MANUEL DEVELOPPEUR

Romain BADAMO-BARTHELEMY

Christelle BODARD

David BUI

Alan DAMOTTE

Robin EUDES

Ombeline ROSSI

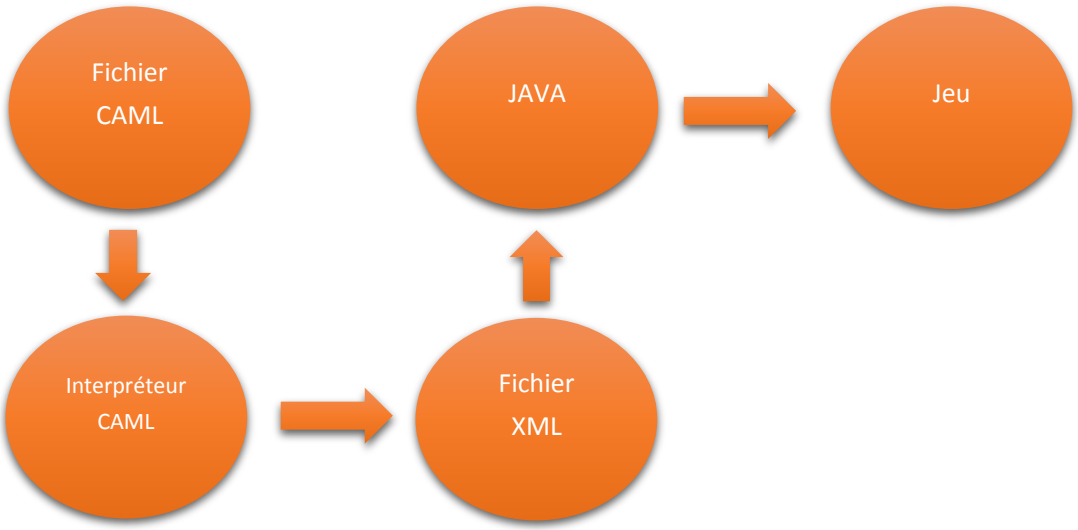
Table des matières

Introduction

Ce manuel donne toutes les informations nécessaires à la compréhension du jeu Pricman, par exemple les conventions de codage utilisées. Le jeu Pricman reprend les mêmes principes que le jeu Pacman mais avec quelques spécificités.

L'utilisateur remplit des données dans un fichier Caml afin de définir les comportements des Fantômes et Pacmans puis leur gestion se fera à partir d'algorithmes implémentés en Java. La partie pourra être visualisée par l'utilisateur via une interface graphique.

L'architecture globale de notre programme est décrite par le graphe suivant :



Partie 1

Structure du fichier Ocaml

Le fichier input.ml peut être modifié par l'utilisateur afin de créer de nouveaux automates et comportements, à condition qu'il respecte la syntaxe du langage et la grammaire utilisés. Si les modifications effectuées sont invalides, cela entraînera une erreur lors de la génération du fichier Parser.xml par le fichier parser.ml. Voici les différents noms de comportement, noms d'automates, états et identifiants, conditions et actions utilisables.

Personnage	Nom de l'automate	Nom du comportement
Pacman dirigé par le joueur	pm	PM
Pacman automatique facile	paf	PAF
Pacman automatique moyen	pam	PAM
Pacman automatique difficile	pad	PAD
Fantôme facile	fnf	FTF
Fantôme moyen	fnm	FTM
Fantôme difficile	fnd	FTD

Etats et descriptions :

Leur identifiant doit être compris entre 0 et 3.

Etats	Desciption de l'état
Avance	Se déplace dans la direction indiquée par le joueur ou choisie arbitrairement pour les personnages automatiques
Vers PacGum	Se déplace vers le PacGum le plus proche
Suit	Suit un adversaire
Fuit	Fuit un adversaire
Fuitniv2	Fuit en cherchant des échappatoires proches
Rejoint	Rejoint un objectif indiqué par un clic pour les Pacman automatiques
Chasse	Poursuit un adversaire vu par un allié

Conditions :

Conditions	Description des conditions
Rien	Ne perçoit aucune condition particulière
Voit Fantôme + Vulnérable	Voit un fantôme vulnérable
Voit Fantôme + Invulnérable	Voit un fantôme invulnérable
Voit Pacman + Vulnérable	Voit un pacman en étant soi-même vulnérable
Voit Pacman + Invulnérable	Voit un pacman en étant soi-même invulnérable
A un objectif	A reçu un ordre donné par un clic souris
PM à points	A un allié qui voit un Pacman

Les actions liées aux transitions ont les mêmes noms que les états.

Voici la grammaire à utiliser lors de la définition des comportements :

```
type nom = string ;;
type id = int ;;
type condition = string ;;
type action = string ;;
type etat = id * nom ;;
type transition = etat * etat * condition list * action list ;;
type automate = nom * transition list ;;
type comportement = nom * automate ;;
 type jeu = comportement list ;;
```

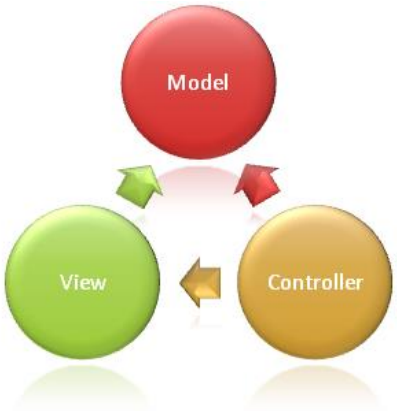
Ecriture du fichier XML

Le fichier Parser.xml se génère simplement, en utilisant la fonction ecriretout présente dans le fichier parser.ml. Celle-ci récupérera automatiquement les informations entrées par l'utilisateur dans input.ml et les traitera afin de remplir un fichier XML. Si le fichier input.ml a été correctement écrit, le fichier XML généré contiendra la liste des comportements des personnages et leurs automates. Sinon, lors de la compilation à l'aide de la commande ocaml parser.ml, une erreur indiquera quel est le type mal défini.

Partie 2

Architecture du logiciel

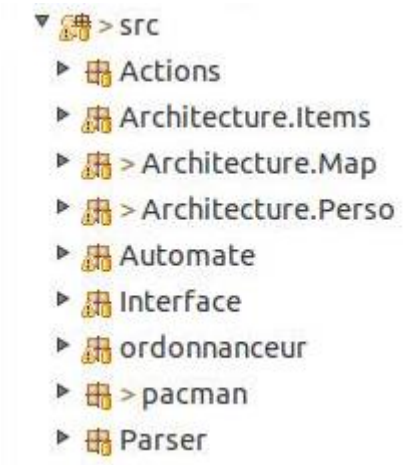
Le logiciel est basé sur une architecture MVC (Modèle-Vue-Contrôleur). Il est découpé de la façon suivante :



Le ***modèle*** assure la gestion des évènements et le fonctionnement des automates.
Le ***contrôleur*** déclenche les évènements et informe des changements au modèle.
La ***vue*** gère l'affichage du jeu.

Organisation du code

Le code est découpé en plusieurs paquets :



Interface

Ce package comprend l'intégralité des classes nécessaires à l'affichage de l'interface : le menu, la fenêtre de dialogue permettant le choix du début de partie (classes **Dialog**, **FenetreDialog**, **DialogInfo**), la fenêtre de jeu (classe **FenetreDeJeu**), la carte (classe **map**), l'affichage de la fin de jeu (classe **FinDuJeu**).
Il contient également toutes les classes permettant la gestion évènementielle des boutons (classes ***_bouton** ou ***_button**), ainsi que la classe permettant de charger les sprites (**SpriteLoader**).

Actions

Contient toutes les classes permettant de gérer les actions possibles pour les pacmans et les fantômes.

Automate

Contient les divers composants des comportements : les automates, les états, les conditions et les définitions de leurs listes. A chaque boucle de jeu, en fonction de l'état actuel des personnages, de leurs comportements et des conditions extérieures, la fonction évoluer présente dans Perso.java (dans le package Monde.Perso) sélectionnera la transition appropriée et l'exécutera. Ces comportements sont remplis lors de l'initialisation du jeu, en lisant le fichier xml grâce à Parser.java.

Monde.Items

Contient la définition des bonus et des pacgums. Chacun d'entre eux disposent d'une image ou sprite associé qui sera appelé par la méthode GetImageBonus() lorsque la carte sera redessinée chaque tour. La reconnaissance des bonus se fait ici à l'aide d'un appel à insanteof.

Monde.Map

Contient la définition des éléments de cartes basiques (vide, mur, porte, interrupteur) possédant chacun la primitive getImageCase() renvoyant l'image associé à l'élément de décor. Vide possède un attribut Bonus ainsi qu'un booléen danger appelés lors de la mise à jour de la carte à chaque tour de jeu. L'état de ces attributs influe sur l'image renvoyée par getImageCase().

La porte et l'interrupteur sont reliés de la manière suivante : la porte possède un id fixe en attribut, et à chaque fois que l'interrupteur est activé, les portes dont l'id est égal à celui présent dans Global sont ouvertes et les autres fermées. L'id de Global est ensuite changé, ce qui permet un roulement de l'ouverture des portes.

La classe **Matrice** contient l'initialisation de la position des pacmans, des fantômes, et de la carte dans la méthode carte1() faite principalement à l'aide de la méthode build().

La classe matrice s'occupe également de la gestion des interactions des personnages avec le décor et entre personnages dans la méthode maj_matrice(). Les cases sur lesquels les personnages se trouvent sont identifiées à l'aide d'instanceof, chaque type de case et de contenu dans le cas de la case Vide se traitant différemment. Dans le cas où des bonus doivent être rajoutés, il suffit de rajouter son effet associé dans la méthode maj_case_bonus (i,x,y) appelée par maj_matrice().

Monde.Perso

La classe Perso contient les primitives utiles à tous les personnages de la carte. Elle contient notamment les méthodes Avancer() qui permet d'avancer d'une case dans la direction en attribut en prenant en compte le tore, Avancer_demie_case() qui permet d'avancer plus lentement, ainsi que les méthodes indiquant la validité des cases adjacentes du perso (case_haut_valide, etc..).

Chaque personnage a un sprite associé, dans le cas du pacman, le sprite renvoyé dépend de l'état du pacman (normal, invisible, invincible) qui est défini par les booléens invisible et invincible en attribut.

Les classes descendant de Perso sont définies selon des automates différents, et disposent de primitives différentes. PacmanJoueur dispose par exemple de peut_tourner(), indiquant si la direction demandée par le clavier est accessible ou non, PacmanAuto possède chercherPacgum() renvoyant la case du pacgum la plus proche. De plus, la méthode faisant appel à la partie automate evoluer(Comportement cpt) est redéfinie pour chaque sous-classe.

Ordonnanceur

Ce package contient la classe permettant d'effectuer un tour de jeu avec la mise à jour de l'affichage et des scores. C'est également à l'intérieur de cette classe que sont effectués les appels aux calculs avant l'affichage. Cette boucle de jeu est répétée à intervalle de temps constant.

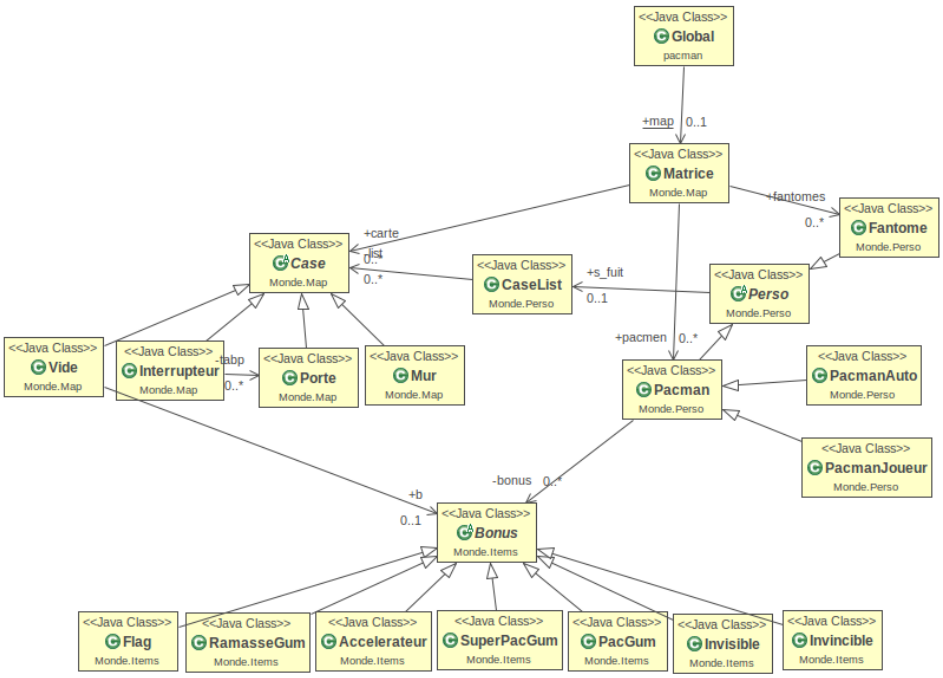
Pacman

Contient deux classes servant simplement à stocker des variables « globales » afin qu'elles soient accessibles à tout le projet. La classe **EnumDifficulte** contient une chaine de caractère initialisée en début de partie et indiquant la difficulté choisie. La classe **Global** contient des déclarations de label, d'entier, des chaines de caractères… Certains ne sont pas modifiables une fois utilisés, d'autres le sont, notamment le score des joueurs, l'affichage des bonus, etc.

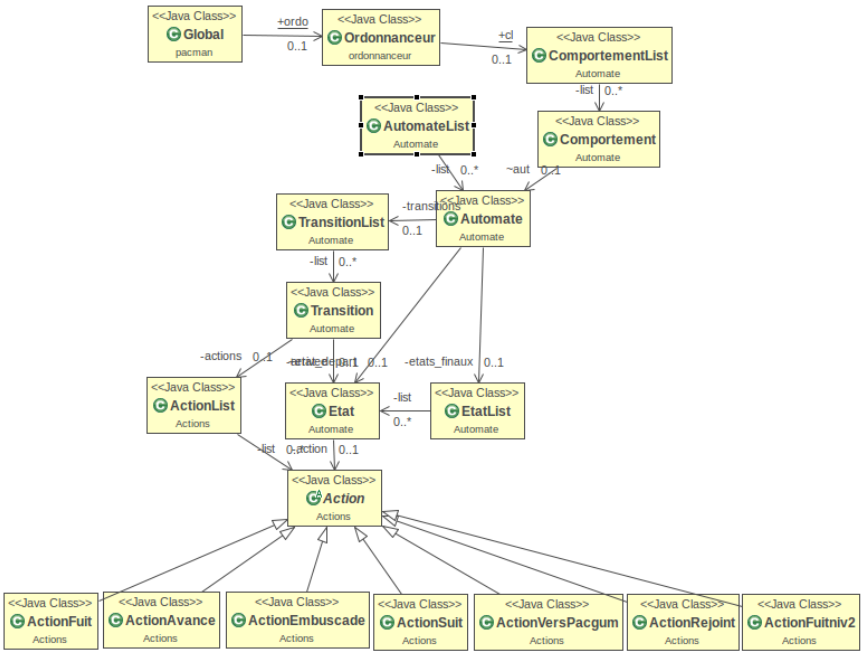
Parser

Ce package contient les classes effectuant le parser du fichier XML vers java.

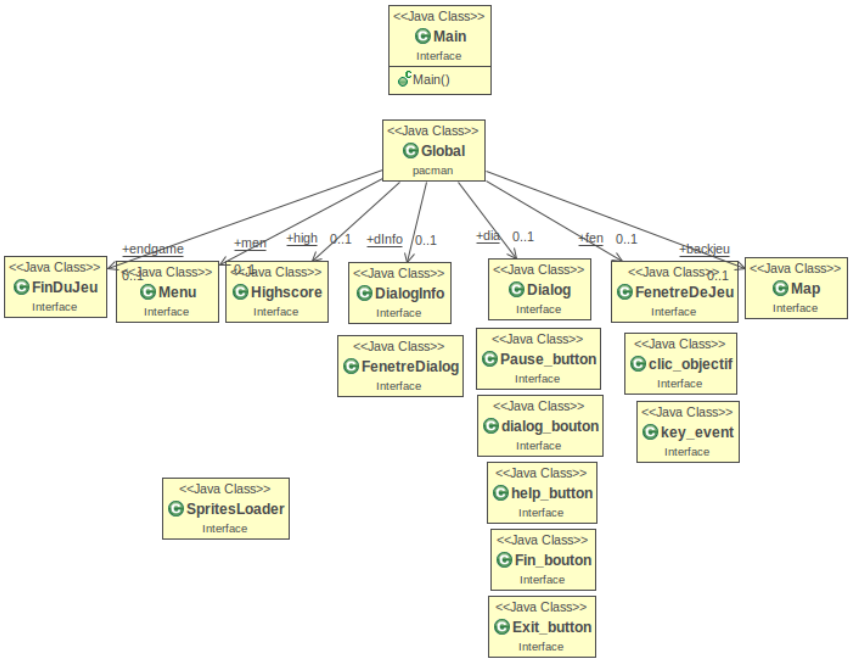
Hiérarchie des classes



Il s'agit ici de la partie du projet gérant le monde du jeu à l'aide d'une matrice représentant chacune des cases. Une case peut être occupée par un fantôme (géré par un automate), par un automate (dirigé par un joueur ou par un automate), par un mur (ou une porte), par un bonus, ou par une pacgum.



Il s'agit ici de la partie du projet qui gère les automates des fantômes et des pacmans ainsi que les actions qui leur sont disponibles. Elle gère leur comportement en fonction des automates en entrée et des transitions disponibles.



On retrouve ici la gestion de l'interface du jeu. C'est dans cette partie qu'est initialisée toute l'interface graphique comprenant le menu, la fenêtre de dialogue permettant le choix en début de partie, la fenêtre principale de jeu et enfin la fenêtre de fin de jeu. C'est également dans cette partie du projet que ce trouve toutes les classes permettant la gestion évènementielle des boutons et des clics, ainsi que la classe permettant de charger les sprites des différents fantômes, pacmans, bonus, etc…

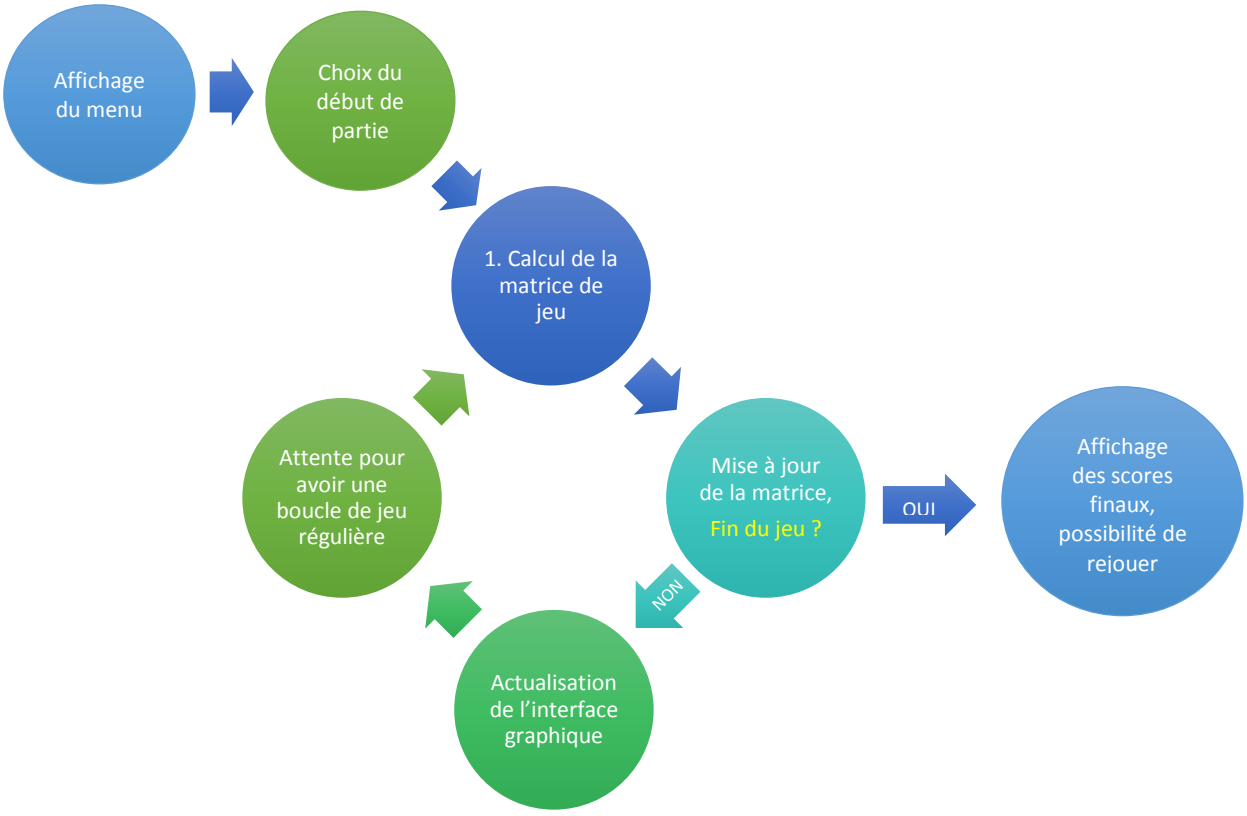
Principe de fonctionnement du jeu

Lorsque qu'un joueur débute une partie, plusieurs actions s'enchainent :

1. Lecture du fichier XML
2. Création des automates correspondants dans java
3. Initialisation du jeu (affichage) et de la matrice de jeu
5. Affichage du monde et contrôle du jeu
6. Mise à jour des états et des actions

A chaque pas de temps, l'application effectue une boucle qui consiste à effectuer les calculs puis mettre à jour l'interface.

Le fonctionnement de l'application peut être résumé de la manière suivante :



Ajout de fonctionnalités

Ajout de nouvelles cartes :

Toute nouvelle carte devra être implémentée dans la classe matrice à l'aide de la méthode build et des paramètres appropriés. Le jeu contient déjà 2 cartes : la première pour le mode facile, la seconde pour les modes moyen et difficile.

Ajout d'automates

Il est possible d'ajouter de nouveaux automates à condition de respecter la syntaxe précisée avant.