# Programmation avancée
# Rapport de contribution

## 1 Introduction

Ce rapport se focalisera sur les contributions que j'ai apportées dans le projet. Certaines parties ayant été faites en collaboration avec mon binôme, Lucas CHEEKHOOREE, ces dernières seront indiquées dans le document.
J'ai également porté beaucoup de soin aux messages des *commits* sur le Git[1] afin de pouvoir retracer chaque tâche effectuée.

## 2 Les débuts du projet

La première prise en mains du projet a été relativement compliquée. Nous avons donc commencé par nous familiariser avec la base du projet, mise à disposition sur Moodle.
J'ai ainsi débuté mes travaux par de l'analyse des différents tronçons de code afin de les commenter et de pouvoir les adapter à Pac-Man (le personnage) notamment.

À la suite de cela, je me suis lancé dans des tests afin d'incorporer Pac-Man et de rendre le personnage contrôlable, même si cela se faisait au détriment de l'aspect objet, au départ. Cela fait, j'ai poursuivi dans ma lancée en incorporant les animations du personnage, et la gestion en cas de rencontre avec un fantôme.
Pour gérer ce cas, j'ai trouvé une fonction de SDL2 permettant de retourner un booléen en cas d'intersection entre deux rectangles SDL, facilitant ainsi la tâche.

Cela était néanmoins écrit dans un formatage proche du C, permettant de s'abstraire de l'aspect objet dans un premier temps, afin de se focaliser sur la prise en main de la librairie. Ce point a été corrigé par mon binôme, après mes différents essais.

## 3 Contrôle de Pac-Man et gestion des obstacles

Une fois Pac-Man contrôlable, il a fallu implémenter les murs. Cela s'est fait de la même manière que pour l'interaction avec les fantômes, à l'aide de la fonction *SDL_HasIntersection()*.

Nous avons décidé de récupérer les coordonnées des murs de la *bitmap* dans un vecteur afin de les initialiser par la suite avec une boucle. Cela m'a pris beaicup de temps à mesurer sur l'image fournie, mais a permis de débugger plus facilement par la suite les différents murs, et ce notamment grâce aux commentaires que j'ai mis en place.

Pac-Man étant contrôlable et les murs implémentés, un problème est vite apparu : le personnage n'est pas centré sur le chemin. Pour résoudre cela, j'ai proposé d'ajuster la largeur des murs en empiétant sur le chemin, afin qu'il ne reste que la largeur exacte de Pac-Man, l'obligeant ainsi à être centré.

Cette solution a néanmoins posé un autre souci, qui est que lorsque l'on se présente à une intersection, il fallait chronométrer son action à la perfection afin d'être aligné au pixel près avec le chemin voulu. Le problème a été résolu par Lucas, avec une fonction de tentatives d'accès à un chemin dans la direction souhaitée pendant $x$ tours.

[1] Dépôt Git du projet [https://git.unistra.fr/coco/pacman]

Les tunnels sont un élément important de Pac-Man, je les ai donc implémenté en suivant la même idée que les murs, à la différence que lorsqu'un personnage subit une collision avec le tunnel, sa coordonnée en abscisse est modifiée dans le but de le téléporter de l'autre côté de la carte. Cela étant géré dans une classe commune au fantôme et à Pac-Man, la solution a été nativement intégré aux deux entités.

De la même manière que pour Pac-Man, j'avais implémenté les prémisses du comportement des fantômes. Plusieurs solutions s'offraient à nous, mais nous n'avons pas tenté de répliquer le comportement original des fantômes ; il aurait été trop complexe de gérer les notions de plus court chemin vers Pac-Man notamment.

Ainsi, j'ai mis en place un algorithme se basant sur une valeur générée aléatoirement afin de choisir une direction à chaque mise à jour de la fenêtre. Cette première approche a été améliorée par Lucas, dans le but de gérer les intersections au milieu d'un chemin.

Nous avons par la suite mis en place la cage d'apparition des fantômes de manière conjointe. Pour ma part, j'ai mis en place les 3 fantômes manquants (Pinky, Inky et Clyde) dans le *spawn* et fait en sorte qu'ils attendent que leur statut change afin de pouvoir démarrer la chasse de Pac-Man.

Leur moment d'activation dépend du compteur de la boucle principale, sur lequel je me suis basé, car chaque tour fait plus ou moins 16 ms. Ainsi, les fantômes apparaissent chacun après un temps prédéterminé à l'aide d'une constante.

Gérer la porte de la cage a été l'une des choses les plus difficiles à intégrer. Il m'a fallu gérer le fait qu'elle ne soit qu'accessible aux fantômes s'y trouvant (afin de les laisser en sortir). J'ai donc tenté plusieurs approches infructueuses, dont notamment, la création d'un mur invisible à passer afin de détecter la sortie du *spawn*.

Cependant, j'ai trouvé une solution propre et ne nécessitant pas de téléporter le fantôme. En effet, le problème avec la solution précédente était que le fantôme se « coinçait » dans la porte, et il était donc nécessaire de le téléporter de quelques pixels… une solution peu élégante. J'ai donc revu ma copie et créé une fonction détectant si le fantôme se trouve au-dessus de la cage, afin de lui bloquer l'accès à la porte grâce à une variable, et cela sans faire usage de téléportation visible.

De là, nous avons ainsi pu collaborer sur l'élaboration d'une méthode permettant aux fantômes bleu et orange de sortir de la cage, en les centrant en face de la porte, grâce à la position de base du fantôme rose.

## 4 Statistiques et menu d'accueil

Pour gérer les différentes mécaniques et règles du jeu, telles que la fin de partie ou l'enregistrement des statistiques, il a fallu implémenter une nouvelle classe.
J'ai donc pris en charge cette tâche en ayant pour objectif d'implémenter les deux en même temps. En effet, les statistiques, et notamment le score, se doivent d'être mis à jour sur l'interface en permanence et sont interdépendants.

Lorsque la partie se finit (quand Pac-Man n'a plus de vie), nous devons réaliser plusieurs actions. Tout d'abord le score est écrit à la fin d'un fichier texte, afin de gagner en performances (le tri lors de l'écriture demanderait de tout réécrire à chaque insertion).
Ainsi, lorsqu'il est nécessaire d'afficher le score maximal, j'ai implémenté une fonction qui récupère tout le contenu du fichier dans un vecteur, afin de pouvoir tirer parti de la fonction de trie de la bibliothèque de C++. Cela permet donc d'afficher les 10 scores les plus élevés sur l'écran de titre mais également le score maximal en haut à droite.

Afin d'afficher ces scores, il a été nécessaire de réaliser un découpage de la *bitmap* afin de récupérer les coordonnées de chaque chiffre et lettres. Cela permet d'une part de créer des vecteurs d'index afin d'afficher des mots ou des phrases, qui par la suite seront affichés à l'aide de boucles for, mais également d'afficher chaque chiffre contenu dans le nombre du score (séparés à l'aide d'une fonction).
Il a fallu gérer l'emplacement vertical, mais surtout le centrage horizontal, qui m'a demandé beaucoup de tentatives et de réflexion.

Le reste du menu se compose des logos, qui n'ont pas posé de problèmes à centrer, ainsi que la phrase « *Press space key* » permettant de lancer la partie, qui elle, m'a posé beaucoup de difficulté à faire clignoter à une fréquence convenable (sans causer de crise d'épilepsie…).

La difficulté quant à la mise à jour d'éléments, du fait de la nature de SDL, est que lorsqu'on fait appel à un *BlitScaled*, l'élément s'y trouve fondu et n'est plus supprimable. J'ai donc fait appel à des rectangles comblés de noir, ajustés exactement à la taille de l'élément à masquer, afin de pouvoir mettre à jour l'interface : le score pendant la partie, mais également les vies sous forme de Pac-Man sous la carte et finalement la phrase citée précédemment, font appel à cette méthode.

## 5 Cohérence du code, usages de nouveautés de C++ et aspect objet

Une partie importante a été de maintenir le formatage du code de manière cohérente tout au long du projet, que ça soit dans la convention de codage, mais aussi les noms des variables, des fonctions et l'ajout des commentaires pour toutes les fonctions.
Sur ce dernier point, plus on avançait dans le projet plus le code s'encrassait et les différents éléments dans les fichiers s'éparpillaient. J'ai donc entrepris de réorganiser les *headers* afin de gagner en clarté ; parallèlement, j'ai commenté toutes les méthodes avec le style de *Doxygen* pour obtenir les indications pour les fonctions dans l'IDE.

J'ai également apporté différents éléments du C++, comme les boucles « *for i: liste* », mais aussi l'initialisation de toutes les variables avec des accolades en place de l'égal utilisé en C, permettant entre autres de s'assurer que le bon type de contenu est assigné à la variable.

Dans la même lignée, par habitude j'utilise des constantes pour toutes valeurs fixes. J'ai ainsi pu faire usage des « *static const* » dans les différentes classes afin de tirer parti des constantes, tout en maintenant l'aspect objet et l'organisation du projet.

Quant au codage orienté objet du projet, ma plus grande contribution a été la refonte du fichier *pacman.cpp* fourni de base, afin de ne plus avoir aucune variable globale dans le projet, mais d'apporter une gestion objet de toutes les variables liées à une partie (dans *game.cpp*). Cela a permis de nettoyer et de rendre plus lisible le code, tout en permettant de séparer le *main*, qui devenait volumineux au milieu de fonctions clés, dépendantes de variables globales.

J'ai également pris en charge la création de la partie interface du projet, en gérant notamment le menu de titre, mais également en apportant une classe permettant de gérer tous les accès à l'interface.

Finalement, le dernier travail que j'ai entrepris a été la réorganisation du projet dans un dossier *src* (pour les fichiers .cpp) et *inc* (pour les fichiers .hpp). Cela m'a amené à prendre en mains CMake, que je ne connaissais pas du tout, afin de gérer les différents dossiers, ainsi que l'emplacement de l'exécutable.

## 6 Mécaniques de jeu

Tout comme les tunnels, le fait de manger une « super pacgomme » et de rentrer dans l'état « *pellet* » (permettant de manger les fantômes) est un point essentiel de Pac-Man. Pour implémenter cela, j'ai ajouté un état aux fantômes les rendant vulnérable lorsqu'ils rentrent en contact avec Pac-Man. D'une part, les points sont attribués à notre personnage, et ce en fonction du nombre de fantômes mangés pendant la période « *pellet* », mais également leur position est réinitialisée et ils sont remis en attente pendant une durée déterminée.

Finalement, ne reste plus que la fin de partie. Elle se produit lorsque Pac-Man n'a plus de vie et renvoie le joueur vers l'écran de titre où il peut comparer son score de la partie (en haut à gauche de l'écran), à ceux de parties précédentes. Il pourra ainsi relancer directement une nouvelle partie ou simplement quitter ce cher Pac-Man…

## 7 Conclusion

Ce projet m'a beaucoup aidé à comprendre l'intérêt et l'importance de l'orienté objet dans la programmation. J'ai également beaucoup apprécié prendre en mains C++, qui m'avait laissé une mauvaise expérience par le passé, couplé à QT, mais également SDL2 qui m'a permis de concevoir l'une de mes premières interfaces !

Finalement, Pac-Man a été un sujet très prenant, d'une part par la complexité qui n'apparaît pas au premier coup d'œil, mais aussi le jeu en lui-même qui était très amusant à débugger en y jouant. J'ai ainsi pu revoir mes classiques, qui n'étaient clairement pas au niveau !