

UE Projet

Groupe 6

Ismail CHAF-I

Lounès MEDDAHI

Matthieu MEDENG ESSIA

Yanis ARAB



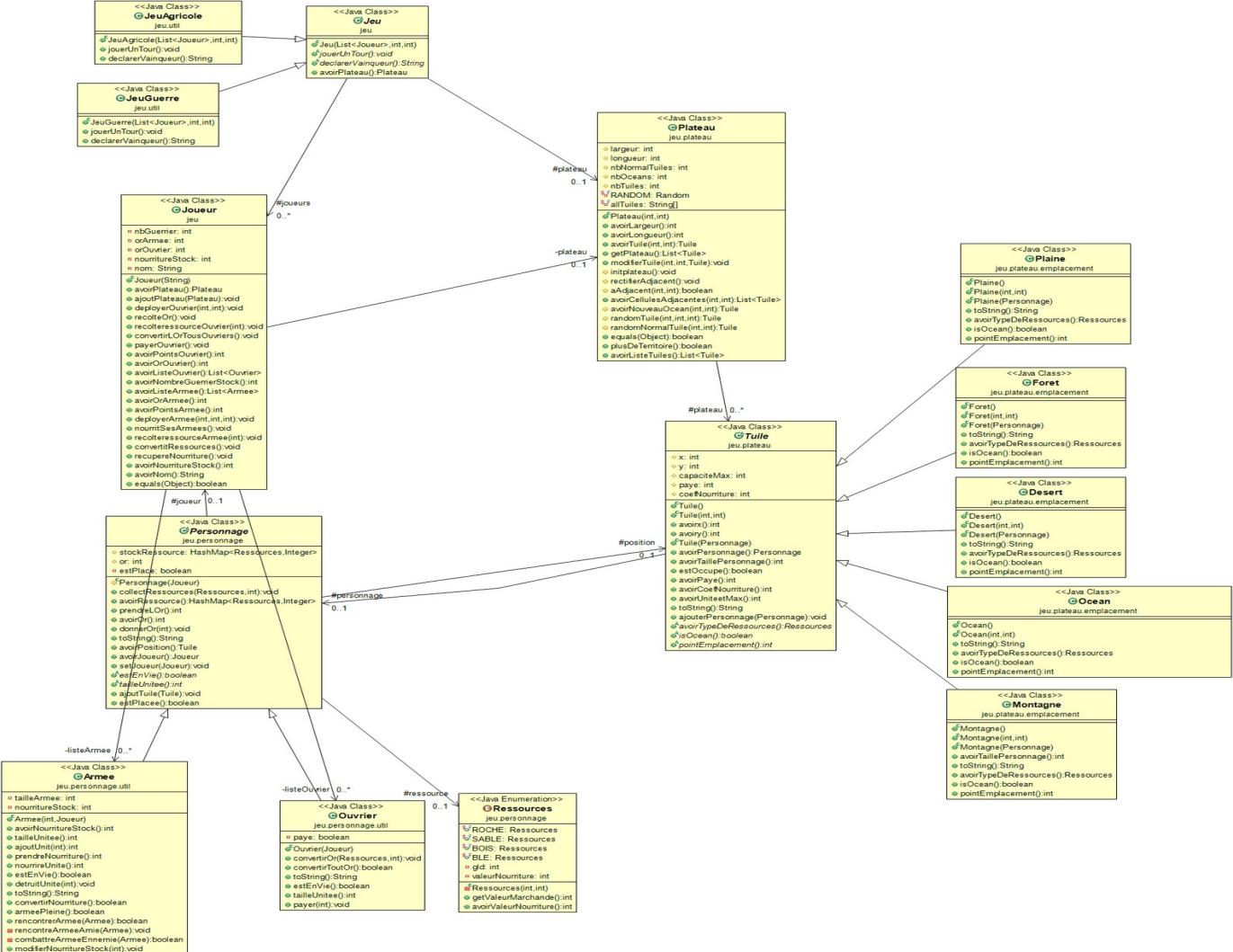
Modélisation & état d'avancement :

1. Modélisation des personnages

2. Modélisation du plateau

3. Modélisation des actions

4. Modélisation complète



Modélisation des personnages :

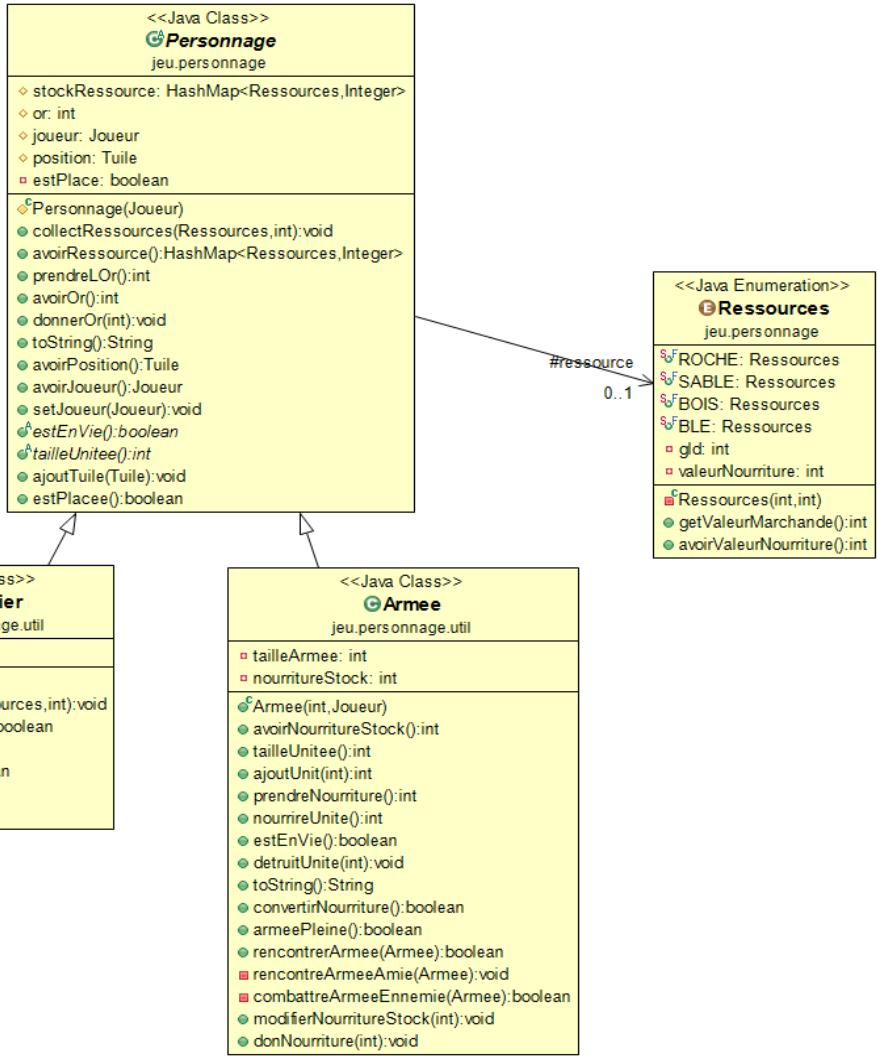
1. Mise en commun :

Qu'est ce qui est commun aux deux?

2. Ce qui serait utile :

De quoi a-t-on besoin?

Dans quel but ?

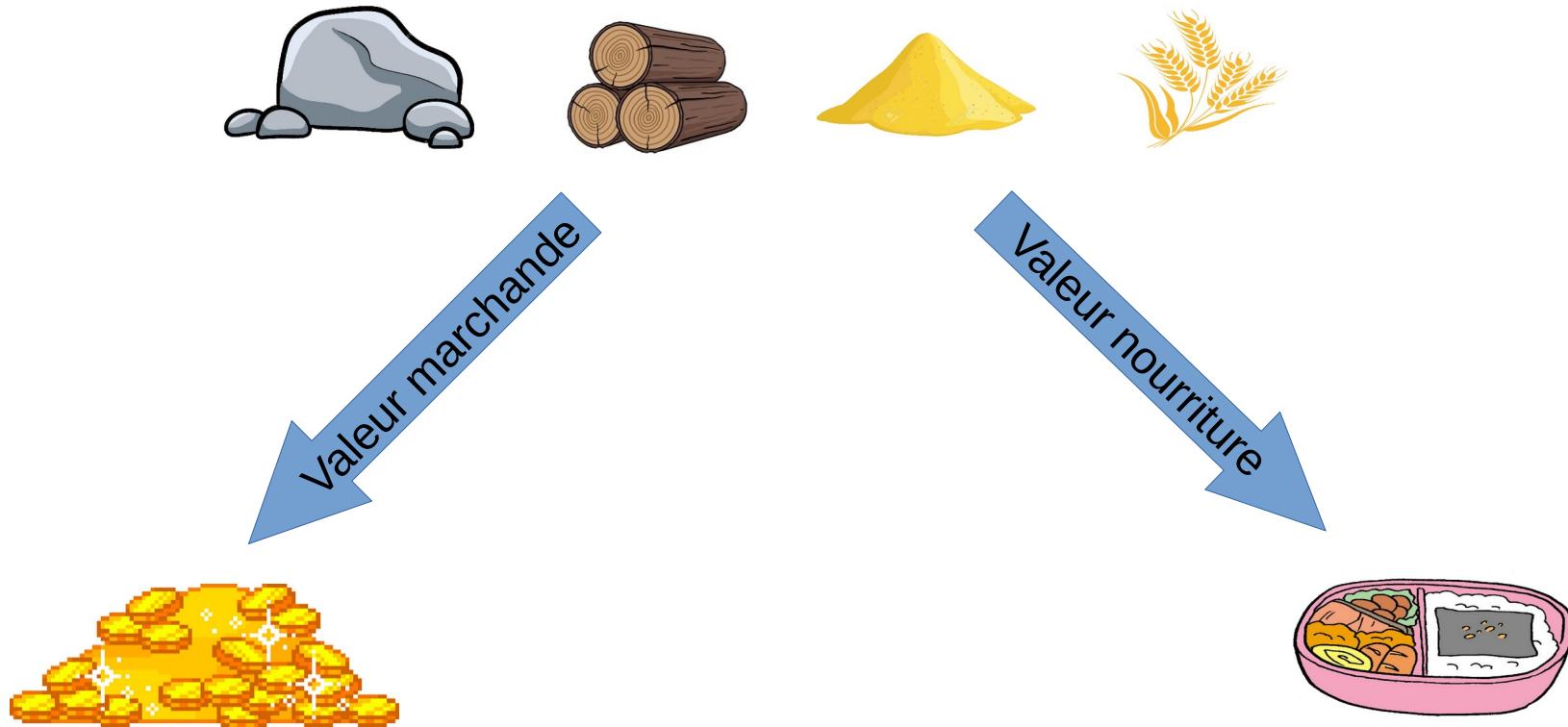


Nos personnages :

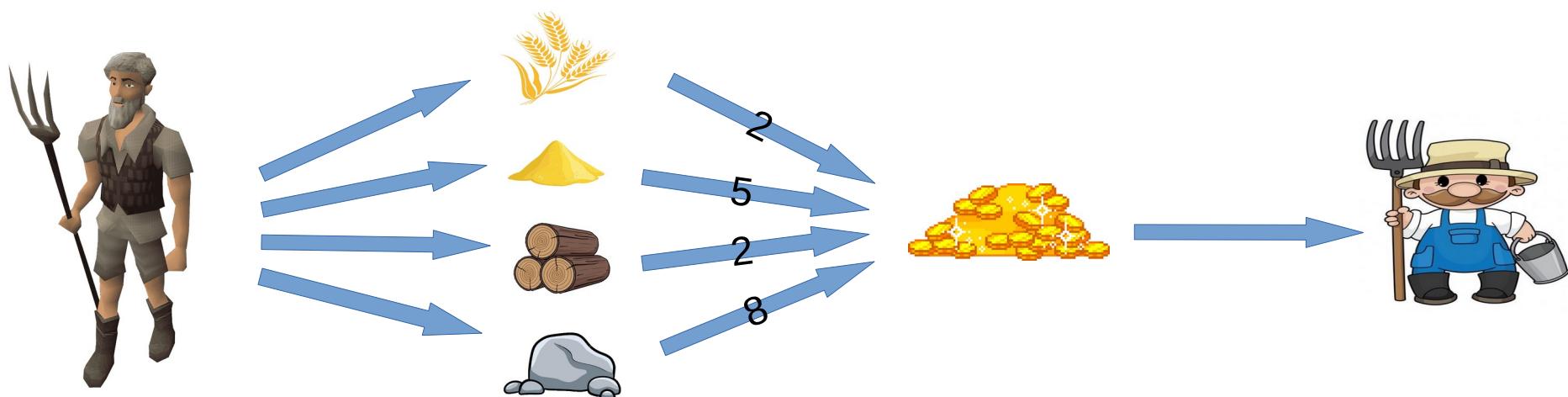
L'or	Les ressources	Emplacement	Autres
Le Stocker	Les récolter	Déployer	Est en vie (taille)
L'utiliser	Les convertir	Vérifier	Propriétaire



Les Ressources :



Les ouvriers :



Ouvrier (joueur)	collectRessources(Ressources, quantité)	ressource	convertirToutOr()	or	Recuperer	Payer (salaire)
Prête pour être déployée	Vont s'ajouter au stock actuel de ressource		1)on a des ressources: vont s'ajouter au stock d'or 2)Si on a rien: rien ne va se passer		Le joueur va tout récupérer	1)On nourrit nos ouvriers 2)Ils arrêtent de travailler

Exemple :

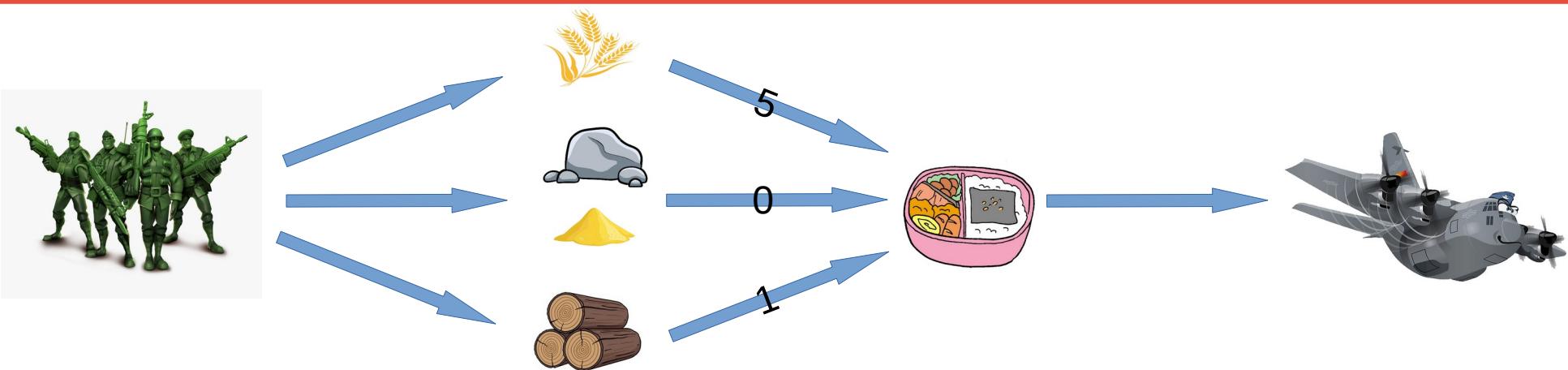
```
16 public class CameraMain{
17     public static void main(String[] args) throws Exception{
18         Plateau plateauTest = new Plateau(1,1);
19         plateauTest.modifierTuile(0, 0, new Plaine());
20
21         Joueur propriétaire = new Joueur("Proprio");
22         propriétaire.ajoutPlateau(plateauTest);
23
24         propriétaire.deployerOuvrier(0, 0);
25
26         System.out.println("on va demander à notre ouvrier de recolter 4 blés,");
27         propriétaire.recolteressourceOuvrier(4);
28
29         System.out.println("puis de les convertires en or et de nous les donner");
30         propriétaire.convertirLOrTousOuvriers();
31         propriétaire.recolteOr();
32         System.out.println("notre "+propriétaire.avoirNom()+" a désormais "+propriétaire.avoirOrOuvrier()+" or, mais va en utiliser 1 pour payer l'ouvrier");
33         propriétaire.payerOuvrier();
34         System.out.print("le proprio a desormais plus que "+propriétaire.avoirOrOuvrier()+" or");
35     }
36 }
```

The screenshot shows a Java application window. At the top, the code for CameraMain.java is displayed, showing a main method that creates a Plateau, adds a plain tile at position (0,0), adds a player named "Proprio", deploys an worker at (0,0), and then performs a series of actions involving harvesting wheat and converting it to gold, while printing messages to the console.

In the bottom panel, the "Console" tab is active, showing the output of the program:

```
<terminated> CameraMain [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe (8 mai 2021 à 10:56:07 – 10:56:08)
on va demander à notre ouvrier de recolter 4 blés,
Notre ouvrier sur la tuile (0,0) va récolter 4 ressources
puis de les convertires en or et de nous les donner
notre Proprio a désormais 23 or, mais va en utiliser 1 pour payer l'ouvrier
le proprio a desormais plus que 22 or
```

Les armées :



Armee(Taille, joueur)	collectRessour ces(Ressources, quantité)	ressource	convertirNourriture ()	nourr iture Stock	Recup erer	nourrireUnité()
Prête pour être déployée	Vont s'ajouter au stock actuel de ressource		1)on a des ressources: vont s'ajouter au stock 2)Si on a rien: rien ne va se passer		Le joueur va tout récupérer	1)On nourrit nos armée 2)On détruit nos unités

Exemple :

```
16 public class CameraMain{
17     public static void main(String[] args) throws Exception{
18         Plateau plateauTest = new Plateau(1,1);
19         plateauTest.modifierTuile(0, 0, new Plaine());
20
21         Joueur proprietaire = new Joueur("France");
22         proprietaire.ajoutPlateau(plateauTest);
23
24         proprietaire.deployerArmee(0, 0, 4);
25
26         System.out.println("on va demander à l'armée de recolter 4 blés:");
27         proprietaire.recolteressourceArmee(4);
28
29         System.out.println("puis de les convertires en nourriture et de les donner:");
30         proprietaire.convertitRessources();
31         proprietaire.recupereNourriture();
32         System.out.println("notre "+proprietaire.venirNom()+" a désormais "+proprietaire.venirNourritureStock()+" nourriture");
33         proprietaire.nourritSesArmees();
34         System.out.print("le proprio a désormais plus que "+proprietaire.venirNourritureStock()+" nourritures en stock ");
35
36     }
37 }
```

Console

```
<terminated> CameraMain [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe (8 mai 2021 à 11:44:28 – 11:44:29)
on va demander à l'armée de recolter 4 blés:
Le joueur France va récolter 4 ressources de type Plaine
puis de les convertires en nourriture et de les donner:
on va convertir 4 ressources en nourriture
On a récupéré 20 unité de nourriture
Ce qui nous fait un stock de 30 nourritures
notre France a désormais 30 nourriture en stock, mais va en utiliser 4 pour nourrir l'armée
le proprio a désormais plus que 26 nourritures en stock
```

Tests des personnages :

Les tests passent tous avec succès

Finished after 0,038 seconds

Runs: 26/26 Errors: 0 Failures: 0

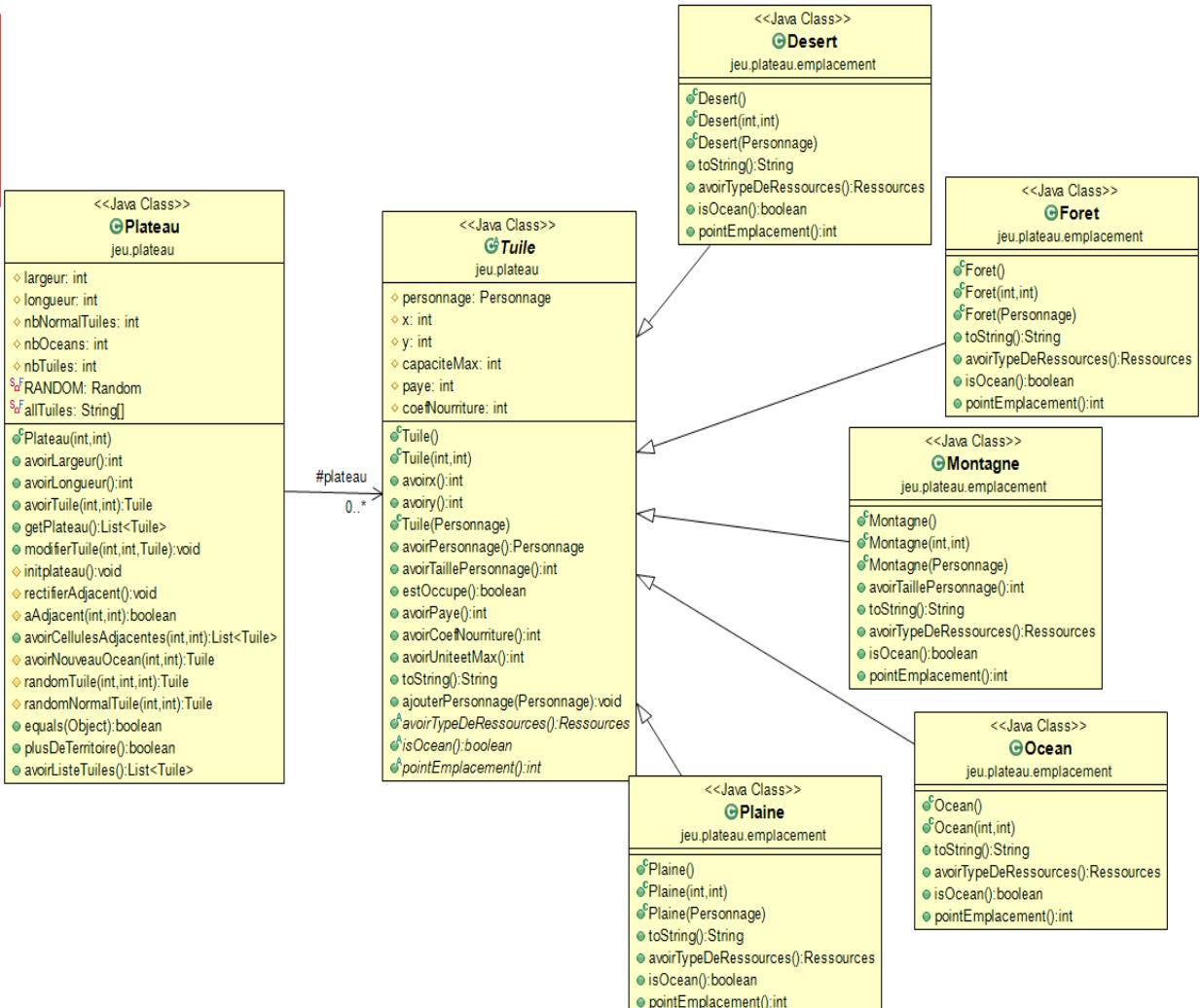
<p><<Java Class>></p> <p>ArmeeTest</p> <p>jeu.personnage.util</p>
<ul style="list-style-type: none">▫ j1: Joueur▫ j2: Joueur▫ t1: Tuile▫ t2: Tuile▫ t3: Tuile▫ armee1: Armee▫ armee2: Armee▫ armee3: Armee▫ armee4: Armee▫ armee5: Armee

<p><<Java Class>></p> <p>OuvrierTest</p> <p>jeu.personnage.util</p>
<ul style="list-style-type: none">▫ j: Joueur▫ t: Tuile▫ ouvrier: Ouvrier <p>OuvrierTest()</p> <ul style="list-style-type: none">● before():void● testOuvrierSurOcean():void● donneesCorrectementrees():void● orCorrectementConvertiSable():void● orCorrectementConvertiBois():void● orCorrectementConvertiBois():void● orCorrectementConvertiRoche():void● exceptionCorrectementDeclenchee():void● testgiveOr():void● testgetPosition():void● testgetJoueur():void● testPasPayerUnOuvrier():void● testRemplacerUnOuvrierPasPaye():void● testRecolterDesRessourcesAPartirDeSaTuile():void● testDePrendreToutLOr():void

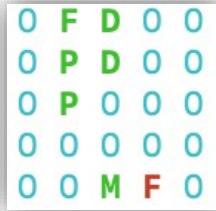
Modélisation du plateau :

1. Mise en commun

2. Ce qui serait utile :
*De quoi a-t-on
besoin?
*Dans quel but ?

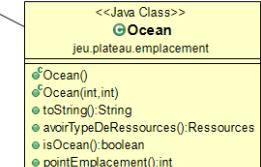
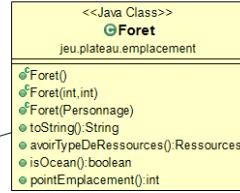
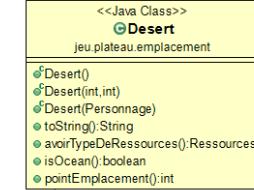
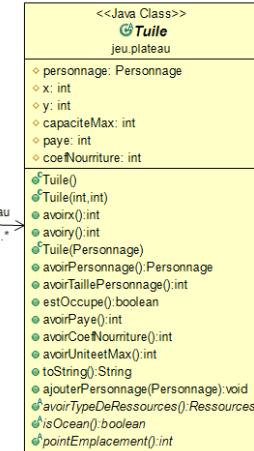


Modélisation du plateau

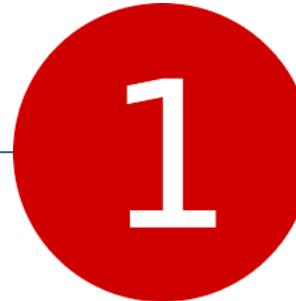
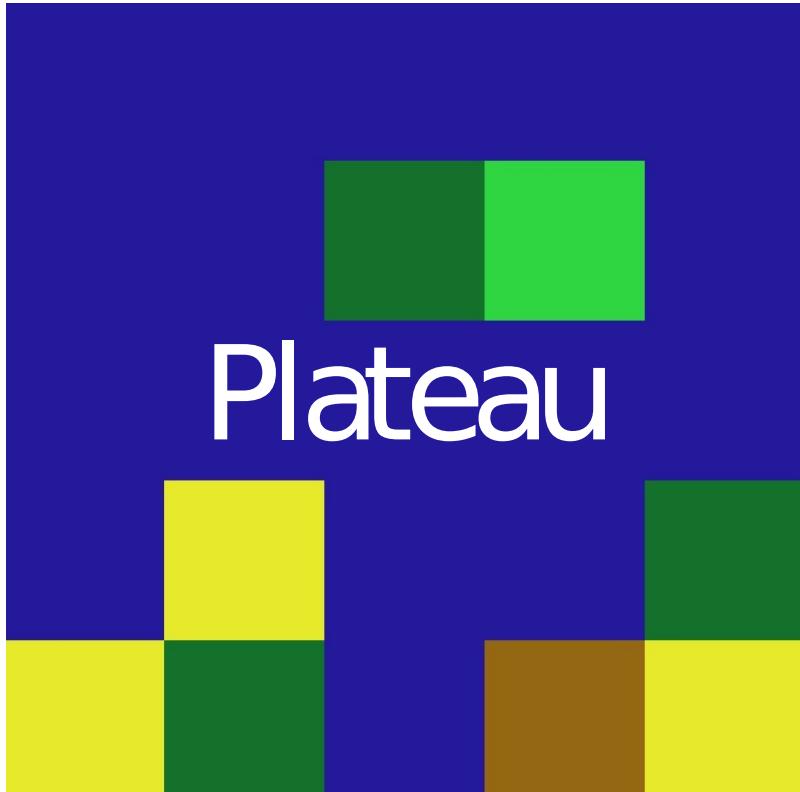


#plateau

0 .. *



Plateau



le plateau
doit
comporter
au minimum
deux tiers
de tuiles de
type océan .



toutes les tuiles de
type montagne,
plaine, désert ou
forêt doivent au
moins avoir une tuile
adjacente qui n'est
pas de type océan.



1

✓ L'initialisation de plusieurs variables :

* *minocean == nbr minimum de tuiles océans dans le tableau initialisé .*

* *nbocean == multiplie par un nbr random entre 2/3 et 1 et ajouté le nbr minOceans .*

* *nbtuile == longueur * largeur .*

* *nbNormalTuiles == nbr de tuiles hors ocean .*

✓ Méthodes :

* *avoirNouveauOcean ,*

* *initplateau,*

* *avoirNouveauOcean ,*

initplateau

1. On commence par initialiser que 'laterre do it ère terre' par false et mon random par 0
2. On parcours longueur et largeur
 1. Si on trouve que 'laterre do it ère terre' == true j 'appelle la méthode randomTuile(0, x, y) avec un paramètre 0 car le rôle de cette méthode :
 1. Si n!=0. On utilise la méthode getNouveauOcean Qui attribue a la tuile un océan tout en vérifions la condition qu'on dispose toujours de tuile océan
 2. Si n=0 la position this.x et this.y devient une tuille normal(foret, montage....) a l'aide la fonction randomNormalTuile(xPosition, yPosition);
 1. Qui attribue au hasard random soit allTuiles =
{"Foret","Desert","Montagne","Plaine"};
On redonne a notre Booléen false
 - 3.
2. Si je trouve que 'laterre do it ère terre' == fasle dans se cas il faut distinguer les cas exemple
 1. Cas num 1: les dernier tuiles de lignes du plateau doivent êtres des ocean s comme ça on a pas pas le problème avec la première tuile de la ligne suivante (adjacente)
 2. Cas num 2: si c'est pas dans la dernier tuiles de la lignes on n utilise un Radom pour avoir (soit terre soit océan) 1. Si c'est terre on appelle la fonction hasAdjacent(x, y) qui étudie tout les cas possible (les alentours de la tuile étudié) si elle a un adjacent bah on fait la négation pour que 'laterre do it ère terre'== false 2. Sinon 'laterre do it ère terre'== false

2

✓ Méthodes :

* avoirCellulesAdjacentes ,

*aAdjacent,

*rectifierAdjacent,

Tuile



1. Choix d'une classe mère abstraite



- facilite et organise codage.
- méthodes communes.
`(public abstract Ressources avoirTypeDeRessources();)`
- Optimisation.

2. Constructeurs

- `Tuile()`
- `Tuile(int x, int y)`
(x,y coordonnées)

3. méthodes

abstraites

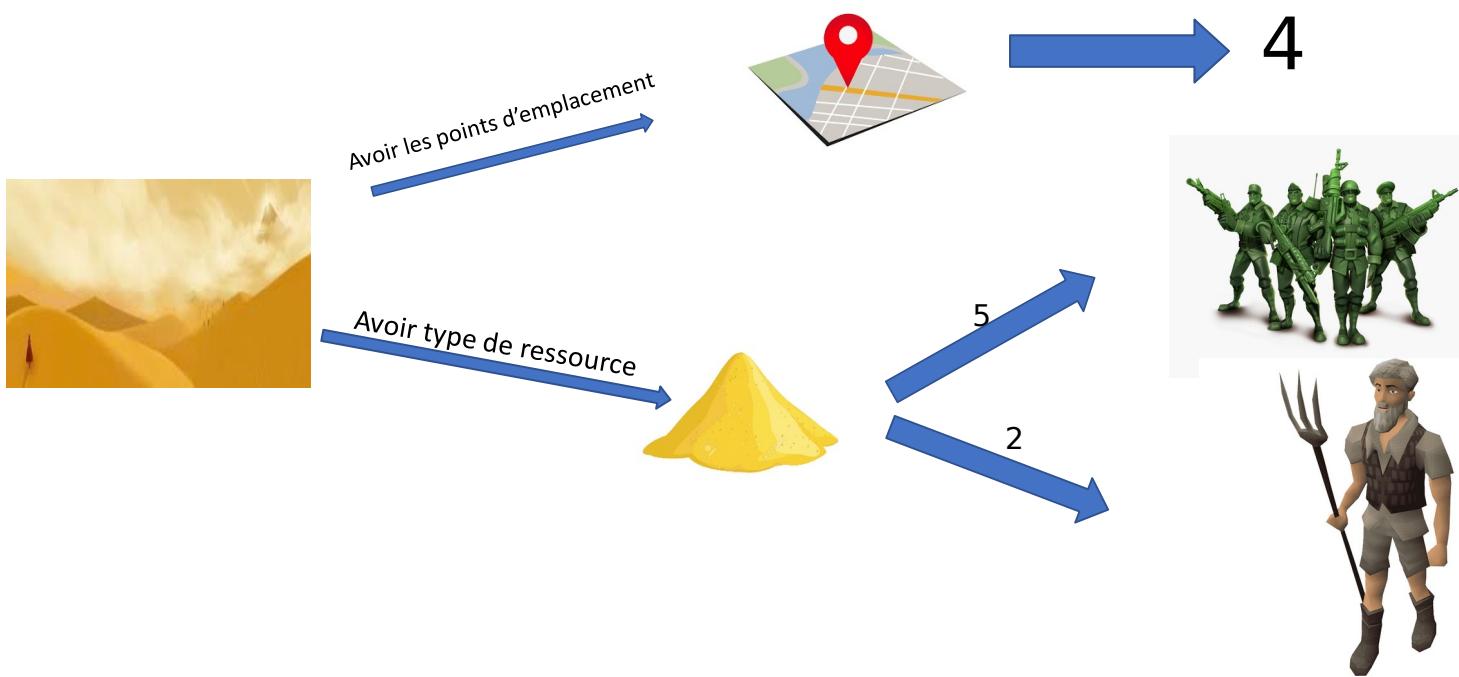
concrètes

TUILÉ



- ✓ **Appartiennent au même package :** jeu.plateau.emplacement
- ✓ **même méthodes :** `toString()` , `avoirTypeDeRessources()`, `isOcean()`, `pointEmplacement()...`
- ✓ **Importe de plusieurs class :** jeu.personnage.Personnage , jeu.personnage.Ressources , jeu.plateau.Tuile .

Désert



Fôret

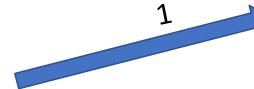


Avoir les points d'emplacement



2

Avoir type de ressource



Océan



Avoir les points d'emplacement



0

Avoir type de ressource

null



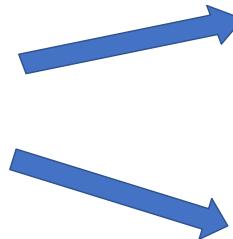
Montagne



Avoir les points d'emplacement



Avoir type de ressource



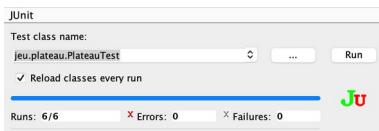
4



Tests du plateau:

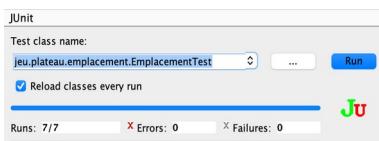
- **Les tests de la classe Plateau fonctionnent :**

(make jeu.plateau.PlateauTest)



- **Les tests Emplacement fonctionnent :**

(make jeu.plateau.emplacement.EmplacementTest)



```
<<Java Class>>
jeu.plateau
GPlateauTest
jeu.plateau

p0: Plateau
p1: Plateau
p2: Plateau
p3: Plateau
p4: Plateau
p5: Plateau
p6: Plateau
p7: Plateau
p8: Plateau
p9: Plateau
p10: Plateau
p11: Plateau
plateaux: ArrayList<Plateau>

FPlateauTest()
before():void
testAvoirLargeur():void
testAvoirLongueur():void
testInitPlateau():void
testAdjacente():void
testRectifierAdjacente():void
testAvoirCellulesAdjacents():void
suite():Test
```

```
<<Java Class>>
jeu.plateau.emplacement
GEmplacementTest
jeu.plateau.emplacement

montagne: Tuile
foret: Tuile
desert: Tuile
plaine: Tuile
ocean: Tuile
j: Joueur
perso1: Personnage
perso2: Personnage

EmplacementTest()
before():void
testgetPersonnage():void
testestOccupe():void
testgetSoldatMax():void
testAjouterDeuxArmeesSurUneTuile():void
testDeChangerUnMort():void
testIsOcean():void
testPointsEmplacement():void
```

Modélisation des actions :

Que souhaitons nous pouvoir faire ?

	Armee	Ouvrier
Points:	°)Or °)Territoires	°)Or
Déploiement	°)2 Conditions	°)1 Condition
Points importants:		
Récompense	°)Nourriture	°)Salaire
Autres	°)Combat	

<<Java Class>> Joueur jeu
<ul style="list-style-type: none"> ▫ listeArmee: List<Armee> ▫ listeOuvrier: List<Ouvrier> ▫ nbGuerrier: int ▫ orArmee: int ▫ orOuvrier: int ▫ nourritureStock: int ▫ nom: String ▫ plateau: Plateau <ul style="list-style-type: none"> ● Joueur(String) ● avoirPlateau():Plateau ● ajoutPlateau(Plateau):void ● deployerOuvrier(int,int):void ● recolteOr():void ● recolteressourceOuvrier(int):void ● convertirLOrTousOuvriers():void ● payerOuvrier():void ● avoirPointsOuvrier():int ● avoirOrOuvrier():int ● avoirListeOuvrier():List<Ouvrier> ● avoirNombreGuerrierStock():int ● avoirListeArmee():List<Armee> ● avoirOrArmee():int ● avoirPointsArmee():int ● deployerArmee(int,int,int):void ● nourritSesArmees():void ● recolteresourceArmee(int):void ● convertitRessources():void ● recupererNourriture():void ● avoirNourritureStock():int ● avoirNom():String ● equals(Object):boolean

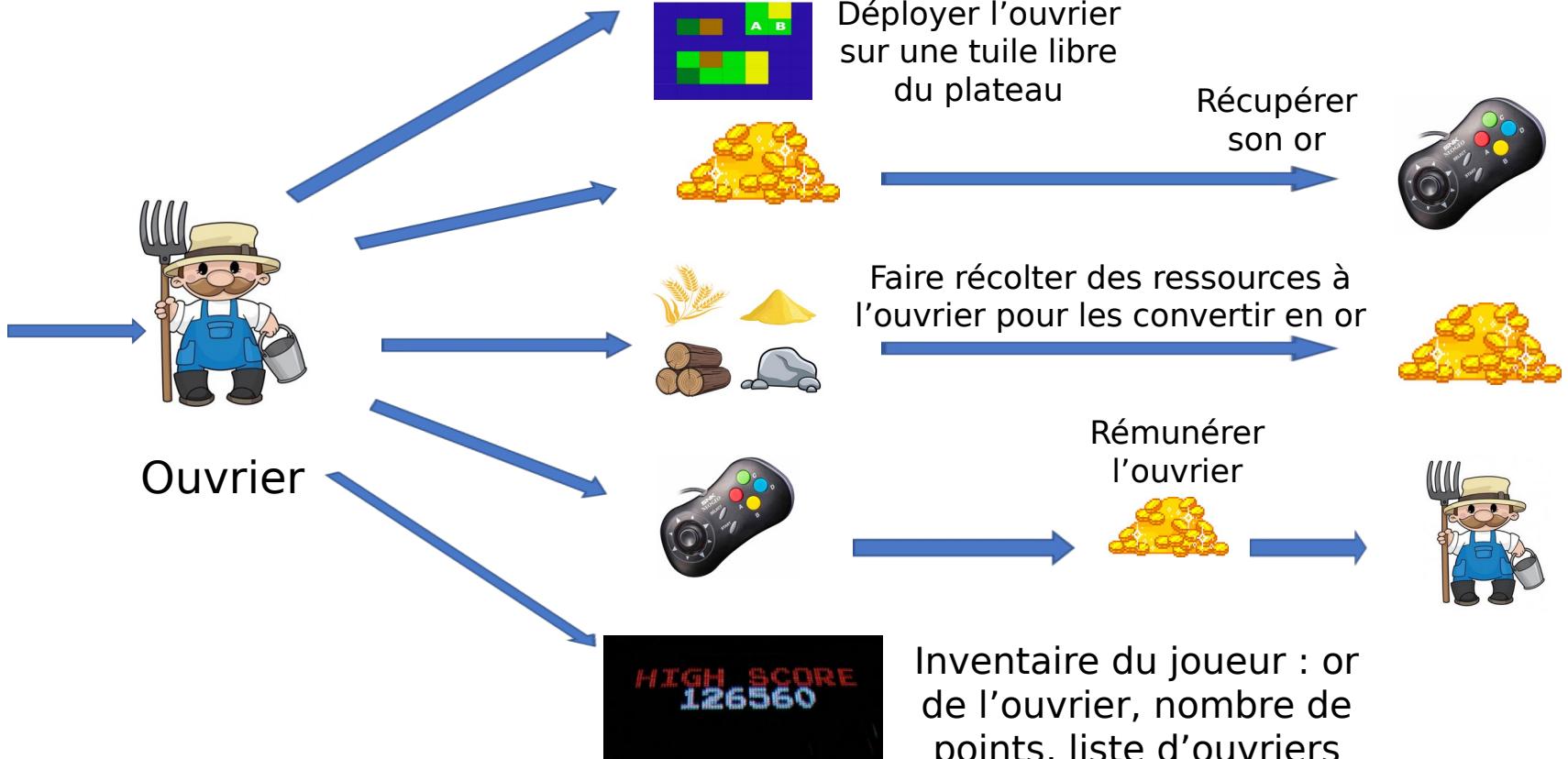
Actions du joueur sur l'ouvrier :



Joueur



Ouvrier

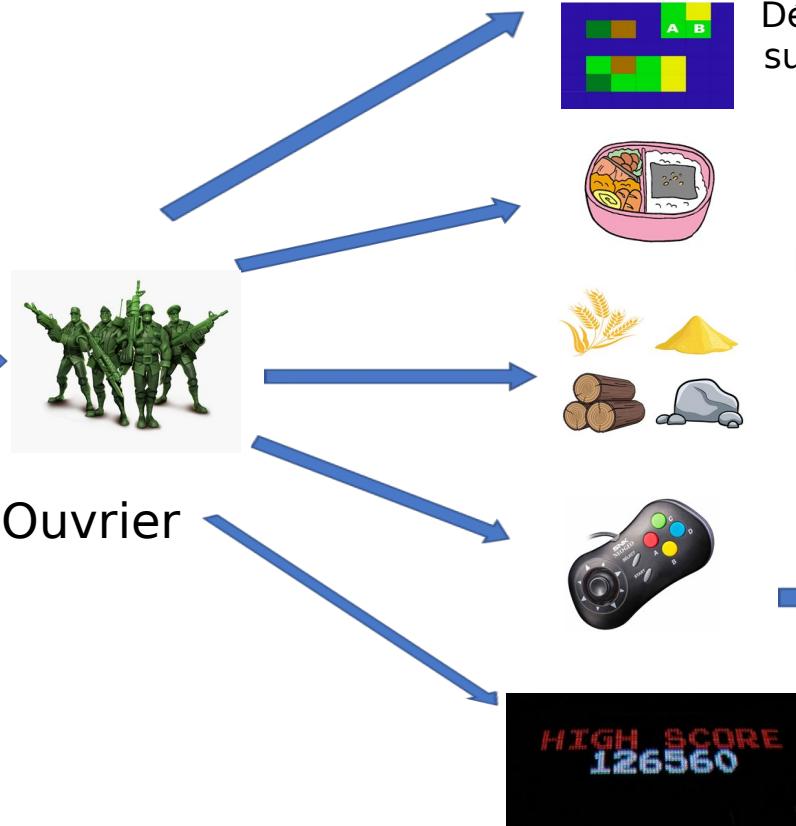


Actions du joueur sur l'armée:



Joueur

Ouvrier



Déployer l'ouvrier
sur une tuile libre
du plateau

Récupérer
ses vivres

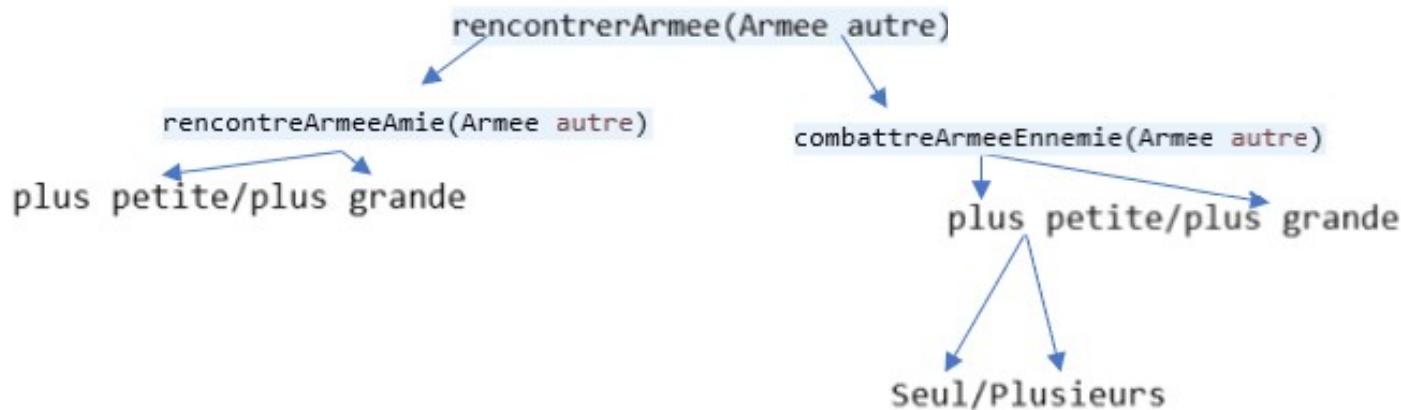
Faire récolter des ressources à
l'ouvrier pour les convertir en
nourriture

Nourrir
l'armée



Inventaire du joueur : or
de l'ouvrier, nombre de
points, liste d'ouvriers

Combat entre armées :



Pas de problème

- Personne dans les alentours:
->Rien

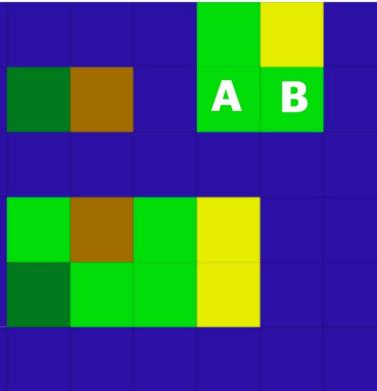
- Allié:
-On est plus grand -> bonus
-Sinon -> Rien

Problème (ennemie proche)

- On est moins nombreux:
->On fuit

- On est plus nombreux:
-Il est seul->Rallie
-Ils sont plusieurs->Bataille (+bonus)

Exemples :



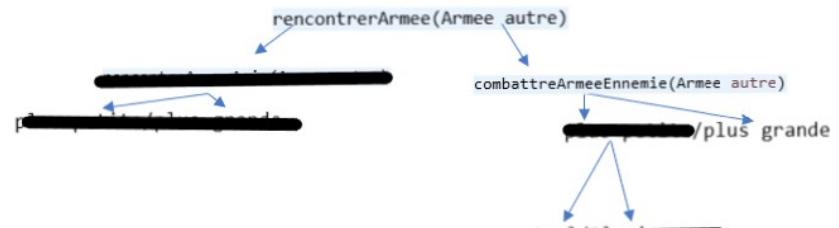
1. Ennemis plus nombreux

B a déployé 2 unités de son armée sur la tuile de coordonnées (0,2) qui est de type Desert
Armée détectée dans les environs.

On envoie des éclaireurs sur le camp ennemi au loin ...

Ils reviennent avec une mauvaise nouvelle.... Ils sont plus que nous(environ 5)...

Restons cachés alors...



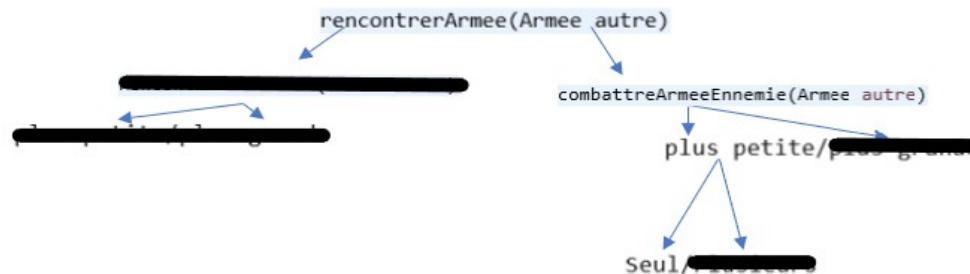
2. Ennemis moins nombreux

B a déployé 3 unités de son armée sur la tuile de coordonnées (6,2) qui est de type Montagne
Armée détectée dans les environs.

ON TOMBE SUR UN ENNEMI !!! L'armée de A qui a 1 Guerrier

On a plus de guerrier que lui !!! à l'attaqueeeeeeee

Il avait qu'un guerrier !! On récupéré son armée et 2 or au passage !!



Tests des actions :

Les tests passent tous avec succès

Finished after 0,049 seconds

Runs: 14/14 Errors: 0 Failures: 0

<<Java Class>>

 JoueurTest

jeu

- joueur: Joueur
- plateau: Plateau
- ouvrier1: Ouvrier

 JoueurTest()

 Before():void

 testAvoirPlateau():void

 testExceptionQuandOnDeploieUnOuvrierSurUneTuileInexistante():void

 testDeployerUnOuvrier():void

 TestRecolterToutOrOuvrier():void

 testDuNombreDePoint():void

 testPayerTousLesOuvriers():void

 testExceptionQuandOnDeploieUneArmeeSurUneTuileInexistante():void

 testExceptionQuandOnDeploieUneArmeeTropGrandeSurUneTuile():void

 testDeploieUneArmeeSurUneTuile():void

 TestRecolterTouteLaNourritureArmee():void

 testDeNourrirNosUnites():void

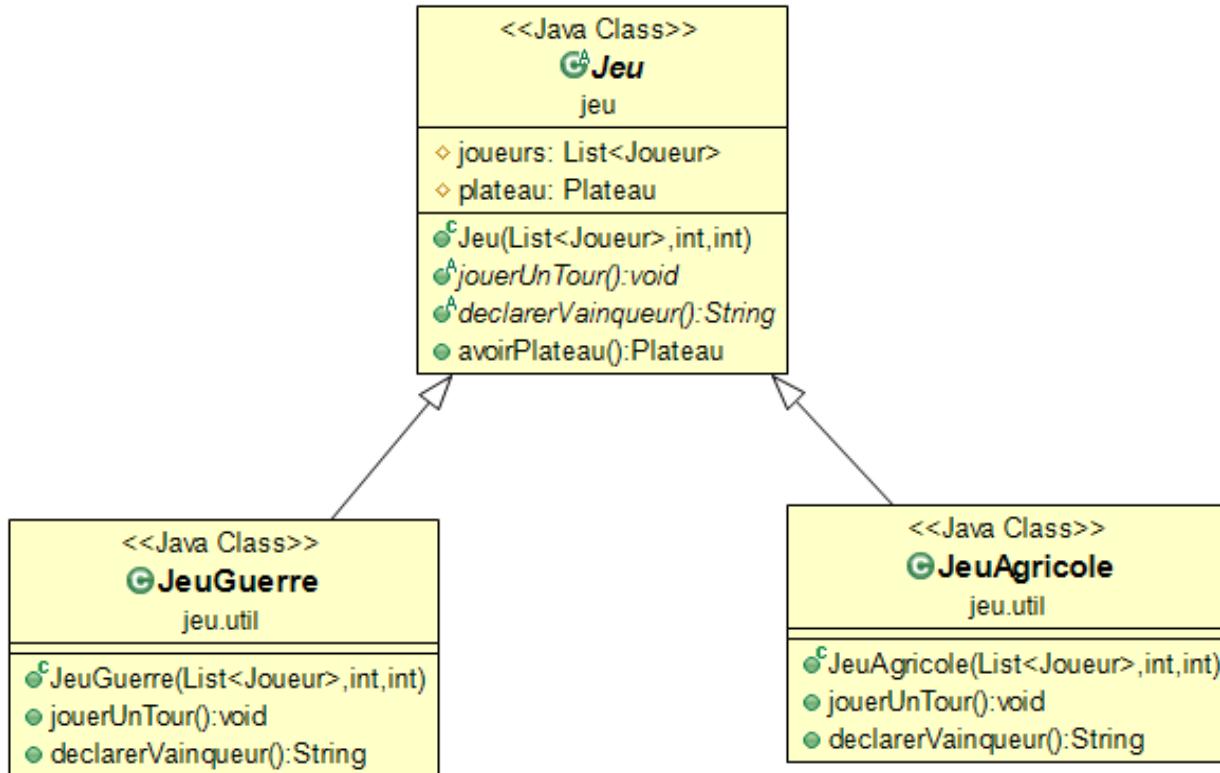
 testBaisseNombreDeGuerrier():void

 testDeDeployerTropArmee():void

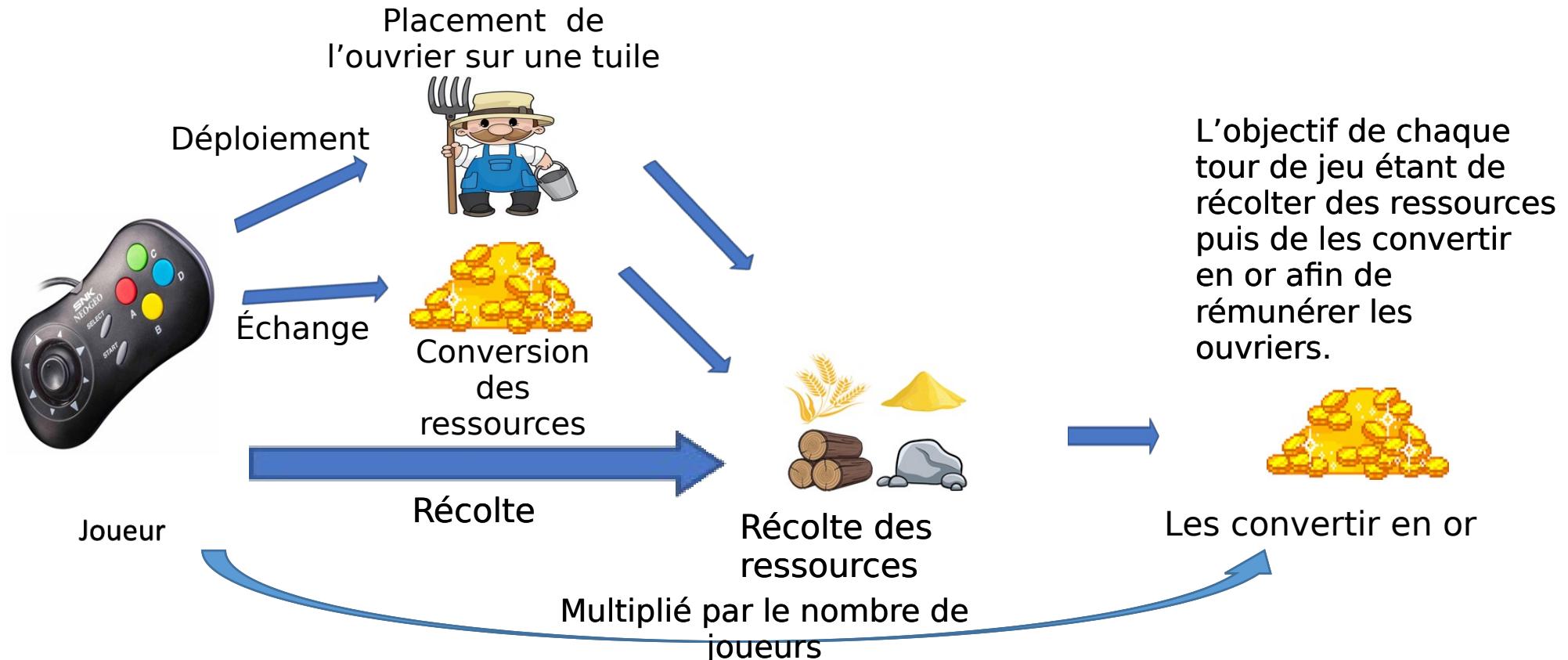
 testCompteDePoints():void

 TestBonusPoints():void

Modélisation complète :



Exploitation agricole modélisée



La théorie : conquête des territoire modélisée

```
public void jouerUnTour(){
    System.out.println("##### Nouveau tour de jeu #####");
    for(Joueur joueur : this.joueurs){
        Scanner sc = new Scanner(System.in);
        System.out.println("-----C'est à " + joueur.getNom() + " de jouer!-----");
        System.out.println("Déployer une armée? Oui/Non");
        String action = sc.nextLine();
        if(action.equals("Oui")){
            System.out.println("Tuiles disponibles :");
            for(Tuile tuile : this.plateau.getPlateauListeTuiles()){
                if(!tuile.isOcean() || tuile.estOccupe() || (tuile.estOccupe() & !tuile.averPersonnage().estEnVie())){
                    System.out.println("Tuile de coordonnées (" + tuile.getAxeX() + "," + tuile.getAxeY() + ") qui est de type " + tuile.toString());
                }
            }
            System.out.println("Choisissez une tuile parmi celles proposées ainsi que le nombre d'unités à déployer.");
            int x = sc.nextInt();
            int y = sc.nextInt();
            int u = sc.nextInt();
            try{
                joueur.deployerArmée(x, y, u);
                System.out.println(joueur.getNom() + " a déployé " + u + " unités de son armée sur la tuile de coordonnées (" + this.plateau.getPlateauTuile(x,y).getAxeX() + "," + this.plateau.getPlateauTuile(x,y).getAxeY() + ") qui est de type " + this.plateau.getPlateauTuile(x,y).toString());
            }
            catch(Exception e){
                System.out.println("Coordonnées ou nombre d'unités invalides.");
            }
            List<Tuile> adjacentes = this.plateau.getPlateauCellulesAdjacentes(x, y);
            for(Tuile tuile : adjacentes){
                Armée armee1 = (Armée) this.plateau.getPlateauTuile(x,y).averPersonnage();
                Armée armee2 = (Armée) tuile.averPersonnage();
                if(tuile.estOccupe()){
                    System.out.println("Armée détectée dans les environs.");
                    boolean reaction;
                    try{
                        reaction = armee1.rencontrerArmée(armee2);
                    }
                    catch(Exception e){
                        System.out.println("Exception levée!");
                    }
                }
            }
        }
    }
}
```

Inventaire du joueur en fin de tour :

```
joueur.recolteressourceArmee(q);
//Elles vont convertir les ressources
joueur.convertitRessources();
joueur.recupereNourriture();
//On va utiliser la nourriture en stock afin de nourrir nos armées
joueur.nourritSesArmees();

System.out.println("#####RÃ©capitulatif du joueur "+joueur.avoirNom()+"#####");
System.out.println("#"+joueur.avoirNom()+" n'a plus que "+joueur.avoirNombreGuerrierStock()+" guerriers prÃ©t Ã  la bataille et un stock de "+
joueur.avoirNourritureStock()+" nourritures en stock #");
System.out.println("#Listes des tuiles possÃ©dÃ©es par le joueur:#");
for(Armee armee : joueur.avoirListeArmee()) {
    System.out.println("#-On a une armÃ©e de taille "+armee.tailleUnitee()+" sur la tuile ("+armee.avoirPosition().avoirX()+" , "+armee.avoirPosition().avoirY()+" )#");
}
System.out.println("#####");

}

public String declarerVainqueur() throws DrawException{
Joueur gagnant = null;
boolean egalite = false;
for(Joueur joueur: this.joueurs) {
    if(gagnant==null) {
        gagnant=joueur;
    }
    else if(joueur.avoirPointsArmee()>=gagnant.avoirPointsArmee()) {
        if(joueur.avoirPointsArmee()==gagnant.avoirPointsArmee()) {
            egalite = true;
        }
        else {
            gagnant = joueur;
        }
    }
}

if(egalite){
    throw new DrawException("match nul!");
}
return gagnant.avoirNom() + " remporte la partie.;
```

La théorie : exploitation agricole modélisée

```
public void jouerUnTour(){
    for(Joueur joueur : this.joueurs){

        Scanner sc = new Scanner(System.in);
        System.out.println("C'est " + joueur.avoirNom() + " de jouer!");
        System.out.println("Deployer un ouvrier, changer des ressources ou attendre? Deployer/Echanger/Attendre");

        String action = sc.nextLine();
        if(action.equals("Deployer")){
            System.out.println("Tuiles disponibles :");
            for(Tuile tuile : this.plateau.avoirListeTuiles()){
                if(!(tuile.isOcean()) || tuile.estOccupe() || (tuile.estOccupe() && !tuile.avoirPersonnage().estEnVie()) ){
                    System.out.println("Tuile de coordonnes (" + tuile.avoirX() + "," + tuile.avoirY() + ") qui est de type "+tuile.toString());
                }
            }
            System.out.println("Choisissez une tuile parmi celles proposes.");
            int x = sc.nextInt();
            int y = sc.nextInt();
            try{
                joueur.deployerOuvrier(x,y);
            }
            catch(Exception e){
                System.out.println("Coordonnes ou ressources indisponibles.");
            }
            System.out.println(joueur.avoirNom() + " a deploy un ouvrier sur la tuile de coordonnes (" + this.plateau.avoirTuile(x,y).avoirX() + "," + this.plateau.avoirTuile(x,y).avoirY() + ") qui est de type "+this.plateau.avoirTuile(x,y).toString());
        }
        if(action.equals("Echanger")){
            joueur.convertirOrTousOuvriers();
            System.out.println(joueur.avoirNom() + " a chang ses ressources contre de l'or.");
        }
        System.out.println("Choisissez la quantit de ressources que vous rcolterez.");
        int q = sc.nextInt();
        joueur.recolteresourceOuvrier(q);
        System.out.println(joueur.avoirNom() + " a rcolt " + q + " units de ressources chez ses ouvriers.");
        joueur.recolteOr();
        joueur.payerOuvrier();

        System.out.println("#####Rcapitulatif du joueur "+joueur.avoirNom()+"#####");
    }
}
```

Déclaration du vainqueur



Joueurs

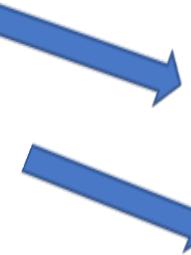


Évaluation des points gagnés par chaque joueur

Un seul joueur cumule le plus de points



Il remporte la partie



Match nul!

Plusieurs joueurs obtiennent le nombre maximal de points

Capacité de joueurs :

À noter que la gestion des participants à l'aide de listes permet un nombre potentiellement illimité de joueurs.



```
System.out.println("#"+joueur.avoirNom()+" a "+joueur.avoirOrOuvrier() +" Or en stock. #");
System.out.println("#Listes des tuiles possdes par le joueur#");
for(Ouvrier ouvrier : joueur.avoirListeOuvrier()) {
    System.out.println("#-)un ouvrier sur la tuile ("+ouvrier.avoirPosition().avoirx()+" , "+ouvrier.avoirPosition().avoiry()+" qui est de type"+
ouvrier.avoirPosition().toString()+"#");
}
System.out.println("#####");

}

/** Dclare le vainqueur de la partie ou dclenche une exception en
 * cas d'galit.
 * @return le joueur victorieux de la partie.
 * @exception DrawException lance en cas d'galit.
 */
public String declarerVainqueur() throws DrawException{
    Joueur gagnant = null;
    boolean egalite = false;
    for(Joueur joueur: this.joueurs) {
        if(gagnant==null) {
            gagnant=joueur;
        }
        else if(joueur.avoirPointsOuvrier()>=gagnant.avoirPointsOuvrier()) {
            if(joueur.avoirPointsOuvrier()==gagnant.avoirPointsOuvrier()) {
                egalite = true;
            }
            else {
                gagnant = joueur;
            }
        }
    }

    if(egalite){
        throw new DrawException("match nul!");
    }
    return gagnant.avoirNom() + " remporte la partie.";
}
```

Mains finaux :

<<Java Class>>

CJeuMain

jeu

CJeuMain()

Smain(String[]):void

<<Java Class>>

CAgricoleMain

jeu

CAgricoleMain()

Smain(String[]):void

<<Java Class>>

CGuerreMain

jeu

CGuerreMain()

Smain(String[]):void

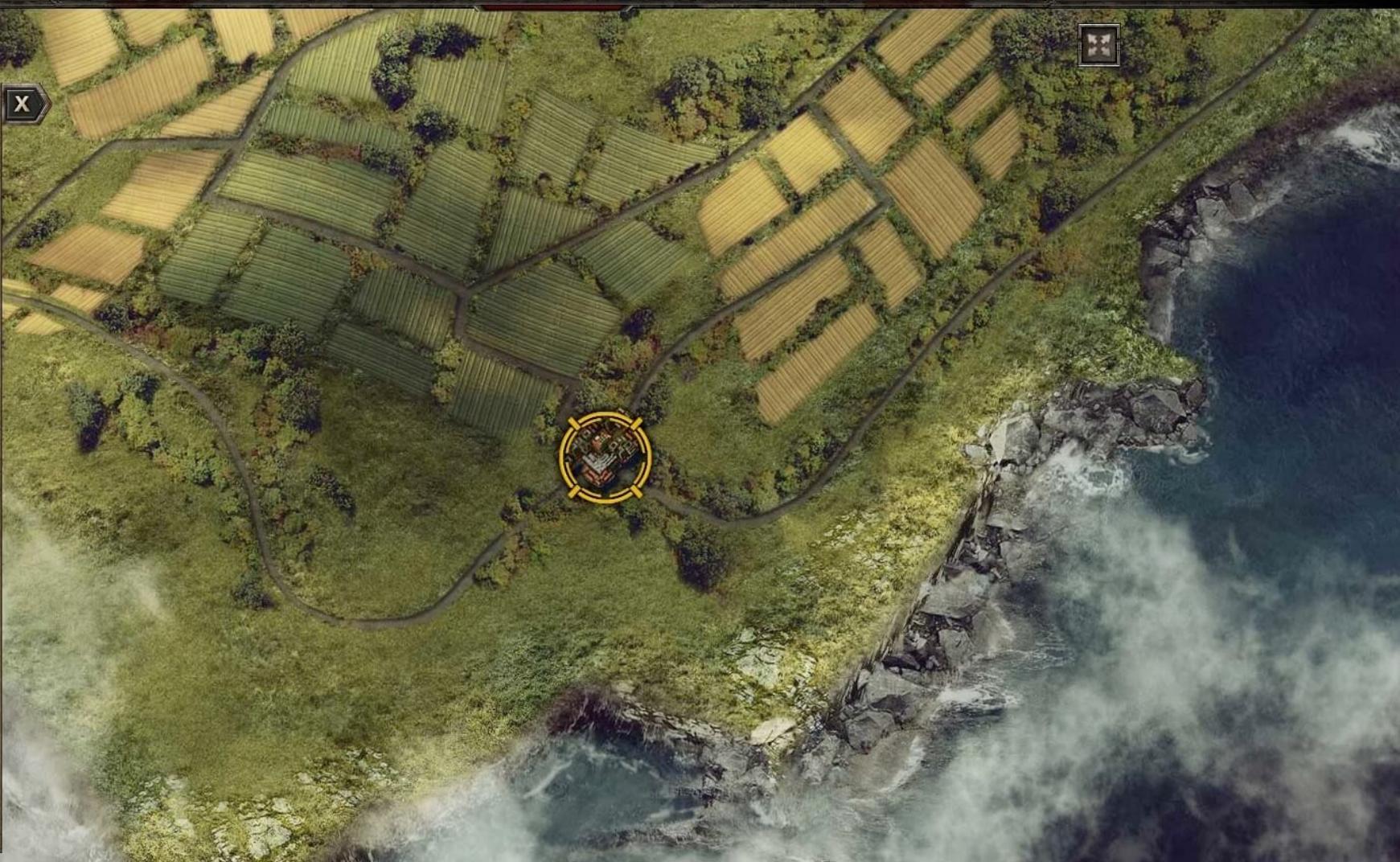
13,081 Md. 2,769,637 34,200 217,300,205 10,201 bl. 47,711,737 102 2,075 Mil.



Aperçu

X

[Aperçu][10/20/2017 11:45:47 - Système]
Connectez-vous au serveur de chat !
[Aperçu][10/20/2017 11:45:48 - Système]
Bienvenue sur le chat !



⊕ ⊖ ⊕ ? X



Déroulement de la partie tant que les 6 tours n'ont pas été joués et qu'il reste des tuiles libres



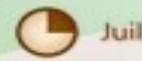


ABONDANCE

HAB : 26 865

EXPERTISE AGRICOLE : 86 %

OBJECTIFS



Juill



/ 9000 unités



/ 9000 unités



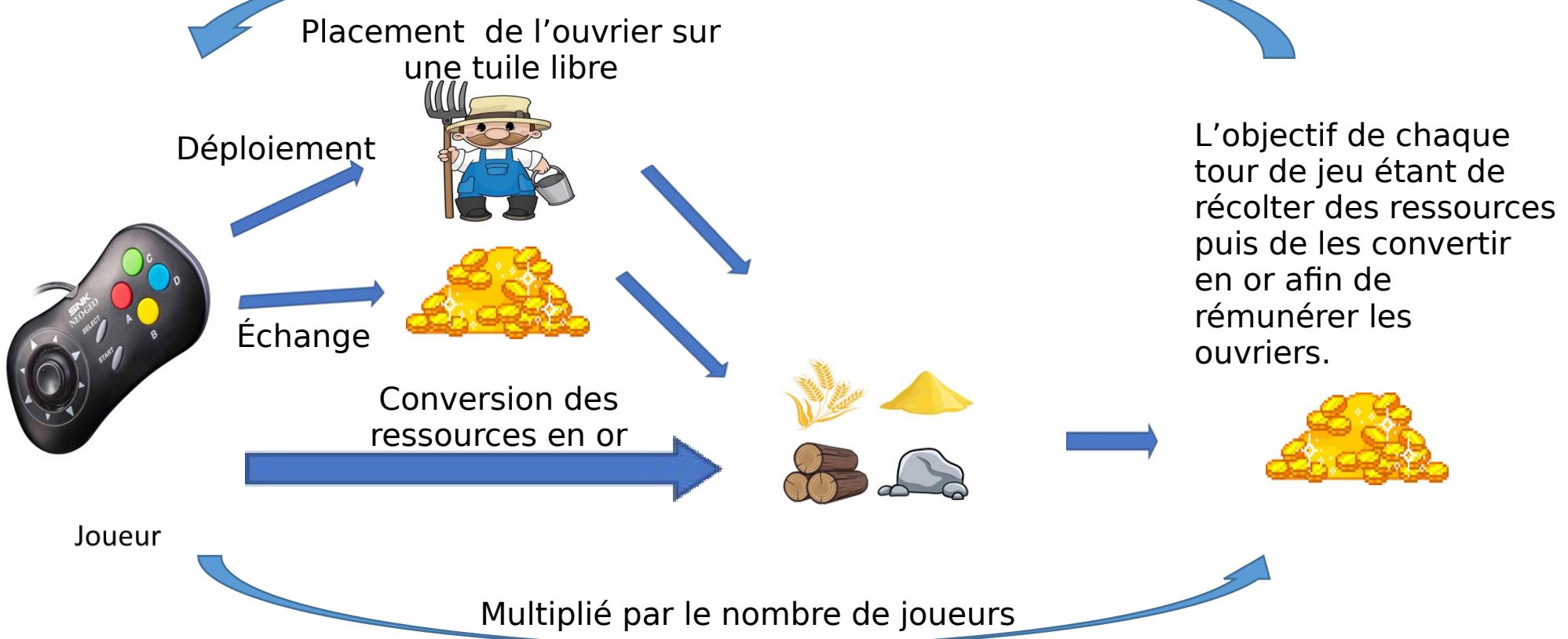
/ 9000 unités



453 562



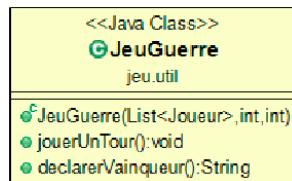
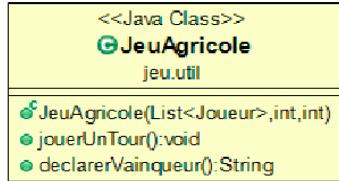
Déroulement de la partie tant que les 6 tours n'ont pas été joués et qu'il reste des tuiles libres



L'extensibilité :

Ajouter un Jeu :

Jeu



Ajouter un Emplacement :

Tuile



Ajouter un Personnage :

Personnage

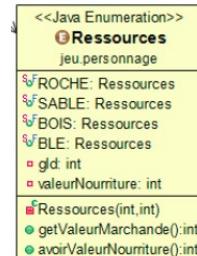


Ajouter une action :

Dans la classe Personnage

Ajouter une ressource:

Dans l'enum Ressources



L'extensibilité : exemples

Jeu de pêche :

Personnage



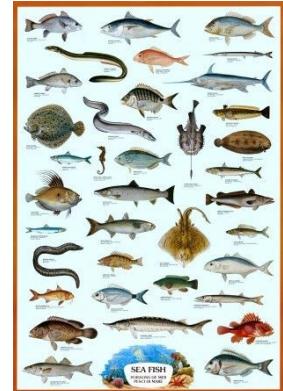
Actions



Emplacements



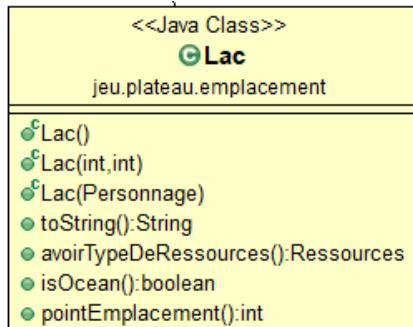
Ressources



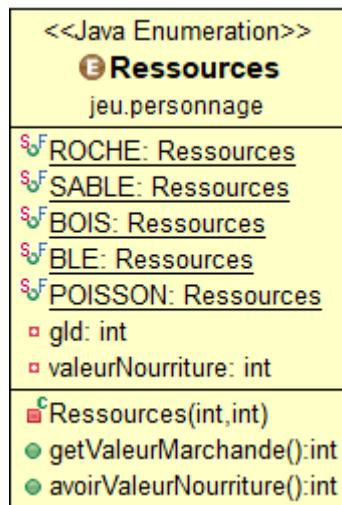
L'extensibilité : exemple

En UML ça donnerait :

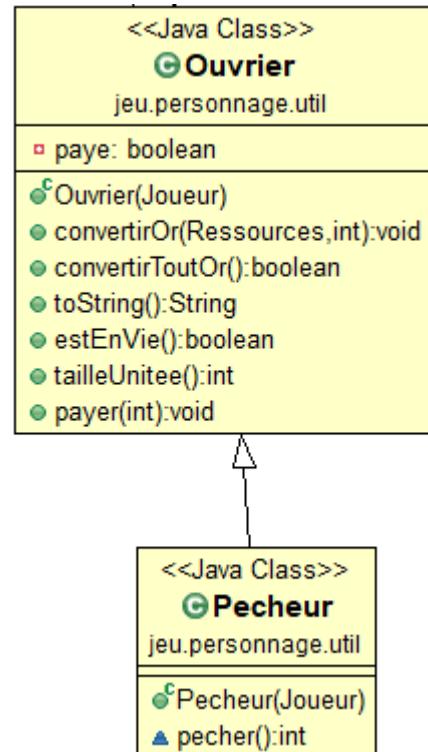
Emplacement



Ressources



Ajout du joueur



L'extensibilité : exemples

Jeu de chasse :

Personnage



Emplacements



Actions



Ressources



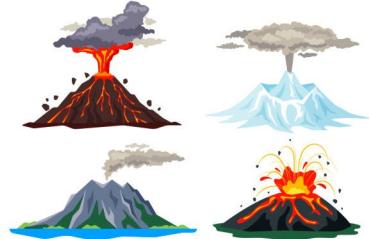
L'extensibilité : exemples

Jeu d'expédition scientifique :

Personnage



Emplacements



Actions



Ressources



Fonctionnement adopté en équipe :

Spécialisation :

Rotation des rôles en grande partie

Répartition :



Difficultés principales :

Programmation :

Classes :

Plateau

Tuile...

Adaptation :

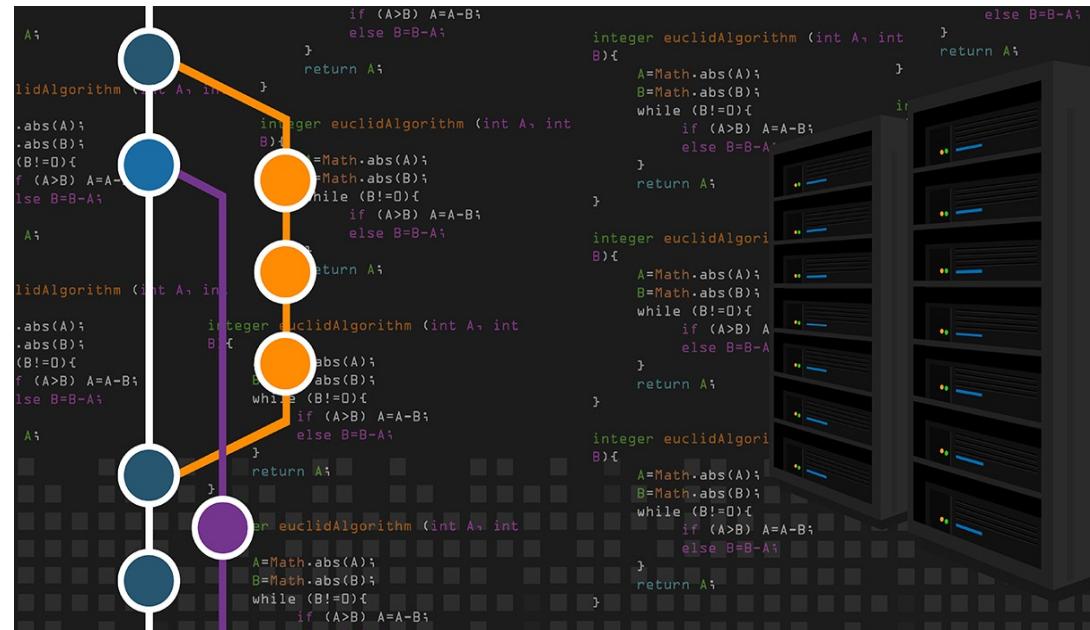
Mise à jour permanente des classes et évolutions fréquentes

Gestion :

Coordination

Répartition des tâches

Amplifications des difficultés due à au 100 % distanciel



Bilan :

Les aspects positifs:

Réalisation d'un projet du début à la fin

Expérimentation du travail en groupe

Mise en pratique des connaissances acquises en cours de POO

Maîtrise d'outils de développement

Git et GitLab

Eclipse



GitLab



Les aspects négatifs:

Tout ce qui découle de l'actuelle crise sanitaire

