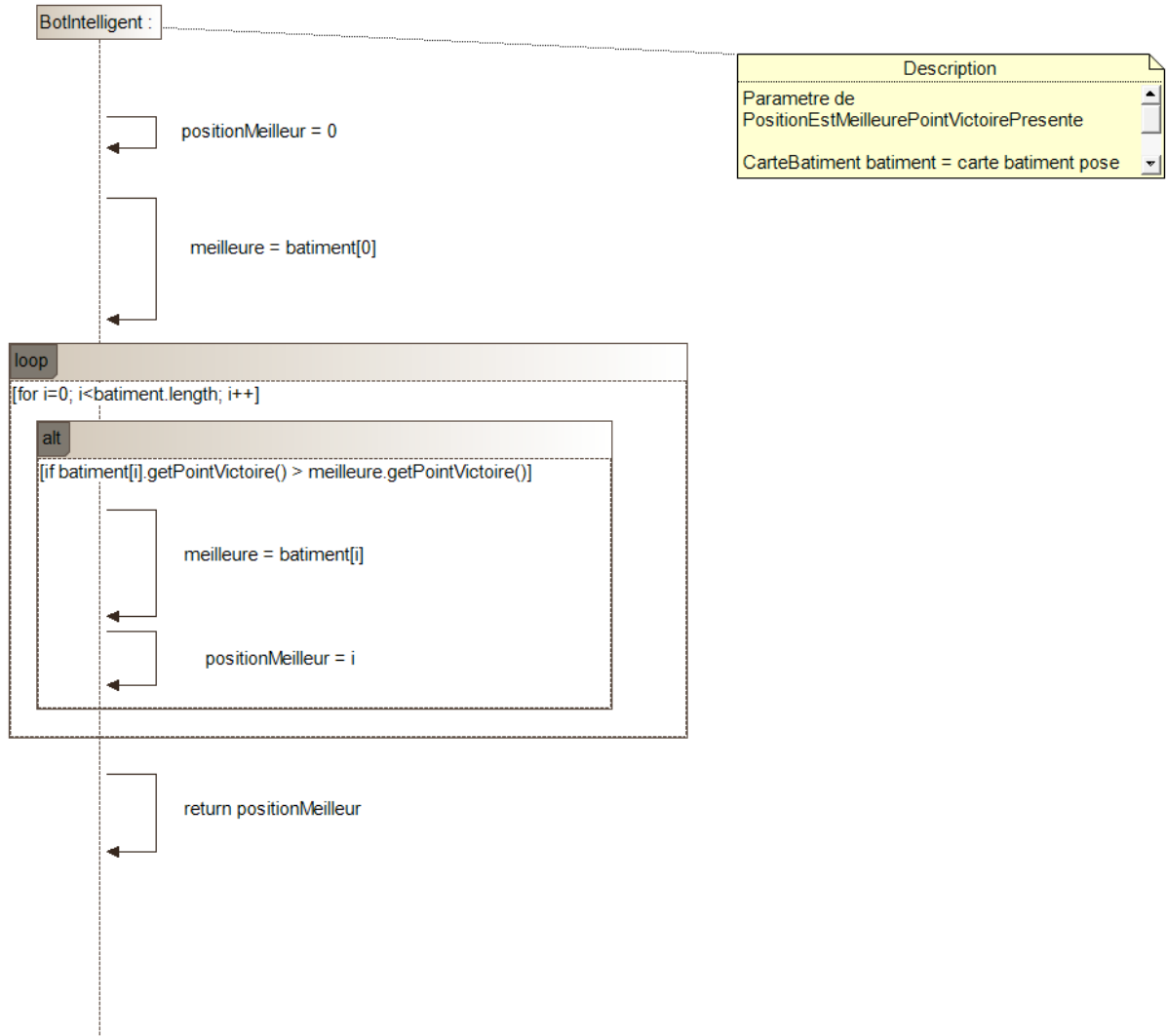
Ensuite on rentre dans la boucle for avec comme longueur le nombre de cartes a pioché.

Pour chaque boucle qu'on fait dans le for on regarde la carte i (valeur de la boucle) et on regarde si la carte de la valeur i a plus de points de victoire que la carte meilleure définie dans la variable.

Si oui on récupère la valeur i et on l'enregistre dans positionMeilleur.

Sinon on laisse l'ancienne valeur et on avance dans la boucle

Une fois la boucle finie en retourne la position de la carte bâtiment avec plus de victoire



estMeilleurePointVictoirePresente

Dans la classe BotIntelligent

La méthode récupère la première carte bâtiment des cartes présenté et la définit pour le moment comme la meilleure, elle rentre dans une boucle for de la longueur du tableau.

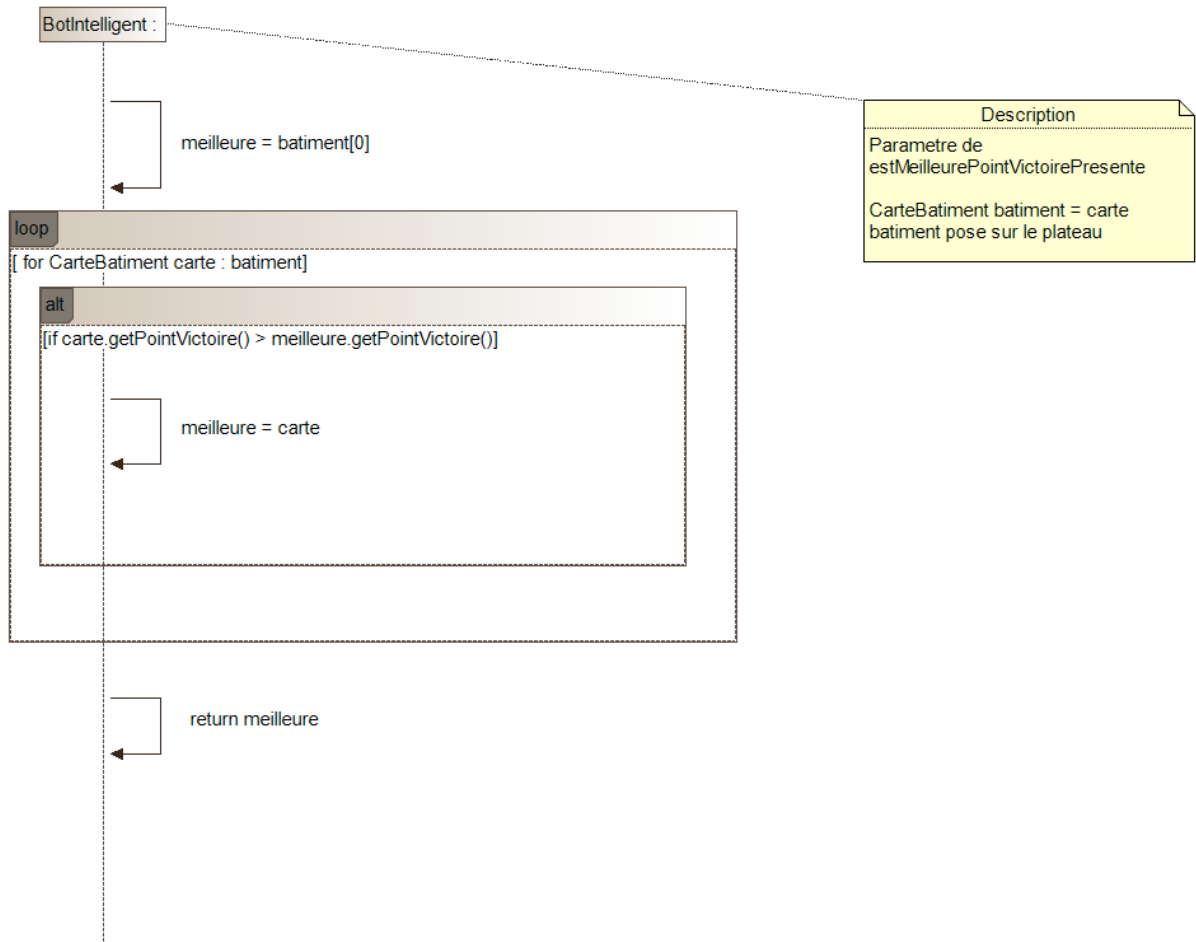
Ensuite on rentre dans la boucle for avec comme longueur le nombre de carte à piocher.

Pour chaque boucle qu'on fait dans le for on regarde la carte i (valeur de la boucle) et on regarde si la carte de la valeur i a plus de point de victoire que la carte meilleure définie dans la variable.

Si oui on l'enregistre comme nouvelle meilleure et on avance dans la boucle

Sinon on laisse l'ancienne carte meilleure et on avance dans la boucle

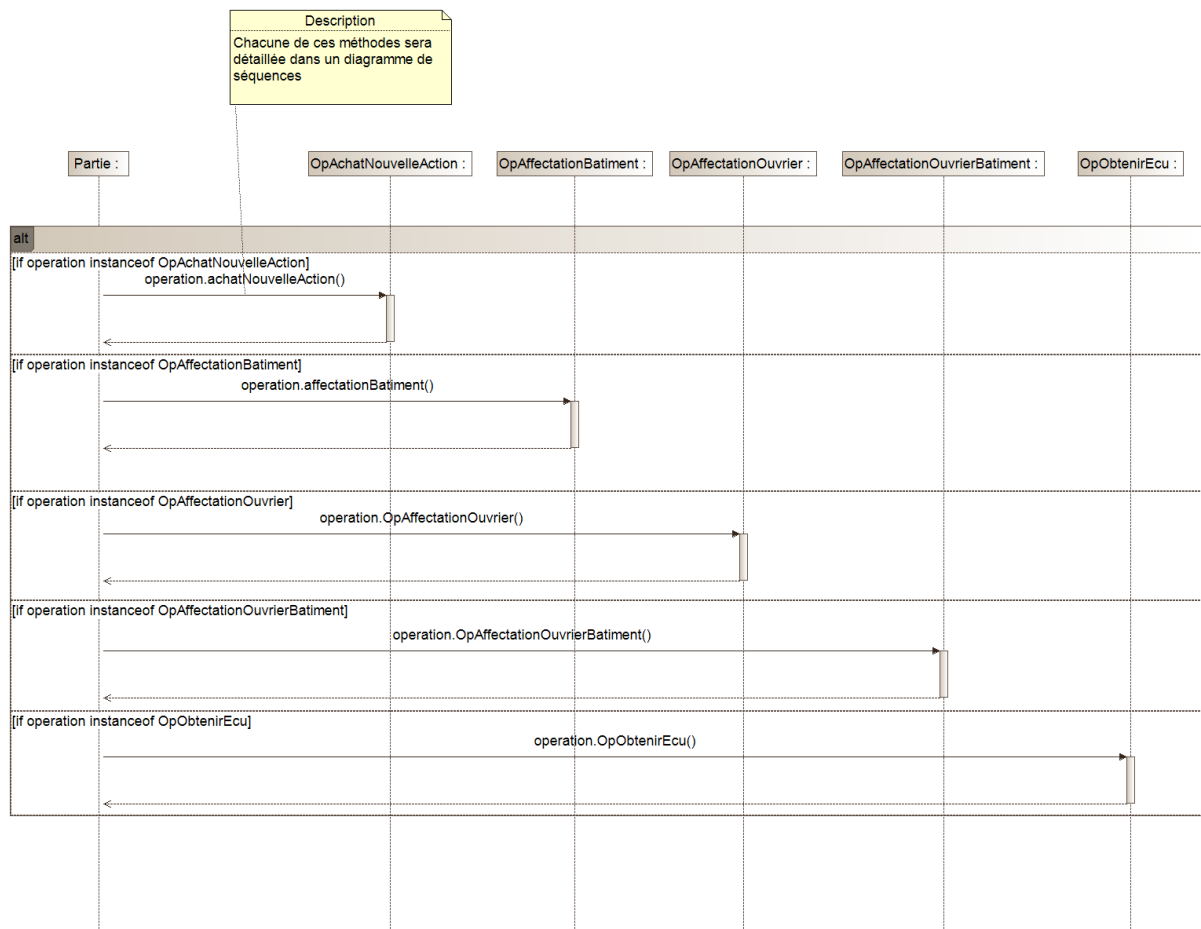
Une fois la boucle finie on retourne la meilleure carte bâtiment présenté avec le plus de victoire.



Operation

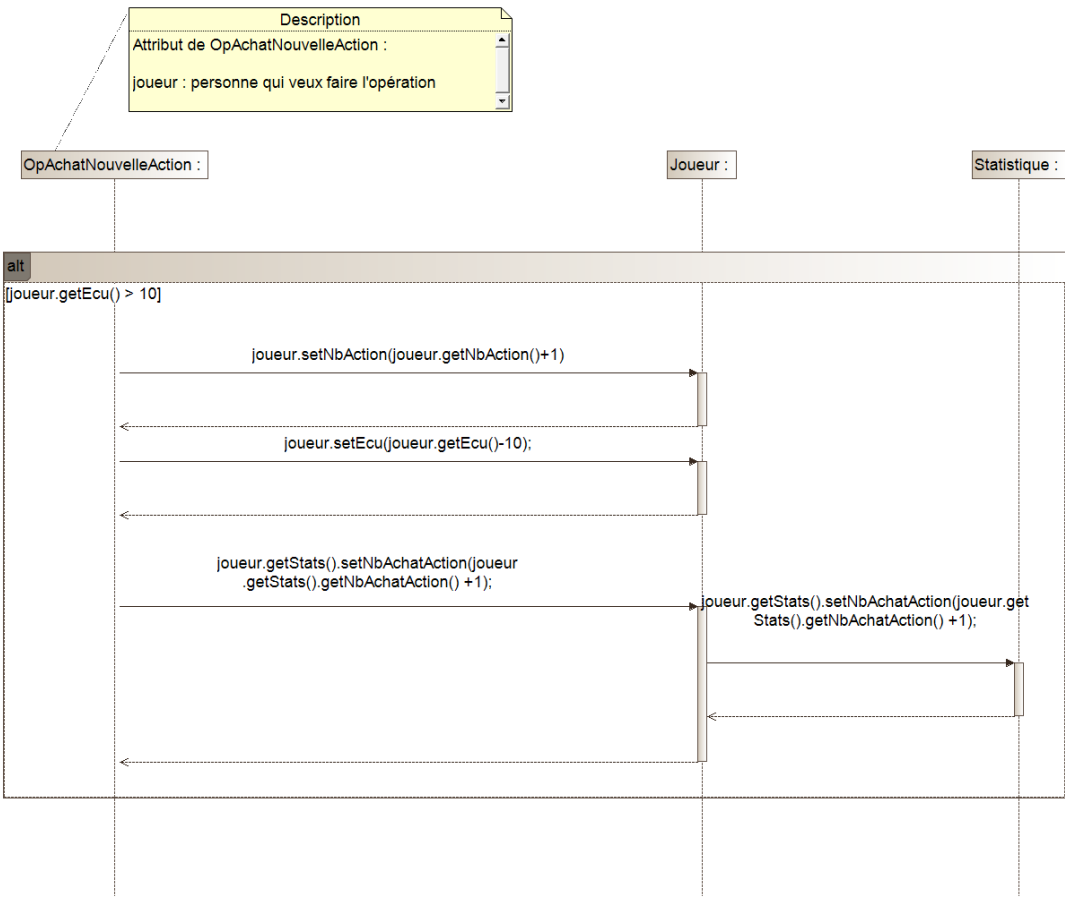
FaireOperation

Après avoir reçu l'opération de joueur, la partie l'exécute. En fonction de son instanciation, une certaine méthode est appelée. Cette méthode va alors faire une action.



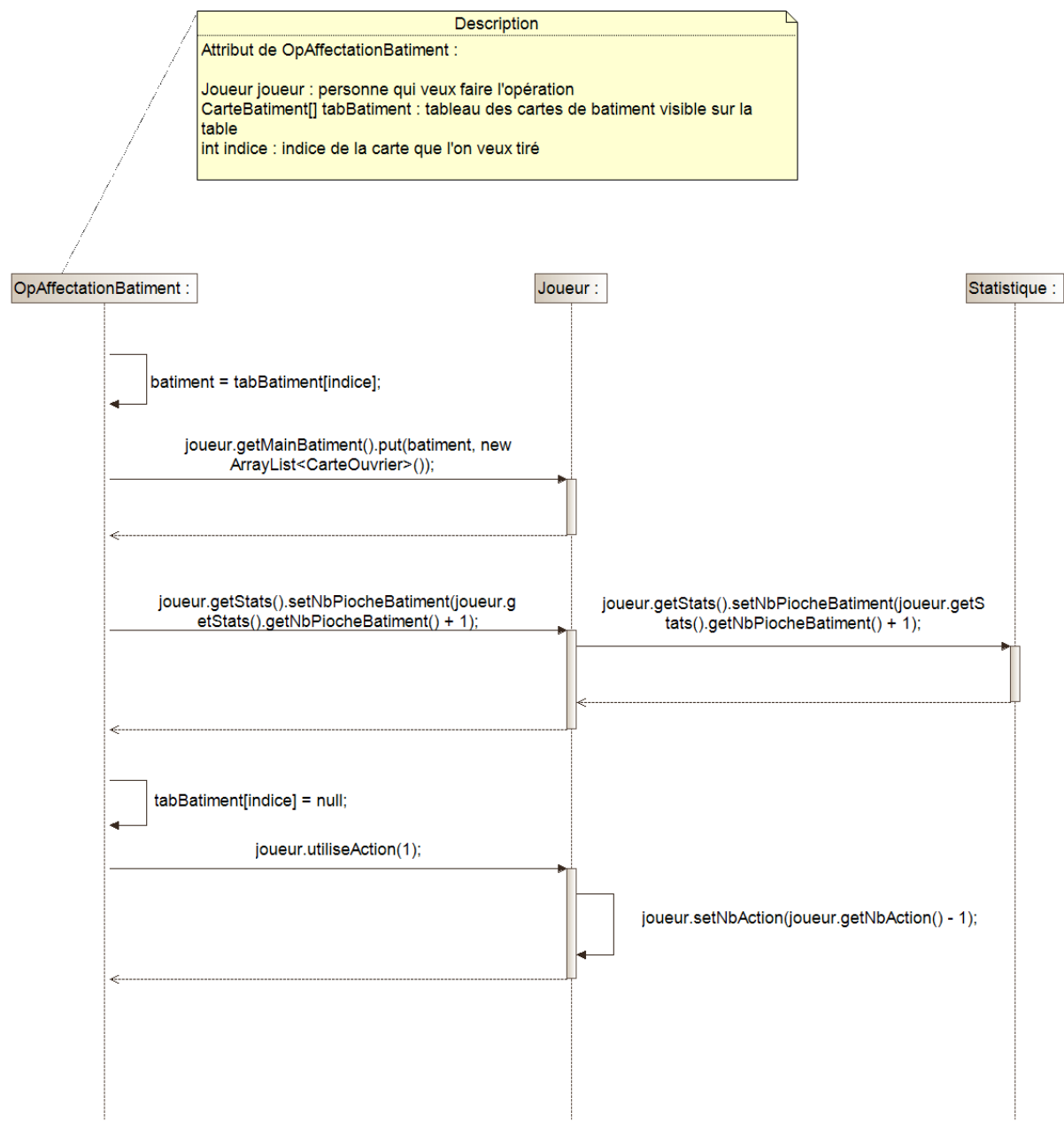
achatNouvelleAction

Cette méthode permet au joueur d'acter une action contre 10 écus.



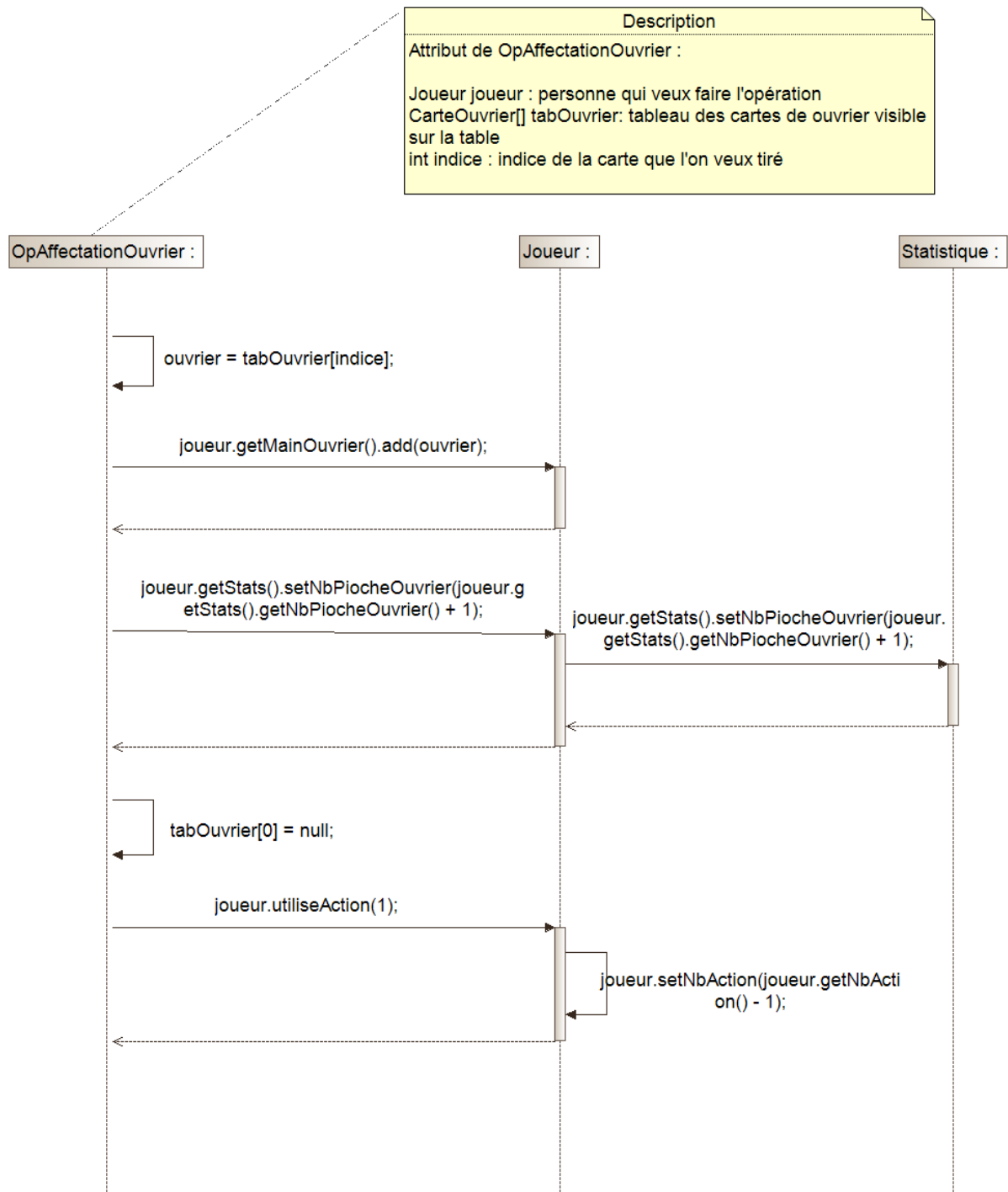
AffectationBatiment

Ce diagramme décrit l'action de tirer une carte bâtiment.



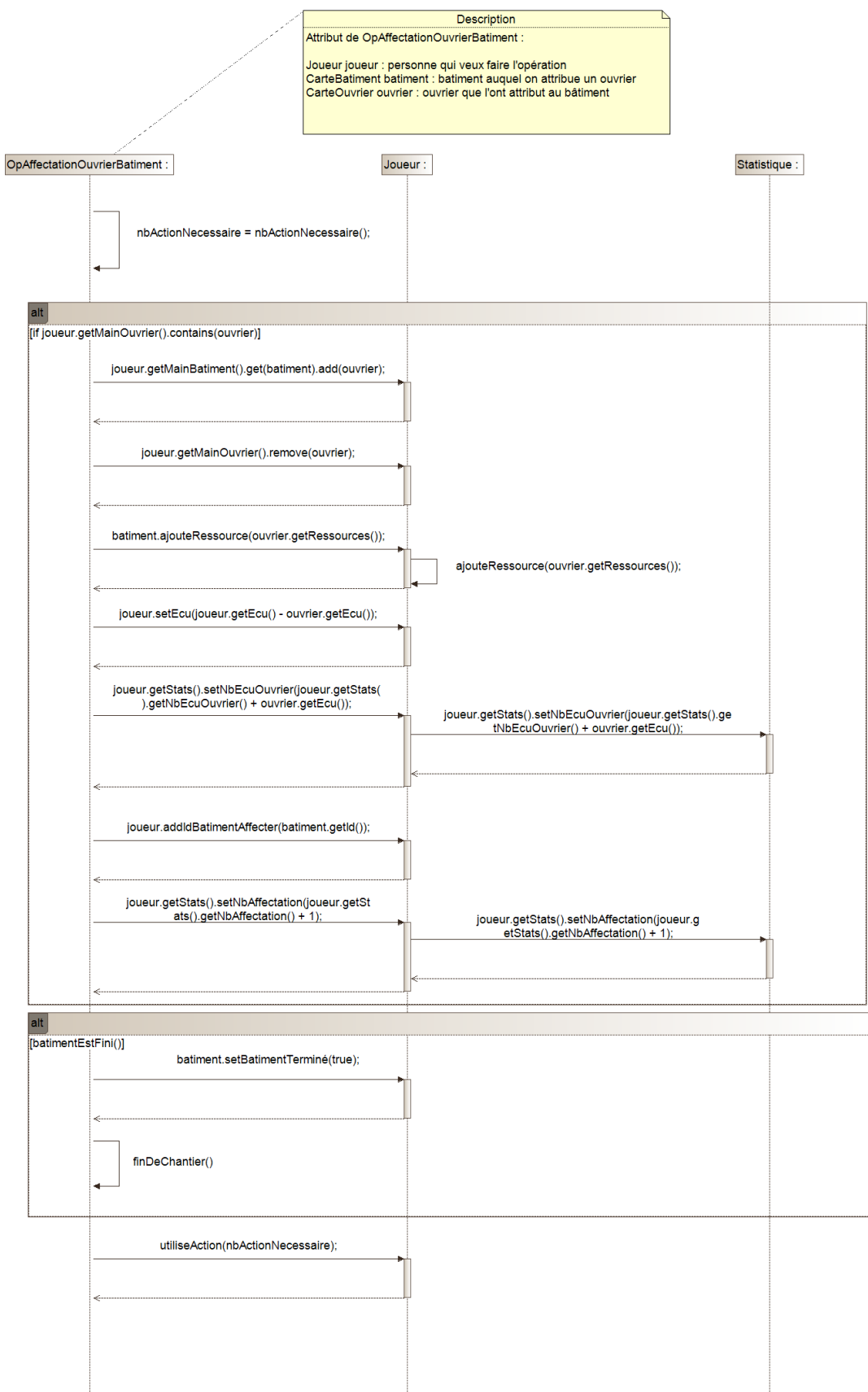
affectionOuvrier

Ce diagramme décrit l'action de tirer une carte ouvrier.



affectationOuvrierBatiment

Ce diagramme nous explique l'action d'affecter un ouvrier à un bâtiment.



ajouteRessource

Ce diagramme montre l'ajout des ressources à une carte bâtiment.



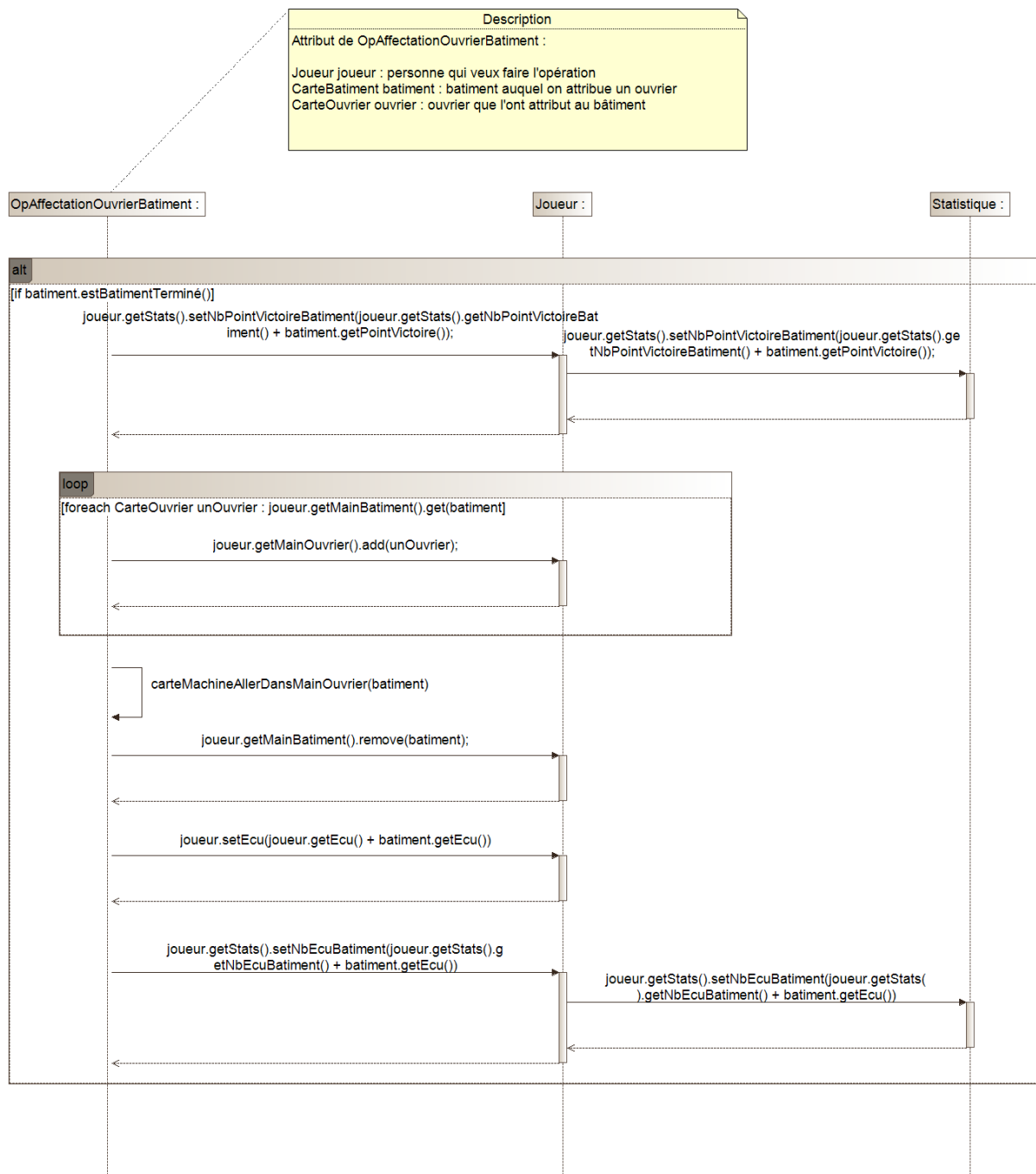
batimentEstTerminé

Ce diagramme montre comment se déroule la vérification qu'un bâtiment est fini.



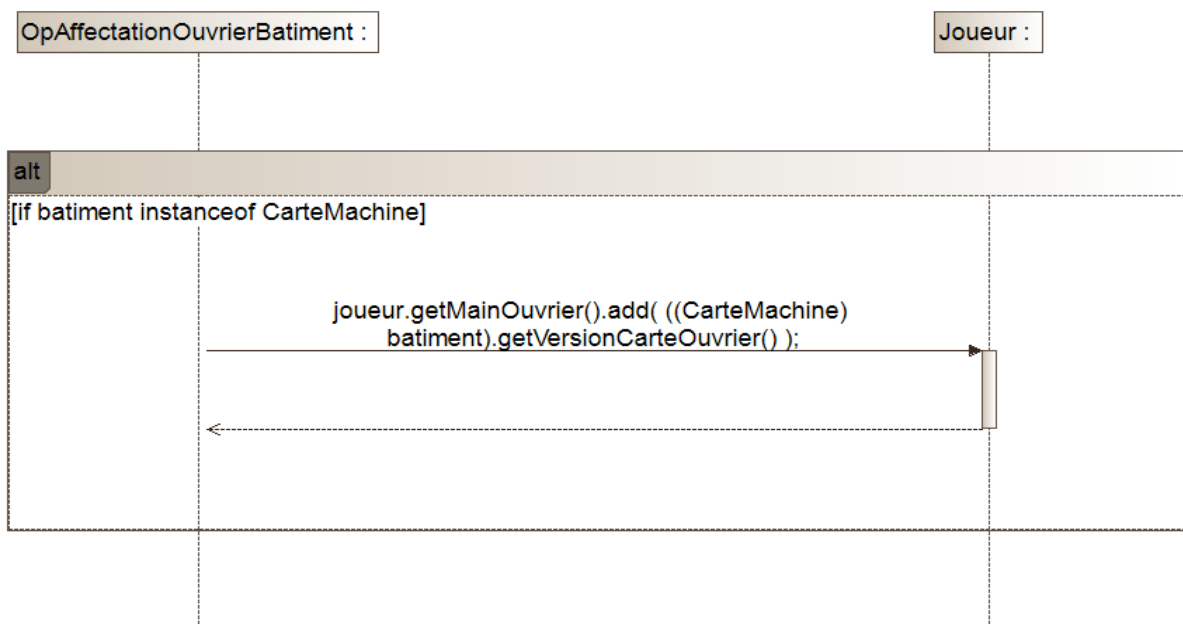
finDeChantier

Ce diagramme décrit ce qui se passe lorsqu'un bâtiment est fini.



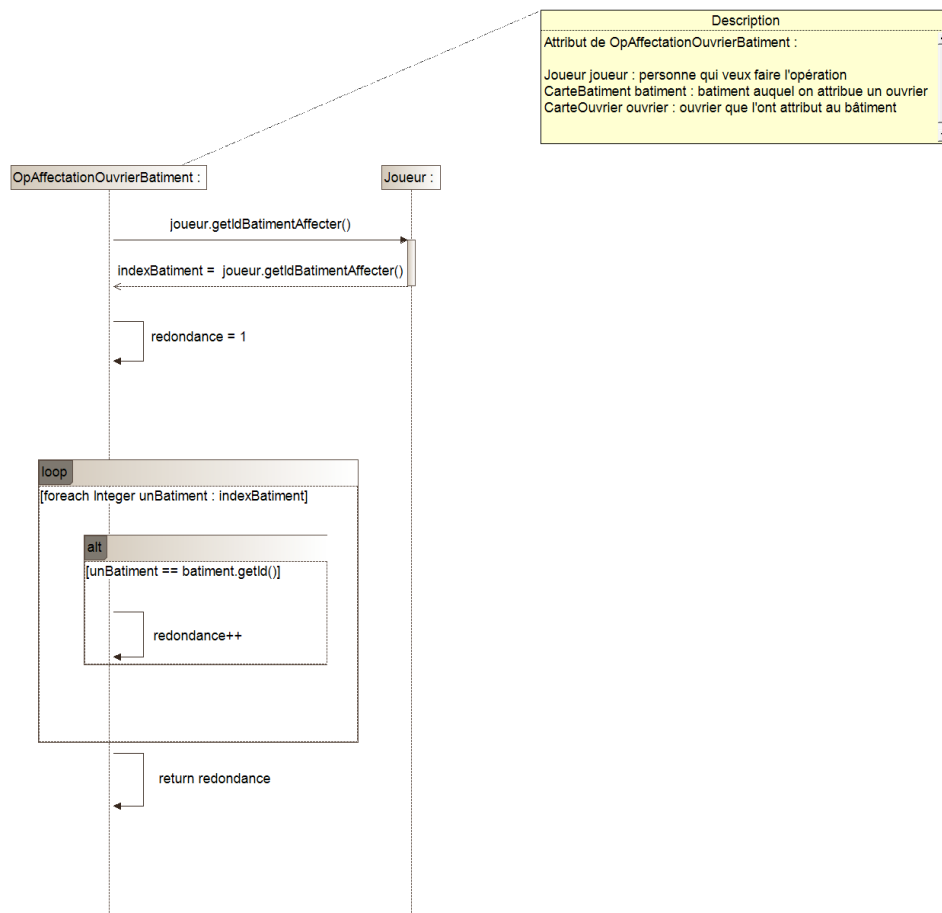
carteMachineAllerDansMainOuvrier

Lorsqu'un bâtiment se termine, si celui-ci est une machine, alors je le convertis en ouvrier



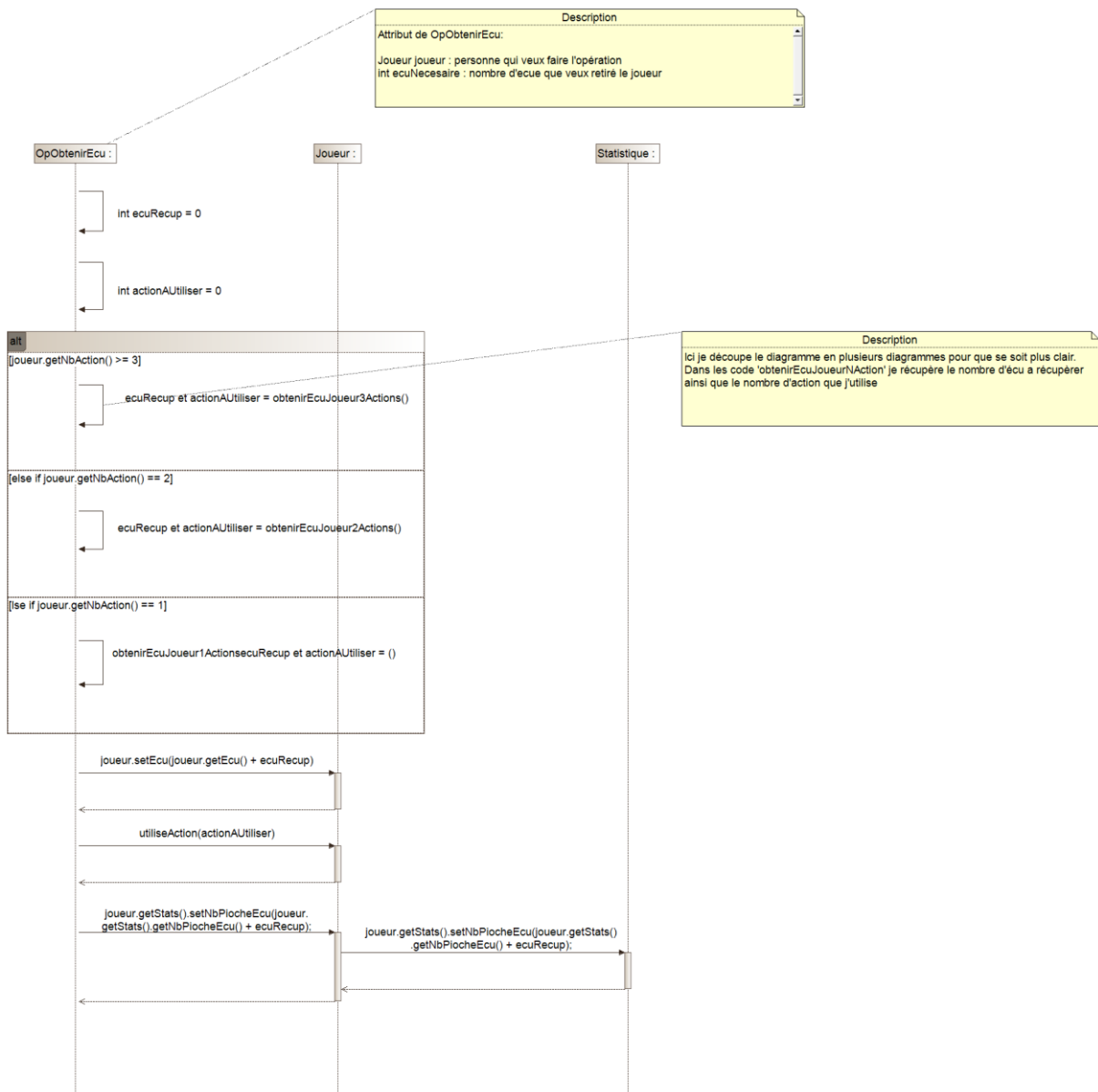
nbActionNecessaire

Cette méthode me permet de connaître le nombre d'action nécessaire lorsque j'affecte un joueur à un bâtiment. Par exemple, si un joueur attribue deux fois un ouvrier à un même bâtiment lors du même tour, il payera 3 actions au lieu de deux.

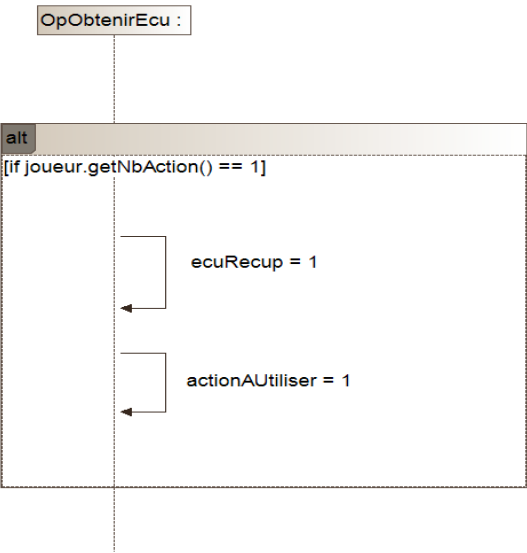


obtenirEcu

Ici nous décrivons l'obtention d'écus par le joueur. Les 3 diagrammes suivants décrivent plus en détail l'algorithme.

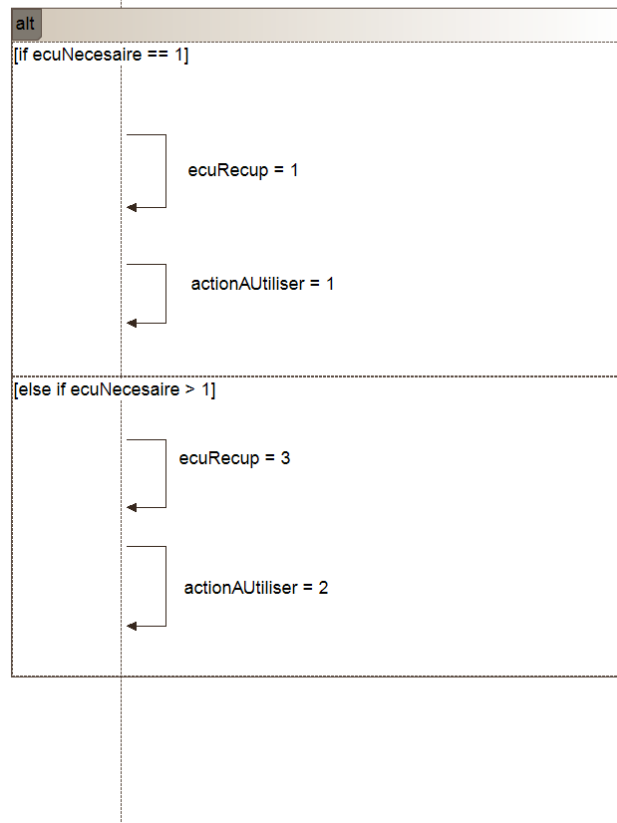


obtenirEcuJoueur1Actions



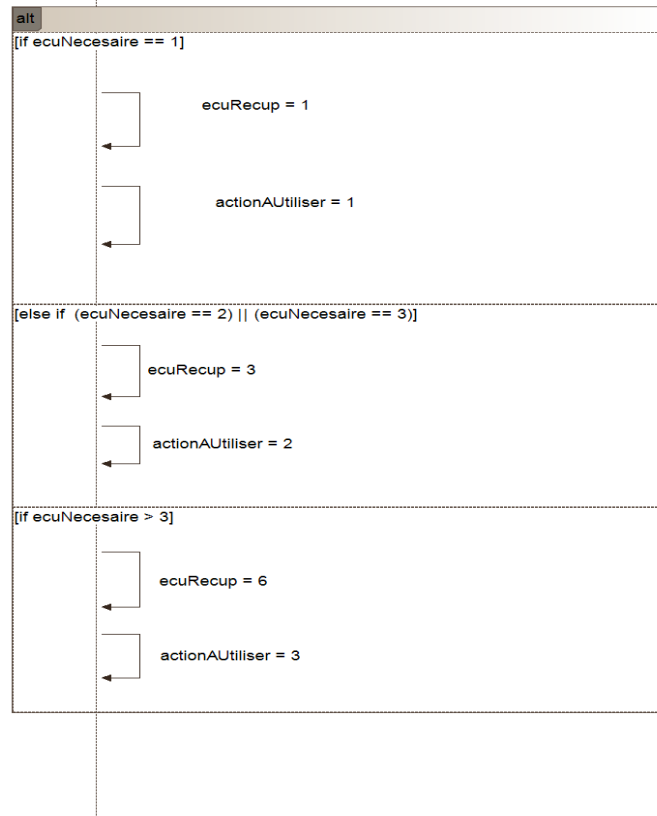
obtenirEcuJoueur2Actions

OpObtenirEcu :



obtenirEcuJoueur3Actions

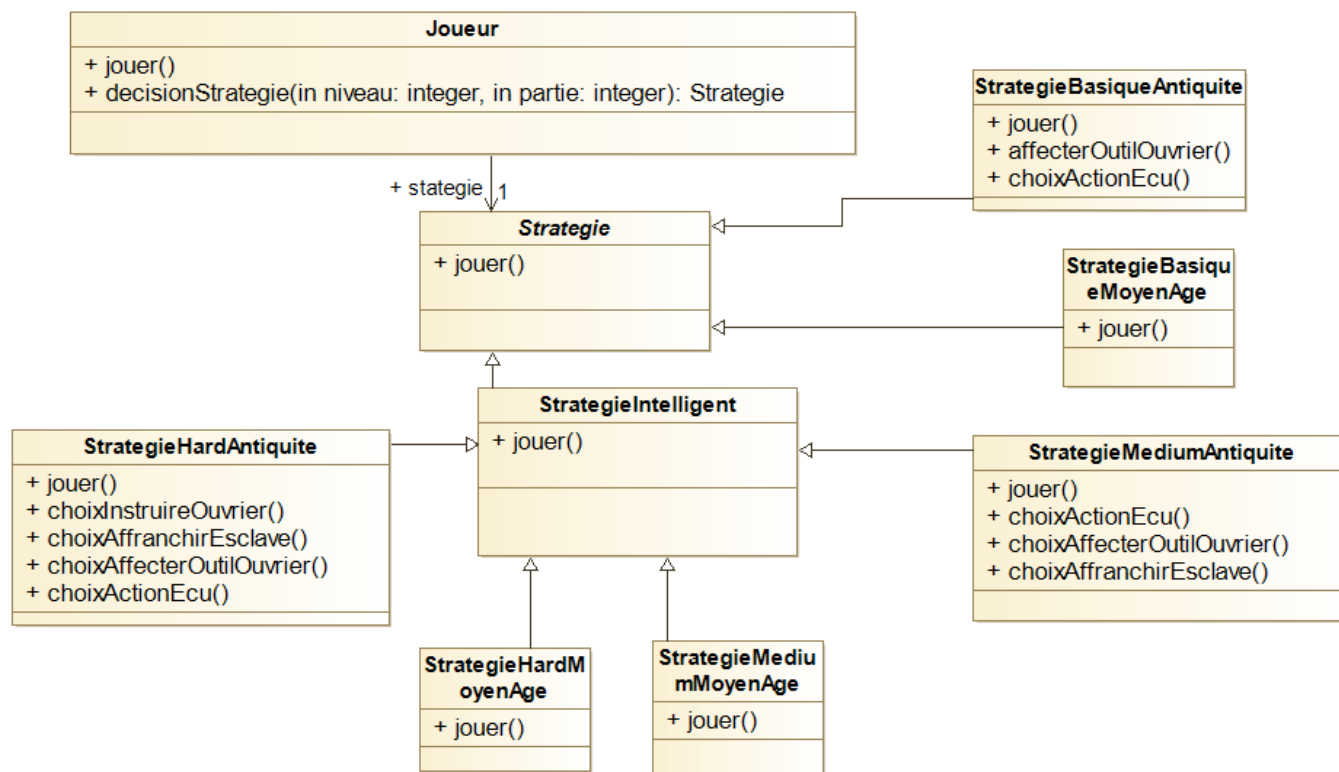
OpObtenirEcu :



Prise en compte de l'Evolution

Evolution prévue

Diagramme de Classe



Liste des évolutions prévues

Première partie de l'évolution A

Le commanditaire a du mal à savoir qui joue et se perd dans la temporalité. De ce fait, il nous a demandé de mettre des couleurs pour chaque bot avec un fond blanc. Nous prévoyons de faire :

- Ajout d'un logger pour pouvoir afficher la traçabilité des parties
- Le logger permettra également de créer des fichiers de logs
- Ajout de couleur pour une meilleure visibilité de la traçabilité du jeu

Seconde partie de l'évolution A

Le commanditaire souhaite installer un nouveau mode de jeu : Les bâtisseurs de l'antiquité. Nous avons prévus :

- Le terme « écu » va être remplacé par « sesterces »
- Le terme « Tuile » sera également remplacé par « décoration »
- Ajout d'une première nouvelle carte université
- Cette nouvelle carte université permettra d'instruire un ouvrier permettant d'augmenter le nombre de ressource apporté par l'ouvrier durant l'ensemble de la partie. Cette carte devra être attribuée sur un ouvrier qui n'est pas en chantier.
- Ajout d'une deuxième nouvelle carte emprunt :

- L'intérêt de cette nouvelle carte emprunt sera d'obtenir beaucoup de Sesterces grâce à une action, le remboursement de l'emprunt sera gratuit, cependant 2 points de victoire vous seront retirés à la fin de la partie
- Grâce à cette évolution il sera possible de réaliser des investissements, de ce fait une troisième et une quatrième carte vont être créées.
- La troisième nouvelle carte se nommera esclave :
- Cette carte esclave sera considérée comme un ouvrier, sa particularité sera qu'il nécessite aucun sesterce. Cet esclave devra être affranchi pour bénéficier de la quatrième carte outils
- Ajout d'une quatrième nouvelle carte outils :
- Un outil permet de déduire ses ressources aux ressources nécessaires pour terminer le chantier d'un bâtiment. Cet outil devra être attribué à un ouvrier
- Ajout d'une nouvelle classe abstraite Stratégie qui regroupera une classe Stratégie Moyen-Age et une Stratégie Antiquité
- La partie sera gérée d'une manière différente car la désignation du gagnant sera différente selon les stratégies moyen-âge et antiquité (gestion des esclaves, des emprunts, etc....)

Evolution B

Le commanditaire souhaite pouvoir affronter ses amis. De ce fait, il aimerait avoir un moyen d'avoir une application en format client – serveur. Pour cela, on a prévu :

- Un découpage en plusieurs modules va être effectué
- Le premier module concernera le client incluant le joueur et les stratégies
- Le deuxième module concernera le serveur incluant la partie
- Le troisième module sera un module commun incluant les cartes et les opérations
- Le Serveur lancera la partie, dira qui joue et validera les opérations des joueurs
- Le Client rejoindra la partie, et enverra au serveur les opérations qu'il a choisies

L'organisation de ses 4 dernières itérations

Par rapport à l'organisation des 4 dernières itérations, nous allons nous organiser de la manière suivante :

Pour l'itération 6, nous allons ajouter les arguments au main afin de lancer N parties, en plus d'ajouter plusieurs exec. Nous allons ajouter des statistiques globales afin de récupérer des informations sur plusieurs parties. De plus, nous allons utiliser des stratégies au lieu de bot, et allons modifier la stratégie basique. Enfin, nous allons commencer à mettre en place les classes des cartes du nouveau mode de jeu en plus des cartes concrètes dans la factory. La classe partie sera aussi modifiée pour prendre en compte ces changements.

Pour l'itération 7, nous allons ajouter des loggers, en plus de couleur pour l'affichage sur console. De plus, nous allons modifier le nommage en fonction du mode de jeu. Par exemple, pour l'argent ce sera écu ou sesterce. Nous allons aussi ajouter une stratégie par difficulté pour le mode de jeu de l'antiquité. La modification du bot moyen pour le mode « Moyen âge » est aussi prévue lors de ce sprint. Enfin, nous allons modifier la partie pour prendre en compte les esclaves et les emprunts.

Lors de l'itération 8, nous allons commencer à scinder le projet en client et en serveur en plus de créer un projet multi module. Il y aura le module client, le module serveur et le module commun. Nous allons créer une classe réseau Client et une classe réseau Serveur. En fonction de l'avancement des stratégies, soit nous continuons la stratégie basique et moyen, soit nous commençons les stratégies difficiles.

Enfin lors de l'itération 9, nous allons globalement poursuivre l'itération 8 et faire le test unitaire du client et du serveur.

Conclusion

Pour les points forts, notre solution peut être facilement maintenable et évoluable en cas de besoin grâce à une bonne organisation du sujet. De plus les principes Solid et Grasp sont globalement respectés ce qui nous confère un certain confort de codage. L'équipe de développement est travailleuse et possède une bonne cohésion ce qui confère une bonne organisation.

Pour les points faibles, il y a un bug qui empêche le dépassement des 8 Tours lors d'une partie. Ce problème est principalement provoqué par les bots qui prennent trop de cartes. De ce fait, on en arrive vite à cours et nous n'avons pas encore géré cette éventualité. De plus, la plupart des bots ne sont pas complètement implémentés mais nous sommes déjà en train d'effectuer des mises à jour pour résoudre ce problème en plus de futures évolutions.

Glossaire

C

commanditaire	
celui qui a commandé le logiciel.....	3