Jennifer Salle & Samuel Rollet

Projet de Programmation Orientée Objet
Seconde Année

Rapport Final

Ecole Supérieure d'Ingénieurs de Luminy
Université de la Méditerranée - Aix Marseille II
Janvier 2007

Bordereau de Livraison

REDACTEUR / *COMPILED BY :*	**Jennifer Salle & Samuel Rollet**
DATE :	**18/01/2007**
REFERENCE :	**RapFin PacMan 2007**

TITRE / *TITLE* :	**Rapport Final**

VISA / *VALIDATION*

DIFFUSION / DISTRIBUTION LIST

Samuel Rollet
Jennifer Salle
Marc Daniel
Sébastien Mavromatis

Table des Matières

1. INTRODUCTION

1.1 Objectif

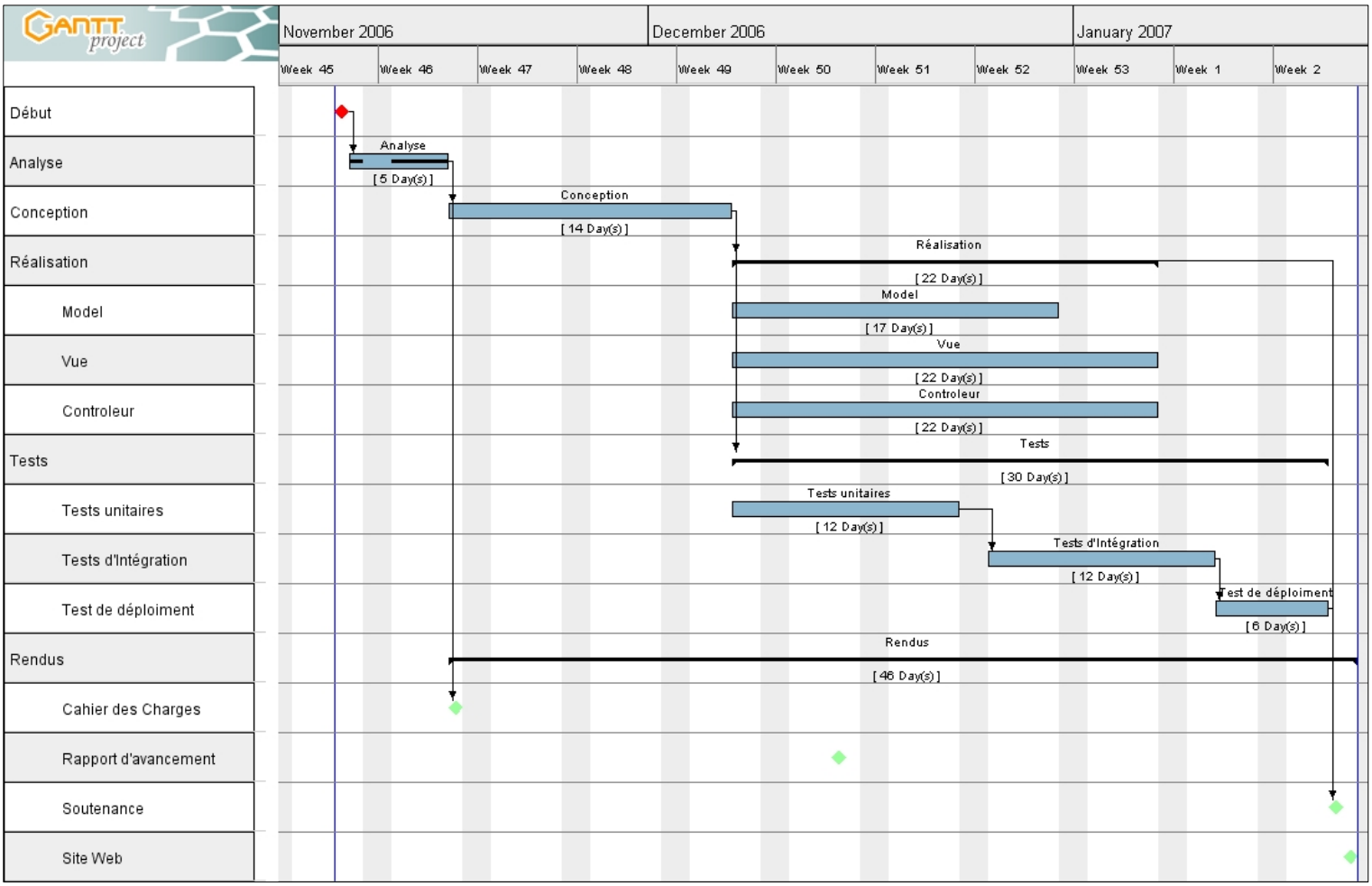
Les jeux de labyrinthes sont des jeux vidéo très classiques qui consistent à se déplacer dans un labyrinthe peuplé de créatures plus ou moins hostiles et plus ou moins sophistiqués. L'objectif de ce projet était la réalisation d'un tel jeu inspiré du classique PacMan, avec de nombreuses options absentes du jeu d'origine. Ce projet est désigné sous le nom de PacMan 2007.

1.2 Contexte

Dans le cadre du projet de programmation orienté objet de seconde année du département informatique de l'Ecole Supérieure d'Ingénieur de Luminy, nous avons réalisés ce projet qui nous a permis de mettre en œuvre les connaissances que nous avons acquises lors des cours et des TP de Programmation Orientée Objet, de Conception Orientée Objet et d'Infographie. Pour cela nous avons travaillés en binôme, celui-ci était constitué de Jennifer Salle et Samuel Rollet. Les enseignants chargés d'encadrer ce projet étaient M. Marc Daniel et M. Sébastien Mavromatis.

2. PLANNING

2.1 Planning Initial



2.2 Enchaînement des Etapes du Projet

La réalisation de ce projet s'est divisée en trois principales phases :

- Une première phase d'analyse du problème, qui s'est soldée par la remise du Cahier de Charges le 17 Novembre 2006.

- Une seconde phase de conception, pendant laquelle nous avons décomposé le programme en modules, définit les relations entre les modules et définit leur constitution interne. Cette phase sera réalisée en utilisant les différents diagrammes UML.

- Une troisième phase de réalisation des objectifs fixés dans le cahier des charges en s'appuyant sur les résultats de phase de conception. Cette troisième phase s'est divisée en trois parties :

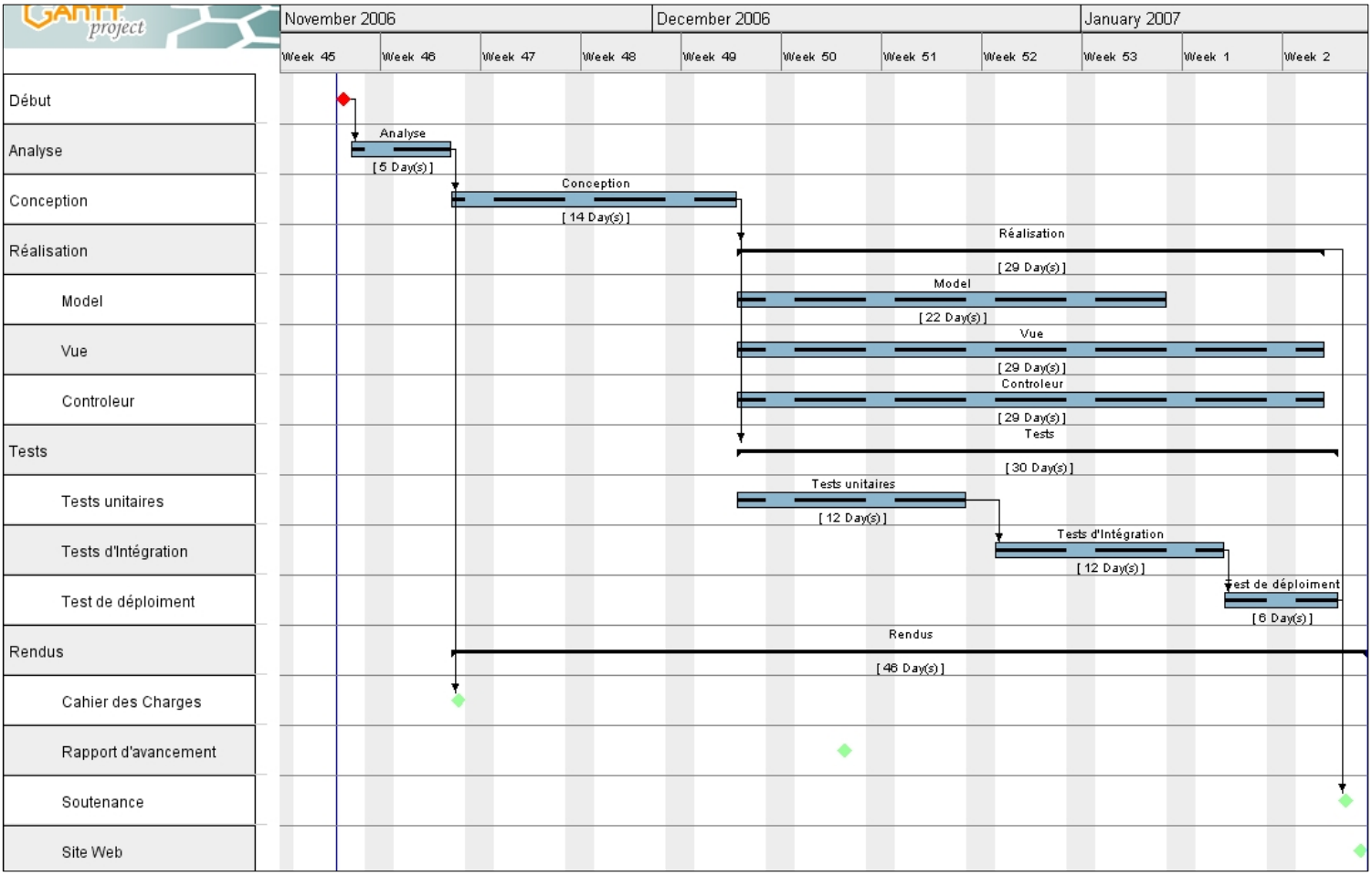
- Model : constitué des classes qui représentent les données de l'application.
- Vue : constitué des éléments qui se chargent de la représentation visuelle des données (interface).
- Contrôleur : qui gère les données, le déroulement du programme et commande la vue.

Une phase de test s'est déroulée en parallèle de la phase de développement. Elle s'est découpée en trois parties dépendantes de l'avancement de la phase de réalisation :

- Tests unitaires : qui permettent de tester l'intégrité de chaque modules.
- Tests d'intégration : pour valider l'interaction entre les modules dépendants.
- Tests de déploiement : pour valider le bon fonctionnement du programme.

L'objectif étant d'acquérir des connaissances sur les différentes technologies utilisées durant le développement, l'ensemble des tâches a été réalisé en binôme pour favoriser une meilleure compréhension du produit fini. Les éventuelles parties développées individuellement ont fait l'objet d'une explication de la personne en charge à l'autre membre de l'équipe.

2.3 Planning Effectif



On peut observer un allongement de la phase de réalisation du projet. Cela s'explique par l'importance des différents modules prévus pendant la phase de conception, et par la mobilisation de notre équipe sur d'autres projets simultanément.
Cependant, nous sommes en mesure de livrer le projet à la date prévue.

3. CONCEPTION

Cette partie du rapport présente le résultat de la phase de conception effectuée pendant ce projet. Il rappelle tout d'abord les règles du jeu qui ont étés utilisés pour concevoir les structures de données. Il présente ensuite les différents états du jeu, qui ont été déterminant dans l'organisation de ces mêmes structures. Il expose enfin le model de conception utilisé et les différentes structures de données qui en ont résulté.

3.1 Règles du jeu

Le PacMan 2007 permet à l'utilisateur de se divertir avec ce jeu de labyrinthe évolué.

Le but du jeu est de gagner un maximum de points à l'aide du personnage contrôlé par le joueur nommé PacMan. Pour cela, il évolue dans un labyrinthe où il doit ramasser des éléments disséminés, tout en évitant de rencontrer ses ennemis, les fantômes, qui se déplacent de façon intelligente sur le plateau de jeu.

Au début d'une nouvelle partie le PacMan dispose de trois vies. Les fantômes sont produits dans la crypte. Il ne sortent que individuellement de celle-ci.

Par défaut les fantômes (bleu) peuvent manger le PacMan, ce qui lui fait perdre une vie. Le joueur doit alors tout faire pour les éviter. Pour chaque vie perdue, l'ensemble des personnages retrouve leurs positions initiales du niveau courant. Quand le PacMan perd toutes ses vies, la partie est terminée. Si le PacMan mange un big PacGum, il peut manger les fantômes, devenus bleus, pendant une période donnée ce qui augmente son score de 1000 points par fantôme. Pendant cette période les fantômes cherchent à fuir le PacMan. Lorsqu'ils sont mangés par le PacMan, ils deviennent gris et doivent revenir à la crypte pour se régénérer. De plus, lorsque le PacMan mange un Big PacGum, le score augmente de 100 points.

A chaque fois que le PacMan accumule 10.000 points il gagne une vie supplémentaire.

Les différents personnages peuvent utiliser les téléporteurs pour passer d'un téléporteur à l'autre. Les destinations des téléporteurs sont aléatoires et ne sont pas garantis.

Le PacMan peut également collecter des bonus qui modifient son comportement pendant une durée limitée. Les bonus sont les suivants :

- Le **passe muraille** : lui permet de traverser les murs.
- Le **drap** : lui permet de ne pas être mangé par les fantômes.
- Le **speed** : lui permet d'accélérer.
- Le **lance roquettes** : lui permet de tuer les fantômes à distance.
- La **grenade :** lui permet de poser une bombe capable de détruire les murs.
- La **surprise** : bonus aléatoire avec l'effet d'un bonus précédent, ou sans effet.

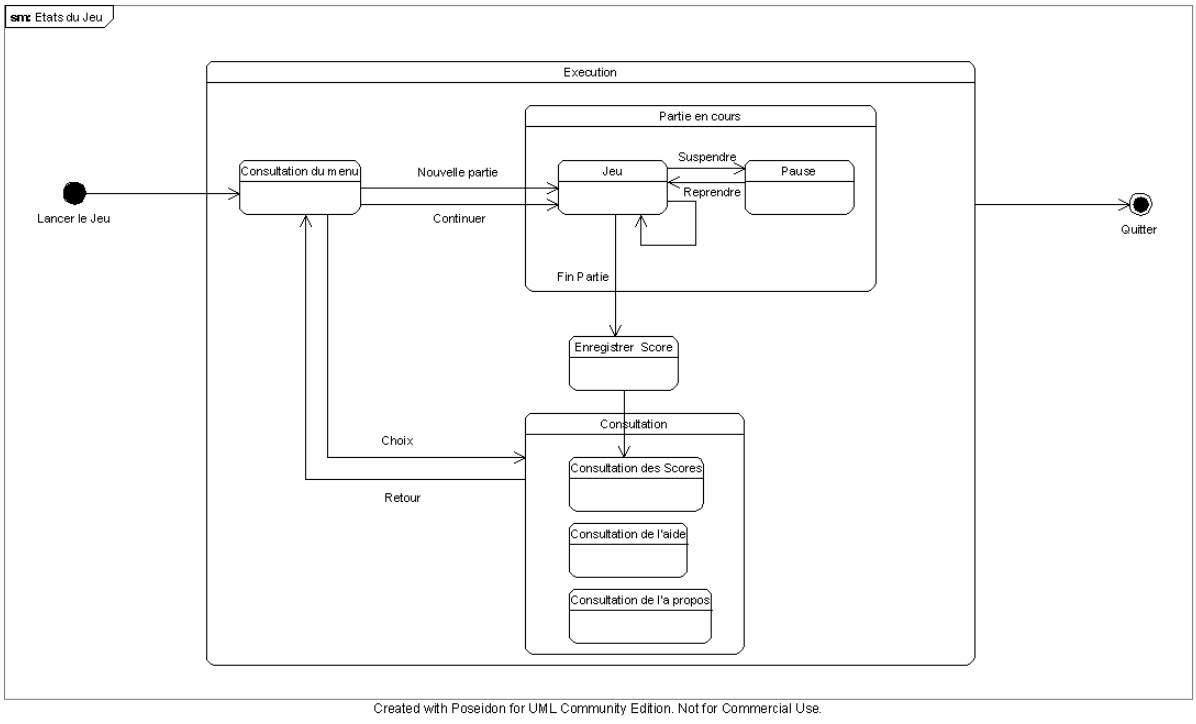
Les fantômes peuvent eux aussi collecter certains éléments afin d'augmenter leurs compétences individuels. Les bonus maléfiques sont les suivants :

- Le **bad speed** : permet au fantôme qui le ramasse d'accélérer pendant une durée limitée.
- Le **missile** : permet au fantôme de tuer le PacMan à distance.
- Le **cloneur** : permet à un fantôme de se dédoubler. Son clone reste sur le plateau jusqu'à ce que le PacMan meurt.

Les bonus sont affichés aléatoirement et pendant un temps donné.

Pour augmenter son score, le PacMan doit collecter les PacGums. Pour chaque PacGum, le score augmente de 10 points. Lorsque l'ensemble des PacGums du plateau a été ramassé, le jeu passe au niveau supérieur. Chaque niveau offre une configuration différente du labyrinthe. De plus, la difficulté augmente à chaque niveau : intelligence des fantômes, vitesse, structure du labyrinthe.

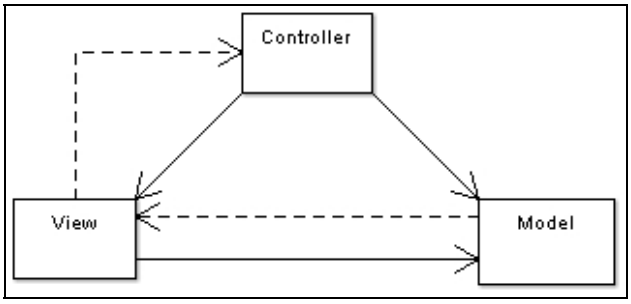
3.2 Etats du jeu



3.3 Model de Conception

Le modèle de conception MVC (décrit par Xerox pour le langage Smalltalk) vise à séparer les éléments d'une application en trois unités : le Model, la Vue, et le Contrôleur; ce qui a pour objectif de séparer le traitement, le stockage et la présentation des données. Ce modèle de conception permet la mise en place d'applications bien structurées, et faciles à maintenir, en effet, les modifications sont

indépendantes et plus aisées, aussi bien au niveau du traitement que de la présentation.
Cette architecture sépare les applications graphiques en trois parties :
- le Model, qui représente les données de l'application
- la Vue, qui est la représentation visuelle des données du Model, et transmet les actions de l'utilisateur aux Contrôleurs.
- le Contrôleur, qui effectue le traitement des données et des actions de l'utilisateur.



Nous avons décidé de suivre ce model pour développer PacMan 2007. Ainsi on peut diviser les classes développées en trois groupes :

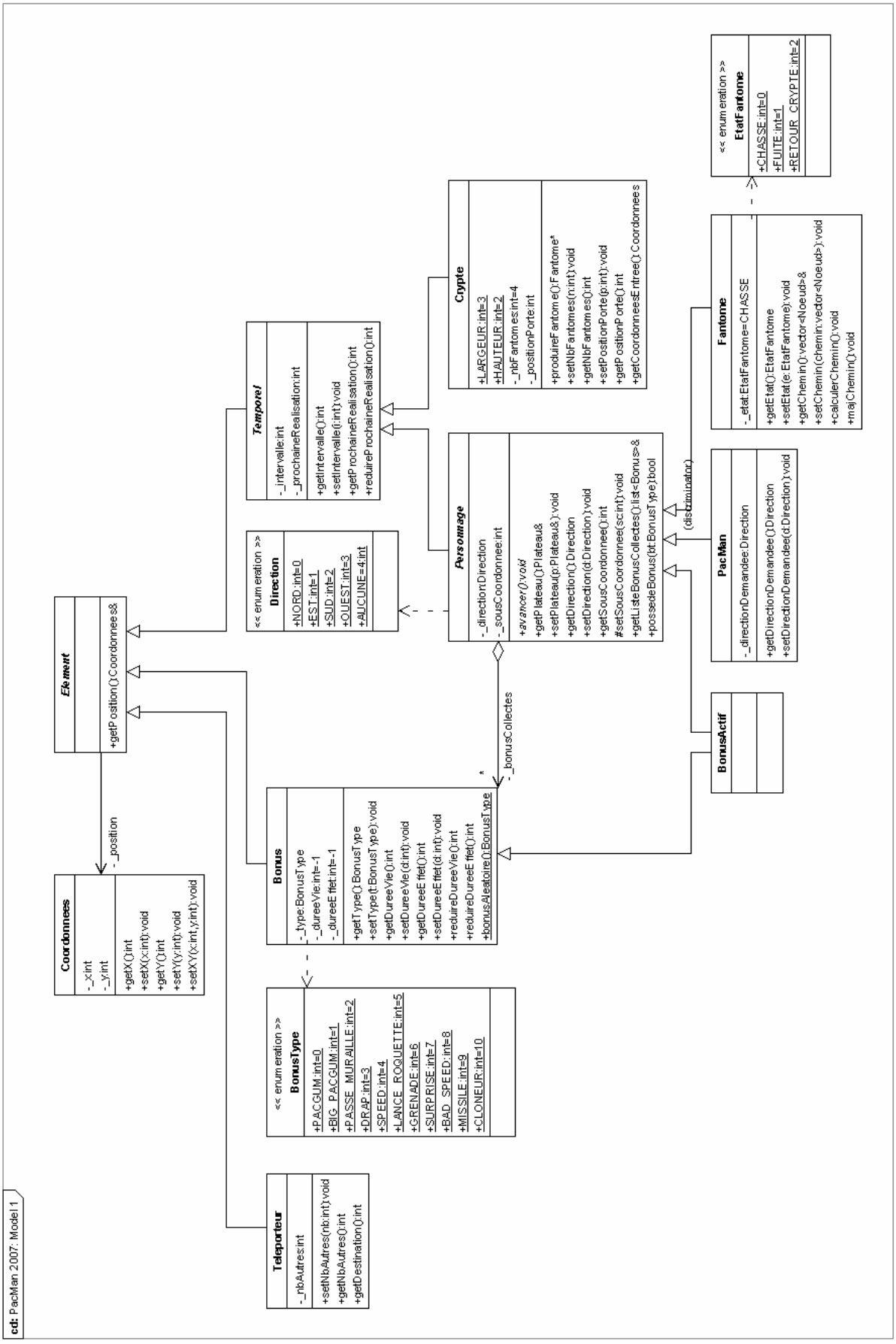
- Pour la vue, les classes dont les noms sont de la forme *Afficheur….cpp* chargées d'afficher les différents écrans et les composants du jeu. Ces classes exécutent principalement du code GLUT et OpenGL.
- Pour le contrôleur, les classes dont les noms sont de la forme *Controleur….cpp*, chargées de gérer l'exécution du jeu. Elles sont chargées de contrôler l'évolution des données du jeu au cours du temps.
- Pour le model, les autres classes, dont le nom correspond à l'élément du jeu où à une généralisation de ceux-ci.

La structure des classes est présentée dans la partie suivante, suivant les trois parties Model, Vue et Contrôleur.

3.4 Structures de Données

Les diagrammes présentés ci-après sont disponibles dans le dossier *UML/* des documents livrés avec le projet pour une consultation plus détaillée.

3.4.1 Model


cd: PacMan 2007: Model 1
Element
+getPosition():Coordonnees&
Coordonnees
-_x:int
-_y:int
+getX():int
+setX(x:int):void
+getY():int
+setY(y:int):void
+setXY(x:int,y:int):void
-_position
Teleporteur
-_nbAutres:int
+setNbAutres(nb:int):void
+getNbAutres():int
+getDestination():int
Bonus
-_type:BonusType
-_dureeVie:int=-1
-_dureeEffet:int=-1
+getType():BonusType
+setType(t:BonusType):void
+getDureeVie():int
+setDureeVie(d:int):void
+getDureeEffet():int
+setDureeEffet(d:int):void
+reduireDureeVie():int
+reduireDureeEffet():int
+bonusAleatoire():BonusType
<< enumeration >>
BonusType
+PACGUM:int=0
+BIG_PACGUM:int=1
+PASSE_MURAILLE:int=2
+DRAP:int=3
+SPEED:int=4
+LANCE_ROQUETTE:int=5
+GRENADE:int=6
+SURPRISE:int=7
+BAD_SPEED:int=8
+MISSILE:int=9
+CLONEUR:int=10
*
-_bonusCollectes
Temporel
-_intervalle:int
-_prochaineRealisation:int
+getIntervalle():int
+setIntervalle(i:int):void
+getProchaineRealisation():int
+reduireProchaineRealisation():int
<< enumeration >>
Direction
+NORD:int=0
+EST:int=1
+SUD:int=2
+OUEST:int=3
+AUCUNE=4:int
Personnage
-_directionDirection
-_sousCoordonnee:int
+avancer():void
+getPlateau():Plateau&
+setPlateau(p:Plateau&):void
+getDirection():Direction
+setDirection(d:Direction):void
+getSousCoordonnee():int
+setSousCoordonnee(sc:int):void
#setSousCoordonnee(sc:int):void
+getListeBonusCollectes():list<Bonus>&
+possedeBonus(bt:BonusType):bool
Crypte
+LARGEUR:int=3
+HAUTEUR:int=2
-_nbFantomes:int=4
-_positionPorte:int
+produireFantome():Fantome*
+setNbFantomes(n:int):void
+getNbFantomes():int
+setPositionPorte(p:int):void
+getPositionPorte():int
+getCoordonneesEntree():Coordonnees
(discriminator)
PacMan
-_directionDemandee:Direction
+getDirectionDemandee():Direction
+setDirectionDemandee(d:Direction):void
Fantome
-_etat:EtatFantome=CHASSE
+getEtat():EtatFantome
+setEtat(e:EtatFantome):void
+getChemin():vector<Noeud>&
+setChemin(chemin:vector<Noeud>):void
+calculerChemin():void
+majChemin():void
<< enumeration >>
EtatFantome
+CHASSE:int=0
+FUITE:int=1
+RETOUR_CRYPTE:int=2
BonusActif

Ce premier diagramme présente les différents composants du plateau.

3.4.1.1 Eléments

Tous ces composants ont des propriétés communes regroupées dans la classe abstraite *Element*.
Les *Element*s se caractérisent par leur *Coordonnees* sur le plateau de jeu.

3.4.1.2 Eléments Statiques et Intemporels

Les *Element*s qui sont statiques et ne dépendent pas du temps pendant le jeu : *Teleporteur* et *Bonus*, héritent directement de *Element*.
Les *Bonus* sont caractérisés par leur type définit dans l'énumération *BonusType*.

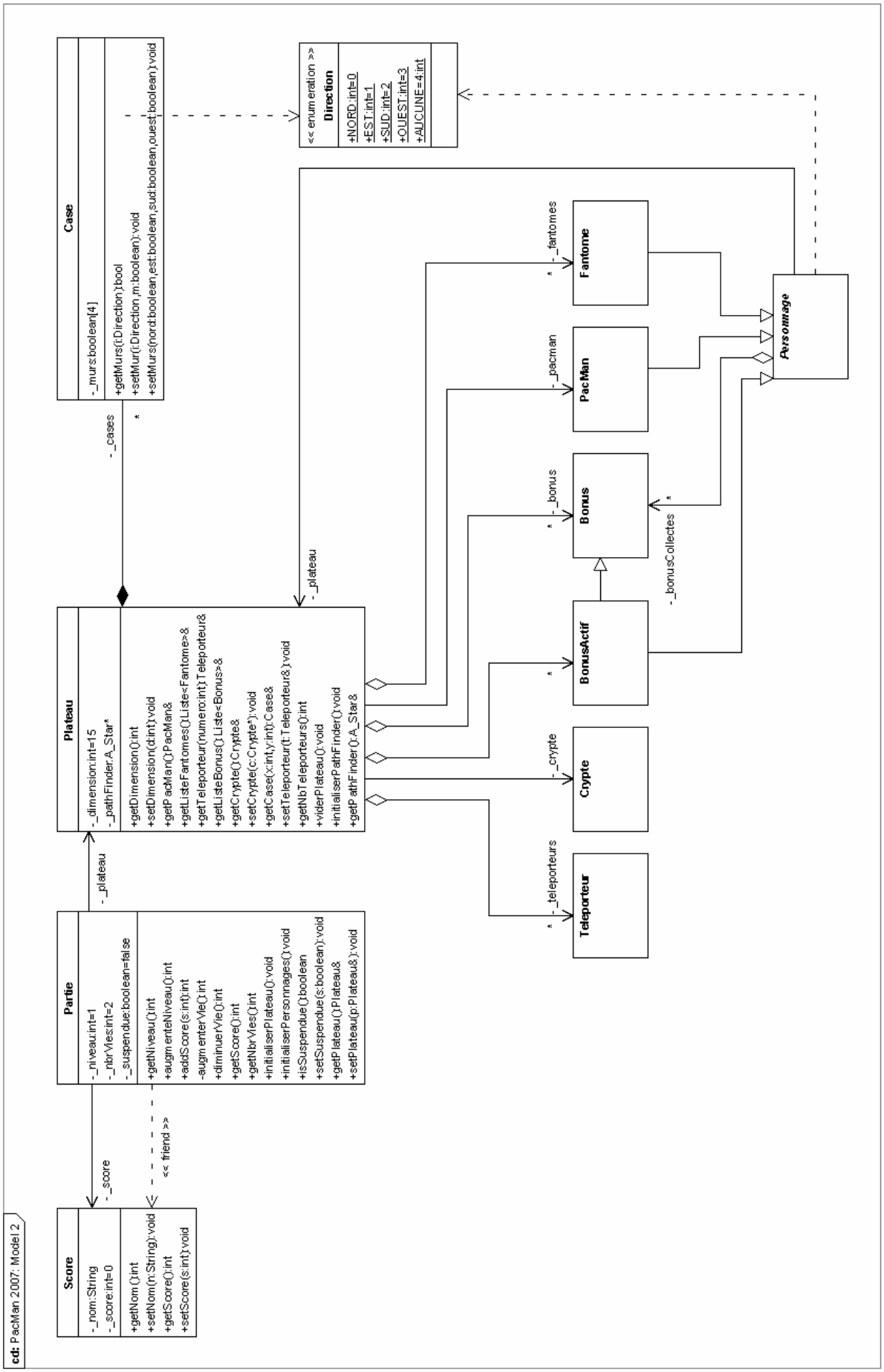
3.4.1.3 Eléments Dépendant du Temps

Tous les autres *Elements* ont des propriétés communes relatives à l'écoulement du temps, qui sont généralisées dans la classe abstraite *Temporel*.
La classe *Crypte* hérite de *Temporel* directement car il s'agit d'un élément fixe sur le plateau.

3.4.1.4 Eléments Mobiles

Les personnages sont des éléments mobiles, leurs propriétés relatives aux déplacements sont regroupées dans la classe *Personnage*.
Les personnages peuvent posséder une liste de *Bonus*, qui peut être vide. Ces *Bonus* étant ramassés sur le plateau et pas créés par le *Personnage*, le lien représentant la possession d'un *Bonus* par un *Personnage* est une agrégation.
Un personnage se caractérise par sa direction, définie dans l'énumération *Direction*, et par sa *_sousCoordonnee*, identifiant sa position á l'intérieur d'une case du plateau de jeu.
Les personnages sont : le *PacMan* et les *Fantomes*.
Les *Fantome*s se caractérisent par leur état, définit dans *EtatFantome*.
Le *PacMan* se caractérise par la direction demandée par le joueur.

Le *BonusActif* posséde à la fois les caractéristiques d'un *Bonus* et d'un *Personnage*. Il s'agit d'un bonus collecté par un personnage, et ensuite activé sur le plateau, tel une bombe. C'est donc un bonus qui dépend du temps et qui peut se déplacer.


cd: PacMan 2007: Model 2
Case
-_murs:boolean[4]
+getMurs(i:Direction):bool
+setMur(i:Direction,m:boolean):void
+setMurs(nord:boolean,est:boolean,sud:boolean,ouest:boolean):void
<< enumeration >>
Direction
+NORD:int=0
+EST:int=1
+SUD:int=2
+OUEST:int=3
+AUCUNE=4:int
-_cases
*
-_plateau
Plateau
-_dimension:int=15
-_pathFinder:A_Star*
+getDimension():int
+setDimension(d:int):void
+getPacMan():PacMan&
+getListeFantomes():Liste<Fantome>&
+getTeleporteur(numero:int):Teleporteur&
+getListeBonus():Liste<Bonus>&
+getCrypte():Crypte&
+setCrypte(c:Crypte*):void
+getCase(x:int,y:int):Case&
+setTeleporteur(t:Teleporteur&):void
+getNbTeleporteurs():int
+viderPlateau():void
+initialiserPathFinder():void
+getPathFinder():A_Star&
-_fantomes
Fantome
-_pacman
PacMan
-_bonus
Bonus
-_bonusCollectes
BonusActif
-_crypte
Crypte
-_teleporteurs
Téléporteur
Personnage
-_plateau
Partie
-_niveau:int=1
-_nbrVies:int=2
-_suspendue:boolean=false
+getNiveau():int
+augmenteNiveau():int
+addScore(s:int):int
-augmenterVie():int
+diminuerVie():int
+getScore():int
+getNbrVies():int
+initialiserPlateau():void
+initialiserPersonnages():void
+isSuspendue():boolean
+setSuspendue(s:boolean):void
+getPlateau():Plateau&
+setPlateau(p:Plateau&):void
<< friend >>
-_score
Score
-_nom:String
-_score:int=0
+getNom():int
+setNom(n:String):void
+getScore():int
+setScore(s:int):void

Ce second diagramme présente le regroupement des éléments sur le plateau de jeu.

3.4.1.5 Plateau de Jeu

Le *Plateau* est un carré dont la taille est définie dans la classe, composé de *_dimension*_dimension Case*s.
Une *Case* permet de connaître la présence de murs sur ses quatre faces à une coordonnée donnée du plateau.
Les *Case*s ne pouvant pas exister sans *Plateau*, et étant créées par celui-ci, le lien entre *Plateau* et *Case* est une composition.
Un *Plateau* supporte l'ensemble des éléments du jeu : plusieurs *Teleporteur*s, plusieurs *Fantome*s, plusieurs *Bonus*, un *PacMan*, et une *Crypte*.
Un *Plateau* se caractérise aussi par un élément *A_Star*, dont les données dépendent de la structure du labyrinthe, et qui effectue le calcul du chemin d'un point à l'autre.

On voit ici qu'un *Personnage* connaît le *Plateau* sur lequel il se trouve. Il doit en effet connaître la présence de murs sur son parcours pour pouvoir se déplacer.

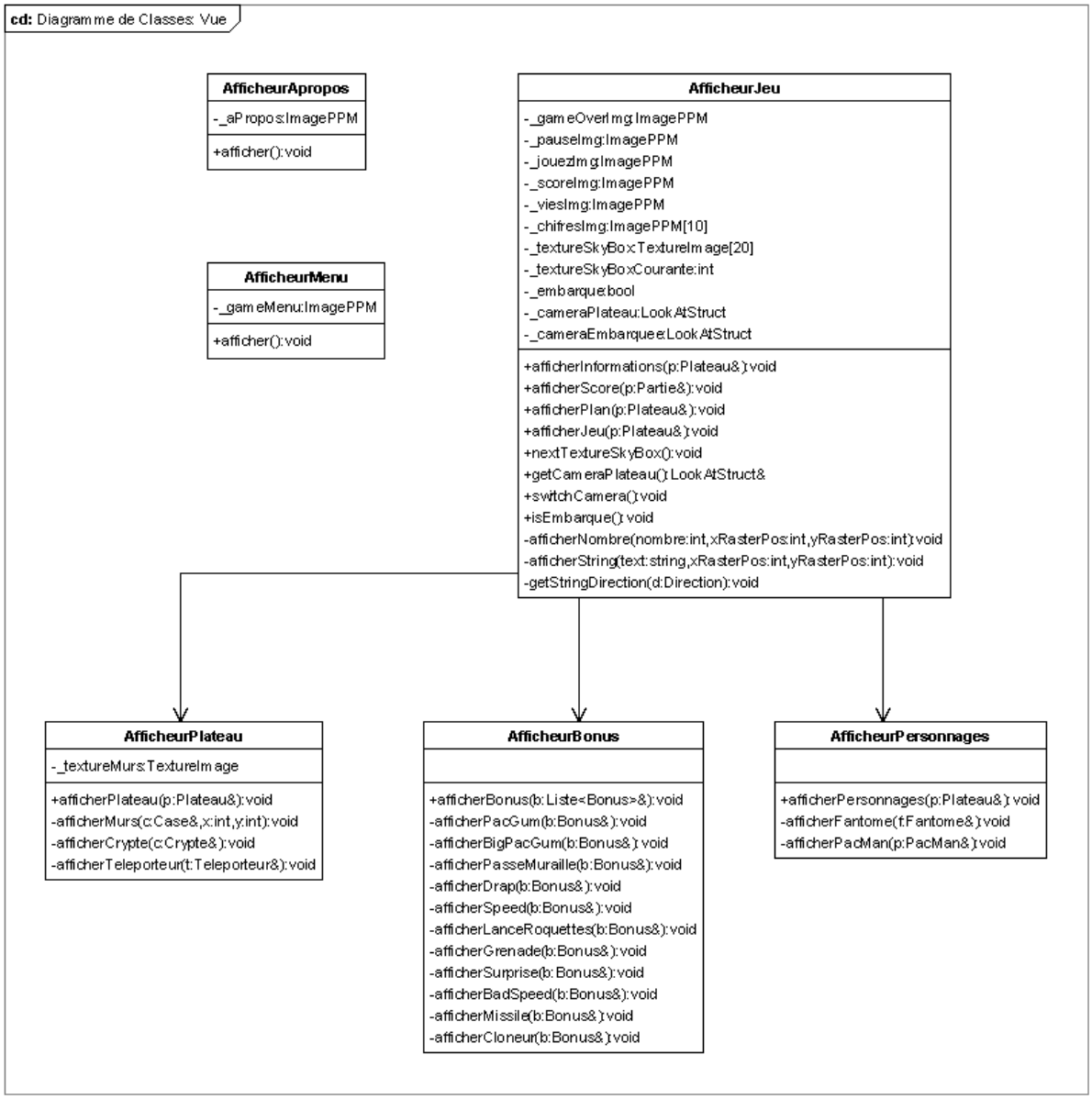
3.4.1.6 Partie

Une *Partie* regroupe les informations nécessaires à l'exécution du jeu. Elle se caractérise par un niveau, un nombre de vie, un *Plateau*, un *Score*, et un état (suspendue ou non).

La classe *Score* regroupe un score et un nom associé.

Partie modifiant en permanence le score qui lui est associé, elle est déclarée comme *friend* de *Score*.

3.4.2 Vue

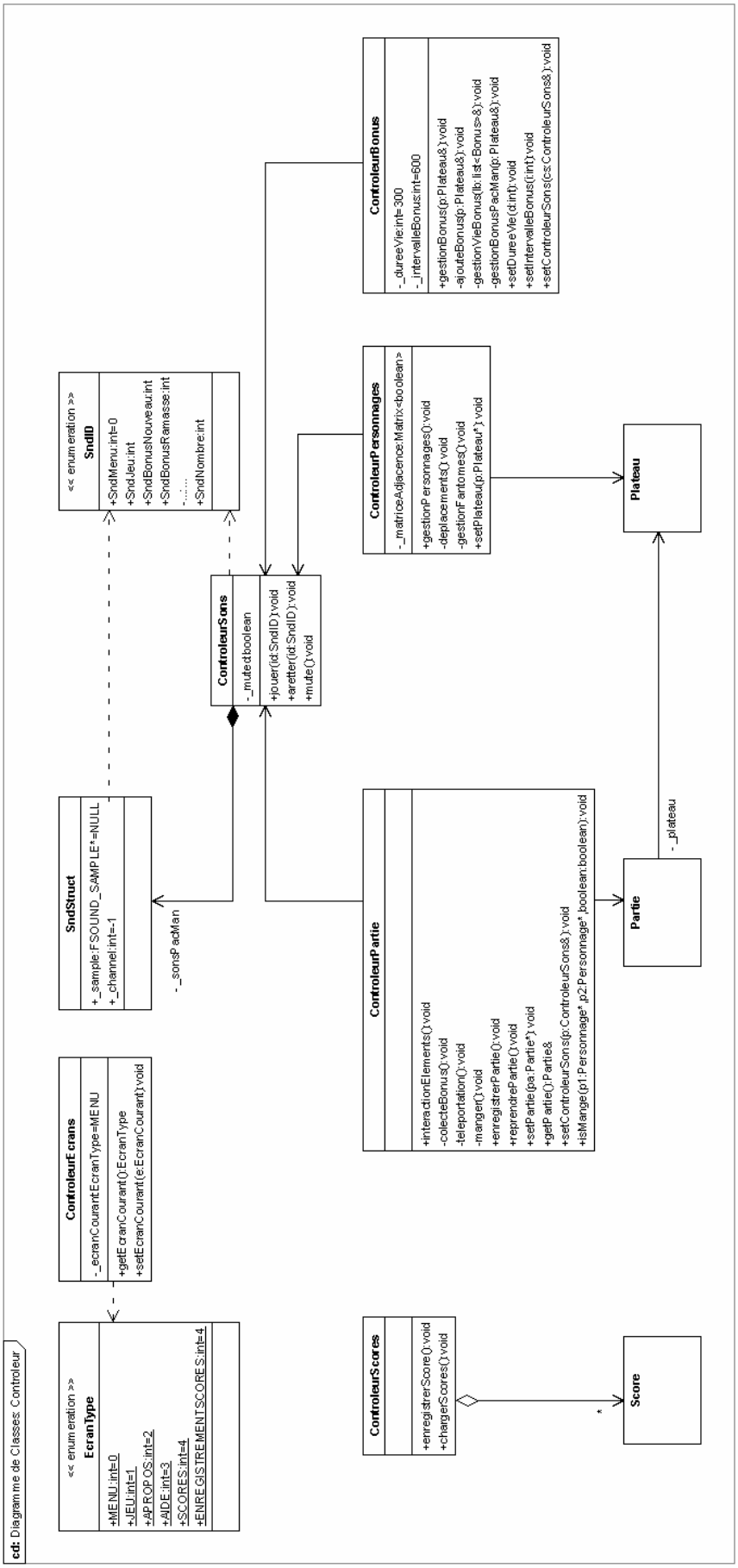


Ce diagramme présente les différentes classes chargées de l'affichage du jeu.
Chaque écran est affiché par une classe différente, celles-ci se caractérisant notamment par les images qu'elles affichent.

L'écran de jeu étant plus complexe, l'affichage des différents composants a été délégué à différentes classes.
AfficheurJeu se caractérise par ses images, mais aussi par les informations sur la skybox courante et sur la caméra : embarquée ou non.

Le plateau est affiché par la classe *AfficheurPlateau*, qui se charge de construire les murs ainsi que les éléments statiques : *Crypte* et *Teleporteur*s.
AfficheurBonus se charge d'afficher les différents *Bonus* du jeu.
AfficheurPersonnages se charge de l'affichage du *PacMan* et des *Fantome*s présent sur un *Plateau*.

3.4.3 Contrôleur



Ce diagramme présente les différentes classes chargées de contrôler l'exécution du programme, de mettre à jour les données, et de traiter les interactions avec l'utilisateur.

3.4.3.1 Sons

Le son du programme est géré par la classe *ControleurSons*. Celle-ci possède une série de sons qu'elle charge à sa création dans des *SndStruct*. Ce lien est identifié par la composition entre *ControleurSons* et *SndStruct*.
SndStruct regroupe les données d'un fichier son dans *_sample* et le numéro de canal sur lequel le son est joué.
La liste des sons de PacMan 2007 est définie dans l'énumération *SndID*.

3.4.3.2 Jeu

Le déroulement du jeu est contrôlé par plusieurs classes :

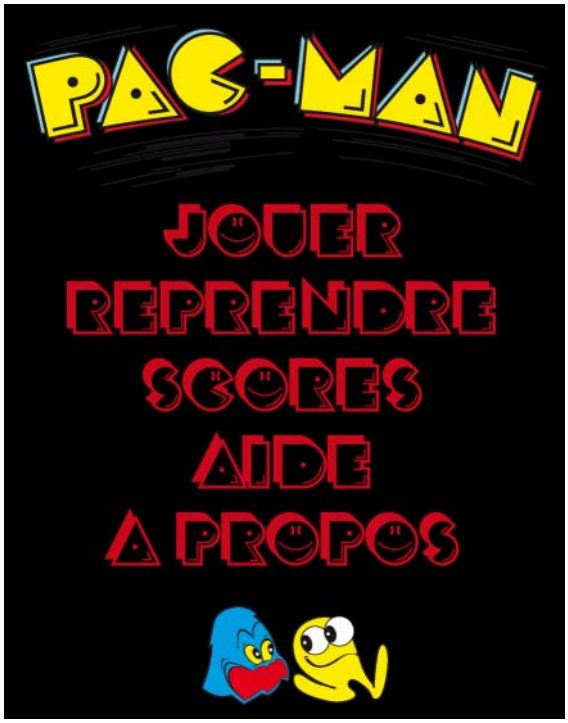
- *ControleurPartie* se charge de :
  - l'interaction entre les éléments du plateau (mort des personnages, téléportation, collecte des bonus)
  - La sauvegarde des parties
- *ControleurPersonnages* se charge du déplacement des personnages, et de la gestion des fantômes (production, retour à la crypte).
- *ControleurBonus* contrôle la présence des bonus sur le plateau : ajout de nouveaux bonus, gestion du temps d'apparition des bonus et des bonus collectés par le PacMan.

3.4.3.3 Ecrans

Le *ControleurEcrans* se charge d'identifier l'écran affiché par le jeu. Les différents écrans sont définis dans l'énumération *EcranType*.

3.5 Manuel d'utilisation

3.5.1 Menu principal



Dans le menu principal, l'utilisateur peut :

- Débuter une partie : l'utilisateur peut débuter une nouvelle partie en cliquant sur "Jouer"
- Reprendre une partie : l'utilisateur peut continuer une partie déjà commencer en cliquant sur "Reprendre"
- Consulter les meilleurs scores : l'utilisateur peut consulter les meilleurs scores enregistrés en cliquant sur "Scores"
- Afficher l'aide : l'utilisateur peut consulter l'aide en cliquant sur "Aide"
- Consulter la rubrique "A Propos" : l'utilisateur peut consulter la rubrique "A Propos" en cliquant sur "A propos"

3.5.2 Menu Score, Aide et A propos

Lorsque l'utilisateur consulte l'une des trois rubriques « Scores », « Aide » ou « A propos », celui-ci peut revenir au menu principal en cliquant sur « Retour ».

3.5.3 Pendant le jeu

Pendant le jeu, l'utilisateur peut :

- Diriger le PacMan à l'aide des touches directionnelles
- Utiliser les bonus collectés (lance roquette, grenade…) en pressant la touche « Espace »
- Mettre la partie en pause en pressant la touche « P » et revenir au jeu en appuyant une seconde fois sur cette touche
- Activer ou désactiver le son en pressant la touche « M »
- Changer la skybox en pressant la touche « B »
- Activer ou désactiver la camera embarqué en pressant « E »
- Revenir au menu principal en pressant « Echappe »

A tout moment l'utilisateur peut quitter le jeu en fermant la fenêtre. La partie en cours est alors enregistrée automatiquement afin que le joueur puisse la reprendre ultérieurement.

3.5.4 Fin du jeu

A la fin d'une partie, une fenêtre permettant d'enregistrer le score du joueur s'affiche. Si celui-ci souhaite enregistrer son score, il doit saisir son nom puis cliquer sur « OK ».

3.6 Choix Techniques

Les choix techniques énoncés dans le cahier des charges ont été respectés : utilisation du langage objet C++, avec les librairies OpenGL et GLUT pour l'interface.

La conception s'est faite suivant les règles de modélisation UML et en utilisant le logiciel Poseidon for UML.

Le développement a été effectué sous Windows en utilisant DevC++.

La librairie utilisée pour intégrer des sons dans le programme est FMOD. Nous avons effectué des tests sur les librairies OpenAL et FMOD afin de déterminer la plus performante, et cette dernière s'est avérée la plus simple à utiliser.

Les commentaires du code source ont été fait suivant la norme Doxygen pour faciliter la génération de la documentation.

4. RÉALISATION

4.1 Interfaces

L'interface entre le programme et l'utilisateur se fait de façon visuelle, sonore, et par l'intermédiaire du clavier et de la souris.

4.1.1 Interface Visuelle

L'interface visuelle se divise en plusieurs écrans :

4.1.1.1 Ecran de démarrage

Il affiche un menu proposant à l'utilisateur :
- Jouer
- Reprendre
- Scores
- Aide
- A propos

4.1. 1.2 Ecran de Scores

Il affiche la liste des meilleurs scores sauvegardés, avec le nom du joueur.

4.1. 1.3 Ecran d'Aide

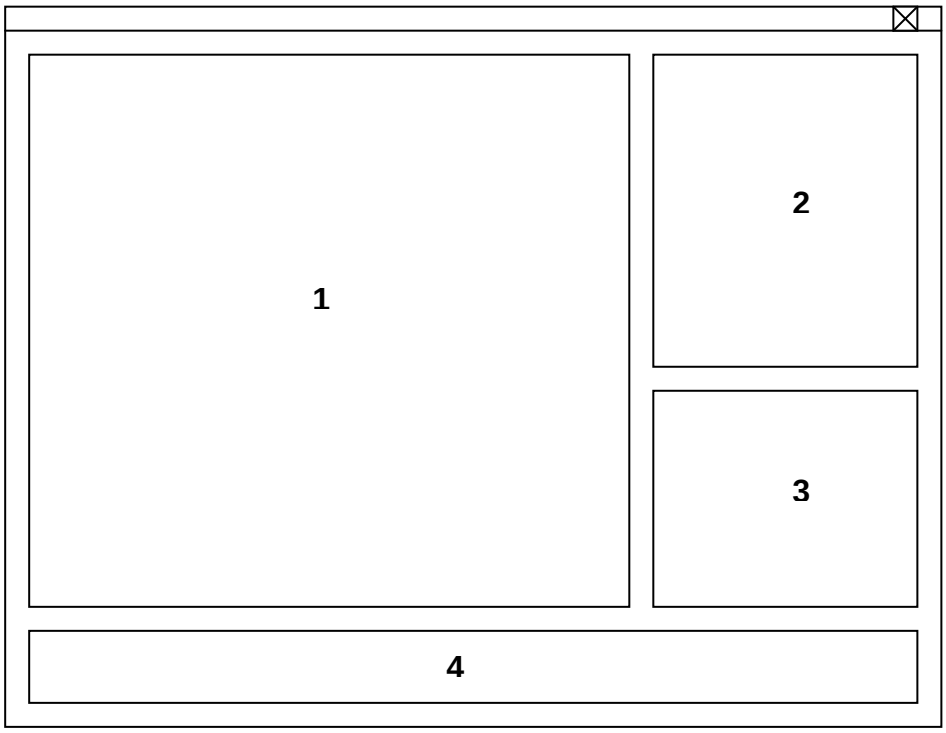
Il affiche la liste des commandes permettant à l'utilisateur de jouer.

4.1. 1.4 Ecran d'à Propos

Il affiche les informations relatives au logiciel.

4.1.1.5 Ecran de Jeu

Il se divise en plusieurs parties :



Prototype de l'interface

Partie 1 : Elle affiche le plateau de jeu en 3D.
Partie 2 : Elle affiche des informations complémentaires sur le Jeu : positions, déplacements, quantités.
Partie 3 : Affiche une miniature du labyrinthe en 2D avec la position courante du PacMan, des fantômes, des téléporteurs et des bonus.
Partie 4 : Affiche le niveau courant, le score, le nombre de vie restant au PacMan.

4.1.2 Interface Sonore

Le programme produit une musique pendant que l'utilisateur joue. De plus, il produit des sons destinés à informer l'utilisateur sur l'état de la partie en cours :

- Mort du PacMan.
- Apparition et disparition d'un bonus sur le plateau.
- Ramassage d'un bonus par les personnages.
- Fantôme mangé par le PacMan.
- Passage au niveau suivant.
- Fin de la partie.

L'utilisateur a la possibilité d'activer ou de désactiver les effets sonores.

4.1.3 Interface Clavier et Souris

Le clavier permettra à l'utilisateur de contrôler le jeu. Pendant une partie, il pourra diriger le PacMan, utiliser les bonus, stopper le son, mettre en pause, quitter.
La souris permet à l'utilisateur de naviguer dans le menu de l'écran de démarrage, ainsi que de passer de cet écran aux écrans de jeu, de scores, d'aide et d'à propos.

4.2 Algorithmes

4.2.1 Intelligence des Fantômes

Afin de rendre le jeu plus difficile et que les fantômes aient un déplacement plus réaliste, ils devaient respecter les règles suivantes :

- Quand ils chassent le PacMan : chercher à se rapprocher le plus possible de celui-ci
- Quand le PacMan chasse les fantômes : chercher à se cacher dans la partie du plateau la plus éloignée du PacMan
- Après avoir été mangés par le PacMan : retourner á l'entrée de la crypte pour être régénérés.

Pour mettre en place ces comportements, il était nécessaire d'utiliser un algorithme de recherche du plus court chemin.
Nous avons étudié les possibilités d'utiliser deux algorithmes différents : Dijkstra et A*.
Dijkstra garantit de découvrir le plus court chemin, mais est plus coûteux que A* en mémoire et en temps de calcul.
A* quand à lui est moins coûteux, mais ne garantit pas la découverte du plus court chemin. Cependant la solution trouvée avec A* est l'une des meilleures.

Le recours au calcul d'un chemin est assez fréquent dans PacMan 2007 :

- Quand un nouveau fantôme est produit
- Quand un fantôme est arrivé à la fin du chemin qu'il suivait (celui-ci étant plus court quant le fantôme est proche du PacMan, les calculs deviennent plus fréquents)
- Quand un fantôme est téléporté (sa nouvelle position étant inconsistante par rapport au chemin qu'il suivait, il doit recalculer son chemin)
- Quand un fantôme est mangé par le PacMan (il doit calculer le chemin pour retourner à la crypte.

Etant donné la nécessité de calculer régulièrement des chemins, l'algorithme A* a été retenus pour ce projet.

4.3 Description des fichiers

4.3.1 Organisation du Projet

Le code source du projet, disponible dans le dossier *src/* des documents livrés avec ce rapport s'organise de la façon suivante :

src/ Contient l'ensemble des classes du model, de la vue et du contrôleur.
Le fichier *PacMan2007.cpp* contient le point d'entrée du programme, ainsi que les fonctions de base pour le fonctionnement de GLUT (initialisation, timers, affichage)
Le fichier *PacMan2007.dev* est le fichier du projet DevC++.

src/icon/
Contient l'icône du projet pour l'exécutable windows.

src/img
Contient les images nécessaires au projet : images pour le menu et l'affichage d'informations, au format *.ppm* ; images pour les textures, au format *.tga*.

src/niveaux
Contient les fichiers détaillant la structure de chaque niveau, au format *.pm* ; leur structure est décrite plus loin.

src/outilsAStar
Contient la classe chargée du calcul des plus courts chemins utilisée pour l'intelligence des fantômes.

src/outilsImage
Contient les modules chargés de l'ouverture des images utilisées dans le projet, aux formats *.ppm* et *.tga*.

src/snd
Contient les fichiers sons utilisés par le programme.

Les fichiers sources **.cpp* débutent par une entête (nom de la classe, auteurs et description) respectant les normes d'écriture de commentaire du *Doxygen*, indiquant les inclusions les déclarations des méthodes.
Les fichiers *.h, quant à eux, contiennent les prototypes des classes et de leurs méthodes.

4.3.2 Fichiers

Chaque classe se compose de deux fichiers : un *.cpp* et un *.h*. Les fichiers des classes sont :

AfficheurAPropos.cpp
AfficheurAPropos.h

AfficheurBonus.cpp
AfficheurBonus.h

AfficheurJeu.cpp
AfficheurJeu.h

AfficheurMenu.cpp
AfficheurMenu.h

AfficheurPersonnages.cpp
AfficheurPersonnages.h

AfficheurPlateau.cpp
AfficheurPlateau.h

Bonus.cpp
Bonus.h

Case.cpp
Case.h

ControleurBonus.cpp
ControleurBonus.h

ControleurClavier.cpp
ControleurClavier.h

ControleurEcran.cpp
ControleurEcran.h

ControleurPartie.cpp
ControleurPartie.h

ControleurPersonnages.cpp
ControleurPersonnages.h

ControleurSons.cpp
ControleurSons.h

Coordonnees.cpp
Coordonnees.h

Crypte.cpp
Crypte.h

Element.cpp
Element.h

Fantome.cpp
Fantome.h

PacMan.cpp
PacMan.h

Partie.cpp
Partie.h

Personnage.cpp
Personnage.h

Plateau.cpp
Plateau.h

Score.cpp
Score.h

Teleporteur.cpp
Teleporteur.h

Temporel.cpp
Temporel.h

Des fichiers *.h* sont utilisés pour définir des constantes sous formes d'énumérations :

BonusType.h
Direction.h
EcranType.h
EtatFantome.h

Le point d'entrée du programme et les fonctions d'initialisation de GL et GLUT se trouvent dans :

PacMan2007.cpp

Des fonctions utilitaires supplémentaires pour faciliter l'utilisation d'OpenGL ont été développées et placées dans :

utilitairesGL.cpp
utilitairesGL.h

4.3.3 Fichiers de Niveaux

La structure de chaque niveau est définie dans un fichier du dossier *src/niveaux*. Chacun de ces fichiers définit les éléments présents sur le plateau et leur position.
Les fichiers comportent :

- Un magic-number permettant de confirmer la validité du fichier
- La position du PacMan sur le Plateau
- Les informations sur la crypte : nombre de fantômes, l'intervalle de production des fantômes, sa position sur le plateau, et la position de la porte sur la crypte.
- Les informations sur les téléporteurs : nombre, positions sur le plateau.
- Les informations sur les murs du labyrinthe : pour chaque case, une série de quatre 0 ou 1 pour indiquer la présence ou l'absence de mur.
- Les informations sur les bonus présents au lancement du jeu : PacGum, BigPacGum, pas de bonus.

5. RÉSULTATS

5.1 Problèmes Rencontrés

Le développement de PacMan 2007 nous a confronté à différents problèmes, aussi bien logiciels que matériels.

5.1.1 Intelligence des Fantômes

La mise en place de l'intelligence des fantômes a causé des problèmes mineurs au départ. Les fantômes trouvaient trop facilement le PacMan, et leurs chemins se superposaient.
Ce problème s'est résolu avec la mise en place des autres composants du jeu : téléporteurs, BigPacGum et autres bonus. La présence de ceux-ci modifie régulièrement la position du PacMan et la position et l'état des fantômes, évitant le problème de superposition des fantômes.

5.1.2 Inclusions Croisées

Comme présenté dans la partie sur la conception relative aux éléments, les personnages doivent connaître le plateau sur lequel ils se trouvent, notamment pour permettre leur déplacement ; et le plateau doit connaître les personnages qu'il contient.

Les spécificités techniques du C++ rendent difficile les inclusions croisées de fichiers pour les classes fortement liées. Si deux classes sont fortement liées, il est impossible de déclarer des instances de chacune dans sa classe liée.

Cela provoque des erreurs de compilation difficiles à comprendre, et nous a obligé à modifier l'implémentation de ces classes, notamment en utilisant des pointeurs à la place des références.

5.1.3 Différence avec les Prévisions

Certaines options du jeu n'ont pas pu être développées par manque de temps, notamment certains bonus, la gestion de l'enregistrement des scores et de la partie en cours.

5.2 Ecrans

PacMan 2007 se divise en deux écrans principaux, l'écran de menu qui permet de naviguer vers les autres écrans, et l'écran de jeu, qui affiche le plateau, et permet à l'utilisateur d'interagir avec les personnages.

5.2.1 Menu

On peut voir ici un aperçu de l'écran de menu.

5.2.2 Plateau de Jeu

Comme indiqué dans le chapitre réalisation, l'écran de jeu affiche :

- le plateau de jeu
- une miniature du plateau de jeu
- des informations sur l'état du jeu : score, nombre de vies
- des informations complémentaires sur les différents éléments du plateau

On peut observer ici : le PacMan en jaune, les fantômes en bleu, différents bonus (PacGum en rose, BigPacGum en rouge, Speed en bleu), la crypte, les téléporteurs.

6. CONCLUSION

Le produit fournit au client respecte les principales exigences de fonctionnement définie dans le cahier des charges ; en effet il est possible d'exécuter notre programme pour jouer à PacMan 2007.
L'importante phase de conception s'est avérée bénéfique pour la réalisation de ce projet. En effet, elle a permis un développement plus rapide et une structuration importante des différents composants.
Cependant, certaines options prévues pendant la conception n'ont pas pu être implémentées. Toutefois, cela n'empêche pas un bon fonctionnement du produit.
De plus, le planning initial a bien été respecté et nous avons pu livrer le produit à la date prévue.
Le logiciel développé pendant le projet est accessible par l'intermédiaire du site web.

Equipe de développement:

Samuel Rollet *Jennifer Salle*

ANNEXE 1
Code Sources

Les codes sources sont disponibles sous leur format original dans le dossier *sources/* des documents fournis avec le projet.

Vous pouvez consulter ci-après un exemple du code réalisé pour ce projet pour la classe PacMan.

```
/*
 * PacMan2007 - Copyright (C) 2007
 * by   Samuel Roller & Jennifer Salle
 * Full copyright notice in main.c
 */

/**
 * @file PacMan.h
 * @author Samuel Rollet
 * @author Jennifer Salle
 *
 * Entête de la classe: PacMan, classe contenant les informations
specifiques
 * au PacMan.
 */

#ifndef _PACMAN_H
#define _PACMAN_H

#include <assert.h>

#include "Direction.h"
#include "Personnage.h"
#include "Plateau.h"

using namespace std;

class PacMan : public Personnage {
      private:
            Direction _directionDemandee;

      public:
            PacMan();
            PacMan(Direction d);
            void avancer();
            Direction getDirectionDemandee();
            void setDirectionDemandee(Direction d);

            ~PacMan(){ }
};

#endif
```

```
/*
 * PacMan2007 - Copyright (C) 2007
 * by   Samuel Roller & Jennifer Salle
 * Full copyright notice in main.c
 */
/**
 * @file PacMan.cpp
 * @author Samuel Rollet
 * @author Jennifer Salle
 * @since 2007
 * @version 1.0
 *
 */

#include "PacMan.h"

#include <iostream>

PacMan::PacMan():_directionDemandee(AUCUNE){
}

PacMan::PacMan(Direction d):_directionDemandee(d){
}

void PacMan::avancer(){
      if(getDirection()!=AUCUNE || _directionDemandee!=AUCUNE){
      //quand le PacMan est au centre de la case:
            if(getSousCoordonnee()==5){
                  if(possedeBonus(PASSE_MURAILLE)){
                        //si il a demandé à tourner:
                  if(_directionDemandee!=getDirection() &&
_directionDemandee!=AUCUNE){
                              //on teste si il ne va pas sortir du plateau
                              if(   (_directionDemandee==NORD &&
getPosition().getY()<14)
                                    ||(_directionDemandee==SUD &&
getPosition().getY()>0)
                                    ||(_directionDemandee==EST &&
getPosition().getX()<14)
                                    ||(_directionDemandee==OUEST &&
getPosition().getX()>0))
                              {
                              //on le fait tourner
                              setDirection(_directionDemandee);
                              _directionDemandee=AUCUNE;
                        }
                  }
                  //si il veut continuer tout droit:
                        //on teste si il est au bord du plateau
                        if(   (getDirection()==NORD &&
getPosition().getY()==14)
                              ||(getDirection()==SUD &&
getPosition().getY()==0)
                              ||(getDirection()==EST &&
getPosition().getX()==14)
                              ||(getDirection()==OUEST &&
getPosition().getX()==0))
                        {
                              //si oui, on l'empêche d'avancer
                              setDirection(AUCUNE);
                              return;
                        }
```

```
                }
                else{
                //si il à demandé à tourner:
                if(_directionDemandee!=getDirection() &&
_directionDemandee!=AUCUNE){
                            //on teste si il n'y a pas de mur dans la
direction où il veut aller
                      if(
      !getPlateau().getCase(getPosition().getX(),getPosition().getY()).getM
ur(_directionDemandee)){
                            //on le fait tourner
                            setDirection(_directionDemandee);
                            _directionDemandee=AUCUNE;
                }
                }
                //si il veut continuer tout droit:
                      //on teste si il y a un mur devant lui
                      if(
      getPlateau().getCase(getPosition().getX(),getPosition().getY()).getMu
r(getDirection())){
                            //si oui, on l'empêche d'avancer
                            setDirection(AUCUNE);
                            return;
                }
          }
      }
          //Si le PacMan est en bord de case, on le fait passer sur la case
suivante
      if(getSousCoordonnee()==9 && (getDirection()==NORD ||
getDirection()==EST)){
            setSousCoordonnee(0);
            if(getDirection()==NORD)
                  getPosition().setY(getPosition().getY()+1);
            if(getDirection()==EST)
                  getPosition().setX(getPosition().getX()+1);
          }
      else if(getSousCoordonnee()==0 && (getDirection()==SUD ||
getDirection()==OUEST)){
            setSousCoordonnee(9);
            if(getDirection()==SUD)
                  getPosition().setY(getPosition().getY()-1);
            if(getDirection()==OUEST)
                  getPosition().setX(getPosition().getX()-1);
      }
          //sinon on le fait avancer
      else if(getDirection()!=AUCUNE){
            if(getDirection()==NORD || getDirection()==EST)
                  setSousCoordonnee(getSousCoordonnee()+1);
            else
                  setSousCoordonnee(getSousCoordonnee()-1);
            }
      }
}

Direction PacMan::getDirectionDemandee(){
      return _directionDemandee;
}

void PacMan::setDirectionDemandee(Direction d){
      _directionDemandee=d;
}
```