Projet

Combat pour la Vie

Référence : Conception-CPLV-0E
Fournisseur
Date : 14 janvier 2019
Version/Édition : 0E
État : Préliminaire

Type de diffusion : Diffusion restreinte
Autre référence :

FICHE DE SUIVI DES AUTORISATIONS ET DIFFUSIONS

AUTORISATIONS PRESTATAIRE

	Fonction	Nom	Date	Visa
Auteur	Binôme	Duret Romain	29/12/2018	
Auteur	Binôme	Thomas Martins	29/12/2018	
Validé par				
Vérifié par				
Vérifié par				
Approuvé par				

AUTORISATIONS CLIENT

	Fonction	Nom	Date	Visa
Approuvé par				
Approuvé par				
Approuvé par				

DIFFUSION INTERNE

Nom	Fonction	Action	Date	Nb exemplaire(s)

Historique des révisions

Date	Description et justification de la modification	Auteur	Pages / Chapitre	Edition / Révision
29/12/2018	Création	Thomas Martins	Toutes	0A
02/01/2019	Avancée	Thomas Martins	Toutes	0B
08/01/2019	Corrections	Romain Duret	Toutes	0C
13/01/2019	Avancée	Thomas Martins	Toutes	0D
14/01/2019	Avancées	Tous	Toutes	0E

Table des matières

1 Introduction

1.1 Objet du document

La conception générale d'un logiciel est une démarche rationnelle, qui consiste à réduire la complexité initiale en la décomposant en constituants de niveau inférieur.

La conception permet de préciser *comment* vont être réalisées les spécifications.

La conception préliminaire permet d'élaborer une solution technique répondant aux spécifications. Elle précise l'architecture de cette solution (interfaces internes, constituants principaux, ...), ainsi que son comportement dynamique (logique d'enchaînements, diagrammes d'état, parallélisme, synchronisation, ...).

L'architecture du matériel cible est présentée dans la conception générale si cela n'a pas été fait dans un autre document.

Il faut présenter succinctement la structure du document :

- description et analyse des ressources matérielles et logicielles,
- diagramme des catégories ou des classes principales,
- structure statique (interfaces externes, interfaces internes, ...),
- structure dynamique (synchronisation entre tâches, séquencement des traitements, passages entre les différents états ou mode de traitement, ...),
- implémentation sur le matériel cible (allocation des ressources CPU mémoire bande passante, adresse des périphériques, couche basse de protocoles spécifiques, ...).

La conception générale peut être réalisée avec plus ou moins de détail en fonction de la taille du projet. Dans notre cas, pour un *petit projet*, la conception générale est suffisante.

1.2 Responsabilités

La rédaction de la conception générale est de la responsabilité du Chef de projet. Il juge de son état complet et décide de sa présentation en revue de conception.

La conception générale est souvent livrable, elle permet de décrire de façon complète le travail à réaliser.

1.3 Evolution

La conception générale fait partie de la référence de réalisation du système; toute modification de cette référence intervenant après le prononcé de revue de conception doit être traitée comme une demande d'évolution.

Quand dans la phase de développement des détails d'implémentation apparaissent, l'évolution du document de conception générale évolue en fonction des types de projets.

Pour notre projet, la seule référence étant la conception générale, celle-ci doit être mise à jour.

1.4 Outils utilisés

Les documents de base sont rédigés avec le logiciel Word sur Windows.

Les documents UML ont été conçus sur StarUML, en respectant les conventions UML.

L’utilisation de papier et crayons a été une utilisation nécessaire pour la conception de l’implémentation de certains mécanismes du projet.

Terminologie

1.5 Abréviations

UML	**U**nified **M**odeling **L**anguage

1.6 Définitions des termes employés

attribut	un attribut est une information caractéristique mémorisée par un objet.
cas d'utilisation	cas d'utilisation du système, par extension il représente également la technique de modélisation mise en euvre dans UML (use case).
catégorie	une catégorie consiste en un regroupement logique de classes à forte cohérence interne et faible couplage externe, associée au concept UML de package. Ce concept permet une présentation plus synthétique du diagramme des classes d'un système réel.
classe	une classe définit un ensemble d'objets similaires potentiels. Elle fournit le modèle de la structure et les possibilités de chaque objet.
objet	un objet est une instance d'une classe, c'est une entité informatique unique possédant ses propres attributs et opérations
opération ou méthode	une opération est un traitement spécifique qu'un objet est en charge de fournir.
tâche	une tâche représente un élément manipulé par le système et ordonnançable de manière individuelle. cela peut représenter un process d'un système Unix, une tâche d'un moniteur temps-réel, un thread d'une application.
threads	codes exécutés de manière concurrente par le système d'exploitation mais partageant le même espace mémoire.

2 Description et analyse de l'environnement

2.1 Description des ressources matérielles

Notre jeu ne demandant pas de grande puissance de calcul pour fonctionner, n'importe quel ordinateur d'aujourd'hui sera capable de le faire fonctionner.

Il aura besoin d'un ordinateur avec un écran pour l'affichage.

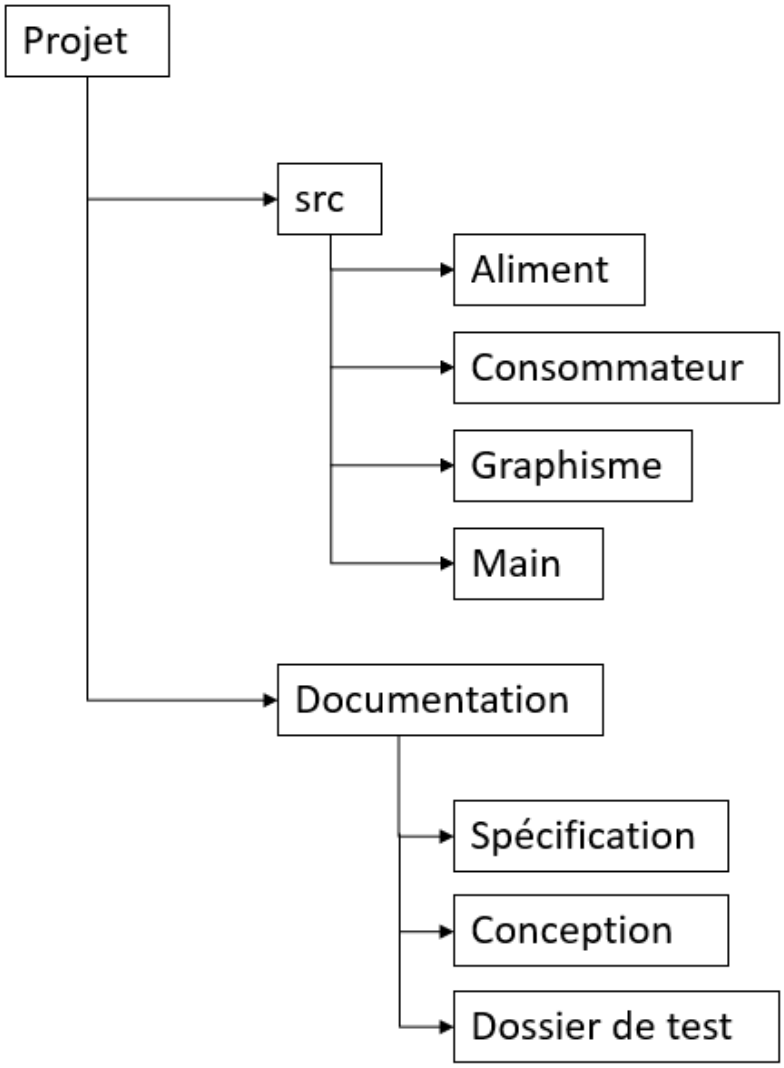
2.2 Description des ressources logicielles

Pour fonctionner notre jeu aura besoin des logiciels suivant :

- Java (Java SE-10)

2.3 Organisation de l'espace de travail

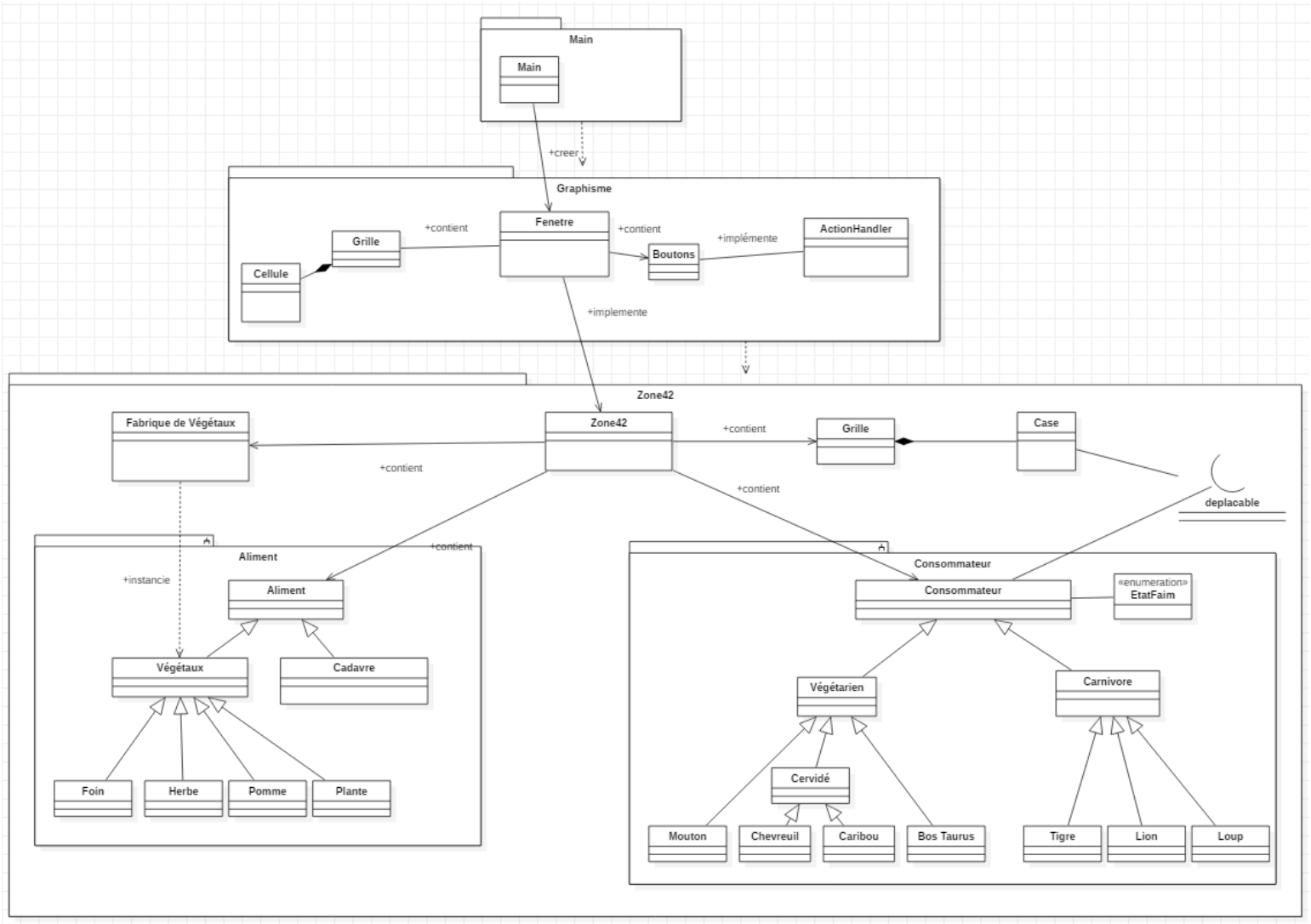
Notre espace de travail est l'IDE Eclipse, les fichiers sont stockés dans un dépôt GitHub. Les fichiers sont regroupés de la manière suivante :



3 Structure statique

3.1 Décomposition générale

Notre projet va se décomposer de la façon suivante :



On va avoir 3 principaux Packages qui seront :

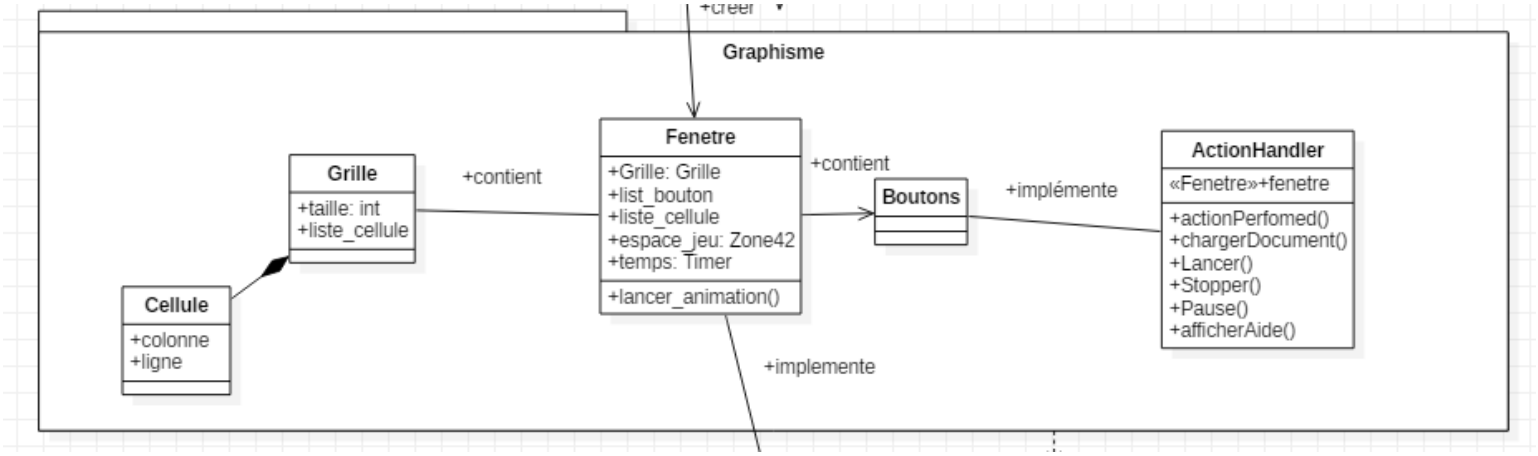
- Main : Package qui va contenir le code principal pour faire fonctionner le projet. C'est dans ce package qu'on retrouvera la classe Main qui contiendra le programme du jeu qui tournera en boucle.

- Graphisme : Dans ce package on va retrouver toutes les classes liées à l'Interface Homme-Machine. On va principalement avoir la classe Fenêtre qui contiendra les éléments d'autres classes permettant le bon affichage du jeu à savoir les classes Boutons, Grille et Cellule.
  Ce package contiendra aussi une classe ActionHandler qui va servir à interpréter les actions de l'utilisateur.

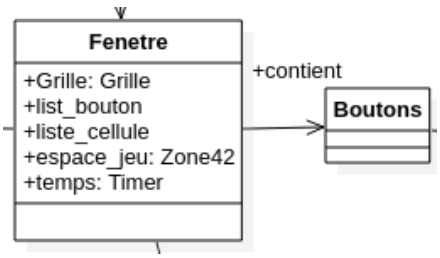
- Zone42 : Ce package contient tout les acteurs essentiels au jeu. La classe Zone42 du package du même nom va contenir les éléments des autres classes. Dans ce package on retrouve également 2 autres packages qui sont Aliment et Consommateur. Dans le package Aliment on va retrouver les aliments necéssaire à la survie des Consommateurs Végétarien et Carnivore.
  Le Package Consommateur contient les classes Consommateur, Végétarien et Carnivore ainsi que leurs classes filles.

3.2 Allocations fonctionnelles et spécifications

3.2.1 Package Graphisme



Classe Fenêtre



Attributs :

- private Grille grille : Objet de type Grille qui représente la grille du jeu. Il n'y en a qu'une seule.

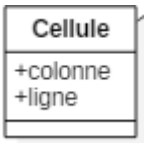
- private Boutons [] liste_bouton : Tableau d'objet de type Bouton qui répertorie la liste de tous les boutons
- private Cellule [] liste_cellule : Tableau d'objet de type Cellule qui répertorie la liste de toutes les Cellules.
- private Zone42 espace_jeu : La zone accueillant l'ensemble des Entités
- Integer temps : Objet de type Integer qui donne le temps en cycle.

Classe Grille

Attributs :

- Private int taille : Objet de type Entiers qui renseigne la taille de la grille.
- Cellule [] liste_cellule : Tableau d'objet de type Cellule qui répertorie la liste de toutes les Cellules.

Classe Cellule



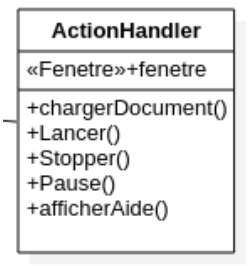
Attributs :

private int colonne : Nombre de colonnes dans une cellule

private int ligne : Nombre de lignes

Classe Bouton

Classe ActionHandler



Attributs :

- private Fenetre fenetre : Objet de type Fenetre

Méthodes :

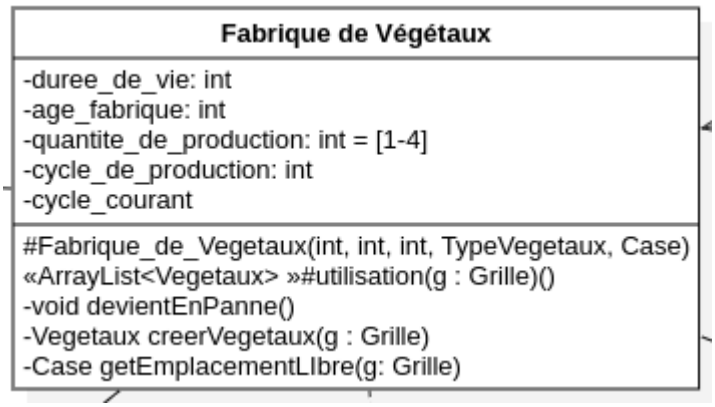
- public void chargerDocuments() : Permet de charger le fichier saisi par l'utilisateur
- public void lancer() : Lance le jeu
- public void stopper() : Arrête le jeu
- public void pause() : Met le jeu en pause
- public void afficherAide() : Affiche une fenêtre contenant de l'aide pour l'utilisateur.

Dans ce paragraphe est présenté rapidement pour chaque catégorie/classe:

- son rôle,
- son lien avec les cas d'utilisation décrits dans les spécifications.

3.2.2 Package Zone42

Classe Fabrique de Végétaux



Attributs :

- private Integer duree_de_vie : Durée de vie en cycle de la fabrique
- private Integer age_fabrique : Age de la fabrique
- private EtatFabrique etat : Etat de la machine
- private TypeVegetaux type : Type de végétaux que produit la fabrique
- private Case emplacement : Emplacement du consommateur sur la grille
- private Integer quantite_de_production : Quantité produite par cycle de production (entre 1 et 4)
- private Integer cycle_de_production : activation de la production tous les X cycles
- private Integer cycle_courant : Compteur pour connaitre quand la machine doit produire

Méthodes :

- public Fabrique_de_Vegetaux(int vie, int quantite, int cycle, TypeVegetaux tv, Case c) :

Constructeur de Fabrique_de_Vegetaux, les paramètres sont les suivants :

- vie : Entier représentant la durée de vie d'une Fabrique de végétaux
- quantite : Entier représentant la Quantité produite par cycle de production (entre 1 et 4)
- cycle : Entier, production tous les X cycle
- TypeVegetaux : Objet de type TypeVegetaux, Type de végétaux que produit la fabrique
- Case c : Objet de type Case qui renseigne l'emplacement d'une Fabrique

- public ArrayList<Vegetaux> utilisation(Grille g) :

Méthode permettant de faire fonctionner une Fabrique en y regroupant toutes les actions necessaires.
Vérifie si le cycle actuel correspond bien à un cycle de production.
Vérifie si la fabrique tombe en panne lors de production de végétaux.
Si ça correspond, elle renvoit un arraylist de vegetaux produit selon la quantite maximale de production de la machine.
Sinon, elle renvoi null en incrémentant le compteur

Le paramètre g de type Grille est la Grille unique du jeu.

- private void devientEnPanne() :

Fonction qui réalise un test pseudo aléatoire pour vérifier si une machine est en panne ou non

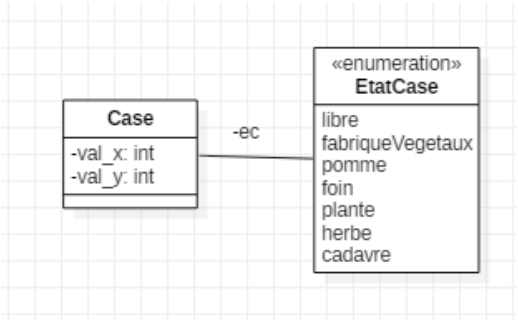
- private Vegetaux creerVegetaux(Grille g)

Créer un végétal et le retourne. S'il n'y a pas d'emplacement Libre, retourne null
Le paramètre est l'unique grille du jeu

- private Case getEmplacementLibre(Grille g)

Méthode qui vérifie les cases aux alentours pour vérifier si un emplacement est libre. Si oui elle renvoie l'emplacement sinon elle renvoie null.

Classe Case



Attributs :

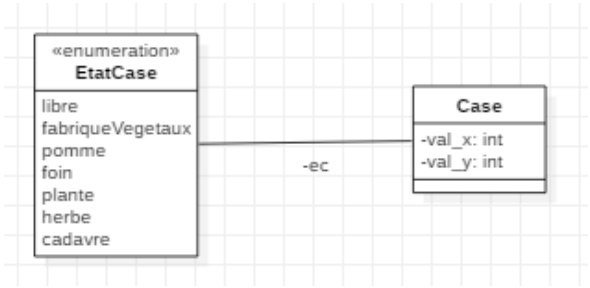
- private int val_x : Position sur l'axe des x
- private int val_y : Position sur l'axe des y
- private EtatCase ec : Renseigne l'état d'une case
- private Integer age_fabrique : Age de la fabrique

Méthodes :

Les méthodes suivantes sont triviales :

- public Case(int a, int b) : Constructeur d'une case qui prend en paramètre la position de la case
- public int getVal_x() : accesseur de l'attribut val_x
- public int getVal_y() : accesseur de l'attribut val_y
- public void setVal_x(int val_x)
- public void setVal_y(int val_y)
- public EtatCase getEc()
- public void setEc(EtatCase ec)

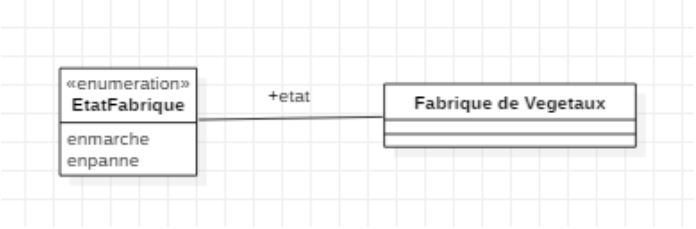
Classe EtatCase



Attributs :

- EtatCase ec, enumeration décrivant ce qui peut se trouver dans une case (libre, fabriqueVegetaux, vegetal,…)

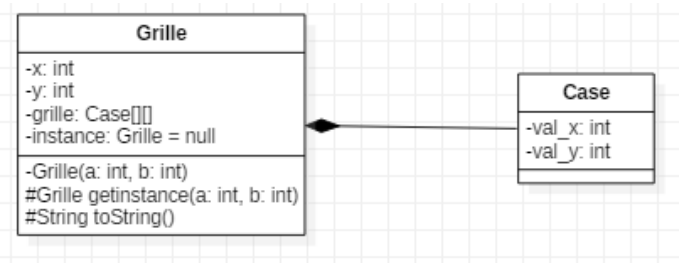
Classe EtatFabrique



Attributs :

- EtatCase ec, enumeration décrivant l'état d'une Fabrique : enmarche ou enpanne.

Classe Grille



Attributs :

- private Integer x : nombre de colonnes
- private Integer y : Nombre de ligne
- private Case[][] tab : Création des cases
- public static Grille instance : attribut pour design pattern Singleton Instancié à null (pas d'instance créé)

Méthodes :

- private Grille(int a, int b) :

Constructeur implémenté par le design pattern Singleton
a nombre de colonne
b nombre de ligne

- public static Grille getinstance(int a, int b)

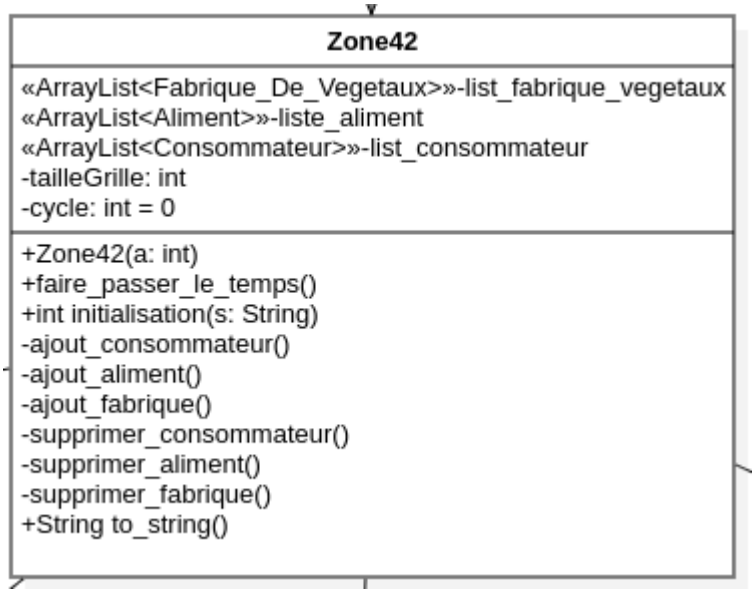
Récupère l'instance Grille ou en créé une s'il n'y en a pas.
a nombre de colonne
b nombre de ligne
Retourne une nouvelle grille si instance==null, sinon LA grille

- public String toString() :

Affichage de la grille

- public EtatCase getEtat(Case c)
- public EtatCase getEtat(int x, int y)
- public void setEtat(EtatCase ec, int x, int y)
- public void setEtat(EtatCase ec, Case c)
- public int getX()
- public int getY()

Classe Zone42



Attributs :

- private ArrayList<Fabrique_de_Vegetaux> list_fabrique_vegetaux : Liste de Fabrique d'aliment de la Zone42
- private ArrayList<Aliment> list_aliment : Liste d'aliment de la Zone42
- private ArrayList<Herbivore> list_herbivore : Liste des consommateurs de la Zone42
- private ArrayList<Carnivore> list_carnivore
- private Integer tailleGrille : taille de la grille de jeu
- private int cycle : L'unité du temps du jeu le cycle

Méthodes :

- public Zone42(int a) : Constructeur de Zone42 qui initialise la Grille avec une dimension de a cases * a cases
- public void faire_passer_le_temps() : simule la durée d'un cycle.
- public int initlialisation(Strings s) : initialise la simulation à partir d'un fichier (emplacement donné en paramètre) et instancie les objets correspondants.
- public int ajout_carnivore(Carnivore c) :

Rajoute un Carnivore à la liste
c Carnivore à rajouter
return 1 si ok, 0 si erreur

- public int supprime_carnivore(Carnivore c) :

Supprime un Carnivore à la liste
c Carnivore à rajouter
return 1 si ok, 0 si erreur

- public int ajout_herbivore(Herbivore h) :

Rajoute un Herbivore à la liste

h Herbivore à rajouter
return 1 si ok, 0 si erreur

- public int supprime_herbivore(Herbivore h) :

Supprime un Herbivore à la liste
h Herbivore à supprimer
return 1 si ok, 0 si erreur

- public int ajout_aliment(Aliment a) :

Rajoute un aliment
a aliment à rajouter
return 1 si ok, 0 si erreur

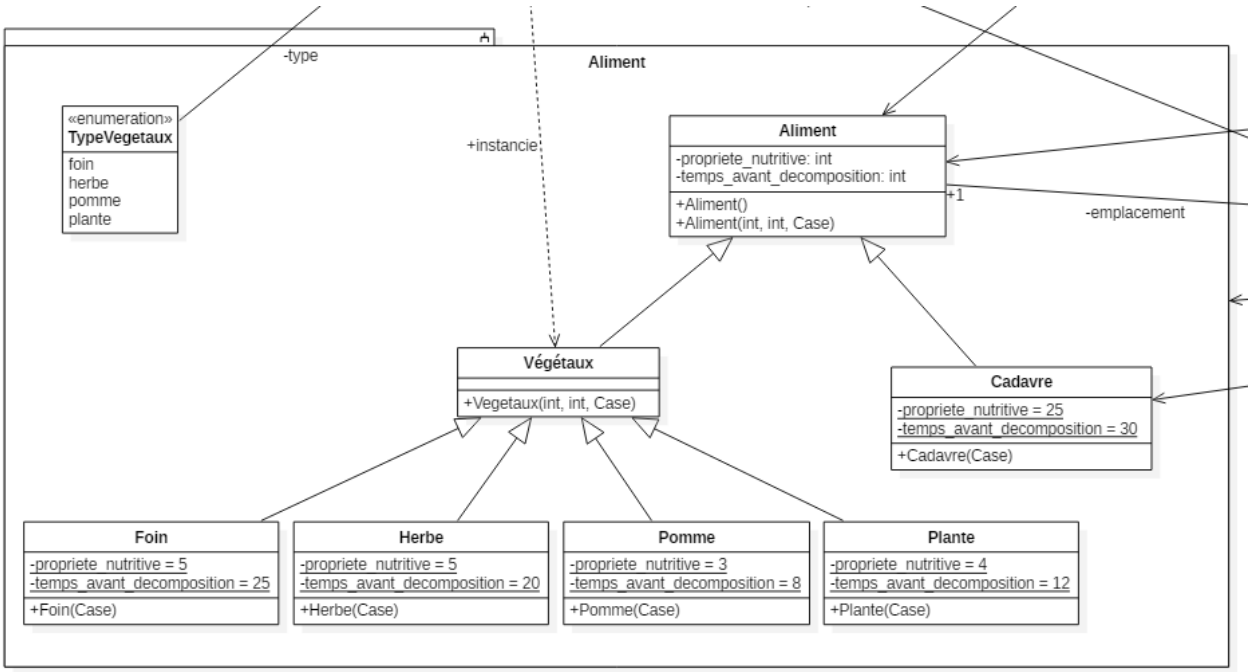
- public int initialisation() :

Initialise la zone selon les paramètres passés par l'utilisateur (emplacement du fichier d'initialisation donné en paramètre) et instancie les objets correspondants.
return 1 si ok, 0 si erreur

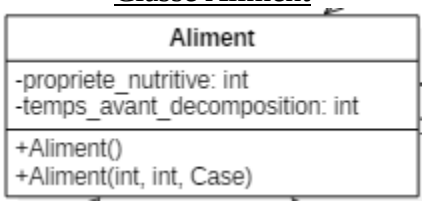
- public String toString() :

Créer une chaine de caractère pour afficher le contenu de la zone 42.

3.2.3 Package Aliment



Classe Aliment



Attributs :

- private Integer propriete_nutritive : Capacité de satisfaire la faim des consommateurs
- private Integer temps_avant_decomposition : temps en cycle avant leurs disparitions
- private Case emplacement : Emplacement de l'aliment sur la grille

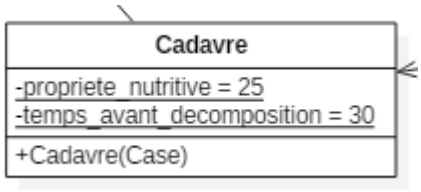
Méthodes :

- public Aliment() : constructeur vide
- public Aliment(int valeur_nutritive, int valeur_conservation, Case c) :

Constructeur simple
valeur_nutritive Capacité de satisfaire la faim des consommateurs
valeur_conservation temps en cycle avant disparition
c emplacement de l'aliment

- Ainsi que des getters et setters, triviaux.

Classe Cadavre



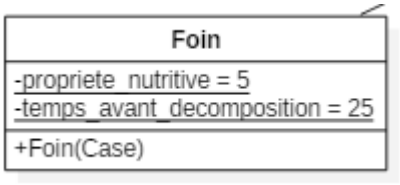
Attributs :

- public static final int propriete_nutritive : Capacité de satisfaire la faim d'un Carnivore
- public static final int temps_decomposition : temps en cycle avant leurs disparitions

Méthodes :

- public Cadavre(Case emplacement) : Constructeur de Cadavre qui fait appel au constructeur de sa classe mère Aliment

Classe Foin



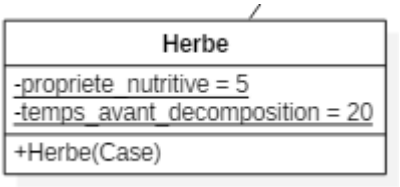
Attributs :

- public static final int propriete_nutritive : Capacité de satisfaire la faim d'un Carnivore
- public static final int temps_decomposition : temps en cycle avant leurs disparitions

Méthodes :

- public Foin(Case emplacement) : Constructeur de Foin qui fait appel au constructeur de sa classe mère Aliment

Classe Herbe



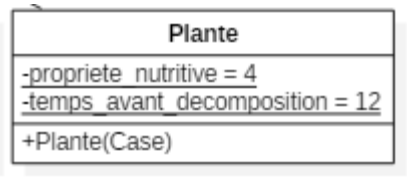
Attributs :

- public static final int propriete_nutritive : Capacité de satisfaire la faim d'un Carnivore
- public static final int temps_decomposition : temps en cycle avant leurs disparitions

Méthodes :

- public Herbe(Case emplacement) : Constructeur d'Herbe qui fait appel au constructeur de sa classe mère Aliment

Classe Plante



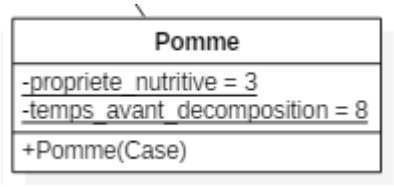
Attributs :

- public static final int propriete_nutritive : Capacité de satisfaire la faim d'un Carnivore
- public static final int temps_decomposition : temps en cycle avant leurs disparitions

Méthodes :

- public Plante(Case emplacement) : Constructeur de Plante qui fait appel au constructeur de sa classe mère Aliment

Classe Pomme



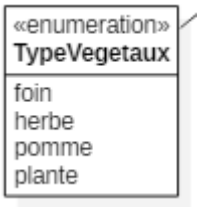
Attributs :

- public static final int propriete_nutritive : Capacité de satisfaire la faim d'un Carnivore
- public static final int temps_decomposition : temps en cycle avant leurs disparitions

Méthodes :

- public Pomme(Case emplacement) : Constructeur de Pomme qui fait appel au constructeur de sa classe mère Aliment

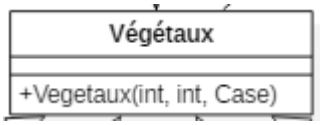
Classe TypeVegetaux



Attributs :

Énumération décrivant le type d'un Végétaux (foin, herbe, pomme ou plante)

Classe Végétaux



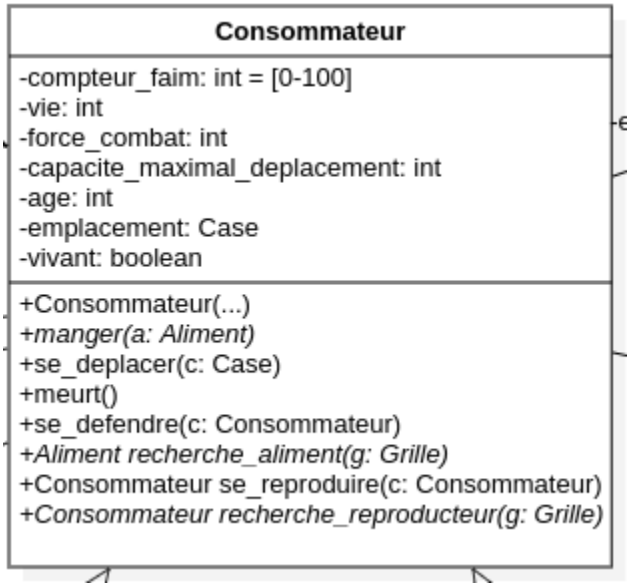
Méthodes :

- public Vegetaux(int valeur, int valeur2, Case emplacement) :

Constructeur de Végétaux faisant appel au constructeur de la classe mère Aliment

3.2.4 Package Consommateur

Classe Consommateur



Attributs :

- Private Integer compteur_faim :

Compteur pour la fin, entier allant de 100 à 0. Cela influe sur l'EtatFaim du Consommateur (voir chapitre sur la classe EtatFaim). Décrémente à chaque cycle.

- Private Integer vie :

Vie du consommateur

- Private Integer force_combat :

Force en combat du Consommateur

- Private Integer Capacite_maximale_deplacement :

Capacité du Consommateur de déplacement par cycle

- Private Integer age :

Age (en nombre de cycle) du consommateur

- Private Case emplacement :

Emplacement du Consommateur

- Private boolean vivant :

Boolean pour dire si le Consommateur est bien vivant (true si vivant, false si mort)

Méthodes :

- public Consommateur(…)

Constructeur de Consommateur

- public abstract manger(Aliment a) :

Mange l'Aliment si c'est possible.

- public se_deplacer(Case c) :

Déplace (si c'est possible) le Consommateur sur cette case

- public meurt() :

Se déclare mort (si la vie est égale ou inférieure à 0, vivant passe à false)

- public se_defendre(Consommateur c) :

Se défend d'une attaque d'un autre Consommateur (passé en paramètre)

- public abstract Aliment recherche_aliment(Grille g) :

Recherche un aliment autour du consommateur

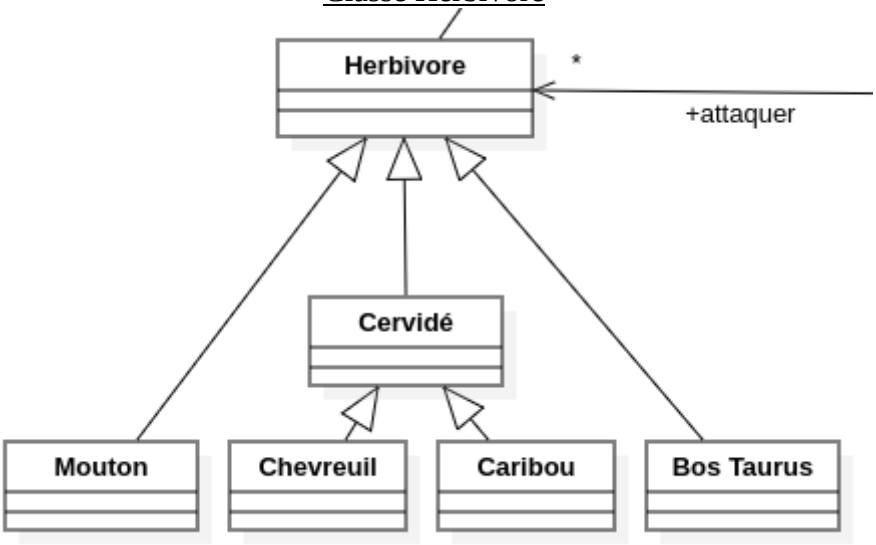
- public abstract Consommateur se_reproduire(Consommateur c) :

Si les conditions sont réunies, les deux Consommateurs se reproduisent et instancient un nouveau Consommateur de même espèce.

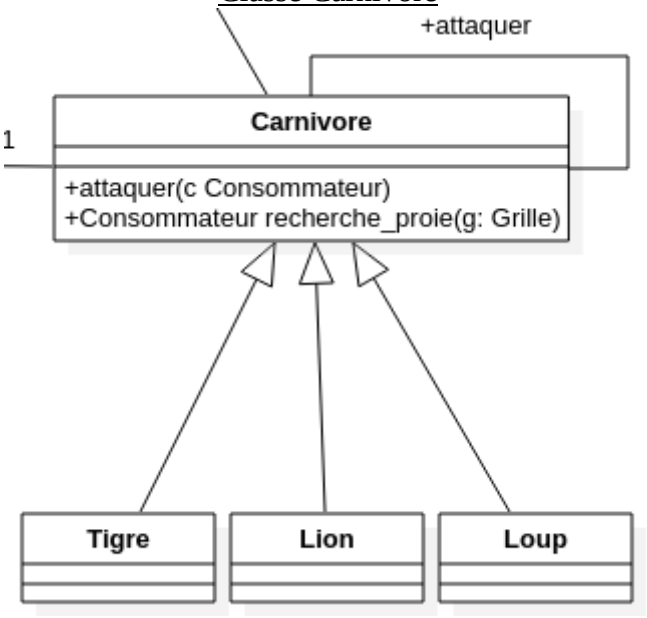
- public abstract Consommateur recherche_reproducteur(Grille g) :

Méthode qui fait qu'un Consommateur recherche un membre de son espèce de sexe opposé autour de lui et le renvoi en paramètre. Renvoi null s'il ne trouve rien.

Classe Herbivore



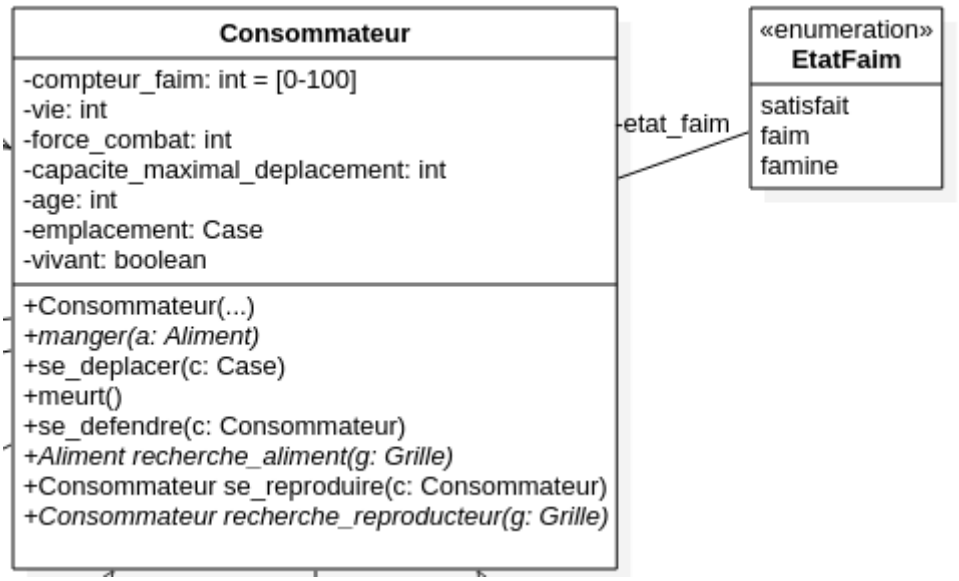
Classe Carnivore



Méthodes :

- public attaquer(Consommateur c) : attaque le Consommateur c (phase de combat)
- public Consommateur recherche_proie(Grille g) : recherche une proie autour du Carnivore et le renvoi s'il en trouve un. Null sinon.

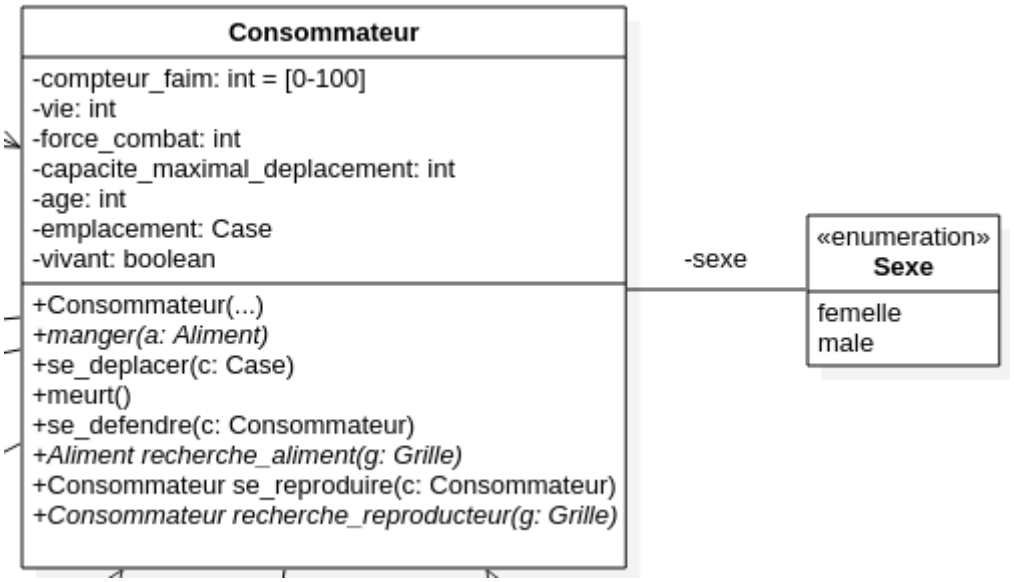
Classe EtatFaim



Enumération pour indiquer l'état de faim de l'animal :

- satisfait (compteur de faim entre 100 et 65) : recherche un animal de la même espèce pour se reproduire.
- faim (compteur de faim entre 64 et 20) : recherche de la nourriture activement, ne peut plus se reproduire.
- famine (compteur de faim entre 19 et 0) : réduit la vie de 1 à chaque cycle. Le Consommateur se déplace de façon maximum.

Classe Sexe



Enumération pour indiquer le Sexe de l'animal (femelle ou male).

4 Aspects dynamiques

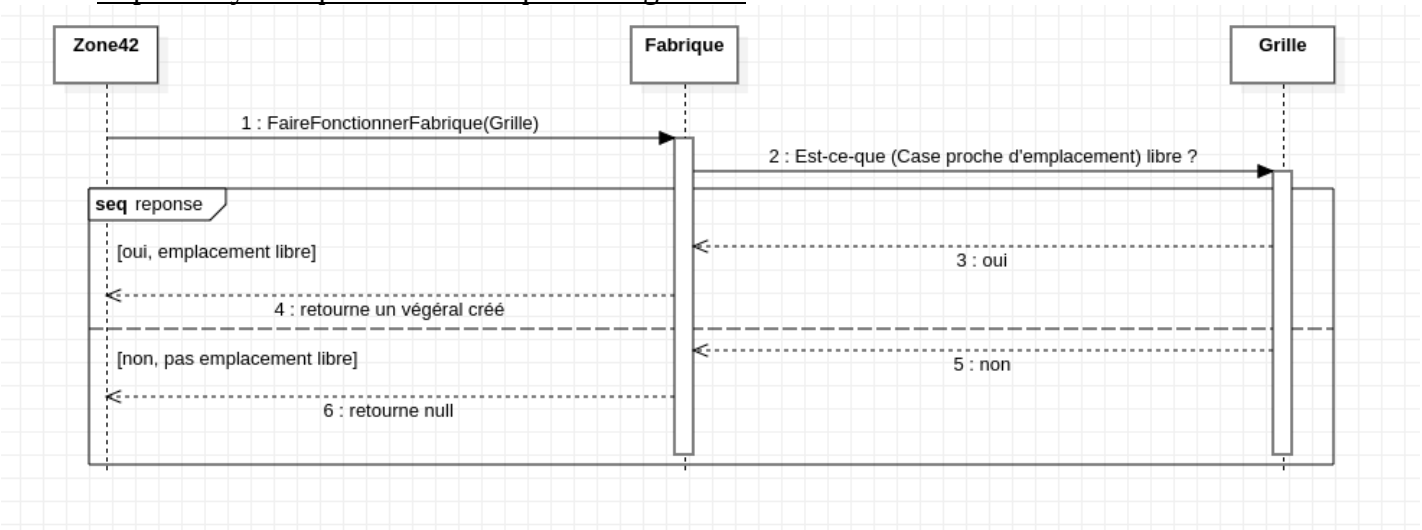
Les interactions dynamiques entre les constituants sont présentées de préférence sous forme de graphiques.

4.1 Interfaces externes

Récupérer fichier d'initialisation

4.2 Echanges entre éléments

4.2.1 Aspects Dynamiques de la Fabrique de Végétaux :

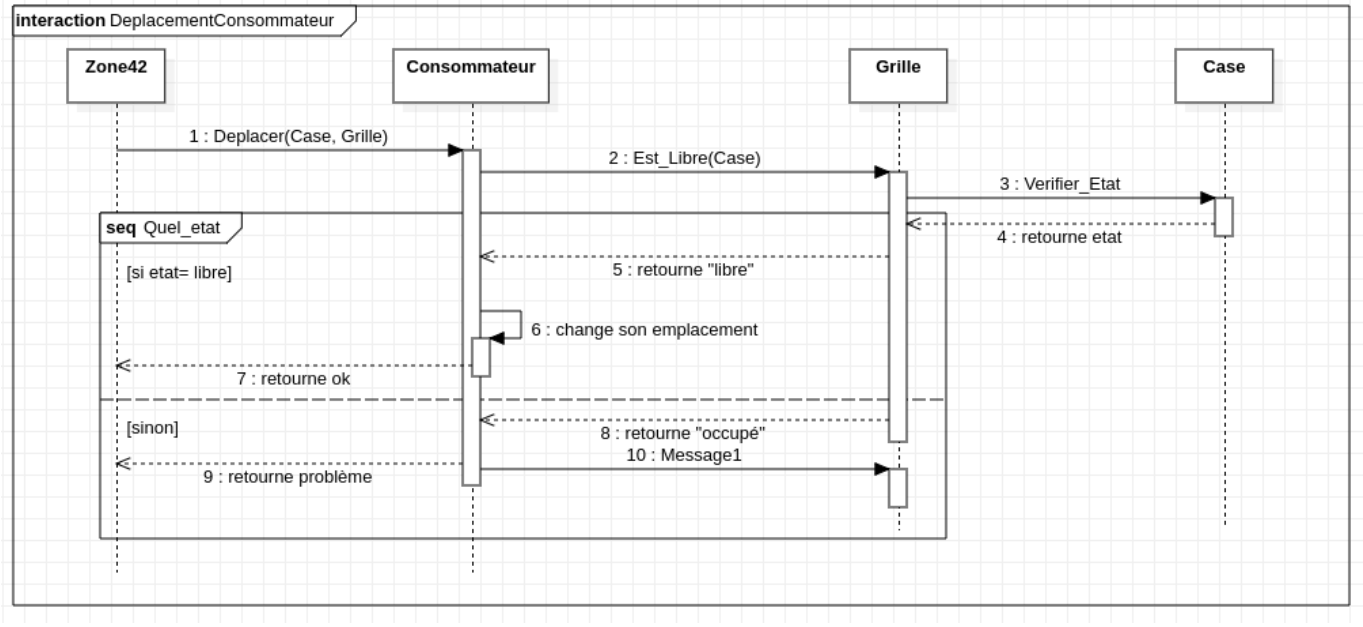


On commence par appeler FaireFonctionnerFabrique(Grille) qui va vérifier en premier si un emplacement à proximité est libre.

Si oui alors on peut créer un nouvel Aliment et le retourner.

Sinon on ne créer rien et on retourne null.

4.2.2 Aspects Dynamiques du déplacement d'un Consommateur :

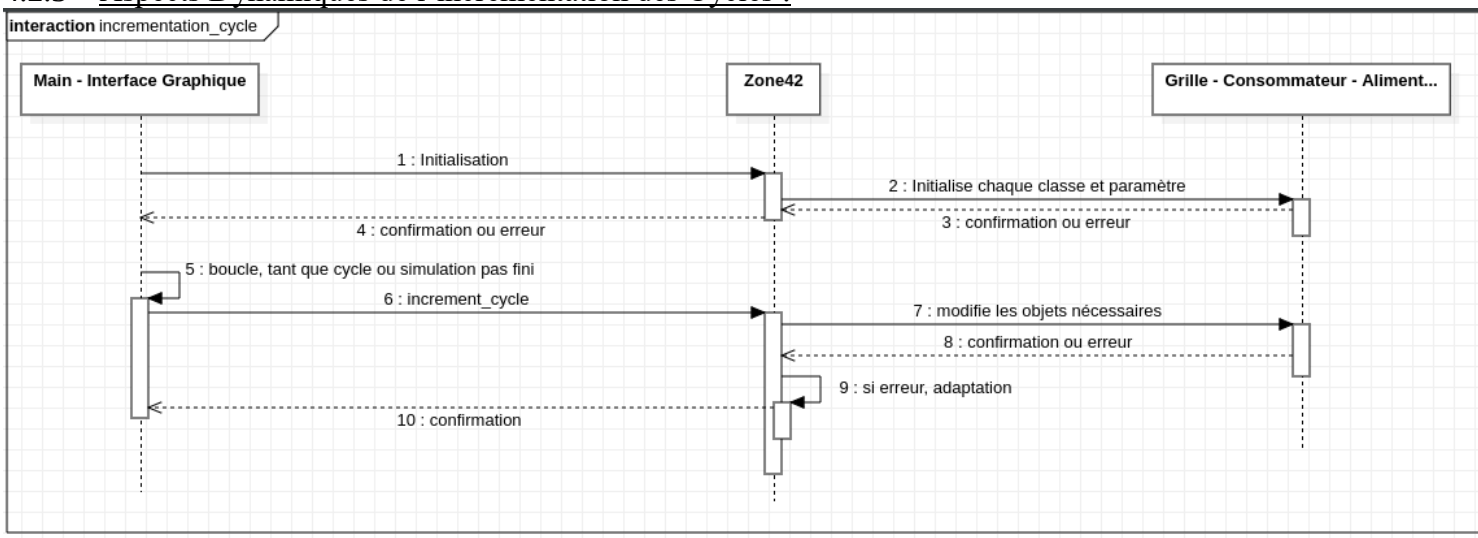


Le Consommateur va d'abord initier le déplacement avec la méthode Deplacer(Case, Grille). Il va vérifier que l'emplacement sur lequel il veut se déplacer est libre en appelant la méthode Est_Libre(Case) où Case est la Case sur laquelle veut se déplacer le Consommateur. Cette méthode appelle Verifier_Etat qui renseigne l'état de la Case.

Si celle-ci est libre, l'état libre sera retourné sinon on cherche un autre emplacement.

Si aucun emplacement est libre on retourne un état occupé et un statut problème.

4.2.3 Aspects Dynamiques de l'incrémentation des Cycles :



Pour incrémenter les Cycles on commence par initialiser la Zone42. Si l'initialisation se passe bien, Les cycles vont pouvoir être incrémentés.

L'incrémentation des cycles se fait à chaque « réveil » de la fonction principale qui est « endormie » à l'aide de la méthode *sleep(time)* où time représente un temps en milliseconde. En temps normal la période entre 2 réveils est de 1 seconde.

Cette période pourra être changé si on souhaite accélérer ou ralentir le temps.

4.3 Comportement du logiciel en erreur

Erreurs possibles :

- Mauvaise variable (entiers) => tronquer (on change la variable provoquant une erreur)
- Pas de création => ignorer
- Autre erreur => Fin prématurée du jeu