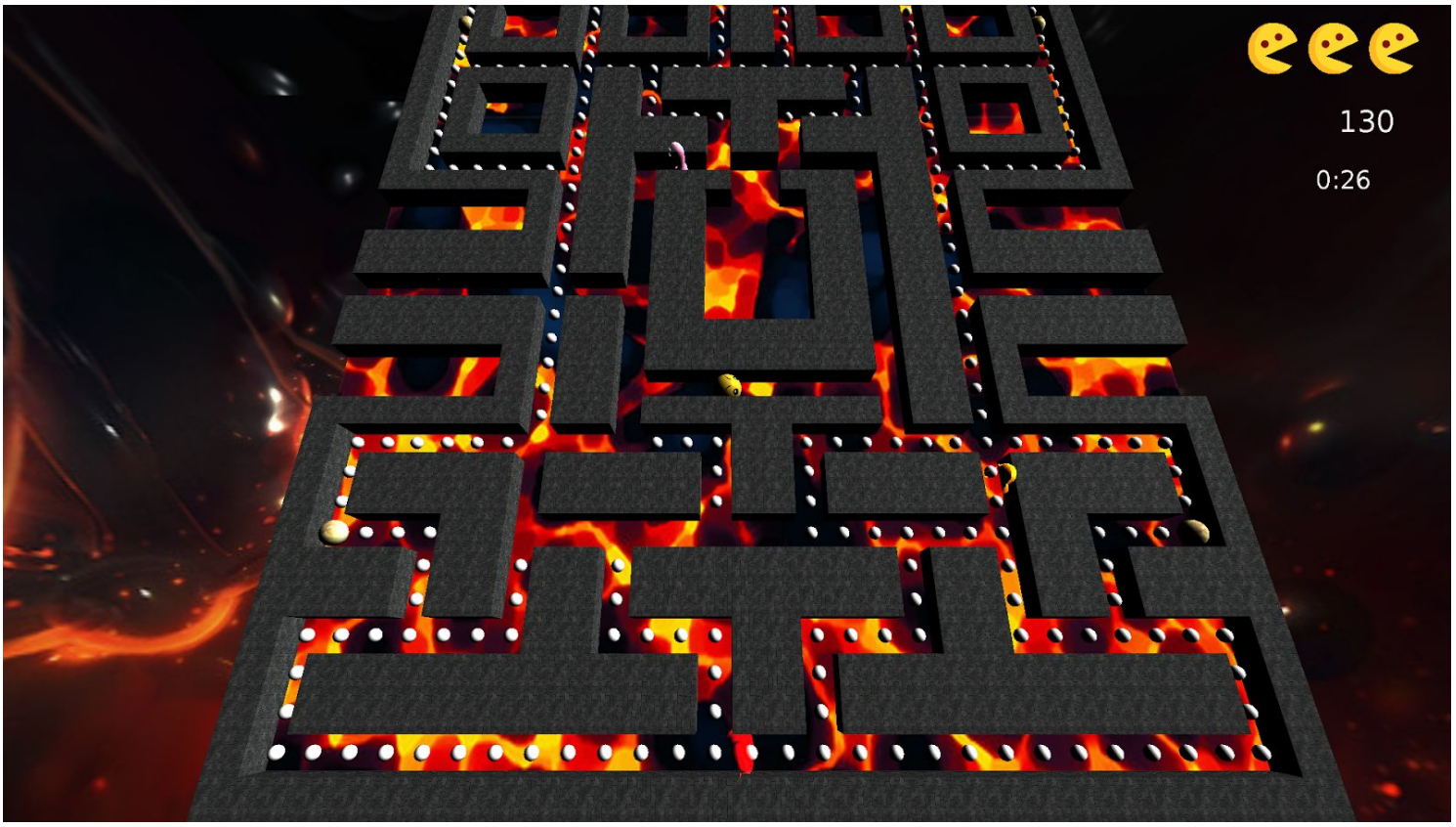
IMACMAN 3D

Rapport de projet

ISSA Laure
NASR David
ROSE Daphné



IMAC
INGÉNIEUR
ESIPE

Le 11/01/2018

Sommaire

Description du projet

MAGMAN est un jeu vidéo réalisé par trois étudiants de l'école d'ingénieur IMAC dans le cadre du cours de Synthèse d'Image associé aux TD d'OpenGL.
Le projet a été réalisé en C++ et OpenGL 3.0 en 3 semaines.

MAGMAN est basé sur le jeu Pacman qui est à la base un jeu en 2 dimension. Pour l'occasion, nous avons transformer le jeu en un jeu en 3 dimensions, avec de nouveaux décors et de nouvelles perspectives !

Mode d'emploi

Comment récupérer le projet ?
Le projet est disponible à l'adresse suivante : https://github.com/Davdouv/IMACMAN
On peut aussi le récupérer avec cette ligne de commande si on a git installé sur son PC :

$ git clone https://github.com/Davdouv/IMACMAN.git

Enfin on se place dans le dossier Build et on rentre ces lignes de commande :

$ cmake ../Code && make
$ ./IMACMAN_src/IMACMAN_NAME_OF_SRC_FILE

Comment jouer ?
Le jeu se joue sur clavier et souris ou bien à la manette.

On se déplace dans le menu avec les flèches directionnelles (avec le joystick) et on sélectionne avec le bouton Entrée (ou bouton A).

Pour déplacer Pacman on utilise les touches Z,Q,S,D (Haut, Droite, Bas, Gauche) (ou joystick). Pacman se déplace sans cesse, il changera de direction uniquement lorsqu'aucun mur ne lui bloquera le passage. Pacman retient la dernière direction demandée.

On peut changer de caméra en appuyant sur la touche C (bouton X). En mode TPS (vue du dessus), on peut zoomer en laissant enfoncé le bouton droit de la souris et en effectuant un mouvement vers l'avant ou vers l'arrière (ou RT / LT).

En jeu, on peut mettre Pause en appuyant sur Echap (ou bouton Y). S'ouvre alors un menu qui fonctionne comme le menu d'accueil.

On peut accèder à ces informations en jeu en appuyant sur I (ou bouton Start).

Pour le reste, ce sont les mêmes règles que le jeu Pacman original.

Gestion de projet

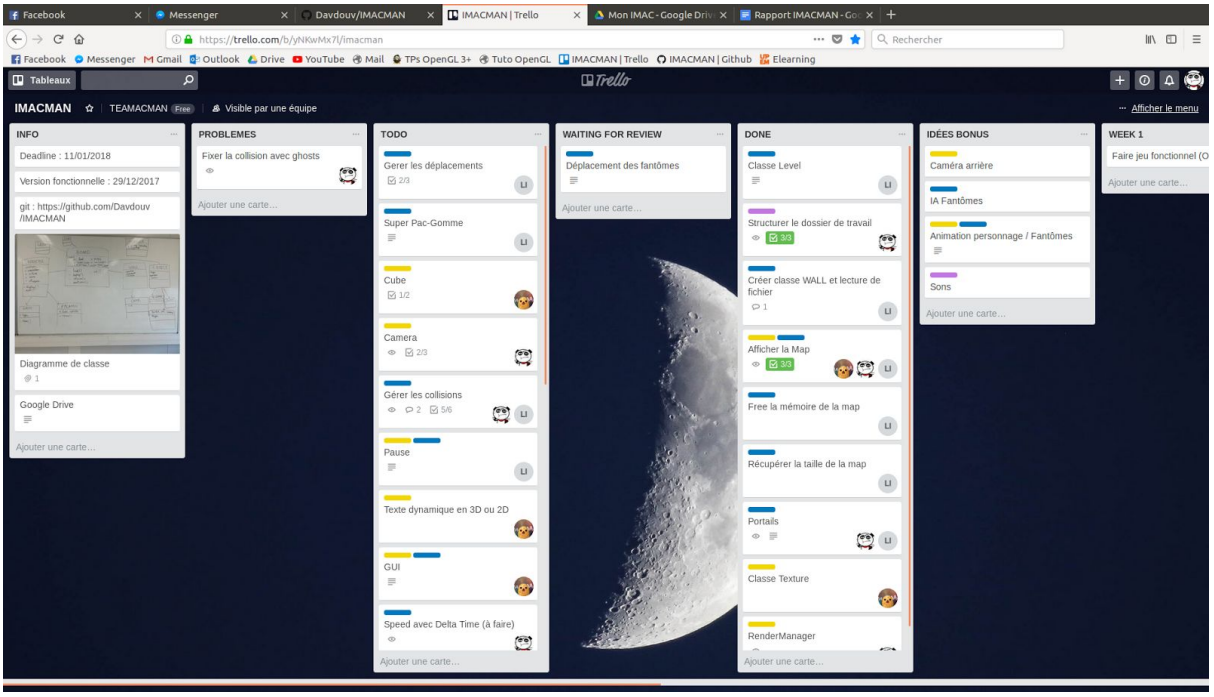
❖ Répartition des tâches

David : chef de projet (moteur de rendu, moteur de jeu)
Daphné : moteur de rendu
Laure : moteur de jeu

❖ Outils de gestion de projet

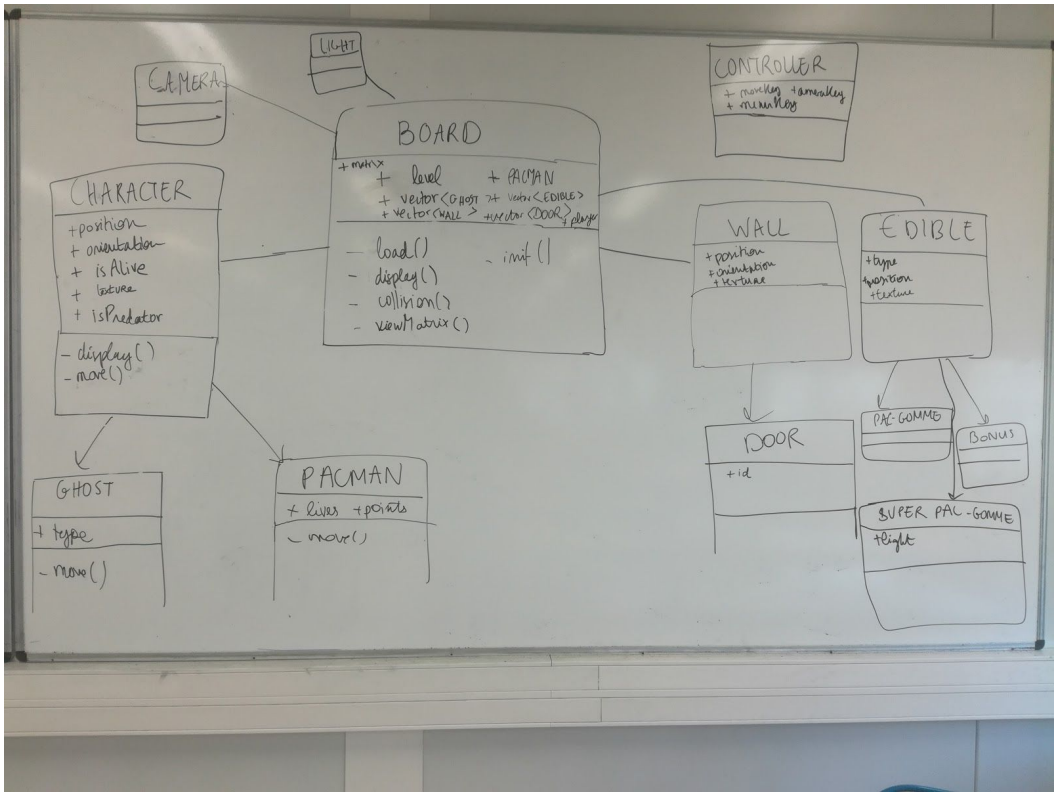
Messenger : La communication avant tout
Github (https://github.com/Davdouv/IMACMAN) : Le partage de fichiers facile
Trello : La distribution des tâches simplifiée

Impression écran du tableau Trello (on était en milieu de projet)



Architecture du programme

❖ Diagramme de classe



Nous avons principalement gardé l'architecture ci-dessus auquel nous avons rajouté le Game Manager pour le moteur de jeu et le Render Manager pour le moteur de rendu.

Résultats obtenus

Le code du jeu est dissocié en 2 grandes parties, le moteur de jeu et le moteur de rendu.

❖ Moteur de jeu

C'est le chef d'orchestre, c'est lui qui fait fonctionner notre jeu, qui connaît ses règles et tous les éléments qui le composent.

Création de la Map

La Map contient tous les objets du jeu : les objets statiques (mur, pac-gomme, portail) et les objets dynamiques (personnages : pacman, fantôme).
La classe Map a 3 attributs :

- un objet de type Pacman
- un vecteur de Ghost
- une matrice d'objets statiques

Séparer les personnages de la matrice permet de gérer la présence de plusieurs objets sur une même case et d'économiser la réécriture des personnages sur la map à chaque loop (passage dans la boucle).

La map classique du jeu est générée à partir d'un fichier texte où sont enregistrées les positions initiales de Pacman, des fantômes et des objets statiques. Lorsque le joueur enregistre sa partie, d'autres informations sont ajoutées telles que les positions courantes des personnages, le nombre de points et de vies du joueur.

Gestion du jeu

La boucle du jeu se compose de 3 parties :

- Une gestion des événements du joueur grâce à un Controller
  - Le joueur interagit avec le jeu grâce au clavier et à la souris
  - Les événements sont stockés dans une classe Controller
- Une gestion des déplacements, collisions et effets sur le joueur grâce à un Game Manager
- Une gestion de l'affichage des éléments grâce à un Render Manager (décrit plus bas)

Le déroulement de la partie se fait donc grâce à une fonction du Game Manager qui aura entre autre pour tâches :

- le déplacement des fantômes

Au lancement du jeu -et quand ils meurent- les fantômes attendent de pouvoir sortir de la zone de Spawn, chacun son tour. Une fois sortis, ils se déplacent de manière totalement aléatoire dans la Map : une orientation initiale leur est attribuée tant qu'il est possible de se

déplacer dans cette direction, sinon ils changent d'orientation jusqu'à ce que le chemin soit accessible.

- le déplacement de Pacman

Pacman se déplace sans cesse. Le joueur lui indique une direction à prendre (récupérée dans le Controller). Si Pacman peut tourner il prendra la direction demandée par le joueur. Il retient la direction demandée et s'exécute dès que possible (comme dans la version originale).
Les déplacements sont adaptés en mode FPS car Pacman regarde toujours en face donc les orientations ne sont pas les même qu'en vue du dessus.

- les collisions

Plusieurs cas de collisions sont possibles et impactent différemment le jeu :

- Collision entre un personnage et un mur : le personnage ne peut pas passer.
- Collision entre Pacman et une pac-gomme : le joueur gagne des points.
- Collision entre Pacman et un fantôme pendant l'état Super : le joueur gagne des points et le fantôme retourne à sa position initiale.
- Collision entre Pacman et un fantôme pendant l'état Normal : le joueur perd une vie et tous les personnages retournent à leur position initiale.
- Collision entre Pacman est une Super Pac Gomme : l'état super est activé.
- Collision entre pacman et un portail : le personnage se retrouve téléporté dans le portail associé.

Une collision est détectée lorsque les coordonnées dans le repère du graph sont équivalentes. On s'est aussi rapidement posé des questions du type "Comment faire lorsque Pacman est situé entre 2 cases ?".
L'algorithme du déplacement (en très résumé) est le suivant :

- Si on est entre 2 cases, on ne peut pas changer de direction.
- Sinon
    - On vérifie si on peut se déplacer à l'intérieur de la cellule dans laquelle on est.
    - Sinon, on va rentrer dans une nouvelle case, on vérifie si ce n'est pas un mur.

Les autres tâches de la fonction de jeu sont liés au joueur (perdu, gagné) et à l'interface (pause)

Gestion des interfaces

Le jeu dispose d'un menu principal et d'un menu de pause issus de la même classe.
Le menu principal est la première chose que l'on voit en lançant le jeu. Il permet de jouer une nouvelle partie, continuer la dernière partie sauvegardée ou quitter le jeu proprement.
Le menu de pause dispose d'un bouton pour recommencer la partie, un autre pour sauvegarder la partie et un troisième pour quitter le jeu.
Les interactions se font grâce aux touches récupérées par le Controller. En pause, la boucle de jeu n'est pas utilisée (pas de gestion de collision etc.).

❖ Moteur graphique

Il s'occupe de l'affichage des éléments et commande la carte graphique pour qu'on puisse voir notre beau jeu.

Le moteur graphique est géré par le Render Manager appelé à la suite du Game Manager dans la boucle de jeu. Il affiche pour chaque frame les éléments visibles par la caméra sélectionnée.

Formes primitives

Nous avions déjà une sphère, nous avions besoin d'un cube pour les murs du labyrinthe, et d'un plan pour le sol. Le plan est aussi utilisé par l'interface graphique (UI).

Nous avons développé une surcouche à OpenGL qui facilite la création et l'affichage de ces formes primitives. Chaque forme possède son VBO, son IBO et son VAO accessibles grâce à des getters. Elles possèdent une méthode de dessin.

Le Render Manager contient une seule instance de chacune des 3 primitives utilisées pour dessiner tous les éléments du jeu.

Le Render Manager contient également des fonctions qui, par exemple, facilitent le placement d'un objet en fonction de la taille de la map, ou encore la modification des variables uniformes des shaders.

Caméras

Le jeu dispose de deux caméras. Une caméra (par défaut) vue du dessus à la troisième personne (TPS) et une caméra vue à la première personne (FPS).

La caméra FPS est placée légèrement derrière Pacman et un peu en hauteur de sorte à avoir une vue agréable (Pacman n'est pas rendu en mode FPS car on voit à travers ses yeux).

La caméra TPS était plus compliquée à placée car elle devait être centrée sur Pacman et en même temps inclinée. On a ainsi rencontré un effet de zoom avec la rotation des axes que nous avons corrigé.

Textures

La classe texture gère le chargement d'une texture image par son chemin et renvoie le pointeur sur la texture générée pour son affichage par le Render Manager.

Skybox

Afin d'obtenir un bel environnement, nous avons implémenté une skybox créant un cube texturé à 6 faces différentes. La texture *cubemap* du cube est générée dans la classe Cubemap à partir de 6 images.

User Interface

L'User Interface affiche les vies, le score et le menu. Ces éléments étant en 2D, nous avons utilisé le plan que nous avons texturé et redimensionné. Pour afficher l'UI au même endroit à l'écran sans dépendre de la caméra, nous avons "collé" les plans à la caméra d'origine (position 0,0,-5).

Pour afficher du texte (score, temps) nous avons utilisé la librairie SDL_TTF permettant de charger des polices de caractère et de les convertir en texture à afficher sur un plan.

Le score et le temps doivent s'afficher dynamiquement, ce qui implique la création régulière de nouvelles textures. Ce procédé est cependant lourd et fait baisser les performances de jeu.

Modéles 3D :

Pour avoir de beaux fantômes, nous avons chargé un modèle 3D (réalisé par un ami). Nous l'avons chargé grâce à la fonction loadObj de la bibliothèque glimac.

Nous avons stocké une couleur différente pour chaque fantôme qui est envoyée au shader, et activé la couche alpha lors du rendu des fantômes pour avoir un effet de transparence.

Animations :

Lors de la phase de polish nous en avons rajouté quelques améliorations en terme d'animation, pour que le jeu soit à la fois plus esthétique et plus compréhensible.

Nous avons animé le sol en déplaçant la coordonnée Y de sa texture en fonction du temps. Nous avons également animé Pacman en dessinant deux demi-sphère avec une rotation en fonction du temps également.

Enfin nous avons animé l'interpolation en mode FPS pour mieux comprendre dans quel sens on tourne (plutôt que d'avoir une rotation nette).

Shaders :

Nous utilisons principalement 2 shaders. Un shader de texture simple pour le sol et l'UI, et un shader qui prend en compte la lumière directionnelle, les matériaux, la texture et la couleur de l'objet.

Tous les shaders sont stockés dans une liste de programmes. Ces programmes sont appelés à chaque passage dans la boucle de rendu pour changer le shader utilisé en fonction de l'objet dessiné.

❖ Autres

Audio

Le son est important dans un jeu car il participe à l'ambiance et apporte beaucoup d'informations. Nous avons utilisé la librairie SDL_Mixer et une classe Audio Manager qui gère toute la partie audio afin de jouer des musiques et effets.

Performance de l'ordinateur

La performance de l'ordinateur est un paramètre à ne pas négliger. En effet, un ordinateur puissant passera beaucoup plus vite dans la boucle de jeu qu'un ordinateur basique. De ce fait les personnages se déplaceront beaucoup plus rapidement, et inversement pour un ordinateur peu performant. Nous avons donc pris en compte le delta time dans le calcul de la vitesse qu'on met à jour à chaque frame pour avoir même vitesse sur n'importe quel ordinateur.

Joystick

Le jeu peut être joué avec une manette. Il faut la brancher avant de démarrer le jeu pour qu'elle soit prise en compte.

Une résolution qui s'adapte

Plutôt que de fixer la résolution, nous avons mis le jeu en plein écran avec la résolution de l'écran sur lequel on joue.

Améliorations souhaitées

❖ L'intelligence artificielle

Nous avons tenté d'implémenter l'intelligence artificielle représentant les comportements de chaque fantôme mais nous n'avons malheureusement pas réussi à le faire fonctionner. Notre démarche était la suivante :

Une fonction nextMove(float x, float y) qui prend en paramètres les coordonnées d'une position d'arrivée. La fonction retourne la direction que le fantôme doit prendre pour arriver le plus rapidement possible à la position d'arrivée depuis sa position courante. L'algorithme reprenait le principe de l'algorithme de Djikstra.

Pour chacun des fantômes, on appelle cette fonction avec des paramètres différents:

- Pour **Shadow**, qui suit Pacman comme son ombre, on appelle nextMove avec les positions courantes de Pacman.

- Pour **Speedy,** qui se dirige toujours vers Pacman, on divise la Map en 4 parties. Selon la position de Pacman dans une de ses parties, on appelle nextMove avec comme point d'arrivée : en haut à droite, en haut à gauche, en bas à droite ou en bas à gauche.

- Pour **Bashful**, qui suit Pacman mais le fuit dès qu'il s'en rapproche, on appellera nextMove() comme pour Shadow, sauf quand il se trouve trop près de Pacman. Dans ce cas, c'est dans la direction inverse qu'on enverrait ce fantôme.

- Pour **Pokey**, qui se déplace totalement aléatoirement dans le labyrinthe, pas besoin d'intelligence artificielle, la démarche qu'on utilise dans notre version actuelle fonctionnerait avec le comportement de ce fantôme.

Les problèmes rencontrés lors de cette implémentation se trouvaient au niveau de la complexité de temps. A chaque loop, les positions d'arrivée changent, donc la recherche du plus court chemin devait se faire à tout moment du jeu, ce qui ralentissait le jeu.

L'autre idée était donc d'envoyer les fantômes vers la position d'arrivée voulue sans forcément emprunter le plus court chemin. Il suffisait de déterminer où se trouvaient l'arrivée par rapport à la position du fantôme et de l'envoyer en haut/droite, haut/gauche, bas/droite ou bas/gauche.

Cependant, après implémentation, les fantômes nous paraissaient hésitants et effectuaient plusieurs aller-retours "inutiles" ce qui a rendu le jeu moins fluide et intéressant que lorsqu'ils se déplaçaient tous de manière aléatoire.

❖ Une mini-map

On aurait aimé avoir notre cerise sur le gâteau avec une mini-map mais après une tentative échouée et un manque de temps, cela ne s'est pas fait. Mais nous avons compris la technique utilisée : le render target. C'est une technique qui permet de créer une texture à partir d'une vue caméra, que l'on va ensuite plaquer sur un plan. Cette technique peut également être utilisée pour faire des miroirs ou encore le reflet de l'eau (avec une inversion de matrice).

Crédits

Merci aux TD d'OpenGL menés par Maxime Maria et conçus par Laurent Noël.

Les musiques et sons utilisés sont libres de droits. Certaines textures viennent d'internet et d'autres ont été créés par nous mêmes.

Merci à Mehdi Benhamar pour avoir créé le modèle 3D de fantôme ainsi que la musique du mode super pacgomme.