**Ecole Supérieure d'Ingénieurs de Luminy**
Département Informatique
Université de la Méditerranée
Luminy case 925
13 288 Marseille Cedex 09
Tél. 04 91 82 85 00
informatique@esil.univmed.fr

# Projet Programmation Orientée Objet


## Rapport final

20 décembre 2007

ANDRE Julie
DUPRAT Boris
2ème année

**Enseignants responsables :**
DANIEL Marc
MAVROMATIS Sébastien

# Sommaire

# I – Introduction

Nous avons réalisé un jeu de labyrinthe de type Pac-Man dans le cadre du projet de Programmation Orientée Objet proposé aux étudiants de deuxième année de la filière Informatique de l'ESIL.

L'objectif de ce rapport est de présenter le projet mais surtout notre travail. Il contient une description détaillée du jeu, de la phase de conception du projet, de son développement et un bilan.

Ce projet, réalisé en binôme, a pour but de nous familiariser avec la programmation orientée objet.

# II – Présentation du projet

## 1) Sujet

Nous avons réalisé un jeu de labyrinthe de type Pac-Man.

Ce projet a pour but de mettre en œuvre les connaissances acquises lors des cours et TP de Programmation Orientée Objet et Infographie. Il s'agit également de gérer des collisions entre les différents objets composant le plateau de jeu.

Pour cela, nous avons travaillé en binôme, celui-ci étant constitué de Julie André et Boris Duprat.

## 2) Règles du jeu

Notre jeu s'inspire fortement du célèbre jeu d'arcade Pac-Man, sorti par Namco en 1979. Les règles de notre jeu sont donc identiques au jeu original à quelques détails près.

Le but du jeu consiste à déplacer un personnage nommé **Pac-Man** à l'intérieur d'un **labyrinthe** afin de lui faire manger toutes les petites **pastilles** jaunes qui s'y trouvent.

*Pac-Man*

Au sein de ce labyrinthe, on trouve 4 fantômes qui sont les ennemis du Pac-Man. L'utilisateur doit donc éviter de se faire dévorer par l'un d'entre eux.
Chacun des fantômes a une personnalité qui lui est propre. Leurs stratégies pour venir à bout du Pac-Man sont donc toutes différentes :

- **Bashful** alias "**Inky**" : Timide et craintif, il s'enfuit dès que le Pac-Man s'approche d'une pastille spéciale.

- **Pokey** alias "**Clyde**" : C'est le plus imprévisible de tous. Il a tendance à fureter dans le labyrinthe au gré de ses envies sans toujours tenir compte de la position de PacMan.

- **Speedy** alias "**Pinky**" : Il communique en permanence avec ses camarades. Dès qu'un de ses trois alliés voit le Pac-Man, il est mis au courant de sa position et peut ainsi commencer sa traque.

- **Shadow** alias "**Blinky**" : C'est probablement le plus intelligent de tous. Il connait en permanence la position du Pac-Man. Ainsi, il le suit quoi qu'il arrive.

Si jamais l'un de ces fantômes aperçoit le Pac-Man, il se met immédiatement à ses trousses.

Pour lutter contre ces fantômes, le Pac-Man peut manger l'une des 4 **pastilles spéciales** présentes dans le labyrinthe. Ces pastilles, lorsqu'elles sont avalées par le Pac-Man, rendent les fantômes vulnérables durant une période de 10 secondes. L'utilisateur peut donc en profiter pour éliminer ses adversaires. Durant cette période de vulnérabilité, les fantômes craintifs n'ont qu'une chose en tête : fuir le plus loin possible du Pac-Man.
Attention, tout fantôme dévoré ressuscitera au bout d'un certain temps.

*Pastille*

*Pastille spéciale*

*Fantôme vulnérable*

Une fois l'ensemble des pastilles mangées, l'utilisateur passe au niveau suivant. La difficulté du jeu augmente avec les niveaux. En effet, plus le joueur avance dans la partie, plus la vitesse des fantômes augmente. Il devient donc de plus en plus difficile d'échapper aux féroces fantômes.

L'utilisateur doit bien évidemment tenter de réaliser le meilleur score. Il existe plusieurs moyens d'augmenter celui-ci. Une pastille rapporte 10 points, 50 points pour une pastille spéciale. Chaque fantôme dévoré rapporte quand à lui 200 points. Des **bonus** permettent également d'augmenter le score. Ceux-ci apparaissent de manière aléatoire aussi bien sur le plan spatial que temporel. Ils se matérialisent sous la forme de petits objets.

La valeur de ces bonus augmente avec les niveaux :

| Bonus | Niveaux | Points |
|---|---|---|
| Cerise  | 1 | 100 |
| Fraise  | 2 | 300 |
| Orange  | 3 - 4 | 500 |
| Pomme  | 5 - 6 | 700 |
| Cloche  | 7 - 8 | 1000 |
| Clé  | 9 - ∞ | 2000 |

Il est également possible de gagner des vies. En effet, tous les 10 000 points, une vie vous est rajoutée.

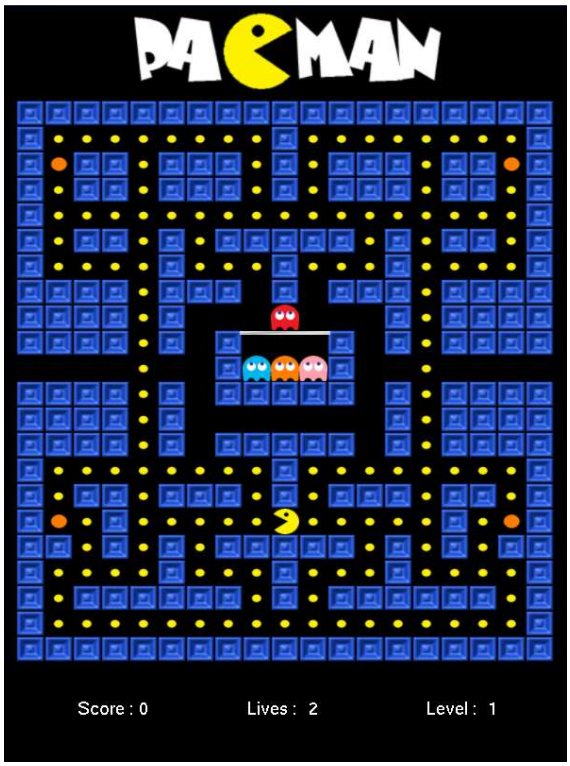
## 3) Interfaces

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

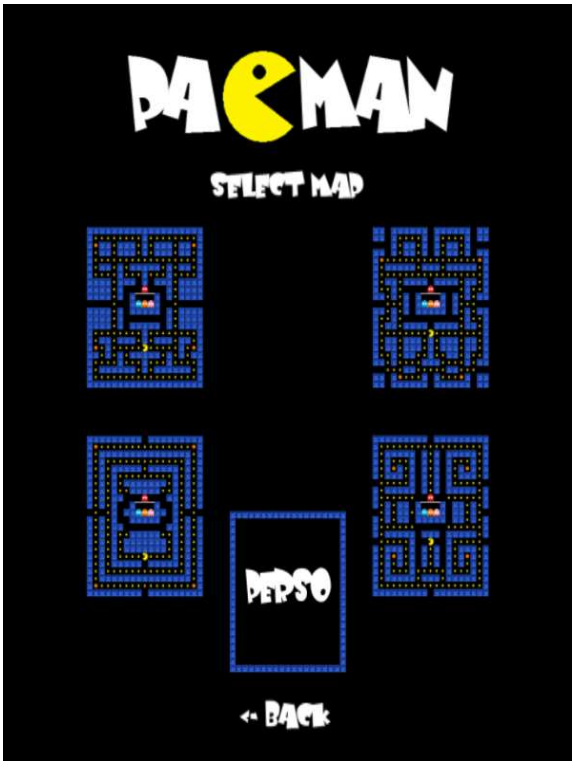
### *a) Aspect visuel*

L'interface visuelle est composée de 7 modes d'affichage constituant les 7 états du jeu.

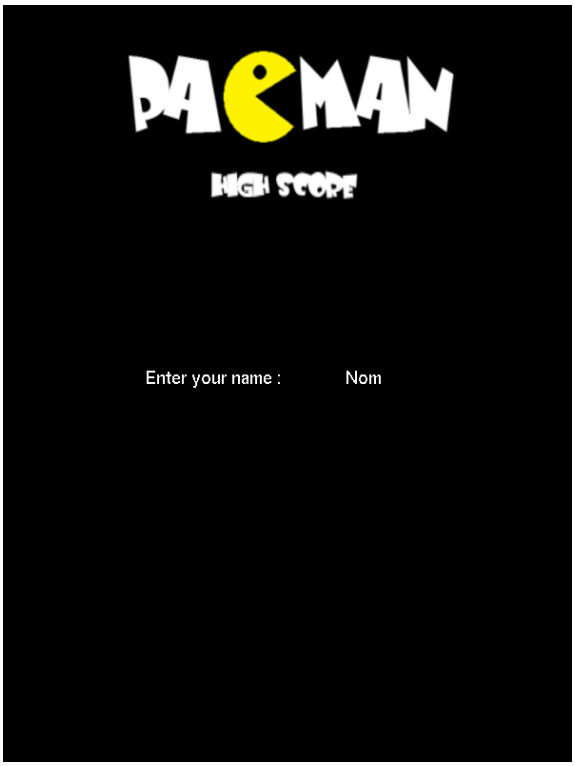

- Menu Principal : C'est ce menu qui apparait lorsque l'utilisateur lance le programme. Il permet d'accéder à l'ensemble des autres interfaces en cliquant avec la souris sur les zones concernées. L'utilisateur peut également revenir à tout moment à ce menu principal en utilisant la touche Echap du clavier.




- Partie (Start) : Interface accédée à partir du menu principal, permettant de jouer au jeu. Cet écran est composé de 3 parties : Un en-tête, le jeu en lui-même, et une zone d'informations relatives à la partie (Score, vies restantes, niveau).




- Sélection du labyrinthe (Select Map) : Ce menu permet de choisir l'un des 5 labyrinthes proposés.




- Saisie du nom : Ce menu permet à l'utilisateur de saisir son nom afin qu'il soit intégré au fichier contenant les 10 meilleurs scores. Cet interface n'apparait qu'une fois une partie terminée, et à condition que le score du joueur rentre dans le Top 10. Si cette condition est remplie, l'utilisateur est invité à saisir son nom au clavier, puis à le valider en appuyant sur la touche entrée du clavier.


- High Score : Cette partie affiche à l'écran les 10 meilleurs scores enregistrés. Elle est accessible depuis le menu principal, mais apparait également automatiquement à la fin d'une partie.


- Aide (Help) : Cette interface permet d'afficher les informations relatives au jeu comme ses règles et son mode de fonctionnement (contrôle)


- A propos (About) : Affiche les informations relatives au logiciel.

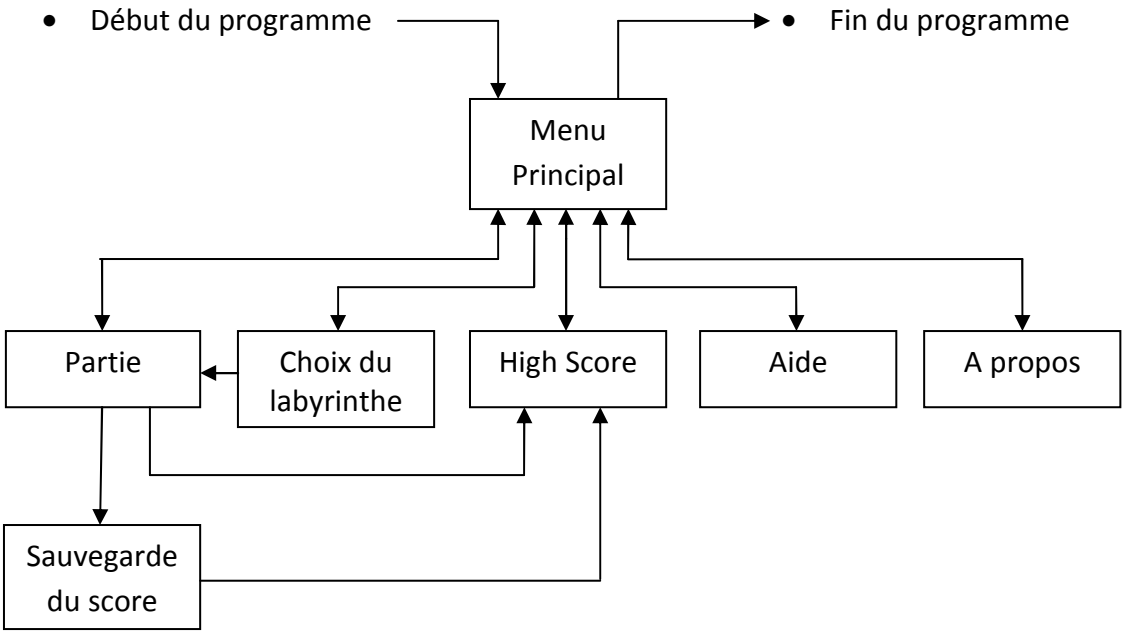



*Diagramme d'états*

### b) *Aspect contrôle*

L'utilisateur peut à tout moment interagir avec le programme à l'aide du clavier ou de la souris. La souris est utilisée pour la navigation dans les menus. Le clavier permet principalement de contrôler le Pac-Man au cours de la partie. Ce contrôle se fait à l'aide des 4 touches directionnelles ou des 4 touches Q, Z, S, D du clavier. La touche Echap permet de revenir au menu principal, ou de quitter l'application si l'utilisateur se trouve déjà à cet endroit.

## 4) Contraintes de développement

Le temps imparti pour la réalisation de ce projet étant assez court (1 mois), une organisation rigoureuse et une bonne répartition des taches a été nécessaire.

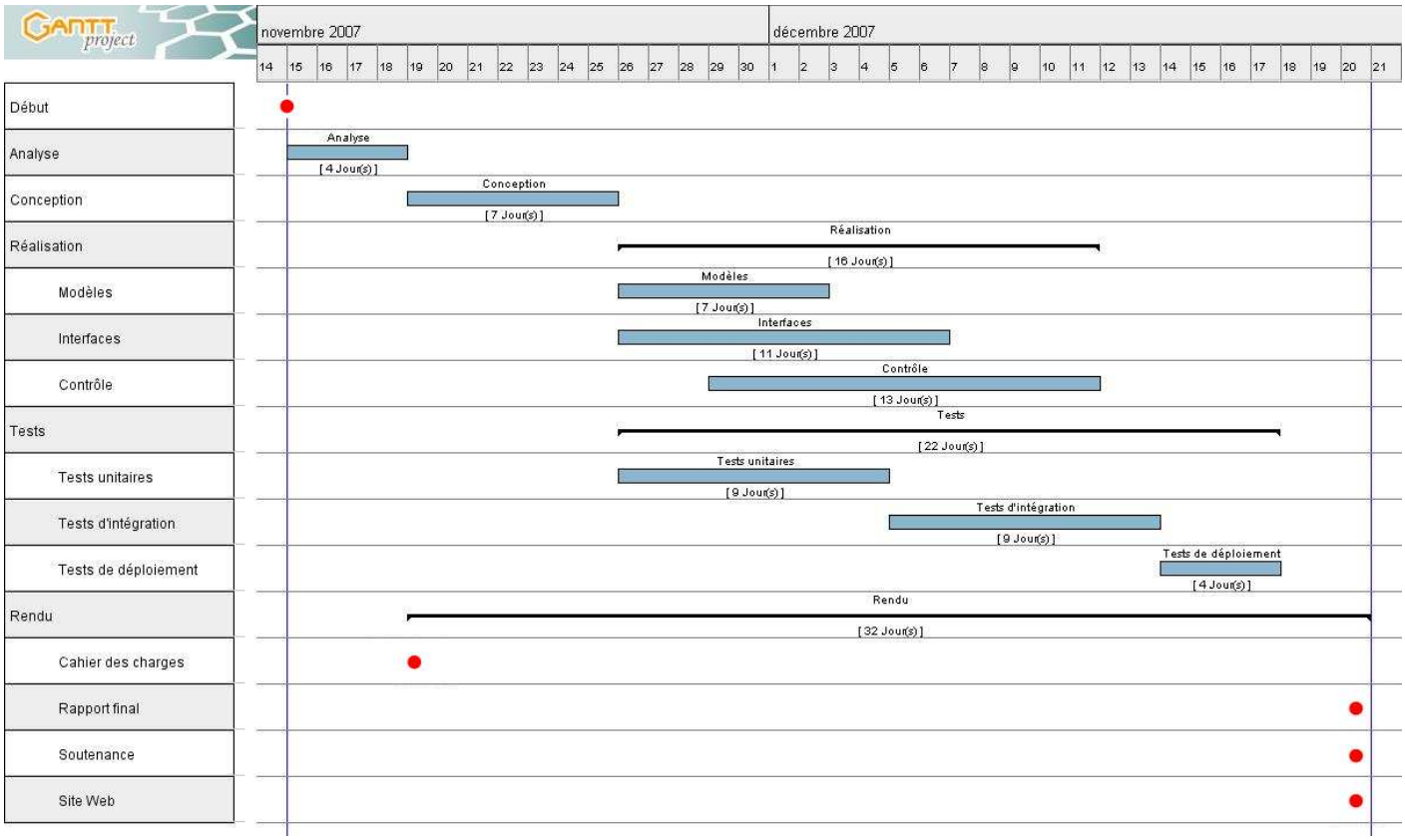
Notre programme devant être compatible Windows et Linux, nous avons décidé d'effectuer notre conception sur ces deux systèmes. Nous avons veillé à chaque séance de travail à ce que le projet fonctionne correctement sous Linux et Windows XP/Vista.

Ce projet étant réalisé dans le cadre du cours de Programmation Orientée Objet, il a été demandé de suivre une architecture utilisant des objets. De plus, dans un souci de clarté et de réutilisabilité du code, nous avons opté pour une conception modulaire. En effet, chaque fonctionnalité importante du programme est implémentée dans un module dédié (affichage, perso, jeu, menu…).

Le programme étant un jeu, nous avons essayé de faire en sorte que celui-ci soit agréable à l'œil, et facile à prendre en main.

# III – Planning

## 1) Planning initial




## 2) Enchainement des phases du projet

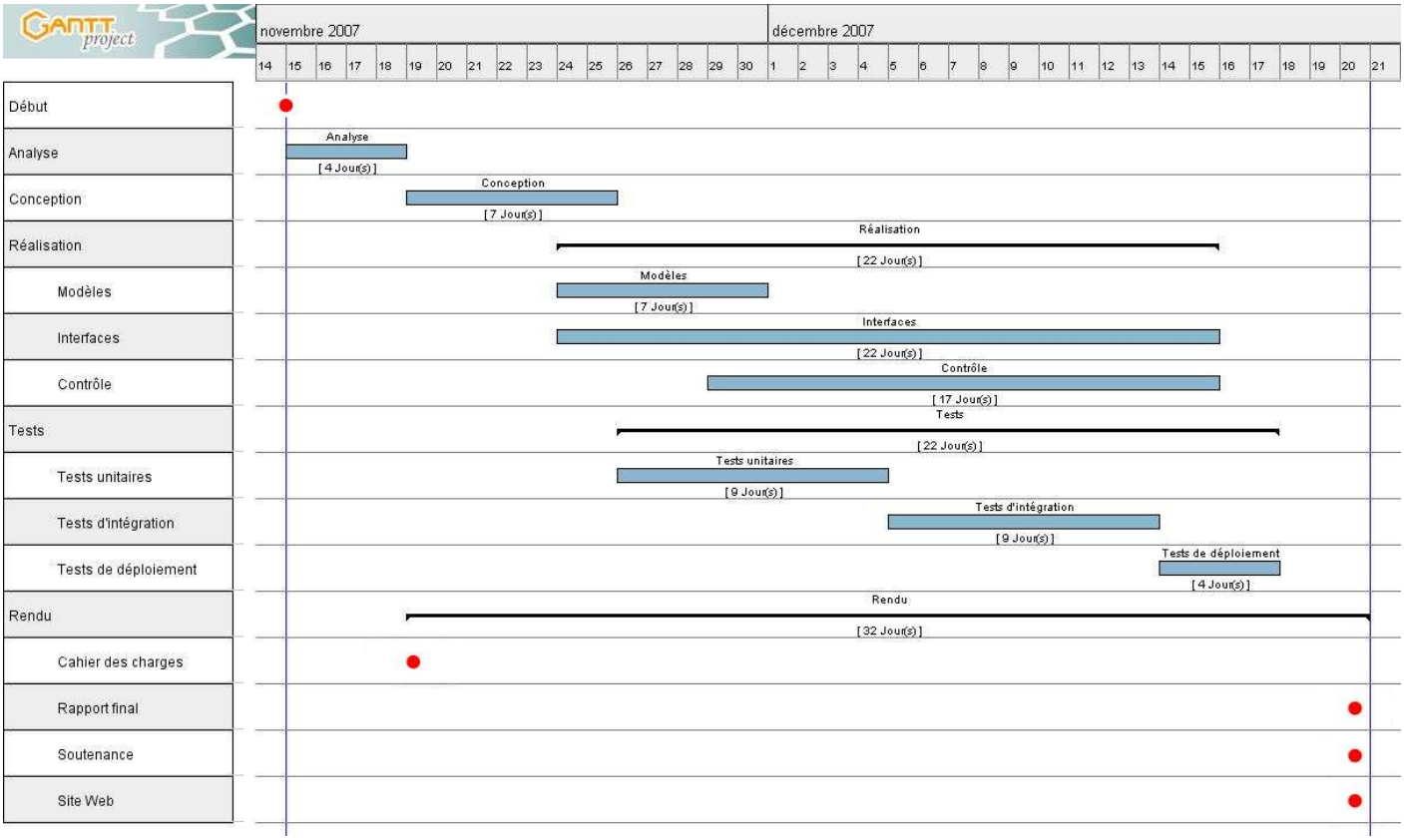
La réalisation de ce projet a été découpée en 4 principales phases :

- La première étape correspond à l'analyse du problème. Cette étape s'est soldée par la remise du cahier des charges le 19 novembre 2007.
- La seconde étape est une phase de conception, durant laquelle nous avons mis en place l'architecture générale du programme avec notamment le découpage en plusieurs modules, la détermination des relations entre ces modules, et bien sur leur composition.
- L'étape suivante est la réalisation des objectifs fixés dans le cahier des charges en s'appuyant sur les résultats de la phase de conception.
  Cette étape peut elle-même se découper en 3 parties :
  - Modèles : mise en place des données de l'application (les classes notamment).

- Interfaces : Mise en place des éléments permettant une interface avec l'utilisateur (Affichage, contrôles clavier/souris…).
- Contrôles : Ce dernier point concerne la gestion des données, le déroulement du programme (interactions entre les objets, algorithmes de déplacements…)

- Enfin, la dernière étape constitue la phase de tests. Ces tests sont effectués en parallèle avec la phase de développement. On distingue 3 types de tests :
  - Tests unitaires : Ils permettent de vérifier l'intégrité de chaque module.
  - Tests d'intégration : Ils valident l'interaction entre les différents modules communiquant entre eux.
  - Tests de déploiement : Ces derniers tests permettent de vérifier le bon fonctionnement du programme.

## 3) Planning effectif



# IV – Conception

## 1) Langage utilisé

Nous avons développé ce projet en utilisant le langage C++, afin d'utiliser la programmation orientée objet.

Nous avons utilisé les bibliothèques graphiques OpenGL et Glu pour gérer l'affichage 2D. Nous avons également utilisé la bibliothèque Glut pour gérer le système de fenêtrage et les interruptions utilisateurs (gestion du clavier et de la souris). Ces bibliothèques sont gratuites et portables.

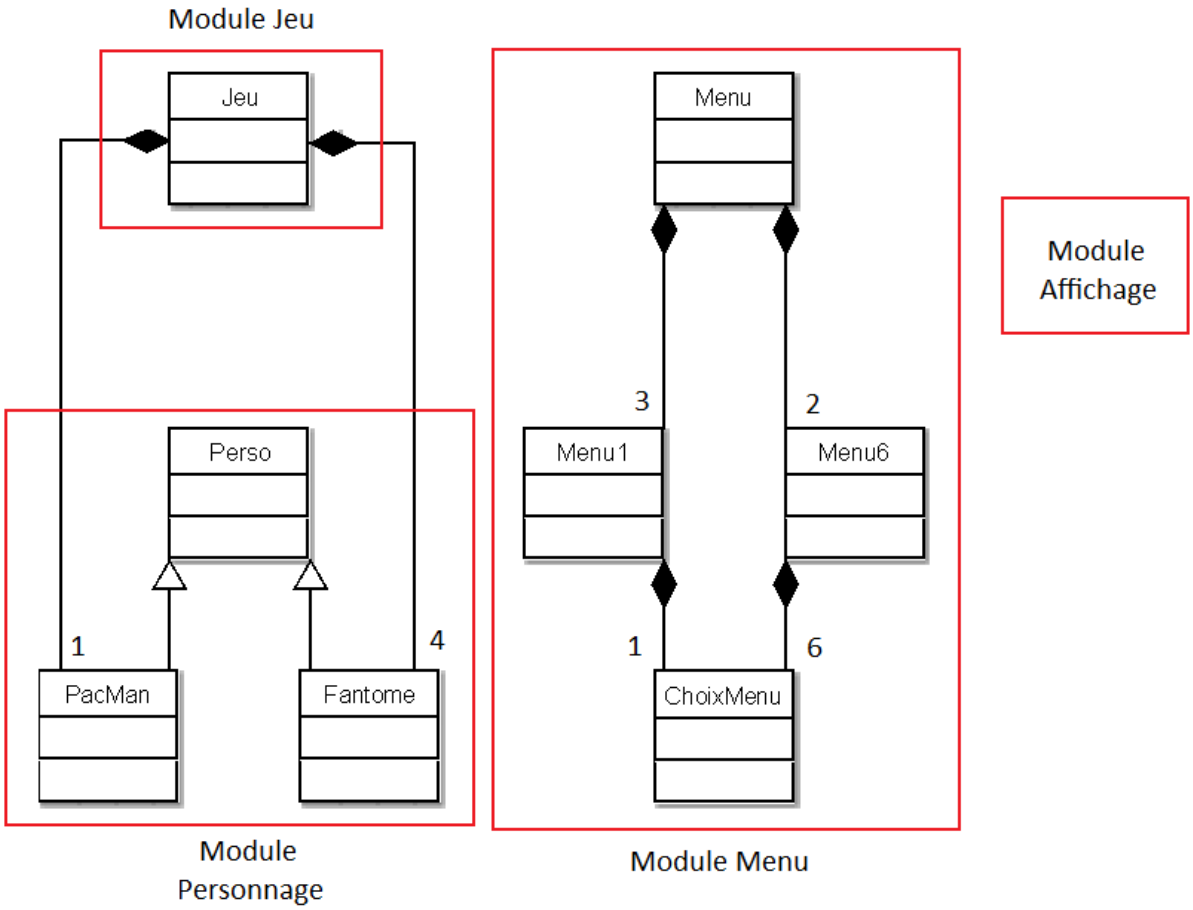
Nous nous sommes servis d'un loader d'images au format bmp pour afficher nos images.

Nous nous sommes servis du compilateur GCC. Nous avons développé et compilé notre code sous Linux, Windows XP et Windows Vista. Nous avons utilisé l'environnement de développement Microsoft Visual Studio 2005 pour le développement sous Windows.

## 2) Architecture générale

Nous avons défini 4 modules principaux : Personnages, Plateau de jeu, Menus, Affichage.

- Le module Personnage rassemble les 3 classes suivantes : Perso, PacMan et Fantome. La classe Perso rassemble les caractéristiques d'un personnage du jeu (PacMan ou Fantome). Nous l'avons spécialisée selon que le personnage soit Pac-Man ou un des quatre fantômes.
- Le module plateau de jeu représente la classe Jeu contenant tous les éléments nécessaires au déroulement d'une partie, à savoir la carte, le Pac-Man et les 4 fantômes.
- Le module Menus correspond aux 4 classes relatives aux menus
- Le module Affichage est regroupé dans le fichier du même nom et contient toutes les fonctions relatives à l'affichage…



## 3) Structure des fichiers

Pour que notre logiciel fonctionne correctement, il est nécessaire que l'exécutable du jeu se trouve dans le même dossier que les dossiers maps et images.

Nous avons l'arborescence suivante :

- Dossier code contenant le code source
  - Dossier maps contenant les cartes du jeu et le fichier des meilleurs scores
  - Dossier images contenant toutes les images du jeu
  - Exécutable du jeu

Par souci de clarté et de réutilisabilité du code, nous nous sommes imposé des normes de programmation.

- Tous nos fichiers comportent un entête avec le nom du fichier, les auteurs, la date de création du fichier et un synopsis expliquant le contenu du fichier. Eventuellement, il peut y avoir une section ToDo expliquant les améliorations que l'on pourrait apporter et/ou une section Bugs répertoriant les bugs connus.
  Il est impératif de modifier cet entête en cas de modification en ajoutant la date et les modifications apportées.

- Au niveau du code, dans l'ordre on rencontre :
  - Les inclusions de fichiers systèmes
  - Les inclusions de fichiers de nos modules
  - Les variables globales définies ailleurs mais utilisées dans le fichier
  - Les déclarations de variables globales pour le fichier
  - Les déclarations de classes (pour un fichier d'entête de classe .h)
  - Les déclarations de fonctions (pour un fichier d'entête .h)
  - Les définitions de fonctions (pour un fichier .cxx)

- Toutes les macros définies sont dans le fichier Define.h.
- Toutes les variables globales sont définies soit dans Main.cxx, soit dans Affichage.cxx mais en aucun cas d'en un fichier de classe.
- Chaque fonction définie a d'abord son profil déclaré dans un fichier d'entête .h.
- Le nom des données membres d'une classe commence par un _.
- Le nom des variables et fonctions est le plus explicite possible sans être trop long. Si nécessaire, il est accompagné d'un commentaire.
- Le nom des variables et fonctions commence par une majuscule à chaque début de mot. Par exemple : CeciEstUneFonction ()

## 4) Traitement des erreurs

Nous avons utilisé la directive assert () pour traiter les erreurs de notre programme. Ainsi, lorsqu'une erreur se produit le programme est arrêté.

Il aurait été préférable de mettre en place un système permettant de gérer des exceptions mais nous avons préféré utiliser ce que nous maîtrisons vraiment.

# V – Conception détaillée

Cette section présente chaque module de manière détaillée : les données et fonctions membres pour les classes, les variables globales et fonctions pour les autres modules.

Les algorithmes compliqués (intelligence artificielle) seront détaillés en pseudo-code.

## 1) Classe Perso

Dans notre jeu, un personnage peut être soit Pac-Man, soit un des 4 fantômes. Leurs caractéristiques communes sont gérées dans cette classe.

- Les données membres

Elles sont déclarées `protected` pour que seules les classes filles (PacMan et Fantome) puissent y accéder directement.

`unsigned int _x, _y` : représentent les coordonnées du personnage dans la carte.

`float _xf, _yf` : représentent les coordonnées du personnage dans le menu principal (pour l'animation)

`int _decalx, _decaly` : représentent le décalage à appliquer pour l'affichage de l'image. Cela permet d'avoir plus de fluidité au niveau de l'affichage.

`unsigned int _direction` : représente le sens du mouvement (1:haut, 2:bas, 3:gauche, 4:droite)

`unsigned int _vitesse` : fréquence à laquelle est appelé le timer qui gère le déplacement

`unsigned int _img` : numéro correspondant à un fichier image

- Les fonctions membres

  - 2 constructeurs (par défaut, par valeur et par recopie)

```
Perso (unsigned int x = 16, unsigned int y = 9, float xf = 0,
float yf = 0, int DecalX = 0, int DecalY = 0, unsigned int
Direction = 3, unsigned int Vitesse = VITESSE_PACMAN, unsigned
int Img = 10);
Perso (const Perso & P);
```

- Des accesseurs pour les données membres

```
unsigned int GetX (void) const;
unsigned int GetY (void) const;
float GetXf (void) const;
float GetYf (void) const;
int GetDecalX(void) const;
int GetDecalY(void) const;
unsigned int GetDirection (void) const;
unsigned int GetVitesse (void) const;
unsigned int GetImg (void) const;
```

- Des modifieurs pour les données membres

```
void SetX (unsigned int x);
void SetY (unsigned int y);
void SetXf (float x);
void SetYf (float y);
void SetDecalX(int decalx);
void SetDecalY(int decaly);
void SetDirection (unsigned int Direction);
void SetVitesse (unsigned int Vitesse);
void SetImg (unsigned int Img);
```

- Des fonctions membres particulières

```
void Avance (unsigned int Map [NbLi][NbCol]);
```

Cette fonction récupère la direction du personnage et l'avance d'une case sauf s'il y a mur auquel cas elle ne fait rien.

- Un destructeur

```
~Perso (void);
```

- Une surcharge d'operateur

```
ostream & operator << (ostream & flux, const Perso & P);
```

Elle ne sert qu'à afficher les valeurs des données membres à la console (pour le débogage).

## 2) Classe PacMan

Comme expliqué précédemment, cette classe hérite de la classe Perso.

- Les données membres

Elles sont déclarées `private` pour qu'on ne puisse pas y accéder directement.

`int _nbVies` : c'est le nombre de vies restantes du joueur

`unsigned int _score` : score du joueur pour la partie depuis le premier niveau

`unsigned int _imgPrec` : sauvegarde l'image précédente du Pac-Man pour l'animation de la bouche

`unsigned int _directionClavier` : direction enregistrée au clavier. Ce sera la prochaine direction du Pac-Man dès qu'il pourra la prendre.

- Les fonctions membres

  - 2 constructeurs (par défaut, par valeur et par recopie)

```
PacMan (unsigned int x = 16, unsigned int y = 9, unsigned int
Direction = 3, unsigned int Vitesse = VITESSE_PACMAN, unsigned
int Img = 10, int NbVies = 2, unsigned int Score = 0, unsigned
int ImgPrec = 10, unsigned int DirectionClavier = 3);
PacMan (const PacMan & P);
```

  - Des accesseurs pour les données membres

```
int GetNbVies (void) const;
unsigned int GetScore (void) const;
unsigned int GetImgPrec (void) const;
unsigned int GetDirectionClavier(void) const;
```

  - Des modifieurs pour les données membres

```
void SetNbVies (int NbVies);
void SetScore (unsigned int Score);
void SetImgPrec (unsigned int ImgPrec);
void SetDirectionClavier(unsigned int Dir);
```

  - Un destructeur

```
~PacMan (void);
```

- Une surcharge d'operateur

```
ostream & operator << (ostream & flux, const PacMan & P);
```

Elle ne sert qu'à afficher les valeurs des données membres à la console (pour le débogage).

## 3) Classe Fantôme

Comme expliqué précédemment, cette classe hérite de la classe Perso.

- Les données membres

Elles sont déclarées `private` pour qu'on ne puisse pas y accéder directement.

`unsigned int _etat` : Correspond à l'état du fantôme (mort, vivant ou vulnérable)

- Les fonctions membres

- 2 constructeurs (par défaut, par valeur et par recopie)

```
Fantome (unsigned int x = 8, unsigned int y = 9, unsigned int
Direction = 1, unsigned int Vitesse = VITESSE_FANTOME,
unsigned int Img = 20, unsigned int Etat = 1);
Fantome (const Fantome & F);
```

- Des accesseurs pour les données membres

```
unsigned int GetEtat (void) const;
```

- Des modifieurs pour les données membres

```
void SetEtat (unsigned int Etat);
```

- Des fonctions membres particulières

```
void Vulnerabilite (int img);
```

Après avoir mangé une pac-gomme spéciale, les fantômes deviennent vulnérables. Cette fonction les remet dans leur état normal au bout de 10s (s'ils ne sont pas morts).

```
void Ressuciter (int img);
```

Si un fantôme se fait manger par Pac-Man pendant qu'il est vulnérable, le fantôme réapparait au centre de la carte au bout d'un certain temps.

- Un destructeur

```
~Fantome (void);
```

- Une surcharge d'operateur

```
ostream & operator << (ostream & flux, const Fantome & F);
```

Elle ne sert qu'à afficher les valeurs des données membres à la console (pour le débogage).

# 4) Classe Jeu

## a) *Interface détaillée*

Cette classe matérialise le plateau de jeu.

- Les données membres

Celles qui représentent les éléments principaux sont déclarées `public` afin de faciliter l'accès au plateau de jeu pour les autres modules. Les autres sont déclarées `private`.

`private :`

`char * _fichierMap` : nom du fichier contenant la carte
`int _niveau` : niveau en cours
`int _bonusVie` : sert à détecter si l'on va gagner une vie (à chaque 10 000 points on en gagne une)

`public :`

`unsigned int Map[NbLi][NbCol]` : Carte du jeu
`PacMan P` : Pac-Man
`Fantome F1` : fantôme bleu : Bashful alias "Inky"
`Fantome F2` : fantôme orange : Pokey alias "Clyde"
`Fantome F3` : fantôme rose : Speedy alias "Pinky"
`Fantome F4` : fantôme rouge : Shadow alias "Blinky"

- Les fonctions membres

  - 2 constructeurs (par défaut et par recopie)

```
Jeu (void);
Jeu (const Jeu & J);
```

  - Des accesseurs pour les données membres

```
char * GetFichierMap (void) const;
int GetNiveau (void) const;
int GetBonusVie (void) const;
```

- Des modificateurs pour les données membres

```
void SetFichierMap (char * FichierMap);
void SetNiveau (int Niveau);
void SetBonusVie (int BonusVie);
```

- Des fonctions membres particulières

➢ **Fonctions d'initialisation**

```
bool Init (int mode);
```
Cette fonction initialise la carte et les positions des fantômes et du Pac-Man. Elle est appelée au commencement d'une partie, au début d'un niveau ou lorsque le Pac-Man se fait manger par un fantôme (pour réinitialiser les positions des personnages).

```
void PlaceBonus (void);
```
Elle place un bonus dans une case vide de la carte de manière aléatoire.

➢ **Fonction de débogage**

```
void AfficheCarte (void);
```
Elle affiche les valeurs de la carte (Elle sert pour le débogage uniquement).

➢ **Fonctions de fin de partie**

```
bool PartieFinie (void);
```
Elle parcourt la carte et regarde s'il reste des pastilles. Lorsque toutes les pac-gommes sont mangées, le niveau est fini. La fonction renvoie donc vrai.

```
bool GameOver (void);
```
Elle renvoie vrai si Pac-Man s'est fait manger et qu'il n'a plus de vies.

```
void SauvegardeScore (string NomJoueur, int & StopNomJoueur);
```
Cette fonction est appelée en fin de partie. Elle lit le fichier des scores, regarde si le joueur rentre dans le Top 10. Si oui, elle lui demande son nom puis l'inscrit dans le fichier des scores. Elle appelle ensuite la fonction d'affichage des meilleurs scores.

➢ **Fonctions d'interaction entre les objets**

```
int PacManMangePastille (void);
```
Cette fonction augmente le score du joueur selon ce que Pac-Man mange (pastilles, pastilles spéciales ou bonus). Si Pac-Man prend une pastille spéciale, cette fonction rend les fantômes vulnérables. Tous les 10 000 points, cette fonction rajoute une vie au joueur.

```
void PacManMangeFantome (void);
```
Cette fonction gère le fait que si Pac-Man mange un fantôme vulnérable, le fantôme meurt pour un certain temps et le score augmente.

```
bool FantomeMangePacMan (void);
```

Cette fonction gère le fait que si un fantôme mange le Pac-Man, il perd une vie. On réinitialise les positions des personnages.

- **Fonctions d'intelligence artificielle**

```
int PacManVisibleDepuis(unsigned int x, unsigned int y);
```
Cette fonction regarde si un fantôme (à la position (x,y)) voit Pac-Man, c'est-à-dire s'il est sur la même ligne ou sur la même colonne que Pac-Man et s'ils ne sont pas séparés par un mur. Si le fantôme voit Pac-Man, alors il va vers lui.

```
int EloigneToiDePacMan (int x, int y, int dir);
```
Si un fantôme (à la position (x,y)) se trouve à une intersection ou qu'il ne peut plus continuer dans sa direction, cette fonction va regarder quelle est la meilleure direction à prendre pour s'éloigner de Pac-Man et renvoie cette direction. Sinon, elle ne change pas la direction.

```
bool PacManProchePastilleSpe (int & X, int & Y);
```
Cette fonction détecte si le Pac-Man s'approche d'une pastille spéciale. Si oui, elle renvoie vrai. Sinon elle renvoie faux.

```
void IA_DeplacerF1 (void);
```
Cette fonction détermine l'intelligence des déplacements du fantôme bleu.
Tout d'abord on teste si l'on voit Pac-Man. Si oui on va vers lui. Sinon on regarde si Pac-Man est proche d'une pastille spéciale. Si oui, on s'éloigne de Pac-Man le plus possible. Si on ne se trouve dans aucun de ces 2 cas, alors à chaque intersection on choisit une direction de manière aléatoire.

```
void IA_DeplacerF2 (void);
```
Cette fonction définit l'intelligence des déplacements du fantôme orange.
Tout d'abord, on regarde si l'on voit Pac-Man. Si oui on va vers lui. Sinon, à chaque fois qu'on est à une intersection, on prend une direction aléatoirement.

```
void IA_DeplacerF3 (void);
```
Cette fonction représente l'intelligence des déplacements du fantôme rose.
Tout d'abord, on regarde teste si l'on voit Pac-man. Si oui on se dirige vers lui. Sinon on demande aux autres fantômes si l'un d'entre eux voit Pac-Man. Si oui, il nous communique sa position et on choisit la meilleure direction pour se rapprocher de Pac-Man. Si on n'est dans aucun de ces cas, on choisit une direction de manière aléatoire à chaque fois que l'on est à une intersection.

```
void IA_DeplacerF4 (void);
```
Cette fonction détermine l'intelligence des déplacements du fantôme rouge.
Si on est vivant, on choisit la meilleure direction pour aller vers Pac-Man. Si on est vulnérable, on s'éloigne le plus possible de Pac-Man.

➢ **Fonctions de déplacement**

`void MiseAJourDirectionPacMan (void);`
Cette fonction regarde la direction rentrée au clavier par le joueur. Pac-Man empreinte cette direction dès que possible. Cela permet de mémoriser la direction au clavier pour une meilleure jouabilité.

`void MouvementPacMan (int timer);`
Cette fonction est appelée constamment car elle gère les déplacements du Pac-Man ainsi que les images associées. Cette fonction appelle toutes les fonctions d'interaction entre objet pour détecter si une action particulière est à faire. Par exemple, si le Pac-Man mange une pastille spéciale, cette fonction rend les fantômes vulnérables et affiche le compte à rebours.

`void MouvementFantomes (int timer);`
De la même façon que la fonction précédente, celle-ci gère les déplacements des fantômes. Elle appelle les fonctions d'intelligence artificielle et gère les images des fantômes.

- Un destructeur

`~Jeu(void);`

### b) *Algorithmes d'Intelligence Artificielle*

Les fonctions suivantes sont détaillées plus loin :
- PacManVisibleDepuis ()
- EloigneToiDePacMan ()
- PacManProchePastilleSpe ()

Quelque soit l'algorithme, avant de changer une direction, on vérifie toujours que celle-ci est possible.

#### *Bashfull alias « Inky » : le fantôme bleu (IA_DeplacerF1 ())*

Si fantôme vivant & fantôme voit Pac-Man (appel à `PacManVisibleDepuis ()`)
    Aller vers Pac-Man
Sinon si fantôme vivant & Pac-Man proche d'une pastille spéciale
    (appel à `PacManProchePastilleSpe ()`)
    S'éloigner de Pac-Man (appel à `EloigneToiDePacMan ()`)
Sinon si fantôme vivant
    *Algorithme : Direction aléatoire* (cf. fantôme orange → `IA_DeplacerF2 ()`)
Sinon (fantôme vulnérable)
    S'éloigner de Pac-Man (appel à `EloigneToiDePacMan ()`)

### *Pokey alias « Clyde » : le fantôme orange (IA_DeplacerF2 ())*

Si fantôme vivant & fantôme voit Pac-Man (appel à `PacManVisibleDepuis ()`)
    Aller vers Pac-Man
Sinon si fantôme vivant

    *Algorithme : Direction aléatoire*

> Prendre une direction aléatoire si possible et autre que retour arrière
> Sinon on fait un retour arrière

Sinon (fantôme vulnérable)
    S'éloigner de Pac-Man (appel à `EloigneToiDePacMan ()`)

### *Speedy alias « Pinky » : le fantôme rose (IA_DeplacerF3 ())*

Si fantôme vivant & fantôme voit Pac-Man (appel à `PacManVisibleDepuis ()`)
    Aller vers Pac-Man

Sinon si fantôme vivant & un autre fantôme voit Pac-Man
(appel à `PacManVisibleDepuis ()`)

    *Algorithme : Meilleure direction pour aller vers Pac-Man*
(cf. fantôme rouge → `IA_DeplacerF4 ()`)

Sinon si fantôme vivant
    *Algorithme : Direction aléatoire* (cf. fantôme orange → `IA_DeplacerF2 ()`)

Sinon (fantôme vulnérable)
    S'éloigner de Pac-Man (appel à `EloigneToiDePacMan ()`)

### *Shadow alias « Blinky » : le fantôme rouge (IA_DeplacerF4 ())*

Si fantôme vivant & fantôme voit Pac-Man (appel à `PacManVisibleDepuis ()`)
    Aller vers Pac-Man
Sinon si fantôme vivant

*Algorithme : Meilleure direction pour aller vers Pac-Man*

```
Selon la direction qu'on a :
    Si on est à une intersection
        Si on est plus près en hauteur que sur les côtés
            On se rapproche de Pac-Man sur les côtés
            (si fantôme à gauche de Pac-Man, on va à droite
            sinon, on va à gauche)
        Sinon (on est plus près sur les côtés qu'en hauteur)
            On se rapproche de Pac-Man en hauteur
            (si fantôme au dessus de Pac-Man, on va en bas
            sinon, on va en haut)
    Sinon si on ne peut plus continuer dans la même direction
        Si on est sur la même ligne (ou même colonne)
            On prend une direction aléatoirement
        Sinon si le fantôme est à gauche (ou au dessus) de Pac-Man
            On va à droite (ou au dessous)
        Sinon si le fantôme est à droite (ou au dessous) de Pac-Man
            On va à gauche (ou au dessus)
        Sinon (on n'a pu prendre aucune direction)
            On fait marche arrière
    Sinon
        On continue dans la même direction
```

Sinon (fantôme vulnérable)
    S'éloigner de Pac-Man (appel à `EloigneToiDePacMan ()`)

### *Fonction PacManVisibleDepuis ()*

Cette fonction renvoie la direction à prendre si on voit Pac-Man depuis la case passée en paramètre. Si on ne le voit pas, renvoie 0.

```
Si fantôme sur la même ligne que Pac-Man
        Si fantôme à gauche de Pac-Man
                On parcourt la ligne entre fantôme et Pac-Man pour savoir s'il y a un mur
                S'il n'y a pas de mur, on renvoie droite
        Sinon (fantôme à droite de Pac-Man)
                On parcourt la ligne entre fantôme et Pac-Man pour savoir s'il y a un mur
                S'il n'y a pas de mur, on renvoie gauche
Sinon si fantôme sur la même colonne que Pac-man
        Si fantôme au dessus de Pac-Man
                On parcourt la colonne entre fantôme et Pac-Man pour savoir s'il y a un mur
                S'il n'y a pas de mur, on renvoie bas
        Sinon (fantôme en dessous de Pac-Man)
                On parcourt la colonne entre fantôme et Pac-Man pour savoir s'il y a un mur
                S'il n'y a pas de mur, on renvoie haut
Sinon
        Renvoie 0
```

### *Fonction PacManProchePastilleSpe ()*

Cette fonction renvoie vrai si le Pac-Man est proche d'une pastille spéciale. Dans le cas contraire, elle renvoie faux.

```
On parcourt la carte pour mémoriser où sont les pastilles spéciales.
Si le Pac-Man est autour de l'une d'entre elles
        Renvoie vrai
Sinon
        Renvoie faux
```

### *Fonction EloigneToiDePacMan ()*

Cette fonction renvoie la direction à prendre pour s'éloigner le plus possible de Pac-Man à partir de la case passée en paramètre.

```
Selon la direction (exemple pour aller en haut)
        Si on est à une intersection
                Si fantôme plus près de Pac-Man en hauteur que sur les côtés
                        Si fantôme au dessus de Pac-Man
                                Renvoie haut
                        Sinon si fantôme à gauche de Pac-Man
                                Renvoie gauche
                        Sinon si fantôme à droite de Pac-Man
                                Renvoie droite
                        Sinon
                                On teste chaque direction
                Sinon (fantôme lus près de Pac-Man sur les côtés qu'en hauteur)
                        Si fantôme à gauche de PacMan
                                Renvoie gauche
                        Sinon si fantôme à droite de Pac-Man
                                Renvoie droite
                        Sinon si fantôme au dessus de Pac-Man
                                Renvoie haut
                        Sinon
                                On teste chaque direction
        Sinon si on ne peut plus continuer dans la direction
                Si fantôme à gauche de Pac-Man
                        Renvoie gauche
                Sinon si fantôme à droite de Pac-Man
                        Renvoie droite
                Sinon
                        On teste chaque direction
        Sinon
                On garde la même direction
```

# 5) Les classes concernant le menu

Nous avons un module permettant de gérer notre menu.

## a) *Classe ChoixMenu*

Cette classe permet d'afficher un item dans un menu.

- Les données membres

Elles sont déclarées `private` pour qu'on ne puisse pas y accéder directement.

`int _img` : représente l'image de l'item du menu
`double _x, _y` : Correspond aux coordonnées de l'image dans la fenêtre

- Les fonctions membres

  - Un constructeur par défaut

```
ChoixMenu (void);
```

  - Des accesseurs pour les données membres

```
int GetImg (void) const;
double GetX (void) const;
double GetY (void) const;
```

  - Des modifieurs pour les données membres

```
void SetImg (int Img);
void SetX (double x);
void SetY (double y);
```

  - Un destructeur

```
~ChoixMenu (void);
```

### b) Classe Menu1

Cette classe propose un menu à 1 item.

- Les données membres

Elles sont déclarées `public` pour faciliter l'accès.

`ChoixMenu C1` : Correspond à l'item du menu

- Les fonctions membres

  - Un constructeur par défaut

```
Menu1 (void);
```

  - Des fonctions membres particulière

`void Init (void)`: initialise l'item du menu

  - Un destructeur

```
~Menu1 (void);
```

### c) Classe Menu6

Cette classe propose un menu à 6 items.

- Les données membres

Elles sont déclarées `public` pour faciliter l'accès.

`ChoixMenu C1, C2, C3, C4, C5, C6` : Correspond aux items du menu

- Les fonctions membres

  - Un constructeur par défaut

```
Menu6 (void);
```

  - Des fonctions membres particulière

`void Init (int menu)`: initialise les items du menu

  - Un destructeur

```
~Menu6 (void);
```

### d) Classe Menu

Cette classe représente notre menu.

- Les données membres

Celles qui représentent les éléments principaux sont déclarées `public` afin de faciliter l'accès au plateau de jeu pour les autres modules. Les autres sont déclarées `private`.

```
private :
     int _etat; //0 : menu principal, 1 : partie, 2 ; select
map, 3 : high score, 4 : help, 5 : about, 6 : quit
public :
     Menu6 MenuPrincipal;
     Menu6 MenuSelectMap;
     Menu1 MenuHighScore;
     Menu1 MenuHelp;
     Menu1 MenuAbout;
```

- Les fonctions membres

  - Un constructeur par défaut

```
Menu (void);
```

  - Des accesseurs

```
int GetEtat (void) const;
```

  - Des modifieurs

```
void SetEtat (int Etat);
```

  - Des fonctions membres particulière

`void Init (void)`: initialise le menu

  - Un destructeur

```
~Menu (void);
```

## 6) Le module Affichage

Ce module s'occupe de la gestion de toutes les images du jeu.

```
void Pause (unsigned int msec);
```
Cette fonction permet de faire une pause dans le jeu.

```
void rectangleplein (float x1, float y1, float x2, float y2,
int direction);
```
Elle trace un rectangle dans lequel on pourra appliquer une texture.

```
void ChargeTextures (void);
```
Cette fonction charge en mémoire toutes les images du jeu grâce au module Texture.

```
void EcritTextei (int x, int y, char * string);
void EcritTextef (float x, float y, char * string);
```
Ces deux fonctions permettent d'écrire du texte dans la fenêtre OpenGL aux coordonnées passées en paramètre.

```
void AfficheScore (void);
void AfficheVies (void);
void AfficheNiveau (void);
void AfficheTemps (void);
```
Ces quatre fonctions permettent d'afficher les informations relatives à la partie.

```
void InitAnimation (void);
void AnimationMenu (int param);
void GestionImages (int param);
```
Ces trois fonctions permettent d'afficher une petite animation d'introduction dans le menu principal.

```
void MenuP (void);
```
Cette fonction affiche le menu principal du jeu.

```
void Start (void);
```
Cette fonction permet de lancer une partie et gère tous les évènements qui peuvent se passer.
Tout d'abord elle initialise la carte. Puis elle affiche la carte, c'est-à-dire toutes les images formant la carte. Elle affiche également les informations relatives à la partie (nombre de vies restantes, score, niveau en cours)
Ensuite elle regarde constamment :

- si on n'est pas en fin de partie
- si on n'est pas en fin de niveau auquel cas elle réinitialise la carte

```
void SelectMap (void);
```
Elle affiche le menu correspondant qui permet de choisir une carte.

```
void HighScore (void);
```
Cette fonction lit le fichier des scores et affiche les 10 meilleurs scores à l'écran.

```
void Help (void);
void About (void);
```
Ces deux fonctions affichent les menus correspondants.

```
void GestionImgPacMan (void);
void GestionImgFantomes (void);
```
Ces deux fonctions gèrent l'animation des personnages pour leur donner un aspect vivant.

```
void Decale (int param);
void AnimationMouvement (int x1, int y1, int x2, int y2, int
P);
```
Ces deux fonctions permettent d'afficher des images entre les cases de la carte pour donner une impression de fluidité.

```
void GestionImagesMenu (float x, float y, int menu);
void GestionChoixMenu (float x, float y, int menu);
```
Ces deux fonctions permettent de mettre en surbrillance un menu pointé par la souris.

## 7) Le fichier Main.cxx

C'est le fichier principal qui appelle tous les autres. Il contient uniquement des fonctions servant pour la bibliothèque glut et la fonction main () ainsi que des variables globales.

# VI - Conclusion

Nous avons rencontré peu de difficultés durant ce projet. Nous pouvons tout de même citer :

- Un problème un niveau des passages de paramètres pour les fonctions de la bibliothèque glut car celle-ci est écrite en C. nous avons rencontré un léger problème d'incompatibilité étant donné que nous développons en C++, les fonctions membres de nos classes ne sont pas acceptées en paramètre.

- Nous aurions aimé développer plus de fonctionnalités pour notre jeu et le peaufiner un peu plus mais nous n'avons pas eu assez de temps. Par exemple, nous pourrions ajouter du son, améliorer les algorithmes d'intelligence artificielle, permettre de mettre le jeu en pause, permettre d'effacer des caractères dans la saisie du nom du joueur en cas d'erreur, … Mais ce ne sont que des détails.

Cependant nous avons respecté nos engagements pris dans le cahier des charges, aussi bien au niveau du développement des fonctionnalités du jeu qu'au niveau des délais imposés.

- Ce projet nous a permis de nous familiariser avec la programmation orientée objet et de mieux maîtriser ce concept.

- Le travail en binôme a été particulièrement appréciable car il est tout à fait adapté à ce type de projet. Il nous a permis de travailler rapidement et efficacement.

- Nous nous sommes beaucoup investis dans ce projet car il nous a particulièrement intéressés étant donné que c'est un jeu finalisé avec une interface graphique. Nous avons tenté de le rendre le plus agréable possible.

# VII – Autres Documents

Vous pouvez consulter d'autres documents rattachés à ce projet notamment :

- Le cahier des charges
- Le code source
- Le manuel d'utilisation
- La page web présentant ce projet : http://julie.andre.perso.esil.univmed.fr/PacMan/projetPacMan.html