Projet

Jeu de la Vie

Référence	:	POO jeu de la vie
Fournisseur		
Date	:	10 Décembre 2018
Version/Édition	:	1.0
État	:	Préliminaire
Type de diffusion	:	Diffusion restreinte
Autre référence	:	

FICHE DE SUIVI DES AUTORISATIONS ET DIFFUSIONS

AUTORISATIONS PRESTATAIRE

	Fonction	Nom	Date	Visa
Auteur	Directeur de projet	Martinez/Vu Hoai		
Validé par				
Vérifié par				

AUTORISATIONS CLIENT

	Fonction	Nom	Date	Visa
Approuvé par				
Approuvé par				

DIFFUSION INTERNE

Nom	Fonction	Action	Date	Nb exemplaire(s)

DIFFUSION EXTERNE

Nom	Fonction	Action	Date	Nb exemplaire(s)

Historique des révisions

Date	Description et justification de la modification	Auteur	Pages / Chapitre	Edition / Révision
05/03/2014	Création		Toutes	0A

Table des matières

Préambule

Tout au long de ce document de spécifications, nous allons utiliser des règles pour une meilleure compréhension :

- Le nom des Personnage sera en *Italique*
- Le nom des classes sera en **gras**.

Introduction

1.1 Objet du document

Le jeu se présentera dans une fenêtre. L'utilisateur se verra attribuer un espace prédéfini pour pouvoir organiser la manifestation. Il aura un nombre donné de *Manifestant* à placer sur l'espace de jeu. Une fois cela fait, le jeu se lancera, les *Manifestants* se développeront indépendamment. Au bout d'un certain seuil de *Manifestants*, des *CRS* apparaîtront de manière aléatoire sur la carte afin de contrer cette manifestation. Les *Manifestants* gagnent s'ils atteignent un certain nombre que nous avons spécifié dans le fichier paramètre, mais ils perdent s'ils retombent au même nombre de départ. Le jeu sera régi par des règles simples afin de pouvoir lui laisser une autonomie dans son fonctionnement.
Les règles sont celles-ci :

-> Un *Manifestants* devient N*eutre* s'il est entouré de personne N*eutre* sans ami *Manifestant*

-> Un *CRS* est neutralisé s'il est entouré d'au moins six *Manifestants*

-> Une personne N*eutre* devient un *Manifestant* s'il est entouré d'au moins deux *Manifestants*

-> Le jeu se finit si les *Manifestants* atteignent le nombre de base en début de partie ou s'ils atteignent une certaine valeur définie par paramètre (voir description du fichier param.ini définit dans les chapitres suivants)

-> Si un *Manifestant* a pour voisin deux *CRS* alors il est placé en garde à vue et la case devient vide

-> Seul les *CRS* et les *Neutres* peuvent se déplacer de manière aléatoire

Afin de répondre à ces exigences, nous devons répartir les différentes demandes en fonctionnalité.
L'utilisateur peut choisir où faire apparaître les *Manifestants*, il peut en faire apparaître un nombre limité. Les personnages du jeu évoluent sans l'aide de l'utilisateur. Ils se développent indépendamment et de manière autonome. Ils se déplacent aléatoirement sur une grille. Les *CRS* apparaîtront aléatoirement sur la carte par petit groupe (allant de 1 à 5 personnes). L'utilisateur peut réinitialiser la partie s'il le souhaite ou mettre la simulation sur pause.
Enfin l'utilisateur pourra interagir avec le jeu via une interface. Les personnages sont représentés par des carrés de couleur qui évoluent sur une grille de taille fixe. Cases jaunes pour les *Manifestants*, grises pour les *Neutres*, bleu pour les *CR*S et blanches s'il n'y a personne.
Des messages seront affiché à un intervalle fixe afin d'indiquer différents événements tel que le dépassement d'un certain seuil de *Manifestants*, l'apparition des *CRS* ou encore la fin de la partie.
L'utilisateur utilise sa souris pour placer les *Manifestants* sur la *grille*. Pour faciliter le positionnement des *Manifestants*, on définit qu'un clic de l'utilisateur place des *Manifestants*. Il lui suffit de cliquer sur des carrés blancs.

1.2 Evolution

Dans une perspective d'évolution du projet, l'implémentation d'une fonctionnalité qui sauvegarde et charge un plateau voulu est pertinente. En effet, l'utilisateur peut quitter le jeu et revenir sans perdre l'état du plateau de jeu.

1.3 Outils utilisés

Pour la réalisation de ce projet, nous utiliserons principale le langage JAVA pour la partie programmation, GITHUB pour stocker le projet afin qu'il soit accessible par tous et pour finir de Word ou Google Docs pour la partie administrative afin de remplir les documents techniques liés au projet

Terminologie

1.1 Abréviations

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

1.2 Définitions des termes employés

Use case	Cas d'utilisation du système, par extension il représente également une technique de modélisation mise en œuvre dans UML
Classe	Association de données et de traitements modélisant un élément du système

Exigences

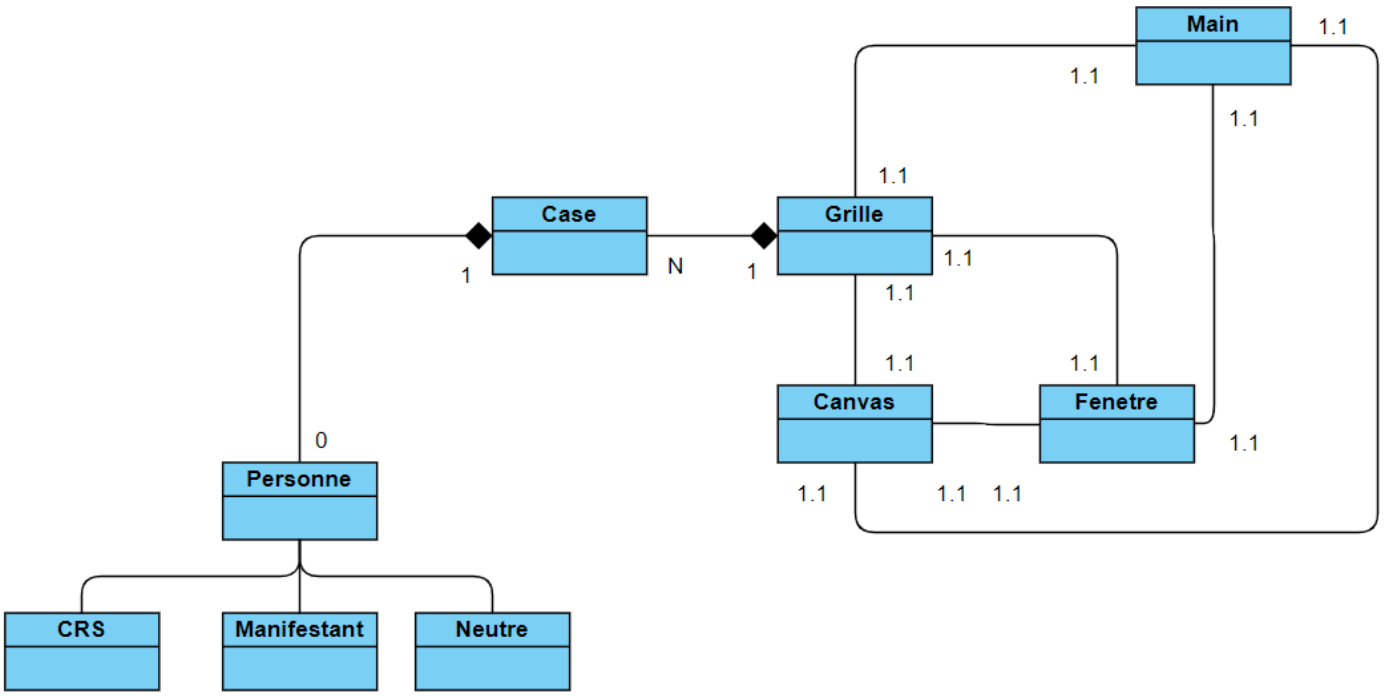
1.1 Présentation de la mission du produit logiciel

Le jeu doit envoyer des messages d'événements, d'erreurs si certaines règles ne sont pas respectées. Le jeu doit aussi prendre en compte le choix de l'utilisateur pour la mise en place des individus.

Nous devons alors traiter les erreurs qui peuvent être émise par l'utilisateur ou par le jeu lui-même et les régler pour garantir une bonne expérience de jeu. Nous devons prendre en compte la mise en place de l'utilisateur et changer le plateau du jeu en fonction de cela.

1.1.1 Position du logiciel dans le système

Le jeu s'intègre dans un ensemble de classes sous java. Le jeu sert à analyser et voir se développer des individus de manière autonome. Le jeu est composé de plusieurs classes qui communiquent entre elles. Nous avons donc 9 classes principales (**Personne**, ***Manifestants***, ***CRS***, **Neutre**, **Canvas**, **Grille**, **Fenetre**, **Case**, **Main**). La classe **Personne** est la classe mère de ***Manifestants***, **Neutre** et de ***CRS***. Nous avons aussi une classe **Main** qui sert à lancer les différentes étapes du jeu et à contrôler la partie en cours. Pour finir, nous avons une classe **Grille** qui sert à organiser le plateau avec **Case.** Enfin **Fenetre** et **Canevas** qui gèrent la partie visuelle de notre programme.
Pour l'équipement externe, nous avons besoin d'une souris pour que l'utilisateur puisse interagir avec le jeu.



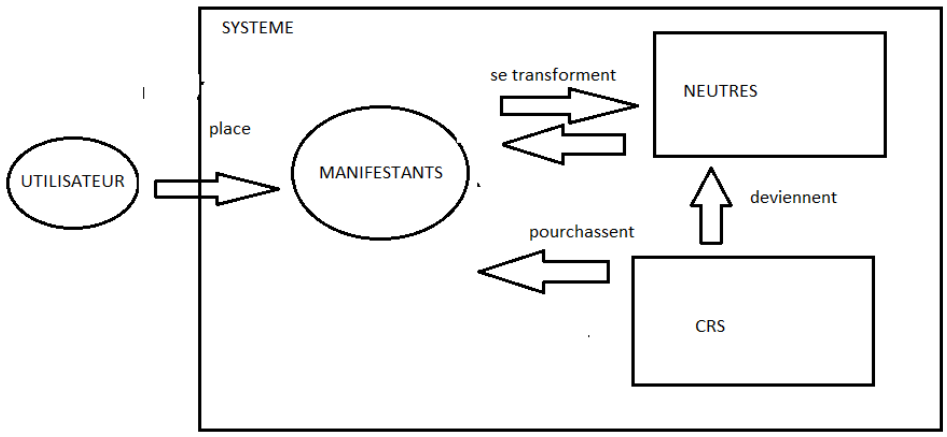
1.1.2 Fonctions générales du logiciel

Le jeu aura plusieurs fonctionnalités :

->L'utilisateur peut placer les *Manifestants* où il le souhaite sur la carte
->L'utilisateur peut arrêter la partie ou mettre en pause à tout moment

1.2 Exigences Fonctionnelles

D'un point de vue fonctionnel, le jeu doit être intuitif et facile à comprendre. L'utilisateur n'aura que des actions simples comme le positionnement des *Manifestants*, la mise en pause ou l'arrêt de la partie.
Lors du lancement du programme, le jeu affiche un plateau de jeu à l'écran avec toutes les *Neutres* et les cases vides. L'utilisateur n'a plus qu'à cliquer sur une case pour la changer en *Manifestant*. Dans ce cas, le logiciel crée un *Manifestant* et place sur les cases vides l'entité *Manifestant*. Une fois que l'utilisateur a placé tous les *Manifestants*, la partie se lance. La classe **Main** contiendra une boucle permettant à la partie de s'exécuter jusqu'aux conditions d'arrêt (précisées en introduction). Nous commençons par analyser l'évolution des cases en fonction du positionnement des personnages à proximité (cases voisines). Nous commençons par vérifier le type de la classe et ses voisins, puis nous appliquons la méthode adéquate pour rendre la case neutre ou pour qu'elle devienne un *Manifestant*. Nous incrémentons le nombre total et le nombre de *Manifestants* du groupe dans lequel il est. Si deux groupes de *Manifestants* se rejoignent, alors on les fait fusionner.
Ensuite, nous vérifions le nombre de *Manifestant* sur le plateau, en fonction du nom, nous affichons un message pour informer l'utilisateur et nous positionnons des *CRS*.
Les *CRS* se déplacent automatiquement vers le plus gros groupe de *Manifestants*. Enfin on vérifie s'il y a une rencontre entre *CRS* et *Manifestant*. Si c'est le cas, nous dispersons le groupe de *Manifestants* en plusieurs sous-groupes sinon les *CRS* deviennent neutres.
Pour finir, la boucle reprend.



1.2.1 Fonctionnalité ou cas d'utilisation

Cas d'utilisation :
Début :

1- Le joueur lance le jeu
2- La carte est générée avec des personnes neutres
3- Le joueur place des *Manifestants.*
    3b - L'utilisateur clique sur le bouton lancement pour valider ses choix.
4- La partie commence
5- Les *Manifestants* manifestent.
6- A partir d'un certain nombre de *Manifestant*, les *CRS* sont placés automatiquement sur la carte
    6b- Fenêtre d'affichage : les *CRS* arrivent
7- Les *CRS* "chassent" les *Manifestants*
8- Lorsque qu'il ne reste qu'un nombre insuffisant de *Manifestants* la partie est perdu, si on dépasse un nombre de *Manifestant* la partie est gagnée
    8b- Écran de victoire/défaite

Fin.

1.2.2 Exigence Opérationnelles :

A Environnement matériel

Matériels nécessaires :

-Souris : nécessite un seul port

-ressources demandées :

-ressources nécessaires pour lancer un exécutable (100 Mo)

-puissance de calcul (test de tous les objets) : peu de RAM nécessaire

B Environnement Logiciel

Codé avec Java SE 11.
Compilateur : java JDE
Logiciel de programmation Eclipse 2018
Logiciel de stockage : GitHub

Mise en œuvre

Le jeu est développé à des fins éducatives. Il est exploitable par le tuteur uniquement.

1.1 Performances

Pour ce jeu, les performances attendues sont celle-ci:

- Temps de réponse très court pour éviter d'avoir un jeu trop saccadé
- Temps d'affichage du plateau doit être court aussi
- Changement de classes et utilisation des méthodes doivent être courte aussi

1.2 Sécurité

Il n'y a pas de Sécurité à proprement parler. Notre jeu ne contient aucune donnée sensible et il n'a pas accès aux données sur le PC. Nous n'avons donc pas besoin d'apporter une sécurité en plus.